Odometría visual monocular basada en redes adversarias generativas

---

Javier Alejandro Cremona
cremonajavier@gmail.com

Director
Pire, Taihú
pire@cifasis-conicet.gov.ar

Co-Director
Uzal, Lucas
uzal@cifasis-conicet.gov.ar

Facultad de Ciencias Exactas, Ingeniería y Agrimensura
Universidad de Rosario
Av. Pellegrini 250 - Planta Baja - (S2000BTP)
Rosario - Rep. Argentina.
Teléfono: +54 - 341 - 4802649/52 - interno 112
http://www.fceia.unr.edu.ar

Odometría visual monocular basada en redes adversarias generativas

Uno de los principales desafíos a resolver en el campo de la robótica móvil es el de lograr que un robot autónomo pueda estimar su posición y orientación en el entorno que se encuentra navegando. Al problema de estimar el desplazamiento de un robot incrementalmente a partir de una secuencia de imágenes entrante se lo denomina VO (por las siglas en inglés de *Visual Odometry*). En los últimos años las redes neuronales convolucionales (CNN) han sido aplicadas a diversos problemas de *Computer Vision*, y VO no fue la excepción.

En este trabajo se propone el uso de una CNN para resolver el problema de VO. Para esto, se emplea un tipo particular de CNN conocido como GAN (por las siglas en inglés de *Generative Adversarial Network*) que permite realizar su entrenamiento mediante una técnica semi-supervisada, la cual incluye imágenes reales e imágenes generadas artificialmente por la propia red. Se evaluó la performance y la precisión de dicho sistema en datasets de dominio público. Concretamente, la propuesta se centra en vehículos autónomos pero la solución podría extrapolarse a otros tipos de robots. Los resultados obtenidos muestran que el sistema es capaz de trabajar en tiempo real y su precisión es comparable con sistemas similares del estado del arte. El método propuesto recibe el nombre de WGANVO y su código se libera junto con todo el software desarrollado como software libre para facilitar su uso por la comunidad de Robótica Móvil y de Aprendizaje Profundo.

Índice general

Capítulo 1

Introducción

En el campo de la robótica móvil [1, 2] uno de los principales problemas a resolver es el de dotar a un robot autónomo con la capacidad de estimar su posición y orientación (pose) de manera precisa en el entorno que se encuentra. Este problema es conocido en robótica móvil como problema de *Localización*. A su vez, se define a la odometría como el proceso de estimar el desplazamiento de un robot a través del tiempo, a partir de mediciones obtenidas desde sus sensores.

Para el cálculo de odometría, se ha hecho uso de diversos sensores tales como Unidades de Medición Inercial (IMU), odómetros, lasers y cámaras, siendo estas últimas de principal interés dado su bajo costo, portabilidad y la abundante información que proveen de la escena observada. Cuando se utilizan cámaras como sensores para el cálculo de la odometría se la denomina *Odometría Visual* (VO, por sus siglas en inglés) [3, 4].

VO se ha convertido en objeto de estudio en el último tiempo, logrando grandes resultados a partir de los avances en el campo de Visión por Computadora [3, 5, 6, 7, 8]. Generalmente los métodos tradicionales de VO se basan en el uso de geometría multi-vista (*Multi View Geometry*) para estimar la pose del robot [9].

Por otra parte, en la última década, las Redes Neuronales Convolucionales (CNN, por sus siglas en inglés) han sido exitosamente aplicadas para resolver diversos problemas de Visión por Computadora [10]. Algunos ejemplos en los que se han utilizado CNN son: detección de objetos, clasificación de objetos, segmentación semántica, y extracción de características visuales (*features*) [11, 12]. En particular, resultan relevantes las Redes Adversarias Generativas (GAN, por sus siglas en inglés) [13]. Mediante esta metodología se entrenan dos redes simultáneamente. Una red G, llamada Generador, es capaz de generar muestras sintéticas similares al dataset, mientras que, a su vez, otra red D, llamada Discriminador, se especializa en discriminar muestras reales de aquellas generadas por G. La red discriminadora logra representaciones de alto nivel de abstracción de los datos prescindiendo de datos supervisados para su entrenamiento y puede ser reutilizada en un esquema semi-supervisado para resolver otras tareas, como por ejemplo clasificación multi-clase [11, 14, 15].

En los años recientes, ha surgido el interés de resolver el problema de VO utilizando CNN con un enfoque *end-to-end* [16, 17, 18, 19]. El mencionado enfoque propone utilizar una red convolucional para resolver completamente una tarea, en lugar de dividir el problema en distintas etapas de trabajo. Tal división es común en los métodos tradicionales de VO [20, 7].

La presente tesina propone resolver el problema de VO mediante una red convolucional basada en GAN, utilizando el enfoque *end-to-end*. Esto resulta en un Discriminador D actuando como regresor para el cual el Generador G sirve como una fuente adicional de información para mejorar su precisión, además de los datos supervisados.

1.1. Motivación

Los métodos de VO del estado del arte generalmente constan de varios módulos bien definidos, independientemente de si se los clasifican como directos o indirectos (ver Sección 2.1). Estos módulos representan distintas etapas en las que se suele dividir el problema. Durante la implementación de un método, una consideración importante a tener en cuenta es el tiempo de ejecución, ya que se espera que un sistema de VO

responda en tiempo real. En este contexto, por tiempo real nos referimos a que el método pueda operar, como mínimo, a la velocidad a la que la cámara captura las imágenes (expresada en FPS, *frames* por segundo). Esto implica que cada módulo requiere un esfuerzo extra, tanto en el diseño como en la implementación, para asegurar que los tiempos de respuesta globales no se vean afectados. Creemos que podemos reemplazar este *pipeline* tradicional con una CNN, evitando las complejidades que surgen al implementar los módulos individualmente. A esta forma de trabajar se la conoce como *end-to-end*. En este contexto, la CNN aparece como un algoritmo más simple, en el sentido de que, una vez entrenada, contamos con una función con entrada y salida bien definidas que comprende todo el método.

Las CNNs han exhibido excelentes resultados en problemas de clasificación en el área de Visión por Computadora [12, 21]. Sin embargo, su utilización en problemas de regresión no es tan masiva. Nuestra propuesta aborda el problema de VO como un problema de regresión, intentando demostrar que una arquitectura similar utilizada para problemas de clasificación puede adaptarse y utilizarse como un regresor para obtener los coeficientes de la transformación relativa entre dos *frames*.

Una de las desventajas de utilizar redes neuronales, especialmente CNN, es el hecho de necesitar una gran cantidad de datos para el entrenamiento. Cuando se trabaja en un esquema supervisado, adicionalmente se necesitan datos etiquetados, esto es, para cada muestra de entrenamiento, contar con la salida esperada de la red para ese ejemplo. En la práctica, no siempre es sencillo contar con grandes volúmenes de datos etiquetados. Para mitigar este problema, se desarrollaron esquemas de entrenamiento no supervisados [22, 23, 14, 24].

El presente trabajo propone entrenar una CNN con un esquema semi-supervisado para resolver el problema de VO de forma *end-to-end*. Creemos que con la aplicación de una CNN podemos reemplazar la implementación de todos los módulos del *pipeline* de un método de VO tradicional. Adicionalmente, el esquema semi-supervisado nos permite entrenar con un menor volumen de datos. Más concretamente, nuestra CNN toma como entrada un par de imágenes de una secuencia, generalmente un par de *frames* consecutivos y tiene como salida la estimación del cambio de posición y orientación de la cámara entre los dos *frames* recibidos.

1.2. Objetivo

El objetivo del presente trabajo es desarrollar un sistema de VO monocular basado en CNN. La propuesta busca entrenar el modelo en forma semi-supervisada. Esto es, la parte supervisada será planteada como un problema de regresión: dados dos *frames* consecutivos predecir los coeficientes de la transformación (compuesto por un vector de orientación y un vector de traslación) que representan el movimiento de la cámara entre ambos *frames*. La red será entrenada en simultáneo en forma adversaria (GAN) de modo de propiciar la extracción de características relevantes en forma no supervisada. En particular, se adaptará un modelo basado en WGAN [25, 26] para entrenar la red con un esquema semi-supervisado. Se extenderá la salida del Discriminador para resolver un problema de regresión.

La solución propuesta utiliza, en lo posible, estándares abiertos y software libre como *Robot Operating System* (ROS) [27] y OpenCV [28].

1.3. Organización del trabajo

El capítulo 2 presenta el problema de VO y el estado del arte en sistemas de VO, incluyendo aquellos que utilizan *Deep Learning* y geometría multivista. En el capítulo 3 se presentan conceptos básicos de redes neuronales y de visión por computadora necesarios para describir el presente trabajo. En el capítulo 4 se explica el método propuesto, detallando las modificaciones realizadas a la CNN y al dataset. En el capítulo 5 se detallan los experimentos realizados y se discuten los resultados obtenidos. Finalmente, en el capítulo 6 se presentan las conclusiones y posibles trabajos futuros derivados de la presente tesina.

Capítulo 2

Estado del arte

En este capítulo se presentan diferentes trabajos de VO que componen el estado del arte. En la Sección 3.2 se presenta una definición más precisa del problema de VO.

2.1. *Visual Odometry*

Visual Odometry (VO) [3, 4] es el proceso de estimar incrementalmente la pose de un robot a través del tiempo utilizando información visual. Un robot que posee cámaras a bordo puede valerse de las imágenes capturadas para, mediante un algoritmo de VO, localizarse en un ambiente que nunca antes haya visitado.

Los métodos de VO suelen ser divididos en dos grandes grupos: métodos basados en *features* (o indirectos) y métodos directos. En este contexto, los *features* son puntos salientes y fácilmente distinguibles en una imagen, permitiendo realizar un seguimiento (*tracking*) de los mismos de una imagen a otra. El seguimiento de *features* en sucesivas imágenes permite estimar el movimiento de una cámara. Por su parte, los métodos directos trabajan sobre la intensidad de los píxeles sin reparar en cuáles son los puntos más distinguibles de la escena.

En años recientes se desarrollaron métodos que utilizan aprendizaje automatizado o aprendizaje profundo para resolver el problema de VO. Como estos métodos no caen en ninguno de los grupos mencionados, podemos considerar una nueva clasificación dentro de los métodos de VO. En las siguientes subsecciones se describen métodos de VO que componen el estado del arte divididos en métodos basados en *features*, métodos directos y métodos que utilizan *Deep Learning*.

2.1.1. Métodos basados en *features*

Los primeros trabajos en VO pueden clasificarse como métodos basados en *features*. StereoScan [30] es un método basado en *features* que propone una reconstucción de la escena 3D en tiempo real conjuntamente con un eficiente algoritmo de VO. El método cuenta con cuatro etapas bien diferenciadas: extracción de *features*, estimación del movimiento, cálculo de disparidad y reconstrucción 3D densa. Para obtener los *features*, StereoScan busca correspondencias entre dos pares estéreo de *frames* consecutivos. Después de obtener dichos *features*, realiza una triangulación a partir de *features* del *frame* previo para obtener cientos de puntos 3D [31], los cuales son proyectados en el *frame* actual con el objetivo de minimizar el error de reproyección. De esta manera, obtiene una transformación que estima el movimiento de la cámara. En la tercera etapa, el método se encarga de calcular un mapa denso de disparidad para finalmente reconstruir la escena 3D en la cuarta etapa.

Por su parte, ORB-SLAM2 [32] es un método de SLAM (*Simultaneous Localization And Mapping*) [33, 34], un problema de la robótica móvil más general en el que se estima un mapa del entorno en el que se mueve el robot, a la vez que se estima la pose del mismo en el mapa. ORB-SLAM2 aprovecha la naturaleza paralelizable del problema al utilizar tres hilos de ejecución en paralelo. El primer hilo se ocupa de calcular el movimiento de la cámara. Primero extrae *features* ORB [35] de la imagen actual para luego buscar correspondencias en el mapa reconstruido. Luego, estima el movimiento con la información obtenida, y finalmente decide si la

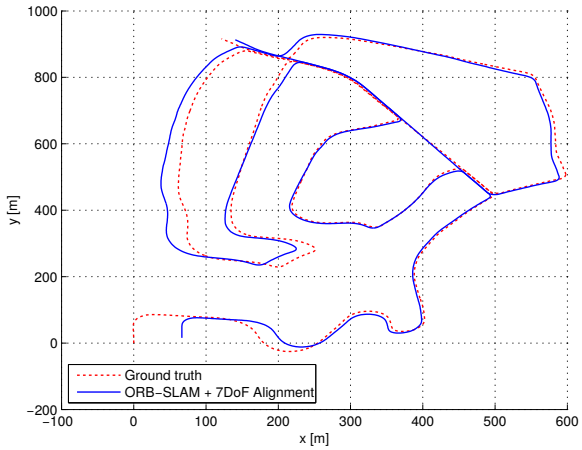

Figura 2.1: Trayectoria real (*ground-truth*) de una secuencia del dataset KITTI [29] (en rojo) junto a una estimación producida por el sistema ORB-SLAM [20] (en azul). Imagen extraída de [20].

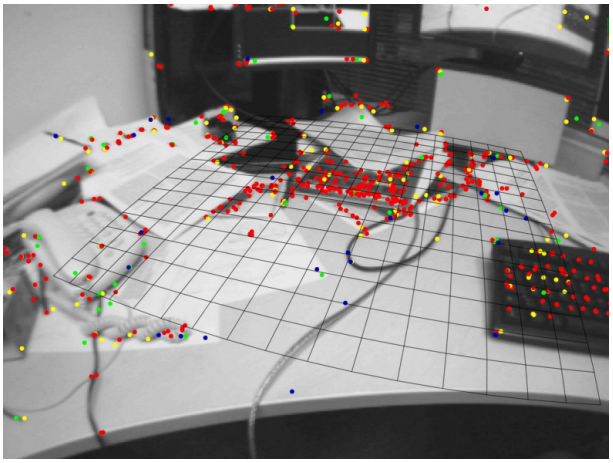

Figura 2.2: Puntos del mapa construído por PTAM proyectados sobre un *frame*. Imagen extraída de [36].

imagen actual se inserta como un nuevo *keyframe* (los *keyframes* son utilizados para generar puntos cuando un área nueva se está visitando). El segundo hilo se encarga de procesar nuevos *keyframes* y actualizar un mapa local en los alrededores de la posición actual. Por último, el tercer hilo realiza una optimización en busca de lugares previamente visitados. Dicha optimización se realiza para incrementar la precisión de la estimación de la pose actual, reduciendo el error acumulado por el método. A esta técnica se la conoce como cierre de ciclos (*loop closure*). En la Figura 2.1 se puede ver una trayectoria estimada por ORB-SLAM2 sobre una secuencia del dataset KITTI [29].

De forma similar, PTAM [36] separa la estimación de la pose de la construcción del mapa en hilos de ejecución diferentes. PTAM utiliza una técnica de optimización iterativa conocida como *Bundle Adjustment*, la cual le permite refinar o perfeccionar la pose de la cámara al minimizar la distancia entre la reproyección del modelo tridimensional y los puntos asociados en la imagen. Inicialmente, este método monocular fue diseñado específicamente para ser utilizado en aplicaciones de realidad aumentada. Por tal razón, su precisión es óptima solamente en ambientes pequeños. Para mitigar este problema, se implementaron diversos métodos basados en PTAM. Uno de ellos es S-PTAM [37], el cual utiliza cámaras estéreo, lo que le permite recuperar la escala real de la escena y mayor robustez en entornos altamente dinámicos. En la Figura 2.2 se pueden ver puntos de un mapa generado por PTAM proyectados sobre la imagen.

2.1.2. Métodos directos

Por su parte, los métodos directos trabajan directamente sobre los píxeles de las imágenes, evitando el costo del procesamiento de *features*. En general, surgieron en los últimos años, y tienen la ventaja de aprovechar más información de la escena que los métodos basados en *features*.

DSO (*Direct Sparse Odometry*) [7] es un sistema de VO perteneciente al grupo de los métodos directos, ya que no utiliza la clásica adquisición de *features*, sino que utiliza la intensidad de los píxeles para buscar los puntos más importantes de una escena. Entre los puntos obtenidos no existe una noción de vecindad como en un enfoque denso, de allí que el sistema es clasificado como *sparse,* o disperso. DSO optimiza conjuntamente todos los parámetros involucrados, tales como parámetros intrínsecos y extrínsecos de la cámara, y valores de profundidad, con el objetivo de minimizar el error fotométrico (diferencia de intensidad entre píxeles). Este ajuste se hace sobre una ventana de tamaño fijo para cada punto. El algoritmo conserva en todo momento un conjunto de tamaño fijo de puntos y *keyframes* activos, actualizándolos constantemente. Cada nueva imagen entrante se inicializa con respecto a dicho conjunto, y al crearse un nuevo *keyframe* se proyecta un mapa de profundidad semi-denso. DSO ha demostrado un buen rendimiento sobre zonas de baja textura, por ejemplo, paredes blancas. En tales superficies es mucho más complicado obtener *features*, por lo que este método aventaja a aquellos basados en *features* en este aspecto. En la Figura 2.3 se pueden observar distintos mapas de profundidad generados por DSO.

Otro método que vale la pena mencionar es LSD-SLAM (*Large-Scale Direct SLAM*) [38]. Es un método monocular que consta de tres componentes principales: *tracking*, estimación del mapa de profundidad y optimización del mapa global. El componente de *tracking* toma una nueva imagen y trata de estimar la pose con respecto al *keyframe* actual, utilizando la pose de imágenes previas como inicialización. El componente de estimación del mapa de profundidad debe decidir si la imagen se utilizará como un nuevo *keyframe* o para refinar el mapa de profundidad sobre el *keyframe* actual. Si la imagen se convierte en un nuevo *keyframe*, se inicializa un mapa de profundidad de dicho *keyframe,* proyectando puntos de *keyframes* previos sobre éste. Finalmente, el componente de optimización del mapa global se ocupa de la detección y cierre de ciclos. LSD-SLAM demuestra gran precisión en ambientes en los que hay grandes variaciones en la escala de la escena.

DTAM (*Dense Tracking and Mapping*) [39] permite estimar el movimiento de una cámara y la reconstrucción de la escena. Alternadamente, construye un mapa denso de profundidad texturado utilizando imágenes casi superpuestas y estima el cambio de posición y orientación generando imágenes sintéticas a partir del mapa reconstruido. DTAM exhibe grandes resultados en aplicaciones de realidad aumentada, superando a PTAM en varios aspectos como precisión y robustez.

Por su parte, SVO (*Semi-direct Visual Odometry*) [8] utiliza un enfoque híbrido. Combina los factores de éxito de los métodos basados en *features* con la precisión y la velocidad de los métodos directos. SVO no utiliza extracción ni búsqueda de correspondencia de *features* para estimar la pose, dos procesos que resultan computacionalmente costosos. La extracción de *features* sólo es necesaria cuando se selecciona un *keyframe* para inicializar nuevos puntos 3D. Esto permite una mayor velocidad de procesamiento. Debido a que el método es monocular, utiliza filtros bayesianos para estimar la profundidad de los *features*. Un punto 3D se agrega a un mapa global sólo cuando el filtro correspondiente converge, lo cual requiere procesar múltiples *frames* consecutivos. La estimación de la pose se realiza en paralelo en un hilo aparte. En primera instancia, entre *frames* consecutivos se minimiza el error fotométrico de los píxeles correspondientes a las localizaciones proyectadas de un mismo punto 3D. Es decir, los puntos 3D visibles se proyectan en los *frames* consecutivos y se calcula la diferencia en intensidad de los píxeles. Dado que esta reproyección está parametrizada por la pose relativa entre ambos *frames*, se busca la pose relativa que mejor se ajusta, minimizando las diferencias de intensidades. Posteriormente, se mejora individualmente la localización de los puntos proyectados en el *frame* actual utilizando *keyframes* previos. Finalmente, se realiza un refinamiento de la pose absoluta del *frame* actual y de puntos cercanos del mapa. Originalmente, SVO fue pensado para ser utilizado en micro-vehículos aéreos, exhibiendo mayor precisión y robustez que otros algoritmos diseñados con el mismo propósito [40, 41], los cuales en su mayoría están basados en PTAM.

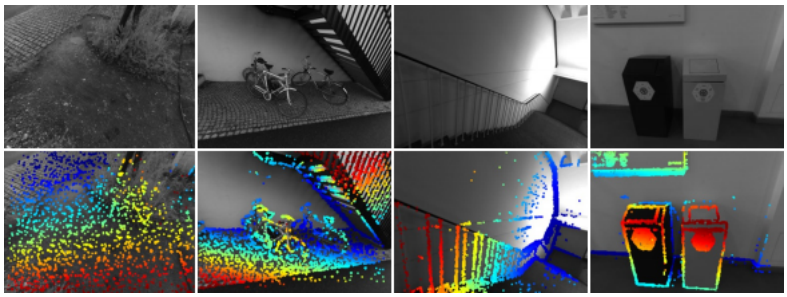

Figura 2.3: Ejemplos de mapas de profundidad generados por DSO. Imagen extraída de [7].

2.1.3. Métodos que utilizan técnicas de *Deep Learning*

Para resolver el problema de VO se utilizaron técnicas de *Deep Learning* de diferentes maneras. Por un lado, métodos como DVSO (*Deep Virtual Stereo Odometry*) [42] utilizan redes neuronales para resolver simplemente algún módulo de su *pipeline* de trabajo. En particular, DVSO es una mejora a DSO [7] y está basado fuertemente en dicho método. Utiliza una CNN para predecir los mapas de profundidad necesarios al iniciar un nuevo *keyframe*. Además, esta predicción sirve para agregar restricciones geométricas a la optimización conjunta que realiza DSO.

Otro enfoque utilizado es el de abordar completamente el problema de VO mediante *Deep Learning*. Este es el enfoque elegido en el presente trabajo, y se lo conoce como *end-to-end*. En esta propuesta se reemplazan todos los módulos del *pipeline* de un algoritmo tradicional de VO con técnicas de *Deep Learning*, comúnmente CNNs. La principal ventaja es que, a diferencia de los métodos tradicionales, no requiere invertir esfuerzo en cada uno de los módulos para asegurar un buen rendimiento en diferentes ambientes. Más aún, los métodos monoculares tradicionales sufren el problema de no poder estimar la verdadera escala del mundo real, esto es, la relación entre tamaños en el mundo y en la imagen. Dado que utilizan conceptos geométricos para estimar la pose, se ven afectados por el hecho de que con una única cámara no se tiene noción de la distancia de los objetos a la cámara. Como veremos en la Sección 3.1, una cámara proyecta puntos 3D del mundo en un plano imagen bidimensional, y durante este proceso se pierde el valor de profundidad de cada punto 3D dificultando la reconstrucción de la escena tridimensional. Los métodos monoculares que aplican Deep Learning no se ven afectados por este problema ya que las redes convolucionales durante el entrenamiento pueden aprender qué tamaño tienen los objetos en el mundo real.

DeepVO [16] aborda el problema de forma *end-to-end*. Sus autores proponen entrenar una red neuronal convolucional recurrente profunda. La arquitectura propuesta puede verse en la Figura 2.4. La parte convolucional de la red captura distintas características y patrones de las distintas imágenes. A su vez, la parte recurrente modela el movimiento de la cámara a partir de una secuencia de imágenes. Las redes recurrentes son ideales para capturar información de secuencias temporales. Al analizar un par de imágenes cuentan con información de movimientos previos, por lo que la estimación del movimiento de la cámara se ve ampliamente beneficiada. A diferencia de nuestro trabajo, DeepVO utiliza un esquema de entrenamiento supervisado. Esto implica que la red neuronal de DeepVO necesita gran cantidad de datos con etiqueta para ser entrenada, es decir, secuencias de imágenes para los que se conozcan de antemano los cambios de posición y orientación.

UnDeepVO [22] es otro ejemplo de método *end-to-end*. Los autores proponen un esquema basado en dos CNNs: una CNN recibe dos imágenes monoculares consecutivas y estima el movimento de la cámara entre ambas, mientras que la otra red genera mapas de profundidad a partir de la entrada. El entrenamiento utiliza una novedosa función de costo que permite trabajar en forma no supervisada. Una desventaja es que durante la etapa de entrenamiento se necesitan imágenes estéreo para recuperar la escala absoluta de la escena y generar mapas de profundidad, a pesar de que el método es monocular.

Por su parte, GANVO [43] es un método no supervisado que combina redes adversarias generativas con DeepVO y UnDeepVO. Una red, conocida como Generador, construye mapas de profundidad, mientras que una red convolucional recurrente estima la pose de pares de *frames* recibidos. Con la información de la profundidad y la pose estimada se genera una imagen sintética a partir de una de las imágenes del par entrante con el objetivo de generar una imagen lo más similar posible a la otra imagen del par. Finalmente,

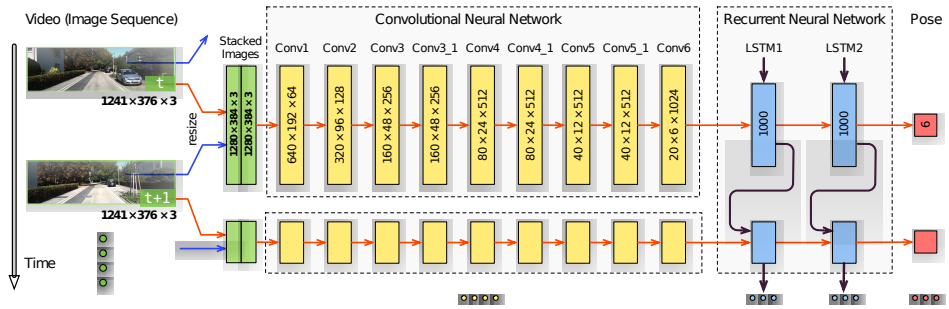

Figura 2.4: Arquitectura propuesta por DeepVO. Imagen extraída de [16].

otra red, conocida como Discriminador, recibe pares de *frames*, tanto pares del dataset como pares generados sintéticamente. La salida del Discriminador es un valor de probabilidad indicando si el par es real o sintético. A partir de esta salida, el Generador y la red recurrente aprenden a ser más precisos en su tarea. Si bien obtiene mejores resultados que los obtenidos por el método aquí presentado, GANVO incluye más información, como lo son los mapas de profundidad, y presenta una arquitectura más compleja que incluye redes convolucionales recurrentes.

Un abordaje más completo y complejo del problema es el que presenta GeoNet [44]. Los autores dividen el problema en la reconstrucción de la escena estática por un lado, y en la localización de los objetos dinámicos por el otro. Para ello, combinan información de la profundidad, la pose y el flujo óptico, implementando una CNN para resolver cada una de estas sub-tareas. Notablemente, GeoNet muestra gran precisión en cada una de ellas con el agregado de ser un método no supervisado.

Un problema ligeramente diferente es el que aborda PoseNet [45]. En este caso, a partir de una imagen se debe estimar la ubicación de la cámara en una escena conocida. A este problema se lo conoce como *Global Localization*. A diferencia de VO, aquí se tiene conocimiento del entorno y a partir de una imagen se debe encontrar la pose de la cámara asociada a dicha imagen en ese entorno. Por lo general, son imágenes tomadas de manera esporádica en distintas locaciones. Por su parte, un sistema de VO debe funcionar en ambientes para los que no se cuenta con información *a priori* del entorno. Además, VO se basa en la estimación secuencial de poses sobre imágenes que corresponden a un movimiento continuo de la cámara. PoseNet realiza la estimación mediante una CNN. Más específicamente, la red de PoseNet es entrenada con un conjunto de imágenes de una escena y testeada con otro conjunto de imágenes, pero que pertenecen al mismo ambiente. La arquitectura utilizada está basada en GoogLeNet [21] y utiliza más de 20 capas convolucionales. Dicho método funciona en tiempo real, mostrando buen rendimiento tanto en ambientes interiores como exteriores y en condiciones de tiempo desfavorables.

En este capítulo se presentaron diferentes métodos de VO que componen el estado del arte, junto a sus fortalezas y debilidades. Si bien los métodos basados en *Deep Learning* han demostrado resultados prometedores en VO, su rendimiento y robustez aún está por detrás de aquellos métodos basados en *features* y directos. En el siguiente capítulo se detallan conceptos y definiciones de *Deep Learning* y conceptos básicos de geometría proyectiva necesarios para comprender el trabajo propuesto en esta tesina.

Capítulo 3

Conceptos previos

En este capítulo se presenta en detalle el marco conceptual utilizado a lo largo del presente trabajo. La mayoría de los conceptos sobre aprendizaje automatizado son tratados en los libros [46] y [47]. Por su parte, los conceptos geométricos relacionados a la representación del movimiento de la cámara son expuestos en [9].

3.1. El módelo de cámara *pinhole*

El modelo *pinhole* provee una forma de relacionar los píxeles en una imagen 2D con los puntos 3D que representan en el mundo y es uno de los modelos más utilizados para representar una cámara. Una cámara *pinhole* ideal es una cámara sin lentes y con un pequeño hoyo de apertura, mediante el cual pasan los rayos de luz y proyectan una imagen invertida en el lado opuesto de la cámara. En la presente sección se utiliza la siguiente notación: $\dot{\mathbf{x}} = [x \;\; y \;\; z \;\; 1]^\top$ es la representación en coordenadas homogéneas de $\mathbf{x} = [x \;\; y \;\; z]^\top$.

Consideremos el plano $Z = f$, al que llamaremos plano de la imagen, donde f es la distancia focal. Se denomina eje principal a la línea que pasa por el origen de coordenadas y que es perpendicular al plano de la imagen. Este punto en el que, precisamente, el eje principal encuentra al plano de la imagen se denomina punto principal. En este modelo, un punto $\mathbf{x} = [x \;\; y \;\; z]^\top$ en el mundo es mapeado en el plano de la imagen en un punto $\mathbf{u} = [u \;\; v]^\top$. Este es el punto en el que la línea que une a $\mathbf{x}$ y el origen de coordenadas interseca dicho plano. Como puede verse en la Figura 3.1, por semejanza de triángulos podemos concluir que $x/z = u/f$ y, análogamente, $y/z = v/f$. Luego, $\mathbf{u} = \left[\frac{fx}{z} \;\; \frac{fy}{z}\right]^\top$.

La anterior expresión asume que el origen de coordenadas en el plano de la imagen es el punto principal. Como esto no necesariamente es así, se obtiene una expresión más general de este mapeo:

$$\mathbf{u} = \left[\frac{fx}{z} + p_x \quad \frac{fy}{z} + p_y\right]^\top,$$

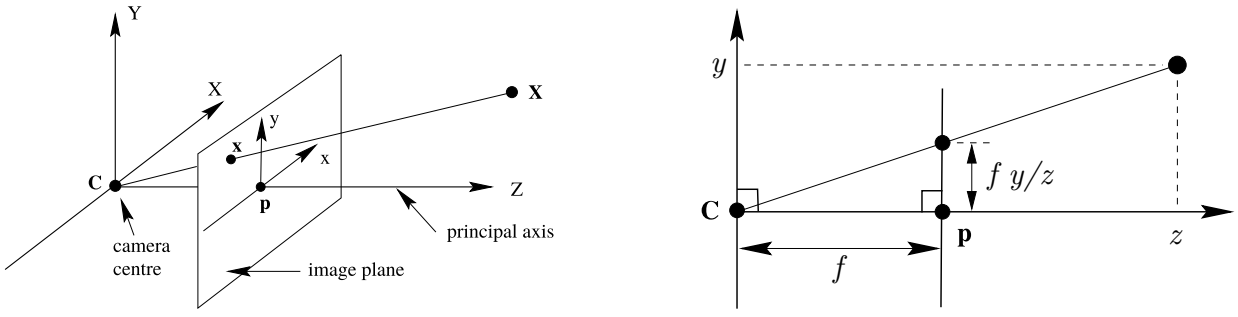

Figura 3.1: Modelo de cámara *pinhole*. **C** es el centro de la cámara, **p** es el punto principal y f es la distancia focal. Figura extraída de [9].

donde $[p_x \;\; p_y]$ son las coordenadas del punto principal en este nuevo sistema de coordenadas del plano de la imagen. En su forma matricial, resulta:

$$\mathbf{u} = \pi\left(\mathbf{K}\,\mathbf{x}\right), \tag{3.1}$$

donde

$$\mathbf{K} = \begin{bmatrix} f & 0 & p_x \\ 0 & f & p_y \\ 0 & 0 & 1 \end{bmatrix},$$

y $\pi : \mathbb{R}^n \to \mathbb{R}^{n-1}$ se define como:

$$\pi\left(\left[a_1 \ldots a_n\right]^\top\right) = \frac{1}{a_n} \begin{bmatrix} a_1 \\ \vdots \\ a_{n-1} \end{bmatrix}.$$

La matriz $\mathbf{K}$ es conocida como matriz de parámetros intrínsecos o matriz de calibración de la cámara.

Los puntos considerados en el mapeo hasta aquí, son puntos expresados en un sistema de coordenadas conocido como sistema de coordenadas de la cámara. Los puntos 3D del mundo generalmente se expresan en otro sistema de coordenadas fijo e independiente del movimiento de la cámara conocido como sistema de coordenadas del mundo. El sistema de coordenadas de la cámara y el sistema de coordenadas del mundo están relacionados mediante una transformación rígida $\mathbf{T}_{cw}$. Este tipo de transformaciones preservan las distancias entre puntos y las formas de los objetos. En particular, $\mathbf{T}_{cw}$ se define como

$$\mathbf{T}_{cw} = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0} & 1 \end{bmatrix}, \tag{3.2}$$

donde $\mathbf{R}$ es una matriz de rotación 3×3 y $\mathbf{t}$ es un vector de traslación perteneciente a $\mathbb{R}^3$. Luego,

$$\mathbf{x}_c = \pi\left(\mathbf{T}_{cw}\,\dot{\mathbf{x}}_w\right), \tag{3.3}$$

donde $\mathbf{x}_w$ es un punto expresado en coordenadas del sistema del mundo, y $\mathbf{x}_c$ es el mismo punto expresado en el sistema de coordenadas de la cámara. De esta forma queda expresado cómo se relacionan dos sistemas de coordenadas. La transformación $\mathbf{T}_{cw}$ pertenece al grupo Euclídeo Especial $\mathrm{SE}(3)$ (en inglés, *Special Euclidean group*). Utilizaremos elementos de este grupo para relacionar dos sistemas de coordenadas, como veremos en la Sección 3.1.1.

Finalmente, combinando las ecuaciones (3.1) y (3.3), la proyección de un punto 3D en el mundo al plano de la imagen está dada por:

$$\mathbf{u} = \pi(\mathbf{K}\ \pi(\mathbf{T}_{cw}\,\dot{\mathbf{x}}_w)).$$

3.1.1. Representación del movimiento de la cámara

En el presente trabajo, intentaremos estimar el movimiento realizado por un robot observando imágenes capturadas por una cámara. Por lo tanto, es necesaria una representación del movimiento de la cámara.

Como vimos en la Sección 3.1, al capturar una imagen se define un sistema de coordenadas de la cámara cuyo origen es el centro de la cámara. Por lo tanto, cada vez que la cámara se mueve, se define un nuevo sistema de coordenadas. Dijimos que una transformación $\mathbf{T} \in \mathrm{SE}(3)$ se puede utilizar para relacionar dos sistemas de coordenadas. Por lo tanto, utilizaremos transformaciones de $\mathrm{SE}(3)$ para representar el movimento de una cámara. Formalmente, $\mathrm{SE}(3)$ es un grupo definido como:

$$\mathrm{SE}(3) = \left\{ \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0} & 1 \end{bmatrix} \in \mathbb{R}^{4\times 4} \;\mid\; \mathbf{R} \in \mathrm{SO}(3),\, \mathbf{t} \in \mathbb{R}^3 \right\},$$

donde $\mathrm{SO}(3)$ es el grupo formado por las matrices de rotación:

$$\mathrm{SO}(3) = \left\{ \mathbf{R} \in \mathbb{R}^{3\times 3} \;\mid\; \mathbf{R}^\top\mathbf{R} = \mathbf{I},\, \det(\mathbf{R}) = 1 \right\}.$$

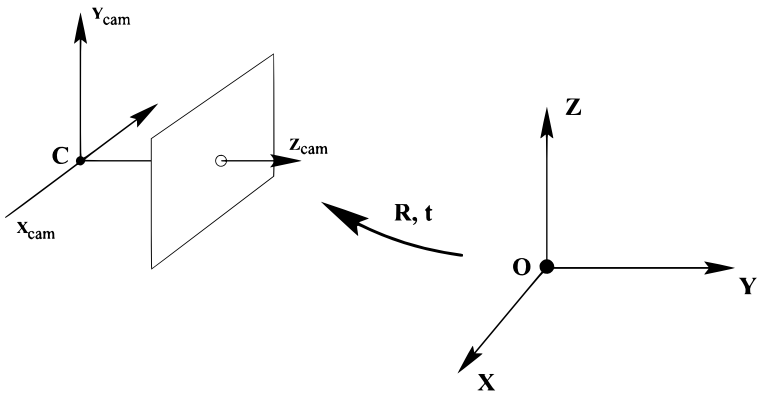

Figura 3.2: Transformación euclídea entre el sistema de coordenadas del mundo y el sistema de coordenadas de la cámara. Figura extraída de [9].

En la ecuación 3.2 definimos $\mathbf{T}_{cw} = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0} & 1 \end{bmatrix}$ como la transformación que convierte puntos del sistema de coordenadas del mundo al sistema de coordenadas de la cámara. La Figura 3.2 ilustra este punto. Su inversa viene dada por

$$\mathbf{T}_{wc} = \mathbf{T}_{cw}^{-1} = \begin{bmatrix} \mathbf{R}^\top & -\mathbf{R}^\top \mathbf{t} \\ \mathbf{0} & 1 \end{bmatrix}.$$

La transformación $\mathbf{T}_{wc}$ convierte puntos de la cámara al sistema de coordenadas del mundo. Por lo tanto, si consideramos el origen de coordenadas $\mathbf{o}_c = [0, 0, 0]^\top$ del sistema de la cámara, el cual es la posición de la misma en dicho sistema, resulta que $\mathbf{p}$ definido por

$$\mathbf{p} = \pi \left(\mathbf{T}_{wc} \dot{\mathbf{o}}_c \right) = -\mathbf{R}^\top \mathbf{t},$$

representa la posición de la cámara en el mundo. Esto es particularmente útil cuando una trayectoria viene dada por una secuencia de transformaciones porque nos permite graficarla en el espacio.

El presente trabajo implementa una CNN cuya entrada son dos *frames* consecutivos de una cámara y su salida es la estimación del movimiento de la cámara entre ambos *frames*. Inicialmente, se podría implementar la red para que su salida sea una transformación $\mathbf{T} \in \mathrm{SE}(3)$. Lamentablemente, resulta difícil asegurar la condición de que la parte rotacional de $\mathbf{T}$ sea una matriz ortogonal y con determinante igual a 1. Por lo tanto, en problemas de optimización se utilizan representaciones alternativas, como lo son los *quaternions*. Un *quaternion* es una representación inicialmente creada como una extensión de los números reales, de forma similar a los números complejos. Introduce las unidades imaginarias i, j, y k, donde $i^2 = j^2 = k^2 = -1$. Generalmente, se los representa de la forma:

$$a + bi + cj + dk,$$

donde a, b, c, y d son números reales. La norma de un *quaternion* $q = a + bi + cj + dk$ viene dada por:

$$\|q\| = \sqrt{a^2 + b^2 + c^2 + d^2}.$$

Los *quaternions* con norma 1, conocidos como *quaternions* unitarios, pueden utilizarse para representar rotaciones. Son una buena alternativa en problemas de optimización, ya que se parametrizan con 4 valores contra los 9 valores que requiere una matriz de $\mathrm{SO}(3)$. Además, resulta más simple asegurar la condición de que un *quaternion* tenga norma 1 que la ortogonalidad de una matriz. En consecuencia, la salida de la red está conformada por un *quaternion* junto a un vector de traslación $\mathbf{t}$. Si se convierte el *quaternion* a la matriz de rotación correspondiente, junto a $\mathbf{t}$ se obtiene una transformación $\mathbf{T} \in \mathrm{SE}(3)$.

A continuación, se presenta el problema de VO. En la presente tesina, abordaremos el problema de VO utilizando técnicas de *Deep Learning*. Más concretamente, utilizaremos CNNs, por lo que, posteriormente, introduciremos conceptos de *Deep Learning* relevantes a este trabajo.

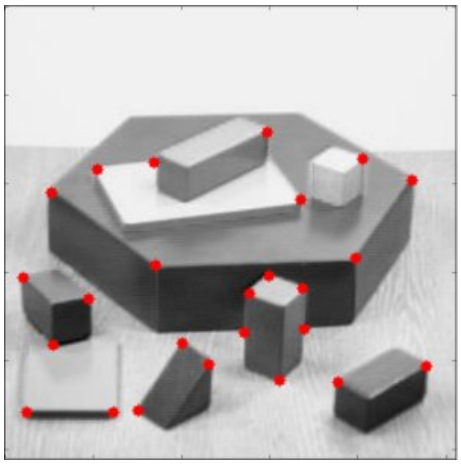

Figura 3.3: Diversos *features* extraídos de una imagen.

3.2. *Visual Odometry*

VO [3, 4] es un problema de gran relevancia en el campo de la robótica móvil. Se lo define como el proceso de estimar la posición y orientación (pose) de un robot incrementalmente a través del tiempo, usando como entrada las imágenes de una o más cámaras a bordo. Los métodos de VO permiten estimar el movimiento realizado por el robot a partir de examinar los cambios producidos en las imágenes capturadas durante su navegación.

La posición y la orientación de un robot determinan unívocamente la ubicación del mismo en el espacio, y se la denomina pose. El movimiento se modela como una serie de poses del robot en el tiempo. Para cada instante t, el par $(\mathbf{p}_t, \mathbf{o}_t)$ representa la pose del robot en ese momento, donde $\mathbf{p}_t$ es la posición y $\mathbf{o}_t$ es la orientación correspondiente. Si I_t es una imagen (o conjunto de imágenes) obtenida en el tiempo t, VO estima la pose a partir de examinar los cambios entre I_t y I_{t-1}.

En general, los métodos de VO se pueden dividir en dos grandes grupos: métodos basados en *features* y métodos directos. Los primeros determinan el movimiento luego de extraer y buscar correspondencias entre *features* de una secuencia de imágenes, mientras que los métodos directos trabajan sobre la intensidad de los píxeles de las imágenes. Adicionalmente, los métodos también pueden clasificarse según la cantidad de cámaras utilizadas, siendo estos monoculares, en el caso en que se utilice una única cámara o estéreo en caso de que se utilicen dos o más. La ventaja de las cámaras estéreo es que proveen información acerca de la profundidad, es decir, es posible calcular a qué distancia están los objetos de la cámara. Esto permite realizar una reconstrucción precisa de la escena 3D a partir de las imágenes observadas y conocer la relación entre tamaños en el mundo y en la imagen. Debido a que los métodos monoculares no conocen esta relación, sólo pueden estimar una versión a escala de la trayectoria real.

A fin de ejemplificar, en lo que resta de la sección se detallan las distintas etapas involucradas en la implementación de un método basado en *features* estéreo. El lector debe tener en cuenta que existen múltiples variantes de métodos de VO y que lo que aquí se describe tiene únicamente como finalidad clarificar el problema descripto en esta sección y abordado en esta tesina. Para conocer en concreto un método de VO, ver el capítulo 2 y la bibliografía correspondiente.

En primera instancia, un método basado en features recibe las imágenes capturadas por las cámaras y busca en ellas puntos salientes y fácilmente distinguibles visualmente, como por ejemplo, esquinas y bordes. Esto es posible a partir de analizar la intensidad de los píxeles y la diferencia de intensidad con los píxeles vecinos. Estos puntos se conocen como *features* y son fácilmente reconocibles de una imagen a otra. En la Figura 3.3 se muestran ejemplos de *features* detectados en una imagen. Entre los extractores de *features* más destacados podemos mencionar a SIFT [48], SURF [49], ORB [35], y BRISK [50]. Los mencionados *features* son almacenados junto a información de su vecindario de modo tal que esa información local, conocida como *feature descriptor*, sirva como identificador del mismo entre distintas imágenes.

A su vez, al trabajar con cámaras estéreo obtenemos dos (o más) imágenes de la misma escena. Como paso posterior a la extracción de *features*, se realiza una asociación entre *features* de ambas imágenes mediante los correspondientes *descriptors* con el objetivo de identificar aquellos pares de *features* que corresponden al mismo punto de la escena. Esta etapa es conocida como *feature matching*. Si conocemos la distancia o relación que hay entre ambas cámaras estéreo, a partir de estos pares de features es posible calcular los puntos 3D del mundo que representan mediante la técnica conocida como triangulación.

Posteriormente, para una nueva imagen entrante se realiza el mismo proceso. Entre *frames* consecutivos también se realiza una asociación de *features*. Este proceso muchas veces involucra un *feature tracker*, como por ejemplo KLT [51, 52]. Un *feature tracker* permite realizar un seguimiento de un *feature* en una secuencia de imágenes.

Una vez obtenidas las correspondencias 3D-2D entre puntos y *features*, se minimiza el error de reproyección para ajustar una transformación $\mathbf{T} \in \mathrm{SE}(3)$. Esto implica minimizar la distancia entre la proyección de un punto en una imagen y su medición. Para esto, generalmente se utiliza un algoritmo de optimización como Levenberg-Marquardt o Gauss-Newton.

En resumen, un método de VO resulta en un proceso iterativo e incremental. Debido a que los cálculos se basan en estimaciones previas, sumado a posibles mediciones con ruido por parte de los sensores, los métodos de VO acumulan error a lo largo del tiempo. Vale destacar que un método de VO no necesariamente construye un mapa del entorno que se transita. Aquellos métodos que sí mantienen un mapa pueden corregir la trayectoria a partir de técnicas como *loop closure,* que implica reconocer lugares previamente visitados.

Por su parte, recientemente se han utilizado CNNs para resolver el problema de VO. Wang et al. [16] proponen un modelo supervisado basado en redes recurrentes en el que se estiman las poses directamente a partir de imágenes RGB. En [22], los autores proponen un esquema de entrenamiento no supervisado, y predicen simultáneamente la pose y el mapa de profundidad de una imagen. Nuestro objetivo es entrenar una CNN para estimar la pose de la cámara. Por tal motivo, en las siguientes secciones se describen distintos conceptos de Deep Learning necesarios para comprender el método propuesto.

3.3. Redes neuronales artificiales

Las redes neuronales [46] son modelos computacionales ampliamente estudiados en el campo del Aprendizaje automatizado. Se utilizan para aproximar funciones, cuya salida puede ser discreta o continua. Su éxito en los últimos años se debe a la precisión que muestran a la hora de realizar tareas específicas, y a la flexibilidad para adaptarlas a distintos problemas.

Las redes neuronales están formadas por unidades de cómputo interconectadas, llamadas neuronas artificiales. Las neuronas se disponen en grupos, denominados capas. Cada neurona recibe múltiples valores como entrada, y produce un valor como salida. Formalmente,

$$y = f\left(b + \sum_{i=1}^{n} w_i x_i\right), \tag{3.4}$$

donde x_i son los valores de entrada de la neurona, cada uno pesado por un valor real w_i denominado peso, b es un valor real denominado *bias*, f es una función no lineal, e y es la salida de la neurona. Luego, si el valor de salida de una neurona es el valor de entrada de otra neurona, se dice que ambas neuronas están conectadas. La Figura 3.4 muestra la representación de una neurona.

Las neuronas se organizan en capas. Usualmente, las capas están ordenadas, siendo la primera de ellas, la capa de entrada, y la última, la capa de salida. Al resto de las capas, se las suele llamar capas ocultas, o capas intermedias. Típicamente, neuronas de una capa están conectadas a neuronas de la capa siguiente. La capa inicial recibe valores reales como entrada, y la red los propaga a las capas subsiguientes, aplicando en cada neurona la fórmula (3.4), hasta llegar a la capa de salida. En la Figura 3.5 se muestra una representación de una red con tres capas como ejemplo.

Cada cómputo hecho en una neurona aplica una función no lineal, comúnmente llamada función de activación. La razón para introducir no linealidad en la red es que de esta forma podemos aprender funciones más complejas. De lo contrario, la red simplemente se comportaría como una función lineal, ya que sería una

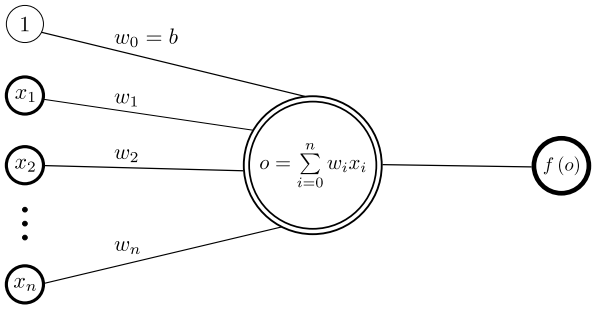

Figura 3.4: Representación de una neurona. Los valores x_i son las entradas de una neurona, w_i son los pesos, f es la función de activación y b es el *bias*.

combinación de múltiples funciones lineales, independientemente de cuántas capas tenga. Las neuronas de una misma capa aplican la misma función de activación.

Para llevar a cabo el entrenamiento de la red es necesario definir una función de costo. Esta función indica cuán buena es la red para una tarea determinada. Entrenar una red implica ajustar iterativamente los pesos w_j de todas las neuronas. A lo largo del entrenamiento, la red es alimentada con múltiples valores de entrada, conocidos como ejemplos de entrenamiento. Por su parte, la función de costo generalmente recibe como parámetros la salida de la red $\hat{y}_i$ para un ejemplo de entrenamiento X_i y el valor esperado y_i para dicho ejemplo. Luego, el objetivo es minimizar la función de costo a través del entrenamiento. El método más utilizado para entrenar es *backpropagation* utilizando el algoritmo del descenso por gradiente. Básicamente, mediante este método se calcula la derivada de la función de costo respecto a los pesos, en cada paso de entrenamiento:

$$\nabla w_j = \frac{\partial L}{\partial w_j},$$

y posteriormente se actualizan los pesos:

$$w_j = w_j - \alpha \nabla w_j,$$

donde L es la función de costo, y α es un hiperparámetro llamado *learning rate*, que sirve para poner en cierta escala la magnitud de la actualización del peso.

Existen diferentes tipos de entrenamiento. Cuando los datos de entrenamiento consisten de ejemplos de entrenamiento X_i y la salida esperada y_i de la función que intentamos aproximar con la red neuronal para cada uno de los ejemplos de entrenamiento, hablamos de aprendizaje supervisado. Al conjunto de valores y_i se lo suele llamar etiquetas o *target* (dependiendo de si son valores discretos o continuos). Por otro lado, cuando no se cuentan con los valores y_i, hablamos de aprendizaje no supervisado. Dependiendo del problema, se pueden resolver tareas supervisadas con algoritmos de aprendizaje no supervisado, para las cuales se suele derivar el *target* a partir de características de la entrada.

Uno de los problemas que puede surgir al entrenar es el sobreajuste. El sobreajuste se produce cuando el modelo no puede generalizar correctamente debido a que se ajustó demasiado a los datos de entrenamiento, y por lo tanto no estima correctamente al observar datos nuevos.

3.4. Redes neuronales convolucionales

Las redes neuronales convolucionales (CNN, acrónimo de *Convolutional Neural Networks*) son un tipo especial de redes neuronales artificiales. Este tipo de red neuronal se aplica comúnmente en tareas en las que se utiliza material visual como imágenes o videos. Fueron propuestas originalmente en la década del '80 [53], aunque recién a partir del año 2012 se empezaron a utilizar masivamente en el campo de visión artificial [12].

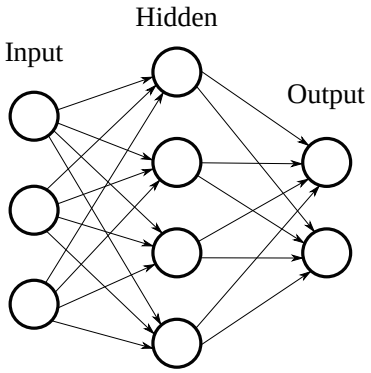

Figura 3.5: Representación de una red neuronal con una capa de entrada con tres neuronas, una capa oculta con cuatro neuronas, y una capa de salida con dos neuronas.

Una CNN se caracteriza por tener una o más (en general, varias) capas convolucionales. Una capa convolucional es un tipo de capa particular, en el que las conexiones entre neuronas son locales, es decir, una neurona recibe la salida de un subconjunto acotado de neuronas de la capa anterior. Esto se contrapone con las capas *fully-connected*, en las que una neurona recibe la salida de todas las neuronas de la capa anterior. Una de las ventajas de utilizar capas convolucionales es que los pesos son compartidos entre conexiones, haciendo que el número total de parámetros (pesos) a entrenar por la red sea menor que el de una red que utiliza sólo capas *fully-connected*.

Una capa convolucional se basa en la operación conocida como convolución. Sea $\mathbf{I}$ una matriz de dimensiones $h \times w \times d$, y $\mathbf{K}$ una matriz de dimensiones $f_h \times f_w \times d$. El resultado de la convolución entre $\mathbf{I}$ y $\mathbf{K}$ es una matriz $\mathbf{C}$ con dimensiones $(h - f_h + 1) \times (w - f_w + 1) \times 1$, en donde:

$$\mathbf{C}(i,j) = \sum_m \sum_n \sum_p \mathbf{I}(i+m, j+n, p)\, \mathbf{K}(m,n,p)\,.$$

En el contexto de visión artificial, $\mathbf{I}$ es la representación computacional de una imagen, cuyos valores representan los valores de intensidad de los píxeles, $\mathbf{K}$ representa un filtro (o *kernel*), y la convolución entre $\mathbf{I}$ y $\mathbf{K}$ es la aplicación de un filtro sobre la imagen que permite realizar distintas operaciones tales como detección de bordes, reducción de ruido, y mejora de la nitidez, entre otras. En la Figura 3.6 se muestra un ejemplo de convolución. En lo que a redes neuronales respecta, una capa convolucional contiene n filtros. Dicha capa recibe como entrada una imagen, y se realiza una convolución con cada uno de los filtros. Posteriormente, se aplica una función de activación no lineal, como se ve en la ecuación (3.4). Al resultado se lo conoce como *feature map*. Los *feature maps* serán entrada de las subsiguientes capas convolucionales. Nótese que la definición (3.4) es consistente con lo detallado en este párrafo, si consideramos que los filtros representan los pesos, y que estos pesos son compartidos entre distintas neuronas.

Uno de los motivos fundamentales por el que se utilizan capas convolucionales cuando se trabaja con imágenes es debido a que preservan la estructura espacial de los datos. En otras palabras, la convolución aprovecha la información que otorgan píxeles cercanos ya que opera sobre regiones de la imagen. La idea es que cada filtro permite detectar un patrón en la imagen. Las capas convolucionales se suelen agrupar en distintos bloques. Los filtros de los primeros bloques detectan patrones simples como líneas y bordes, y a medida que se llega a las capas superiores se detectan abstracciones de alto nivel. Por ejemplo, si se trabaja con imágenes de rostros, las primeras capas identifican líneas rectas y curvas, mientras que capas superiores podrían llegar a identificar distintas partes de un rostro como ojos, nariz, o boca.

Otro tipo de capa es la denominada capa de *pooling*. Este tipo de capa se suele utilizar para reducir los requerimientos computacionales progresivamente a través de la red, y además, para minimizar la probabilidad de sobreajuste, ya que disminuye la dimensionalidad de los *feature maps*. De esta manera, se mantiene un modelo más simple, evitando que la red se concentre en detalles irrelevantes para la tarea abordada durante el entrenamiento. La capa de *pooling* define una ventana, la cual opera sobre regiones de la imagen aplicando

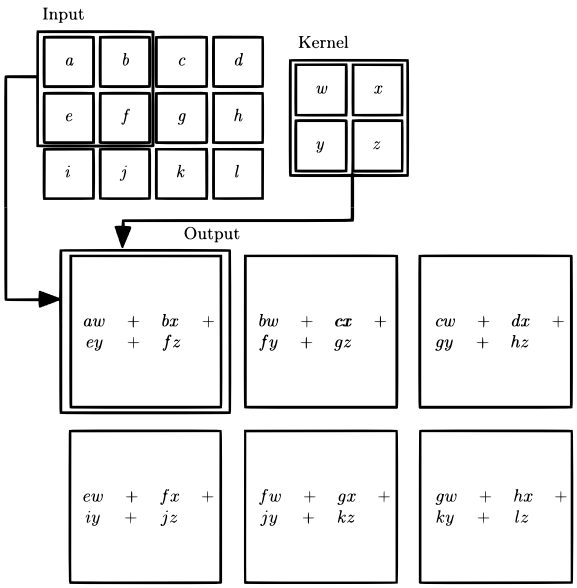

Figura 3.6: Ejemplo de una convolución 2D entre una imagen (*Input*) de tamaño 4×3 y un filtro (*Kernel*) de 2×2. El resultado (*Output*) es una imagen de tamaño 3×2. Figura extraída de [47].

una operación, convirtiendo los valores de la ventana en un único valor. La operación más utilizada es el máximo de los valores de la ventana, en esa caso la capa se conoce como *max-pooling*.

3.4.1. *Batch Normalization*

La técnica de *batch normalization* [54] permite acelerar el aprendizaje de una red sin afectar la estabilidad durante el entrenamiento. Básicamente, consiste en normalizar la salida de cada capa oculta. Debido a la dificultad para calcular la media y la varianza para todos los datos de entrenamiento, se calculan estas estadísticas por cada *batch* $\mathcal{B}$.

$$\mu_{\mathcal{B}} = \frac{1}{m}\sum_{i=1}^{m} x_i,$$

$$\sigma^2_{\mathcal{B}} = \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2.$$

Luego, se realiza la normalización:

$$\hat{x_i} = \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}},$$

donde $\epsilon \neq 0$ para evitar una división por 0. Finalmente, la salida está dada por:

$$y_i = \gamma \hat{x_i} + \beta,$$

donde γ y β son parámetros a entrenar (de igual forma que los pesos), permitiendo de esta forma que el modelo aprenda la distribución de salida.

3.4.2. ReLU

Una de las funciones de activación más utilizadas es el rectificador. Se define como:

$$f(x) = \text{máx}\,(0, x)\,.$$

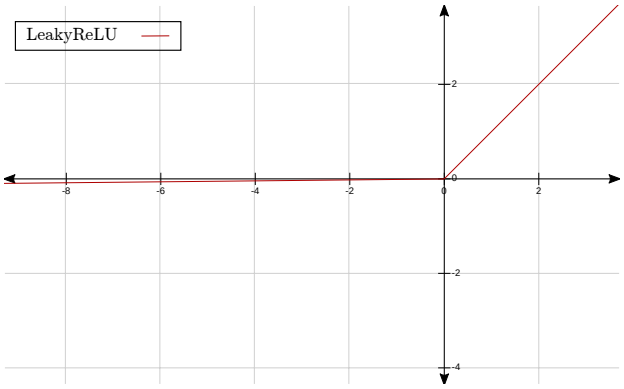

Figura 3.7: Gráfico de la función de activación LeakyReLU con $a = 0.01$.

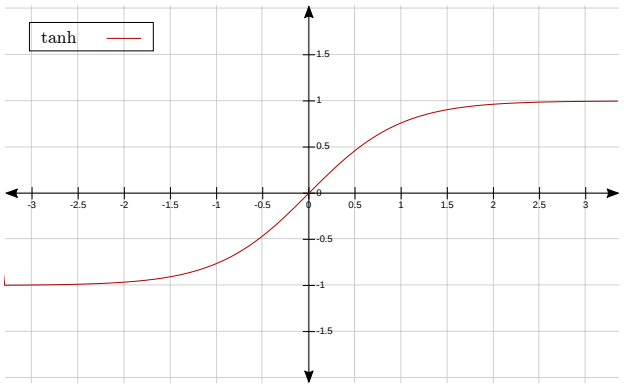

Figura 3.8: Gráfico de la función de activación tanh.

A las unidades que utilizan esta función de activación se las suele conocer como unidad lineal rectificada o ReLU (del inglés, *Rectified Linear Unit*)[55], aunque en general, al rectificador se lo suele llamar directamente ReLU. Su uso se debe a que son computacionalmente simples de calcular, y que suelen converger más rápido que otras alternativas como la tangente hiperbólica o la sigmoidea. Nótese que esta función es constante para valores de entrada negativos. Por lo tanto, su derivada es 0 en esos valores. Esto implica que (si ocurriese para todos los valores del batch) cuando los valores de entradas para ReLU sean negativos, los pesos no se actualizarían, provocando la "muerte" de la neurona. Para solucionar este problema, se suele utilizar una variante conocida como LeakyReLU [56]. Se define como:

$$f(x) = \begin{cases} x & x > 0 \\ ax & x \leq 0 \end{cases},$$

donde a suele ser 0.01. La Figura 3.7 muestra un gráfico de la función LeakyReLU con $a = 0.01$.

3.4.3. Tangente hiperbólica (tanh)

Otra función de activación comúnmente usada es tanh. Una de sus principales ventajas es que su rango es $(-1, 1)$. El hecho de que el rango esté limitado implica que las activaciones no crecerán a grandes magnitudes. Esto es una propiedad deseable ya que, de lo contrario, resultaría en actualizaciones de pesos con valores muy grandes, volviendo inestable el entrenamiento de la red. En la Figura 3.8 se presenta el gráfico de la función tanh.

Figura 3.9: Imágenes generadas por una red GAN. La red se entrenó sobre un dataset de dormitorios [57]. Figura extraída de [14].

3.5. Redes generativas adversarias

Las Redes Adversarias Generativas, conocidas como GAN (del inglés, *Generative Adversarial Networks*) [13] están constituidas por dos CNNs con roles definidos que se entrenan en un esquema competitivo. Por un lado se tiene un Generador G, que es una CNN que recibe un vector $\mathbf{z}$ de números aleatorios y devuelve una imagen. Y por otro lado el Discriminador D, que es una CNN que recibe una imagen de entrada y devuelve un escalar en el intervalo $[0, 1]$. El Discriminador D se entrena en forma estándar para resolver el siguiente problema de clasificación binario: dada una imagen devolver la probabilidad de que esta provenga del dataset (que sea "real"), en oposición a que sea una imagen generada por el Generador G (que sea "falsa"). Este entrenamiento se consigue actualizando los parámetros de D haciendo un descenso por gradiente estocástico sobre una función de costo J para el problema de clasificación binaria. En simultáneo a cada actualización de los parámetros de D, se actualizan los parámetros de G con el objetivo opuesto: aumentar esa función de costo J con un ascenso por gradiente, es decir, con el objetivo de "engañar" al Discriminador. Este esquema de entrenamiento adversario ha despertado gran interés en el área, dada la calidad de los resultados obtenidos, pero por sobre todo por la potencialidad que implica entrenar modelos en forma no supervisada.

Formalmente, D y G compiten en una especie de juego de suma cero (juego en el que la ganancia de un participante es igual a la pérdida del oponente) en el que debemos optimizar la siguiente función:

$$\min_G \max_D L(D, G) = \mathbb{E}_{x \sim p_r(x)} \left[\log D\left(x\right)\right] + \mathbb{E}_{\mathbf{z} \sim p_z(\mathbf{z})} \left[\log\left(1 - D\left(G\left(\mathbf{z}\right)\right)\right)\right],$$

donde p_r es la distribución de los datos sobre las muestras reales (en este contexto, las muestras del dataset), y p_z es la distribución del ruido que toma G como entrada. Durante el entrenamiento, alternadamente se optimizan D y G. Cabe destacar que el entrenamiento del Generador no se realiza directamente con los datos reales, sino que sus pesos se actualizan a través de la retroalimentación que obtiene del Discriminador. En la Figura 3.9 se muestran imágenes generadas por una red GAN entrenada sobre un dataset [57] compuesto por imágenes de dormitorios.

3.5.1. Wasserstein GAN

Las primeras versiones de GAN eran difíciles de entrenar debido al delicado equilibrio del entrenamiento adversario [11]. Uno de los principales problemas que sufre la formulación original de GAN es el denominado desvanecimiento del gradiente. Arjovsky et al. [58] demostraron que a medida que el Discriminador mejora, el gradiente del Generador se desvanece. Un gradiente muy pequeño hace que en cada paso del entrenamiento el valor de los pesos cambie muy poco, dificultando notablemente el aprendizaje. En el peor de los casos, el aprendizaje del Generador se podría llegar a detener por completo.

Una de las soluciones propuestas es cambiar la función de costo [25]. La distancia de Wasserstein es una medida de la distancia entre dos distribuciones de probabilidad. Se define como:

$$W\left(p_r, p_g\right) = \inf_{\gamma \in \prod\left(p_r, p_g\right)} \mathbb{E}_{(x,y) \sim \gamma} \left[\|x - y\|\right],$$

donde $\prod(p_r, p_g)$ es el conjunto de todas las distribuciones conjuntas cuyas distribuciones marginales son p_r y p_g, respectivamente. Intuitivamente, podría pensarse como la cantidad de "esfuerzo" requerida para convertir la distribución p_g en p_r. En este contexto, p_g es la distribución de los datos generados por el Generador.

Calcular todas las posibles distribuciones conjuntas en $\prod(p_r, p_g)$ es un problema intratable. Por esto, no podemos utilizar $W(p_r, p_g)$ como una función de costo. En su lugar, los autores proponen transformar esta fórmula basándose en la dualidad de Kantorovich-Rubinstein [59], obteniendo así

$$W(p_r, p_g) = \sup_{\|f\|_L \leq 1} \mathbb{E}_{x \sim p_r}[f(x)] - \mathbb{E}_{x \sim p_g}[f(x)].$$

Esta nueva fórmula requiere que $\|f\|_L \leq 1$, es decir, que f sea 1-Lipschitz continua. Una función $f : \mathbb{R} \to \mathbb{R}$ es 1-Lipschitz continua si

$$|f(x_1) - f(x_2)| \leq |x_1 - x_2| \ \forall x_{1,} x_2 \in \mathbb{R}.$$

Entonces, adaptando esta fórmula a nuestro problema, en lugar de tener un Discriminador que distingue muestras reales de muestras generadas por G, tenemos una función 1-Lipschitz continua D. La salida de esta función ya no es una probabilidad en $[0, 1]$, sino que retorna un valor real. Luego, la optimización viene dada por

$$\min_G \max_{D \in \mathcal{D}} \mathbb{E}_{x \sim p_r}[D(x)] - \mathbb{E}_{\tilde{x} \sim p_g}[D(\tilde{x})],$$

donde $\mathcal{D}$ es el conjunto de todas las funciones 1-Lipschitz, y p_g es la distribución definida por $\tilde{x} = G(\mathbf{z})$, $\mathbf{z} \sim p(\mathbf{z})$. Los autores probaron que una función de costo basada en la distancia de Wasserstein tiene gradientes más suaves en todo su dominio, por lo tanto no sufre el problema del desvanecimiento del gradiente. Este modelo es conocido como Wasserstein GAN, o WGAN [25].

Surge el problema de tener que asegurar que D cumpla con la condición de ser Lipschitz continua durante el entrenamiento. Los autores proponen una solución simple: restringir los pesos w a una ventana $[-c, c]$, para algún valor real c [25]. Si bien resulta sencillo de implementar, esta técnica tiene sus desventajas. Por un lado, el modelo resulta ser muy sensible a este hiperparámetro. Por otro lado, y más importante, resulta ser muy restrictiva: el hecho de recortar los pesos limita la capacidad del modelo de aprender funciones complejas [26].

Alternativamente, para forzar la condición de que una función sea 1-Lipschitz, Gulrajani et al. [26] proponen y demuestran que sean p_r y p_g dos distribuciones, y f^* una función 1-Lipschitz y solución óptima de $\max_{\|f\|_L \leq 1} \mathbb{E}_{y \sim p_r}[f(y)] - \mathbb{E}_{x \sim p_g}[f(x)]$, entonces la norma del gradiente de f^* es 1 en casi todas partes. En base a este teorema, podemos agregar un término de penalización del gradiente a la función de costo:

$$L = \mathbb{E}_{\tilde{x} \sim p_g}[D(\tilde{x})] - \mathbb{E}_{x \sim p_r}[D(x)] + \lambda \mathbb{E}_{\hat{x} \sim p_{\hat{x}}}\left[(\|\nabla_{\hat{x}} D(\hat{x})\|_2 - 1)^2\right], \tag{3.5}$$

donde λ es un hiperparámetro, y $\hat{x} = t\tilde{x} + (1-t)x$, $0 \leq t \leq 1$. Esta es una versión que cumple con una restricción más débil. Debido a la intratabilidad de calcular la norma del gradiente para cada x, se hace el cálculo para un $\hat{x}$ elegido al azar, ya que se muestrea a partir de $\tilde{x}$ y x, y de un parámetro t tomado de una distribución uniforme entre 0 y 1. Este modelo es conocido como WGAN-GP [26].

En el presente capítulo se introdujo el marco teórico utilizado en esta tesina. Además de detallar el problema de VO, se presentó una representación del movimiento de la cámara. Por último, se definieron conceptos de *Deep Learning*, especificando qué tipo de redes neuronales implementamos en nuestro trabajo. En el próximo capítulo veremos estos conceptos aplicados a la resolución del problema de VO desarrollada en nuestra propuesta.

Capítulo 4

Método propuesto

En este capítulo se presenta WGANVO, el método propuesto para resolver el problema de odometría visual mediante redes neuronales profundas. Este método utiliza como entrada pares de *frames* consecutivos de una cámara monocular y retorna el desplazamiento estimado entre los dos momentos en los que fueron capturados los *frames*.

Para resolver el problema de odometría visual, se desarrolló y entrenó un modelo basado en WGAN con penalización del gradiente (WGAN-GP) [26]. El mismo está constituido por dos redes con roles bien definidos: un Generador G, el cual genera pares de *frames* a partir de un vector aleatorio recibido como entrada; y un Discriminador D, que recibe pares de *frames* como entrada, y se encarga de discernir si la entrada es un par generado por G, o un par obtenido desde el conjunto de datos de entrenamiento. A su vez, D fue adaptado para predecir el cambio de posición y orientación. Por lo tanto, a la salida de D se le añadió una capa como un vector de siete componentes: tres componentes que representan la estimación del cambio de posición entre los dos *frames,* y cuatro componentes que representan la estimación del cambio de orientación (*quaternion*).

4.1. Dataset

El dataset principal utilizado en el presente trabajo es KITTI [29]. En el capítulo 5 se muestra cómo se repartieron las secuencias en conjuntos de entrenamiento y conjunto de test. Este dataset cuenta con 22 sequencias de imágenes (numeradas comúnmente del 00 al 21) adquiridas a 10 fps con una cámara estéreo ubicada en la parte frontal de un auto. La Figura 4.1 muestra al vehículo empleado para capturar las imágenes. En las imágenes del dataset puede verse a dicho coche conduciendo por un área residencial, y en las mismas

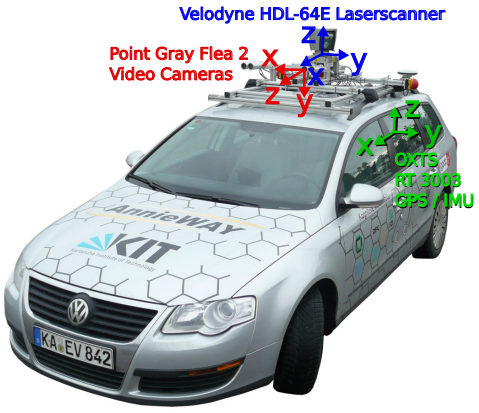

Figura 4.1: Auto utilizado para capturar el dataset KITTI. El mismo está equipado con cámaras estéreo, escáner láser 3D y sistema de navegación GPS/IMU. Imagen obtenida de [29].

Figura 4.2: Imagen perteneciente al dataset KITTI [29].

Figura 4.3: Pares de *frames* consecutivos del dataset KITTI. Fueron procesados para disminuir su resolución a 128×96 px.

pueden apreciarse otros autos y peatones, entre otros elementos típicos de una escena urbana. Como ejemplo, se muestra una imagen perteneciente al dataset en la Figura 4.2. Cabe destacar que las imágenes están desdistorsionadas y rectificadas. Las secuencias $\{00, 01, \ldots 10\}$ cuentan con la localización precisa para cada imagen en forma una matriz de transformación de tamaño 3×4, provista por los autores y obtenida mediante sensores como GPS y escaneo láser. Tal información es utilizada como *ground-truth*. Estas matrices toman un punto en el i-ésimo sistema de coordenadas correspondiente al *frame* i y lo proyectan en el sistema de coordenadas del mundo (ver Sección 3.1.1).

Las imágenes fueron preprocesadas de forma *offline*. Si bien el dataset corresponde a imágenes obtenidas con una cámara estéreo, solamente tomaremos las imágenes correspondiente a la cámara izquierda, ya que trabajaremos de forma monocular. La resolución, inicialmente de aproximadamente 1241×376 px, fue disminuida a 128×96 px. Para ello, se recortó la región central de la imagen con un tamaño de 500×375 px, y luego se reescaló al tamaño final. La razón principal para este cambio de resolución es disminuir los tiempos de cómputo; las CNNs suelen trabajar con resoluciones similares a la escogida. Adicionalmente, al recortar la región central de las imágenes descartamos información que resulta irrelevante para el entrenamiento o proveen información de menor interés, tales como el cielo, o los costados de una carretera. En la Figura 4.3 se pueden apreciar, a modo de ejemplo, tres pares de frames ya procesados.

Después del preprocesamiento, las imágenes fueron almacenadas de a pares de *frames* consecutivos junto a la transformación relativa entre ambos sistemas de coordenadas de la cámara. Más específicamente, consideremos a una secuencia como un conjunto de imágenes $\{I_1, I_2, \ldots, I_n\}$, y con un *ground-truth* asociado conformado por n matrices 3×4, donde la k-ésima matriz está asociada a la localización de la imagen I_k. Si a una matriz del *ground-truth* le añadimos una cuarta fila con valores $[0, 0, 0, 1]$, entonces obtenemos una matriz 4×4 perteneciente a $\mathrm{SE}(3)$, como puede verse en la Sección 3.1.1. Luego, podemos considerar que cada imagen I_k tiene asociada una matriz $\mathbf{T}_{wk} \in \mathrm{SE}(3)$ como *ground-truth*. Dicha transformación relaciona el sistema de coordenadas del k-ésimo *frame* con el sistema de coordenadas del mundo:

$$\dot{\mathbf{x}}_w = \mathbf{T}_{wk} \dot{\mathbf{x}}_k,$$

donde $\mathbf{x}_w$ es un punto expresado en coordenadas del mundo y $\mathbf{x}_k$ es un punto expresado en coordenadas de la cámara del k-ésimo *frame*. En este caso, en que la transformación relaciona el sistema de coordenadas de la cámara en un *frame* con el sistema de coordenadas del mundo, se suele hablar de pose absoluta. En particular, en KITTI el origen del sistema de coordenadas del mundo se establece en el lugar en el que empieza la navegación. Finalmente, el dataset procesado es almacenado como:

$$S = \left\{ \left((I_i, I_{i+1}), \mathbf{T}_{i(i+1)} \right) \mid i \in \{1, \ldots, n-1\} \right\}, \tag{4.1}$$

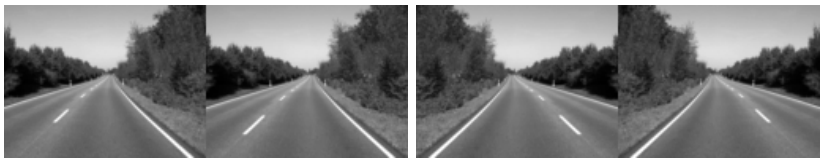

Figura 4.4: A la izquierda un par de *frames* del dataset original, y a la derecha, el correspondiente par espejado.

donde $\mathbf{T}_{i(i+1)}$ es la transformación dada por:

$$\mathbf{T}_{i(i+1)} = \mathbf{T}_{wi}^{-1}\mathbf{T}_{w(i+1)}.$$

A dicha transformación se la suele conocer como transformación relativa.

4.1.1. Dataset aumentado

Una de las claves al entrenar una CNN es contar con una gran cantidad de datos. Con el fin de contar con una mayor cantidad de imágenes, generamos nuevos *frames* a partir del dataset de KITTI. Para ello, en el presente trabajo se espejaron todas las imágenes para las cuales se cuenta con *ground-truth* (las secuencias $\{00, 01, \ldots 10\}$). Posteriormente, a estas imágenes se les aplica la misma reducción que a las originales, es decir, su resolución pasa a ser de 128×96 px. Finalmente, las transformaciones relativas pueden ser obtenidas a partir de las transformaciones relativas de las imágenes originales. Para cada par de imágenes espejadas $\left(I_i^*, I_{i+1}^*\right)$ obtenidas a partir de (I_i, I_{i+1}) con transformación relativa $\mathbf{T}_{i(i+1)} = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0} & 1 \end{bmatrix}$, la transformación relativa viene dada por:

$$\mathbf{T}_{i(i+1)}^* = \begin{bmatrix} \mathbf{R}^* & \mathbf{t} \\ \mathbf{0} & 1 \end{bmatrix},$$

donde

$$\mathbf{R}^* = \mathbf{MRM},$$

y

$$\mathbf{M} = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

En la Figura 4.4 se puede ver el resultado de espejar *frames* originales del dataset.

4.1.2. Construcción de una trayectoria

Hasta el momento, mencionamos que las secuencias de imágenes tienen asociada su trayectoria correspondiente representada con elementos de SE(3). Tal formato nos será de utilidad para la evaluación de la trayectoria estimada por la red pero resulta poco intuitivo de analizar visualmente ya que es una transformación entre dos sistemas de coordenadas. Si se quiere visualizar la trayectoria en el espacio, es posible recuperar la ubicación de la cámara para cada *frame,* representada en cada caso por un punto 3D. Estos puntos se expresan en el sistema de coordenadas del mundo.

Sea T un conjunto de transformaciones asociado a una secuencia de imágenes,

$$T = \left\{\mathbf{T}_{i(i+1)} \mid i \in \{1, \ldots, n-1\}\right\},$$

podemos entonces recuperar las transformaciones $\mathbf{T}_{wk}$, ya que el inicio del origen de coordenadas del mundo viene dado por el inicio de la navegación:

$$\mathbf{T}_{wk} = \prod_{i=1}^{k-1} \mathbf{T}_{i(i+1)}.$$

Figura 4.5: Trayectoria de la secuencia 00 de KITTI.

De acuerdo a lo explicado en la Sección 3.1.1, a partir de las transformaciones $\mathbf{T}_{wk}$ se obtiene la pose de la cámara en el espacio para cada k. En la Figura 4.5 se muestra la trayectoria para la secuencia 00 de KITTI.

4.2. Estructura de WGANVO

El modelo utilizado está basado en WGAN-GP. En particular, se experimenta con distintas arquitecturas de Generador y Discriminador. En el capítulo 5 se detallan las diferentes arquitecturas empleadas. En esta sección se comenta a grandes rasgos los cambios hechos a la formulación original de WGAN-GP. Tales modificaciones aplican a cualquiera de las arquitecturas con las que se experimenta. Un esquema general puede observarse en la Figura 4.6.

4.2.1. Generador

El Generador recibe como entrada un vector aleatorio tomado de una distribución normal. A partir de dicha entrada, se aplican capas generalmente convolucionales, para finalmente, devolver como entrada un par de imágenes generadas. Dichas imágenes tienen la misma resolución que aquellas pertenecientes al dataset de entrenamiento. Esta salida se utiliza posteriormente para entrenar al Discriminador sin etiquetas, de la forma explicada en la Sección 4.3.

4.2.2. Discriminador

El Discriminador recibe como entrada un par de *frames* consecutivos de una secuencia. Posteriormente, se emplean capas convolucionales para extraer características visuales de las imágenes entrantes. Luego la red se divide en dos bloques. Por un lado se tiene una capa lineal, cuya salida es la probabilidad de que la entrada sea un par obtenido desde el dataset de entrenamiento (en oposición a que sea un par generado por G). Por otro lado, se tienen tres capas *fully-connected*, las cuales se encargan de estimar la transformación relativa entre los dos *frames* de entrada, representada por un vector de siete componentes, indicando el cambio en la posición y en la orientación. Tres de estos valores indican la estimación del cambio de posición, y los cuatro restantes representan un *quaternion*, indicando el cambio de orientación. Estos valores permiten recuperar la transformación relativa entre dos *frames* consecutivos, como se explica en la Sección 4.3.

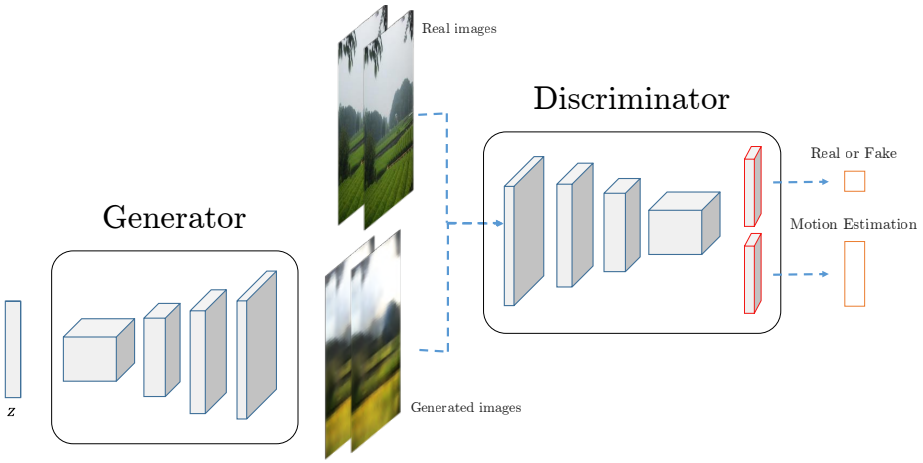

Figura 4.6: Esquema general de la red neuronal convolucional implementada. La entrada del Generador es un vector z aleatorio tomado de una distribución normal. La entrada del Discriminador consta de pares de *frames*, tanto generados como provenientes del dataset. Como puede verse, la salida del Discriminador fue adaptada para estimar la estimación del movimiento.

4.3. Entrenamiento

En este apartado se detalla el entrenamiento de nuestra CNN. Como se mencionó anteriormente, el Generador toma como entrada un vector aleatorio de 128 componentes sampleado de una distribución uniforme y su salida es un par de *frames* consecutivos. En la Figura 5.3 se pueden ver ejemplos de *frames* generados. El Discriminador es alimentado inicialmente con pares de imágenes generadas por el Generador y luego con pares de *frames* del dataset definido en la Sección 4.1. De esta manera, el Discriminador cuenta con dos fuentes de información. Por un lado, se espera que el Discriminador aprenda detalles de alto nivel al entrenarlo para separar *frames* reales de aquellos que son generados. Mientras que por otra parte, se entrena al mismo para resolver el problema de VO.

Los ejemplos de entrenamiento provenientes del dataset tienen la forma $\left(\left(I_i, I_{i+1}\right), \left(\mathbf{x}_{i(i+1)}, \mathbf{q}_{i(i+1)}\right)\right)$, donde I_i, I_{i+1} son *frames* consecutivos de una secuencia de video, $\mathbf{x}_{i(i+1)}$ es un vector traslacion en $\mathbb{R}^3$ entre los tiempos en que fueron capturadas I_i e I_{i+1}, y $\mathbf{q}_{i(i+1)}$ es el cambio de orientación asociado, representado por un *quaternion* unitario. Debido a que el *ground-truth* se almacenó como elementos de SE (3), durante la carga de datos se toma la parte rotacional de cada transformación relativa $\mathbf{T}_{i(i+1)}$ y se la convierte al *quaternion* unitario asociado $\mathbf{q}_{i(i+1)}$, mientras que la parte traslacional de $\mathbf{T}_{i(i+1)}$ resulta en $\mathbf{x}_{i(i+1)}$. Como ya se mencionó, la salida del Discriminador adaptado está compuesta por la estimación del vector traslacional $\hat{\mathbf{x}}_{i(i+1)}$, y la estimación del *quaternion* correspondiente $\hat{\mathbf{q}}_{i(i+1)}$ (en adición a la salida probabilística característica del Discriminador). Dado que $\hat{\mathbf{q}}_{i(i+1)}$ no suele ser un *quaternion* unitario, en tiempo de test se divide a $\hat{\mathbf{q}}_{i(i+1)}$ por su norma. En la Sección 3.1.1 se explica por qué se usan *quaternions* para entrenar.

Además de la función de costo para el entrenamiento adversario presentada en la ecuación (3.5), se tiene una función de costo adicional representando la parte supervisada del entrenamiento. La misma se minimiza con el objetivo de que la red aprenda a estimar la transformación relativa.

4.3.1. Función de costo

La parte supervisada del entrenamiento emplea una función de costo que minimiza la parte rotacional y la parte traslacional por separado. Para la parte traslacional se define:

$$L_{\mathbf{x}}\left(I_m, I_n\right) = \left\|\mathbf{x} - \hat{\mathbf{x}}\right\|_2,$$

donde $\mathbf{x}$ es el cambio de posición asociado a (I_m, I_n) y $\hat{\mathbf{x}}$ es la estimación del cambio de posición de la red cuando la entrada es el par de *frames* (I_m, I_n). A su vez, para la parte rotacional se define:

$$L_{\mathbf{q}}\left(I_m, I_n\right) = \left\|\mathbf{q} - \hat{\mathbf{q}}\right\|_2,$$

donde $\mathbf{q}$ es el cambio de orientación asociado a (I_m, I_n) y $\hat{\mathbf{q}}$ es la estimación del cambio de orientación de la red cuando la entrada es (I_m, I_n). Finalmente, se define la siguiente función de costo, basada en el trabajo de Kendall *et al.* [60]

$$L_{\beta}\left(I_m, I_n\right) = L_{\mathbf{x}}\left(I_m, I_n\right) + \beta L_{\mathbf{q}}\left(I_m, I_n\right),$$

donde β es un hiperparámetro a ajustar para darle la misma importancia al error de orientación y al error de posición, ya que suelen estar en distintas escalas. En el capítulo 5 se evalúa una función de costo alternativa basada en el error de reproyección a partir de puntos 3D de la escena calculados de forma *offline*.

El entrenamiento, por lo tanto, consta de minimizar inicialmente la función de costo de entrenamiento adversario de WGAN-GP presentada en la ecuación (3.5) durante cierto número de iteraciones y, posteriormente, minimizar L_{β}. La salida del Discriminador correspondiente a la estimación del movimiento para imágenes sintéticas como entrada es simplemente ignorada, ya que no aporta información al minimizar $L_{\beta}\left(I_m, I_n\right)$ debido a que no se cuenta con poses *ground-truth* para dichas imágenes. En el capítulo 5 se proveen más detalles del entrenamiento. Además, se analiza la influencia del entrenamiento semi-supervisado propuesto en nuestro trabajo.

En el presente capítulo se presentó el método propuesto para resolver el problema de VO. Se detalló el procesamiento del dataset junto a la estructura de la CNN basada en WGAN-GP y el entrenamiento realizado. En el próximo capítulo se exponen las evaluaciones realizadas sobre la CNN implementada.

Capítulo 5

Experimentos

En el presente capítulo se realiza una evaluación exhaustiva de WGANVO. Inicialmente, se presentan diferentes métricas empleadas en la evaluación de los métodos de VO. Luego, se exponen distintas arquitecturas de Generador y Discriminador implementadas en los experimentos y se describen en detalle las distintas pruebas realizadas para evaluar WGANVO. Las mismas incluyen una evaluación de la técnica semi-supervisada para analizar su influencia en la regresión de la pose, una comparación con el sistema del estado del arte ORB-SLAM2 [32] y la implementación de una función de costo alternativa basada en la reproyección de puntos 3D sobre la imagen. Finalmente, se realiza una discusión sobre el método propuesto contrastando el mismo con métodos similares que emplean otras técnicas de *Deep Learning*.

5.1. Métricas

En esta sección se presentan diferentes métricas [61] para evaluar la precisión de un método de VO. RPE (del inglés *Relative Pose Error*) es una métrica que se enfoca en la precisión local de un método, por lo que resulta particularmente útil para métodos de VO. Asumiendo que la trayectoria estimada viene dada por $\hat{\mathbf{T}}_{w1}, \dots \hat{\mathbf{T}}_{wn} \in \mathrm{SE}(3)$, y que el *ground-truth* para la misma trayectoria viene dado por $\mathbf{T}_{w1}, \dots \mathbf{T}_{wn} \in \mathrm{SE}(3)$, el RPE en el i-ésimo *frame* se define como:

$$\mathbf{E}_{ij} = \left(\mathbf{T}_{wi}^{-1}\mathbf{T}_{wj}\right)^{-1}\left(\hat{\mathbf{T}}_{wi}^{-1}\hat{\mathbf{T}}_{wj}\right).$$

Usualmente se escoge $j = i+1$. De esta forma, obtenemos $n-1$ errores individuales. Generalmente se computa la raíz del error cuadrático medio sobre la parte traslacional de los errores individuales:

$$\mathrm{RMSE}\left(\{\mathbf{E}_{12}, \dots, \mathbf{E}_{(n-1)n}\}\right) = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n-1}\left\|trans\left(\mathbf{E}_{i(i+1)}\right)\right\|^2},$$

donde $trans\left(\mathbf{E}_{i(i+1)}\right)$ retorna la parte traslacional de $\mathbf{E}_{i(i+1)}$.

Otra métrica que se suele utilizar es ATE (del inglés *Absolute Trajectory Error*). Permite evaluar la consistencia global de un método, y es más común en sistemas de SLAM. En lugar de centrarse en los desplazamientos relativos locales, compara las distancias absolutas entre la estimación y el *ground-truth* para cada tiempo t_i. En este caso, además de tener la misma longitud, se requiere que $\hat{\mathbf{T}}_{w1}, \dots \hat{\mathbf{T}}_{wn}$ y $\mathbf{T}_{w1}, \dots \mathbf{T}_{wn}$ estén expresados en el mismo sistema de coordenadas. Luego, el ATE en el i-ésimo *frame* se define como:

$$\mathbf{F}_i = \mathbf{T}_{wi}^{-1}\hat{\mathbf{T}}_{wi}.$$

Sobre estos n errores individuales se suele calcular la raíz del error cuadrático medio sobre la parte traslacional:

$$\mathrm{RMSE}\left(\{\mathbf{F}_1, \dots, \mathbf{F}_n\}\right) = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left\|trans\left(\mathbf{F}_i\right)\right\|^2}.$$

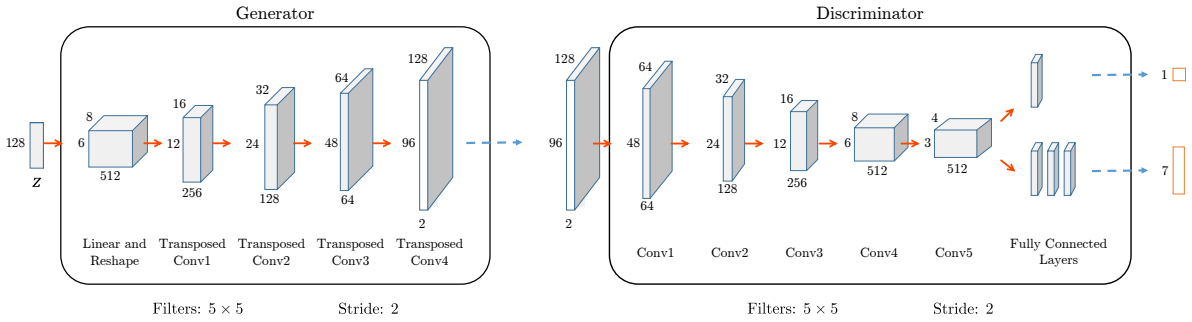

Figura 5.1: Configuración de la red utilizando un Generador y un Discriminador basados en DCGAN [14].

5.1.1. Métrica utilizada por KITTI

KITTI define su propia métrica [62] para evaluar el rendimiento de un método de VO o SLAM. La misma computa errores rotacionales y traslacionales para todas las posibles subsecuencias de una trayectoria de longitudes $\{100, 200 \ldots, 800\}$ metros. Cada uno de estos errores se promedian por la longitud correspondiente y estos a su vez se promedian por la cantidad de subsecuencias para cada una de las longitudes. Formalmente, para todas las subsecuencias de longitud $\Delta \in \{100, 200 \ldots, 800\}$ metros se calculan el error traslacional:

$$\epsilon_t \left(\mathbf{E}_{i(i+K)}\right) = \frac{\left\| trans \left(\mathbf{E}_{i(i+K)}\right)\right\|_2}{\Delta},$$

y el error rotacional:

$$\epsilon_r \left(\mathbf{E}_{i(i+K)}\right) = \frac{\arccos\left(\frac{\mathrm{Tr}\left(rot\left(\mathbf{E}_{i(i+K)}\right)\right)-1}{2}\right)}{\Delta},$$

donde $rot\left(\mathbf{E}_{i(i+K)}\right)$ retorna la parte rotacional de $\mathbf{E}_{i(i+K)}$, y K representa la cantidad de *frames* tal que la distancia entre la pose en el *frame* i y el *frame* $i + K$ es igual a Δ. Luego, se promedian ϵ_r y ϵ_t sobre todas las subsecuencias. De esta forma, a diferencia de la anterior métrica presentada, el error traslacional y el error rotacional se analizan por separado. En esta tesina el error traslacional se presenta como porcentaje y el error rotacional se presenta en grados por 100 metros (°/100m) ya que es el formato empleado por la mayoría de los trabajos que utilizan esta métrica.

5.2. Arquitecturas utilizadas

En lo que resta del capítulo se realizan distintas evaluaciones sobre WGANVO cuya estructura general se explica en la Sección 4.2. Para ello, se emplean distintas arquitecturas para el Generador y Discriminador. En esta sección se describe en detalle las distintas configuraciones utilizadas.

5.2.1. Configuración DCGAN

En esta configuración el Generador y el Discriminador están basados en DCGAN [14]. A continuación se describen las distintas capas, mientras que en la Figura 5.1 se puede apreciar información sobre las capas convolucionales y las dimensiones de los *feature maps* retornados en cada caso.

Generador

El Generador recibe como entrada un vector aleatorio de 128 componentes muestreado de una distribución normal con media 0 y desvío estándar 1. A una capa lineal de entrada le siguen cuatro capas convolucionales transpuestas con el objetivo de aumentar la dimensión de los *feature maps*. Luego de cada una de las capas convolucionales mencionadas se realiza una operación de *batch normalization* y se aplica una función de

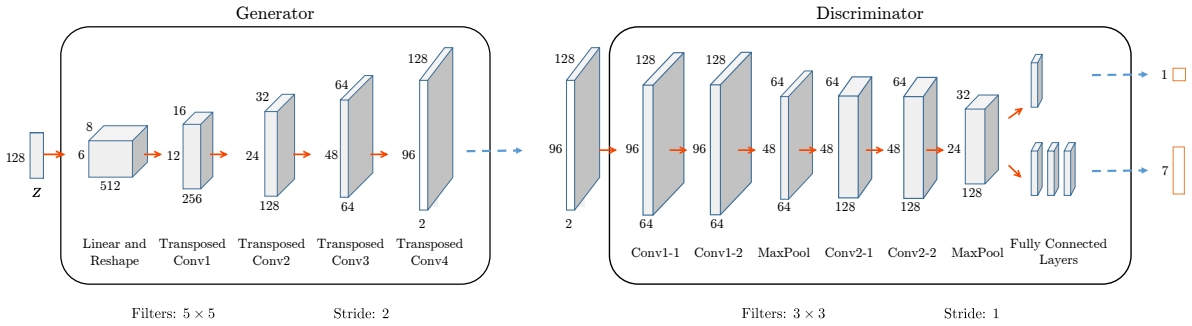

Figura 5.2: Configuración que utiliza un Discriminador basado en VGG [63].

activación ReLU, excepto por la última capa, la cual utiliza la función tanh. Por último, la salida son dos imágenes generadas con la misma resolución que las imágenes del dataset.

Discriminador

El Discriminador está compuesto inicialmente de cinco capas convolucionales, cada una seguida de una función de activación LeakyReLU. Luego, se aplana la salida de la última capa convolucional para convertirla en un vector. En este punto, la red se divide en dos bloques. Por un lado se tiene una capa *fully-connected*, cuya salida es la probabilidad de que la entrada sea un par obtenido desde el dataset de entrenamiento. Por otro lado, se tienen tres capas *fully-connected*, las cuales se encargan de estimar la transformación relativa entre los dos *frames* de entrada, representada por un vector de siete componentes. Cuatro de estos componentes corresponden a un *quaternion* y los restantes tres componentes corresponden a un vector de traslación $\mathbf{t} \in \mathbb{R}^3$. Al convertir el *quaternion* en la matriz de rotación correspondiente, junto al vector $\mathbf{t}$ se obtiene la transformación relativa estimada entre los dos *frames*. Sobre estas últimas capas se utiliza *dropout* como técnica de regularización, con un valor del 50 %. Este red se caracteriza por no utilizar *batch normalization* a diferencia del Discriminador original de DCGAN, ya que la función de costo de entrenamiento adversario penaliza el gradiente del Discriminador con respecto a cada entrada independientemente (ver Sección 3.5.1), y no sobre el *batch* en su conjunto.

5.2.2. Configuración basada en VGG

VGG-16 [63] es una de las arquitecturas más utilizadas en la actualidad en problemas de clasificación de imágenes y *Computer Vision*. En la presente configuración se utiliza el mismo Generador que en la anterior, pero el Discriminador está basado en VGG, en el sentido que utiliza bloques convolucionales compuestos por 2 o 3 capas convolucionales, intercalados por capas de *pooling* y finaliza con capas *fully-connected*. La arquitectura original de VGG-16 utiliza hasta 5 bloques convolucionales, pero por cuestiones de recursos computacionales, en el presente trabajo se utilizan solamente dos bloques. En la Figura 5.2 se describen las capas convolucionales de la mencionada configuración.

Generador

El Generador está basado en DCGAN, y tiene la misma forma que el detallado en la configuración anterior.

Discriminador

El Discriminador está compuesto por dos bloques de dos capas convolucionales, donde cada capa es seguida de una función de activación LeakyReLU. Luego de cada bloque se aplica una capa de *pooling*. Finalmente, se aplana la salida de la última capa de *pooling* para convertirla en un vector. En este punto, la red se divide en dos bloques. Por un lado se tiene una capa *fully-connected*, cuya salida es la probabilidad de que la entrada sea un par obtenido desde el dataset de entrenamiento. Por otro lado, se tienen tres capas *fully-connected*, las cuales se encargan de estimar la transformación relativa entre los dos *frames* de entrada, representada por

un vector de siete componentes. Cuatro de estos componentes corresponden a un *quaternion* y los restantes tres componentes corresponden a un vector en $\mathbf{t} \in \mathbb{R}^3$. Al convertir el *quaternion* en la matriz de rotación correspondiente, junto al vector $\mathbf{t}$ se obtiene la transformación relativa estimada entre los dos *frames*. Sobre estas últimas capas se utiliza *dropout* como técnica de regularización, con un valor del 50 %.

5.2.3. Comparación entre las configuraciones propuestas

Se realiza una comparación sobre ambas arquitecturas con el objetivo de definir si alguna de ellas presenta un mejor rendimiento sobre la otra. Para cada ejecución, se entrena durante un cierto número fijo de iteraciones minimizando la función de costo adversaria y, al finalizar, se busca minimizar la función de costo L_β definida en la Sección 4.3.1. Los datos utilizados para el entrenamiento son las secuencias $\{00, 01 \ldots, 10\}$ y las secuencias espejadas de las mismas. Se toman todas las secuencias para entrenar excepto una secuencia que se utiliza para testear. Además se quita la secuencia espejada correspondiente de los datos de entrenamiento. Por ejemplo, se entrena la red con las secuencias $\{01, 02 \ldots, 10\}$ y sus secuencias espejadas y se testea sobre la secuencia 00. Esto se repite para cada una de las secuencias del conjunto $\{00, 01 \ldots, 10\}$. En este experimento se entrena con 5000 iteraciones para la parte no supervisada y 20000 iteraciones para la regresión de la pose. En cada ejecución se utiliza un *batch* de tamaño 100 y se emplea el optimizador Adam con un *learning rate* de 10^{-4}, y la función de costo con parámetro $\beta = 100$.

Luego de realizar la comparación para cada secuencia, no se encontró una marcada diferencia en rendimiento. Sin embargo, se encontró que al quitar la quinta capa convolucional del Discriminador de la configuración DCGAN el rendimiento disminuía. Vale la pena subrayar que la arquitectura VGG-16 original posee 16 capas convolucionales. Una arquitectura de tal complejidad resulta más difícil de entrenar y evaluar. Por motivos de tiempo y falta de recursos computaciones, no se experimentó en este trabajo con redes que cuenten con un mayor número de capas convolucionales. Se adoptó la configuración DCGAN para realizar el resto de los experimentos.

5.3. Consideraciones generales sobre el entrenamiento

En las secciones posteriores, se presentan resultados sobre diversas evaluaciones realizadas sobre la CNN basada en WGAN-GP. En todos los experimentos realizados se utiliza la técnica *k-fold cross validation* debido a que no se cuenta con un gran volumen de datos. En nuestro caso, utilizamos $k = 5$, por lo tanto en cada caso el conjunto de validación utilizado corresponde al 20 % de las muestras de entrenamiento. En cada una de las iteraciones, se mide el error evaluando la red sobre dicho conjunto con el objetivo de seleccionar el modelo que mejor generaliza sobre datos no vistos. El error se mide utilizando la función de costo L_β definida en la Sección 4.3.1 con $\beta = 100$. En lo que sigue del capítulo, los valores presentados corresponden al promedio de los resultados obtenidos para los 5 modelos, mientras que las trayectorias presentadas corresponden al resultado obtenido sobre el mejor de los 5 modelos (es decir, el modelo para el cual se obtiene el mínimo valor en la función de costo).

5.4. Imágenes generadas

Al trabajar con modelos generativos siempre resulta de interés observar las imágenes generadas por el Generador. Analizar visualmente la salida de esta red sirve como un primer indicio de que la misma está aprendiendo *features* de alto nivel sobre las imágenes observadas. El primer experimento consta de entrenar la red utilizando el entrenamiento adversario entre el Generador y Discriminador y sin realizar el entrenamiento del Discriminador para la regresión de la pose. Para el entrenamiento se tomaron las secuencias $\{00, 01 \ldots, 10\}$ de KITTI y se construyó el dataset con pares de *frames* consecutivos como se explica en la Sección 4.1. El tamaño del *batch* utilizado es de 100 pares de *frames* y se entrenó durante 20000 iteraciones. Se emplea el optimizador Adam con un *learning rate* de 10^{-4}, y en la función de costo L_β el parámetro β es 100. En este experimento se utiliza la configuración basada en DCGAN descripta en la Sección 5.2.1.

Al finalizar el entrenamiento, se puede observar que la red genera imágenes que se asemejan bastante a la realidad. En la Figura 5.3 se observan tres pares de imágenes como ejemplo. Se observan características de alto

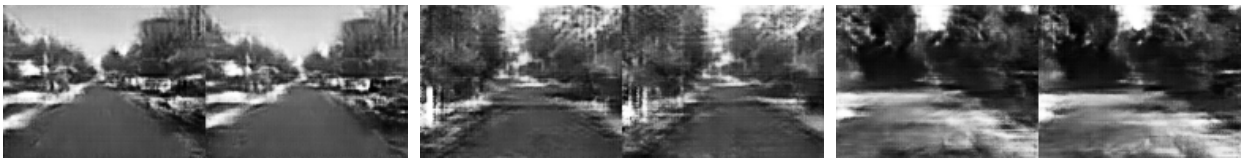

Figura 5.3: Ejemplos de pares de frames generados por el Generador de WGANVO. En cada par se observa un ligero desplazamiento que representa el movimiento de la cámara.

Figura 5.4: 100 imágenes generadas por la red generativa de WGANVO para distintos valores del vector aleatorio z que toma el Generador por entrada. Dado que se generan pares de imágenes, en este caso estamos observando las imágenes correspondientes al primer elemento del par.

nivel como el camino en el centro de la imagen y árboles a cada lado del camino. En la parte superior de las imágenes se distingue el cielo y podemos apreciar que el camino avanza hasta el horizonte. Adicionalmente, entre las imágenes de cada par se observa un ligero desplazamiento que simula ser el movimiento de la cámara. Con el objetivo de mostrar una mayor cantidad de imágenes generadas, en la Figura 5.4 se presentan 100 imágenes obtenidas a partir de ejecutar el Generador con 100 vectores distintos como entrada. Dichas imágenes corresponden al primer elemento del par generado.

Como puede observarse, la red es capaz de obtener detalles de alto nivel, generando imágenes similares a las reales. Las siguientes secciones se enfocan en el Discriminador en lugar del Generador, analizando los resultados obtenidos al abordar el problema de VO.

5.5. Entrenamiento semi-supervisado

En esta sección se analiza la influencia del entrenamiento semi-supervisado en nuestro trabajo. Con este objetivo en mente, se realizan tres tipos de entrenamientos diferentes. El primero de ellos consta en realizar el entrenamiento semi-supervisado propuesto en la Sección 4.3. El mismo implica realizar el entrenamiento adversario característico de WGAN-GP, y, al finalizar, entrenar el Discriminador para obtener la regresión

de la pose. Para resolver estas tareas, el Discriminador recibe como entrada dos fuentes de información: imágenes generadas por el Generador en primer lugar, y luego imágenes reales del dataset. Como segundo experimento, se propone entrenar una CNN exactamente igual al Discriminador, excepto que en este caso se remueve la salida de la red que distingue entre imágenes reales y generadas. Dicha red es entrenada para resolver solamente la tarea de estimar el movimiento, ignorando el Generador y, por lo tanto, sin realizar el entrenamiento adversario. En consecuencia, la entrada de la red sólo consta de imágenes reales. Finalmente, el tercer experimento consta de realizar simultáneamente, y en el mismo paso, el entrenamiento adversario y la regresión de la pose sobre el Discriminador. En todos los experimentos se utiliza la configuración basada en DCGAN descripta en la Sección 5.2.1. Los datos utilizados para el entrenamiento son las secuencias $\{00, 01 \ldots, 10\}$ y sus secuencias espejadas. Se toman todas las secuencias para entrenar excepto una secuencia que se utiliza para testear. Además se remueve la secuencia espejada correspondiente de los datos de entrenamiento. Por ejemplo, se entrena la red con las secuencias $\{01, 02, \ldots, 10\}$ y sus secuencias espejadas y se testea sobre la secuencia 00. Esto se repite para cada una de las secuencias de $\{00, 01 \ldots, 10\}$. En cuanto a la cantidad de iteraciones, para el primer caso se entrena con 10000 iteraciones para la parte no supervisada y 40000 iteraciones para la regresión de la pose, mientras que para el segundo y tercer caso se entrena con 50000 iteraciones. En cada ejecución se utiliza un *batch* de tamaño 100 y se emplea el optimizador Adam con un *learning rate* de 10^{-4}, y la función de costo L_β con parámetro $\beta = 100$.

En la Tabla 5.1 se detallan las resultados obtenidos de este experimento con una evaluación utilizando la métrica de KITTI (ver Sección 5.1.1). Como puede observarse, el entrenamiento semi-supervisado en general influye positivamente para la parte traslacional, mejorando los resultados obtenidos por la red. La secuencia 01 representa un caso particular en el que los errores son notablemente mayores a los errores estimados en otras secuencias. Dicha secuencia se caracteriza por ser la única con velocidades mayores a 60 km/h. En consecuencia, resulta desafiante para la red realizar estimaciones precisas sobre la misma. Además, la secuencia transcurre en una autopista, y los componentes de la escena se encuentra a una mayor distancia de la cámara. Entre los datos de entrenamiento no se encontraban movimientos a velocidades similares a aquellos de la secuencia 01, y es probable de que el rendimiento de la red se haya visto afectado por este motivo. Una prueba interesante es añadir pares de *frames* a las muestras de entrenamiento en los que en lugar de utilizar *frames* consecutivos, se formen pares tomando imágenes cada k *frames*. Es decir, en adición al dataset construido originalmente, se incorporarían los pares $\{(I_1, I_{k+1}), (I_2, I_{k+2}) \ldots, (I_{n-k}, I_n)\}$. Variando el valor de k se obtienen nuevos desplazamientos que podrían simular distintas velocidades para la red y eventualmente podrían mejorar los resultados sobre la secuencia 01.

5.6. Comparación de distintas funciones de costo

Como alternativa a la función de costo L_β definida en la Sección 4.3.1, se plantea utilizar una función de costo basada en el error de reproyección propuesta en [60]. En este caso, el error de reproyección viene dado por las diferencias entre proyecciones de la escena 3D sobre el plano de la imagen usando el *ground-truth* y la pose estimada. Concretamente, se define como:

$$L_{proy}\left(I_m, I_n\right) = \frac{1}{|G_n|} \sum_{\mathbf{x} \in G_n} \left\| \pi\left(\mathbf{K}\left(\mathbf{R}_{mn}\,\mathbf{x} + \mathbf{t}_{mn}\right)\right) - \pi\left(\mathbf{K}\left(\hat{\mathbf{R}}_{mn}\,\mathbf{x} + \hat{\mathbf{t}}_{mn}\right)\right)\right\|_2,$$

donde $\mathbf{R}_{mn}$ y $\mathbf{t}_{mn}$ son la parte rotacional y traslacional, respectivamente, de la transformación relativa entre el *frame* I_m y el *frame* I_n, $\hat{\mathbf{R}}_{mn}$ y $\hat{\mathbf{t}}_{mn}$ son las correspondientes estimaciones de la red, $\mathbf{K}$ es la matriz de calibración de la cámara, y G_n es un conjunto de puntos 3D observables desde la imagen I_n expresados en coordenadas de la cámara. Una de las ventajas de esta función de costo es que no requiere de un parámetro β para balancear la parte rotacional y la traslacional.

Para obtener los puntos 3D es necesario utilizar las imágenes estéreo KITTI. Las imágenes de la cámara derecha que originalmente habían sido descartadas, aquí se emplean junto a las imágenes de la cámara izquierda de forma *offline* para la obtención de los mencionados puntos 3D. Para esto, se extraen *features* en el par estéreo utilizando SIFT [48]. Posteriormente, se realiza una búsqueda de correspondencias (*matching*) entre las imágenes del par con el objetivo de encontrar puntos que correspondan al mismo punto 3D de la escena observada. Finalmente, empleando la calibración, se realiza la triangulación de puntos utilizando el

	GAN+VO		Only-VO		Simult. GAN+VO	
Sequence	t_{rel} (%)	r_{rel} (°/100m)	t_{rel} (%)	r_{rel} (°/100m)	t_{rel} (%)	r_{rel} (°/100m)
00	**10.54**	3.22	24.04	4.84	12.03	**3.03**
01	**24.94**	3.54	26.34	4.07	45.29	**3.21**
02	18.28	**2.74**	**14.50**	3.89	32.70	4.45
03	8.96	5.70	**7.68**	**3.14**	10.89	3.97
04	**14.14**	3.24	14.81	3.67	17.04	**2.54**
05	**7.01**	3.85	9.18	**2.51**	9.22	2.64
06	**7.87**	**2.19**	11.26	2.97	30.16	6.68
07	7.71	3.79	**6.52**	**3.74**	18.10	4.78
08	**9.04**	3.85	9.94	**3.59**	15.14	3.81
09	**10.49**	4.91	13.65	**2.85**	36.46	4.83
10	**10.89**	**5.03**	12.91	5.54	17.44	6.64

Tabla 5.1: Comparación entre los diferentes tipos de entrenamiento. GAN+VO representa el entrenamiento manteniendo por separado la minimización de la parte adversaria de la parte de VO. Only-VO representa el entrenamiento de una red idéntica al Discriminador, ignorando el entrenamiento adversario. Simult. GAN+VO representa el entrenamiento simultáneo entre la parte adversaria y la parte de VO. Se utiliza la métrica de KITTI, y t_{rel} y r_{rel} representan error traslacional y rotacional promediado sobre intervalos de 100 a 800 metros. En **negrita** se muestran los valores que representan el menor error.

algoritmo DLT (acrónimo de *Direct Linear Transformation*) descripto en [9]. Para realizar todo el proceso se utilizó la librería OpenCV [28].

En este experimento se realizan dos tipos de ejecuciones de la red empleando la configuración basada en DCGAN. Por un lado, se utiliza la función de costo L_β con parámetro $\beta = 100$, y por el otro se utiliza L_{proy}. Se realizan ejecuciones entrenando de la misma forma que la llevada a cabo en el experimento de la Sección 5.5, testeando sobre cada una de las secuencias de $\{00, 01 \ldots, 10\}$. En todos los casos se entrena durante 10000 iteraciones para la parte no-supervisada y 40000 iteraciones para la parte supervisada. Además, se utiliza un *batch* de tamaño 100 y se emplea el optimizador Adam con un *learning rate* de 10^{-4}.

En la Tabla 5.2 se exponen los resultados obtenidos. Como puede verse a simple vista, no se reportan grandes diferencias a favor de una u otra función de costo. En favor de L_{proy}, no requiere la validación de un hiperparámetro, mientras que L_β requiere validar β. Adicionalmente, en nuestro trabajo encontramos que los mejores resultados obtenidos con L_β emplean un valor $\beta = 100$, y lo aplicamos en todos los experimentos, pero de acuerdo a los autores este valor es dependiente de la escena observada durante el entrenamiento [60].

5.7. Comparación con ORB-SLAM2

ORB-SLAM2 [32] es un sistema de SLAM considerado como uno de los métodos más representativos del estado del arte, por lo que resulta de interés una comparación contra un método de este tipo. Para el presente experimento se modificó el código original de ORB-SLAM2 para desactivar el *loop closure* y la optimización global que realiza el método. De esta forma, se asemeja más a un algoritmo de VO. Luego, se ejecutó el método en sus dos versiones, monocular y estéreo, sobre las secuencias $\{00, 01 \ldots, 10\}$ de KITTI en su resolución original.

En la Tabla 5.3 se compara nuestro método contra los resultados obtenidos de las ejecuciones de ORB-SLAM2. Como se mencionó en la Sección 3.2, los métodos monoculares basados en *features* sufren del problema de no poder estimar la escala real de una trayectoria. Por lo tanto, para el caso monocular la trayectoria fue alineada y escalada con el *ground-truth* mediante el método Sim(3) Umeyama [64]. Esto implica encontrar un conjunto de parámetros (rotación, traslación y escala) con el objetivo de intentar recuperar la escala real de la trayectoria estimada. Puede observarse que si bien el método presenta un gran rendimiento sobre KITTI en su versión estéreo, el mismo disminuye para la versión monocular. Para las secuencias 01 y 09, ORB-SLAM2 monocular presenta problemas durante la estimación de la pose y la localización falla, por lo

	L_β		L_{proy}	
Sequence	t_{rel} (%)	r_{rel} (°/100m)	t_{rel} (%)	r_{rel} (°/100m)
00	**10.54**	**3.22**	11.01	4.41
01	24.94	3.54	**23.31**	**3.27**
02	18.28	**2.74**	**16.11**	2.88
03	**8.96**	5.70	9.68	**3.41**
04	**14.14**	**3.24**	14.91	3.71
05	**7.01**	**3.85**	7.18	3.91
06	7.87	**2.19**	**7.56**	2.37
07	**7.71**	3.79	7.82	**3.74**
08	**9.04**	3.85	9.41	**3.19**
09	**10.49**	4.91	10.5	**3.85**
10	**10.89**	**5.03**	10.97	5.41

Tabla 5.2: Comparación entre las dos funciones de costo L_β y L_{proy}. Se utiliza la métrica de KITTI, y t_{rel} y r_{rel} representan error traslacional y rotacional promediado sobre intervalos de 100 a 800 metros. En **negrita** se muestran los valores que representan el menor error.

que no reportamos el error. Como se mencionó anteriormente, la secuencia 01 suele ser desafiante ya que transcurre en una autopista, a una velocidad mayor a la media del dataset y los puntos se encuentran a gran distancia de la cámara. La Figura 5.5 muestra una comparación de las trayectorias obtenidas en este experimento sobre las secuencias $\{00, 01 \ldots, 10\}$ de KITTI. Adicionalmente, en la Tabla 5.4 se presenta el tiempo promedio que tarda cada sistema en procesar un *frame*. Los tres métodos se ejecutaron en una computadora con procesador Intel Core i7-3770 con 12 GB de memoria RAM y una GPU GeForce GTX 770 con 2 GB de memoria.

En conclusión, WGANVO presenta un rendimiento menor que ORB-SLAM2 estéreo. Sin embargo, este es un resultado esperable, ya que los métodos basados en *features* son el fruto de décadas de investigación y, en consecuencia, han alcanzado una madurez que se ve reflejada en su precisión y robustez. Por su parte, las técnicas de *Deep Learning* recién se han empezado a aplicar en los últimos años para resolver el problema de VO. No obstante, la precisión de WGANVO es comparable a la de ORB-SLAM2 monocular para algunas de las secuencias. Más aún, WGANVO no sufre el problema de desconocer la escala real de la escena, un problema general que presentan los métodos monoculares basados en *features*, ya que puede inferir la misma a partir del entrenamiento. En cuanto a los tiempos de ejecución, WGANVO puede procesar en promedio a un *frame rate* de 50 fps. Para poner en contexto, el dataset KITTI fue grabado a 10 fps, por lo que el método puede procesar imágenes obtenidas desde una cámara semejante sin ningún tipo de inconvenientes.

5.8. Discusión

Para finalizar este capítulo vale la pena realizar algunos comentarios generales acerca de otros trabajos que utilizan técnicas de *Deep Learning* para resolver el problema de VO. Existen diferentes trabajos que fueron publicados durante el desarrollo de este trabajo, los cuales utilizan información de profundidad de la escena [65, 22, 43]. La profundidad, cuando no está disponible como *ground-truth*, se suele estimar con una CNN. Las relaciones geométricas entre la profundidad, la calibración de la cámara y la transformación relativa entre dos *frames* permite sintetizar el *frame* siguiente. En consecuencia, se obtiene una señal supervisora de entrenamiento: se comparan el *frame* sintético y el *frame* original mediante alguna métrica adecuada [24, 23]. Este es un resultado muy importante, ya que permite entrenar una CNN en forma no supervisada para estimar la transformación relativa entre dos *frames* y la profundidad de la escena. Es importante destacar que la obtención de datos etiquetados para resolver este problema es muy costosa, por lo que estos trabajos representan un gran avance.

	WGANVO (Ours)		ORB-SLAM2 (Mono)		ORB-SLAM2 (Stereo)	
Sequence	t_{rel} (%)	r_{rel} (°/100m)	t_{rel} (%)	r_{rel} (°/100m)	t_{rel} (%)	r_{rel} (°/100m)
00	10.54	3.22	23.01	**0.30**	**0.88**	**0.30**
01	24.94	3.54	-	-	**1.39**	**0.23**
02	18.28	2.74	6.63	**0.24**	**0.81**	0.28
03	8.96	5.70	1.12	0.19	**0.73**	**0.17**
04	14.14	3.24	0.70	0.22	**0.48**	**0.15**
05	7.01	3.85	12.34	**0.22**	**0.61**	0.25
06	7.87	2.19	17.71	0.27	**0.76**	**0.23**
07	7.71	3.79	11.10	**0.36**	**0.88**	0.47
08	9.04	3.85	12.69	**0.30**	**1.05**	0.32
09	10.49	4.91	-	-	**0.83**	**0.26**
10	10.89	5.03	3.90	0.30	**0.57**	**0.26**

Tabla 5.3: Comparación entre el método propuesto en esta tesina y ORB-SLAM2 monocular y estéreo, sin *loop closure* ni optimización global. Se utiliza la métrica de KITTI, y t_{rel} y r_{rel} representan error traslacional y rotacional promediado sobre intervalos de 100 a 800 metros. En **negrita** se muestran los valores que representan el menor error.

	WGANVO (Ours)	ORB-SLAM2 (Mono)	ORB-SLAM2 (Stereo)
Sequence	time (ms)	time (ms)	time (ms)
00	**20.70**	36.33	111.24
01	**21.12**	-	146.26
02	**20.32**	40.81	119.23
03	**19.98**	40.77	110.18
04	**20.57**	40.18	114.84
05	**20.59**	39.31	116.27
06	**20.11**	39.99	113.88
07	**20.41**	41.70	111.49
08	**20.72**	37.13	113.32
09	**20.44**	-	109.46
10	**19.88**	40.37	104.89

Tabla 5.4: Tiempo promedio (en milisegundos) que demora cada método en procesar un *frame*. En **negrita** se muestran los valores que representan el mejor tiempo.

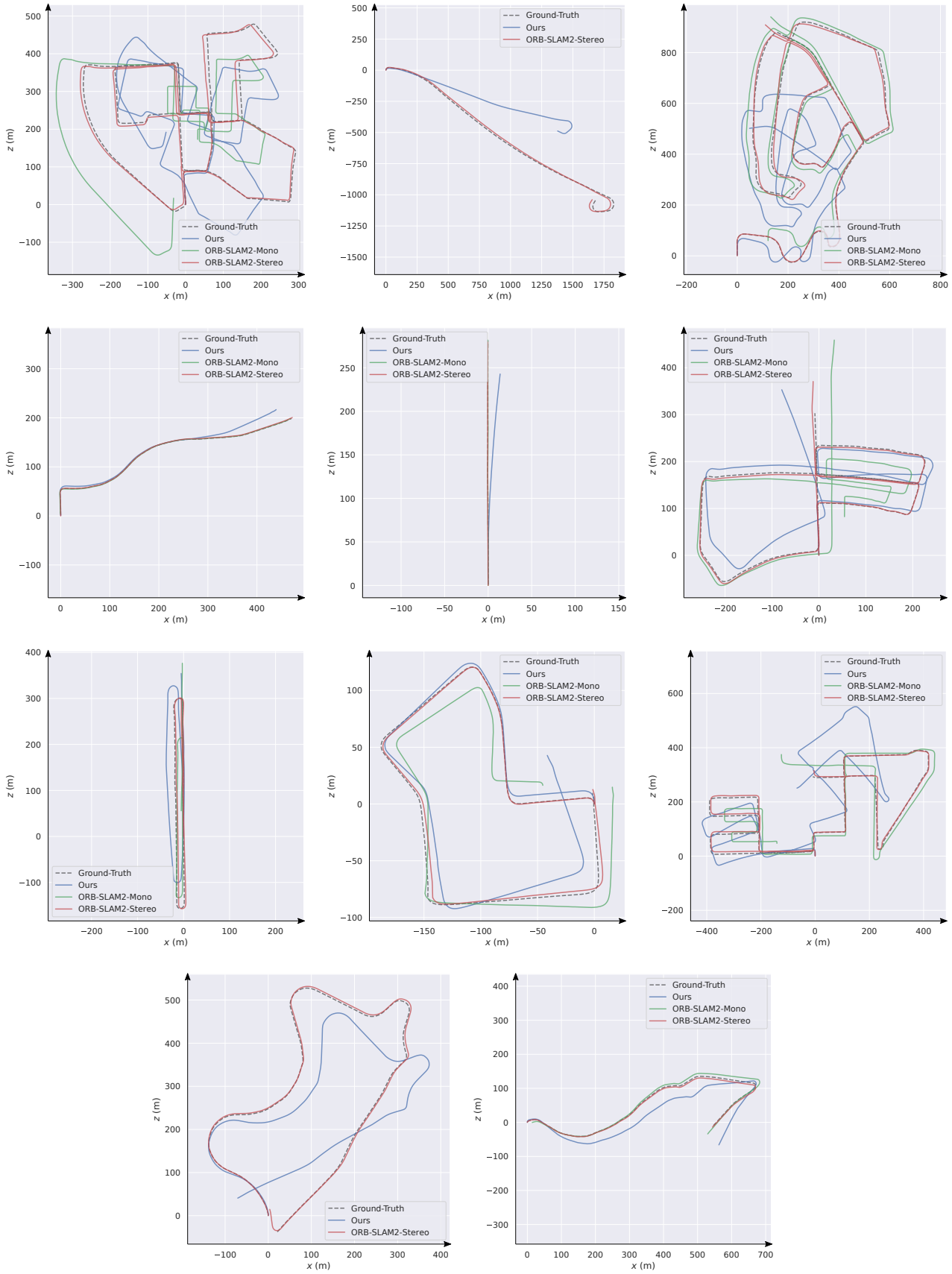

Figura 5.5: Comparación de nuestro método con ORB-SLAM2 monocular y estéreo (sin *loop closure* ni optimización global) sobre las secuencias $\{00, 01 \ldots, 10\}$.

	WGANVO(Ours)		DeepVO		UnDeepVO	
Sequence	t_{rel} (%)	r_{rel} (°/100m)	t_{rel} (%)	r_{rel} (°/100m)	t_{rel} (%)	r_{rel} (°/100m)
00	10.54	3.22	-	-	**4.14**	**1.92**
01	**24.94**	**3.54**	-	-	-	-
02	18.28	2.74	-	-	**5.58**	**2.44**
03	8.96	**5.70**	**8.49**	6.89	-	-
04	14.14	**3.24**	**7.19**	6.97	-	-
05	7.01	3.85	**2.62**	3.61	3.40	**1.50**
06	7.87	**2.19**	**5.42**	5.82	-	-
07	7.71	3.79	3.91	4.60	**3.15**	**2.48**
08	9.04	3.85	-	-	**4.08**	**1.79**
09	**10.49**	**4.91**	-	-	-	-
10	10.89	**5.03**	**8.11**	8.83	-	-

Tabla 5.5: Comparación entre nuestro método y métodos similares basados en *Deep Learning*. Los resultados de DeepVO fueron reportados en [16], mientras que los resultados de UnDeepVO fueron reportados en [22]. Se utiliza la métrica de KITTI, y t_{rel} y r_{rel} representan error traslacional y rotacional promediado sobre intervalos de 100 a 800 metros. En **negrita** se muestran los valores que representan el menor error. DeepVO se entrenó sobre las secuencias 00, 02, 08 y 09, mientras que UnDeepVO se entrenó sobre las secuencias $\{00, 01, \ldots 08\}$.[1]

Otro enfoque interesante aplicado en los últimos años implica la implementación de redes neuronales recurrentes (en inglés, *Recurrent Neural Networks*). Este tipo de redes han demostrado ser eficientes para resolver tareas que implican entradas secuenciales como el procesamiento del habla y el lenguaje. Esto se debe a que las redes recurrentes almacenan un estado mediante el cual pueden acceder a información de elementos anteriores de la secuencia. El movimiento de una cámara puede considerarse como una tarea de este tipo. Una decena de *frames* consecutivos están claramente correlacionados entre ellos en cuanto al movimiento, en especial en datasets en los que no hay giros bruscos como KITTI. Consecuentemente, redes neuronales recurrentes han sido aplicadas para resolver el problema de VO [16, 43].

En la Tabla 5.5 se presentan resultados sobre distintos métodos que utilizan *Deep Learning* entrenados de forma *end-to-end* para resolver el problema de VO. DeepVO implementa redes recurrentes sobre *features* obtenidos por una CNN, mientras que UnDeepVO consiste en dos CNNs, una para la estimación de la profundidad y una para la predicción de la transformación relativa. Por su parte, debemos mencionar al método denominado GANVO [43], quien combina ambas técnicas utilizando GAN: el Generador se encarga de estimar un mapa de profundidad, el cual, junto a una red recurrente que estima la tranformación relativa entre dos *frames,* sirve para sintetizar un *frame*. El Discriminador se encarga de distinguir entre *frames* sintéticos y reales.

Resulta evidente que los métodos que aplican las técnicas mencionadas en esta sección aventajan al método propuesto en esta tesina. Los métodos similares a UnDeepVO realizan una estimación de la profundidad de la escena, resultando en un algoritmo más complejo que el nuestro. Por su parte, los métodos basados en redes recurrentes capturan información sobre secuencias de *frames,* a diferencia de nuestra propuesta, la cual analiza pares de *frames* consecutivos independientemente del resto de los *frames* contiguos. No obstante, los mencionados trabajos aún se encuentran por debajo de los métodos del estado del arte en cuanto a rendimiento y robustez. Para empezar, dichos sistemas no pueden ser aplicados sobre cualquier escena: una red entrenada sobre imágenes de una escena urbana probablemente no presente un buen rendimiento sobre ambientes interiores. Del mismo modo, existen datasets mas desafiantes que KITTI sobre los cuales es probable que no se obtengan buenos resultados. Los mismos implican rotaciones y giros bruscos. Un dataset más representativo de la realidad es Oxford RobotCar [66], el cual presenta distintas condiciones climáticas, escenas diurnas

[1] En los mencionados trabajos se refieren al error como RMSE promedio de los errores rotacional y traslacional, pero creemos que esto es un error, ya que constatamos que el código provisto por KITTI para la medición de dichos errores no calcula un RMSE.

y nocturnas, y una mayor cantidad de objetos dinámicos en la escena, como peatones, bicicletas y autos. Desafortunadamente, dicho dataset no cuenta con las poses correspondientes al *ground-truth*. Finalmente, vale subrayar que, debido a que los mencionados métodos entrenan y testean con el mismo dataset, posiblemente se estén sobreajustando a características específicas de KITTI como su velocidad media o la calibración de la cámara. Acerca de este último punto, debemos mencionar el reciente trabajo, conocido como CAM-Convs [67], en el que los autores entrenan una CNN para estimar mapas de profundidad de forma tal que el modelo sea independiente de la cámara. Para esto, se incluyen los parámetros internos de la cámara en las convoluciones permitiendo que la red aprenda la relación entre estos parámetros y la profundidad de la escena.

Capítulo 6

Conclusiones

En esta tesina se presentó WGANVO, un método de *Visual Odometry* (VO) basado en CNNs (*Convolutional Neural Networks*) que permite estimar el movimiento de un robot. En particular se empleó un tipo particular de CNN cuya arquitectura se basa en GAN (*Generative Adversarial Networks*). Más concretamente, se utilizó una variante de GAN conocida como WGAN-GP [26], ya que resulta más estable durante el entrenamiento que la formulación original de GAN. Mediante dicha metodología se entrenan dos redes simultáneamente en un esquema competitivo. Una red, conocida como Generador, se encarga de generar imágenes similares a las utilizadas en el entrenamiento. Por otro lado, la otra red, conocida como Discriminador, se encarga de distinguir entre imágenes reales e imágenes producidas por el Generador. De esta forma, el Generador aprende a sintetizar imágenes cada vez más similares a las del dataset a la vez que el Discriminador se especializa en la tarea de diferenciar imágenes reales de imágenes generadas. Consecuentemente, el Discriminador captura detalles y patrones de alto nivel de las imágenes que recibe como entrada. Así mismo, se modificó la entrada del Discriminador, de forma tal que acepte un par de *frames* y se extendió su salida con el objetivo de retornar directamente el movimiento a partir de la entrada. De esta manera se obtuvo un modelo capaz de estimar la transformación relativa entre dos *frames*.

El entrenamiento de WGANVO consta en realizar el entrenamiento adversario minimizando la función de costo original de WGAN-GP y posteriormente, minimizar la función de costo de VO de forma supervisada utilizando las poses de cada *frame* (*ground-truth*). En consecuencia, el Discriminador aprende a distinguir características visuales de alto nivel a partir de datos etiquetados y no etiquetados. Esto en la literatura es conocido como entrenamiento semi-supervisado.

Se utilizó el dataset KITTI [29] para entrenar y testear la red. Con el objetivo de contar con más *frames* para el entrenamiento, se espejaron las imágenes del dataset para las que se contaba con el *ground-truth* correspondiente. Esto permitió duplicar el número de imágenes disponibles para entrenar. Además se realizó un pre-procesamiento que consistió en disminuir la resolución de las imágenes para ahorrar recursos computacionales.

Para la evaluación se decidió correr el método basado en *features* del estado del arte conocido como ORB-SLAM2 [32], en sus versiones monocular y estéreo, y contrastar los resultados con nuestro sistema. Paralelamente, se analizó la influencia del entrenamiento semi-supervisado, comparando modelos que fueron entrenados de forma adversaria y modelos que no emplean el entrenamiento característico de GAN. Para finalizar, se realizó una revisión de aquellos métodos que emplean CNNs entrenadas de forma *end-to-end* para resolver el problema de VO, concluyendo que actualmente los mejores resultados se obtienen mediante la utilización de redes neuronales recurrentes por un lado y, por el otro, mediante el uso de información geométrica adicional del entorno como lo son los mapas de profundidad.

En conclusión, se observaron mejoras en precisión cuando se aplica entrenamiento semi-supervisado. No obstante, nuestra propuesta no alcanza el rendimiento presentado por un método del estado del arte como ORB-SLAM2 estéreo. Métodos como ORB-SLAM2 se encuentran muy maduros ya que son el resultado de décadas de investigación. Por otra parte, las técnicas de *Deep Learning* se han comenzado a aplicar recientemente para resolver el problema de VO. Sin embargo, para varias secuencias, nuestro método tiene una performance similar a la de ORB-SLAM2 monocular (alineado y escalado). Más aún, nuestro método

no presenta el problema de desconocer la escala real de la escena, un problema general que presentan los sistemas monoculares basados en *features*.

6.1. Trabajos futuros

Como trabajo futuro derivado de la presente tesina, se identifican varias mejoras que pueden ser investigadas. En primer lugar la aplicación de redes recurrentes en trabajos recientemente presentados ha significado un importante avance. Este tipo de redes han demostrado ser eficientes para resolver tareas que implican entradas secuenciales ya que almacenan un estado que les permite acceder a elementos anteriores de la secuencia. El movimiento de un robot puede considerarse un problema de este tipo. Durante el desarrollo de esta tesina han surgido diferentes trabajos [16, 43] que sacan provecho de la naturaleza secuencial del problema, por lo que en futuras líneas de investigación las redes recurrentes deben ser seriamente consideradas a la hora de encarar el problema de VO. Un posible trabajo a futuro sería combinar la capacidad de extracción de patrones y características visuales de WGANVO con redes recurrentes que se encarguen de estimar el movimiento de la cámara.

Otra mejora más concreta es la utilización de los parámetros de calibración de la cámara durante la etapa de entrenamiento con el fin de generar un modelo independiente de la cámara utilizada para capturar las imágenes de entrenamiento. De esta forma es posible entrenar y testear el algoritmo sobre diferentes datasets, e incluso entrenar con datasets con diferentes cámaras sin que el rendimiento se vea afectado. CAM-Convs [67] marcó un precedente en este aspecto, ya que estiman la profundidad de una escena generando un modelo independiente de la cámara, al concatenar los parámetros de calibración con los *features maps* convolucionales.

Para finalizar, una mejora interesante es la de trabajar con cámaras estéreo. Incorporar más información de la escena puede resultar útil a la hora de estimar el movimiento de la cámara. En ese sentido, se puede sacar provecho de las relaciones o restricciones geométricas que existen entre la profundidad de la escena, la calibración de la cámara y el movimiento de la misma para tener una mejor estimación de dicho movimiento.

Bibliografía

[1] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press, 2005.

[2] Bruno Siciliano and Oussama Khatib. *Springer Handbook of Robotics*. Springer Publishing Company, Incorporated, 2nd edition, 2016.

[3] David Nistér, Oleg Naroditsky, and James Bergen. Visual odometry. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 1, pages 652–659. IEEE, June 2004.

[4] Davide Scaramuzza and Friedrich Fraundorfer. Visual Odometry [Tutorial]. *Proceedings of the IEEE Robotics Automation Magazine*, 18(4):80–92, Dec 2011.

[5] Andrew I Comport, Ezio Malis, and Patrick Rives. Real-time quadrifocal visual odometry. *The International Journal of Robotics Research*, 29(2-3):245–266, 2010.

[6] Nicola Krombach, David Droeschel, and Sven Behnke. Combining Feature-based and Direct Methods for Semi-dense Real-time Stereo Visual Odometry. In *Proceedings of the 14th International Conference on Intelligent Autonomous Systems (IAS)*, July 2016.

[7] Jakob Engel, Vladlen Koltun, and Daniel Cremers. Direct Sparse Odometry. *Proceedings of the IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(3):611–625, March 2018.

[8] Christian Forster, Matia Pizzoli, and Davide Scaramuzza. SVO: Fast semi-direct monocular visual odometry. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 15–22, May 2014.

[9] Richard Hartley and Andrew Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, New York, NY, USA, 2 edition, 2003.

[10] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436, 2015.

[11] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved Techniques for Training GANs. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, pages 2234–2242, USA, 2016. Curran Associates Inc.

[12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[13] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[14] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks. *CoRR*, abs/1511.06434, 2015.

[15] Augustus Odena. Semi-supervised learning with generative adversarial networks. *ArXiv e-prints*, June 2016.

[16] Sen Wang, Ronald Clark, Hongkai Wen, and Niki Trigoni. DeepVO: Towards end-to-end visual odometry with deep Recurrent Convolutional Neural Networks. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 2043–2050, May 2017.

[17] Vikram Mohanty, Shubh Agrawal, Shaswat Datta, Arna Ghosh, Vishnu Dutt Sharma, and Debashish Chakravarty. DeepVO: A Deep Learning approach for Monocular Visual Odometry. *CoRR*, abs/1611.06069, 2016.

[18] Keisuke Tateno, Federico Tombari, Iro Laina, and Nassib Navab. CNN-SLAM: Real-Time Dense Monocular SLAM with Learned Depth Prediction. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6565–6574, July 2017.

[19] Pulkit Agrawal, Joao Carreira, and Jitendra Malik. Learning to See by Moving. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 37–45, Dec 2015.

[20] Raúl Mur-Artal, José María Martínez Montiel, and Juan D. Tardós. ORB-SLAM: A Versatile and Accurate Monocular SLAM System. *Proceedings of the IEEE Transactions on Robotics*, 31(5):1147–1163, Oct 2015.

[21] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going Deeper with Convolutions. In *Proceedings of the Computer Vision and Pattern Recognition (CVPR)*, pages 1–9, June 2015.

[22] Ruihao Li, Sen Wang, Zhiqiang Long, and Dongbing Gu. UnDeepVO: Monocular Visual Odometry through Unsupervised Deep Learning. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, International Conference on Robotics and Automation, pages 7286–7291, United States, September 2018. IEEE.

[23] Ravi Garg, Vijay Kumar B.G., Gustavo Carneiro, and Ian Reid. Unsupervised CNN for Single View Depth Estimation: Geometry to the Rescue. In Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling, editors, *Proceedings of the European Conference on Computer Vision – ECCV*, pages 740–756, Cham, 2016. Springer International Publishing.

[24] Tinghui Zhou, Matthew Brown, Noah Snavely, and David G. Lowe. Unsupervised learning of depth and ego-motion from video. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6612–6619, July 2017.

[25] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 214–223, International Convention Centre, Sydney, Australia, 06–11 Aug 2017. PMLR.

[26] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. Improved Training of Wasserstein GANs. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 5767–5777. Curran Associates, Inc., 2017.

[27] Morgan Quigley, Ken Conley, Brian P. Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y. Ng. ROS: an open-source Robot Operating System. In *ICRA Workshop on Open Source Software*, 2009.

[28] Gary Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[29] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision Meets Robotics: The KITTI Dataset. *The International Journal of Robotics Research, IJRR*, 32(11):1231–1237, September 2013.

[30] Andreas Geiger, Julius Ziegler, and Christoph Stiller. StereoScan: Dense 3D Reconstruction in Real-time. In *Proceedings of the IEEE Intelligent Vehicles Symposium (IV)*, pages 963–968, June 2011.

[31] Andreas Geiger, Martin Roser, and Raquel Urtasun. Efficient Large-Scale Stereo Matching. In *Proceedings of the Asian Conference on Computer Vision (ACCV)*, pages 25–38, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[32] Raúl Mur-Artal and Juan D. Tardós. ORB-SLAM2: an open-source SLAM system for monocular, stereo and RGB-D cameras. *Proceedings of the IEEE Transactions on Robotics*, 33(5):1255–1262, 2017.

[33] Hugh Durrant-Whyte and Tim Bailey. Simultaneous localization and mapping: part I. *IEEE Robotics Automation Magazine*, 13(2):99–110, June 2006.

[34] Tim Bailey and Hugh Durrant-Whyte. Simultaneous localization and mapping (SLAM): part II. *IEEE Robotics Automation Magazine*, 13(3):108–117, September 2006.

[35] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. ORB: An efficient alternative to SIFT or SURF. In *Proceedings of the International Conference on Computer Vision*, pages 2564–2571, Nov 2011.

[36] Georg Klein and David Murray. Parallel Tracking and Mapping for Small AR Workspaces. In *Proceedings of the IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, pages 225–234, November 2007.

[37] Taihú Pire, Thomas Fischer, Javier Civera, Pablo De Cristóforis, and Julio Jacobo Berlles. Stereo parallel tracking and mapping for robot localization. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1373–1378, September 2015.

[38] Jakob Engel, Thomas Schöps, and Daniel Cremers. LSD-SLAM: Large-Scale Direct Monocular SLAM. In David Fleet, Tomas Pajdla, Bernt Schiele, and Tinne Tuytelaars, editors, *Proceedings of the European Conference on Computer Vision – ECCV*, pages 834–849, Cham, 2014. Springer International Publishing.

[39] Richard Newcombe, Steven Lovegrove, and Andrew Davison. DTAM: Dense Tracking and Mapping in Real-time. In *Proceedings of the 2011 International Conference on Computer Vision*, ICCV '11, pages 2320–2327, Washington, DC, USA, 2011. IEEE Computer Society.

[40] Michael Blösch, Stephan Weiss, Davide Scaramuzza, and Roland Siegwart. Vision based MAV navigation in unknown and unstructured environments. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 21–28, May 2010.

[41] Jakob Engel, Jürgen Sturm, and Daniel Cremers. Accurate figure flying with a quadrocopter using onboard visual and inertial sensing. In *Proceedings of the Workshop on Visual Control of Mobile Robots (ViCoMoR) at the IEEE/RJS International Conference on Intelligent Robot Systems (IROS)*, Oct. 2012.

[42] Nan Yang, Rui Wang, Jörg Stückler, and Daniel Cremers. Deep virtual stereo odometry: Leveraging deep depth prediction for monocular direct sparse odometry. In *Proceedings of the European Conference on Computer Vision (ECCV)*, September 2018.

[43] Yasin Almalioglu, Muhamad Risqi U. Saputra, Pedro P. B. de Gusmao, Andrew Markham, and Niki Trigoni. GANVO: unsupervised deep monocular visual odometry and depth estimation with generative adversarial networks. *CoRR*, abs/1809.05786, 2018.

[44] Zhichao Yin and Jianping Shi. GeoNet: Unsupervised Learning of Dense Depth, Optical Flow and Camera Pose. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1983–1992, June 2018.

[45] Alex Kendall, Matthew Grimes, and Roberto Cipolla. PoseNet: A Convolutional Network for Real-Time 6-DOF Camera Relocalization. In *Proceedings of the International Conference on Computer Vision (ICCV)*, pages 2938–2946, Dec 2015.

[46] Thomas M. Mitchell. *Machine Learning*. McGraw-Hill, Inc., New York, NY, USA, 1 edition, 1997.

[47] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[48] David G. Lowe. Object recognition from local scale-invariant features. In *Proceedings of the Seventh IEEE International Conference on Computer Vision*, volume 2, pages 1150–1157 vol.2, Sept 1999.

[49] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. SURF: Speeded Up Robust Features. In Aleš Leonardis, Horst Bischof, and Axel Pinz, editors, *Proceedings of the European Conference on Computer Vision – ECCV*, pages 404–417, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[50] Stefan Leutenegger, Margarita Chli, and Roland Y. Siegwart. BRISK: Binary Robust invariant scalable keypoints. In *Proceedings of the International Conference on Computer Vision*, pages 2548–2555, Nov 2011.

[51] Carlo Tomasi and Takeo Kanade. Detection and tracking of point features. Technical report, International Journal of Computer Vision, 1991.

[52] Jianbo Shi and Carlo Tomasi. Good features to track. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 593–600, Jun 1994.

[53] Yann LeCun, Bernhard Boser, John S. Denker, Donnie Henderson, R. E. Howard, Wayne Hubbard, and Lawrence D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Comput.*, 1(4):541–551, December 1989.

[54] Sergey Ioffe and Christian Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *Proceedings of the 32Nd International Conference on International Conference on Machine Learning*, ICML'15, pages 448–456. JMLR.org, 2015.

[55] Vinod Nair and Geoffrey E. Hinton. Rectified Linear Units Improve Restricted Boltzmann Machines. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, pages 807–814, USA, 2010. Omnipress.

[56] Andrew L. Maas, Awni Y. Hannun, and Andrew Y. Ng. Rectifier nonlinearities improve neural network acoustic models. In *Proceedings of the ICML Workshop on Deep Learning for Audio, Speech and Language Processing*, 2013.

[57] Fisher Yu, Yinda Zhang, Shuran Song, Ari Seff, and Jianxiong Xiao. LSUN: Construction of a Large-scale Image Dataset using Deep Learning with Humans in the Loop. *arXiv preprint arXiv:1506.03365*, 2015.

[58] Martín Arjovsky and Léon Bottou. Towards principled methods for training generative adversarial networks. *CoRR*, abs/1701.04862, 2017.

[59] Cédric Villani. *Optimal Transport: Old and New*. Grundlehren der mathematischen Wissenschaften. Springer, 2009 edition, September 2008.

[60] Alex Kendall and Roberto Cipolla. Geometric loss functions for camera pose regression with deep learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6555–6564, July 2017.

[61] Jürgen Sturm, Nikolas Engelhard, Felix Endres, Wolfram Burgard, and Daniel Cremers. A Benchmark for the Evaluation of RGB-D SLAM Systems. In *Proceedings of the International Conference on Intelligent Robot Systems (IROS)*, pages 573–580, October 2012.

[62] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite. In *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3354–3361, June 2012.

[63] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[64] Shinji Umeyama. Least-squares estimation of transformation parameters between two point patterns. *Proceedings of the IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(4):376–380, April 1991.

[65] Reza Mahjourian, Martin Wicke, and Anelia Angelova. Unsupervised Learning of Depth and Ego-Motion from Monocular Video Using 3D Geometric Constraints. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5667–5675, June 2018.

[66] Will Maddern, Geoffrey Pascoe, Chris Linegar, and Paul Newman. 1 year, 1000 km: The Oxford RobotCar dataset. *Proceedings of the The International Journal of Robotics Research*, 36(1):3–15, 2017.

[67] Jose M. Facil, Benjamin Ummenhofer, Huizhong Zhou, Luis Montesano, Thomas Brox, and Javier Civera. CAM-Convs: Camera-Aware Multi-Scale Convolutions for Single-View Depth. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11818–11827, June 2019.