Mapeo de objetos online sobre sistemas SLAM basados en visión estéreo

22 de marzo de 2018

Javier Martin Corti
javiermcorti@gmail.com

Director

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

Co-Director

Grinblat, Guillermo Luis
grinblat@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

Mapeo de objetos *online* sobre sistemas SLAM basados en visión estéreo

Para poder navegar de manera segura en un entorno desconocido, un robot autónomo móvil debe poder construir una representación del ambiente en el cual se encuentra (mapeo), al mismo tiempo que estima su posición (localización). Este problema es conocido en la robótica móvil como SLAM por el acrónimo en inglés de *Simultaneous Localization and Mapping*. Asimismo, robots que pretenden interactuar con su entorno de manera "inteligente" además de navegarlo, necesitan información semántica de su ambiente que complemente la información geométrica.

En este trabajo se propone un sistema de SLAM basado en visión estéreo que realiza una reconstrucción de objetos de su entorno (mapa). El sistema resulta de la inclusión de un módulo de detección de objetos basado en *Deep Learning* con imágenes, al sistema de SLAM S-PTAM (*Stereo Parallel Tracking and Mapping*) del estado del arte. El módulo detección de objetos se encarga de detectar y estimar la pose de los objetos en el espacio de manera *online*, mientras que S-PTAM se encarga de estimar de manera precisa la pose de la cámara en tiempo real. El sistema se evaluó en un entorno real, logrando buenos resultados de localización de objetos.

El sistema se implementó utilizando el framework ROS (*Robot Operating* System) y la librería de *Deep Learning Caffe*. El código ha sido liberado bajo licencia GPLv3, con el objetivo de facilitar el uso y modificación del sistema por parte de la comunidad robótica.

Índice general

Capítulo 1

Introducción

La robótica móvil tiene un fuerte impacto en ámbitos tales como la industria, la agricultura, y en diversas aplicaciones que van desde misiones de exploración planetaria hasta la realización de actividades domésticas o de servicio. Uno de los desafíos actuales de la robótica móvil es lograr que el robot pueda llevar a cabo sus tareas sin la necesidad de un operador humano, es decir, con completa autonomía. 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. Dicha tarea presenta una mayor complejidad cuando el robot no cuenta con una representación del entorno (mapa) que se encuentra transitando. Por este motivo, es necesario que el robot vaya construyendo un mapa del entorno a medida que se va localizando en el mismo. Este problema donde se aborda la tarea de localización y construcción de un mapa del entorno de manera simultánea se denomina SLAM (*Simultaneous Localization and Mapping* [1, 2]).

Para abordar el problema de SLAM se ha hecho uso de diversos sensores tales como GPS, unidades inerciales, odómetros, láser y cámaras. Siendo estas últimas de principal interés dado su bajo costo, portabilidad y la abundante información que proveen de la escena observada. Además, las cámaras son sensores pasivos por lo que no interfieren con otros sensores y pueden utilizarse en entornos interiores, donde el uso de GPS (*Global Positioning System*) no es posible. Las técnicas para resolver SLAM que utilizan cámaras como sensor principal dieron lugar al término *Visual* SLAM. Dentro de este campo, el uso de las cámaras estéreo se ha vuelto de gran interés dado que permiten, a diferencia de las cámaras monoculares, reconstruir una escena tridimensional a escala del entorno. Actualmente existen numerosos enfoques para resolver el problema de SLAM mediante el uso de visión como por ejemplo: PTAM [3], LSD-SLAM [4], OKVIS [5], ORB-SLAM2 [6], VINS-Mono [7], DSO [8] y S-PTAM [9]. Sobre este último se hace foco en esta tesina.

En particular, en los últimos años, se han realizado importantes avances incorporando información semántica a la representación del mapa generado, dando origen al término SLAM Semántico (*Semantic SLA*M [10]). La inclusión de información semántica (por ejemplo, un sendero, una puerta, una heladera, etc...) permite al robot tener una mayor comprensión del entorno que se encuentra navegando y a su vez utilizar dicha información para mejorar su localización. Asimismo, un mapa rico en información permite que el robot pueda interactuar con el mismo de una manera más "inteligente" que si solo se tuviera una representación puramente

geométrica del entorno. Uno de los casos de uso más interesantes es el de conocer qué objetos se encuentran presentes en el entorno y la disposición de los mismos (orientación y posición). Otros ejemplos de información semántica relevante incluyen el reconocimiento del entorno (por ejemplo, tipo de habitación), la detección de caminos, y la detección de superficies planares.

Para lograr la extracción de información semántica del entorno, la comunidad científica de robótica móvil ha adoptado técnicas de Aprendizaje Automatizado (en inglés *Machine Learning* [11]). Una de las técnicas que ha cobrado mayor relevancia dentro del área de aprendizaje automático es la de redes profundas o aprendizaje profundo (en inglés *Deep Learning* [12]). Las redes profundas se han posicionado entre las técnicas de Aprendizaje Automatizado más utilizadas en la actualidad gracias a su gran efectividad comparada con los métodos tradicionales [13]. De esta manera se ha comenzado a aplicar *Deep Learning* en áreas como visión por computadora [14, 15]. Otros ejemplos involucran reconocimiento de texto manuscrito en diferentes idiomas [16] o el reconocimiento de rostros, ya con desempeño similar al de un ser humano [17].

1.1. Objetivo

El objetivo general del presente trabajo es desarrollar un método de SLAM que permita detectar objetos del entorno utilizando técnicas de *Deep Learning* y construir un mapa con información semántica de qué objetos y dónde se encuentran en el entorno en tiempo real. De manera más específica, se propone desarrollar un método de SLAM con localización de objetos basado únicamente en visión estéreo, capaz de obtener resultados de manera *online*. Para lograrlo, se extiende el sistema de SLAM estéreo S-PTAM (*Stereo Parallel Tracking and Mapping*) [18, 9] con un módulo de detección de objetos. Para la extracción de la información semántica se utiliza el método de detección de objetos basado en redes profundas llamado Faster R-CNN [19]. El sistema final debe funcionar de manera *online*, por tal motivo se dispone de una GPU (*Graphic Processing Unit*) para el procesamiento de imágenes.

La solución propuesta debe, en lo posible, utilizar estándares abiertos y software *open source* como lo son Robot Operating System (ROS) [20], OpenCV [21] y Caffe [22].

1.2. Organización del trabajo

El capítulo 2 presenta brevemente el problema de SLAM y el estado del arte en sistemas de SLAM semánticos. En el capítulo 3 se presentan las principales herramientas utilizadas y los fundamentos básicos de visión y *Deep Learning* usados a lo largo de todo el trabajo. En el capítulo 4 se detalla el método propuesto, presentando las modificaciones realizadas a la red neuronal y a S-PTAM para lograr obtener un mapa de objetos. En el capítulo 5 se evalúa y se discuten los resultados obtenidos. Por último, en el capítulo 6 se presentan las conclusiones y trabajo futuro que se derivan del presente trabajo de tesina.

Capítulo 2

Trabajos relacionados

En este capítulo, se presenta una breve introducción al problema de SLAM y se exponen los trabajos relacionados al área de SLAM semántico. También se presenta el impacto que tiene *Deep Learning* en los sistemas de SLAM.

2.1. SLAM

El problema de localización y mapeo simultáneo, conocido como SLAM (*Simultaneous Localization and Mapping*) es el proceso mediante el cual un robot móvil es capaz de construir incrementalmente un mapa del entorno que se encuentra transitando y, simultáneamente, usar este mapa para determinar su posición y orientación en el mismo. SLAM puede ser considerado como un problema del tipo "el huevo y la gallina", ya que se necesita un buen mapa para poder localizarse, y al mismo tiempo es necesario tener una buena estimación de la pose para poder elaborar un mapa preciso. El principal concepto que aportó SLAM para resolver este dilema es que ambos problemas (localización y mapeo) pueden ser abordados de forma simultánea.

La principal aplicación de algoritmos de SLAM es en el ámbito de la robótica móvil, donde su solución resulta útil para la navegación autónoma de los mismos. Recientemente, también se ha aplicado en el ámbito de VR (realidad virtual) y AR (realidad aumentada).

Debido al gran campo de aplicación, el problema de SLAM ha sido estudiado utilizando diferentes sensores: IMUs (Unidades de Medición Inercial) [23], SONAR [24], infrarrojos [25], escáneres láser [26], GPS (Sistema de Posicionamiento Global) [27], encoders [28] y cámaras [29, 30, 31, 32].

En la Figura 2.1.1 se ilustra el problema de SLAM. A partir de la detección y seguimiento de marcas naturales del ambiente (*landmarks*), los sistemas de SLAM pueden estimar tanto la posición del robot como la ubicación de estas marcas en el entorno. Intuitivamente, cada medición de una de estas marcas resulta en una restricción espacial entre la posición del robot y el *landmark*. El mapa es construido incrementalmente con las posiciones estimadas de dichos *landmarks*, las cuales son ajustadas a lo largo de la trayectoria a medida que son observadas desde distintas posiciones por el robot.

Una completa introducción a SLAM, descripción del problema y soluciones al mismo pueden encontrarse en [1, 2, 33].

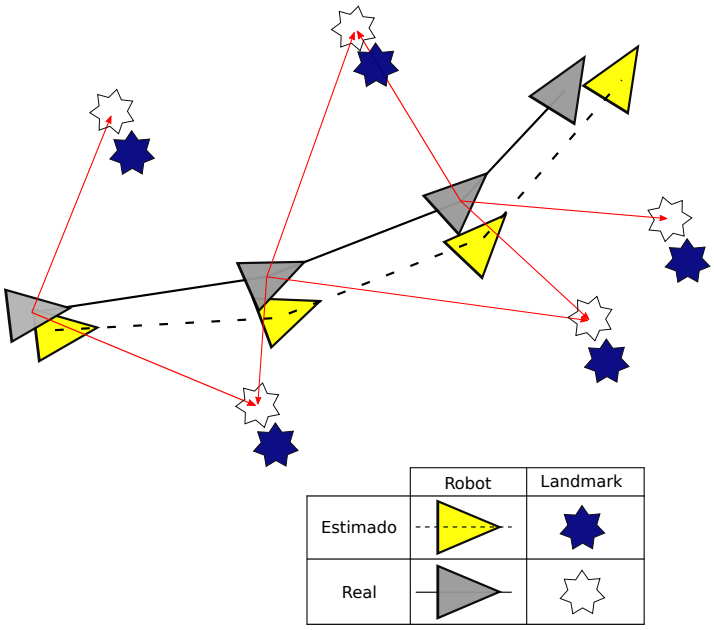


Figura 2.1.1: Representación del problema de SLAM. Las posiciones reales no son conocidas en ningún momento. El robot observa los *landmarks* desde cada posición a medida que se desplaza (lineas rojas). Simultáneamente, las posiciones del robot y de los *landmarks* son estimadas en base a estas observaciones. Imagen adaptada de [1].

2.1.1. Visual SLAM

Este trabajo se focaliza exclusivamente en métodos de SLAM que usan cámaras como sensores, denominados *Visual SLAM*. En este caso, los *landmarks* mencionados anteriormente pasan a ser marcas salientes en las imágenes: esquinas, bordes y zonas de alto gradiente. Estas marcas salientes reciben el nombre de *features*, y su extracción y asociación son pasos fundamentales en muchos métodos de *Visual SLAM*. Sin embargo, no todos sistemas de *Visual* SLAM usan *features*, por ejemplo, los métodos directos [34, 35, 36] y [4] que utilizan la información de intensidad de los píxeles directamente.

Dependiendo del tipo de cámara utilizada, existen dos enfoques predominantes en los sistemas SLAM, denominados monocular y estéreo [37]. Los sistemas de SLAM visual estéreo proveen información sobre la profundidad de los píxeles mediante una única observación [38, 39, 18], lo cual requiere mayor número de cómputos en el caso monocular [40, 41, 42]. Por otro lado, los sistemas estéreo deben procesar el doble de información visual.

En la Figura 2.1.2 puede observarse un típico mapa disperso de puntos (*features*) generado por un sistema de *Visual SLAM* basado en *features*, en este caso S-PTAM [18, 9]. Se puede apreciar que la información de este mapa no es adecuada para un robot que necesita interactuar con su entorno, ya que cada punto es igual a todos los demás y carente de significado alguno. La creciente demanda de autonomía en robots frente a tareas más complejas ha dado lugar a la búsqueda de cómo incorporar información semántica a este mapa, siendo este enfoque denominado SLAM *Semántico*.

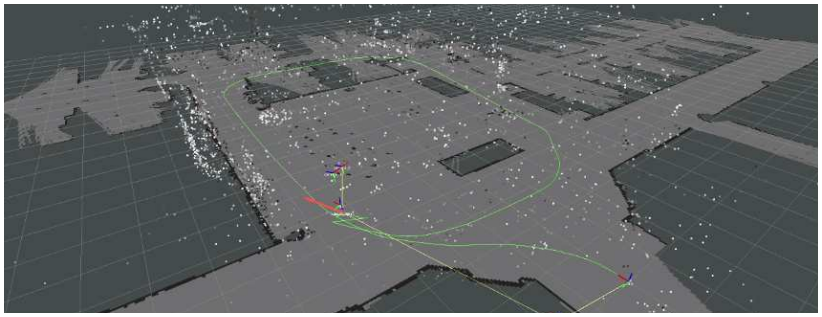


Figura 2.1.2: Mapa disperso generado por el sistema S-PTAM [18] sobre una secuencia del Dataset Level 7 [43] en un entorno de oficinas. Se observa el mapa de puntos (*map points*) disperso y la trayectoria del robot (línea verde).El contorno de las habitaciones fue agregado como referencia.

2.2. El estado del arte en SLAM semántico

En los últimos años las técnicas de *deep learning* han formado parte del estado del arte en tareas como clasificación de imágenes y detección de objetos. En este capítulo se presentan los trabajos más relevantes en el área de SLAM semántico, primero mencionando los que no utilizan *deep learning* y luego el estado del arte que sí hace uso de esta nueva tecnología.

2.2.1. Trabajos relevantes en SLAM semántico

En [44] se plantea el problema de *Semantic Structure from Motion* (SSFM), donde se define el marco teórico para el ajuste simultáneo de puntos, superficies y objetos, a partir de dos vistas de una misma escena. En este caso se emplea un tipo de SVM (*Support Vector Machine*) para realizar la clasificación de los objetos. El sistema fue concebido para operar de forma *offline*, lo que es contrario a lo que se propone en esta tesina, donde se busca una performance de tiempo real.

De manera similar, SLAM++ [45] se enfoca en el problema de SLAM orientado a objetos, presentando un sistema SLAM monocular de tiempo real, basado en imágenes RGB-D, donde las principales entidades sobre las que se realiza la localización y el mapeo son los objetos. Es decir, los objetos son los principales *landmark*s (ver Figura 2.2.1), dotando al sistema de localización 3D, relocalización, *loop closure* y reconstrucción densa. Sin embargo, en SLAM++, los objetos están representados como *meshes* 3D que deben conocerse previamente a la exploración del entorno. Es decir, funcionan más como marcadores 3D que como verdaderos objetos semánticos (por ejemplo, si existe en el entorno una silla cuyo modelo 3D es desconocido por el sistema, el método de detección fallará). Esto difiere del enfoque propuesto en esta tesina, donde se propone poder reconocer objetos (de determinadas clases) de manera general.

En [46] se hace uso de un sistema de SLAM monocular RGB para mejorar la performance de un detector de objetos basado en *features* y VBoW (*Visual Bag of Words*), gracias a la acumulación de detecciones de distintos puntos de vista. A diferencia del trabajo realizado en esta tesina, este trabajo se enfoca principalmente en la detección, y no en la localización 3D de objetos.

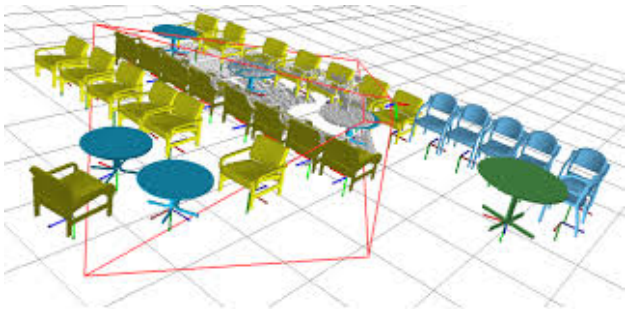


Figura 2.2.1: Mapa del entorno elaborado por SLAM++[45] con la pose estimada de la cámara (frustum en rojo). La cantidad de objetos en la escena le proporciona un gran número de observaciones de modelos 3D que el sistema usa para localizarse. Imagen obtenida de [45].

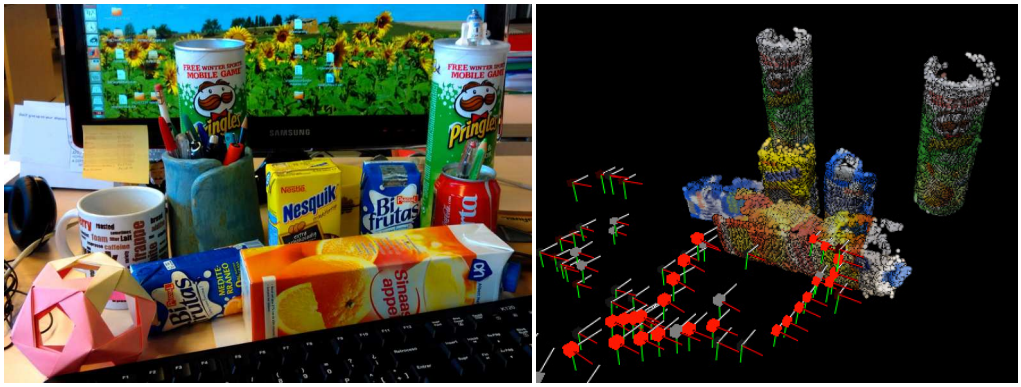


Figura 2.2.2: Una escena (imagen izquierda) filmada y su reconstrucción (imagen derecha) a través de la nube de puntos generada por Real-time Monocular Object SLAM[47]. También se puede observar el recorrido de la cámara (imagen derecha) mediante la visualización de las distintas poses estimadas (en rojo). Imagen obtenida de [47].

Otro trabajo relevante es el de Real-time Monocular Object SLAM [47], que realiza detección de objetos utilizando *features* ORB [48] y BoW [49]. Incorpora una base de datos de alrededor de 500 objetos, escaneados con una cámara monocular RGB-D. El método funciona en tiempo real y sigue el paradigma de PTAM [3], usando una cámara RGB-D como sensor. Además, cada modelo de objeto proporciona restricciones rígidas entre puntos particulares que pertenecen a él. Cuando esos puntos son detectados, se incorporan las restricciones entre ellos impuestas por el objeto al mapa de PTAM a optimizar, logrando que los objetos contribuyan a bajar el error de localización. Una desventaja de este método es que requiere objetos con un alto nivel de textura para lograr identificarlos, como se muestra en la Figura 2.2.2, además de no poder identificar objetos que no pertenecen a la base de datos.

También se puede mencionar a Dense Planar SLAM [50], un método similar a SLAM++ que extrae y utiliza información semántica de nivel intermedio, como lo son las superficies planares. Si bien esta información puede ser más útil que una nube de puntos, no permite realizar distinciones de más alto nivel (por ejemplo, entre una pared y una puerta). Las superficies detectadas por el sistema se pueden ver en la Figura 2.2.3.

Una de las motivaciones principales del trabajo realizado en esta tesina fue la de incorporar

Figura 2.2.3: Una escena en una cocina (imagen superior izquierda), la detección de superficies planares (imagen superior derecha) por Dense Planar SLAM y una posible aplicación en realidad aumentada (imagen inferior). Imagen obtenida de [50].

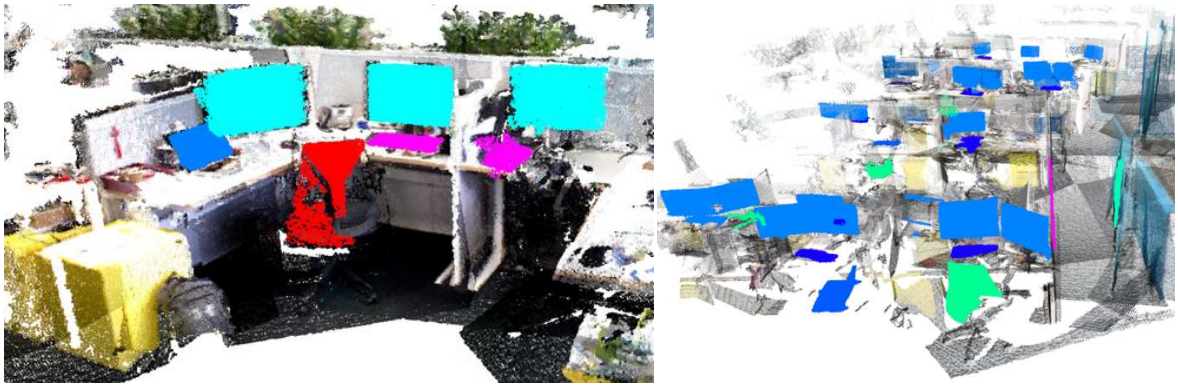


Figura 2.2.4: Mapa semántico de objetos generado por el sistema propuesto en "Meaningful maps with object-oriented semantic mapping". Imagen obtenida de [51].

técnicas de *Deep Learning* a un sistema SLAM para que se extraiga información semántica de los objetos de cualquier escena y no esté limitado a la detección de objetos ya conocidos de manera *offline*.

2.2.2. Incorporando técnicas de *Deep Learning*

En cuanto al uso de técnicas de *Deep Learning* en métodos de SLAM semántico se puede mencionar a [51], un trabajo reciente que utiliza el sistema de SLAM ORB-SLAM2 [6] con imágenes RGB-D y la red profunda SSD [52] para realizar detección de objetos y estimar sus ubicaciones en el mapa. La Figura 2.2.4 muestra la reconstrucción de un mapa denso de puntos, resaltando aquellos que pertenecen a objetos.

El uso de la red profunda para la extracción de información semántica libera al sistema del requerimiento de contar con una base datos de modelos a reconocer, como es el caso de algunos de los trabajos mencionados en la sección anterior (Sección 2.2). Una diferencia con el trabajo aquí propuesto, radica en el método para localizar los objetos una vez obtenidas las detecciones de la red neuronal. En [51], se utiliza un costoso (más costoso que el procesamiento realizado por la red neuronal) algoritmo de detección de planos basado en [53] para obtener

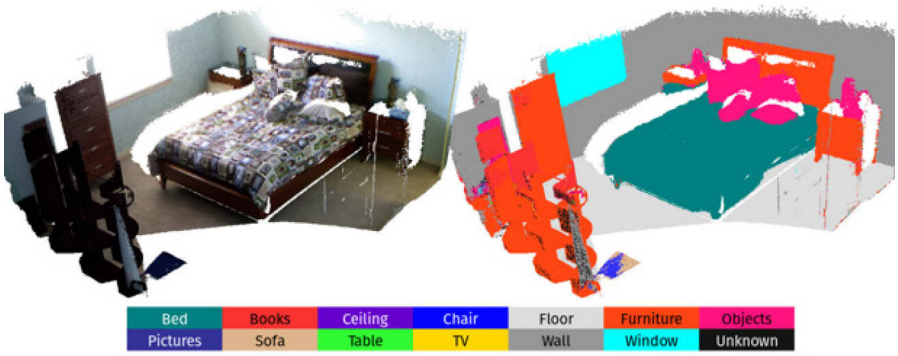


Figura 2.2.5: Mapa denso (izq) y segmentación semántica (der) realizada por el sistema SemanticFusion. Imagen obtenida de [54].

una segmentación de puntos 3D del objeto que determinan su posición. En esta tesina se modifica la red neuronal para que estime la pose 3D de los objetos de manera directa que luego es refinada. Este trabajo no publicó errores de localización de los objetos en la escena, aunque sí reporta métricas relacionadas a la recuperación de información de los objetos de la escena. Sin embargo, este trabajo comparte con esta tesina la limitación de ubicar objetos en el mapa, pero no usarlos para mejorar la localización. Es decir, el método de SLAM sigue funcionando a base de puntos y no de objetos.

El sistema SemanticFusion presentado en [54], combina un método de SLAM denso, propuesto en [55], con una red neuronal convolucional (definición en la Sección 3.3.3) que realiza la clasificación a nivel de píxeles, planteando una inferencia bayesiana para fusionar cada resultado de la red. De esta manera se obtiene una clase para etiquetar semánticamente cada punto 3D del mapa denso como se muestra en la Figura 2.2.5.

Deep Learning también puede ser aplicado en el contexto de sistemas SLAM para obtener otro tipo de información del entorno. Por ejemplo, en [56] se propone utilizar la red convolucional presentada en [57] para refinar el cálculo de profundidad. La red convolucional estima, dada una imagen, la profundidad de la escena observada, y esta información se utiliza para completar áreas de baja textura donde fallan los métodos basados en la triangulación de puntos.

Capítulo 3

Conceptos previos

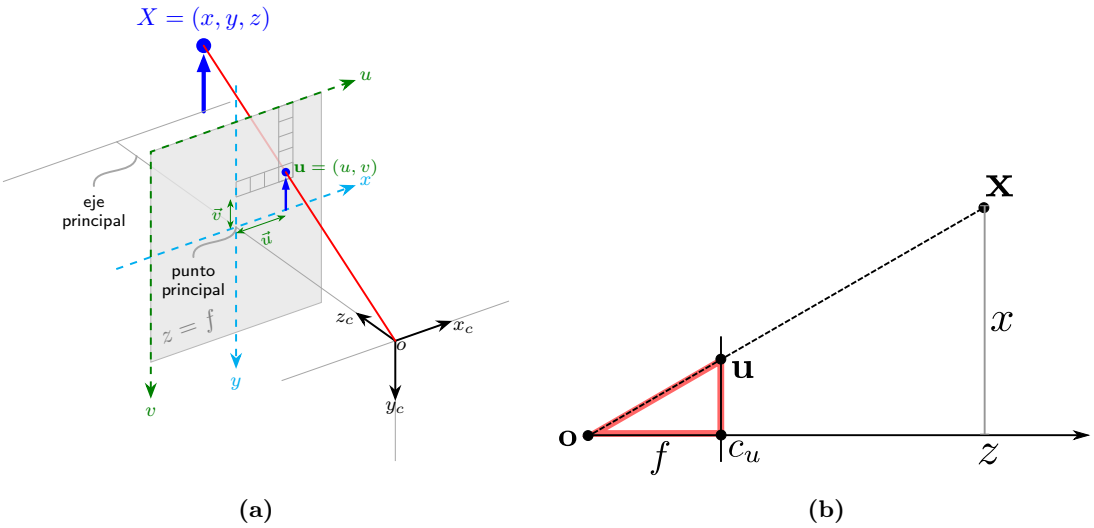
En este capítulo se exponen los conceptos necesarios para poder extraer mediciones espaciales y semánticas del mundo real, utilizando las imágenes proporcionadas por una cámara. Estos incluyen el modelo de cámara usado para relacionar puntos del espacio con los de la imagen y las técnicas de *Machine Learning* (en particular *Deep Learning*) aplicadas al problema de detección de objetos. También se detallan las principales herramientas utilizadas en este trabajo: el sistema de SLAM S-PTAM [9] y la red neuronal Faster R-CNN [19].

3.1. Modelo de cámara *pinhole*

El modelo de cámara *pinhole* (cámara estenopeica) describe la relación entre las coordenadas de un punto en el espacio 3D y su proyección en el plano de la imagen de una cámara *pinhole* ideal, donde la apertura esta definida por un punto y no se usa un lente para enfocar. Este modelo no toma en cuenta distorsiones geométricas ni otros efectos producidos por un lente, pero es útil como una primera aproximación de la correspondencia entre el mundo 3D y una imagen 2D [29].

En el modelo de cámara *pinhole,* el punto de la imagen $\mathbf{u} = \begin{bmatrix} u & v \end{bmatrix}^\top$ es determinado como la intersección entre el *plano de la imagen* —también denominado *plano focal*— y el rayo que une el punto del mundo 3D $\mathbf{x} = \begin{bmatrix} x & y & z \end{bmatrix}^\top$ con el *centro focal* $\mathbf{o}$ de la cámara. En la Figura 3.1.1 se ilustra esta definición y las relaciones geométricas involucradas en este modelo de cámara. Llamamos *rayo principal* o *eje principal* de la cámara al rayo que se origina en el centro focal $\mathbf{o}$ y es perpendicular al plano de la imagen. El punto donde este rayo intersecta al plano de la imagen es denominado *punto principal* $\mathbf{c}$. Consideremos que el centro focal se encuentra en el origen de coordenadas y el plano focal expresado como $Z = f$. Utilizando la propiedad de semejanza de triángulos puede verse que el punto 3D $\mathbf{x} = \begin{bmatrix} x & y & z \end{bmatrix}^\top$ es mapeado al punto $\mathbf{u} = \begin{bmatrix} fx/z & fy/z \end{bmatrix}^\top$ en el plano de la imagen. Esto da lugar a la siguiente proyección central:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} \longmapsto \begin{bmatrix} fx/z \\ fy/z \end{bmatrix}. \tag{3.1.1}$$



(a) **(b)**

Figura 3.1.1: Relaciones geométricas definidas por el modelo de cámara *pinhole*. En (a) el plano de la imagen es perpendicular al *eje principal* y se encuentra a distancia f del centro focal de la cámara **o**, que es el origen del sistema de coordenadas de la cámara. El punto 3D **X** se proyecta sobre el plano de la imagen en el punto **u**. En (b) se ilustra la proyección sobre el plano XZ de la cámara de la misma situación descripta en (a), mostrando las relaciones geométricas entre **X** y **u** para las coordenadas x y z (análogo para las coordenadas y y z).

Matriz de calibración intrínseca K

En la ecuación (3.1.1) se asume que el origen de coordenadas del plano de la imagen se encuentra en el punto principal. Esto no se tiene que cumplir necesariamente, y el punto principal puede ubicarse en las coordenadas $\begin{bmatrix} c_u & c_v \end{bmatrix}^\top$ en lugar de la coordenada canónica $[0, 0]$.

Notamos con $\dot{\mathbf{x}} = \begin{bmatrix} x & y & z & 1 \end{bmatrix}^\top$ la representación en coordenadas homogéneas del punto 3D $\mathbf{x} = \begin{bmatrix} x & y & z \end{bmatrix}^\top$. De esta manera, la proyección central del modelo de cámara *pinhole* de la ecuación (3.1.1) puede expresarse como:

$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \longmapsto \begin{bmatrix} fx + zc_u \\ fy + zc_v \\ z \end{bmatrix} = \begin{bmatrix} f & 0 & c_u & 0 \\ 0 & f & c_v & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}. \tag{3.1.2}$$

Definiendo la *matriz de calibración intrínseca* $\mathbf{K}$ como:

$$\mathbf{K} = \begin{bmatrix} f & 0 & c_u \\ 0 & f & c_v \\ 0 & 0 & 1 \end{bmatrix}, \tag{3.1.3}$$

entonces la ecuación (3.1.2) se puede reescribir de la siguiente manera:

$$\dot{\mathbf{u}} = \mathbf{K} \begin{bmatrix} \mathbf{I} & \mathbf{0} \end{bmatrix} \dot{\mathbf{x}} \,.$$

Matriz de proyección P

En general, los puntos del ambiente son expresados en referencia a un sistema de coordenadas conocido como *mundo*. Para terminar de definir la correspondencia entre estos puntos y los observados en la imagen 2D hace falta agregar la transformación entre los sistemas de coordenadas de la cámara y el mundo (Figura 3.1.2), llamada la *matriz extrínseca*.

La matriz extrínseca tiene entonces, la forma matricial de una transformación rígida: $[\mathbf{R} \mid \mathbf{t}]$, donde $\mathbf{R}$ es una matriz de rotación de 3×3 y $\mathbf{t} \in \mathbb{R}^3$ es un vector columna de traslación.

En coordenadas homogéneas, sea $\dot{\mathbf{x}}^{\mathrm{w}} = \begin{bmatrix} x & y & z & 1 \end{bmatrix}^\top$ un punto del espacio en referencia al sistema de coordenadas del mundo y $\dot{\mathbf{x}}^{\mathrm{c}}$ el mismo punto expresado en el sistema de coordenadas de la cámara, entonces la transformación $\mathbf{E}^{\mathrm{cw}}$está dada por:

$$\dot{\mathbf{x}}^{\mathrm{c}} = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \mathbf{E}^{\mathrm{cw}} \dot{\mathbf{x}}^{\mathrm{w}} \,.$$

En particular, $\mathbf{E}^{\mathrm{cw}}$es una transformación perteneciente al grupo de movimientos de cuerpo rígido 3D (*Lie Group*, $\mathbf{SE(3)}$) [58].

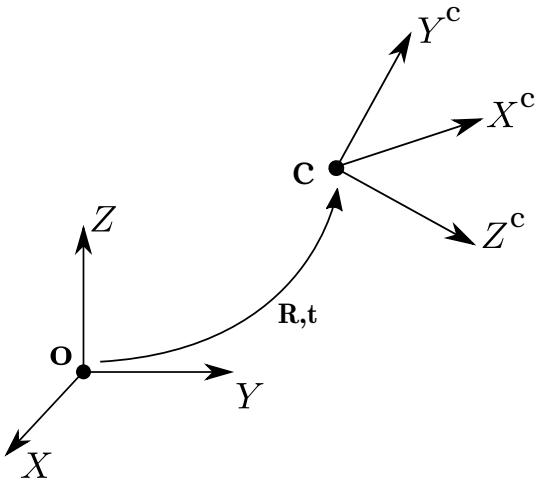


Figura 3.1.2: En general, el sistema de coordenadas de la cámara (**C**) se encuentra relacionado con el sistema de coordenadas del mundo (***O***) mediante una rotación **R** y una traslación **t**.

Utilizando la matriz de calibración intrínseca **K** definida en la ecuación (3.1.3),y la matriz extrínseca $\mathbf{E}^{\mathrm{cw}} = [\mathbf{R} \mid \mathbf{t}]$, se define la matriz de proyección **P**:

$$\mathbf{P} = \mathbf{K} \begin{bmatrix} \mathbf{R} & \mathbf{t} \end{bmatrix} . \tag{3.1.4}$$

De esta manera, es posible proyectar cualquier punto 3D $\dot{\mathbf{x}}^{\mathrm{w}}$ en el sistema de coordenadas del mundo al correspondiente punto $\dot{\mathbf{u}}$ en el plano de la imagen mediante:

$$\dot{\mathbf{u}} = \mathbf{P}\dot{\mathbf{x}}^{\mathrm{w}} . \tag{3.1.5}$$

3.2. S-PTAM

En [18] se presenta un sistema SLAM basado en *features* llamado S-PTAM (*Stereo Parallel Tracking and Mapping*). Utilizando una cámara estéreo como sensor principal, construye y mantiene un mapa disperso del entorno para obtener una localización precisa de la cámara. S-PTAM sigue el enfoque propuesto en PTAM [3] (*Parallel Tracking and Mapping*) que divide las tareas de localización y mapeo en dos hilos de ejecución diferentes —localización y construcción del mapa (*tracking and mapping*)— para aprovechar la capacidad de cómputo de procesadores con dos o más núcleos (ver Figura 3.2.1).

El sistema se inicializa con un mapa vacío y la cámara en el centro del sistema de referencia del mundo. Al recibir el primer par de imágenes estéreo, se agregan puntos al mapa mediante la triangulación de características salientes de las imágenes. Para cada nuevo par de imágenes estéreo, el hilo de *tracking* estima la posición actual minimizando el error de re-proyección entre los puntos del mapa proyectados y sus correspondencias en la imagen. Este par estéreo puede ser seleccionado y agregado al mapa, en cuyo caso es denominado como *keyframe*. Al agregar el *keyframe* al mapa, aquellos features que no fueron asociados con puntos en el mapa

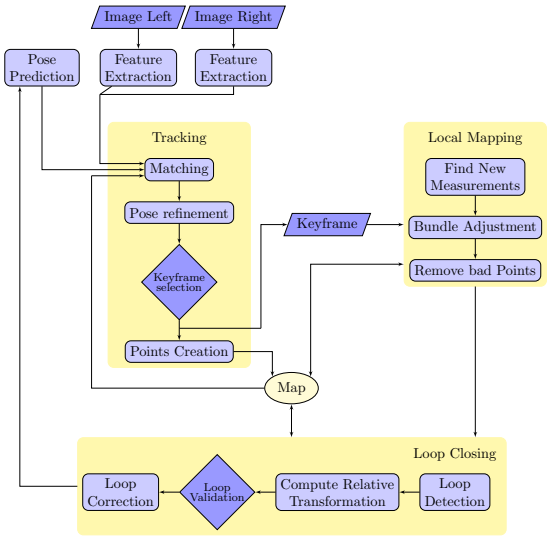


Figura 3.2.1: Esquema general del sistema S-PTAM con el módulo de Loop Closure.

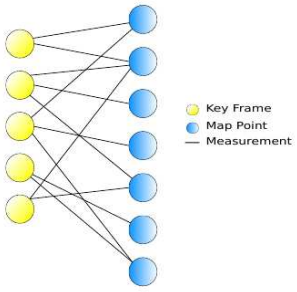


Figura 3.2.2: Ejemplo grafo bipartito que representa al mapa reconstruido por S-PTAM.

son triangulados desde el par de imágenes estéreo, insertándolos junto con el *keyframe* al grafo de puntos y posiciones, mantenido por S-PTAM. Un ejemplo del grafo de *keyframes* y *map points* mantenido por S-PTAM se ilustra en la Figura 3.2.2.

El hilo de *mapping* busca activamente fortalecer el mapa agregando restricciones (aristas) al grafo de S-PTAM, encontrando observaciones de *map points* desde distintos *keyframes*. Para optimizar las posiciones de los entes del mapa, el hilo de *mapping* realiza un *Local Bundle Adjustment* (LBA) sobre el subgrafo de puntos y posiciones definido por los últimos *keyframes* agregados.

El sistema S-PTAM también cuenta con un módulo de *Loop Closure*, responsable de reconocer lugares ya visitados y actualizar el grafo de manera acorde. Esta actualización resulta en una reducción del error de localización, que de otra manera crecería indefinidamente. El módulo de *Loop Closure* no estaba integrado a S-PTAM al comienzo de esta tesina, por lo tanto, no fue utilizado en este trabajo. S-PTAM se implementa como un nodo del *framework* ROS [20] (del inglés, *Robot Operating System*).

3.3. Redes Convolucionales

En esta sección se explican los conceptos necesarios para entender las técnicas usadas para extraer información semántica de una imagen en este trabajo. Se presentaran en orden decreciente de generalidad, comenzando con conceptos de Aprendizaje Automatizado (en inglés, *Machine Learning*), luego redes neuronales y finalizando en redes neuronales aplicadas al problema de detección de objetos. La mayoría de los conceptos son abordados en los libros [11] y [12].

3.3.1. Aprendizaje Automatizado

En 1997, Tom Mitchell dio una definición formal de los algoritmos estudiados en el área de Aprendizaje Automatizado [11]: *"Se dice que un programa aprende de la experiencia E con respecto a cierto tipo de tarea T y una medida de performance P, si el desempeño en tareas de T según la medida P, mejora con la experiencia E"*. Mitchell se refiere a los programas que evolucionan con la experiencia acumulada.

Este trabajo se encuentra en el contexto de lo que se conoce como *Aprendizaje Automatizado Supervisado*, donde se dispone a aprender una función $h : X \to Y$ a partir de datos de entrenamiento $D = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\} \in X \times Y$, donde cada par indica la salida deseada para un determinado valor de entrada. Si Y tiene valores reales se dice que es un problema de *regresión*, mientras que si Y tiene valores discretos se trata de un problema de *clasificación*.

La función h que se busca aprender usando determinado algoritmo de aprendizaje pertenece a una familia de funciones $\mathbb{H}$, llamada el *espacio de hipótesis*. En muchos casos se encuadra la búsqueda de h dentro del espacio de hipótesis como un problema de optimización de la siguiente manera: se asume que los datos de entrenamiento D son muestras independientes de cierta distribución conjunta $P(x, y)$ sobre X e Y. Esto permite modelar el ruido en los datos, ya que y no depende determinísticamente de x sino que es una variable aleatoria con una distribución $P(y|x)$ para un x fijo. También se asume que se tiene una función valuada real no negativa llamada de *costo* o *loss* $L(\hat{y}, y)$ que mide que tan diferente es la predicción $\hat{y}$ del verdadero valor de salida y. Entonces el objetivo del algoritmo de aprendizaje se puede considerar como encontrar la función h^* que minimice la esperanza de la función de costo, es decir, $h^* = \arg\min_{h \in \mathbb{H}} \mathbf{E}[L(h(x), y)]$. Como en la mayoría de los casos, la distribución $P(x, y)$ no se conoce, la esperanza de la función de costo se aproxima en base a la muestra finita D que son los datos disponibles. A esto se lo llama *minimizar el riesgo empírico*, formalmente: $h^* = \arg\min_{h \in \mathbb{H}} \frac{1}{m} \sum_{i=1}^{m} L(h(x_i), y)$.

Una de las desventajas de minimizar el riesgo empírico es que se puede obtener una función $h*$ que tenga un menor error en el conjunto de datos D, que en otros datos de $P(x, y)$, es decir, que no generalice bien. En este caso se denomina que hubo un *sobreajuste* o *overfitting* (en inglés) a los datos de entrenamiento.

Por esta razón, se suele reservar datos fuera del conjunto de entrenamiento para poder calcular el promedio sobre otra muestra independiente de $P(x, y)$ para tener una estimación menos sesgada de la esperanza. A estas muestras se la denominan conjunto de *validación* si se

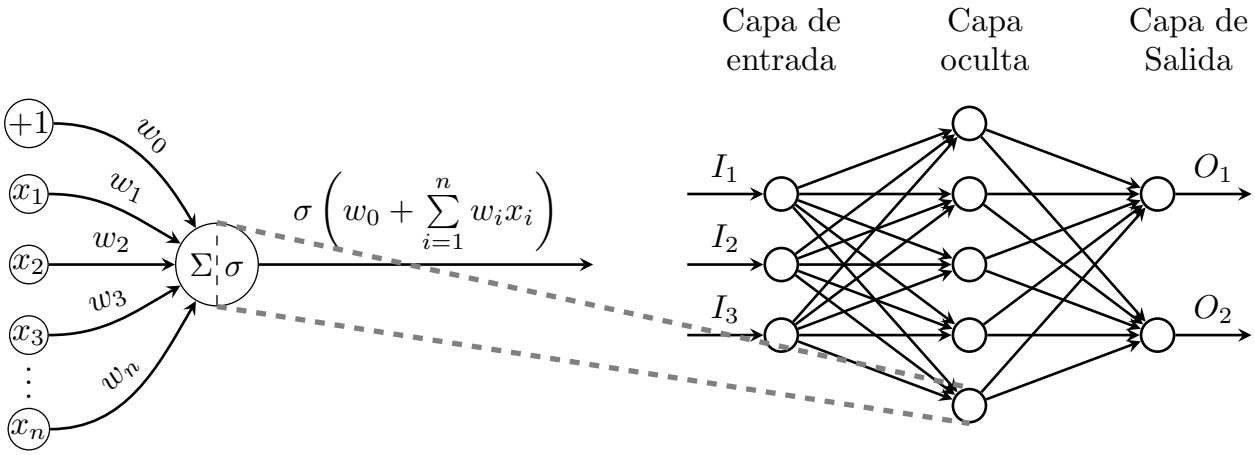


Figura 3.3.1: Neurona o Perceptrón en el contexto de una red neuronal. Las señales fluyen de izquierda a derecha. Notar que w_0 no está asociado a la salida de ninguna neurona, y es llamado *bias*.

utiliza para la selección de hiperparámetros (parámetros de alto nivel del modelo) y conjunto de *test* si se utiliza al final del entrenamiento para evaluar el modelo.

Además de elegir el espacio de hipótesis $\mathbb{H}$ donde buscar h^*, se puede establecer una preferencia por ciertas funciones, de manera implícita o explícita, según la tarea a aprender. Este concepto es conocido como *regularización* y se define como cualquier modificación al algoritmo de aprendizaje cuyo objetivo es reducir el error de generalización (error en el conjunto de *test*) pero no el error de entrenamiento.

Los métodos más emblemáticos del Aprendizaje Supervisado son: *Árboles de Decisión*, *Support Vector Machine*s, *kNN*, *Naive Bayes* y *Redes Neuronales*, siendo éstas últimas las usadas en este trabajo.

3.3.2. Redes Neuronales Artificiales

Las *redes neuronales artificiales* (de ahora en más, redes neuronales) son una familia de modelos que provee un enfoque robusto para aproximar funciones a valores reales, discretos y vectoriales.

Las redes neuronales están compuestas por una colección de unidades conectadas, llamadas neuronas artificiales o *perceptrones*. En implementaciones comunes, la señal en una conexión entre neuronas es un número real y la salida de cada neurona se computa mediante una función no lineal de la suma de sus entradas (ver Figura 3.3.1). Estas conexiones normalmente tienen asociado un peso w_i que se ajusta en el período de entrenamiento. Formalmente una neurona modela la función:

$$y = \sigma\left(w_0 + \sum_{i=1}^{n} w_i x_i\right),$$

donde x_i es un valor de entrada conectado a través de una conexión con peso w_i, y σ es la función de activación, usualmente una de las siguientes:

- $\sigma(x) = \frac{1}{1+e^{-x}}$ (*sigmoidea*)
- $\sigma(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$ (*tangente hiperbólica*)
- $\sigma(x) = \begin{cases} 0 & \text{si} \quad x \leq 0 \\ x & \text{si} \quad x > 0 \end{cases}$ (lineal rectificada).

Esta última función de activación es la utilizada en esta tesina, donde las unidades que emplean el rectificador son conocidas como ReLU (del inglés *Rectified Linear Unit*).

Típicamente, las neuronas artificiales están organizadas en capas, donde las neuronas de una capa tienen como entrada únicamente las salidas de las neuronas de la capa anterior y las señales viajan desde una primera capa de entrada hasta la capa de salida, ilustrado en Figura 3.3.1. Toda capa que no es la de entrada ni la de salida es denominada *capa oculta* o *hidden layer* (en inglés).

Entrenar una red artificial implica obtener un vector de pesos $\mathbf{w} = (w_0, w_1, \ldots, w_n)$ para cada neurona de la red, lo que resulta en un conjunto total de parámetros a entrenar $\mathcal{H} = \{(\mathbf{w}_0, \mathbf{w}_1, \ldots, \mathbf{w}_m)\}$ siendo m la cantidad total de neuronas. Las capas donde cada una de sus neuronas se conecta con todas las neuronas la capa anterior son denominadas *fully connected*.

3.3.3. Redes Neuronales Convolucionales

Las redes neuronales convolucionales, o CNNs (del inglés *Convolutional Neural Networks*) fueron propuestas por Yann LeCun et al [59] a fines de los '80, y relegadas por la comunidad de visión por computadora, por demandar demasiado poder de cómputo. En 2012 volvieron a tomar relevancia cuando *Krizhevsky et al* [14] ganan la competencia 2012 *ImageNet Challenge* por amplio margen, y desde ese entonces ocupan un lugar principal en el estado del arte de *Machine Vision*.

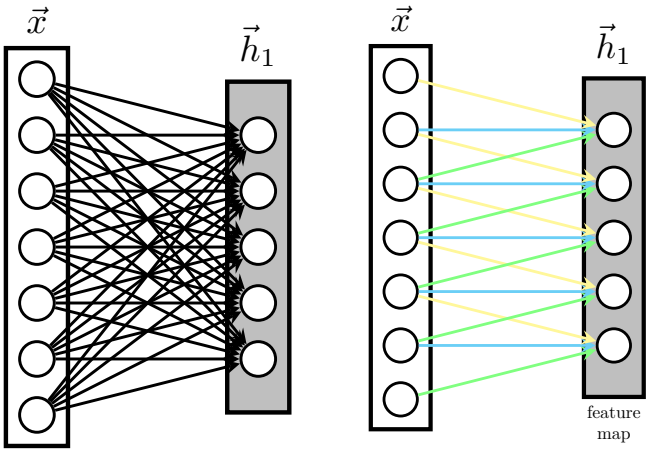
Una CNN es una red neuronal con pesos compartidos y conexiones locales, donde cada neurona de salida de la etapa convolucional se conecta con un subconjunto de neuronas en la entrada de manera regular y compartiendo los pesos $\mathbf{w}$ para cada neurona de salida. En la Figura 3.3.2 se puede ver la comparación de una capa *fully connected*, con una convolucional.

En el contexto de este trabajo, las redes neuronales son aplicadas a imágenes, las cuales se representan como una grilla 2D de píxeles. En el caso de imágenes a color (RGB) se tienen tres matrices 2D de valores, o un tensor 3D de tamaño $3 \times \mathit{ancho} \times \mathit{alto}$. En las distintas etapas de procesamiento de imágenes se dispone de datos organizados como c *canales* de matrices 2D.

La aplicación de una capa convolucional puede ser definida similarmente a una convolución (muchas librerías de *Machine Learning* llaman a la operación una convolución, pero matemáticamente se denomina *cross-correlation*) que en una imagen 2D resulta:

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(i+m, j+n) K(m,n),$$

donde I representa nuestra imagen de entrada 2D, y K es también una matriz 2D llamada el *kernel* de la convolución. La salida S es usualmente referida como un *feature map*, ver Figura 3.3.3.



(a) Cada conexión tiene asociado un peso distinto.

(b) Las conexiones de un mismo color comparten el peso asociado.

Figura 3.3.2: Comparación de conexiones entre una capa fully connected y una convolucional.

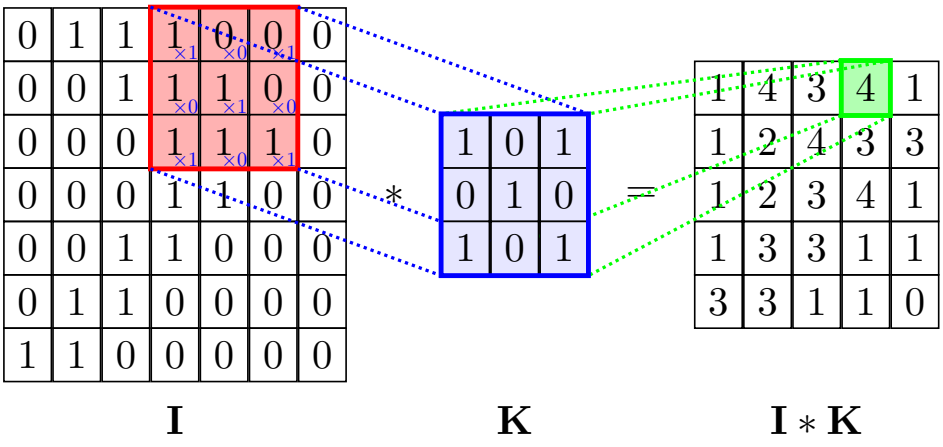


Figura 3.3.3: Ejemplo de una convolución 2D. I representa los datos de entrada y K el kernel de la convolución aplicada. A la matriz resultante I*K se la denomina *feature map.*

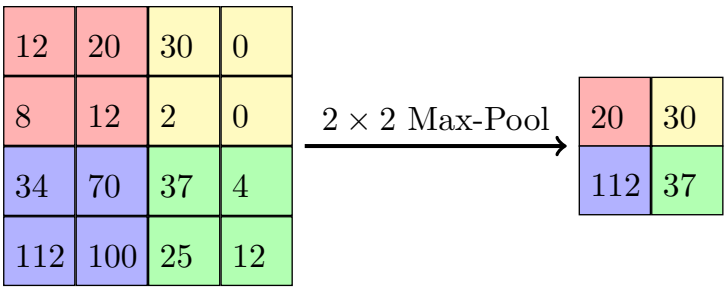


Figura 3.3.4: Ejemplo de operación *max pooling*. Se produce un *downsampling* de la matriz tomando el valor máximo de cada región.

Para cada capa convolucional, los pesos de su *kernel* son los que se obtienen durante la etapa de aprendizaje, que intuitivamente funcionan como filtros de ciertas características relevantes en la imagen, como pueden ser aristas y esquinas en capas cercanas a la entrada, y formas complejas en capas posteriores. Las CNN son especialmente útiles para reconocer patrones en imágenes porque poseen invarianza traslacional (un *feature* logra la misma activación independientemente de dónde se encuentra en la imagen).

Se puede hacer una distinción entre dos tipos de capas utilizadas en la práctica en las CNNs:

- Capas de Convolución: Se encargan de propagar a través de toda su entrada el *kernel* antes mencionado. La única salvedad es que se define un paso de aplicación, de esta manera "salteando" algunas posiciones del *kernel*. Una razón para hacerlo, es para reducir el costo computacional con la desventaja que se está produciendo un *downsampling* de la salida convolucional completa. Sea V el tensor de entrada, Z el de salida y K el *kernel* de 4D donde $K_{i,j,k,l}$ da el peso de la conexión entre una neurona en el canal i de la salida y otra en el canal j de la entrada, con un *offset* de k filas y l columnas entre la neurona de salida y la de entrada, y un paso s en las dos direcciones, entonces la operación c queda definida como:

$$Z_{i,j,k} = c(\mathbf{K}, \mathbf{V}, s)_{i,j,k} = \sum_{l,m,n} \left[V_{l,(j-1)\times s+m,(k-1)\times s+n} K_{i,l,m,n} \right],$$

  donde l, m y n toman todos los valores tal que los índices usados en la convolución sean válidos (los -1 están porque en álgebra lineal la primer entrada de un vector tiene índice 1).

- Capas de Pooling: Una función de *pooling* determina la salida en cierto lugar de la red con un resumen estadístico de unidades vecinas en su entrada. Por ejemplo, *max pooling*, reporta el valor máximo de salida de un entorno rectangular. Otras funciones de *pooling* pueden ser el promedio de un entorno rectangular, la norma $\mathtt{l}_2$ de un entorno rectangular o un promedio ponderado basado en distancia al píxel central. Las capas de pooling reducen la dimensionalidad de la red, aumentando la eficiencia, y ayuda a controlar el sobreajuste. Un ejemplo de la operación *max pooling* se ilustra en la Figura 3.3.4.

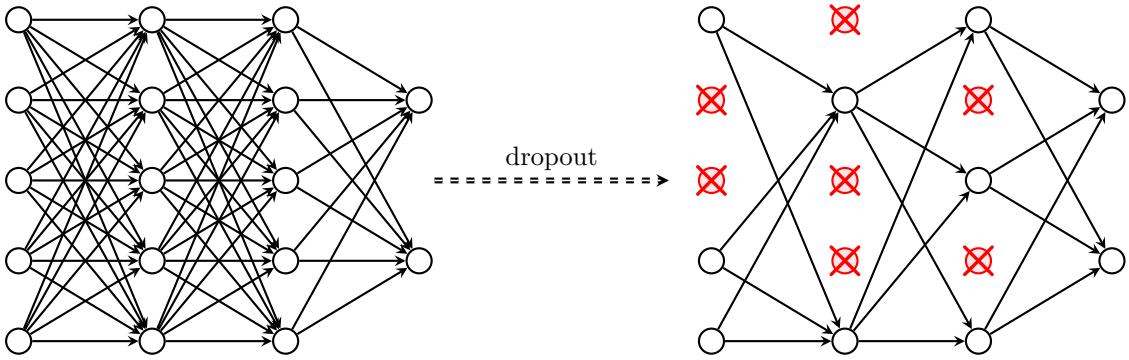


Figura 3.3.5: Ejemplo de como un dropout = 0.5 incide en las conexiones de la red. En la siguiente iteración de entrenamiento serán (probablemente) otras las conexiones deshabilitadas.

Usualmente una capa conceptual de las CNNs, que se repite varias veces, involucra 3 etapas: una primera capa de convolución, seguida por una de regularización (una capa ReLU, por ejemplo) y finalmente una capa de *pooling*. Un ejemplo de una arquitectura de red CNN es la de VGG16 [60], ilustrada en la Figura 3.4.5 (ubicada en la Sección 3.4.4).

3.3.4. Dropout

Un tipo de regularización usado en este trabajo es *dropout*. La técnica de *dropout* [61] provee un método de bajo costo computacional que reduce el *overfitting*. *Dropout* trata de simular el entrenamiento de distintos submodelos de la red, removiendo neuronas al azar. Esto resulta en una manera eficiente de calcular el "promedio" de los distintos submodelos entrenados sobre el mismo dataset, y así evitar co-adaptaciones complejas entre las neuronas al dataset de entrenamiento. Una manera de especificar el dropout es elegir una probabilidad p que va a tener cada nodo de permanecer en la red en una iteración particular del entrenamiento. En la Figura 3.3.5 se ilustra como el *dropout* deja afuera de la red a ciertos nodos (y sus aristas) en una iteración determinada del entrenamiento.

3.4. Detección de Objetos con Redes Profundas

3.4.1. Definición del problema

Uno de los hitos en el uso de las redes profundas en visión por computadora se dio con AlexNet[14] y su aplicación al problema de clasificación en ImageNet. El problema de clasificación se define como la tarea de asignar una única clase a toda una imagen en función de cuál es el objeto o tema preponderante. Sin embargo, en algunas aplicaciones no alcanza con sólo clasificar una imagen, sino que además se debe encontrar las varias instancias de objetos y estimar su posición (ver Figura 3.4.1). A este problema de localizar y clasificar múltiples objetos en una imagen se lo denomina *detección de objetos*. Hay varias maneras de caracterizar una región en una imagen, desde figuras simples (un circulo o una elípse) hasta más complejas (un polígono que se ajuste perfectamente al objeto). También es posible realizar una clasificación

Figura 3.4.1: El problema de clasificación (a) y la ambigüedad cuando hay dos clases presentes(b). Una solución: Realizar detección de objetos (c).

a nivel de píxeles de la imagen, problema denominado como *segmentación*. Una que es muy utilizada es la del *bounding box,* que es un rectángulo con sus lados paralelos a los bordes de la imagen. De esta manera, una respuesta al problema de detección de objetos involucra dar una lista de *bounding boxes* con sus respectivas clasificaciones.

3.4.2. Una primera aproximación

Una de las primeras aplicaciones de CNNs al problema de detección de objetos resultó en la arquitectura R-CNN[62] (en inglés, *Regions with* CNNs), que obtuvo una mejora de cerca del 30 % respecto a los métodos anteriores en la competencia de detección PASCAL VOC 2012[62]. Propone un método de tres etapas (ilustradas en 3.4.2) :

- Extraer posibles regiones de objetos usando un algoritmo específico (el más popular : Selective Search[63]).
- Extraer *features* de cada región usando una CNN.
- Clasificar cada región con SVMs (en inglés, *Support Vector Machines*).

Aunque logró grandes mejoras en el momento de su publicación, este enfoque tiene varios problemas. En primer lugar, es necesario realizar el entrenamiento de cada etapa de manera separada, lo que complica el *pipeline* de entrenamiento. Además, hay que aplicar la extracción de *features* vía CNN a cada región propuesta para cada imagen del dataset de entrenamiento,

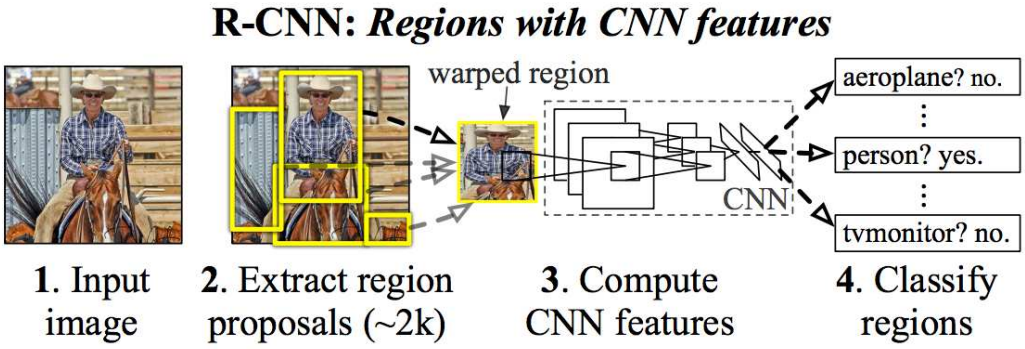


Figura 3.4.2: Arquitectura de R-CNN. Imagen obtenida de [62].

lo que es muy costoso computacionalmente y también en términos de espacio de disco, al momento de entrenar.

3.4.3. La capa de pooling ROI

El método anterior evolucionó rápidamente en Fast R-CNN [64], haciendo mayor uso de las redes profundas. Una de las innovaciones más importantes fue el desarrollo de la capa de pooling ROI (del inglés, *Region of Interest Pooling*). De manera similar a R-CNN, Fast R-CNN usa Selective Search para generar las propuestas de objetos, pero en lugar de procesar cada región de la imagen con una CNN, aplica una sola vez la CNN a la imagen completa (obteniendo un *feature map* de toda la imagen) y luego la capa de pooling ROI se encarga de tomar las regiones propuestas por Selective Search y sólo propagar hacia adelante las regiones de interés del *feature map* sin tener que volver a aplicar la CNN, ilustrado en la Figura 3.4.3.

La incorporación de la capa ROI a la arquitectura logró una mejora de velocidad sustancial, ya que la CNN sólo necesita ser aplicada una vez en lugar de una por cada región propuesta (aproximadamente 2000 veces). Además, la arquitectura unifica la extracción de *features*, clasificación y regresión del *bounding box* en una misma red (ver Figura 3.4.3), lo que permite que el modelo sea diferenciable *end-to-end* y más simple de entrenar. El principal cuello de botella ahora pasa a ser el algoritmo separado de propuesta de objetos (Selective Search).

3.4.4. Faster R-CNN

A mediados del 2015, un equipo de Microsoft Research integrado por Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun, encontró una manera de que el paso de proponer regiones de objetos tenga un costo casi nulo, a través de la arquitectura que llamaron Faster R-CNN[19].

La mejora surgió de la observación que la propuesta de objetos depende de *features* de la imagen ya calculados en la propagación hacia adelante de la CNN (en el primer paso de clasificación). Entonces resulta mucho más eficiente reutilizar esos *features* ya calculados para predecir regiones de objetos que utilizar un algoritmo separado como Selective Search.

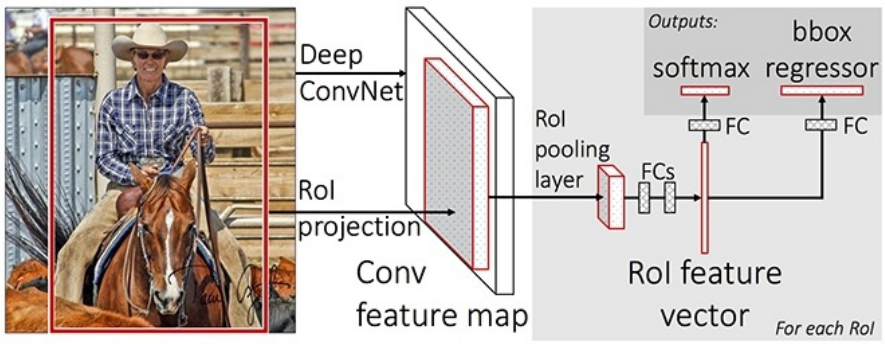
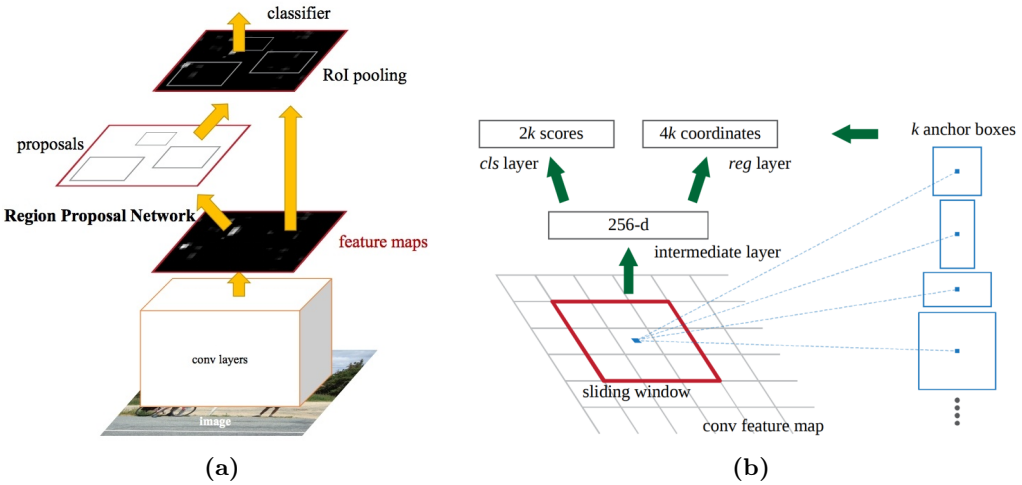


Figura 3.4.3: Arquitectura Fast R-CNN. Imagen obtenida de [64].



(a) **(b)**

Figura 3.4.4: La arquitectura de Faster R-CNN, con detalles de la nueva capa RPN(b). Imágenes obtenidas de [19].

Faster R-CNN agrega a Fast R-CNN una Region Proposal Network (RPN), una red encargada de tomar como entrada los *feature maps* obtenidos de la CNN, y dar como salida las regiones donde puede haber objetos, reemplazando a Selective Search, y obteniendo un modelo completamente entrenable *end-to-end*.

En la Figura 3.4.4a se puede apreciar como la RPN se incorpora en la arquitectura de Faster R-CNN.

La RPN

Para resolver el problema de la proposición de regiones de objetos, la RPN analiza un número predefinido de regiones que pueden llegar a contener un objeto. Esto se implementa mediante *anchor boxes*, un conjunto de *bounding boxes* de dimensiones fijas que están, conceptualmente, distribuidos uniformemente en la imagen original. Para cada uno de estos *anchor boxes* la RPN debe evaluar la probabilidad que contenga un objeto relevante, y cómo se ajustaría ese *anchor box* para encuadrar mejor al objeto.

Como se ilustra en la Figura 3.4.4b, la RPN está implementada eficientemente como una convolución sobre el *feature map* de la imagen, que funciona como una "ventana deslizante, procesando en simultáneo todos los *anchor boxes* para esa posición (obteniendo una salida de 256 o 512 canales). Luego se aplican dos convoluciones en paralelo, una para obtener los puntajes de objeto para cada *anchor box* y otra para ajustar los *bounding boxes*. Si se utilizan *k anchor boxes*, una salida tendrá *2k* canales, indicando para cada *anchor box* la probabilidad de ser un objeto y la de no ser un objeto. La otra salida tendrá *4k* canales, codificando el ajuste a hacerse a cada *anchor box* con cuatro números: $(\Delta_{xcenter}, \Delta_{ycenter}, \Delta_{width}, \Delta_{height})$.

De esta manera se obtienen las regiones con posibles objetos sobre las cuales se aplica ROI *Pooling* para extraer los *features* relevantes a esa región. Con estos *features* se procede de la misma manera que en Fast R-CNN: se clasifica el contenido del *bounding box* (o se lo descarta clasificándolo como la clase "fondo"), y se hace regresión sobre las dimensiones del *bounding box* para que se ajuste mejor al objeto.

VGG16: La arquitectura base

La arquitectura base para Faster R-CNN utilizada en esta tesina es VGG16, presentada en [60] para resolver el problema de clasificación de ImageNet[65].

Como se muestra en la Figura 3.4.5, la arquitectura cuenta con cinco etapas convolucionales (con varias capas convolucionales y de regularización internas) cada una finalizada con una capa de *pooling*, y tres capas *fully connected* al final. Estas cinco capas forman la CNN que obtiene el *feature map* de la imagen que luego procesa la RPN y la capa de *pooling* ROI.

Cuando se entrena Faster R-CNN, esta red se inicializa con los pesos obtenidos de un entrenamiento previo con el dataset ImageNet[65].

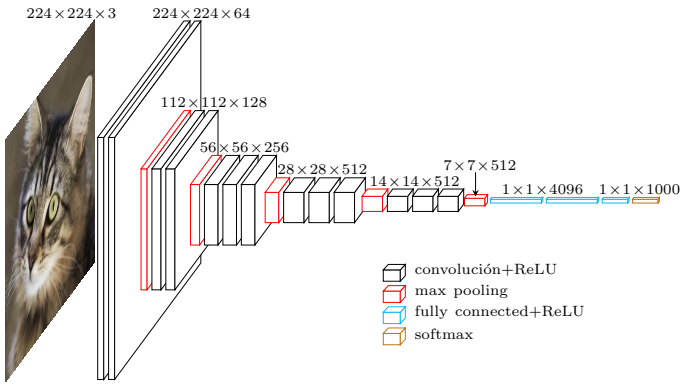


Figura 3.4.5: Arquitectura de VGG16. Las figuras ilustradas representan como se van transformando los datos después de la aplicación de cada capa.

Capítulo 4

Método propuesto

En este capítulo se presenta el método propuesto de mapeo de objetos basado en S-PTAM [18]. Este método utiliza como entrada imágenes estéreo, y de manera *online*, construye un mapa con información semántica y espacial de objetos. Para cada objeto en el mapa, la información semántica se corresponde con su clasificación obtenida de la red neuronal (por ejemplo, silla, mesa,etc...), mientras que la información espacial consiste en un *bounding cube* que aproxima su volumen, posición y orientación en el mapa.

El método se puede dividir en dos tareas principales: la detección de objetos en las imágenes por medio de la CNN, y el procesamiento de esas detecciones en S-PTAM.

4.1. Detección de objetos

Como punto de partida, se utiliza la red de detección de objetos Faster R-CNN [19] (ver Sección 3.4.4), con implementación en *Pycaffe* [22] de código abierto, y apta para hacer detecciones online (con GPU).

El primer aporte realizado en este trabajo fue adaptar Faster R-CNN a la detección de un conjunto nuevo de objetos. Para esto se modificaron las últimas capas de la arquitectura, que dependen de la cantidad de clases con las que se desea trabajar. Durante el desarrollo de la tesina, se decidió también modificar la CNN para obtener la orientación y dimensiones de cada objeto detectado (además de su *bounding box* y clase). Esto resultó en una extensión de la arquitectura y la modificación del código fuente de varias capas para lograrlo.

4.1.1. Extensión de Faster R-CNN para dar la pose del objeto

Como se mencionó en la Sección 3.4.1, esta red tiene como salida *bounding boxes* que permiten ubicar un objeto en la imagen, pero carece de información 3D que permita estimar la posición del objeto en el mundo real. En este trabajo, se exploró la posibilidad de utilizar la CNN para obtener estimaciones de los objetos en el espacio.

Por esta razón se extendió la red con nuevas capas, como se presenta en [66], para poder calcular el *bounding cube* de cada detección. El *bounding cube* se define como el menor para-

Figura 4.1.1: Proyección del *bounding cube* en la imagen (en azul), y vértices que determinan el *bounding box* (en rojo).

lelepípedo rectangular (vulgarmente, *caja*) que contiene al objeto. Este agregado de nuevas capas le permite a la red estimar la orientación y las dimensiones del objeto. Intuitivamente, se puede ver que la orientación del objeto es un factor muy influyente en el *bounding cube*, y ademas está relacionada con su aspecto visual, indicando que se podría estimar usando CNNs. Respecto a las dimensiones, muchos objetos de una misma clase, o subclase, comparten dimensiones similares por lo que está ligado a un problema de clasificación.

A continuación se detalla cómo la regresión de estos parámetros se utiliza para calcular el *bounding cube* de una detección [66].

Relación entre *bounding box* y *bounding cube*

Como se ilustra en la Figura 4.1.1, si se proyecta el *bounding cube* sobre el plano de la imagen, debe encajar perfectamente en los bordes del *bounding box* del objeto (si no fuese así, el *bounding box* no estaría ajustado al objeto). Más precisamente, cada lado del *bounding box* debe contener la proyección de (al menos) un vértice del *bounding cube* del objeto. Esta propiedad será utilizada para poder obtener una localización en el espacio del mismo a partir de una detección en el plano de la imagen.

El *bounding cube* está determinado por su centro $\mathbf{t} = [t_x, t_y, t_z]^T$, dimensiones $\mathbf{d} = [d_x, d_y, d_z]$ y orientación $\mathbf{R} \in \mathbf{SO}(3)$. Dada la pose del objeto en el marco de coordenadas de la cámara $(\mathbf{R}, \mathbf{t}) \in \mathbf{SE}(3)$ y la matriz intrínseca $\mathbf{K}$ (ver Sección 3.1), la proyección de un punto 3D en coordenadas homogéneas $\dot{\mathbf{x}}_o = [X, Y, Z, 1]^T$ en el marco de coordenadas del objeto en la imagen $\dot{\mathbf{x}} = [x, y, 1]^T$ es

$$\dot{\mathbf{x}} = \mathbf{K}\,[\mathbf{R}\,\mathbf{t}]\,\dot{\mathbf{x}}_o.$$

Asumiendo que el origen del sistema de coordenadas del objeto coincide con el centro del *bounding cube* y las dimensiones $\mathbf{D}$ son conocidas, los vértices del *bounding cube* son

$$\mathbf{v}_1 = [d_x/2, d_y/2, d_z/2]^T,$$

$$\mathbf{v}_2 = [-d_x/2, d_y/2, d_z/2]^T ,$$

$$\vdots$$

$$\mathbf{v}_8 = [-d_x/2, -d_y/2, -d_z/2]^T .$$

La restricción de que el *bounding cube* esté encuadrado por el *bounding box*, implica que cada lado de la detección 2D tiene que coincidir con la proyección de (al menos) uno de los vértices del *bounding cube*. Si el *bounding box* está dado por los vértices inferior izquierdo (x_{min}, y_{min}) y superior derecho (x_{max}, y_{max}), y además se conoce que, por ejemplo, el punto 3D $\mathbf{v}_2$ proyectado cae sobre el lado izquierdo de la detección, se puede plantear la siguiente ecuación:

$$x_{min} = \left(\mathbf{K} \ [\mathbf{R}\,\mathbf{t}] \begin{bmatrix} -d_x/2 \\ d_y/2 \\ d_z/2 \\ 1 \end{bmatrix} \right)_x , \tag{4.1.1}$$

donde $(.)_x$ indica la componente x de la proyección perspectiva. Ecuaciones análogas se pueden plantear para $x_{max}, y_{min}, y_{max}$ si se sabe la correspondencia de qué vértice del *bounding cube* se proyecta a que a lado del *bounding box*:

$$x_{max} = \left(\mathbf{K} \ [\mathbf{R}\,\mathbf{t}] \begin{bmatrix} \mathbf{v}_q \\ 1 \end{bmatrix} \right)_x , \qquad y_{min} = \left(\mathbf{K} \ [\mathbf{R}\,\mathbf{t}] \begin{bmatrix} \mathbf{v}_s \\ 1 \end{bmatrix} \right)_y , \qquad x_{max} = \left(\mathbf{K} \ [\mathbf{R}\,\mathbf{t}] \begin{bmatrix} \mathbf{v}_t \\ 1 \end{bmatrix} \right)_y ,$$

donde $(.)_y$ indica la componente y de la proyección perspectiva, y $q, s, t \in \{1 \dots 8\}$ según que vértice corresponda. En total, estas cuatro restricciones alcanzan para determinar $\mathbf{t}$, si se saben $\mathbf{K}$, $\mathbf{R}$ y $\mathbf{d}$. $\mathbf{K}$ es una constante que se conoce por ser la matriz de calibración intrínseca de la cámara (ver Sección 3.1), mientras que $\mathbf{R}$ y $\mathbf{d}$ serán parámetros estimados por la red neuronal.

Estimando R y d

La arquitectura Faster R-CNN aplica una capa *pooling* ROI para propagar hacia adelante los *feature maps* de cada región de objeto (explicado en 3.4.3). Esto quiere decir que para cualquier capa de regresión o clasificación se contará con los datos obtenidos de la subregión de imagen siendo procesada en ese momento. En otras palabras, la salida de la CNN para una sub-región es invariante respecto a su traslación en la imagen. Por esta razón, se decide no estimar directamente la orientación $\mathbf{R} \in \mathbf{SO}(3)$ del objeto respecto a la cámara, ya que en las capas superiores, la CNN pierde la información de donde se encuentra el objeto en la imagen. En cambio, la red estima un ángulo local que es invariante a la traslación del objeto en la imagen.

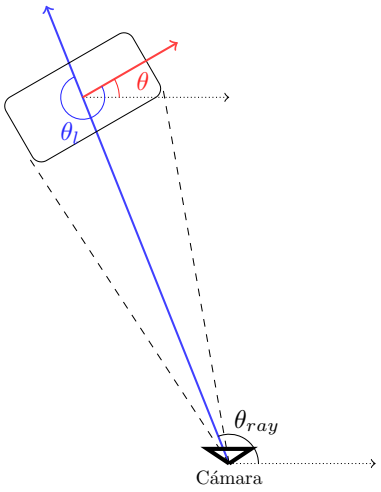


Figura 4.1.2: Ejemplo visual del ángulo local θ_l definido en base a la diferencia entre la orientación absoluta (en rojo) θ y ángulo del rayo θ_{ray}. El rayo (en azul) va desde la cámara hacia el centro del *bounding box* proyectado en la imagen (no necesariamente el centro del objeto).

En este trabajo, la matriz de rotación $\mathbf{R}(\theta)$ está solo parametrizada por una rotación respecto al eje vertical con ángulo θ. Este ángulo se puede descomponer en

$$\theta = \theta_{ray} + \theta_l, \tag{4.1.2}$$

donde θ_l es el ángulo de rotación local que forma el objeto con el rayo que va desde la cámara hacia el centro del *bounding box* y θ_{ray} el ángulo que forma este rayo con la cámara (ver Figura 4.1.2). En lugar de predecir el ángulo θ directamente, la red hace una regresión sobre el ángulo de rotación local θ_l, y se obtiene el ángulo θ al combinar los ángulos θ_l y θ_{ray} según la Ecuación 4.1.2 (el ángulo θ_{ray} se puede calcular a partir la ubicación del *bounding box* en la imagen).

Para estimar el parámetro θ_l con la red, se utiliza el mismo enfoque que tiene Faster R-CNN para hacer regresión sobre los *bounding boxes*, que es discretizar el espacio de soluciones en m clases, cada una asociada a un intervalo angular. Para cada una de estas clases la red estima la probabilidad que θ_l pertenezca a ella y el ángulo residual que falta aplicarle al ángulo central de esa clase para obtener la rotación local. En la práctica, se estima el valor del seno y coseno de este angulo residual, teniendo la red 3 salidas para cada clase angular i: $(c_i, \cos(\Delta\theta_i), \sin(\Delta\theta_i))$. La función de costo se define como

$$L_\theta = L_{conf} + w \times L_{loc}.$$

La confianza L_{conf} es el costo *softmax* de las confianzas de cada clase angular. L_{loc} es la función de costo que trata de minimizar la diferencia entre el ángulo estimado y el ángulo del *ground-truth,* para cada uno de los intervalos (pueden tener solapamiento) a los que pertenece el ángulo del *ground-truth*. Minimizar esta diferencia es equivalente a maximizar su coseno, por lo que

$$L_{loc} = -\frac{1}{m_\theta}\sum \cos\left(\theta_{gt} - c_i - \Delta\theta_i\right),$$

donde θ_{gt} es el ángulo del *ground-truth*, m_θ es la cantidad intervalos que contienen a θ_{gt}, c_i es el ángulo central al intervalo i, y $\Delta\theta_i$ el ángulo residual que debe aplicarse al centro del intervalo i.

Para realizar la estimación de $\mathbf{d} = [d_x, d_y, d_z]$ se podría utilizar un enfoque discreto/continuo como en el caso anterior, pero se decidió hacer una regresión sin discretizar el espacio de soluciones. En la mayoría de las clases, la distribución de sus posibles dimensiones tiene baja varianza, así que se procedió con la técnica tradicional de hacer regresión sobre el residual respecto a la media de los valores de entrenamiento para cada parámetro, con función de costo

$$L_{dim} = \frac{1}{n}\sum \left(d_{gt} - \bar{d} - \delta\right)^2,$$

donde d_{gt} es el *ground-truth* de las dimensiones, $\overline{d}$ son las dimensiones medias para objetos de una determinada categoría y δ es el residual respecto a la media que predice la red. La función de costo total queda definida como

$$L = L_\theta + w' \times L_{dim}.$$

Correspondencia de vértice a lado

Usando las estimaciones de la red de la rotación $\mathbf{R} \in \mathbf{SO}\,(3)$ y dimensiones $\mathbf{d} \in \mathbb{R}^3$, se puede despejar la traslación $\mathbf{t} \in \mathbb{R}^3$ de las ecuaciones 4.1.1 , y calcularlo para cada asignación de lado de *bounding box* a un vértice del *bounding cube*. Después de probar con todas las combinaciones, se elige el $\mathbf{t}$ que minimice el error de reproyección del *bounding box*. En un principio, cada lado puede corresponder a un vértice, lo que resulta en $8^4 = 4096$ configuraciones. Para reducir el tiempo de cómputo, se asume que la cámara se encuentra paralela al suelo y que el objeto siempre se encuentra parado, rotado sobre su eje vertical, pero sin otras inclinaciones, como es el caso de una gran cantidad de objetos cotidianos (por ejemplo, muebles). Como el objeto sólo está rotado respecto al eje Z, y_{min} es determinado por alguno de los vértices de inferiores y y_{max} por alguno de los superiores. Además, se puede suponer que tanto x_{min} como x_{max} quedan determinados por dos de los vértices inferiores (porque todos los puntos de un lado vertical se proyectan a la misma coordenada x de imagen). Por lo tanto, sólo es necesario evaluar $4^4 = 256$ posibilidades para obtener la que minimice el error de reproyección del *bounding box*.

Arquitectura final

En la Figura 4.1.3 se muestra el esquema general de la arquitectura de Faster R-CNN modificada para obtener información de pose. Como se mencionó en la Sección 3.4.4 la arquitectura está basada en VGG16 [67] por lo que **Conv.Net** resume sus cinco etapas convolucionales, y las capas notadas **fc** (en la Figura 4.1.3) incluyen además de una capa *fully connected*, una capa *ReLU* y una de *dropout*. Las nuevas capas usadas en la regresión de pose, siguen este mismo esquema (*fully connected+ReLU+dropout*).

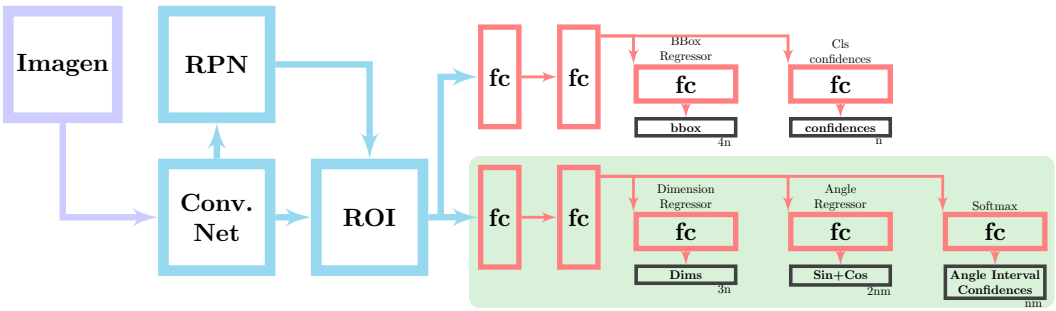


Figura 4.1.3: Arquitectura Faster R-CNN modificada (nuevas capas resaltadas). Función de las capas originales detallada en la Sección 3.4.

4.1.2. Dataset Sintético

Para entrenar la red, es necesario tener un dataset de imágenes anotadas no sólo con *bounding boxes* de los objetos presentes, sino también con la pose 3D de los objetos. Para esto se desarrolló un dataset sintético, del cual se puede obtener el *ground-truth* necesario: para cada objeto en la imagen su clase, *bounding box*, orientación y dimensiones. Las imágenes y anotaciones fueron generadas utilizando la plataforma de simulación V-REP [68]. V-REP es un software de simulación de robots que en este caso permitió emular cámaras y generar imágenes a partir de una escena 3D. Los modelos de los objetos fueron obtenidos del dataset ModelNet [69], que incluye alrededor de 5000 modelos 3D CAD de 10 clases de objetos comunes: cama, bañera, silla, escritorio, cajonera, mesita de luz, monitor, sofá, mesa e inodoro. También se utilizaron modelos de habitaciones de SceneNet [70], y texturas para darle una semblanza realista a las imágenes.

Estos modelos 3D se encuentran consistentemente alineados según su clase (por ejemplo, para todas las sillas su eje Y apunta siempre hacia el respaldar), pero en algunos casos difieren en la unidad de distancia, por esto fue necesario aplicar un pre-procesamiento para llevar a todos los modelos a una misma escala y descartar los que no utilizaban una unidad común como mm, m o pulgadas.

Para generar cada escena 3D se ubican objetos al azar dentro de una región. A la distribución de objetos en el espacio se le aplican ciertas restricciones para que se asemeje a la realidad, como por ejemplo, que no haya colisión entre ellos, que su orientación vertical sea la típica, y que la mayoría (todos, salvo los monitores) se encuentre sobre un mismo piso. Además, se trata de emular la co-ocurrencia de clases de objetos que se da en la vida real, en el contexto de distintas habitaciones, como baño, dormitorio y oficina, por ejemplo.

El centro de la escena se calcula en base a un promedio ponderado (aleatoriamente) de la posición de los objetos y la orientación de la cámara virtual es hacia este punto. La posición de la cámara se elige de manera aleatoria, controlando que los distintos puntos de vista a los objetos sea lo más uniforme posible. Esto se implementó utilizando coordenadas polares para la posición de la cámara, y eligiendo el radio uniformemente distribuido en cierto intervalo, el ángulo de azimut θ uniformemente distribuido en $[0, 2\pi)$, y el ángulo polar φ elegido de una distribución *ad hoc* mayormente uniforme que evita los extremos del intervalo $\left(-\frac{\pi}{2}, \frac{\pi}{2}\right)$. El intervalo válido del radio depende del "diámetro" de la escena de objetos, y de restricciones impuestas por la habitación, para que la cámara permanezca dentro de ella.

Figura 4.1.4: Imágenes tomadas al azar del dataset artificial desarrollado en este trabajo.

También fue necesario tener un control de cuáles objetos se encuentran ocluidos en la imagen final, para no incluirlos en el *ground-truth*. Esto se logró examinando las intersecciones entre los *bounding boxes* de los objetos proyectados en la imagen, y descartando aquellas que sobrepasen un umbral y se encuentren detrás de otro objeto.

Como el algoritmo de generación de escenas fue desarrollado de manera *ad hoc*, requiere el ajuste de varias constantes (limites superiores e inferiores de variables aleatorias) para lograr imágenes que se correspondan con las de un dataset real.

Para generar las imágenes de entrenamiento y de test, se utilizaron conjuntos disjuntos de modelos de 3D de ModelNet. En el conjunto de entrenamiento se incluyeron aproximadamente 15000 imágenes y en el de test aproximadamente 6000, buscando que las ocurrencias de cada clase de objeto sea mayormente uniforme.

Entrenamiento

La arquitectura Faster R-CNN está implementada en *Pycaffe*, por lo que se utilizo el optimizador de Descenso por gradiente estocástico (SGD del inglés *Stochastic gradient descent*) implementado en Caffe [22], con los pesos iniciales de la red Faster R-CNN ya entrenada en el dataset COCO [71].

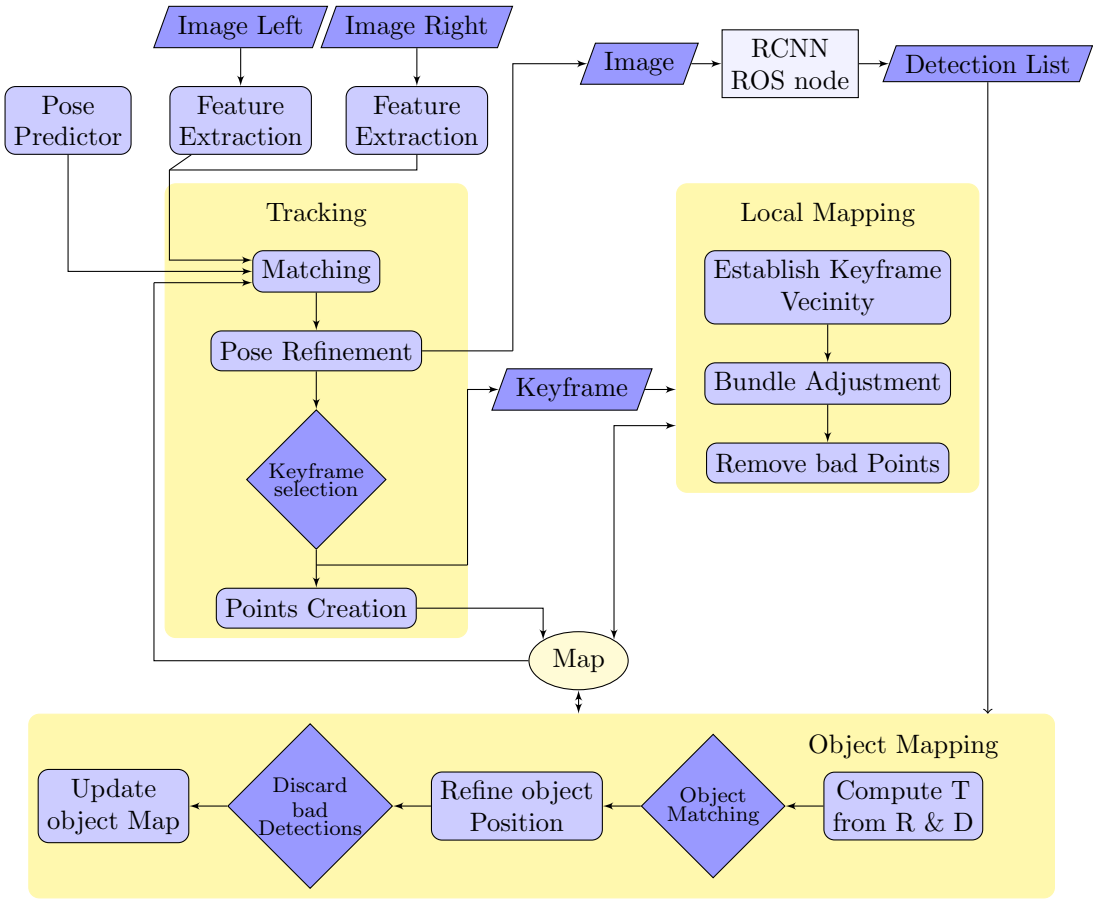


Figura 4.2.1: Esquema S-PTAM modificado para realizar mapeo de Objetos.

El esquema de entrenamiento consistió en un primer entrenamiento preliminar con imágenes reales (Train COCO 2014 [72]) en las clases en común con las de ModelNet, seguido por un entrenamiento con imágenes sintéticas de todas las capas *fully connected* y de la capa RPN, para finalizar con un entrenamiento más largo de las capas dedicadas a la predicción de pose con el dataset sintético.

4.2. S-PTAM + Detección de objetos

En esta sección se describe cómo se integra la información obtenida a partir del detector de objetos a S-PTAM, detallando los componentes y procesos necesarios para dotar a S-PTAM de un mapa de objetos.

4.2.1. Esquema general

En la Figura 4.2.1 se muestra la estructura general de S-PTAM con el módulo de mapeo de objetos.

Una vez que el hilo de *tracking* estima la pose actual para un par de imágenes estéreo, se

asocian esas imágenes con un identificador de pose de cámara y se envía junto con la imagen (izquierda) al módulo de detección de objetos. Este módulo encapsula la red neuronal que proporciona las detecciones de objetos y está implementado como un nodo ROS [20], que simplifica la comunicación con S-PTAM.

El módulo de detección de objetos procesa con la red neuronal la imagen para obtener una lista de los objetos presentes. Luego, envía estas detecciones a S-PTAM, junto con el identificador de pose recibido con la imagen. Este identificador hace referencia a la pose de la cámara de donde se obtuvo la imagen, cuya representación es relativa al *keyframe* más cercano. Esto permite que si S-PTAM refina la pose de dicho *keyframe*, refinará también la pose de la cual se obtuvo la detección de objetos. El módulo de detección fue configurado para que pierda mensajes si se encuentra procesando, de esta manera priorizando la última imagen procesada por S-PTAM. En las pruebas realizadas, se realiza la detección de objetos a una velocidad de 5 hz.

Una vez obtenidas las detecciones, se procesa cada una individualmente, estimando la posición del objeto respecto a la cámara a partir de los datos proporcionados por la red. Luego se pasa a la etapa de asociación de datos, donde se busca si el objeto medido ya pertenece al mapa o si es uno nuevo. Para cada imagen, se proyectan los objetos observables del mapa y se contabiliza si fueron detectados o no. Esta es información muy útil para determinar la confianza de cada objeto como se verá en la Sección 4.2.6.

Una vez establecida la correspondencia de objetos, se puede refinar la posición del mismo usando información del mapa de puntos de S-PTAM. Este refinamiento también permite descartar detecciones que no son lo suficientemente buenas.

Finalmente, se establece la pose del objeto en el marco de referencia de un *keyframe* como una observación del objeto y se fusionan todas las observaciones para obtener el mapa actual.

4.2.2. Estimación de Pose 3D de los objetos

En esta sección se detalla cómo se obtiene la rotación y traslación $\mathbf{R}$ y $\mathbf{t}$ del objeto respecto a la cámara, a partir de la salida de la red (*bounding box*, ángulo de orientación local, y dimensiones, ver Sección 4.1.1). La matriz de rotación $\mathbf{R}$ queda determinada a partir del ángulo de orientación respecto a la cámara, obtenido de la suma del ángulo del rayo del *bounding box* y el ángulo local. Si se desarrolla la ecuación (4.1.1), se obtiene que para un punto $\mathbf{b} = [\mathrm{b}_1, \mathrm{b}_2, \mathrm{b}_3]$ dado, la proyección al plano de la imagen en coordenadas homogéneas es:

$$\mathbf{K} \times \begin{bmatrix} \mathbf{R} & | \, \mathbf{t} \end{bmatrix} \times \begin{bmatrix} \mathbf{b} \\ 1 \end{bmatrix} = \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} \mathbf{r}_1 & | \, t_1 \\ \mathbf{r}_2 & | \, t_2 \\ \mathbf{r}_3 & | \, t_3 \end{bmatrix} \times \begin{bmatrix} \mathbf{b} \\ 1 \end{bmatrix} = \begin{bmatrix} f\,(\mathbf{r}_1\mathbf{b} + t_1) \\ f\,(\mathbf{r}_2\mathbf{b} + t_2) \\ \mathbf{r}_3\mathbf{b} + t_3 \end{bmatrix}.$$

Si $\mathbf{w_{min}}$, $\mathbf{w_{max}}$, $\mathbf{h_{min}}$ y $\mathbf{h_{max}}$ representan los vértices que determinan x_{min}, x_{max}, y_{min} y y_{max} respectivamente:

$$x_{min} = \frac{f\,(\mathbf{r}_1\mathbf{w}_{min} + t_1)}{\mathbf{r}_3\mathbf{w}_{min} + t_3}\ ,\ x_{max} = \frac{f\,(\mathbf{r}_1\mathbf{w}_{max} + t_1)}{\mathbf{r}_3\mathbf{w}_{max} + t_3}\ ,\ y_{min} = \frac{f\,(\mathbf{r}_2\mathbf{h}_{min} + t_2)}{\mathbf{r}_3\mathbf{h}_{min} + t_3} \text{ y } y_{max} = \frac{f\,(\mathbf{r}_2\mathbf{h}_{max} + t_2)}{\mathbf{r}_3\mathbf{h}_{max} + t_3}. \tag{4.2.1}$$

Esto nos permite despejar t_3 de dos maneras, para obtenerlo en base al ancho o al alto del *bounding box*, para luego calcular t_1 y t_2:

$$t_3 = \frac{f\,\mathbf{r}_1\,(\mathbf{w}_{max} - \mathbf{w}_{min}) - \mathbf{r}_3\,(\mathbf{w}_{max}x_{max} - \mathbf{w}_{min}x_{min})}{x_{max} - x_{min}},$$

o

$$t_3 = \frac{f\,\mathbf{r}_2\,(\mathbf{h}_{max} - \mathbf{h}_{min}) - \mathbf{r}_3\,(\mathbf{h}_{max}y_{max} - \mathbf{h}_{min}y_{min})}{y_{max} - y_{min}}.$$

4.2.3. Correspondencia de objetos

Establecer una buena asociación de objetos es fundamental para poder acumular mediciones de un mismo objeto. Para lograrlo, se compara la detección obtenida con la proyección de cada objeto del mapa (ubicado en el campo visual de la cámara). Una manera de medir que tanto coincide una nueva observación con la medición esperada del objeto en el mapa es reproyectar el *bounding box* del objeto en la imagen, y calcular el IoU (*Intersection over Union*, ver Figura 5.1.3a) con el *bounding box* de la nueva detección. El objeto que maximice esta medida de solapamiento será el principal candidato a ser el correspondiente a la medición. Si además se posee información 3D de la medición, también se puede calcular la distancia $\mathtt{L}_2$ entre los centroides de los objetos para resolver algunas ambigüedades. Por supuesto, si ningún objeto supera cierto umbral de IoU, se considera que la detección se hizo sobre un nuevo objeto a agregar al mapa.

4.2.4. Mejora de estimación de profundidad

La traslación $\mathbf{t}$ calculada tiene considerable error, ya que acumula muchas aproximaciones, especialmente en respecto al *bounding box*, y $\mathbf{d}$ donde pequeños cambios producen grandes cambios en t_3. Por eso, es posible utilizar la información mapeada por S-PTAM para refinar la traslación $\mathbf{t}$ y así obtener una estimación más precisa de donde se encuentra el objeto.

En el mapa mantenido por S-PTAM los *map points* identifican a punto 3D cuya posición se conoce. Si se lograra determinar qué *map points* pertenecen al objeto, entonces se podría corregir la posición estimada, en particular la profundidad.

Sin utilizar una primera aproximación de la posición del objeto, en este trabajo se trató de establecer esta asociación *map point*-objeto sin lograr una buena precisión. El mayor problema radicó en que un *bounding box* no es una representación lo suficientemente fina del objeto para esta tarea. Es decir, aún si un *map point* se proyecta dentro del *bounding box*, podría no pertenecer al objeto y no se dispone de otra información para identificar este caso. Además, la distribución de los *map points* no es homogénea, se concentran en lugares con textura y es posible que la mayoría de los puntos en el *bounding box* se encuentren fuera del objeto.

Al tener una aproximación de la posición del objeto en el espacio, se pueden descartar puntos que no se encuentren en el lugar esperado y así minimizar las asociaciones erróneas. Este filtrado de puntos también permite discernir detecciones falsas en la red neuronal, ya que

si no se encuentra una mínima cantidad de *map points* en la vecindad del objeto, se descarta la medición.

Aun así, después del filtrado de puntos, se debe recurrir a alguna heurística para asociar *map points* a objetos. En este trabajo, se obtuvieron buenos resultados asignando sólo el punto más cercano (norma $\mathtt{L_2}$) al centro del *bounding box*, una vez proyectados sobre la imagen. Una vez asociado un *map point* a una medición, se ajusta la traslación $\mathbf{t}$ para que describa a un punto sobre el mismo rayo con origen en la cámara, pero a la misma distancia que el *map point*.

4.2.5. Fusión de las observaciones de objetos

De esta manera, si $\boldsymbol{O} = \{O^t\}$ es el conjunto de objetos en el mapa, para cada objeto O^t se considera el conjunto de sus observaciones $\boldsymbol{q}^t = \{q_i^t\}$. Cada una de estas observaciones q_i^t está parametrizada por una posición 3D (x_i^t, y_i^t, z_i^t), una orientación dada por un ángulo θ_i^t, las dimensiones del *bounding cube* $\left(dx_i^t, dy_i^t, dz_i^t\right)$ y la etiqueta de la categoría del objeto c_i^t (por ejemplo, silla, monitor, etc...). Las pose ((x_i^t, y_i^t, z_i^t) y θ_i^t) es representada relativa al *keyframe* más cercano del cual se obtuvo la observación, por lo que sus valores se actualizan cuando se produce una corrección en la pose del *keyframe*. Para fusionar estos datos y localizar el objeto en el mapa, se calcula la mediana de cada uno de estos parámetros. Es decir, el objeto O^t queda determinado por su posición $\left(X^t, Y^t, Z^t\right)$ donde

$$X^t = mediana(x_i^t),\ Y^t = mediana(y_i^t),\ Z^t = mediana(z_i^t),$$

y así también para sus dimensiones (DX^t, DY^t, DZ^t) donde

$$DX^t = mediana(dx_i^t),\ DY^t = mediana(dy_i^t),\ DZ^t = mediana(dz_i^t).$$

En el caso de la clase del objeto C^t, que se calcula realizando un histograma de las clases observadas c_i^t y determinando la de mayor ocurrencia. Se decidió fusionar los datos de ésta manera para mitigar el impacto de mediciones con mucho error. En el caso del ángulo θ^t, que es una medida circular donde la mediana no está definida, primero se particiona el intervalo $[0, 2\pi)$ en n (en nuestro caso $n = 8$) secciones iguales, y se calcula un histograma de los ángulos observados θ_i^t. Al determinar el intervalo más poblado, se toma la mediana entre el conjunto las observaciones de ese intervalo y sus vecinos.

4.2.6. Confianza de los objetos

Para cada imagen enviada al detector de objetos se consideran los objetos del mapa que deberían ser observados, proyectando sus *bounding cubes* sobre la imagen. De estos objetos, los que fueron efectivamente detectados en la imagen son considerados *inliers* y los que no *outliers*. A lo largo de la ejecución para cada objeto del mapa se contabilizan sus detecciones positivas (en las que fue *inlier*) y sus no-detectecciones (en las que fue un *outlier*). En base a la diferencia de estas métricas, se establece un nivel de confianza en los objetos sobre el cual se definen tres umbrales: uno para objetos confiables, otro para objetos no confiables que pueden volverse confiables en el futuro, y otro para objetos no confiables que deben ser removidos del mapa.

Capítulo 5

Experimentación y resultados

En este capítulo se presentan los experimentos llevados a cabo para evaluar la performance de la red neuronal basada en Faster R-CNN (ver Sección 4.1.1) y del método de mapeo de objetos incorporado a S-PTAM [18]. En el caso de la red neuronal, se buscó medir la efectividad de sus detecciones en datasets reales después de utilizar datos sintéticos de entrenamiento. Por otro lado, se evaluó el error en las predicciones de orientación y dimensiones de objetos en el dataset sintético. Una vez incorporada la red neuronal a S-PTAM, para poder validar el sistema resultante se desarrolló un dataset obtenido de entornos de oficina donde se pudo probar el sistema en su totalidad.

El hardware utilizado para el procesamiento de los experimentos corresponde a una computadora estándar de escritorio con procesador Intel Core i7 de 4.0 Ghz y placa de video NVIDIA GTX 970 de 4 GB.

5.1. Evaluación de la red Neuronal

5.1.1. Datasets

Para la evaluación se utilizaron datasets de imágenes reales, y además el dataset de imágenes sintéticas presentado en la sección 4.1.2. Los datasets de imágenes reales son los que se usaron en el trabajo [19] para entrenar Faster R-CNN: PASCAL [73] y COCO [71]. Ambos datasets fueron concebidos para sus respectivos *challenges* [74, 72], competencias donde los participantes buscan obtener la mejor performance en el conjunto de test de estos datasets y cuyo *ground-truth* no se encuentra disponible al público.

Las imágenes del dataset PASCAL utilizadas se corresponden específicamente al conjunto de test de PASCAL VOC 2007 [74], de 4952 imágenes y 20 clases de objetos de categorías distintas como por ejemplo: avión, caballo, mesa y persona. Las imágenes fueron mayormente obtenidas de Flickr [75]. Algunas de estas, son presentadas en la Figura 5.1.1.

El dataset de COCO es el actual estado del arte en datasets de reconocimiento visual de objetos, utilizado para el problema de detección de objetos, segmentación de objetos, detección de *keypoints* de personas, y *captioning* de imágenes. A diferencia de PASCAL, sigue teniendo un desarrollo activo, incorporando nuevas imágenes y anotaciones. Anualmente se realizan

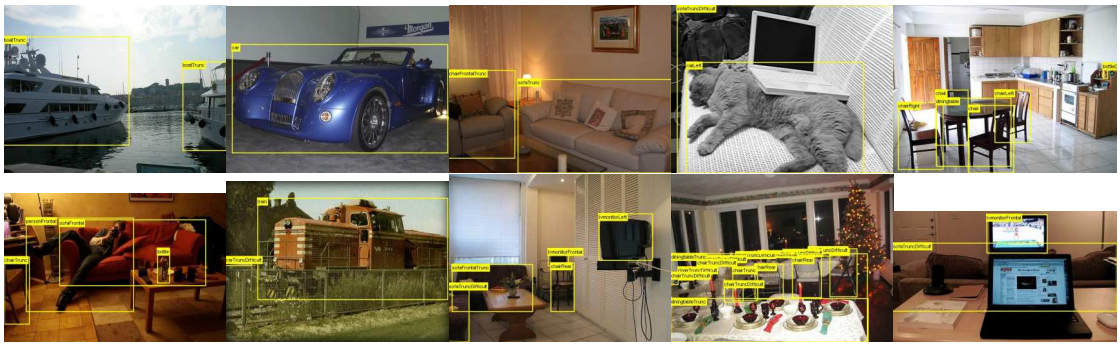


Figura 5.1.1: Imágenes de ejemplo del dataset PASCAL 2007[73], con su *ground-truth* graficado. Imágenes obtenidas de [74].

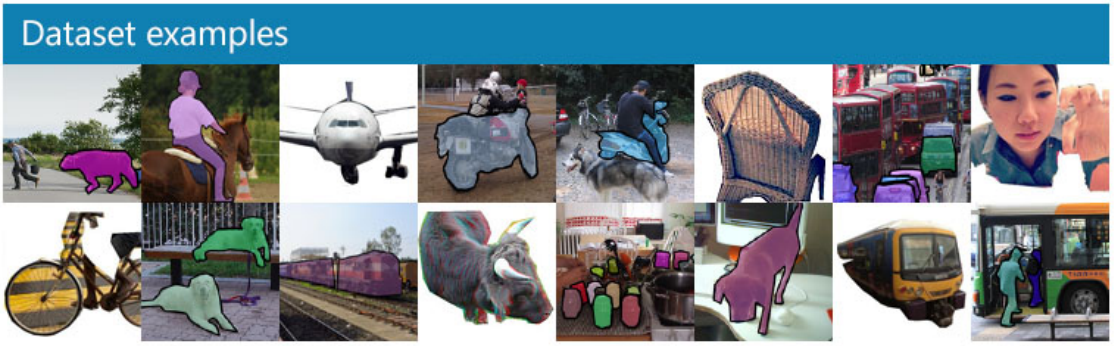


Figura 5.1.2: Recopilación de imágenes del dataset COCO con su *ground-truth* graficado, obtenida de [72].

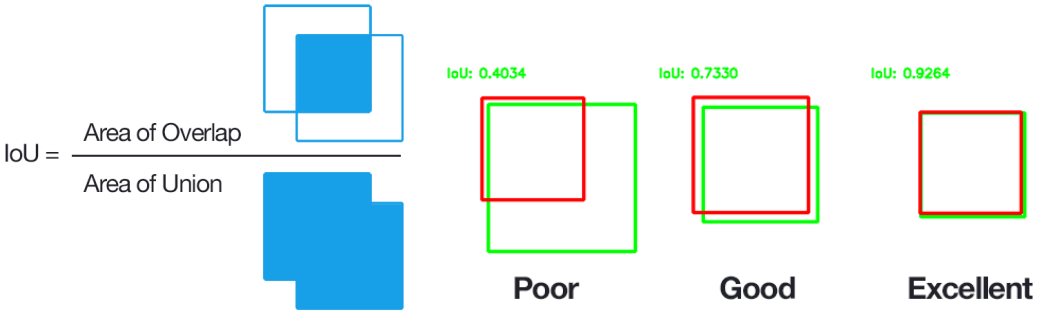
competencias sobre este dataset en los problemas mencionados, y tiene sponsors entre los que se encuentran Microsoft, Facebook, y Google.

Uno de los objetivos de COCO fue recopilar un dataset donde la mayoría de las imágenes sean "no canónicas". Típicas imágenes "canónicas" tienen un sólo objeto en el centro y se encuentran como resultado de la búsqueda en Google de una determinada categoría de objetos o escena. Por esta razón, la mayoría de las imágenes de COCO suelen contener varias clases de objetos. Su fuente principal de imágenes también fue Flikr. Algunos ejemplos se muestran en la Figura 5.1.2.

Las imágenes del dataset COCO utilizadas en este trabajo se corresponden a los conjuntos de *train* (82783 imágenes) y *mini-val* (5000 imágenes) de COCO 2014 [72], con 81 clases de objetos. El conjunto de *train* se utilizó para entrenar el modelo y el conjunto *mini-val* se utilizó para evaluarlo. Este conjunto *mini-val* fue usado en lugar del conjunto de *test* de COCO porque el *ground-truth* de *test* no se encuentra disponible abiertamente.

5.1.2. Métricas

En los problemas de detección, la métrica de evaluación más utilizada [73, 71] es la mAP (*mean Average Precision*), que es la media del AP (*Average Precision*) de cada clase. Para cada clase, el problema de detección se formula con la siguiente pregunta: ¿Dónde están los



(a) **(b)**

Figura 5.1.3: (a) Una ecuación visual de la definición de IoU. (b) Ejemplos y apreciación cualitativa (malo, bueno y excelente) de distintos valores de IoU posibles. Imágenes obtenidas de [76].

objetos de clase i en la imagen (si los hay)? Para determinar si una predicción responde esta pregunta de forma correcta, se hace una asignación de detecciones propuestas a *bounding boxes* del *ground-truth* y se determina si cada una de ellas es un verdadero o falso positivo, según el solapamiento que haya. En PASCAL VOC [73], para ser una detección correcta el grado de solapamiento entre el *bounding box* propuesto B_p y el *bounding box* del *ground-truth* B_{gt} debe superar el 50 % según la formula:

$$IoU(B_p, B_{gt}) = \frac{area(B_p \cap B_{gt})}{area(B_p \cup B_{gt})}, \tag{5.1.1}$$

donde IoU significa Intersección sobre Unión (en inglés *Intersection over Union*), ver Figura 5.1.3a.

Las predicciones de la red que superan este criterio de solapamiento son asignadas a la anotación de *ground-truth* en orden decreciente de confianza en la detección. Objetos del *ground-truth* que quedan sin emparejar son un falso negativo, mientras que detecciones repetidas de un objeto son considerados falsos positivos. De esta manera para cada clase se calcula el AP tomando la curva *precision-recall* considerando la precisión $p(r)$ en función del *recall* r. El AP es el valor medio de $p(r)$ sobre el intervalo $r = 0$ a $r = 1$, es decir, el área por debajo de la curva. En nuestro caso discreto:

$$AP = \sum_{k=1}^{n} P(k)\Delta r(k),$$

donde n es el número de detecciones propuestas, $P(k)$ es la precisión más alta al considerar las k detecciones con mayor confianza, y $\Delta r(k)$ es el cambio en *recall* entre tomar las detecciones hasta $k-1$ o hasta k.

De manera similar, en [77] se define AOS (del inglés, *Average Orientation Similarity*), una medida para evaluar la detección de objetos y su pose de manera conjunta, donde reemplazan la precisión $P(k)$ por la similitud coseno de los ángulos de orientación. Esto es:

$$AOS = \sum_{k=1}^{n} \max_{\tilde{k}:\tilde{k}\leq k} s(\tilde{k})\Delta r(k),$$

con

$$s(k) = \frac{1}{k}\sum_{i=1}^{k} \frac{1+\cos\Delta_\theta^{(i)}}{2}\delta_i,$$

y donde $\Delta_\theta^{(i)}$ es la diferencia del ángulo entre el *ground-truth* y la estimación propuesta en la detección i y $\delta_i = 1$ cuando la detección i fue asignada a un *bounding box* del *ground-truth* y $\delta_i = 0$ en el caso contrario.

Otra métrica alternativa a AOS es AVP (en inglés, *Average Viewpoint Precision*) definida en [78] donde se considera un verdadero positivo a una detección que tiene IoU > 0.5 con el *ground-truth* y además $\Delta_\theta < \frac{2\pi}{n_v}$, con $n_v \in \{4, 8, 12, 16, 24\}$ definiendo el número de vistas posibles. Es decir, se reemplaza la medida continua de similitud de coseno por un umbral que separa las predicciones correctas de las incorrectas.

5.1.3. Resultados en la detección

Se exploró el efecto de utilizar un dataset sintético para entrenar algunas de las capas responsables de la detección de objetos en Faster R-CNN.

Como primer experimento, se partió de una red entrenada con datos reales, y se evaluó su mAP al continuar su entrenamiento con datos sintéticos. Los pesos iniciales fueron los de entrenar Faster R-CNN con el dataset *train* de COCO 2014 [72], refinados para las clases en común con el dataset sintético (cama, silla, monitor, sofá, mesa e inodoro). Luego se continuó el entrenamiento de las capas RPN y de las últimas capas clasificadoras con imágenes del dataset de entrenamiento sintético. Se limitó el ajuste de pesos a estas capas para mantener la mayor parte de la red optimizada a datos reales y que no se sobreajuste a *features* sintéticos. En la Figura 5.1.4 se muestra como cambia el mAP en conjuntos de test de datos reales y de datos sintéticos según la cantidad de iteraciones de entrenamiento con el dataset sintético.

Se puede observar que los datos sintéticos no ayudan al problema de detección en imágenes reales, utilizando este esquema particular de entrenamiento. Además del cambio de texturas y las diferencias visuales entre los datasets, una posible razón para la baja en mAP puede ser debido que la notación de *ground-truth* de objetos parcialmente visibles difiere en el dataset sintético, respecto de los reales. En el caso de objetos en oclusión, el *ground-truth* de las imágenes sintéticas tiene al *bounding box* completo del objeto, mientras que en los datasets reales el *bounding box* es de sólo la región visible. Esto se debe a que este *bounding box* es más útil en la aplicación de esta tesina, donde se necesitan detecciones totales del objeto y no parciales.

La AP de cada clase evaluada en el dataset PASCAL se encuentra en la Tabla 5.1, donde se puede apreciar que la baja en AP es más pronunciada en algunas clases, como es el caso de mesa y monitor, en comparación a otras (silla y sofá). Esto indica que el agregado de datos sintéticos al entrenamiento no perjudica a todas las clases de la misma manera, sino que en el modelo original hay clases que tienen una mayor compatibilidad con sus instancias sintéticas.

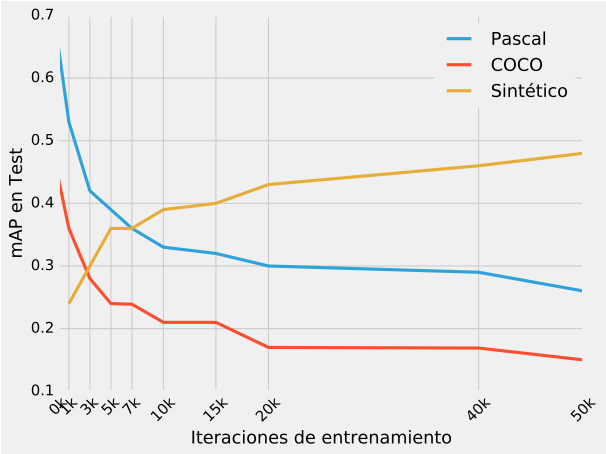


Figura 5.1.4: Efecto del entrenamiento con datos sintéticos en el mAP de datos reales.

n^o de iteraciones	silla	monitor	sofá	mesa
0k	.54	.74	.72	.61
1k	.5	.52	.70	.40
3k	.47	.4	.67	.15
5k	.46	.36	.64	.08
10k	.48	.22	.61	.03

Tabla 5.1: AP para cada clase de objeto en el conjunto de test de PASCAL, al usar el dataset sintético de entrenamiento.

Mientras que en otras clases no hay relación alguna, como por ejemplo entre una mesa real y una mesa sintética. Esto puede deberse a que en el caso de monitores y mesas, los ejemplos sintéticos son paralelepípedos muy simples, que no alcanzan a contener información útil para la CNN.

Luego de realizar estas pruebas se continuó este esquema de entrenamiento, realizando nuevamente un entrenamiento reducido con imágenes reales y luego entrenar con imágenes sintéticas, con el fin de explorar los resultados de un entrenamiento alternado. Concretamente, luego de las primeras 250k iteraciones con el dataset sintético, se entrenó 5k iteraciones en el dataset Train COCO 2014, y luego se volvió a entrenar con las imágenes sintéticas.

Los valores de mAP obtenidos se resumen en la Figura 5.1.5. Se puede observar un mismo comportamiento de mAP que en la primera ronda de entrenamiento con datos sintéticos. La única diferencia parece ser que ahora el modelo se ha adaptado a los dos datasets a costa de perder performance en el dataset real. Por ejemplo, en la iteración 3k se puede obtener un modelo con mAP cercano a 0.5 tanto en el dataset sintético como en el PASCAL, lo que no era posible en la primer ronda de entrenamiento.

Resultados en Orientación

Para evaluar las predicciones de pose y tamaño, se utilizó el conjunto de test de los datos sintéticos. Como fue detallado previamente, el AOS considera la precisión de las detecciones

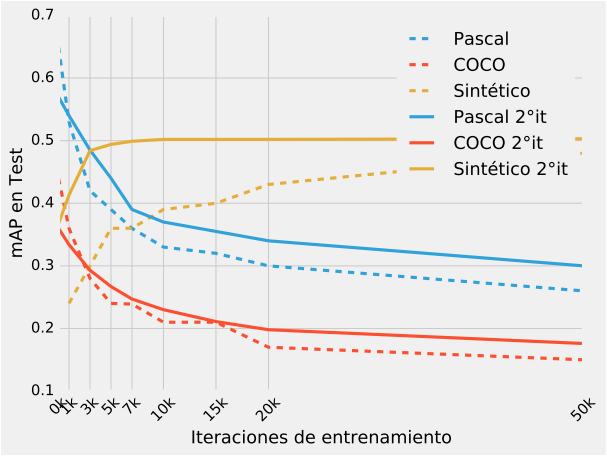


Figura 5.1.5: Efecto de la segunda ronda de entrenamiento con datos sintéticos en el mAP de datos reales. Los valores de la primera ronda también son incluidos a modo de referencia.

	AP	AOS	OS	AVP	AOS*	OS*	AVP*
0k RPN	.09	.07	.77	.05	.08	.89	.06
15k RPN	.35	.25	.71	.14	.33	.94	.25
40k RPN	.49	.35	.71	.20	.46	.94	.35
MultiBin[66] Autos	.89	.88	.99	~	~	~	~
MultiBin[66] Bicicletas	.74	.59	.80	~	~	~	~
ObjectNet3d[79]	.67	.57	.84	.42	~	~	~

Tabla 5.2: Comparación de los máximos valores de *Average Precision* (AP), *Average Orientation Similarity* (AOS), *Orientation Score* (OS) y *Average Viewpoint Precision* (AOS) obtenidos para las distintas iteraciones de entrenamiento de la capa RPN con datos sintéticos. También incluidos a modo de referencia, los valores obtenidos por otros trabajos similares.

ponderada por la similitud del coseno de los ángulos. Por lo tanto, el AP es una cota superior del AOS. El cociente AOS/AP = OS (*Orientation Score*) da una idea del valor absoluto de similitud de orientación, independientemente del valor de AP.

La Tabla 5.2 muestra una comparación entre los máximos valores de AOS, AVP y OS para distintos niveles de entrenamiento de las capas de detección. En todos estos casos se entrenó las capas de orientación por 250k iteraciones. En el caso de AOS*, AVP* y OS* se consideran iguales los ángulos θ y $\theta+\pi$, que son métricas útiles para evaluar objetos simétricos donde esas dos orientaciones son visualmente indistinguibles. AVP y AVP* se calcularon para 12 vistas de los objetos, de la misma manera que en [79]. Para tener valores de referencia, se incluyeron resultados del trabajo MultiBin [66] sobre el cual se basa el detector de pose de esta tesina, y ObjectNet3D [79] un detector similar basado en Faster R-CNN y entrenado sobre un dataset de imágenes reales anotadas con información de pose desarrollado para ese trabajo. Multibin fue entrenado para dos clases: Autos y Bicicletas, por lo que se presentan los valores logrados en ese trabajo de forma separada para cada clase.

Los valores observados de OS y OS*, indican que una alta proporción de detecciones poseen la orientación correcta, independientemente si el AP es mayor o menor. Por lo tanto, se puede

	bañera	cama	silla	escr.	cajonera	monitor	m. luz	sofá	mesa	inodoro	prom.
μ (m)	.32	.53	.22	.43	.34	.21	.19	.60	.49	.20	.35
$\sigma_{\overline{x}}$ (m)	.005	.009	.001	.005	.004	.002	.002	.008	.005	.003	.004
$\ddot{\mu}$ (m)	.34	.58	.23	.44	.36	.2	.2	.63	.56	.20	.37

Tabla 5.3: Resumen del error medido (en metros) en la predicción de dimensiones de objeto por clase. Se muestra la media del error (μ), el error estándar de la media ($\sigma_{\overline{x}}$) y a modo de referencia, el error medido si la predicción fuese una constante igual a las dimensiones medias de los datos de entrenamiento ($\ddot{\mu}$). Al final se muestra el promedio de cada uno de los errores. Abreviaciones: escr.(escritorio), m. luz(mesa de luz), prom.(promedio).

concluir que en los datos sintéticos la red está estimando valores mayormente correctos de orientación. También se puede ver que los trabajos [66] y [79] se obtienen métricas mejores pero similares de OS. En [66] se obtiene un OS particularmente alto, pero sólo para la clase auto, donde el mismo método en la clase bicicletas obtiene un OS similar al resto de los valores presentados.

Resultados de Dimensiones

Para evaluar el error en la predicción de dimensiones se calculó en cada caso la distancia $\mathtt{L}_2$ con el *ground-truth*. Luego la media de estas distancias (μ) y el error estándar de la media ($\sigma_{\overline{x}}$). Los resultados (en metros) se presentan en la Tabla 5.3, donde se comparan con la media del error si la predicción fuese una constante igual a la media de las dimensiones de los datos de entrenamiento ($\ddot{\mu}$). No se puede observar una mejora significativa respecto a una predicción de un valor constante igual a la de la media de los datos. Se puede concluir que la red no logra modelar los ajustes respecto a la media de cada clase de los datos de entrenamiento.

5.2. Evaluación de S-PTAM con mapeo de objetos

5.2.1. Dataset

Para evaluar el error de mapeo de objetos, se realizó la ejecución del sistema S-PTAM sobre seis secuencias grabadas con una cámara estéreo ZED, cada imagen a una resolución de 672×376, a una frecuencia de adquisición de 15hz, y con un *baseline* entre cada cámara de 120mm. La Figura 5.2.1 muestra la cámara ZED.

Encontrar un dataset sobre el cual evaluar el sistema resultó complejo, ya que debía contar con las imágenes de una cámara estéreo para utilizar S-PTAM y además contar con *ground-truth* de los objetos de las clases entrenadas. Inicialmente se trató de utilizar un entorno sintético, pero tuvo la desventaja de contar con pocos *features* debido a la baja textura renderizada, que son requeridos para el correcto funcionamiento de S-PTAM. Finalmente, se grabaron las secuencias en distintas habitaciones de oficina, que contienen el siguiente subconjunto de las 10 clases de objetos utilizados en el dataset de entrenamiento: silla, monitor y sofá. Los escritorios y las mesas fueron excluidos de la evaluación debido a la baja performance de detección en dichas clases.

El *ground-truth* de los objetos se obtuvo utilizando marcadores AR (realidad aumentada) y

Figura 5.2.1: La cámara ZED de Stereolabs. Imagen obtenida de [80].

el paquete de ROS AR_TRACK_ALVAR[81] que conociendo las dimensiones de los marcadores, calcula la pose de los mismos respecto a la cámara. Usando estas transformaciones rígidas proporcionadas por AR_TRACK_ALVAR en cada *frame*, se propagaron manualmente para obtener la posición de los marcadores respecto a la posición inical de la cámara. La posición del centro del *bounding cube* de los objetos respecto a sus marcadores fue estimada con mediciones manuales. Estas posiciones finales también fueron corroboradas con mediciones manuales. El sistema coordenadas de la cámara ZED tiene como origen su sensor izquierdo y sigue la convención ilustrada en la Figura 3.1.1a. El marco de coordenadas del mundo es el marco de coordenadas de la cámara en su posición inicial.

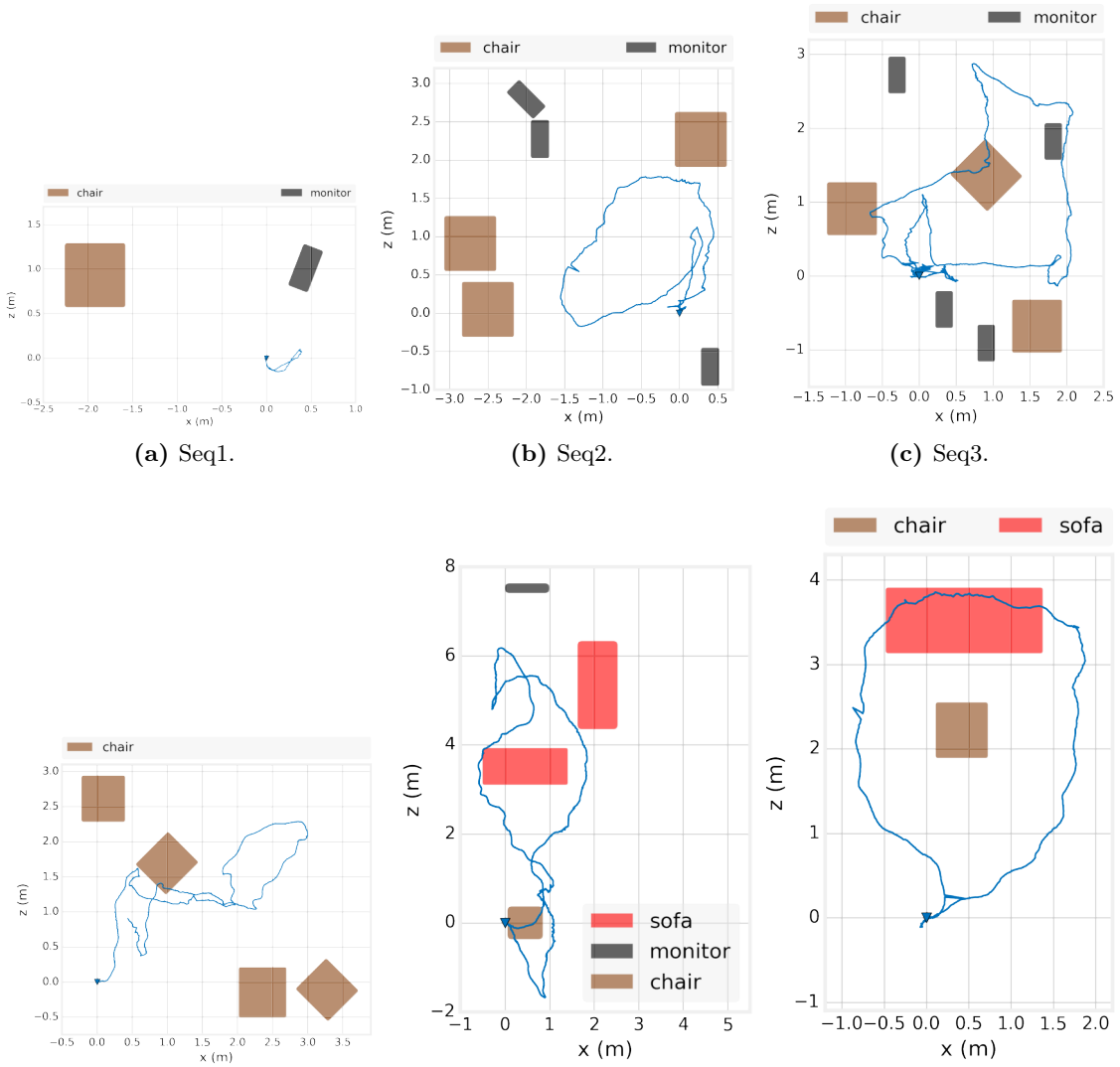
Las secuencias grabadas duran en promedio un minuto, y en la mayoría de ellas la cámara realiza un recorrido circular en la habitación. En la Figura 5.2.2 se muestra un gráfico del *ground-truth* de los objetos para cada secuencia, además del recorrido de la cámara estimado por S-PTAM, proyectado sobre el plano XZ del mundo (vista desde arriba).

5.2.2. Resultados

El sistema realizó su corrida mapeando los objetos de manera *online* y localizándose en tiempo real, y se midieron los errores de los objetos confiables (ver Sección 4.2.6) al final de la ejecución. Se probó el sistema con la red neuronal entrenada con imágenes sintéticas en capas de detección y capas de orientación. Si bien el mAP es más bajo con este modelo en los datasets PASCAL y COCO, se observó una leve mejora de precisión y orientación en las secuencias grabadas.

Los errores observados se resumen en la Figura 5.2.3, donde se promedian los distintos tipos de errores (error de posición, error de orientación y error de dimensiones) en cada secuencia para cada clase del objeto. También se muestra el error promediado de todas las secuencias. En el caso del error de las dimensiones del objeto, se exhibe el error $\mathtt{L}_2$ absoluto, y además el error $\mathtt{L}_2$ relativo al diámetro promedio de la clase de objeto. El diámetro del objeto es considerando la mayor distancia entre dos puntos que estén en su *bounding cube.*

En la Tabla 5.4 se presentan los valores de detección de información de una escena. Para cada secuencia se contabilizan los objetos presentes en el *ground-truth* y los localizados en el mapa por el sistema. En ninguna de las secuencias se obtuvo falsos positivos, es decir, objetos localizados en el mapa que no se correspondan con uno en el *ground-truth.* Tampoco hubo múltiples detecciones del mismo objeto.



(a) Seq1. **(b)** Seq2. **(c)** Seq3.

(d) Seq4. **(e)** Seq5. **(f)** Seq6.

Figura 5.2.2: Proyección sobre el plano XZ del mundo (vista desde arriba) de la trayectoria estimada por S-PTAM (en azul) y del *ground-truth* de los objetos para cada secuencia del dataset.

	Objetos presentes	Objetos relevantes mapeados
Seq1	2	2
Seq2	6	5
Seq3	7	6
Seq4	4	4
Seq5	4	3
Seq6	2	2

Tabla 5.4: Comparación entre el total de objetos presentes en una escena y la cantidad mapeada por S-PTAM.

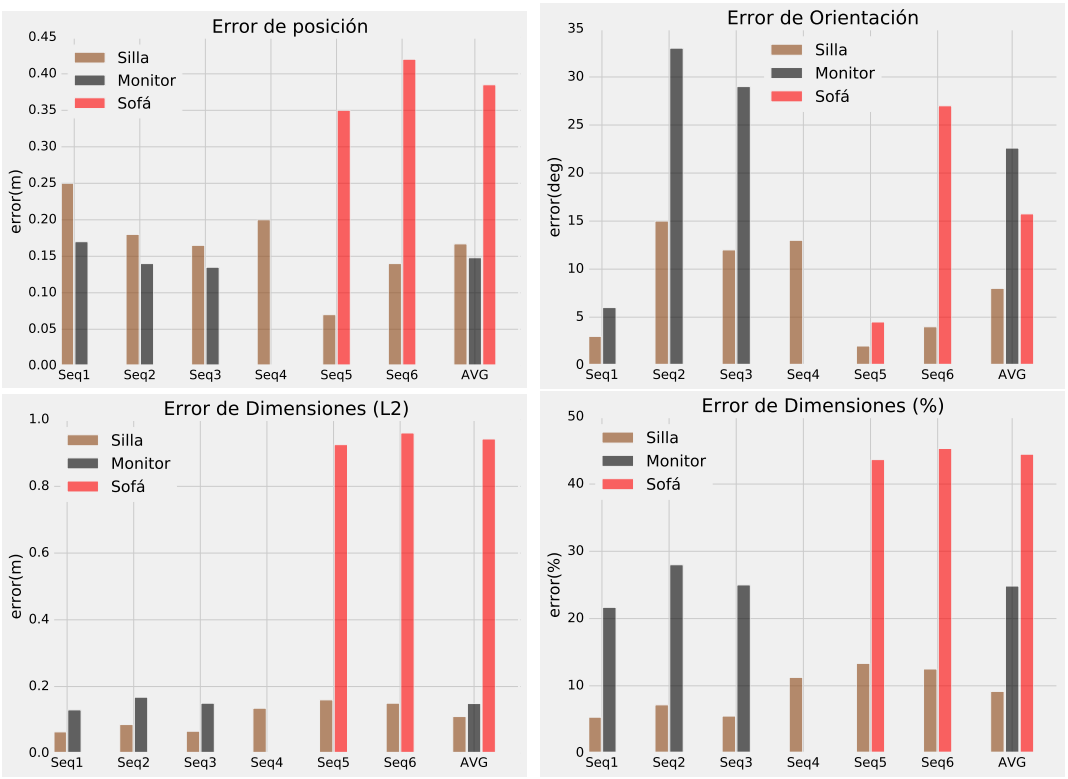


Figura 5.2.3: Resumen de los errores de posición, orientación y dimensiones (absoluto y relativo) medidos en el mapa de objetos de S-PTAM respecto al *ground-truth*. Los errores se presentan promediando los errores de cada clase para cada secuencia, y luego un promedio general. El error relativo de dimensiones es respecto a la diagonal de un objeto promedio perteneciente a esa clase.

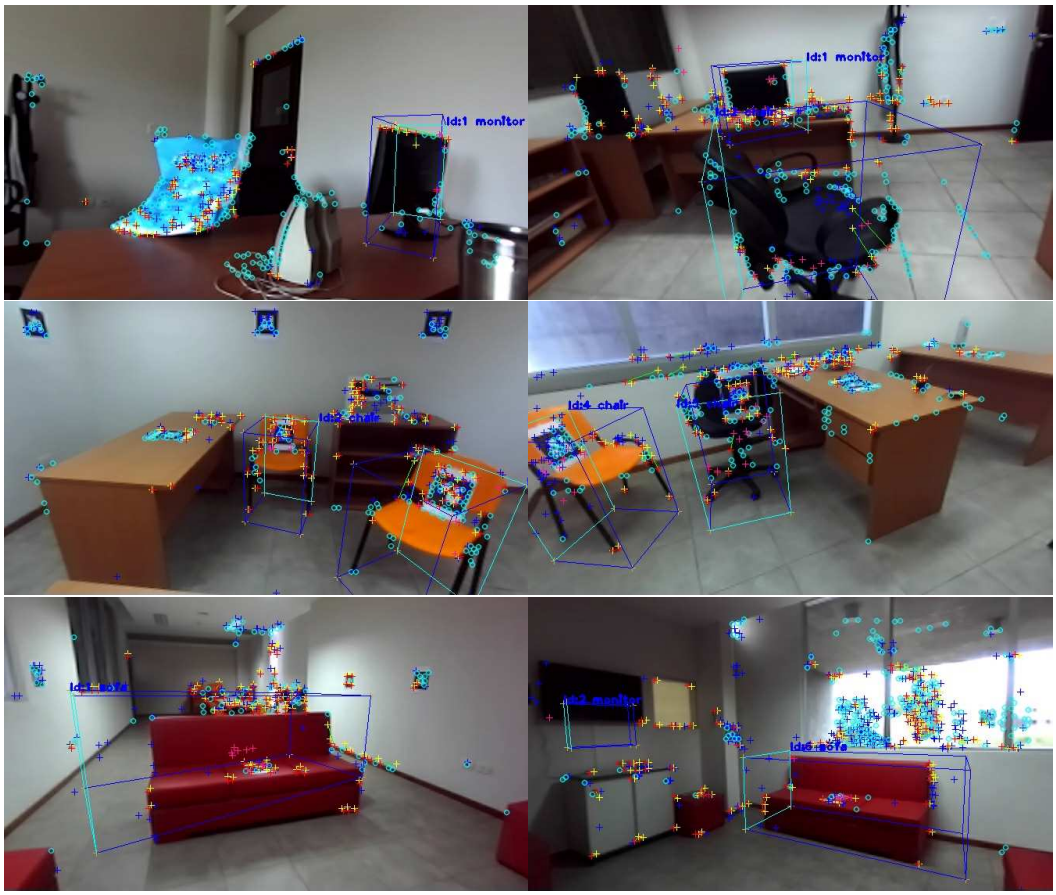


Figura 5.2.4: Capturas de la ejecución del sistema S-PTAM con detección de objetos en el dataset desarrollado.

En la Figura 5.2.4 se muestran algunas capturas del sistema S-PTAM en funcionamiento sobre el dataset desarrollado.

Capítulo 6

Conclusiones

En este trabajo de tesina se presentó un método de *Visual* SLAM que permite detectar y posicionar en un mapa objetos del entorno explorado usando *deep learning* de manera *online*. Para implementarlo se integró un sistema de SLAM de visión estéreo basado en *features* llamado S-PTAM [9] con una red neuronal que resultó de modificar y extender la red Faster R-CNN [19].

Se extendió la red neuronal para que estime la orientación y las dimensiones de cada objeto detectado, además de los datos de clase y *bounding box*. Se obtuvo de esta manera un modelo entrenable *end to end* capaz de aproximar la pose 3D de las detecciones a partir de una imagen (y la matriz intrínseca de la cámara usada).

La red neuronal fue entrenada de forma supervisada, por lo que se desarrolló un dataset de imágenes sintéticas basadas en modelos 3D de objetos, que incluye el *ground-truth* necesario para el entrenamiento (para cada detección: clase, *bounding box*, orientación y dimensiones).

Para la construcción del mapa de objetos, se modificó S-PTAM para que opere conjuntamente y de manera desacoplada con un módulo de detección de objetos que procesa las imágenes que recibe S-PTAM. Para cada detección que recibe del módulo, S-PTAM utiliza la matriz de calibración intrínseca y la pose estimada de la cámara para posicionar la detección en el espacio 3D. Luego, se determina si la detección se corresponde con un objeto ya en el mapa, o con una nueva instancia de objeto, y se refina su posición usando los puntos del mapa y se consolida esta observación del objeto. Finalmente, se fusionan todas las observaciones de un mismo objeto, actualizando la pose del objeto en el mapa.

Se evaluó la performance del detector de objetos siguiendo distintos esquemas de entrenamiento en datasets de imágenes reales y sintéticas. Además, se desarrolló un dataset de seis secuencias capturadas en entornos de oficina para poder probar el sistema completo.

En cuanto al detector de objetos, se pudo concluir que el uso de imágenes sintéticas en la etapa de entrenamiento deteriora en general la detección en imágenes reales. Sin embargo, se pudieron observar buenas estimaciones de orientación en imágenes reales, a pesar de que el entrenamiento de las capas responsables de esa predicción fue realizado exclusivamente con imágenes sintéticas. Respecto a la predicción de dimensiones, no se pudo observar una mejora en las predicciones de la red respecto a la estimación constante de la media de los datos.

Después de evaluar el sistema de mapeo de objetos basado en S-PTAM se concluye que,

para ciertas clases de objetos, este sistema es apto para robots autónomos que necesitan interactuar con su entorno, mostrando un bajo error de posición y orientación ademas de una baja cantidad de objetos no detectados o falsos positivos.

6.1. Trabajo futuro

Como trabajo futuro derivado de la presente tesina, se identifican varias mejoras que pueden ser investigadas.

Durante el período de trabajo de esta tesina, se desarrollaron nuevas arquitecturas para detección de objetos, como lo son SSD [52] o Mask R-CNN [82]. Un posible trabajo a futuro cosistiría en utilizar Mask R-CNN en el módulo de detección de objetos. Mask R-CNN es la cuarta generación en las arquitecturas R-CNN, tomando como base Faster R-CNN e incorporando notables mejoras. Una de ellas es proveer la segmentación del objeto dentro de cada *bounding box* (ver Figura 6.1.1). Actualmente, en este trabajo se implementa una heurística para hallar un *map point* de S-PTAM que pertenezca al objeto para cada detección. El uso de las segmentaciones le permitirían a S-PTAM determinar de manera mucho más precisa si un *map point* pertenece o no al objeto. Además, Mask R-CNN reemplaza a la arquitectura VGG-16 como extractor de features por una Res-Net [83] e introduce una nueva capa de *pooling* más precisa llamada *ROI Align*. En conjunto, estas mejoras implicarían un menor error de localización de objetos y una mejor separación de objetos cercanos entre sí. El código de Mask-RCNN fue liberado recientemente (Febrero de 2018) [84].

Otra mejora posible al detector de objetos puede lograrse mediante el uso de datos reales en lugar de datos sintéticos en el entrenamiento. Recientemente, la Universidad de Stanford desarrolló el dataset ObjectNet3D [79], un dataset de imágenes reales con *ground-truth* de pose y 100 categorías de objetos, que permitirían entrenar la red propuesta en esta tesina con un único dataset.

Otra línea de trabajo planteada a futuro consiste en lograr que las mediciones de los objetos agreguen información a S-PTAM que mejore la localización. Esto resultaría en un sistema de SLAM completamente semántico, ya que este es el concepto principal de SLAM: que la mediciones se utilicen para resolver los problemas de mapeo y localización simultáneamente. En este trabajo, las mediciones sólo se utilizan para construir un mapa de objetos, y no tienen incidencia en la performance del resto del sistema SLAM.

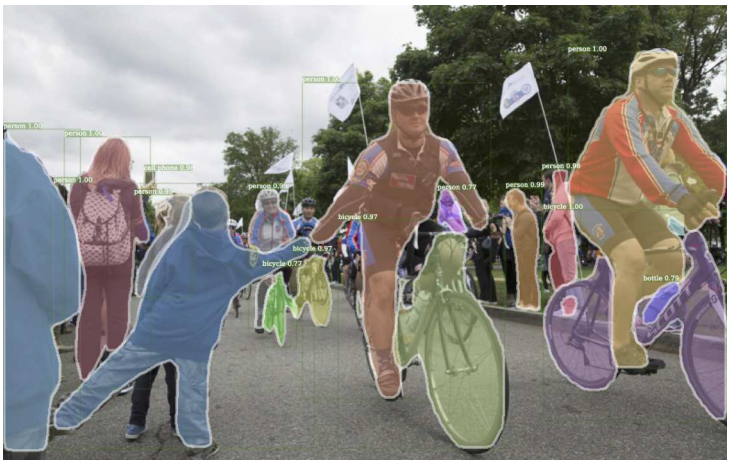


Figura 6.1.1: Ejemplo de la salida de Mask R-CNN sobre la imagen de entrada. Se pueden observar los *bounding boxes* (en verde) y la segmentación a nivel de píxeles para cada instancia de objeto, así como también su clase y confianza. Las clases detectadas son *bicicleta* y *persona*. Imagen obtenida de [84].

Bibliografía

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

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

[3] Georg Klein and David Murray. Parallel tracking and mapping for small AR workspaces. In *Proc. Sixth IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR'07)*, Nara, Japan, November 2007.

[4] Jakob Engel, Thomas Schöps, and Daniel Cremers. *LSD-SLAM: Large-Scale Direct Monocular SLAM*, pages 834–849. Springer International Publishing, Cham, 2014.

[5] Stefan Leutenegger, Simon Lynen, Michael Bosse, Roland Siegwart, and Paul Furgale. Keyframe-based visual–inertial odometry using nonlinear optimization. *The International Journal of Robotics Research*, 34(3):314–334, 2015.

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

[7] Tong Qin, Peiliang Li, and Shaojie Shen. Vins-mono: A robust and versatile monocular visual-inertial state estimator. *arXiv preprint arXiv:1708.03852*, 2017.

[8] J. Engel, V. Koltun, and D. Cremers. Direct sparse odometry. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, March 2018.

[9] Taihú Pire, Thomas Fischer, Gastón Castro, Pablo De Cristóforis, Javier Civera, and Julio Jacobo Berlles. S-ptam: Stereo parallel tracking and mapping. *Robotics and Autonomous Systems*, 93(Supplement C):27 – 42, 2017.

[10] Niko Sünderhauf, Feras Dayoub, Sean McMahon, Markus Eich, Ben Upcroft, and Michael Milford. Slam–quo vadis? in support of object oriented and semantic slam. 2015.

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

[12] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[13] Y. Bengio, A. Courville, and P. Vincent. Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1798–1828, Aug 2013.

[14] 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.

[15] Nicola Jones. Computer science: The learning machines. *Nature*, 505(7482):146–148, 2014.

[16] Alex Graves and Jürgen Schmidhuber. Offline handwriting recognition with multidimensional recurrent neural networks. In *Advances in Neural Information Processing Systems (NIPS)*, pages 545–552. 2008.

[17] Yaniv Taigman, Ming Yang, Marc'Aurelio Ranzato, and Lior Wolf. Deepface: Closing the gap to human-level performance in face verification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1701–1708, 2014.

[18] 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.

[19] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards real-time object detection with region proposal networks. In *Advances in Neural Information Processing Systems (NIPS)*, 2015.

[20] 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.

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

[22] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22nd ACM international conference on Multimedia*, pages 675–678. ACM, 2014.

[23] Gabriel Nützi, Stephan Weiss, Davide Scaramuzza, and Roland Siegwart. Fusion of IMU and Vision for Absolute Scale Estimation in Monocular SLAM. *Journal of Intelligent & Robotic Systems*, 61(1):287–299, 2011.

[24] Lindsay Kleeman. Advanced sonar and odometry error modeling for simultaneous localisation and map building. In *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003) (Cat. No.03CH37453)*, volume 1, pages 699–704 vol.1, Oct 2003.

[25] Fabrizio Abrate, Basilio Bona, and Marina Indri. Experimental EKF-based SLAM for Mini-rovers with IR Sensors Only. In *EMCR*, 2007.

[26] David Cole and Paul Newman. Using laser range data for 3D SLAM in outdoor environments. In *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.*, pages 1556–1563, May 2006.

[27] Justin David Carlson. *Mapping Large, Urban Environments with GPS-Aided SLAM*. PhD thesis, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, August 2010.

[28] Leopoldo Armesto and Josep Tornero. SLAM based on Kalman filter for multi-rate fusion of laser and encoder measurements. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, volume 2, pages 1860–1865 vol.2, Sept 2004.

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

[30] Emanuele Trucco and Alessandro Verri. *Introductory Techniques for 3-D Computer Vision*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1998.

[31] Yi Ma, Stefano Soatto, Jana Kosecka, and Shankar Sastry. *An Invitation to 3-D Vision: From Images to Geometric Models*. SpringerVerlag, 2003.

[32] Hao Li, Bart Adams, Leonidas Guibas, and Mark Pauly. Robust Single-view Geometry and Motion Reconstruction. In *ACM SIGGRAPH Asia 2009 Papers*, SIGGRAPH Asia '09, pages 175:1–175:10, New York, NY, USA, 2009. ACM.

[33] Cesar Cadena, Luca Carlone, Henry Carrillo, Yasir Latif, Davide Scaramuzza, José Neira, Ian D. Reid, and John J. Leonard. Simultaneous Localization And Mapping: Present, Future, and the Robust-Perception Age. *Computing Research Repository (CoRR)*, abs/1606.05830, 2016.

[34] Jan Stühmer, Stefan Gumhold, and Daniel Cremers. *Real-Time Dense Geometry from a Handheld Camera*, pages 11–20. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[35] Gottfried Graber, Thomas Pock, and Horst Bischof. Online 3D reconstruction using convex optimization. In *2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops)*, pages 708–711, Nov 2011.

[36] Richard Newcombe and Andrew Davison. Live dense reconstruction with a single moving camera. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1498–1505, June 2010.

[37] Thomas Lemaire, Cyrille Berger, Il-Kyun Jung, and Simon Lacroix. Vision-Based SLAM: Stereo and Monocular Approaches. *International Journal of Computer Vision*, 74(3):343–364, 2007.

[38] M. Tomono. Robust 3D SLAM with a stereo camera based on an edge-point ICP algorithm. In *2009 IEEE International Conference on Robotics and Automation*, pages 4306–4311, May 2009.

[39] Kuen-Han Lin and Chieh-Chih Wang. Stereo-based simultaneous localization, mapping and moving object tracking. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3975–3980, Oct 2010.

[40] Etienne Mouragnon, Maxime Lhuillier, Michel Dhome, Fabien Dekeyser, and Patrick Sayd. Monocular Vision Based SLAM for Mobile Robots. In *18th International Conference on Pattern Recognition (ICPR'06)*, volume 3, pages 1027–1031, 2006.

[41] Seo-Yeon Hwang and Jae-Bok Song. Monocular Vision-Based SLAM in Indoor Environment Using Corner, Lamp, and Door Features From Upward-Looking Camera. *IEEE Transactions on Industrial Electronics*, 58(10):4804–4812, Oct 2011.

[42] Ethan Eade, Philip Fong, and Mario E. Munich. Monocular graph slam with complexity reduction. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3017–3024, Taipei, Taiwan, October 2010.

[43] Liz Murphy, Timothy Morris, Ugo Fabrizi, Michael Warren, Michael Milford, Ben Upcroft, Michael Bosse, and Peter Corke. Experimental Comparison of Odometry Approaches. In Jaydev P. Desai, Gregory Dudek, Oussama Khatib, and Vijay Kumar, editors, *Experimental Robotics*, volume 88 of *Springer Tracts in Advanced Robotics*, pages 877–890. Springer International Publishing, 2013.

[44] Sid Yingze Bao, Mohit Bagra, Yu-Wei Chao, and Silvio Savarese. Semantic Structure from Motion with Points, Regions, and Objects. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[45] Renato F. Salas-Moreno, Richard A. Newcombe, Hauke Strasdat, Paul H. J. Kelly, and Andrew J. Davison. SLAM++: simultaneous localisation and mapping at the level of objects. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1352–1359, 2013.

[46] Sudeep Pillai and John Leonard. Monocular SLAM Supported Object Recognition. In *Proceedings of Robotics: Science and Systems*, Rome, Italy, July 2015.

[47] Dorian Gálvez-López, Marta Salas, Juan D. Tardós, and J.M.M. Montiel. Real-time monocular object slam. *Robot. Auton. Syst.*, 75(PB):435–449, January 2016.

[48] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. Orb: An efficient alternative to sift or surf. In *Proceedings of the 2011 International Conference on Computer Vision*, ICCV '11, pages 2564–2571, Washington, DC, USA, 2011. IEEE Computer Society.

[49] Dorian Galvez-Lopez and J. D. Tardos. Bags of binary words for fast place recognition in image sequences. *IEEE Transactions on Robotics*, 28(5):1188–1197, October 2012.

[50] Renato F. Salas-Moreno, Ben Glocken, Paul H. J. Kelly, and Andrew J. Davison. Dense planar slam. In *Proceedings of the 2014 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, pages 157–164, September 2014.

[51] Niko Sünderhauf, Trung T. Pham, Yasir Latif, Michael Milford, and Ian D. Reid. Meaningful maps with object-oriented semantic mapping. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2017, Vancouver, BC, Canada, September 24-28, 2017*, pages 5079–5085. IEEE, 2017.

[52] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott E. Reed, Cheng-Yang Fu, and Alexander C. Berg. Ssd: Single shot multibox detector. In *ECCV*, 2016.

[53] Trung T Pham, Markus Eich, Ian Reid, and Gordon Wyeth. Geometrically consistent plane extraction for dense indoor 3d maps segmentation. In *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on*, pages 4199–4204. IEEE, 2016.

[54] John McCormac, Ankur Handa, Andrew J. Davison, and Stefan Leutenegger. Semanticfusion: Dense 3d semantic mapping with convolutional neural networks. *Computing Research Repository (CoRR)*, abs/1609.05130, 2016.

[55] Thomas Whelan, Stefan Leutenegger, Renato Salas Moreno, Ben Glocker, and Andrew Davison. Elasticfusion: Dense slam without a pose graph. In *Proceedings of Robotics: Science and Systems*, Rome, Italy, July 2015.

[56] José M. Fácil, Alejo Concha, Luis Montesano, and Javier Civera. Deep single and direct multi-view depth fusion. *Computing Research Repository (CoRR)*, abs/1611.07245, 2016.

[57] D. Eigen and R. Fergus. Predicting depth, surface normals and semantic labels with a common multi-scale convolutional architecture. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 2650–2658, Dec 2015.

[58] Veeravalli S. Varadarajan. *Lie Groups, Lie Algebras, and Their Representations*. Graduate Texts in Mathematics. Springer-Verlag New York, 1984.

[59] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Comput.*, 1(4):541–551, December 1989.

[60] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *Computing Research Repository (CoRR)*, abs/1409.1556, 2014.

[61] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15(1):1929–1958, January 2014.

[62] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014.

[63] J.R.R. Uijlings, K.E.A. van de Sande, T. Gevers, and A.W.M. Smeulders. Selective search for object recognition. *International Journal of Computer Vision*, 2013.

[64] Ross Girshick. Fast R-CNN. In *Proceedings of the International Conference on Computer Vision (ICCV)*, 2015.

[65] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

[66] Arsalan Mousavian, Dragomir Anguelov, John Flynn, and Jana Kosecka. 3d bounding box estimation using deep learning and geometry. In *CVPR*, 2017.

[67] Matthew D. Zeiler and Rob Fergus. *Visualizing and Understanding Convolutional Networks*, pages 818–833. Springer International Publishing, Cham, 2014.

[68] E. Rohmer, S. P. N. Singh, and M. Freese. V-REP: A versatile and scalable robot simulation framework. In *Proceedings of the 2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1321–1326, November 2013.

[69] Zhirong Wu, Shuran Song, Aditya Khosla, Xiaoou Tang, and Jianxiong Xiao. 3d shapenets for 2.5d object recognition and next-best-view prediction. *Computing Research Repository (CoRR)*, abs/1406.5670, 2014.

[70] Ankur Handa, Viorica Patraucean, Vijay Badrinarayanan, Simon Stent, and Roberto Cipolla. Understanding realworld indoor scenes with synthetic data. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4077–4085, 2016.

[71] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014.

[72] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Coco datasets. `http://cocodataset.org/#download`. Accessed: 2018-02-14.

[73] Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge. *International journal of computer vision*, 88(2):303–338, 2010.

[74] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes Challenge 2007 (VOC2007) Results. `http://www.pascal-network.org/challenges/VOC/voc2007/workshop/index.html`.

[75] Ludicorp. Flickr, 2004.

[76] Adrian Rosebrock. Intersection over union (iou) for object detection. `http://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/`. Accessed: 2018-02-14.

[77] Andreas Geiger. Are we ready for autonomous driving? the kitti vision benchmark suite. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, CVPR '12, pages 3354–3361, Washington, DC, USA, 2012. IEEE Computer Society.

[78] Yu Xiang, Roozbeh Mottaghi, and Silvio Savarese. Beyond pascal: A benchmark for 3d object detection in the wild. In *IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 75–82, 2014.

[79] Yu Xiang, Wonhui Kim, Wei Chen, Jingwei Ji, Christopher Choy, Hao Su, Roozbeh Mottaghi, Leonidas Guibas, and Silvio Savarese. Objectnet3d: A large scale database for 3d object recognition. In *European Conference Computer Vision (ECCV)*, 2016.

[80] www.stereolabs.com. `https://www.stereolabs.com/`. Accessed: 2018-02-14.

[81] Scott Niekum. Ros wrapper for alvar, an open source ar tag tracking library. `http://wiki.ros.org/ar_track_alvar`. Accessed: 2018-02-14.

[82] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask R-CNN. In *Proceedings of the International Conference on Computer Vision (ICCV)*, 2017.

[83] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *arXiv prepring arXiv:1506.01497*, 2015.

[84] Ross Girshick, Ilija Radosavovic, Georgia Gkioxari, Piotr Dollár, and Kaiming He. Detectron. `https://github.com/facebookresearch/detectron`, 2018.