Doctoral Dissertation

Advances in 3D Range Sensors Odometry for Mobile Robotics

Andrés Galeote Luque
2024

Javier González Jiménez
José Raúl Ruiz Sarmiento

Tesis doctoral
Programa de Doctorado en Ingeniería Mecatrónica
Dpt. de Ingeniería de Sistemas y Automática
Universidad de Málaga

AUTOR: Andrés Galeote Luque

https://orcid.org/0000-0001-5853-0838

EDITA: Publicaciones y Divulgación Científica. Universidad de Málaga

DECLARACIÓN DE AUTORÍA Y ORIGINALIDAD DE LA TESIS PRESENTADA PARA OBTENER EL TÍTULO DE DOCTOR

D./Dña ANDRÉS GALEOTE LUQUE
Estudiante del programa de doctorado EN INGENIERÍA MECATRÓNICA de la Universidad de Málaga, autor/a de la tesis, presentada para la obtención del título de doctor por la Universidad de Málaga, titulada: ADVANCES IN 3D RANGE SENSORS ODOMETRY FOR MOBILE ROBOTICS

Realizada bajo la tutorización de JAVIER GONZÁLEZ JIMÉNEZ y dirección de JAVIER GONZÁLEZ JIMÉNEZ Y JOSÉ RAÚL RUIZ SARMIENTO (si tuviera varios directores deberá hacer constar el nombre de todos)

DECLARO QUE:

La tesis presentada es una obra original que no infringe los derechos de propiedad intelectual ni los derechos de propiedad industrial u otros, conforme al ordenamiento jurídico vigente (Real Decreto Legislativo 1/1996, de 12 de abril, por el que se aprueba el texto refundido de la Ley de Propiedad Intelectual, regularizando, aclarando y armonizando las disposiciones legales vigentes sobre la materia), modificado por la Ley 2/2019, de 1 de marzo.

Igualmente asumo, ante a la Universidad de Málaga y ante cualquier otra instancia, la responsabilidad que pudiera derivarse en caso de plagio de contenidos en la tesis presentada, conforme al ordenamiento jurídico vigente.

En Málaga, a 15 de MARZO de 2024

Fdo.: ANDRÉS GALEOTE LUQUE Doctorando/a	Fdo.: JAVIER GONZÁLEZ JIMÉNEZ Tutor/a
Fdo.: JAVIER GONZÁLEZ JIMÉNEZ Y JOSÉ RAÚL RUIZ SARMIENTO Director/es de tesis	

Edificio Pabellón de Gobierno. Campus El Ejido. 29071
Tel.: 952 13 10 28 / 952 13 14 61 / 952 13 71 10
E-mail: doctorado@uma.es

UNIVERSIDAD DE MÁLAGA

DEPARTAMENTO DE
INGENIERÍA DE SISTEMAS Y AUTOMÁTICA

El Dr. D. Javier González Jiménez y el Dr. D. José Raúl Ruiz Sarmiento, directores de la tesis titulada "Advances in 3D Range Sensors Odometry for Mobile Robotics" realizada por D. Andrés Galeote Luque, certifican su idoneidad para la obtención del título de Doctor en Ingeniería Mecatrónica.

Málaga, 15 de marzo de 2024

_____

Dr. D. Javier González Jiménez

_____

Dr. D. José Raúl Ruiz Sarmiento

Dept. of System Engineering and Automation
University of Málaga
Studies in Mechatronics

Advances in 3D Range Sensors Odometry for Mobile Robotics

AUTHOR: Andrés Galeote Luque

SUPERVISORS: Javier González Jiménez
José Raúl Ruiz Sarmiento

In the name of someone
really sore right now.

- Zagreus

Contents

Abstract

Localization plays a fundamental role in mobile robotics, as it enables the safe and correct operation of autonomous agents. Among the different existing ways to obtain the pose of the robot over time, it is worth highlighting odometry. Not only does it provide the relative ego-motion of the robot, which can be used directly to recover the trajectory, but it can also function as the front end or initialization of more complex and accurate localization pipelines, like SLAM. This thesis focuses on odometry algorithms that use data from a variety of range sensors.

Range sensors can measure the distance to objects in the scene, and thus can capture the geometric layout of its surroundings. Examples include depth cameras, lidars and radars, each employing a different approach to obtain the geometric information of the scene. They are fairly popular sensors in robotics since knowing the spatial layout can be very valuable, for example for obstacle avoidance. More relevant to this thesis is comparing consecutively captured point clouds to estimate the rigid transformation between them. This way, the movement experienced by the robot can be recovered, known as range sensor odometry.

The contributions contained within this thesis include various odometry methods designed to operate with different range sensors. The first method employs depth images from depth cameras, and combines techniques from direct and indirect approaches. The result is a hybrid method that extracts planar features from the images, which are then matched with points of the other image based on reprojection. The following two odometry methods make use of modern 3D lidars that can output ordered point clouds, allowing them to be represented as images. At the core, they obtain planar features from the input, and iteratively match them with points based on the latest mo-

tion estimation. The second of these 3D lidar odometry methods builds upon this and decouples the motion estimation by segmenting the ground plane, among other improvements and optimizations. Finally, the fourth 3D odometry method employs the radial velocity captured by Doppler-capable range sensors, and distances from the previous methods by estimating the motion from a single observation, instead of performing data association. This is possible by knowing the kinematic model of the vehicle the sensor is mounted on.

Acknowledgments

Everyone who has studied for a PhD (and their friends and relatives) knows what a turbulent journey it is. Being on the verge of landing this rickety and fragile plane, I am lucky to have lots of people to be grateful to. First in line are my supervisors, Javier González Jiménez and José Raúl Ruiz Sarmiento. These have been years of studying a lot, and thankfully I had great teachers to look up to and learn from. Thank you Javier for granting me this opportunity, and for providing the guidance I very much needed during these years. Raúl, you became my co-supervisor to push the lever of the paper-making machine. And push you did, and soon papers started flying out of me. Thank you.

Being a member of MAPIR comes with the perk of having awesome colleagues that know your pain and celebrate your victories. Mostly while mocking me for eating a *pitufo tortilla* with no mayo. No, you cannot have an opinion without trying it. Thanks for all your help, the laughs, the design committees and putting up with my jokes. Because of you, remote work was really awful. I would also like to thank my Swedish colleagues for the warm and cozy environment you create for visitors like me, even in sub-zero temperatures. Thanks to Martin and Vladimír, working with you was a wonderful experience, I cross my fingers and hope our paper gets accepted. Also, it was really cool getting to drive the Husky.

To all my friends, who have put the effort and the weight in the "life" part of the work-life balance: thank you. Fortunately, you are too many to name you all. Instead, let this be a coupon for a real-life hug as appreciation for everything good you bring into my life. You know I'm not very good at showing my appreciation for what you do, but you truly made this achievement possible. Thank you for your unwavering support and warmth. Also, I hope this is the last time I have to explain what odometry is.

To my very scattered family, although planning the Christmas holidays is an accomplishment on itself (maybe bigger than writing a paper?), I am very grateful to have you. From the way you raised me, to driving me to the university, you have made possible every step that I took in this journey. Your frozen tuppers have been essential fuel through these years. But even more significant have been, and still are, your care, support, and unconditional love. Thank you mom for teaching me that math can be fun. Although that turned out to be a lie, some of it seeped deep into me. Thank you dad for passing down your passion for engineering while discussing how to fix a washing machine, and for the extraordinary illustrations included in this thesis. Thank you ramiro and brother for being the best role models, and for always being there for me when I need it. And to the rest of you, my aunts and uncles, my in-laws, thank you. I love you all very much.

Lastly, but not leastly, thank you Paula. You are the one that suffered the most collateral damage from this PhD. Still, there you stood, by my side, helping me get back on my feet. You supplied an infinite amount of everything one needs to keep their head from imploding: laughter, good food, and love. I cannot wait to celebrate this achievement of ours together. Thanks for being by my side no matter what, and for cooking *your* favourite meals when I was feeling down.

Andrés Galeote Luque
Málaga, December 2023

This thesis has been supported by the grant program PRE2018-085026 and the research projects HOUNDBOT (P20_01302), funded by Andalusian Regional Government, and ARPEGGIO (PID2020-117057GB-I00) and WISER (DPI2017-84827-R), funded by the Spanish Government.

Resumen de la tesis

Introducción

Los humanos tenemos la capacidad de, a partir de dos imágenes del mismo entorno tomadas desde diferentes posiciones, estimar a grandes rasgos la diferencia en dicha posición (conocida como *pose relativa*). Pero, ¿y si, en vez de imágenes a color que estamos acostumbrados a ver, comparamos dos nubes de puntos (grupos de puntos) capturados por un tipo diferente de sensor? El lector puede comprobar por sí mismo si es capaz de adivinar la transformación que haría que ambas nubes de puntos de la Figura 1 coincidiesen en el espacio. Los ordenadores, y por lo tanto también los robots, en cambio, tienen más difícil encontrar la solución correcta a este problema, pero cuando lo hacen esta es mucho más precisa de lo que es posible para el ser humano. Ahora pensemos a lo grande. Visualicemos a un robot que tiene uno de estos *sensores de rango*

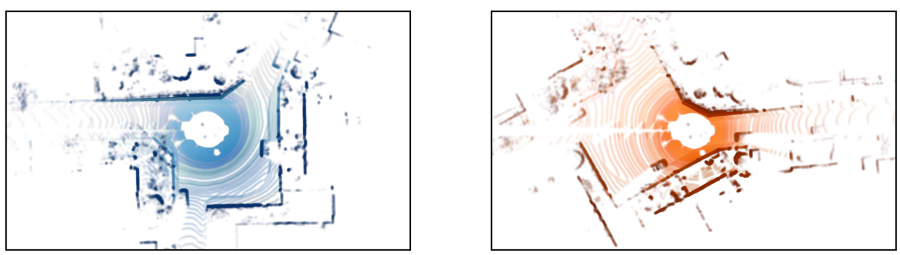


Figure 1: Dos nubes de puntos del mismo escenario capturadas por el mismo sensor desde diferentes posiciones. ¿Puedes estimar la pose relativa entre ellas?

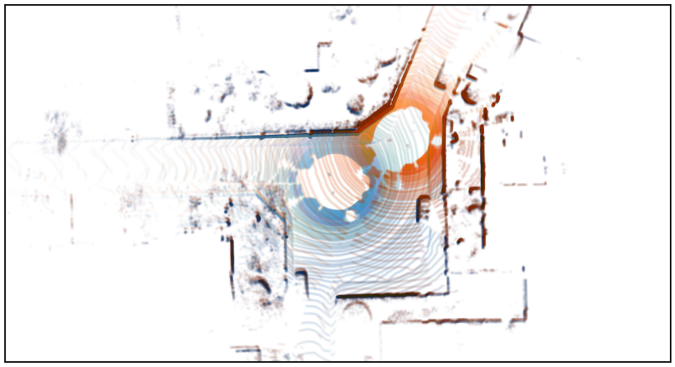


Figure 2: Solución a la pregunta de la Figura 1, las nubes de puntos se pueden mostrar superpuestas cuando se conoce la pose relativa entre ellas.

que captura nubes de puntos de su entorno. Al calcular la pose relativa entre cada pareja consecutiva de nubes de puntos, el resultado se puede componer y acumular. De esta forma, es posible recuperar el movimiento del sensor (y por lo tanto del robot) en el tiempo, es decir, su trayectoria. Este problema es el que la *odometría de sensores de rango* pretende resolver, y en esta tesis describimos los diferentes algoritmos de odometría que hemos desarrollado para diferentes tipos de sensores de rango.

La capacidad de localizarse en su entorno es esencial para que robots móviles y vehículos autónomos puedan ejecutar sus funciones de forma correcta y segura. El cálculo de la localización se puede realizar empleando diferentes sensores y técnicas [63,68]. Sin embargo, los sensores de rango destacan por su habilidad para capturar información geométrica de su entorno, la cual además de ser útil para la localización también lo es para la evitación de obstáculos y la creación de mapas, entre otras tareas. En particular, la odometría de sensores de rango puede recuperar el movimiento de un sensor de rango, y por consiguiente localizar al robot en la escena. Esto es valioso de por sí, pero se puede llevar al siguiente nivel de precisión al crear un mapa del entorno de manera simultánea. Esta técnica, conocida como SLAM del inglés *Simultaneous Localization and Mapping* [9], es utilizada por numerosos sistemas debido a su precisión incluso tras largos periodos de tiempo, y la odometría juega un papel importante en ella.

Esta tesis contribuye con varios algoritmos de odometría desarrollados para diferentes tipos de sensores de rango. Los métodos que aquí se proponen, ya sea por su cuenta o como parte de un sistema SLAM, proveen información de la localización a los robots móviles, permitiéndoles navegar y realizar tareas de alto nivel de manera segura.

Por cierto, el lector puede comprobar si su solución a la pregunta de la Figura 1 es correcta en la Figura 2.

Motivación

El título de esta tesis nace de la fusión de dos conceptos, sensores de rango y odometría. Para una mejor comprensión de la combinación resultante, vamos a describir ambos términos por separado. Se conoce como sensores de rango a aquellos con la habilidad de medir la distancia hasta los objetos de su entorno, por lo que se puede conocer la posición relativa de los puntos observados con respecto al sensor. Esta información geométrica se conoce como *nube de puntos*, y representa un grupo de puntos en el espacio. Cada tipo de sensor de rango utiliza una tecnología y fenómenos físicos diferentes para medir dicha distancia. Una forma de obtener esta información, como es el ejemplo de las cámaras de profundidad como el dispositivo Kinect [110], consiste en proyectar un patrón conocido, y medir cómo se deforma al reflejarse en las diferentes formas de la escena. Otro método ampliamente utilizado consiste en medir el tiempo que tarda una señal emitida por el sensor en ser recibida de vuelta, representado en la Figura 3, como es el caso en cámaras de tiempo de vuelo, lidars y radares. Modificando este último procedimiento de forma que mida la fase de la señal de retorno, se puede también capturar la velocidad de los objetos observados, convirtiéndose así en un sensor de rango con capacidad Doppler.

En cuanto al término odometría, consiste en la estimación de la trayectoria seguida por un sensor en función de sus observaciones [63]. Aplicado a sensores de rango se puede entender como la estimación de la pose relativa entre dos nubes de puntos capturadas consecutivamente, ya que la composición de estas resulta en la trayectoria del sensor. Existe una amplia variedad de maneras de calcular dicha pose relativa, y generalmente tratan de registrar ambas nubes de puntos [41], haciendo que queden superpuestas. La Figura 4 muestra un ejemplo básico. Como consecuencia de la composición de las estimaciones de poses relativas en la odometría, los errores de cada estimación se acumulan con el tiempo, lo que se conoce como *drift* [83]. Para evitar este problema se pueden registrar las nubes de puntos capturadas con un mapa global en vez de con la nube de puntos anterior, pero esto requiere tener dicho mapa global de la zona que no siempre estará disponible. Una técnica popular que aborda este asunto es SLAM [9], que consiste en comparar las nubes de puntos capturadas con un mapa que se está creando simultáneamente con la localización. Los algoritmos de SLAM necesitan localizar las nubes de puntos capturadas con respecto al mapa, por lo que acostumbran a usar un método de odometría para proveer una estimación de alta frecuencia del movimiento. Es por ello que los métodos de odometría tienen que ser, además de precisos, eficientes, de manera que permitan a las tareas de alto nivel (como la construcción del mapa y el cierre de bucle) tener acceso a los recursos computacionales necesarios para su correcta operación.

El Capítulo 2 contiene una explicación detallada de los sensores de rango y la odometría. Los métodos de odometría desarrollados para cámaras de pro-

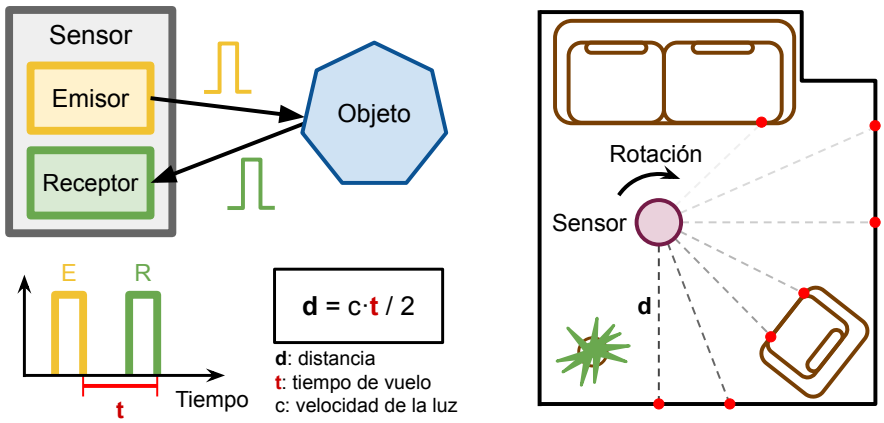


Figure 3: Izquierda: principio de funcionamiento de los sensores de tiempo de vuelo. El emisor manda una señal, y la distancia a un objeto se calcula en función al tiempo que tarda el receptor en capturar la señal reflejada. Derecha: un sensor de rango rotativo midiendo distancias en diferentes direcciones conforme gira. El resultado es una nube de puntos (grupo de puntos rojos) que contiene la información de la geometría del entorno.

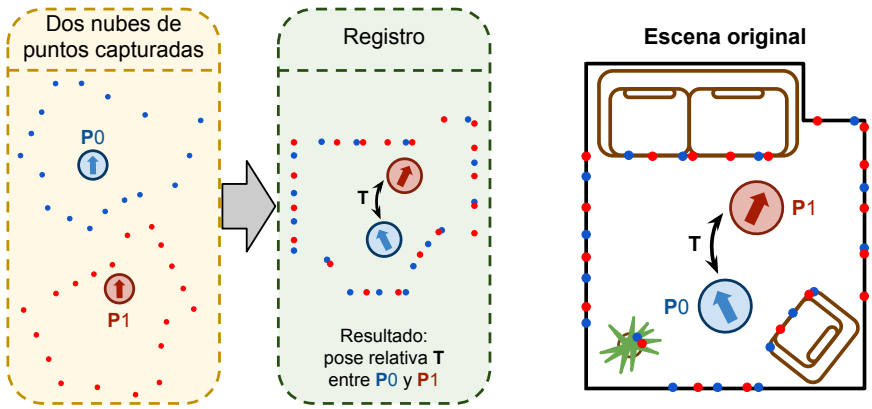


Figure 4: Izquierda: dos nubes de puntos de la misma habitación capturadas por un sensor de rango desde diferentes posiciones (P0 y P1). Su registro resulta en la pose relativa T que relaciona a P0 y P1. Derecha: las nubes de puntos registradas y la escena original donde fueron capturadas.

fundidad, lidar 3D, y sensores con capacidad Doppler serán descritos respectivamente en los Capítulos 3, 4, y 5.

Contribuciones

Las contribuciones más relevantes de esta tesis son:

- El desarrollo de un algoritmo de odometría para cámaras de profundidad que emplea planos característicos, siendo estos emparejados con puntos utilizando la reproyección. El método combina técnicas heredadas de las dos categorías principales de la literatura (métodos directos e indirectos, que serán explicados en el Capítulo 2) para amplificar sus ventajas y reducir sus limitaciones [28]. Trabajo publicado:

  – [28] Andres Galeote-Luque, Jose-Raul Ruiz-Sarmiento, y Javier Gonzalez-Jimenez. **Método de odometría basada en planos para cámaras de profundidad**. En *Jornadas de Robótica, Educación y Bioingeniería*, Málaga, 2022.

- Un método de odometría de lidar 3D que aprovecha el formato imagen que proporcionan los sensores modernos para recuperar el movimiento de manera eficiente mediante la imposición de la restricción de coplanaridad entre parejas de puntos y planos. Las correspondencias se calculan de manera iterativa reproyectando los planos característicos en el plano imagen [27].

  Este trabajo fue mejorado posteriormente, desacoplando la estimación del movimiento en dos pasos más simples, mejorando la eficiencia y la precisión. Esto se hace posible al aprovechar la disposición de estructuras normalmente encontrada en entornos urbanos, lo que permite que el plano del suelo sea identificado y separado de los planos verticales. Además, la extracción de planos se realiza eficientemente mediante la segmentación con *quadtrees*, consiguiendo así planos más grandes y facilitando que se cumpla la restricción de coplanaridad impuesta [29]. Trabajos publicados:

  – [27] Andres Galeote-Luque, Jose-Raul Ruiz-Sarmiento, y Javier Gonzalez-Jimenez. **Efficient 3D lidar odometry based on planar patches**. *Sensors*, 22(18):6976, 2022.

  – [29] Andres Galeote-Luque, Jose-Raul Ruiz-Sarmiento, y Javier Gonzalez-Jimenez. **GND-LO: Ground decoupled 3D lidar odometry based on planar patches**. *IEEE Robotics and Automation Letters*, 8(11):6923-6930, 2023.

- La creación de un método de odometría 3D para sensores con capacidad Doppler que hace uso de la velocidad radial capturada de los objetos del entorno para recuperar el movimiento del vehículo a partir de una única observación. El modelo cinemático del vehículo proporciona las relaciones necesarias entre las velocidades del sensor y del vehículo para estimar el movimiento 3D. El artículo que define este método se encuentra bajo revisión actualmente:

  – Andres Galeote-Luque, Vladimír Kubelka, Martin Magnusson, Jose-Raul Ruiz-Sarmiento, y Javier Gonzalez-Jimenez. **Doppler-only Single-scan 3D Vehicle Odometry**. Enviado y aceptado por *IEEE International Conference on Robotics and Automation*, Yokohama, 2024. Accessible en https://arxiv.org/abs/2310.04113.

Marco de la tesis

Esta tesis es el resultado de cuatro años de labor investigadora como miembro del grupo MAchine Perception and Intelligent Robotics (MAPIR[1]) del Departamento de Ingeniería de Sistemas y Automática[2] de la Universidad de Málaga. La financiación para este trabajo ha sido concedida por el programa "Ayudas para contratos predoctorales para la formación de doctores/as 2018" del gobierno de España (PRE2018-085026), en relación al proyecto WISER (DPI2017-84827-R), centrado en la creación y aprovechamiento de mapas semánticos por robots móviles. La odometría de rango, el núcleo de esta tesis, se incluía como una de las líneas de investigación de dicho proyecto, como parte del algoritmo de localización.

Además, como miembro del grupo MAPIR el autor ha participado en otros proyectos relacionados con la robótica móvil, en concreto HOUNDBOT y ARPEGGIO. El primero pretende avanzar hacia la creación de mapas de gas con un robot móvil que porte una nariz electrónica, para posteriormente encontrar la fuente de gas en el entorno. El segundo, titulado "Percepcion robótica avanzada para Construcción de Mapas y Localización", pretende desarrollar nuevas técnicas y mejorar métodos ya existentes en estas líneas de investigación, desde las perspectivas geométrica y semántica.

Durante esta etapa, el autor ha completado el Programa de Doctorado en Ingeniería Mecatrónica de la Universidad de Málaga. Este programa ha dotado al autor de conocimiento en un abanico de temas incluidos en el campo multidisciplinar que es la mecatrónica (mecánica, electrónica, programación y automática), estableciendo unos cimientos firmes para el desarrollo de la tesis.

[1] mapir.isa.uma.es
[2] uma.es/isa

Para complementar su educación académica, el autor ha realizado una estancia de tres meses en el Centre for Applied Autonomous Sensor Systems (AASS)[3] en la Universidad de Örebro, con el grupo Mobile Robotics & Olfaction (MRO)[4]. Durante esta estacia, la investigación se ha centrado en la odometría aplicada a sensores de rango con capacidad Doppler, y el Capítulo 5 incluye los resultados de la colaboración en esta temática con el Dr. Martin Magnusson (supervisor y jefe del grupo) y el Dr. Vladimír Kubelka.

Estructura de la tesis

Los capítulos restantes de esta tesis se organizan de la siguiente manera:

- El **Capítulo 2** proporciona una introducción de los sensores de rango y la odometría para una mejor comprensión de los próximos capítulos. En él se analizan diferentes sensores, junto con la tecnología que les permite percibir la distancia a objetos de su entorno. Existen distintos enfoques a la hora de usar esta información geométrica para calcular la pose relativa del sensor en el tiempo. Primero se describen algunos métodos genéricos de registro de nubes de puntos, por su capacidad de estimar la pose relativa entre ellas. Después estudiamos los métodos de odometría creados específicamente para ciertos sensores de rango, los cuales presentan un mejor rendimiento al tomar en consideración las características y limitaciones de cada sensor.

- El **Capítulo 3** aborda la odometría para cámaras de profundidad, y el método desarrollado en esta línea. Su principal ventaja reside en la combinación de enfoques indirectos y directos en un único método que extrae planos característicos de las imágenes de entrada, pero evita el proceso de cálculo de correspondencias entre ellos. En vez de eso, cada plano se empareja con un punto de la otra imagen mediante la reproyección de su centroide. Tras la primera estimación del movimiento, esta puede ser retroalimentada al sistema para actualizar el emparejamiento entre elementos característicos, mejorando por lo tanto la precisión de la solución.

- El **Capítulo 4** describe dos nuevos métodos de odometría creados para lidars 3D. El primero se basa en el uso de planos característicos y la reproyección iterativa para recuperar el movimiento del sensor. Incluye dos propuestas de procedimientos para calcular la planitud alrededor de cada punto, cada uno priorizando la precisión o la velocidad sobre la

[3] oru.se/english/research/research-environments/ent/aass
[4] mro.oru.se

otra. El segundo método se desarrolla en base al primero, introduciendo la estimación del movimiento en dos pasos desacoplados, al diferenciar planos pertenecientes al suelo o a las paredes. El resultado consigue mejorar tanto la eficiencia como la precisión, ambas cualidades deseadas en algoritmos de odometría.

- El **Capítulo 5** presenta el método de odometría de una sola observación desarrollado para sensores de rango con capacidad Doppler. Estos sensores, además de la información geométrica de la escena, también perciben la velocidad radial de cada punto capturado. Dado que la velocidad observada de los objetos estáticos es la opuesta a la del sensor, esta información puede ser utilizada para estimar la velocidad 3D de un vehículo que lleve el sensor. Además, empleando la información contenida en una única observación, a diferencia de los algoritmos tradicionales de registro de nubes de puntos.

- El **Capítulo 6** proporciona una conclusión a esta tesis, resumiendo la labor investigadora presentada y aportando nuevos objetivos para futuras investigaciones.

Conclusión y líneas futuras

Esta tesis se ha centrado en el problema de la localización de robots móviles desde la perspectiva de la odometría empleando sensores de rango, en particular cámaras de profundidad, lidar 3D y sensores con capacidad Doppler.

La localización es uno de los pilares esenciales que permiten a los robots móviles completar sus tareas de manera correcta y segura. De entre las múltiples formas disponibles para obtener la localización del robot, cabe destacar la odometría. En función de la información sensorial capturada, la odometría puede proporcionar una estimación del movimiento. Además de su uso directo, también puede jugar un rol importante como parte de los algoritmos de SLAM, los cuales construyen un mapa del entorno para mejorar la precisión a largo plazo. La odometría puede aplicarse a diferentes sensores, pero la información geométrica capturada por los sensores de rango presenta ciertas ventajas: es menos sensible a los cambios de iluminación y simplifica tareas como puede ser la evitación de obstáculos. En esta tesis hemos propuesto métodos de odometría para diferentes sensores de rango, aprovechando las cualidades únicas de cada uno.

El primer sensor abordado fueron las cámaras de profundidad en el Capítulo 3. En él, se propuso un método de odometría que aprovecha las superficies planas encontradas en entornos interiores de forma habitual, lugar donde mejor funcionan estos sensores. El método emplea la representación de la nube de

puntos en formato imagen, combinando técnicas de enfoques directos e indirectos. En resumen, se extraen planos característicos, y estos se emparejan con puntos de la otra imagen a través de la reproyección, en vez del habitual cálculo de correspondencias y el coste computacional que ello conlleva. La estimación del movimiento es válida siempre y cuando las parejas de elementos característicos pertenezcan a la misma superficie de la escena. Para aumentar las probabilidades de que se cumpla esta hipótesis, el emparejamiento es actualizado de forma iterativa en el paso de refinamiento, haciendo uso de la última estimación de la pose relativa. El método resultante consigue operar con una mayor precisión que la competencia, además de mantener un funcionamiento en tiempo real.

Este método de odometría fue posteriormente adaptado para su funcionamiento con lidars 3D. Estos sensores se instalan frecuentemente en vehículos autónomos por su amplio campo de visión, largo alcance, y nubes de puntos densas. En el Capítulo 4 presentamos los dos métodos de odometría propuestos, los cuales aprovechan la representación en formato imagen de las nubes de puntos que proporcionan los sensores modernos. El primero de los dos métodos recupera el movimiento haciendo uso de parches planos extraídos de las nubes de puntos. Para la correcta selección de los parches, se calcula previamente la imagen de planicidad de la escena, que representa cómo de plano es el entorno alrededor de cada píxel. Los parches se extraen de los píxeles que presentan la menor curvatura según la imagen de planicidad. Los parches seleccionados se emparejan posteriormente con puntos de la otra imagen, y la pose relativa se estima como la transformación rígida que minimiza la distancia entre parejas de elementos característicos, asumiendo que cumplen la restricción de coplanaridad. Los emparejamientos se actualizan de forma iterativa en función de la última estimación del movimiento, mejorando así la precisión de la solución. El segundo método presentado construye sobre la base creada por el anterior, desacoplando la estimación del movimiento en dos pasos, lo que resulta en una mejora de la eficiencia y la precisión. Esto se hace posible al categorizar los parches extraídos como pertenecientes al suelo o a paredes. Los parches del suelo se combinan para crear un plano del suelo, el cual se compara con el anterior plano suelo para estimar tres de los seis grados de libertad del movimiento 3D. El movimiento restante se recupera utilizando los parches de las paredes, emparejándolos con puntos de la otra imagen. Para mejorar la extracción de planos característicos, se aplica segmentación de quadtrees a la imagen de planicidad, lo que produce un conjunto de parches de diferentes tamaños localizados en las zonas más planas de la escena.

El último método de odometría, descrito en el Capítulo 5, funciona con sensores de rango con capacidad Doppler. Estos sensores son capaces de medir de manera precisa la fase de la señal de regreso, lo que les permite capturar la velocidad radial de los puntos de la escena. El uso de esta información ha demostrado ser útil en métodos de odometría, por ejemplo añadiendo restricciones a enfoques tradicionales, o incluso aportando una estimación del

movimiento del sensor. Nuesto método aprovecha las velocidades radiales para estimar la velocidad del vehículo que lleva el sensor con capacidad Doppler. En vez de emplear una IMU para observar el movimiento 3D como es habitual en la literatura, hacemos uso del modelo cinemático del vehículo. El resultado es un método de odometría rápido con una elevada precisión a corto plazo. Se puede utilizar directamente para recuperar el movimiento del vehículo, o como inicialización de algoritmos de localización más complejos que se beneficien de conocer una estimación previa del movimiento.

Hay varias rutas que podrían ser investigadas en trabajos futuros relacionados con el tema de odometría de rango. Primero, es importante destacar la importancia que tienen la optimización y el multi-threading. El tiempo de ejecución y los recursos computacionales disponibles son límites constantes en la operación de los algoritmos de odometría. Por ejemplo, el omnipresente ICP tiene problemas a la hora de procesar grandes nubes de puntos en tiempo real, por lo que sus variantes acostumbran a disminuir la resolución de las nubes de puntos, perdiendo información en el proceso. En una situación ideal en la que se tiene acceso a un suministro ilimitado de recursos computacionales y tiempo, la precisión aumentaría, ya que se usaría más información, esta se procesaría mejor, y se podrían hacer más iteraciones. La optimización y multi-threading pueden ayudar a los métodos de odometría en esta dirección, mejorando la localización resultante.

Ahora que ya hemos abordado la mejora más básica y obvia, es el turno del SLAM. La odometría es útil e interesante por sí misma, pero integrarla en algoritmos de SLAM implica producir una mejor localización. El cierre de bucle, así como la comparación de nubes de puntos con un mapa, además de la odometría, son los componentes más significativos del SLAM, que permiten mantener una alta precisión a largo plazo, con poco o ningún drift. Proponemos la creación de algoritmos de SLAM para los métodos de odometría presentados en esta tesis, que aprovechen sus cualidades como el uso de parches planos de diferentes tamaños, o la segmentación del plano del suelo.

Otra limitación común para la mayoría de métodos de odometría de rango es el conocido como el problema de apertura. Es posible que se de el caso de escenarios donde los sensores de rango no capturen información suficiente para que el movimiento sea observable por completo. Ejemplos del problema de apertura incluyen túneles y pasillos, en los que el movimiento del sensor a lo largo de ellos puede ser indeterminado. Para resolverlo, se puede estudiar el uso de otros sensores, como una IMU o una cámara, para aportar más información del movimiento, reduciendo la probabilidad de caer en el problema de apertura.

Finalmente, nos gustaría incluir la segmentación como otro trabajo futuro relacionado con la odometría. Los algoritmos de segmentación tienden a ser lentos, limitando la capacidad de los métodos de odometría de funcionar a tiempo real. Sin embargo, la segmentación del entorno proporciona información de alto nivel, que puede resultar útil para mejorar la precisón y eficiencia de la odometría, como se demostró en el segundo trabajo del Capítulo 4.

1 Introduction

We humans, when presented with two pictures of the same environment from slightly different positions, can roughly estimate said position difference (called *relative pose*). But what if, instead of the color pictures we are used to dealing with, we compare two point clouds (sets of points) captured from a different type of sensor? Well, see Figure 1.1 and check for yourself if you can find the transformation that would make both point clouds coincidental. Computers, and robots by extension, on the other hand, have a harder time finding the correct solution to this problem, but when they do it is far more accurate than what is humanly possible. Now let's think bigger. Let's think of a robot equipped with one of these *range sensors* that captures point clouds from its environment. By calculating the relative pose between each pair of consecutive point clouds, the results can be composed to recover the movement of the sensor (and thus the robot) over time, that is, its trajectory. This problem is what *range sensor odometry* aims to solve, and in this thesis we describe the different odometry algorithms we developed for various types of range sensors.

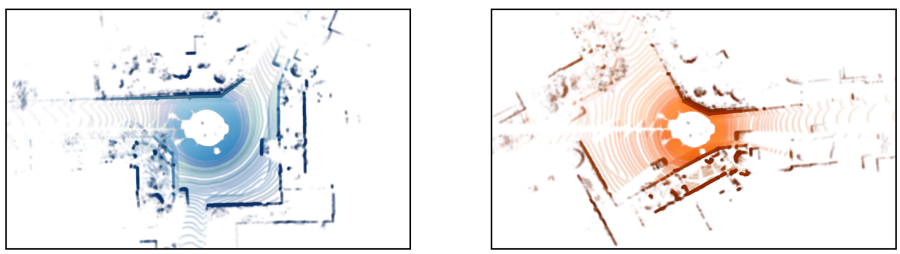


Figure 1.1: Two point clouds of the same environment captured by the same sensor at different locations. Can you guess the relative pose between them?

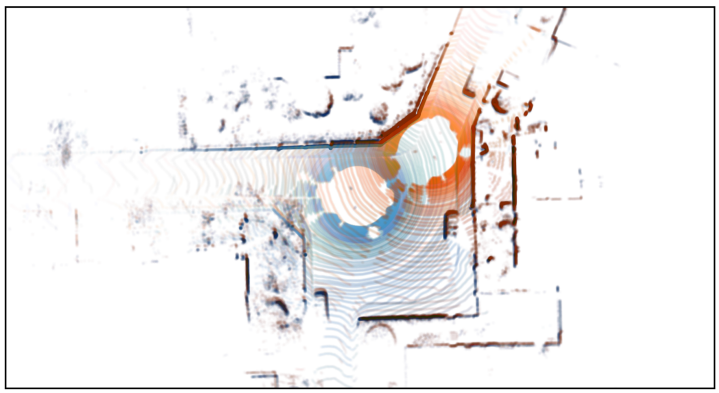


Figure 1.2: Solution to the question from Figure 1.1, both point clouds can be represented overlapping if the relative pose between them is known.

Self-localization in their surroundings is essential for mobile robots and autonomous vehicles to correctly and safely perform their duties. This localization can be calculated employing different sensors and techniques [63, 68]. However, range sensors stand out for their ability to provide geometric information of the environment, which is helpful for localization but also obstacle avoidance and map building, among others. Range sensor odometry in particular can recover the movement of a range sensor, and thus localize the robot in the scene. This in itself is very valuable, but it can be taken to the next level of accuracy by simultaneously mapping the environment. This technique, known as Simultaneous Localization and Mapping (SLAM) [9], is employed by a large number of systems because of its accuracy even after long periods, and odometry plays an important role in it.

This thesis contributes various odometry algorithms developed for different types of range sensors. The proposed methods, be it on their own or used as the front end of a SLAM pipeline, can provide valuable localization for mobile robots, enabling them to complete safe motion and higher-level tasks.

By the way, did you correctly answer the question from Figure 1.1? Check the solution in Figure 1.2.

1.1 Motivation

The title of this thesis comes from the merging of two concepts, range sensors and odometry. Let us then describe both terms separately for a better understanding of the resulting combination. Range sensors are those with the ability to measure the distance to objects around them, meaning the position of the observed points is known relative to the sensor. This geometric information is usually referred to as point cloud, since they represent a group of points in space. Different range sensor types employ different technologies and physical phenomena to measure said distance. One way to obtain this information, employed by depth cameras like the pioneer Kinect [110], is to project a certain pattern on the scene, and measure how it gets deformed by the different shapes. However, the most common approach is to measure the time it takes for an emitted signal to return to the sensor, as in time-of-flight depth cameras, lidars and radars, as seen in Figure 1.3. By modifying this last principle to measure the phase of the returning signal, the velocity of objects in the scene can be also calculated, making it a Doppler-capable range sensor.

As for the term odometry, it consists of estimating the trajectory followed by a sensor only based on its incremental observations [63]. When applied to range sensors, it can be seen as finding the relative pose between two consecutively captured point clouds, since composing the poses between pairs of inputs results in the trajectory of the sensor. A wide variety of approaches to estimating said relative pose can be found in the literature, and they generally try to register both point clouds together [41], meaning making them overlap. See Figure 1.4 for a basic example. Since odometry estimates relative poses, the errors of the composed motion estimation accumulate over time, which is known as drift [83]. To avoid this issue, the input point clouds could be registered to a global map instead of the previous one, but this requires having a map of the area which might not always be available. A popular technique to overcome this is the so-called Simultaneous Localization and Mapping (SLAM) [9], in which the observed point clouds are compared to a map of the scene that is being created while localizing. SLAM algorithms still need to localize the sensor on the map, and they often use an odometry method as the front end to provide a high-frequency motion estimation. It is thus important for odometry algorithms to be accurate, but also efficient such that they allow higher-level tasks (such as map building and loop closure) to have available the necessary computational resources for their correct operation.

A detailed explanation of range sensors and odometry can be found in Chapter 2. The developed odometry algorithms for depth cameras, 3D lidars and Doppler-capable sensors are described in Chapters 3, 4 and 5 respectively.

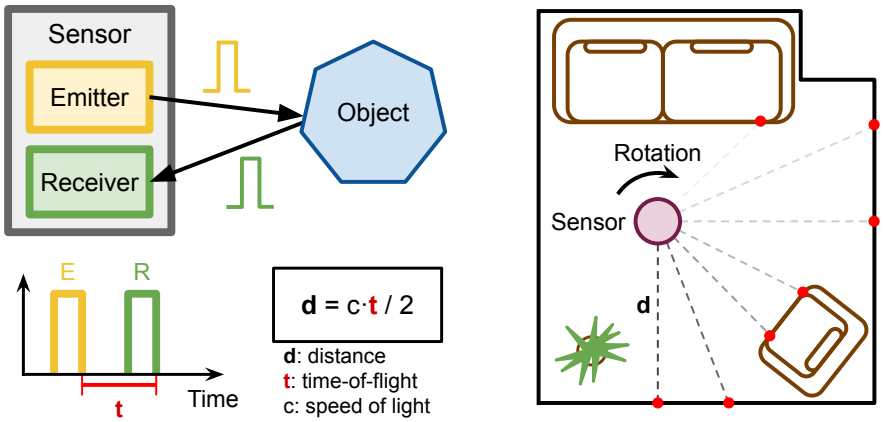


Figure 1.3: On the left, working principle of time-of-flight sensors. The emitter sends a signal, and the distance to an object is calculated based on the time it takes for the receiver to capture the reflected signal. On the right, a rotatory sensor measuring distances at different angles as it spins. This results in a point cloud (group of red points) containing the geometric layout of the environment.

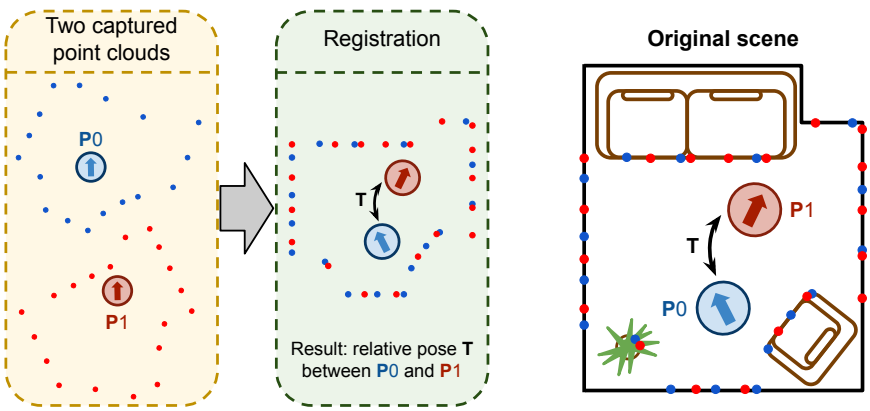


Figure 1.4: Left: two point clouds of the same room captured from different positions (P0 and P1) by a range sensor. Registering them together results in finding the relative pose T between P0 and P1. Right: the registered point clouds and the scene from where they were captured.

1.2 Contributions

The most relevant contributions of this thesis are:

- The development of an odometry algorithm for depth cameras that uses planar features, matching them with points based on reprojection. It combines techniques from the two main categories of the literature (direct and indirect methods, as will be explained in Chapter 2) to reinforce their benefits and reduce their limitations [28]. Published work:

  – [28] Andres Galeote-Luque, Jose-Raul Ruiz-Sarmiento, and Javier Gonzalez-Jimenez. **Método de odometría basada en planos para cámaras de profundidad**. In *Jornadas de Robótica, Educación y Bioingeniería*, Málaga, 2022.

- A 3D lidar odometry method that exploits the image format of modern sensors to efficiently recover the motion by imposing the coplanarity constraint on point-plane pairs. Correspondences are calculated iteratively by reprojecting the 3D planes into the range image provided by the sensor [27].

  This work was later improved by decoupling the motion estimation into two simpler steps, further improving efficiency and accuracy. This is possible due to the special layout of structures found in urban settings, which allows the ground plane to be identified apart from vertical planes. Additionally, the use of quadtrees efficiently extracts bigger planes, aiding in the fulfilling of the coplanarity constraint [29]. Published works:

  – [27] Andres Galeote-Luque, Jose-Raul Ruiz-Sarmiento, and Javier Gonzalez-Jimenez. **Efficient 3D lidar odometry based on planar patches**. *Sensors*, 22(18):6976, 2022.

  – [29] Andres Galeote-Luque, Jose-Raul Ruiz-Sarmiento, and Javier Gonzalez-Jimenez. **GND-LO: Ground decoupled 3D lidar odometry based on planar patches**. *IEEE Robotics and Automation Letters*, 8(11):6923-6930, 2023.

- The creation of a 3D odometry method for Doppler-capable sensors that leverages the captured radial velocity of objects in the scene to recover the movement of the vehicle from a single observation. The kinematic model of the vehicle provides the necessary relations between the sensor and the vehicle velocities to estimate the 3D motion. The article defining these developments has been submitted and is currently under review:

  – Andres Galeote-Luque, Vladimír Kubelka, Martin Magnusson, Jose-Raul Ruiz-Sarmiento, and Javier Gonzalez-Jimenez. **Doppler-only Single-scan 3D Vehicle Odometry**. Submitted and accepted by

IEEE International Conference on Robotics and Automation, Yokohama, 2024. Accessible via https://arxiv.org/abs/2310.04113.

1.3 Thesis framework

This thesis is the result of four years of research work as a member of MAchine Perception and Intelligent Robotics (MAPIR[1]) group of the Systems Engineering and Automation Department[2] of the University of Málaga. Funding for this work has been granted by the "Ayudas para contratos predoctorales para la formación de doctores/as 2018" program of the Spanish government (PRE2018-085026), related to the WISER project (DPI2017-84827-R), focused on the creation and exploitation of semantic maps by mobile robots. Range odometry, the core of this thesis, was included as one of the research lines of this project, as the front end of the localization pipeline.

In addition, as member of MAPIR group the author has participated in other projects related to mobile robotics, namely HOUNDBOT and ARPEGGIO. The former aims to advance towards the creation of gas maps with a mobile robot equipped with an electronic nose, to then find the gas source within an environment. The latter, titled "Advanced Robotic Perception for Mapping and Localization", intends to improve existing methods as well as develop new approaches for both mapping and localization, from both the geometric and semantic perspectives.

During this period, the author completed the doctoral program in Mechatronics Engineering coordinated by the Systems Engineering and Automation Department of the University of Málaga. This program has granted the author knowledge on the wide range of topics included in the multidisciplinary field of mechatronics (mechanics, electronics, programming and automation), setting a strong foundation for the development of this thesis.

To complement his academic education the author has carried out a research stay of three months at the Centre for Applied Autonomous Sensor Systems (AASS)[3] at the University of Örebro, with the group Mobile Robotics & Olfaction (MRO)[4]. During this stay, the research centered on odometry for Doppler-capable range sensors, and Chapter 5 includes the results of the collaboration on this topic with Dr. Martin Magnusson (supervisor and head of the group) and Dr. Vladimír Kubelka.

[1] mapir.isa.uma.es
[2] uma.es/isa
[3] oru.se/english/research/research-environments/ent/aass
[4] mro.oru.se

1.4 Thesis organization

The remaining chapters of this thesis are organized as follows:

- **Chapter 2** gives an overview of both range sensors and odometry for a better understanding of the following chapters. Different sensors are analyzed, along with the technology that allows them to perceive the distance to objects of the scene. When this geometric information is used to calculate the relative pose of the sensor over time, several approaches can be followed. First, some generic point cloud registration methods are described, as they can estimate the relative pose between two point clouds. Then we study odometry algorithms created specifically for certain range sensors, which improve the performance by taking into account the traits and limitations of each sensor.

- **Chapter 3** addresses depth camera odometry, and the method developed in this regard. Its main advantage resides in combining both indirect and direct approaches into one hybrid method that extracts planar features from the input images, but avoids solving for correspondences between them. Instead, each plane is paired with a point of the opposite image by reprojecting its centroid. Once the first estimation of the motion has been calculated, it can be fed back into the algorithm to update the feature pairs, thus improving the accuracy of the solution.

- **Chapter 4** describes two new methods created for 3D lidar odometry. The first one is based on using planar features and iterative reprojection to recover the movement of the sensor. In it, two procedures are proposed to obtain the flatness of each point, each one prioritizing accuracy or speed over the other. The second method builds upon the previous one by decoupling the motion estimation into two steps, differentiating ground from wall planes. The result is an improvement both in efficiency and accuracy, both desired characteristics for odometry algorithms.

- **Chapter 5** presents the single-scan odometry method for Doppler-capable range sensors. These sensors, apart from the geometric information of the scene, also perceive the radial velocity of each captured point. Since the observed velocity of static objects is the opposite of the sensor's, this data can be used to estimate the 3D velocity of a vehicle that is equipped with the sensor. All from the information contained in a single scan, as opposed to traditional point cloud registration algorithms.

- **Chapter 6** provides a conclusion to this thesis, summarizing the research work presented and providing new objectives for further research.

2

Range sensor odometry

Estimating the incremental pose of a range sensor through its consecutive observations is known as range sensor odometry. *In this chapter, we describe various approaches to tackle this problem, as well as analyze how different types of range sensors have shaped the literature. Inside the scope of this thesis, we include odometry applied to depth cameras, lidars and Doppler-capable sensors.*

2.1 Introduction

To provide the necessary background for the content of this thesis, focused on 3D range sensor odometry, it is vital to understand the term *odometry*. In a nutshell, it means to estimate the relative change in pose from sensory data [63], but the approach can vary considerably depending on the sensor used. One of the most common and early examples comes from what is known as *wheel odometry* [16], which employs encoders in the wheels of a vehicle. Based on the measured angle of rotation of the wheel, and knowing its radius, it is possible to estimate its motion. Not only proprioceptive sensors (those that measure internal values of the system) like encoders or Inertial Measurement Units (IMU) can feed the odometry. Nowadays exteroceptive sensors (those that acquire information about the environment) are widely mounted on mobile robots and employed in odometry algorithms. This includes cameras, with the

extensively studied Visual Odometry (VO) [66] which will be of relevance later, and of course the main topic in this thesis, range sensors.

2.2 Range sensor odometry and point cloud registration

Range sensors provide geometric information about the scene in the form of point clouds (PtC) which, despite being hard for humans to interpret, carry crucial data about the sensor's surroundings. Knowing the spatial layout of the environment can assist in obstacle avoidance or the creation of maps, among others.

For the sake of clarity, let's use 2D lidar, one of the earlier range sensors and a very popular device, as an example throughout this section. And let's visualize said sensor mounted on a robot that is navigating inside a room. These lidars can capture 2D point clouds by measuring the distance to points located all around them by continuously spinning their measurement unit (recall Figure 1.3). At fixed time intervals, the sensor provides a new point cloud, denoted as S_k with k being the counter. Since it is moving, point cloud S_k is captured from pose $P_k \in \mathrm{SE}(2)$ within the room, and the next one S_{k+1} from $P_{k+1} \in \mathrm{SE}(2)$. These PtCs will represent the same environment (the room) but from different points of view. The objective of an odometry method is to find the rigid transformation $T \in \mathrm{SE}(2)$ that defines the relative pose between P_k and P_{k+1} employing the information found in the PtCs S_k and S_{k+1} (recall Figure 1.4). How said transformation is extracted from the two input point clouds is the question at hand, and we will describe different approaches found in the literature in this chapter. However, there is one method that has dominated the field of point cloud registration for years, and is still relevant to this day: the Iterative Closest Point (ICP) algorithm [7].

ICP's working principle, as shown in Algorithm 2.1, consists of matching each point from one PtC to its closest counterpart in the other PtC, then finding the rigid transformation that minimizes the distance between each pair, and finally bringing the PtCs closer by applying the estimated transformation. These steps are repeated until the solution converges, creating a simple yet effective method. Despite having earned its popularity, ICP has two main limitations that hinder its performance. The first one is related to the computational cost of having to calculate all the possible distances between the points from both PtCs, since it scales drastically with bigger input point clouds. The second issue is the possibility to fall into local minima instead of converging to the real solution (global minimum), which means it can output unnoticed incorrect estimations of the transformation.

Algorithm 2.1: Iterative Closest Point basic algorithm.

Input: Point clouds S_k, S_{k+1}
Output: Relative pose $T \in \mathrm{SE}(3), T := (R, \vec{t})$
repeat
  $M \leftarrow \emptyset$ `//set of matches;`
  foreach point $\vec{p}_i \in S_k$ **do**
    $d \leftarrow \emptyset$ `//array of distances;`
    foreach point $\vec{q}_j \in S_{k+1}$ **do**
      $d \leftarrow \{d \cup ||\vec{p}_i - \vec{q}_j||^2\}$;
    end
    $j^* \leftarrow \arg\min_j d$ `//closest point from` S_{k+1} `to` $\vec{p}_i$;
    $M \leftarrow \{M \cup \{\vec{p}_i, \vec{q}_{j^*}\}\}$;
  end
  $T \leftarrow \arg\min_{(R,\vec{t})} \sum_{l \in M} ||\vec{p}_l - (R\vec{q}_l + \vec{t})||^2$;
  $S_{k+1} \leftarrow T(S_{k+1})$ `//transform` S_{k+1} `using` T `and repeat;`
until T converges;

There is another method that rivals ICP worth highlighting among point cloud registration, called Normal Distribution Transform (NDT) [8]. NDT converts one input point cloud (known as target or model) into a set of normal distributions that encode the probability of finding a point at a certain position, providing a continuous representation of the model which facilitates the estimation of the optimal transformation. The first step in NDT is to subdivide the model point cloud into a grid of cells, where a different normal distribution is adjusted based on its belonging points. See Algorithm 2.2 for a more detailed explanation. Compared to ICP, NDT provides an increased efficiency for two reasons. First, the computationally expensive point matching is removed, and each point is simply related to the normal distribution of the cell it currently occupies. Second, the memory use of storing a point cloud as a simple set of normal distributions is much lower. Regardless of its obvious advantages, it also presents some limitations: reducing the PtC to a set of distributions, although efficient, comes with losing some information from the source data, thus affecting accuracy. But most importantly, the grid representation comes at the cost of having discontinuous cost functions that can hinder the optimization.

Every odometry approach mentioned until now, including wheel odometry, shares the characteristic of estimating the relative pose incrementally. In the previous example, you can recover the trajectory followed by the 2D lidar by composing the relative transformations obtained from each pair of consecutive observations. Pose composition is usually symbolised with $\oplus$, so the current pose of the sensor could be defined as $P = P_1 \oplus P_2 \oplus \cdots \oplus P_k$. Thus, any error

Algorithm 2.2: Normal Distributions Transform basic algorithm.

Input: Point clouds S_k, S_{k+1}
Output: Relative pose $T \in \mathrm{SE}(3), T := (R, \vec{t})$

```
// Create grid from S_k;
w ← size of the grid;
B ← grid(S_k, w) //divide into groups of points using a grid;
// Obtain normal distribution from each group in B;
D ← ∅      //set of normal distributions related to B;
foreach group b_i ∈ B do
    n ← size(b_i);
    c_i ← Σ(b_i)/n      //centroid of the group of points;
    e_i ← b_i − c_i;
    C_i ← (e_i e_i^⊤)/n      //covariance of the group of points;
    D ← {D ∪ {c_i, C_i}}      //normal distribution of b_i;
end

// Find the relative pose iteratively;
repeat
    M ← ∅      //set of matches;
    // Find matches for every point in S_{k+1};
    foreach point q_j ∈ S_{k+1} do
        d_i ← find closest from D;
        M ← {M ∪ {d_i, q_j}};
    end
    // Minimize distances between pairs and update S_{k+1};
    T ← arg min_(R,t) Σ_{l∈M}((Rq_l + t) − c_l)C_l((Rq_l + t) − c_l)^⊤;
    S_{k+1} ← T(S_{k+1});
until T converges;
```

in the pose estimation is consequently accumulated over the whole trajectory, also known as *drift* [83], reducing the long-term accuracy. A solution to this problem could, for example, consist of comparing the input sensory data with a global map, instead of the previous instance of data. This raises the obvious question of "what map?", to which SLAM systems answer "just create your own". SLAM stands for Simultaneous Localization and Mapping [9], and goes a step ahead of odometry by creating a map with the sensor information to which the new observations are compared, thus providing not an incremental pose estimation, but a global one. Note that SLAM algorithms are far more complex than odometry, because they have to deal with localizing the input data within the map, updating the map with the new information, and other

possible critical tasks like detecting loop closures. Although this thesis will focus on range sensor odometry and not SLAM, it is important to establish the direct connection between these two concepts, since SLAM systems usually employ some type of odometry as its localization front-end.

2.3 Sensor specific odometry

We have now laid the foundations of range sensor odometry and its inevitable connection to point cloud registration. It is thus time to detail how these methods have progressed over time, and the impact different types of range sensors have had on these advances. For that reason, and following this thesis' structure, we will begin with depth cameras.

2.3.1 Depth cameras

The name of these sensors hints at the close relation they have with traditional color cameras. Depth cameras generate depth images, where each pixel holds the value of the depth from the sensor to the observed object, instead of the color or intensity as in traditional cameras and images. The technology and physics leveraged to capture this information from the scene depend on the type of sensor, among which we highlight and describe two because of their prevalence [17].

First, structured light cameras use a projector to illuminate the scene with a known pattern. By examining the distortion of the pattern, the geometry of the scene can be observed (see Figure 2.1 for an example). A famous example of this type of depth camera is the Microsoft Kinect [110], which projects an irregular point pattern in the infrared spectrum, invisible to the human eye but observable to its camera. As a consequence, these sensors can still operate in total darkness, but as a downside, it is highly sensitive to sunlight. Additionally, multiple sensors operating in the same area would interfere with each other. Note that these sensors usually have an additional color camera, thus providing simultaneously depth and color images of the environment, and can be referred to as RGB-D cameras. At the core of structured light cameras lies a camera that captures the projected pattern, and thus the usual operating frequency of these sensors is in the interval of 20-60Hz, depending on the resolution.

Second, time-of-flight cameras follow a similar approach of emitting infrared light and capturing its interaction with the objects in the scene. The main difference resides in their use of pulses instead of a pattern. By measuring the time since the emitter was triggered until the pulse was perceived by the sensor, the pixel-wise depth can be calculated [36, 52]. This technology makes

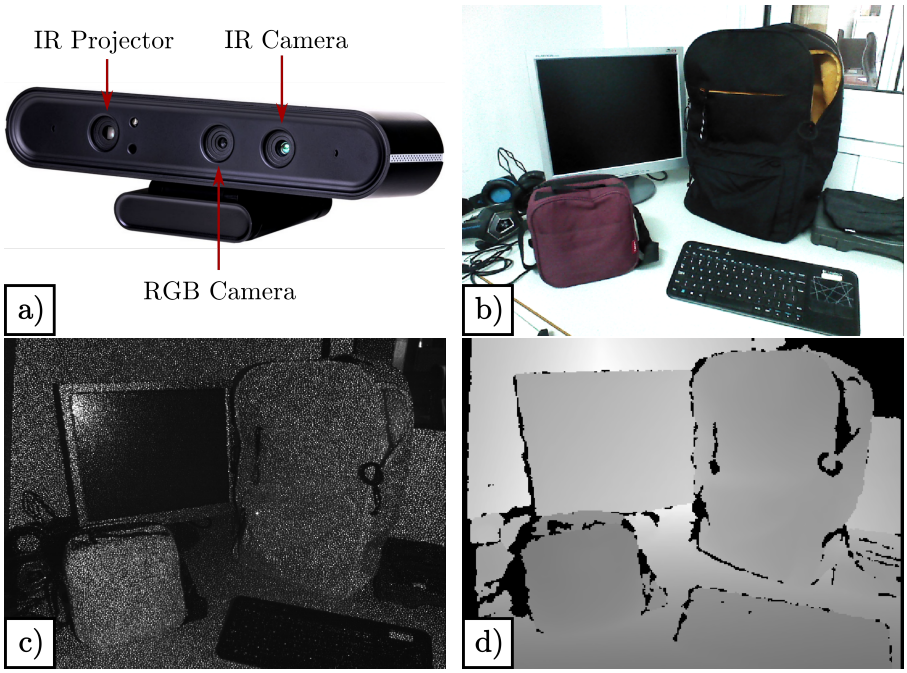


Figure 2.1: Structured light RGB-D camera (a)[1], and the RGB (b), infrared (c) and depth (d) images it captures. Note the infrared pattern in (c).
[1] Source: https://www.orbbec.com/products/structured-light-camera/astra-series/

for lower frequency sensors compared to structured light cameras, and shares most of its limitations: they are sensitive to sunlight and can interfere with other sensors operating in the same area.

Regardless of the technology, depth cameras provide depth images, which can be interpreted as ordered point clouds. Instead of storing the geometric information of the scene as a list of points with no specific order, depth images provide an extra layer of information in the form of the pixel coordinates of each point on the image plane. Of course, methods like ICP can still be used with depth images, but works in the literature mainly focus on exploiting the image representation for improved efficiency and accuracy. Points located close to each other on the image plane are probably also close in the 3D space. This is helpful, for example when finding the closest neighbors of a point, which can be a burdensome task unless accelerated with the use of k-d trees [6] or similar techniques, but becomes trivial by selecting the neighboring pixels in the depth image plane. It was only normal to see how research for odometry applied to depth cameras grew along with this sensor by exploiting this interesting characteristic, inheriting techniques common in image

processing and, most importantly, visual odometry. This has created two main categories of odometry approaches which are also found in visual odometry: *indirect* and *direct* methods [22]. At first, the use of the geometric information was minimal, just acting as assistance to the VO applied to the radiometric images [21,24,40]. But, with time depth images started getting more attention and becoming more important in the odometry pipeline.

Indirect methods, also known as feature-based, are the most widespread in the literature and operate by reducing the input images to a set of features. Different types of features are employed in the literature, including points [70], the combination of points and lines [56], and planes [13,73]. Regardless of the features chosen, correspondences between them are then calculated, and the relative pose of the sensor is estimated by minimizing the distance between paired features. Finding the right correspondences between the two sets of features is essential for these methods to yield an accurate motion estimation, and usually most of the computational resources are poured into this process. When employing plane features, they are usually extracted from the image through some type of plane segmentation, which are complex algorithms that can increase the execution time.

Direct methods, on the other hand, avoid the preprocessing necessary to extract the features by minimizing a cost function that compares the input images directly, inspired by methods like the one presented by Engel et al. [22]. See Figure 2.2 for a visual summary of the difference between indirect and direct methods. Steinbrucker et al. [91] applied this concept to RGB-D cameras, minimizing the radiometric error and using the geometric information only to reproject the images. This work was extended by Kerl et al. [48] by introducing a probabilistic formulation, increasing the robustness of the odometry.

Jaimez and Gonzalez-Jimenez [42] distanced themselves from previous works by discarding the color image, applying the range flow restriction densely to the depth image. This resulted in a method known as DIFODO which provides a high accuracy odometry in a very efficient manner. DIFODO relies on the range flow constraint:

$$\frac{dZ}{dt} = \frac{\partial Z}{\partial t} + \frac{\partial Z}{\partial u}\frac{du}{dt} + \frac{\partial Z}{\partial v}\frac{dv}{dt} + O\left(\Delta t, \frac{du}{dt}, \frac{dv}{dt}\right), \tag{2.1}$$

where (v, u) represent the pixel coordinates within the depth image Z. This equation decomposes the change in depth over time (dZ/dt) into the pixel-wise depth change ($\partial Z/\partial t$), and the derivatives over the image plane ($\partial Z/\partial u$, $\partial Z/\partial v$), along with components of lower order ($O(\Delta t, du/dt, dv/dt)$). The motion of the sensor can be included in this equation through its relation with the other elements ($dv/dt, du/dt$), allowing every pixel to provide one constraint on the movement.

In Chapter 3 we will describe the odometry method we propose for RGB-D cameras. Its novelty comes from combining both indirect and direct approaches

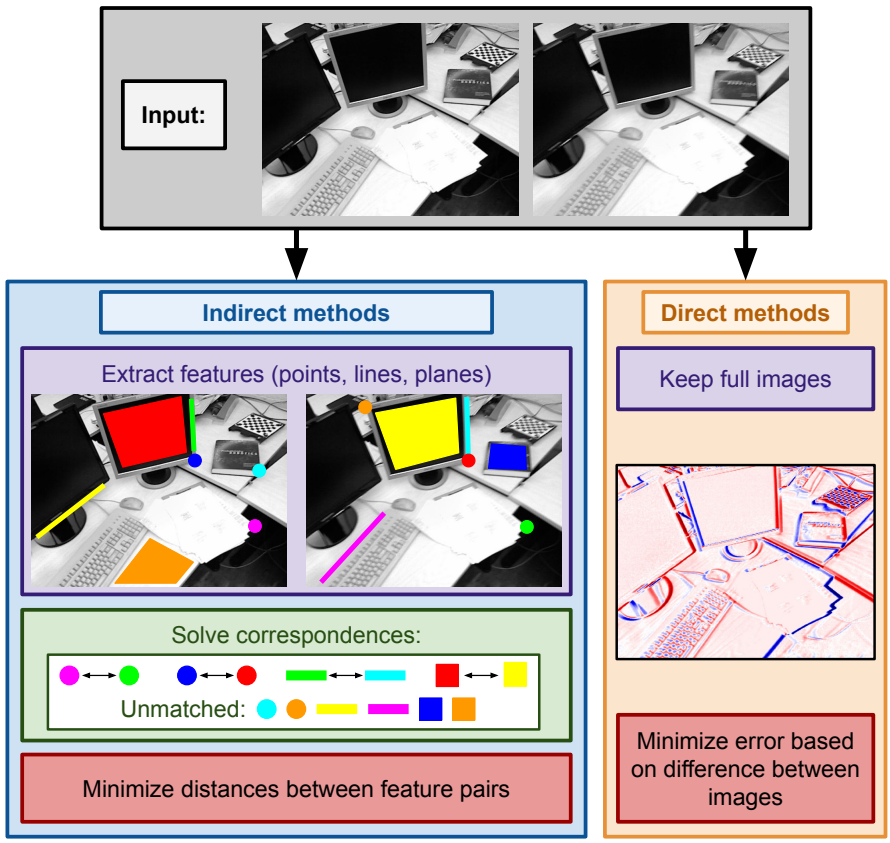


Figure 2.2: Difference between indirect and direct methods. The former reduces the input images to a set of features, like points, lines, planes, or any combination of them. Correspondences between features of both images are found, and finally, the relative pose is estimated by minimizing the distance between feature pairs. Direct methods keep the full images, and obtain the relative pose by minimizing an error function related to the difference between the images. Note that grayscale images are used in this figure for the sake of clarity, but the same principles can be applied to depth images.

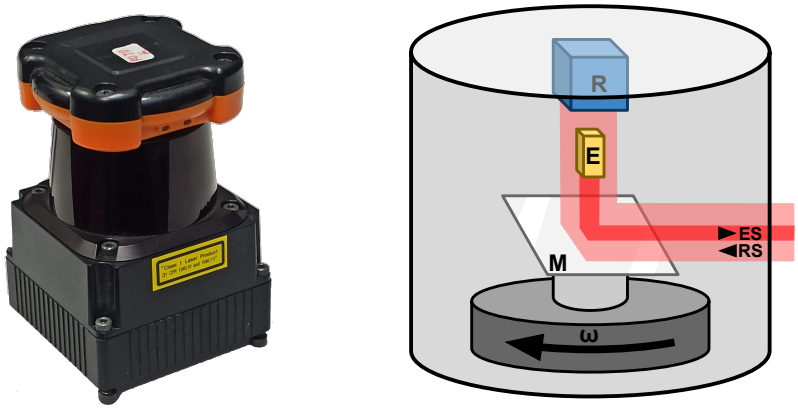
to enhance their advantages. Feature planes are employed, but they are not matched as in traditional feature-based methods. Instead, taking inspiration from direct algorithms, the matches are calculated by reprojecting the centers of the planes onto the other image. The relative pose is estimated as the rigid transformation that minimizes the point-to-plane distance between pairs of features. In a later refinement step, these matches are updated with the latest estimation of the motion, improving the pairing and thus the accuracy of the localization. It is worth highlighting that, despite using planar features, no burdensome segmentation is applied. Instead, the pixel-wise flatness is calculated, and features are selected in flat areas of the scene.

2.3.2 Lidar

Lidars, an acronym for Light Detection and Ranging, are the next range sensor analyzed in this thesis and one of the most widespread. They work similarly to time-of-flight cameras by measuring the time it takes for an emitted laser signal to be received back after bouncing off an object in the scene [71, 80]. These sensors are also influenced by ambient light or reflectivity but not as much as depth cameras [63, 110], which combined with their large operating range make them suitable for both indoor and outdoor environments.

Early lidar models had the sensing unit spinning at high speed, thus allowing us to measure the distance to objects located within a 2D plane in a wide field of view, up to 360^{o}. See Figure 2.3 for a visual representation. With every revolution, a 2D lidar provides the spatial layout of its environment, usually known as *scan*. Although having only geometric information contained within a plane limits the 3D perception of the environment, there exists a significant amount of use cases in which it is more than sufficient. For example, robots moving in indoor settings usually perform planar movement, and knowing the layout of the surroundings at a certain height is enough for them to operate properly. To perform odometry with 2D lidars it is again a matter of registering two consecutive scans (2D point clouds). Apart from the omnipresent ICP and NDT, there exist other authors in the literature that tackle odometry in their own way, like Gonzalez and Gutierrez [31], Diosi and Kleeman [19], and Jaimez et al. [43]. Given the relatively low price of these sensors, and their use in robots moving along flat surfaces, 2D lidars are still relevant to this day, with various existing odometry and mapping implementations freely available and widely used like GMapping [33], RF2O [44], or AMCL [57, 58].

As technology advanced, lidars evolved into 3D lidars, which basically consist of an array of 2D sensors mounted at different angles, producing 3D point clouds. For a quick comparison between the data captured by 2D and 3D lidar see Figure 2.4. The autonomous vehicles industry recognized the potential of these sensors and started employing them, as seen in projects like Argoverse [12,100], Waymo [23,93], and notable datasets like KITTI [30] and KAIST Urban [45]. This relationship kept the 3D lidars improving because of the great



(a) Hokuyo UTM-30LX 2D lidar. (b) Basic components of a rotating 2D lidar.

Figure 2.3: A real 2D lidar sensor example (a) and a diagram showing its working principle (b). A laser emitter (E) sends a signal that bounces off a rotating mirror (M). The emitted signal (ES) leaves the sensor, and returns (RS) after reflecting off some surface of the environment. This is captured by the receiver (R), measuring the distance to the object based on the elapsed time since the signal was emitted. Since the mirror is constantly rotating (ω), the sensor can measure the distance to surrounding objects along multiple rays contained in a 2D plane.

traction autonomous vehicles were having at the time. Not only did the sensors become smaller, cheaper, and more reliable, but some models were introduced that could provide ordered 3D point clouds that could be represented in image format. Even more, modern models can also capture the intensity or reflectivity, adding the grayscale information of each observed point. See Figure 2.5 for a couple of examples. Odometry applied to this information is dominated again by ICP (actually, it was designed for 3D point clouds back in 1992), and some other variants that follow the same general pipeline but add some improvements. Examples of this are the point-to-plane ICP [15], which minimized point-to-plane distance instead of the point-to-point in the original ICP; generalized ICP (GICP) [84] combines both of these distances into a single probabilistic formulation; and multi-channel GICP [86] which adds more information (like color) as different channels. On the contrary, NDT was originally designed for 2D data, until Magnusson et al. [59] adapted the method for 3D point clouds, also extensively comparing it to ICP [60]. Over the years more odometry methods for 3D lidar odometry have been developed, and two main categories stand out in the literature. First are the ICP-based methods, which employ some variant of ICP as part of their pipeline and improve upon

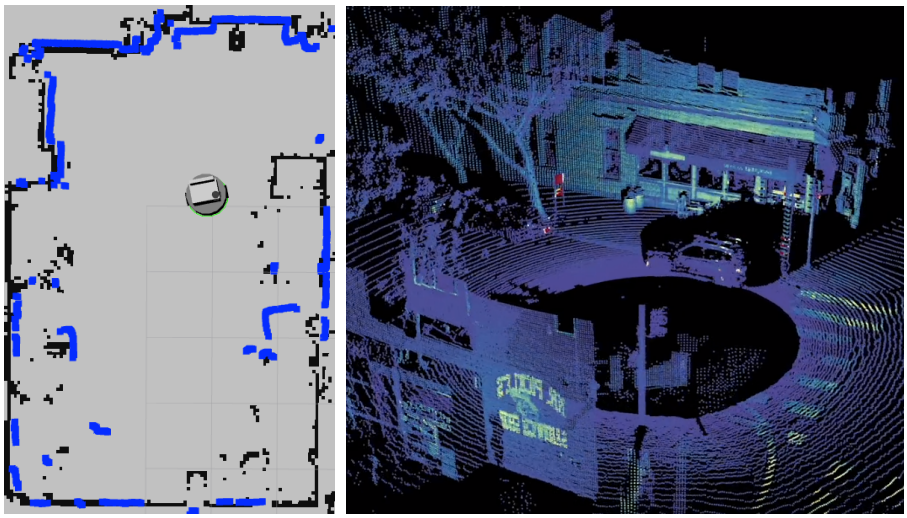


Figure 2.4: On the left, a point cloud captured by a 2D lidar (blue) over a map of the surroundings. On the right[1], a point cloud provided by the Ouster 3D lidar, where even the reflectivity of the objects is captured. Note the enormous progress seen by lidar technology.

[1] Source: https://ouster.com/downloads/lidar-sample-data

it with different contributions, like ELO [111], LOCUS 2.0 [76], SGLO [55], and Reinke et al. [75]. Next in popularity are feature-based methods, with notable examples as Velas et al. [96] for its use of collar line segments, and DiLO [34] for performing bundle adjustment with keypoints located on edges and planes along multiple consecutive scans. Finally, we would like to include in this review 3D lidar odometry methods product of the rise of deep learning in recent years, with works like D3DLO [1], DMLO [54], LO-Net [53] and EfficientLO-Net [98].

As explained before, performing odometry yields a high drift on long trajectories. As an approach to tackle this issue appeared Lidar Odometry and Mapping (LOAM) [108], similar to SLAM since it performs a high frequency and low fidelity odometry as the front end, while a slower backend refines the localization according to a map. When registering the input point clouds to a global map, the drift gets drastically reduced, since the motion estimation is not purely relative and error does not accumulate that much. Note that both terms "LOAM" and "SLAM" are often used interchangeably in the literature regarding lidar localization, both create a map while operating and use it to improve the odometry estimation, sometimes adding loop closure on top of it all. LOAM algorithms can be categorized based on their odometry frontend into ICP-based and feature-based methods just as explained before.

Figure 2.5: Example images captured by the Ouster OS1-128 3D lidar[1]. Top: range image, with closer points appearing darker. Bottom: intensity image, a black and white picture of the scene.

[1] Source: https://static.ouster.dev/sdk-docs/sample-data.html

The former includes Velodyne SLAM [65], SuMa [5], and LiTAMIN2 [106]; and examples of feature-based algorithms are V-LOAM [109], LeGO-LOAM [87], IMLS-SLAM [18], Grant et al. [32], MULLS [67], PaGO-LOAM [85], and LIO-SAM [88]. Some of the listed methods improve their performance by segmenting the ground from the rest of the point cloud. Examples of this include LeGO-LOAM [87] and PaGO-LOAM [85], which separate ground features from the rest. Similarly, ELO [111] applies ICP separately to ground and non-ground points. The work of Chen et al. [14] uses ground segmentation to reject non-planar features on the ground.

In Chapter 4 we will explain the two 3D lidar odometry methods proposed in this thesis. They exploit the image representation of the point clouds provided by modern sensors to efficiently extract planar features (patches). Each patch is iteratively matched with a point from the other image via reprojection using the latest estimation of the motion, unlike traditional feature-based approaches. The movement is estimated by minimizing the point-to-plane distance between feature pairs, and it converges as the matching gets updated. The second method proposed in Chapter 4 goes one step further and decouples the motion estimation in two stages. This is possible by segmenting the ground plane from the set of patches. Unlike other methods in the literature that employ the ground plane, we not only decouple the motion estimation, but do so in a very efficient manner, combining all planar patches into a unique plane.

2.3.3 Doppler-capable range sensors

The last type of range sensor in our list is Doppler-capable range sensors, but before that we have to discuss radars. These sensors observe the distance to objects of the scene by measuring the time it takes for emitted radio waves to bounce back into the radar [90]. They are often overlooked because of their low accuracy and working frequency compared to other range sensors. Additionally, certain artifacts are expected to appear in the observed point clouds,

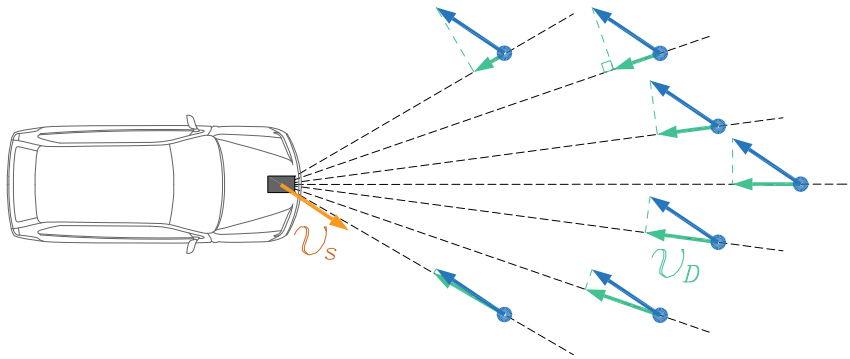


Figure 2.6: A Doppler-capable sensor mounted on a vehicle can perceive the radial velocities $\vec{v}_D$ of objects in the scene. Assuming the scene is static, this velocity is the opposite of the sensor velocity $\vec{v}_s$ projected along the radial direction.

like nonexistent objects produced by the signal reflecting off corners or walls. However, their technology allows them to operate correctly even in challenging conditions like extreme lighting or rain, or in the presence of mist and dust [2]. This makes them particularly suitable for underground operations and construction sites, among other harsh environments.

Similar to lidar, early models worked by scanning their surroundings in 2D. Almost as important as the point cloud registration was a preprocessing step to rid the data of noisy measurements and artifacts, as seen in works like Rohling [79] and Adolfsson et al. [2]. Nowadays there exist 3D radars as well, but there is another development fruit of radar technology progress arguably more important. We refer to the ability of some radars to capture the radial velocity of objects of the scene by precisely measuring the phase of the returning signal, usually known as Doppler velocity [101]. See Figure 2.6 for a graphical representation of the captured Doppler velocity. Note that this technology has also been implemented in certain lidars, so from now on we will refer to all of them as Doppler-capable sensors.

It is fairly common for these sensors to employ Frequency Modulated Continuous Wave (FMCW) technology to capture both the range and radial velocity of objects [102], although it is not the only approach available. They consist of emitting a FMCW, and then analyzing the received reflected signal. The received signal will be delayed compared to the emitted signal based on the distance from the sensor to the observed object. Furthermore, if the object is moving the frequency of the received wave will be different because of the Doppler effect, higher for approaching objects and lower for those moving away from the sensor. See Figure 2.7 for a visual explanation of this concept.

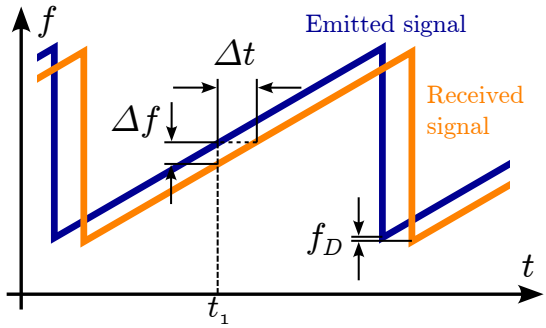


Figure 2.7: Working principle of FMCW radar. The emitted and received signals (blue and orange respectively) are compared. The distance to the reflecting object can be calculated from the difference between the frequencies of the signals at a certain time Δf. The radial velocity is related to the frequency shift f_D. Note that the *sawtooth* modulation has been represented, but other shapes can be employed.

The radial velocity of points from the scene provided by these Doppler-capable sensors has proven to be advantageous for odometry methods, aiding in the segmentation of dynamic and static objects [49], as well as introducing more constraints to the movement estimation [38]. Given the right conditions, it can be as powerful as to recover the 2D ego-motion of the sensor from a single scan of its surroundings [46, 47]. Note that, to estimate the localization, these works rely upon either knowing the kinematic model of the vehicle or having multiple sensors mounted on the robot, since not all three degrees of freedom (DOFs) of a 2D movement are observable with just the radial velocity of points of the scene.

The first publication to recover the ego-motion of a vehicle employing Doppler velocity without data association is due to Yokoo et al. [105], who employed 1D radar sensors mounted on a vehicle with Ackermann steering performing 2D planar motion. They propose two sensor configurations to recover the forward and angular velocities, one with two radars mounted at the front of the vehicle, and another with a single radar and a gyroscopic sensor. Note that this method requires the observed objects to be static to provide a reliable localization estimation. Kellner et al. expanded this concept for a single 2D radar [46]. Since the sensor can only measure the radial velocity of the objects in the direction of the line sensor-object (recall Figure 2.6), the angular velocity cannot be directly observed. To circumvent this issue, again the vehicle has to comply with the Ackermann kinematic model, reducing the DOFs from 3 (complete 2D motion) to 2 (forward and angular velocities). The increase in data points due to leveraging a 2D sensor (compared to Yokoo et al. [105]) removes the need to assume a static scene. By applying RANSAC [25], dy-

namic points can be identified and discarded as outliers. Kellner et al. later developed a method to completely recover the 3 DOFs of the 2D motion by employing multiple radar sensors [47], avoiding the Ackermann model requirement. Stepping into 3D movement, works of both Kramer et al. [50] and Doer and Trommer [20] combine the information from a 3D radar and an IMU to recover the 6 DOFs of the motion, employing batch optimization by the former and an Extended Kalman Filter (EKF) by the latter. The use of an IMU as a way to make the movement observable is reminiscent of the second configuration proposed by Yokoo et al. [105]. Similarly, Yoon et al. [107] employ a Doppler-capable 3D lidar and a gyroscope to decouple the 3D motion estimation. Each sensor provides information on a part of the motion, with the lidar and gyroscope yielding the translation and rotation respectively. Said work includes a study of the observability of the motion, explaining the need for either a gyroscope or multiple sensors to completely recover the 3D movement. Even without a 3D radar, Park et al. [69] estimate the 3D motion of a vehicle by compositing two orthogonal 2D radar sensors, along with an IMU.

As well as providing a motion estimation, the Doppler velocity has proven to be helpful in more traditional odometry algorithms that perform data association, typically registering two consecutive point clouds. The most common approach to using the Doppler velocity is to add a cost function to the optimization problem, where the measured radial velocity is compared to the expected velocity of a point given an estimation of the motion. This can be applied to registration problems commonly found in the literature such as ICP [7] variations [38, 78, 104], NDT [8, 74], and alike approaches [4]. It is worth highlighting the work of Monaco and Brennan [64] stands out for decoupling the motion estimation: the translation is obtained from the Doppler velocity, while the collected spatial information provides the rotation. The recent rise of Neural Networks (NN) has also reached radar odometry as can be seen in the work of Rennie et al. [77], where Doppler velocity is used to obtain an estimation of the motion which can later be fused with the scan registration result.

In Chapter 5 we will describe the 3D odometry method we propose for Doppler-capable sensors. It estimates the ego-motion of a vehicle equipped with said sensor by leveraging only the radial velocities captured from the scene. Instead of using an IMU as is common in the literature, we include the vehicle kinematic model to make the movement observable. Our method is efficient and short-term accurate, making it a suitable candidate for initialization for more complex localization algorithms.

2.4 Odometry evaluation and benchmarks

To finish this chapter, we will now analyze the existing algorithms for evaluating and comparing odometry methods. Two main characteristics are desirable for odometry methods: *efficiency* and *accuracy*, and the goal of this section is to establish the necessary knowledge to measure them.

The first one, efficiency, is as important as in any other computer program or process. It is optimal to leave as many computational resources free for other processes running alongside, enabling the computer to handle as many programs as possible or to allocate resources to the more critical ones. This is especially important in SLAM systems as they include many computationally expensive processes operating simultaneously. Closely related to the efficiency is the operation frequency of the odometry methods. Having the motion estimation available more often results in a finer localization of the robot over time, which is again significant for SLAM algorithms. The speed of an odometry method is easy to measure, it is a matter of measuring the time it takes to estimate the movement from a pair of input point clouds. Note that said measure will be severely influenced by the computer employed. This means that for comparisons, ideally, all methods should be tested in the same PC, although this is not always feasible. Since range odometry is based on the processing of pairs of consecutive point clouds, there exists a threshold milestone in the execution time. If this time is longer than the time it takes the sensor to capture two consecutive measurements, then the method will have to discard some input point clouds. Not only is there a loss of information, but the compared point clouds will be farther apart in time and space, further hindering the motion estimation. On the other hand, if the operating frequency of the odometry is higher than that of the sensor, then it will be able to process all inputs. This is known as real-time operation (sometimes as online), and it is very important for odometry methods. The real-time threshold depends of course on the employed sensor, and is usually around 10-30Hz for range sensors.

The second characteristic, accuracy, is arguably more important than efficiency. Although it depends on the specific use case, it is usually better to have a very precise localization at a low frequency than the opposite. Measuring the accuracy of the trajectory produced by an odometry method is more nuanced than speed. For that reason, several approaches can be found in the literature, each with its properties and uses. But there is a common ground shared by all evaluation techniques: knowing the true trajectory that the odometry was tasked with recovering, known as *ground truth*. Intending to provide a common evaluation setup, there exists public odometry datasets and benchmarks. They include the sensory data captured along the trajectory of a robot, and its ground truth. That way, multiple odometry and SLAM algorithms can evaluate in the same conditions and easily compare between them.

Figure 2.8: Motion capture system used in the TUM RGB-D dataset [92]. Left[1]: multiple cameras mounted on the wall pointed at the measuring zone. Right[1]: Kinect depth camera with reflective markers. By tracking the markers seen by the cameras, the pose of the sensor can be retrieved.

[1] Source: https://cvg.cit.tum.de/data/datasets/rgbd-dataset

2.4.1 Obtaining the ground truth

The act of obtaining the ground truth in itself is a complex topic, since its accuracy is essential for the later evaluation of odometry methods. A common approach is to employ a very precise SLAM algorithm using the data captured by the range sensors. Since the goal is to recover the trajectory of the robot with as much accuracy as possible, this can be done offline and using as many resources as needed. This approach is popular because it does not need extra sensors mounted on the robot, and can work in nearly all scenarios. On the downside, the accuracy of the resulting ground truth depends considerably on the SLAM algorithm chosen, and can be influenced by noise, outliers, etc. The Velodyne SLAM dataset [65] provides an example of this approach.

A better alternative comes from employing motion capture to acquire the trajectory of the robot. It consists of placing exteroceptive sensors in the environment and observing the movement of the robot. The most popular configuration includes setting infrared reflectors on the robot, allowing a group of infrared cameras in the environment to capture its pose over time. These systems are capable of providing a very accurate ground truth of the trajectory, but their complex setup reduces their usability. Not only are they very expensive, also they have a limited range of action which prevents them from capturing longer trajectories. Because of these qualities, motion capture is a powerful technique relegated to short indoor trajectories. An example of a dataset that uses motion capture to provide the ground truth of the trajectory is the famous TUM RGB-D dataset by Sturm et al. [92], shown in Figure 2.8.

Another widely used approach is the use of Inertial Navigation Systems (INS) and Real Time Kinematics (RTK) Global Navigation Satellite Systems (GNSS) like GPS, GLONASS, or Galileo. By combining the information from

both IMU and GNSS, these systems achieve high accuracy, but are not available indoors or in cluttered urban environments. They are often employed in outdoor datasets, as in the popular KITTI benchmark [30]. Other works combine this approach with SLAM techniques to mitigate the limitations, like the NCLT [11] and KAIST complex urban [45] datasets.

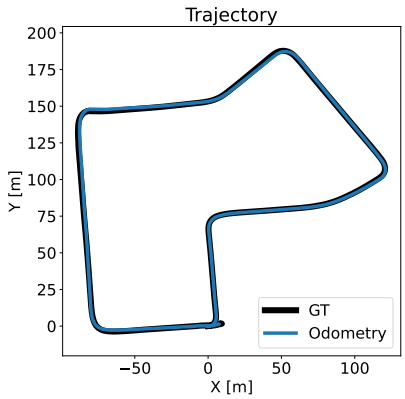
2.4.2 Measuring accuracy

Once the ground truth is available, we can measure the accuracy of the trajectory estimated by an odometry method. A trajectory is a collection of poses and their timestamps. Even when the estimation and the ground truth have the same number of poses and the same timestamps (which is usually not the case), there is no easy and direct way of calculating the difference (or error). From the different approaches that can be found in the literature, there are two that stand out as the most used: Relative Pose Error (RPE) and Absolute Trajectory Error (ATE).

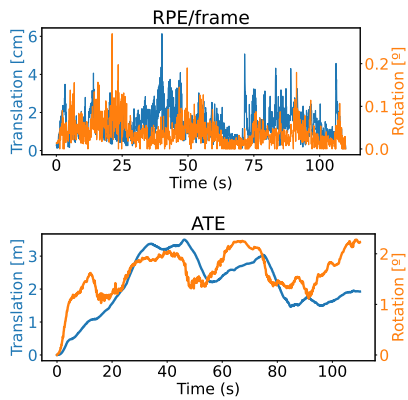
RPE measures the local accuracy over a fixed time interval Δ. It does so by comparing the pose change of the odometry trajectory over the time interval with that of the ground truth. We will now refer to the odometry trajectory as the set of poses $P_1, ..., P_n \in \mathrm{SE}(3)$, and the ground truth as $Q_1, ..., Q_n \in \mathrm{SE}(3)$, both synchronized in time. We can then define the pose difference between them E_i from instance i to $i + \Delta$:

$$E_i := \left(Q_i^{-1} Q_{i+\Delta}\right)^{-1} \left(P_i^{-1} P_{i+\Delta}\right). \tag{2.2}$$

Note that we can obtain the pose difference E_i for $m = n - \Delta$ instances of the trajectory. Since $E_i \in \mathrm{SE}(3)$ is a 4×4 rigid transformation matrix, it cannot be used directly to measure the error. The translation error $\delta_{t,i}$ can be extracted from E_i as the norm of its translational components, and the rotation error can be calculated as the angle of the rotational components $\delta_{r,i} = \arccos\left((\mathrm{tr}(R) - 1)/2\right)$ (R represents the rotation matrix of E_i, and $\mathrm{tr}(R)$ its trace). With both sets $\delta_{t,1:m}, \delta_{r,1:m}$ of m values, the overall accuracy of the odometry method over the whole trajectory can be computed as the mean, median, or root mean squared error (RMSE) of the sets. Although RMSE is a popular choice, we believe it penalizes outliers disproportionately [99]. It is worth highlighting the importance of the Δ chosen, as it defines the time interval along which the error is measured. Low and high Δ values relate to short- and long-term accuracy measurement, respectively. For odometry methods the most common Δ is one frame, hence measuring the average error (or median or RMSE) yielded every frame. See Figure 2.9 for an example of a trajectory and its RPE per frame over time. On the contrary, SLAM algorithms focus on long-term accuracy and often calculate the RPE per second or even bigger time intervals. Sturm et al. [92] even propose to measure the average RPE over all possible Δ values, from 1 to n.



(a) Trajectory of an odometry method compared to the ground truth.



(b) RPE per frame (above) and ATE (below) over time.

Figure 2.9: The trajectory estimated by an odometry method (a), along with its RPE per frame and ATE over time (b). The RPE per frame measures the short-term accuracy, and thus changes frequently. ATE on the other hand focuses on the long-term accuracy, measuring the absolute pose difference and capturing the drift over time.

Absolute Trajectory Error (ATE), as its name implies, measures the absolute pose error at every point in time. Assuming both trajectories (odometry P and ground truth Q) are aligned, the ATE at time i can be computed as follows:

$$F_i := Q_i^{-1} P_i. \tag{2.3}$$

Similar to RPE, the translational and rotational errors are computed from the 4×4 matrix F_i. This calculation can be repeated for every time i, to then obtain the RMSE, mean, or median error over the whole trajectory. Figure 2.9 shows an example of the ATE over time of a trajectory. This metric is rarely employed in odometry evaluation because it focuses on long-term accuracy, unfairly penalizing drift (error accumulated over time) and outliers. For that reason, Lee and Civera [51] proposed an alternative error metric called Discernible Trajectory Error (DTE), which preprocesses the trajectories based on their alignment and scale to increase robustness to outliers.

Although popular, these are not the only accuracy metrics available to evaluate odometry methods. It is common for benchmarks to propose their own evaluation metric. For example, the notable KITTI dataset [30] proposes measuring the RPE over all possible subsequences of 100, 200, ..., 800 meters found in the trajectory. For the overall accuracy, these can be composed based on their associated length to obtain the error per meter of trajectory.

Additionally, other conclusions can be drawn from this data, like the possible correlation between length and error, or similarly between speed and error.

The following chapters will include examples of these accuracy metrics, since we evaluate every odometry method we present and compare it to approaches from the literature. The most common metric employed to this end is the mean RPE, mainly per frame but also per second. This selection focuses on the short-term accuracy, the goal of odometry algorithms, ignoring the unavoidable drift. Without loop closing or comparing point clouds to the map, the best way to reduce the drift created by an odometry method is to improve the accuracy of its individual motion estimations, hence the selected metric. When evaluating on public benchmarks, their proposed accuracy metric is employed to enable a fair and easy comparison with other odometry methods, as is the case with the KITTI dataset used in Chapter 4.

3

Depth camera odometry

This chapter covers the odometry computation of a depth camera (RGB-D) in real-time, using only the depth information. For that, a method is proposed that offers efficiency and higher accuracy than other alternatives found in the literature. To that end, the motion is recovered by minimizing the point-to-plane distance between pairs of features, leveraging the information provided by the flat surfaces of the scene abundantly found in manmade indoor environments. The proposal includes an iterative approach used to refine the solution (rigid transformation between two consecutive images) by updating the matching between the features until convergence. The method has been tested and compared with a state-of-the-art method, resulting in a reduction of 25% of the median of the translational and rotational error, while working at the same frequency (∼30Hz).

3.1 Introduction

Depth cameras excel at providing dense ordered point clouds of the scene at a relatively high frequency. These features make them suitable as input for odometry methods to recover the ego-motion. Especially in indoor environments, where the distances are usually small and the sun is less likely to interfere with their correct performance. Indoor environments are characterized by having

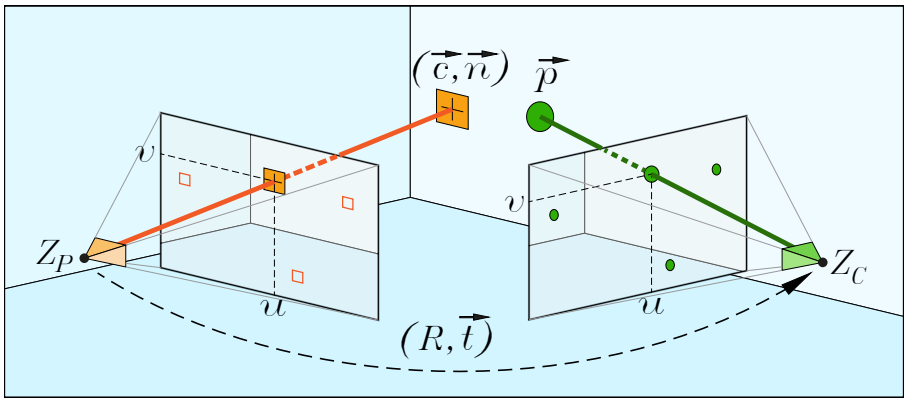


Figure 3.1: Two consecutive depth images from a depth camera (Z_P and Z_C) and the features employed in the presented method. The planes (orange squares), defined by their center $\vec{c}$ and normal vector $\vec{n}$, are extracted from image Z_P, and are matched with points $\vec{p}$ (green dots) from the same pixel coordinates (v, u) of image Z_C. The motion is estimated as the rigid transformation $(R, \vec{t})$ that minimizes the point-to-plane distance between feature pairs. The result is then improved by iteratively updating the corresponding points based on the relative pose.

abundant flat surfaces, a fact that can be exploited by odometry methods to achieve better accuracy.

The literature regarding depth camera odometry can be categorized, similarly to Visual Odometry [66], into indirect and direct approaches. The first group, also known as feature-based, consists of reducing the input depth images to a set of features (like points, lines or planes), which are then paired. The movement is calculated as the rigid transformation that minimizes the distance between pairs. Direct methods on the other hand focus on optimizing a cost function related to the difference between both input images. Despite the clear advantages shown by both types, they are not without limitations.

In this chapter, we present a depth odometry method that combines techniques from both direct and indirect algorithms. As in indirect approaches, our method employs planar features (*patches*) extracted from the input images, focusing the computational resources in the most informative parts of the scene. However, it avoids the burdensome correspondence solver often found in indirect algorithms, opting for an iterative projection-based matching, similar to direct approaches. This proposed hybrid method outperforms state-of-the-art competitors, as proven by experimental evaluation, while keeping real-time performance. See Figure 3.1 for a visual representation of the basic principle behind this method.

3.2 Related work

Although ICP can be applied as an odometry method for depth cameras (as established in Chapter 2), these sensors capture ordered point clouds in the form of images, which enables the use of techniques inherited from visual odometry that enhance the performance of odometry methods. In fact, as commented, the existing approaches can be categorized into two main groups, indirect and direct methods, just like in visual odometry.

Indirect methods are the most widespread and they rely on reducing the input images to sets of features. The type of feature employed varies with each method, with works using points [70], a combination of lines and points [56], and planes [13, 73]. The correct operation of these approaches largely depends on the selection and matching of features. For that reason, most of the computing resources are poured into solving correspondences. Furthermore, methods that use planar features typically apply a burdensome segmentation algorithm to find them, considerably increasing the execution time.

Direct methods avoid the process of selecting and matching features by working directly with the images. Steinbrücker et al. [91] present this concept for depth cameras, in which the movement is recovered by minimizing a cost function related to the radiometric difference between the input images, and the geometric information is used only for reprojection. This work was later extended introducing a probabilistic formulation to increase the robustness [48]. Jaimez and Gonzalez-Jimenez [42] proposed a new angle by discarding the color image and applying the range flow constraint over the whole depth image. This method outperforms its competitors both in accuracy and speed.

The odometry method proposed in this chapter combines techniques from both categories to enhance their advantages. Although it is based on the use of planar features, they are extracted efficiently, and no correspondences are calculated. Instead, the matching point of each plane is obtained via reprojection.

3.3 Method overview

In this section, we summarize how the proposed method estimates the movement of the RGB-D camera along a sequence of depth images. Thus, the goal is to obtain the relative pose (rigid transformation) of the sensor between two consecutive images, defined by a rotation matrix $R \in \mathrm{SO}(3)$ and a translation vector $\vec{t} \in \mathbb{R}^3$. The input data is a pair of depth images referred to as the *previous* Z_P and *current* Z_C images from now on, with $Z : (\Omega \in \mathbb{N}^2) \rightarrow \mathbb{R}$ defined in the image domain Ω. Figure 3.2 shows the pipeline of the proposed method.

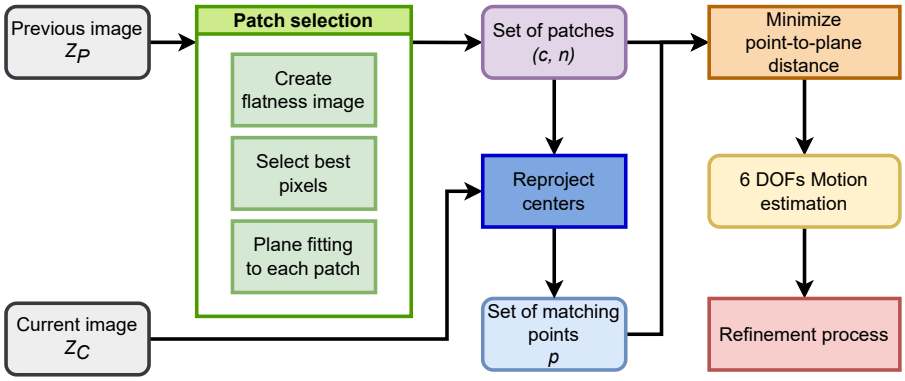


Figure 3.2: Diagram showing the workflow of the presented method. The input is a pair of depth images. A flatness image is calculated from the previous image, and it is then used to select those pixels located on flat surfaces of the scene. Features are extracted from the images using the selected pixels: planar patches and their corresponding points from the previous (Z_P) and current (Z_C) images, respectively. The movement is thus recovered by minimizing the distance between all feature pairs. Finally, an iterative process is applied to refine the solution.

The relative pose is estimated by minimizing the distance between pairs of features, consisting of planar patches extracted from Z_P and points from Z_C. For the solution to be valid, both features must belong to the same flat surface of the scene, hence a correct matching of features is essential to increase the probability of fulfilling this coplanarity constraint. To this end, we create a flatness image, which represents how flat the neighborhood around each pixel is. This image is later used to select those pixels that belong to flat surfaces of the scene. This selection procedure will be further explained in section 3.3.1.

Then, a plane is fitted to the neighborhood of each selected pixel from Z_P, defined by its center and normal vector $(\vec{c}_i, \vec{n}_i)$. Each corresponding point $\vec{p}_i$ is extracted from Z_C based on the initial relative pose between both images. In case this information is not available, the points are assumed to be located in the same image coordinates as the selected planar patches. Otherwise, the patches' centers can be reprojected onto image Z_C, thus finding the pixel coordinates from which to extract the corresponding points. The process of extracting these matching points based on the relative pose will be analyzed in Section 3.3.2.

Once the set of planes $(\mathbf{C}, \mathbf{N})$ has been extracted from Z_P, with $\vec{c}_i \in \mathbf{C}$ and $\vec{n}_i \in \mathbf{N}$, and the corresponding set of points $\mathbf{P}$ from Z_C, with $\vec{p}_i \in \mathbf{P}$, the following step consists of finding the relative rigid transformation $(R, \vec{t})$ that minimizes the distance between feature pairs. The motion estimation will be

valid if each feature pair represents the same flat surface from the scene. The pose estimation will be detailed in Section 3.3.3.

After obtaining the first motion estimation, an iterative refinement process is applied, in which the matching point of each patch is recalculated at each iteration based on the latest relative pose estimation. This increases the chance of the feature pairs fulfilling the coplanarity constraint, thus being a valid pairing. The pose is optimized at each iteration using the new feature pairs, and the process is repeated until convergence. In Section 3.3.4 we will delve into the stages of the refinement.

Lastly, in Section 3.3.5 we will analyze the limitations of this proposed method, mainly the need of having patches oriented in three orthogonal directions for the solution to be observable. It will also explain the procedure employed to tackle said limitation, in which the estimated motion is filtered leveraging previous instances.

3.3.1 Plane selection

In this section, we describe the process of plane selection and extraction, intending to reduce the input image Z_C to a set of planar patches defined by their centers $\mathbf{C}$ and normal vectors $\mathbf{N}$.

To maximize the probability of a point $\vec{p}_i$ belonging to its corresponding plane $(\vec{c}_i, \vec{n}_i)$, planar patches are extracted from certain selected pixels of the image based on the flatness of its neighborhood. The bigger the support flat area, the higher the chances that the selected pixel will belong to the same scene surface in both images, thus fulfilling the previously established coplanarity constraint. In this stage, a plane segmentation algorithm could be applied to aid in the selection of the planar patches, but these methods are usually slow and computationally expensive.

The proposed method makes use of 2D convolutions to efficiently obtain a flatness image, which stores information on how flat the surrounding of each pixel is. See Figure 3.3 for an example. This is possible since the neighborhood around a pixel in the depth image is symmetric when representing a flat surface. By measuring the difference between the depth of the central pixel and the average depth of its neighbors we can obtain a good approximation of the flatness of its surroundings. As mentioned previously, this operation can be implemented to work efficiently by applying a 2D convolution over the whole depth image, using the Laplacian kernel:

$$\frac{1}{8}\begin{bmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{bmatrix}. \tag{3.1}$$

The technique described above is applied to Z_P to create its flatness image, to then select the pixels located in the flattest areas. To ensure a correct distribution of the selected pixels over the whole image, it is first divided into

Figure 3.3: Image of a scene (left) and its corresponding flatness image (right). Light pixels represent areas in which the scene cannot be represented as a plane, examples of this being the edges of the objects.

fixed-size blocks, and the pixels with the higher flatness from each block are selected.

The next step consists of fitting a plane $(\vec{c}_i, \vec{n}_i)$ to each selected pixel from image Z_P. The center of the plane is simply the centroid of the pixel's neighborhood, and the normal vector is obtained by applying Singular Value Decomposition (SVD) to the matrix of differences between each neighbor and the centroid. Thus, the normal vector $\vec{n}_i$ is the left singular vector related to the lowest singular value. This singular value can also be used as a measurement of the fitness of the plane to the neighborhood.

Note that employing the flatness image does not guarantee that the selected patches are located in flat areas of the scene, however, its fitness can later be used to reduce the weight of features extracted from curved surfaces or occlusions.

3.3.2 Point matching

In this section, we explain the extraction of points corresponding to each selected planar patch from image Z_C, based on the initial relative pose between both input images. To fulfill the coplanarity constraint, the paired features must belong to the same surface of the scene. Since the selected patches are extracted from the flattest areas of the image, and taking into consideration the relatively small movement between two consecutive images, the points in the same pixel coordinates in both images have a high probability of belonging to the same surface of the scene. This means the point $\vec{p}_i$ associated with the patch $(\vec{c}_i, \vec{n}_i)$ can be found in the same pixel coordinates, but in the image Z_C. A Gaussian filter is applied around each point to reduce the impact of noise on later stages.

In the case where a prior estimation of the sensor motion is known, this can be leveraged to improve the matching between the point-plane pairs through reprojection. This procedure goes like follows:

1. The center of each selected patch $\vec{c}_i$ is transformed using the known motion estimation $(R, \vec{t})$ to obtain $\vec{c}_i^w$:

$$\vec{c}_i^w = [x_c^w, y_c^w, z_c^w]^\top = R^{-1}(\vec{c}_i - \vec{t}). \tag{3.2}$$

2. The projection function of the depth camera is applied to obtain the coordinates (v, u) in the image plane of the transformed center:

$$v = f_y \frac{y_c^w}{z_c^w} + v_m, \qquad u = f_x \frac{x_c^w}{z_c^w} + u_m. \tag{3.3}$$

3. These coordinates (v, u) can be used to extract the associated point $\vec{p}_i^w$, again applying a Gaussian filter as explained before.

3.3.3 Pose estimation

In this section, we detail the process of estimating the relative pose that minimizes the point-to-plane distance of each feature pair. As mentioned previously, the fitness f_i of each plane is employed to downweight regions whose surroundings show a low flatness. In addition, a weight based on the depth of the center z_c of the patch is included, since depth cameras exhibit noise in their measurements that is proportional to the distance from the object to the sensor [113]. The combined weight is thus calculated as $\alpha_i = 1 - f_i - z_c^2/25$. The Huber loss function ρ is applied to the residual to increase the robustness of the solution to outliers, which are feature pairs that do not fulfill the previously mentioned hypothesis:

$$\rho(e) = \begin{cases} \frac{1}{2}e^2 & \text{if } |e| \leq k \\ k|e| - \frac{1}{2}k^2 & \text{if } |e| \geq k \end{cases}. \tag{3.4}$$

Finally, the motion can be estimated by minimizing the point-to-plane distance between feature pairs after applying the weight and the Huber loss function, and it is optimized by Ceres' [3] implementation of the Levenberg-Marquardt algorithm:

$$(R, \vec{t}) = \arg\min_{R,\vec{t}} \sum_i \rho\left(||\alpha_i \vec{n}_i \cdot (R\vec{p}_i + \vec{t} - \vec{c}_i)||^2\right). \tag{3.5}$$

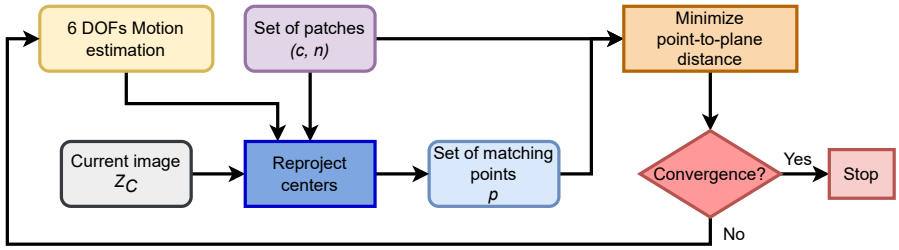


Figure 3.4: Diagram of the iterative refinement applied to the solution. The pose estimation is used to reproject the centers of the patches onto image Z_C, from which the updated matching points are obtained. The motion can be estimated again with the new feature pairs. This process can be repeated until convergence.

3.3.4 Solution refinement

Once the first estimation of the relative pose is obtained, the result is refined through an iterative process in which the matching of the features is recalculated for a better motion estimation. Figure 3.4 shows a diagram of the stages of this process. The matching is updated based on the latest estimation of the pose employing reprojection as explained in 3.3.2. In summary, the centers of the patches are reprojected on image Z_C based on the transformation $(R, \vec{t})$, and the resulting coordinates are used to extract the new corresponding points. The pose estimation is then recalculated with the new matches, applying Equation 3.5 but with the new points $\vec{p}_i^{\,w} \in \mathbf{P}^w$. This process is repeated iteratively until the solution converges.

3.3.5 Motion filter

Using point-to-plane distances allows for a more lenient matching, since it is considered valid as long as the corresponding point belongs to the same surface of the scene. However, the position of each point inside its associated plane is not constrained, which means the selected patches must include a group of orthogonal planes to correctly recover the motion. This issue can be easily observed when the captured scene is a corridor, for example, since the movement of the sensor along the corridor is not constrained. To tackle this issue, a filter has been implemented, inspired by the one used in DIFODO [42], where the previous instance of the motion estimation is leveraged when the scene is not fully constrained.

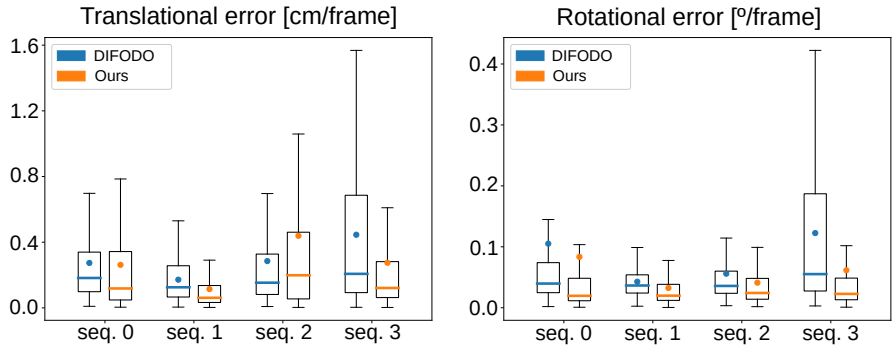


Figure 3.5: RPE per frame of both methods in the Living Room sequences of ICL-NUIM RGB-D dataset. Boxes represent central quartiles, with the colored line and dot showing the median and average RPE respectively.

3.4 Experiments and results

In this section we evaluate the performance of the proposed method on the ICL-NUIM RGB-D dataset [35], and we compare the results with those obtained by DIFODO [42]. The experiments were carried out on a PC with 8GB RAM at 2400MHz, Ubuntu 20.04, and a CPU Intel Core i7-7700HQ at 2.8GHz.

3.4.1 Dataset

The method is evaluated over the 4 indoor sequences of the virtual dataset ICL-NUIM RGB-D called *Living Room*. This benchmark stands out for its realistic sensor trajectories traversing the simulated environment, providing the camera pose ground truth without error. It also includes the usual noise found in this kind of sensor. All sequences were filmed at 30Hz, with the images having a resolution of 640×480.

3.4.2 Results

To compare the results of the proposed method with DIFODO [42], the metric known as Relative Pose Error (RPE) is employed, measured both per frame and per second for short- and long-term accuracy comparison. The execution time for both compared methods is 30ms per frame, meaning they can both reach real-time operation since the common working frequency of RGB-D cameras is 30Hz.

Figure 3.5 shows the box plot of the translational (left) and rotational (right) RPE per frame results. Note how the median and average error of the proposed method is lower than DIFODO's in every sequence, except for

	Error	Method	Sequence 0	1	2	3
Translation	[cm/frame]	DIFODO	**0.4106**	0.2259	**0.4292**	0.6851
		Ours	0.4211	**0.1823**	0.7793	**0.4588**
	[cm/s]	DIFODO	**9.9206**	4.9823	**8.9030**	15.9127
		Ours	10.1152	**4.1853**	21.2612	**11.3200**
Rotation	[°/frame]	DIFODO	0.2278	0.0518	0.0815	0.1861
		Ours	**0.2248**	**0.0471**	**0.0682**	**0.1262**
	[°/s]	DIFODO	5.4471	0.7426	1.3769	3.2798
		Ours	**3.5111**	**0.6341**	**1.3029**	**1.5959**

Table 3.1: Comparison of the RMSE of the translation and rotation error.

the translation error in sequence number 2. On average, the proposed method reduces the median of both translation and rotation RPE per frame by 25%, and the RPE per second by 40%. Table 3.1 shows the RMSE of the error on different sequences: both methods provide comparable accuracy in translation, while we consistently outperform DIFODO by 14% in rotation.

The improved accuracy obtained proven by these results translates into a better localization of the sensor along the sequence. This is because the error yielded when estimating the relative pose between a pair of images accumulates over time, which is known as drift.

3.5 Conclusions

In this chapter, we described the proposed method that quickly and efficiently recovers the motion of a RGB-D camera. It combines techniques from both direct and indirect methods, leveraging the advantages that both approaches provide. The proposed method applies a selection process to extract plane features from one of the input depth images, akin to indirect approaches. Instead of solving correspondences, however, each planar patch is matched with a point from the other image based on the available motion estimation. The relative pose of the camera is then calculated as the rigid transformation that minimizes the point-to-plane distance between feature pairs. A refinement procedure follows, where the matches are iteratively updated based on the latest motion information, which results in a new and more accurate relative pose estimation. Note that the use of plane features exploits the information found in the flat surfaces commonly found in manmade environments. Furthermore,

it allows for a more lenient feature matching, eliminating the need for a computationally expensive correspondence solver, enabling high-frequency operation.

To test the viability of the proposed method, it has been compared on the ICL-NUIM RGB-D dataset [35] with DIFODO [42], a method known for its precision and efficiency. Results show how our proposal can reduce the median of the error by 25% in both translation and rotation.

4

3D lidar odometry

This chapter describes the two odometry methods developed for 3D lidar. They exploit the fact that modern sensors can output ordered point clouds that can be represented as images, thanks to the technological progress seen in 3D lidars because of their relevance in autonomous vehicles. To recover the ego-motion, planar features are efficiently extracted from the range images, to match them with points of its counterpart image. The rigid transformation that minimizes the point-to-plane distance between every feature pair is thus the estimation of the relative sensor pose. By using the information found in the flat surfaces of the scene this way, feature pairs just need to fulfill being coplanar, and thus there is no need to go through the time-consuming process of solving correspondences. Instead, a reprojection approach is employed. When the planar features are big enough compared with the sensor movement, there is a high probability that two points from different images belong to the same flat surface of the scene, satisfying the coplanarity constraint. The second method presented here goes one step further and uses the planar surfaces to decouple the motion estimation, improving both efficiency and accuracy. Both approaches have been evaluated in popular 3D lidar datasets, outperforming state-of-the-art competitors in both efficiency and accuracy.

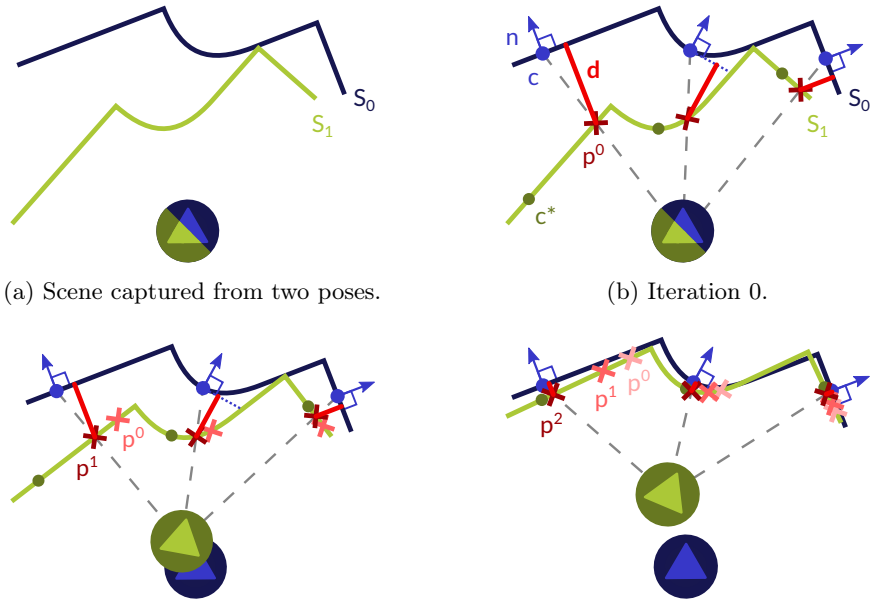
4.1 Introduction

3D Lidar Odometry consists of estimating the relative pose of a moving vehicle by registering consecutive *scans* provided by a 3D lidar sensor. These sensors stand out for yielding accurate geometric information from the scene while performing reliably under changes in lighting conditions, which frequently occur in real-world scenarios [1]. Given the advantages 3D lidars offer, further amplified by the continued development of more advanced sensors, their popularity among autonomous vehicles has been on the rise in recent years. Examples of this are the autonomous driving platforms found in state-of-the-art projects such as Waymo [23, 93], and Argoverse [12, 100], as well as in datasets such as KITTI [30] and KAIST Urban [45].

Among the diversity of environments in which autonomous vehicles are designed to navigate, it is worth emphasizing the relevance of urban areas, which are characterized by having an abundance of planar surfaces, mainly from the ground and the walls of surrounding buildings or other structures. This raises a distinctive layout of orthogonal planes that can be exploited for motion estimation.

In this chapter, we introduce the two developed 3D lidar odometry methods that exploit the information found in flat surfaces of the scene, as well as the image representation of ordered point clouds provided by modern sensors. The first method extracts planar patches from the input range images, but avoids solving correspondences, unlike feature-based approaches. Instead, it iteratively matches each plane with a point of the opposite scan based on reprojection using the latest motion estimation. The relative pose of the sensor can then be calculated by minimizing the point-to-plane distance between pairs. For this estimation to be valid, feature pairs must belong to the same flat surface of the scene, known as the coplanarity constraint. See Figure 4.1 for a visual summary of the method, and Section 4.3 for a more detailed description.

The second method, named GND-LO, improves the previous one by decoupling the motion estimation in two steps, each using patches from the ground and walls respectively. Additionally, the planar features are extracted using quadtree segmentation. These modifications result in a method that provides a higher accuracy with an improved efficiency. See Section 4.4 for an in-depth explanation of this method.



(a) Scene captured from two poses.

(b) Iteration 0.

(c) Iteration 1.

(d) Iteration 2.

Figure 4.1: Various iterations showing the iterative matching employed in both proposed methods (b)-(d), with two point clouds of the same environment as the input (a). Feature pairs are assumed to fulfill the coplanarity constraint, meaning they belong to the same flat surface of the scene. The motion is estimated by minimizing the point-to-plane distances between pairs, shown as red lines. In each iteration the corresponding point (red) of each plane (blue) is updated based on the estimated motion, refining the matching and thus increasing the accuracy of the solution.

4.2 Related work

As established in Chapter 2, 3D lidar odometry has a close relation with point cloud registration, dominated by the Iterative Closest Point (ICP) [7] algorithm. It is only normal that numerous state-of-the-art 3D lidar odometry methods are built upon ICP and its variants, each improving on the original with different contributions, like ELO [111], LOCUS 2.0 [76], SGLO [55], and Reinke et. al [75], for instance. Despite its clear popularity, ICP exhibits two main limitations, namely its computational cost and its tendency to fall into local minima if the problem is not correctly initialized.

Another main category of odometry methods in the literature corresponds to feature-based approaches. They consist of reducing the input scans to a set of features (like points, lines, or planes) and, after correspondences between

the features in both scans have been computed, the motion is recovered by minimizing the distance of each pair. Notable works in this category include Velas et al. [96] for choosing line features, and DiLO [34], which performs bundle adjustment with keypoints located on edges and planes along multiple consecutive scans. The right operation of these methods depends on a correct selection and matching of features, and thus solving correspondences becomes essential. Instead, in our proposal the planar patches are iteratively paired with points according to the current motion estimation, reducing the computation time and allowing room for error in the matching.

It is worth mentioning the relevance of 3D lidar odometry methods as the front end of LOAM and SLAM, which create a map of the surroundings while providing the ego-location. By employing said map, for example, performing scan-to-map registration, the drift over time is greatly reduced. Since these algorithms usually depend on an odometry method to provide the scan-to-scan motion, they can be categorized similarly. Two main groups stand out in the literature, those that leverage feature-based algorithms (e.g. see [18, 32, 67, 85, 87, 108, 109]), and those whose localization algorithm is a variation of ICP (e.g. see [5, 65, 106]). Given the number of parallel processes running in a LOAM or SLAM system, it becomes highly important that the front-end odometry is computationally inexpensive. The methods described in this chapter recover the movement between consecutive scans employing minimal resources, thus becoming suitable candidates as the front-end in this kind of algorithm.

Various methods in the literature achieve better accuracy and efficiency by performing ground segmentation on the input scans. For example, in [85, 87] ground features are separated from the rest to decouple the motion estimation in two steps, each providing half of the DOFs. Similarly, the previously mentioned ELO [111] performs ICP to ground and non-ground features separately. In [14], the ground segmentation is used to reject non-planar features on the ground. In our proposed GND-LO, not only the motion estimation is decoupled into two steps, but the ground registration is performed faster and more efficiently by simply clustering the ground planar patches into a single plane, removing the need of employing an optimization approach.

4.3 Odometry based in planar patches

This section describes how this first proposed method estimates the motion of a 3D lidar over a sequence of range images, including the experiments carried out to evaluate its performance. Thus, the objective is to find the relative pose of the sensor between two consecutive frames, defined by the rotation matrix and translation vector $(R, \vec{t})$, with $R \in \mathrm{SO}(3)$ and $\vec{t} \in \mathbb{R}^3$. The input is a

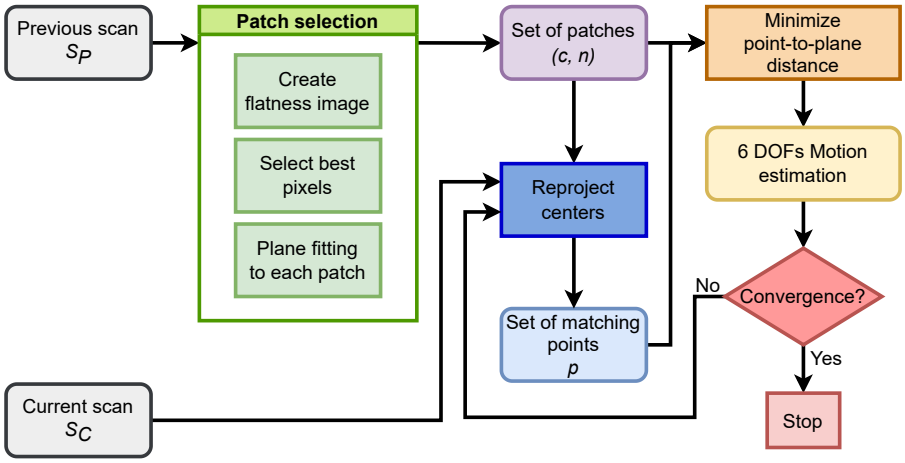


Figure 4.2: Overview graph showing the workflow of the first proposed method. As a preprocessing step, a flatness image is computed from the previous ordered scan S_P to find the pixels most likely located in flat surfaces of the scene. Once planes have been fit to the selected pixels, the iterative motion estimation process can begin. It consists of reprojecting the selected planes onto S_C based on the current motion estimation, which yields their corresponding pixel coordinates. From these image coordinates, the point related to the selected plane can be extracted, completing the feature pair. The relative pose estimation is then obtained by minimizing the point-to-plane distance of feature pairs. This is repeated until convergence.

pair of ordered scans, represented as range images, S_P and S_C (*previous* and *current* scans respectively), where $S_i : (\Omega \in \mathbb{N}^2) \rightarrow \mathbb{R}$ is defined on the image domain Ω. Figure 4.2 shows the general workflow of our method.

The main idea of this method is to create pairs of planar and point features from S_P and S_C respectively, to then impose the coplanarity constraint to each pair. When this hypothesis holds true, the rigid transformation that minimizes the point-to-plane distance between them corresponds to the sensor motion [72]. Note that the supporting flat surface from which the planar patch is extracted does not need to be completely flat for the coplanarity constraint to apply. The bigger the supporting flat surface of the planar features, the higher the probability of fulfilling the hypothesis. Thus, a selection algorithm is employed to choose the pixels more probable of belonging to large flat surfaces.

The approach followed in the proposed algorithm consists of creating a flatness image (as seen in Figure 4.3) that holds the information of how flat the neighborhood around each point is, so pixels can be selected based on this score. Once the pixels from S_P located in flat surfaces have been selected, a

planar patch is fitted to the neighborhood of each one. The details of this selection process will be further explained in Section 4.3.1.

The next step consists of finding the 3D points from S_C related to each selected plane. This is an iterative process, since the points are extracted from the pixel coordinates obtained by reprojecting the planes onto S_C based on the current motion estimation. For the first estimation, the movement is assumed to be zero, resulting in matching each planar feature with the 3D point in S_C located in the same pixel coordinates. This will be expanded on Section 4.3.2.

This reprojection-based matching results in a set of planes from S_P defined by their center and normal vector $(\vec{c}_i, \vec{n}_i)$, and their corresponding points $\vec{p}_i$ from S_C, with $\vec{c}_i \in \mathbf{C}$, $\vec{n}_i \in \mathbf{N}$ and $\vec{p}_i \in \mathbf{P}$. The relative pose $(R, \vec{t})$ that minimizes the distance between the pairs of features also represents the relative pose between the input scans. It is possible, as long as the coplanarity constraint is fulfilled and each point belongs to the corresponding plane, to register both scans this way. Section 4.3.3 will delve into this estimation process.

As explained before, this is an iterative process, so the estimated motion is fed back to recalculate the points associated to each plane. This increases the probability of the paired features fulfilling the coplanarity constraint, which in turn improves the estimation of the relative pose. This can be repeated until convergence.

4.3.1 Flatness-Based Selection

This section describes the preprocessing step which reduces the input frame S_P to a set of selected planar patches defined by their centers $\mathbf{C}$ and normal vectors $\mathbf{N}$. These planar patches will be, at a later stage, matched with points $\mathbf{P}$ from S_C. This pairing will be considered valid if both features belong to the same surface of the scene, satisfying the coplanarity constraint. The fulfillment of this hypothesis depends on two main factors: the size of the selected planar patches and the movement to recover. Note that the latter is given by the speed of the sensor and cannot be controlled, unlike the former. To increase the probability of planar features being correctly matched, they are selected based on how flat their neighborhoods are. The bigger the supporting region of the selected plane ($\vec{c}_i$,$\vec{n}_i$), the higher the probability for the point $\vec{p}_i$ to lie within it in the next frame, even after big movements. Some form of plane segmentation could be applied here to improve the selection, but these algorithms tend to be computationally expensive, preventing the method from running at real time. However, as previously discussed, opting for a faster alternative has the advantages of working with smaller movements, and having the solution (pose estimation) available more often.

Considering this, the proposed selection approach is based on obtaining a flatness image, which holds information about how planar the area around each pixel of S_P is. An example of a flatness image is shown in Figure 4.3. In

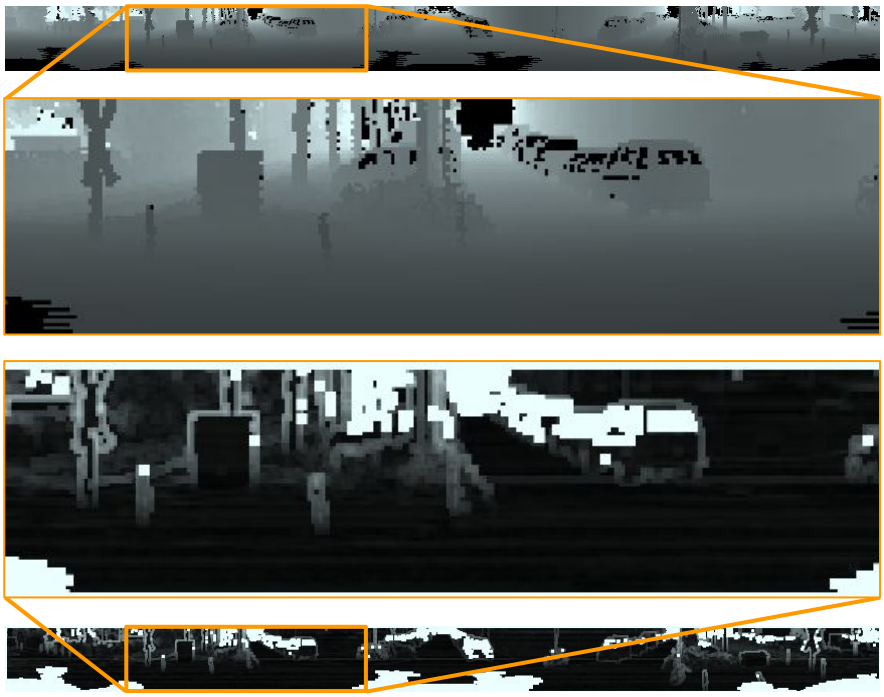


Figure 4.3: Flatness image (below) generated from S_P (above) applying the LSF method. Flat surfaces like the ground and walls appear darker, while non-planar pixels are displayed brighter. Images in the middle are zoomed in areas of the panoramic images above and below.

this article, two ways of obtaining the flatness images are presented, each one prioritizing accuracy or efficiency respectively:

- **Least Squares Fitting (LSF)**: for each pixel, a plane is fitted to its neighborhood using LSF. Then, the error between the fit plane and the surrounding points is used as the value in the flatness image. This technique is very accurate at the cost of being computationally more expensive.

- **Curvature-based (Fast method)**: this approach exploits the function obtained by representing a 3D plane in the spherical coordinates of the range image. These images describe each point in spherical coordinates, where the value of each pixel is the distance (r) from the point to the sensor, and the image row and column relate to the azimuthal (θ) and polar (ϕ) angles respectively. When a plane $ax+by+cz = 1$ is represented

in said spherical coordinates system, the distance r of a point in the plane can be described as a function of ϕ and θ as follows:

$$r(\phi, \theta) = \frac{1}{a\cos\phi\cos\theta + b\sin\phi\cos\theta + c\sin\theta}. \tag{4.1}$$

Thus, the inverse of the distance function is a sine wave:

$$d(\phi, \theta) = \frac{1}{r} = a\cos\phi\cos\theta + b\sin\phi\cos\theta + c\sin\theta. \tag{4.2}$$

Instead of trying to fit a sine wave to check for flat areas, the curvature of the neighborhood can be tested. The curvature of a function can be approximated by its second derivative [26], which means that the second derivative of the inverse of the distance d along both ϕ and θ should be low for pixels located in flat surfaces. However, non-planar areas, like the intersection of two planes, will result in an abrupt change in curvature. The derivatives along both image axes can be efficiently calculated with the use of 2D convolutions, applying the Sobel operator. The value stored in the flatness image is the curvature k of each pixel, calculated as the sum of the absolute value of the second derivative of d along each axis:

$$k = \left|\frac{\partial^2 d}{\partial \phi^2}\right| + \left|\frac{\partial^2 d}{\partial \theta^2}\right|. \tag{4.3}$$

Regardless of the technique used to obtain the flatness image, a Gaussian filter is applied to blur it afterwards, which helps to deal with pixels located near non-planar areas by leveraging the score of their neighborhoods.

Once the flatness image has been created, it is then used to select pixels from S_P based on their score. To ensure a good distribution of pixels around the scene, the image is divided into blocks, and the best pixels from each block are selected. This also serves as a non-maximum suppression step.

A planar patch $(\vec{c}_i, \vec{n}_i)$ is fitted to each neighborhood around the selected pixels in the image S_P, with the centroid of the group being the center of the plane. For the normal vector, the singular value decomposition is applied to the matrix created by stacking the difference of each point of the neighborhood and the centroid. The normal vector is then the left singular vector corresponding to the least singular value. A measurement of the fitness f_i, how planar the group of points is, is also estimated as the least singular value. Low values of this variable mean the group of points actually form a planar surface.

Take into account that it is no guarantee that the selected pixels are located in a planar surface just from the flatness image, but once the planes are fit into their neighborhood, the fitness f_i can be used to downweight selected pixels found near occlusions or in curved surfaces.

4.3.2 Iterative Projection-Based Matching

After the planar patches have been obtained from S_P, their corresponding points $\vec{p}_i$ have to be extracted from S_C. In traditional indirect methods the features from both frames are matched based on some descriptor. Instead, in the proposed method an iterative approach is implemented, inspired by direct methods. It consists of reprojecting the centers of the selected plane onto S_C based on the current motion estimation, to find the pixels in S_C that are more probable of representing the same scene surface as their corresponding planar region. This process works as follows:

1. The center of each plane $\vec{c}_i$ is transformed using the motion estimation $(R, \vec{t})$ to obtain $\vec{c}_i^*$:

$$\vec{c}_i^* = [x_c, y_c, z_c]^\top = R^{-1}(\vec{c}_i - \vec{t}). \tag{4.4}$$

2. The spherical coordinates of the transformed center are then calculated:

$$\begin{cases} r_i &= \sqrt{x_c^2 + y_c^2 + z_c^2} \\ \phi_i &= \arctan2(y_c, x_c) \\ \theta_i &= \arcsin(z_c / r_i) \end{cases} \tag{4.5}$$

3. The row (v) and column (u) on the image are obtained based on the spherical coordinates:

$$\begin{cases} v_i = \arg\min_j |\theta_{vec}(j) - \theta_i| \\ u_i = [(\text{columns} - 1)(\pi - \phi_i)/(2\pi)] \end{cases} \tag{4.6}$$

   Where θ_{vec} represents the vector of the azimuthal angles related to the row numbers, and $[\,]$ is the rounding operator, which returns the integer number closest to the input value.

4. These pixel coordinates can then be used to obtain the points $\vec{p}_i$ from S_C. A Gaussian filter is applied over the neighboring points to reduce negative effect of noise.

During the first iteration, the relative pose between both range images is considered to be composed by the identity rotation $R = I$ and no translation $\vec{t} = 0$, which means that the points $\vec{p}_i$ are located in the same image coordinates as the centers $\vec{c}_i$ of the selected planes, but in the current image. Note that by imposing the coplanarity constraint the only requisite is that the points and planes selected belong to the same surface of the 3D scene, so there is no need for a perfect point-to-point matching. By selecting a pixel in S_P located in a

planar surface as explained in the previous section, there is a high probability that the point located at the same image coordinates in both frames lies on the same surface.

4.3.3 Relative Pose Estimation

This section describes how the relative pose between the two range images is calculated given the set of matching feature pairs, assuming they satisfy the coplanarity constraint. The objective is to find the rigid transformation $(R, \vec{t})$ that minimizes the point-to-plane distance of each pair of features, as shown in Equation 4.7. This is optimized by the Levenberg-Marquardt implementation from the well-known Ceres Solver [3]:

$$(R, \vec{t}) = \arg\min_{R,\vec{t}} \sum_i \rho\left(\left\|\alpha_i \vec{n}_i \cdot (R\vec{p}_i + \vec{t} - \vec{c}_i)\right\|^2\right), \tag{4.7}$$

where $\alpha_i = (1 - f_i)$ serves as a weight of the residual based on the fitness of the selected plane. This way, selected pixels located in surfaces with lower flatness have less impact on the final result. The Huber loss function ρ is applied to the residual to deal with outliers, which are pixels that do not fulfill the hypotheses, whether because they move with respect to the static scene or because they do not lie in the same surface in both frames. Along with the relative pose, the covariance of the solution is calculated, which holds the information of the uncertainty of each parameter of the rigid transformation.

The motion estimation is part of the iterative process previously mentioned (recall Figure 4.2), so the resulting transformation is fed back to update the points $\mathbf{P}$ in S_C corresponding to each planar region. This improves the matching of the features by making it more probable that both of them belong to the same surface of the scene, improving in turn the motion estimation. This procedure is repeated until convergence of the solution.

As a final process, a motion filter is applied to the resulting relative pose, similar to the one found in [42], in which the motion estimation from the previous pair of frames is leveraged if the covariance of the current solution is high. This has proven to be helpful in scenes with low planar information, improving the accuracy of the localization as will be proven in Section 4.3.4.

4.3.4 Experiments and Results

In this section, we evaluate the performance of our method resorting to sequences of 3D lidar scans from the widely-used Velodyne SLAM dataset [65]. The obtained results are compared with different implementations of the state-of-the-art GICP algorithm [84], including single-thread and multi-thread alternatives. For all the experiments, the test platform used is a standard PC with 8GB of RAM, running Ubuntu 20.04 with an Intel Core i7-7700HQ CPU

Table 4.1: Computation time breakdown of the proposed method.

Process	Fast (ST)	LSF (ST)	LSF (MT)
Flatness image	2.5 ms	63.5 ms	26 ms
Plane select. and fitting	3 ms	3 ms	3 ms
Matching points extraction	2.5 ms	2.5 ms	2.5 ms
Motion estimation	25 ms	25 ms	25 ms
Total	33 ms	94 ms	56.5 ms

with 4 cores and 8 execution threads at 2.8GHz. We also present an ablation study exploring different configurations of the proposed method and discuss the limitations of the proposal.

Testbed

The proposed method was evaluated on the two outdoor sequences provided by the Velodyne SLAM dataset [65], captured by a vehicle driving around town in a closed loop over a kilometer in length. The Velodyne HDL-64E sensor, running at 10Hz, provides the geometric information in two formats: point clouds and 870x64 range images.

The error metric employed to compare the different methods is the one proposed by Sturm et al. [92], known as relative pose error (RPE). Although the RMSE of the error will be shown because of its popularity in odometry literature, the mean RPE will be used to compare the methods based on the arguments presented by Willmott and Matsuura [99].

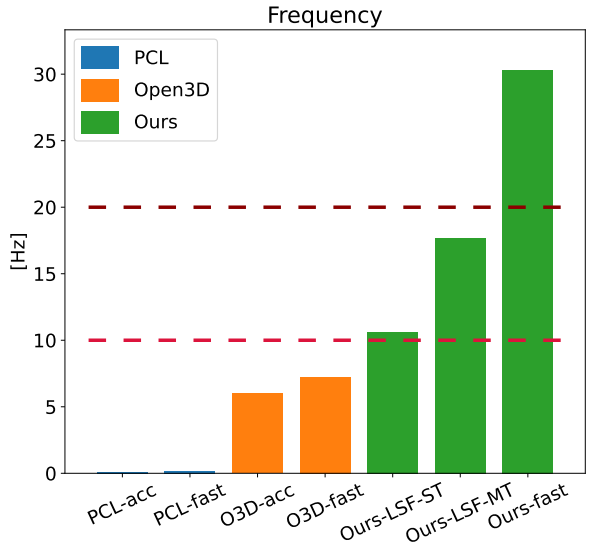
The proposed method is configured with the following parameters:

- Neighborhoods of each pixel are 3x3 blocks.
- The flatness image is divided into 16x16 blocks, and the 4 best pixels are selected from each block.

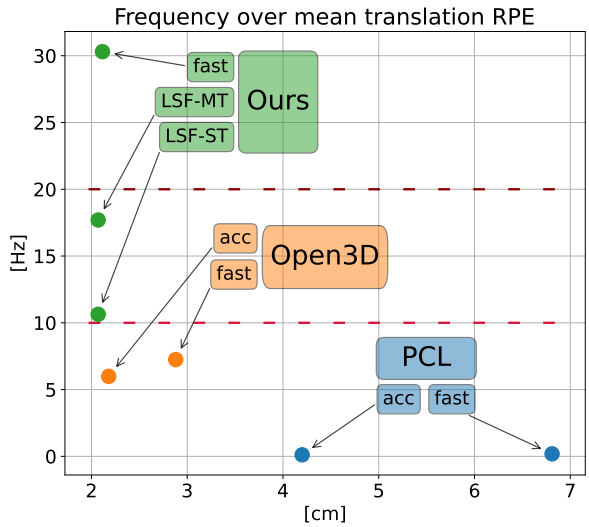
Time Performance

The performance of our method is compared to the widely popular GICP algorithm [84] as implemented in two different C++ libraries: PCL [81] and Open3D (O3D) [112]. The former is a basic single thread (ST) implementation, while the latter applies a loss function to reduce the effect of outliers, and performs at a higher rate because of multi-threading.

The proposed method is tested with both versions of the flatness image introduced in Section 4.3.1, *fast* and *LSF*, with the second method being more accurate at the cost of computation time. For a fair comparison with the GICP implementation of Open3D, a multi-threaded (MT) implementation of



(a) Frame rate of the different methods. The usual working frequencies of 3D lidar, 10 and 20 Hz, are displayed along the data as dashed lines.



(b) Frequency of the different methods over the translational RPE. The usual working frequencies of 3D lidar, 10 and 20 Hz, are displayed along the data.

Figure 4.4: Frequency comparison of the different methods.

Table 4.2: Computation time comparison between the different methods.

Threaded	Method	Time per frame	Frame rate
No	PCL (fast)	5300 ms	0.188 Hz
	PCL (acc.)	8800 ms	0.114 Hz
	Ours (fast)	33 ms	30.30 Hz
	Ours (LSF, ST)	94 ms	10.64 Hz
Yes	Open3D (fast)	138 ms	7.24 Hz
	Open3D (acc.)	167 ms	5.98 Hz
	Ours (LSF, MT)	56.5 ms	17.69 Hz

the LSF variant is included. A breakdown of the elapsed times for every part of the proposed algorithm is shown in Table 4.1 for both versions. It is clear how the use of 2D convolutions in the *fast* version of the method greatly reduces the computational cost. Regarding the threaded implementation of our method, the only process worth parallelizing is the creation of the flatness image, resulting in a 59% reduction of computation time.

With the goal of performing a balanced comparison, a pair of configurations were employed in the GICP calculations, each one prioritizing accuracy or speed over the other one. In summary, seven odometry methods are compared in this section, four of them being a variation of GICP (PCL-fast, PCL-acc., Open3D-fast, Open3D-acc.), and three of them being the different versions of the proposed method (Ours-fast, Ours-LSF-ST, Ours-LSF-MT). The Open3D implementation of GICP and Ours-LSF-MT are multi-threaded, while the rest run on a single thread. Table 4.2 shows the computation time and frequency of each method, where we can see how the slower implementation of our method, using the LSF, outmatches the frame rate of the faster GICP (Open3D-fast), while the fast version achieves 4x its speed. Note that 3D lidar sensors usually run at 10-20Hz, so the presented configurations of the proposed method work in real time. Figure 4.4a shows this information in a more visual way.

Accuracy Results

Regarding the accuracy, Table 4.3 displays the translational and rotational error of the six methods being compared along the two sequences of the dataset. Note that the results obtained from Ours-LSF are the same whether or not it is multi-threaded. In said table we can see how, when comparing the LSF version of our method with the competitor showing the lowest error (Open3D-acc), our proposal is able to achieve a reduction in the translational error of 5% and 19% of the mean and median value of the RPE respectively, while speeding up its frame rate by a factor of 2.9x. Even the proposed Ours-fast version, which works five times faster than Open3D-acc., reaches a similar

Table 4.3: RPE comparison between the different methods.

(a) Translational RPE.

[cm/frame]		PCL fast	PCL acc.	O3D fast	O3D acc.	Ours fast	Ours LSF
RMSE	Scen. 1	11.55	8.54	5.68	**5.14**	6.95	7.04
	Scen. 2	14.68	9.22	**7.47**	7.54	7.98	7.83
	Avg	13.12	8.88	6.58	**6.34**	7.47	7.44
Mean	Scen. 1	6.55	4.31	2.97	2.30	2.22	**2.18**
	Scen. 2	7.07	4.09	2.79	2.06	2.01	**1.96**
	Avg	6.81	4.20	2.88	2.18	2.12	**2.07**
Median	Scen. 1	3.79	2.58	2.31	1.77	1.47	**1.44**
	Scen. 2	3.61	2.38	2.79	1.55	**1.23**	**1.23**
	Avg	3.70	2.48	2.55	1.66	1.35	**1.34**

(b) Rotational RPE.

[°/frame]		PCL fast	PCL acc.	O3D fast	O3D acc.	Ours fast	Ours LSF
RMSE	Scen. 1	0.230	0.198	0.189	**0.184**	0.212	0.203
	Scen. 2	0.228	0.177	0.168	**0.163**	0.257	0.324
	Avg	0.229	0.188	0.179	**0.174**	0.235	0.263
Mean	Scen. 1	0.177	0.154	0.147	**0.142**	0.145	0.144
	Scen. 2	0.171	0.140	0.131	**0.126**	0.128	0.129
	Avg	0.174	0.147	0.139	**0.134**	0.137	0.136
Median	Scen. 1	0.134	0.119	0.112	0.109	0.108	**0.107**
	Scen. 2	0.133	0.111	0.102	0.097	0.093	**0.091**
	Avg	0.134	0.115	0.107	0.103	0.100	**0.099**

accuracy improvement. Regarding the rotation error, a similar performance is achieved, with the difference between GICP and our method being below 3%. Note that there is a notable difference between comparing the RMSE, mean error, and median error of the different methods. We agree with Willmott and Matsuura [99] that the RMSE is not able to correctly capture the general accuracy of the data since it severely penalizes instances of big errors, however scarce they are. The mean absolute error on the other hand allows for a fairer comparison of the accuracy of the presented methods. In addition, the median absolute error conveys the accuracy obtained in most of the frames along each sequence. Figure 4.5 shows a visual representation of this evaluation, where we can check the superior performance of our proposal regarding accuracy.

Version	Translation: mean RPE [cm/frame]	Improvement over previous version
Base (1 iteration)	11.27	-
Base (2 iterations)	3.24	71.25%
Iterations until convergence	2.31	28.70%
Prev. motion filter	2.24	3.03%
Prev. flatness image blurring	2.18	2.68%

Table 4.4: Improvements in accuracy of the different versions of the proposed method.

Moreover, for a qualitative comparison, the trajectories computed by each method are displayed in Figure 4.6.

Figure 4.4b shows the running frequency of each method over their mean translation RPE, allowing a comparison of accuracy and speed combined. For example, it can be seen how Ours-fast only performs 3% better than O3D-acc in translation RPE, but runs 5x faster. Compared to the faster version of GICP, Ours-fast works at 4x the speed, and the accuracy improvement goes up to 26%. Regarding the *LSF* version of the proposed method, it yields a better accuracy, with a 5% and 28% improvement over the accurate and fast versions of Open3D respectively, while running 2.9x and 2.4x faster than them.

Ablation Study

The purpose of this section is to validate the contribution of the different processes employed in our proposal to the resulting performance. Particularly, three procedures are analyzed: the iterative update of the correspondences, the motion filter, and the blurring of the flatness image. Thus, the baseline for this test is the base algorithm running a single iteration.

Based on Table 4.4, the effect of a second iteration proves to be worth it, reducing the translation error over 70%. Allowing the system to iterate until convergence improves the accuracy even further. Regarding the motion filter, it helps to mitigate the effect of scenes where not enough information is found, which happens sparsely throughout the sequences, resulting in an improvement of the accuracy of 3%. Similarly, applying a Gaussian filter to the flatness image helps with selecting pixels located far from non-planar areas, reducing the error 2.6% further.

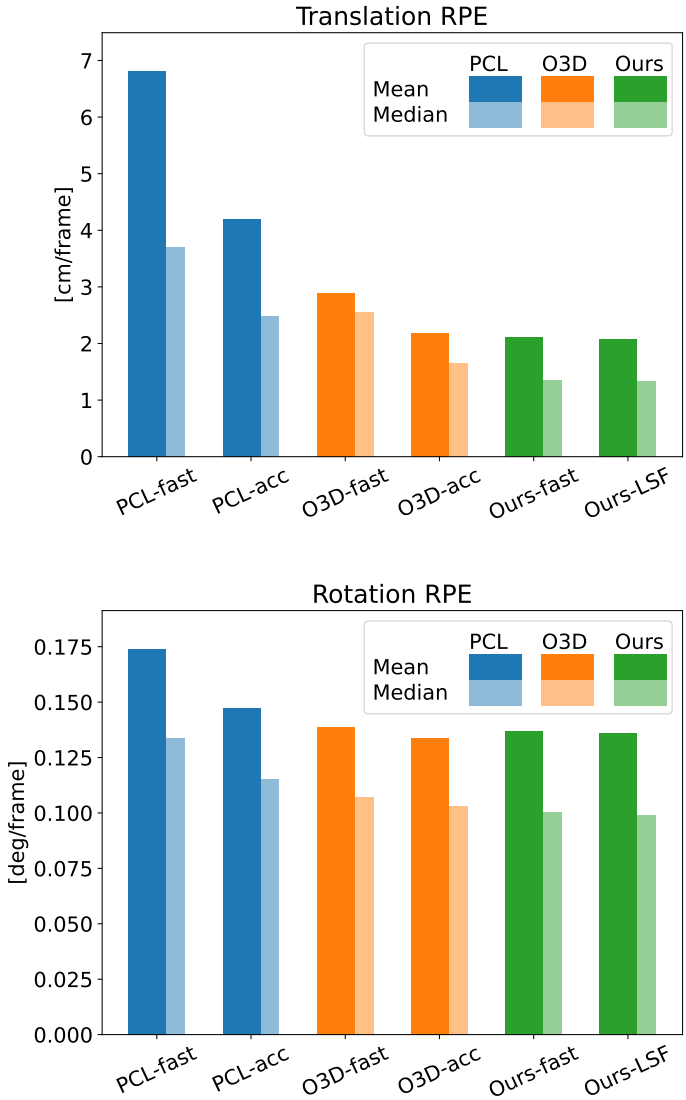


Figure 4.5: Comparison of the mean and median RPE of the different methods. Translational and rotational error are displayed above and below respectively.

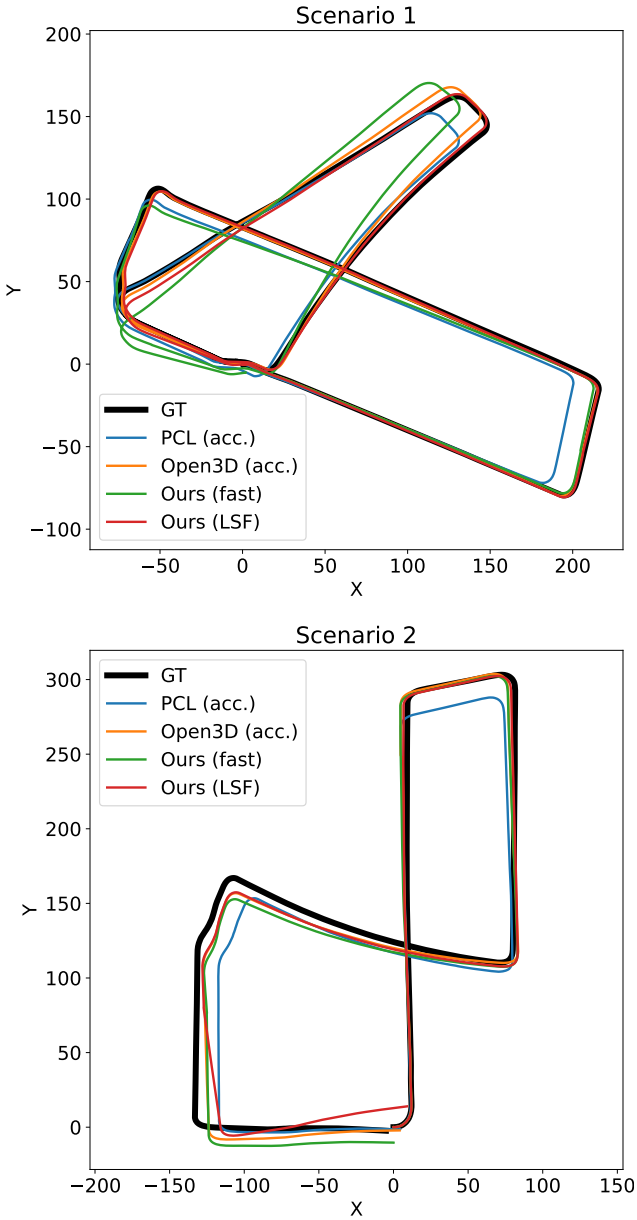


Figure 4.6: Top view of the resulting trajectories of the tested methods over each sequence of the dataset, along with the ground truth.

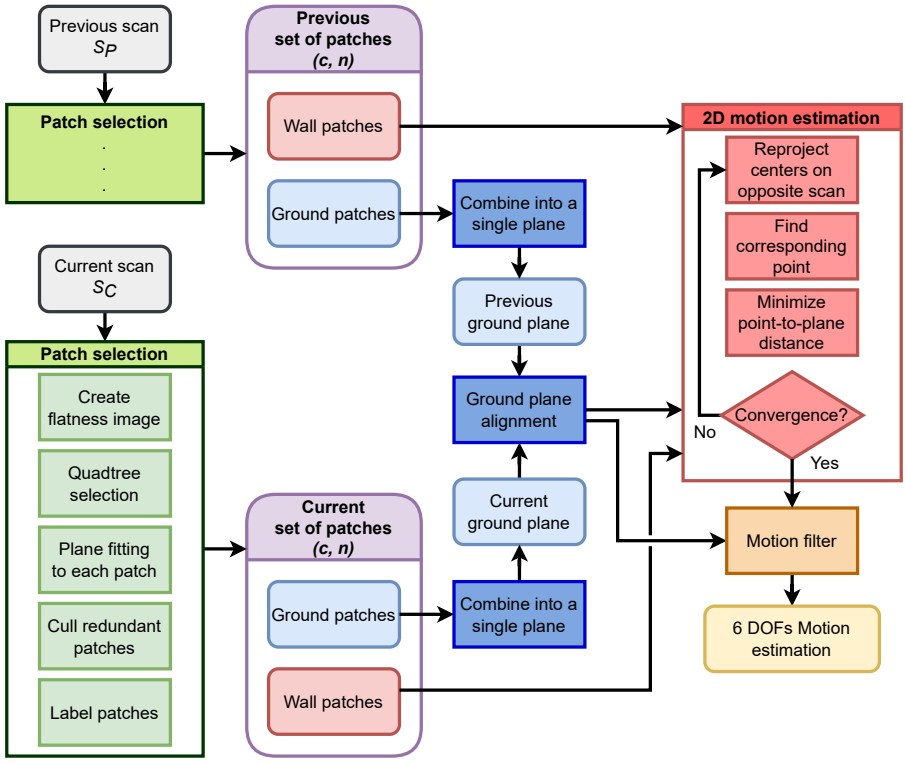


Figure 4.7: Workflow of GND-LO. Data blocks have round corners to distinguish them from process blocks.

4.4 Ground-decoupled odometry

In this section, we will describe our second proposal GND-LO, whose pipeline is shown in Figure 4.7, as well as the experimental evaluation carried out. Basically, it consists of leveraging selected patches from the scene to estimate the relative motion between two consecutive input scans. It becomes clear that the quality of the selected sets of patches will have a great impact on the movement estimation procedure. Therefore, special consideration is taken in the selection process in order to reach high accuracy.

The input to the algorithm is a pair of consecutive scans, referred to as *previous* (S_P) and *current* (S_C) from now on. These scans consist of an ordered point cloud, and so they can also be represented as a range image $S_P, S_C : (\Omega \subset \mathbb{R}^2) \rightarrow \mathbb{R}$. Taking advantage of this fact, the first step is to create a flatness (low-curvature) image from each scan to help discern which points from the scene belong to planar surfaces. A quadtree segmentation is applied

to said images to extract the regions located on flat surfaces of the scene. The group of points included in each selected region of the scans is fit into a planar patch defined by its center $\vec{c}$ and normal vector $\vec{n}$ (see Section 4.4.1). Once the initial set of patches has been extracted from each scan, denoted by $\mathbf{P}_P$ and $\mathbf{P}_C$ respectively, their directional-independence is analyzed. For the motion to be fully observable, each set needs to include patches oriented in three independent directions. The distribution of normal vectors of the patches can be checked via PCA, and then patches that provide redundant information can be culled to reduce the computational cost, while keeping patches in less populated directions (Section 4.4.2). As the final step in the patches' selection, they are labeled as either belonging to the ground, walls, or outliers. Figure 4.8 shows the different stages of the selection process, from the original range image to the final selection of patches, labeled as ground or wall. Of course, once the first two scans in a sequence have been processed, this selection process is only done for each new input scan.

Once each input scan has been reduced to a set of patches with enough information in three independent directions, the motion between the two scans can be estimated. Since the patches are labeled, the motion can be decoupled into two steps, each using the ground and wall patches respectively. In the first stage, the ground patches from each set are combined into a single ground plane. Three of the total six DOFs of the 3D motion can be recovered by registering the obtained ground planes (see Section 4.4.4). The rest of the motion corresponds to the 2D motion that "slides" over the ground plane. To retrieve it, each wall patch is reprojected onto the opposite image to find its matching point, and the transformation is estimated by minimizing point-to-plane distances between pairs. The correspondence of each patch is calculated as an iterative procedure that makes use of the latest estimation of the motion (Section 4.4.5). Finally, a motion filter is applied to the resulting estimation, leveraging previous instances of movement to reduce the uncertainty of the solution. For that, a similar approach as in [42] and [27] has been implemented.

4.4.1 Patches selection via quadtrees

This section details the process to obtain the initial set of patches ($\mathbf{P}_P$ and $\mathbf{P}_C$) from each input scan (S_P and S_C), which is portrayed in Figure 4.8. The objective is for each set of patches to contain the most informative flat areas of the scene, and to this end, the first step is estimating how flat the area around each point of the scan is. This is accomplished by analyzing the curvature around each pixel in the range image representation of the scan. Since the curvature of a function can be approximated by its second derivative [26], the pixel-wise flatness $\mathcal{F}(v, u)$ can be calculated as the sum of the squared second-order derivatives in each direction, which can be efficiently obtained by applying the Sobel operator:

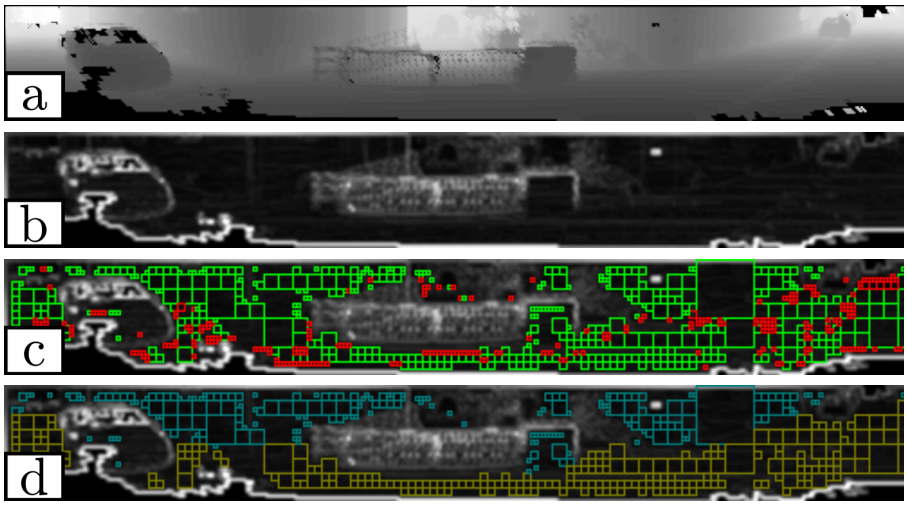


Figure 4.8: Different stages of the extraction and classification of planar patches from a lidar scan. From the original range image (a), the flatness image is created (b). Quadtree segmentation is applied to it and an initial set of patches is selected (c). Based on the directional independence of the set, the redundant patches are culled (red patches). Finally, patches are labeled as either ground or wall (d), shown as yellow and blue respectively. Note that only half of the scan is shown for clarity. The scene captured in these images is a road (visible in both extremes of the image) going over a bridge (identified by its fence), with a couple of building corners on each side of it and a parked car located on the left side.

$$\mathcal{F}(v,u) = \left(\frac{\partial^2 S}{\partial v^2}(v,u)\right)^2 + \left(\frac{\partial^2 S}{\partial u^2}(v,u)\right)^2. \tag{4.8}$$

We call this the flatness image, as pixels with a low value in this image represent points whose surroundings exhibit a low curvature and therefore are more likely located on a flat surface of the scene. See Figure 4.8b as an example.

The resulting flatness image is then leveraged to select groups of points from the scan located on flat surfaces, from which the patches will be created. There are two main advantages associated with selecting patches with a big supporting group of points: reducing the impact of noise, and improving the matching in the future motion estimation process. The former is rather self-explanatory, as a bigger support region means the error of individual points (due to noise, occlusions, or similar) has less impact on the planar patch estimation. The latter is linked to the iterative projection-based matching employed in the 2D motion estimation, where patches with a bigger area are more likely paired

Algorithm 4.1: Proposed quadtree selection algorithm

Input: Flatness image $\mathcal{F}$
Output: Set of initial patches **P**
$max\_std \leftarrow$ maximum variation before a block is divided into four;
$max\_avg \leftarrow$ maximum average flatness to consider a block flat;
$min\_size \leftarrow$ minimum block size to consider;
$sz \leftarrow$ initial block size (e.g. 32);
$\mathbf{P} \leftarrow \emptyset$;
$B \leftarrow$ division of $\mathcal{F}$ into square blocks of size sz;
repeat
  foreach *block* $B_i \in B$ *of size* sz **do**
    if $std(B_i) > max\_std$ **then**
      $B \leftarrow$ division of B_i into blocks of size $sz/2$;
    else if $avg(B_i) < max\_avg$ **then**
      $\mathbf{P} \leftarrow$ patch from points in B_i;
    end
  end
  $sz \leftarrow sz/2$;
until $sz < min\_size$;

with a point that belongs to the same planar surface of the scene, even after big movements.

For this reason, a quadtree segmentation is applied to the flatness image to find the most relevant and flat regions distributed along the scan, which are selected to create patches. Quadtree segmentation is a popular technique employed in image processing to efficiently decompose an image into rectangular subsets of similar pixels [82]. The general operation of quadtrees consists of dividing the input image into blocks of a certain size. The variation of the pixel intensities inside each block is then analyzed, and it is divided into four if the difference is over a threshold. This procedure is repeated for the smaller blocks until the image has been divided into rectangular regions containing groups of pixels with similar values. This procedure is seen in Algorithm 4.1, and the results are shown in Figure 4.8c. By applying it to the flatness image, neighboring pixels with a similar curvature get grouped into blocks of varying sizes, each one representing a plane from the scene. The main advantage of quadtree segmentation is grouping large flat areas from the scene into a single block, in contrast to the fixed-size block segmentation in [27], while keeping a low execution time compared to more advanced segmentation algorithms. The 3D points belonging to the blocks that present an average flatness over a threshold are then grouped into a planar patch, with its center $\vec{c}$ being the centroid of the points. The normal vector $\vec{n}$ of the patch is calculated as the

Algorithm 4.2: Proposed patch culling procedure

Input: Set of initial patches $\mathbf{P}$
Output: Set of selected patches $\mathbf{P}_s$
$goal \leftarrow$ minimum number of patches contributing to each principal direction;
$threshold \leftarrow$ minimum contribution of a patch to an underpopulated principal direction to be added to the selection;
$sz \leftarrow$ starting size;
$\mathbf{P}_s \leftarrow \{P_i \in \mathbf{P} | size(P_i) \geq sz\}$;
$M \leftarrow$ Apply Equation 4.9 with $\mathbf{P}_s$;
repeat
  $sz \leftarrow sz/2$;
  foreach $eval \in eigenvalues(M)$ **do**
    if $eval < goal$ **then**
      foreach $P_i \in \mathbf{P} | size(P_i) = sz$ **do**
        $\mathbf{P}_s = \{\mathbf{P}_s \cup P_i\}$ if its contribution to the current principal direction is above $threshold$;
      end
    end
  end
  Update M;
until $sz <= min\_size$;

eigenvector associated with the smallest eigenvalue of the covariance matrix of the group.

4.4.2 Direction-based culling

The initial set of patches after applying the selection algorithm from the previous section is composed of patches of different sizes, flatnesses, and orientations. Previously, the significance of working with big patches has been explained. On the other hand, small patches are noisy and usually belong to surfaces that can only be considered flat in a small neighborhood. In this section, we describe the culling process implemented to discard low-quality patches while maintaining enough information to keep the vehicle movement observable.

For the motion to be completely determined, the selected set must include patches oriented in three independent directions [94, 95]. This can be analyzed through PCA of the matrix M built with the normal vectors:

$$M_{3\times 3} = \sum_{i}^{N} \vec{n}_i \vec{n}_i^{\top} = \overline{n}\,\overline{n}^{\top}, \tag{4.9}$$

where n_i is the normal vector of patch i, N is the total number of patches, and $\overline{n}$ is the $(3 \times N)$ matrix of normal vectors.

The eigenvectors of M represent the principal directions of the patch orientations, and the eigenvalues provide an approximation of how many patches are oriented in (or more accurately, how many "contribute" to) each of these directions. This lets us identify those directions with redundant information and those where every patch provides significant information. Small patches are removed if they provide redundant information in highly populated directions, and kept otherwise. The culling algorithm can be seen in Algorithm 4.2, and an example of the culled patches in Figure 4.8c.

4.4.3 Patch labeling

As a final step before starting the decoupled motion estimation, the patches are labeled as either ground, wall, or outliers. Although there are several techniques in the literature to segment the ground plane from a point cloud, they tend to be computationally expensive. In GND-LO, we propose a fast and simple yet effective way of patch classification. A patch is considered to belong to the ground if the angle difference between the normal vectors of the patch and the ground plane from the previous scan is below a threshold. This simple process works properly as long as the ground normal vector does not change abruptly along the trajectory. To initialize the ground plane, it is first considered to be perfectly parallel to the XY plane.

As for the patches located in walls, a similar strategy is applied. A wall patch must satisfy that its angular difference to the previous ground plane is $90° \pm$ a threshold. The patches that are not labeled as ground or wall are considered outliers and rejected. Refer to Figure 4.8d for a representation of the final set of labeled patches.

4.4.4 Ground clustering and registration

Now we elaborate on the first stage of the decoupled motion estimation, in which the ground planes extracted from each input scan are registered. Although the patches have already been labeled, they still need to be combined into a single ground plane. Different approaches can be applied to that end, but we found that the best strategy is to obtain the average of the centers and normal vectors weighted by the number of points contained within each patch.

Once the ground planes are defined by their center c_G and normal vector $\vec{n}_G$, the transformation that registers them comes from the following two equations. The rotation $\vec{r}_G$, expressed in axis-angle $(\vec{a},\, \alpha)$ notation, is defined as the cross product between their normal vectors:

$$\vec{r}_G = \vec{n}_G^{\,C} \times \vec{n}_G^{\,P}, \qquad (\vec{a} = \vec{r}_G/\alpha, \quad \alpha = ||\vec{r}_G||)\,. \tag{4.10}$$

Note that superscript is used to indicate belonging to previous (^{P}) or current (^{C}) scan. The translation $\vec{t}_G$ can be obtained as the distance between the ground planes after applying the rotation $\vec{r}_G$. The distance d between two parallel planes can be calculated as the difference between their distances to the origin. Since the rotation center is the origin, the distance to the origin of each plane does not change after applying the rotation $\vec{r}_G$. Hence, the translation can be calculated independently of the rotation as the distance d along the axis $\vec{n}_G^P$:

$$\vec{t}_G = d\vec{n}_G^P = (\vec{n}_G^P \cdot \vec{c}_G^P - \vec{n}_G^C \cdot \vec{c}_G^C)\vec{n}_G^P. \tag{4.11}$$

Note that the translation is only determined along the axis of $\vec{n}_G^P$. Similarly, the rotation $\vec{r}_G$ represents the minimal rotation to align the normal vectors of the ground planes, but the rotation around $\vec{n}_G^P$ remains undetermined.

4.4.5 2D motion estimation

In this section, we cover how the wall patches are exploited to recover the remaining three DOFs, which corresponds to a 2D movement on the ground plane. This is achieved by minimizing the point-to-plane distance between each patch $(\vec{c}, \vec{n})$ and its corresponding point $\vec{p}$ from the opposite scan, whose pairing is found by reprojecting the center of the patch onto the opposite range image. Note that reprojection depends on the relative pose between the scans, thus the matches are iteratively updated based on the latest motion estimation. The optimization problem is solved using the Ceres [3] implementation of Levenberg-Marquardt. To further explain the algorithm, the following notation will be used:

- N_P, N_C: total number of patches included in $\mathbf{P}_P$ and $\mathbf{P}_C$ respectively. Note that in this section $\mathbf{P}$ will refer only to patches labeled as walls since the ground patches have already fulfilled their purpose.
- $i \in [1, N_P], j \in [1, N_C]$: iterators used to go through each set of patches.

The step-by-step procedure is now described:

1. Initialize the rigid transformation T as the one obtained by registering the ground planes:
$$T = T_G = T(r_G, t_G). \tag{4.12}$$
2. Reproject the center $\vec{c}$ of each patch, after applying the corresponding transformation, onto the opposite image making use of the projection function $\pi : \mathbb{R}^3 \to \Omega$, to obtain its pixel coordinates. This function depends on the sensor used, but generally consists of converting a 3D point (first argument) to spherical coordinates, which can then be employed

to locate the corresponding pixel location on the range image (second argument):

$$\mathrm{px}_i^C = \pi(T^{-1}\vec{c}_i^P,\ S_C), \tag{4.13}$$
$$\mathrm{px}_j^P = \pi(T\vec{c}_j^C, S_P). \tag{4.14}$$

3. The point $\vec{p}$ associated to each patch is the one located on the pixel coordinates px of the opposite range image. The inverse projection function $\pi^{-1} : \Omega \to \mathbb{R}^3$ is used.

$$\vec{p}_i^C = \pi^{-1}(\mathrm{px}_i^C,\ S_C), \tag{4.15}$$
$$\vec{p}_j^P = \pi^{-1}(\mathrm{px}_j^P,\ S_P). \tag{4.16}$$

4. Once matching is completed, the point-to-plane distance of each pair is minimized to obtain the optimal transformation. In the following equation, ρ is the Huber loss function, which helps with convergence when there are outliers within the data; and T is not a full 6 DOFs 3D transformation, but is instead generated from one rotation (around $\vec{n}_G^P$) and two translations (along two orthogonal axes perpendicular to $\vec{n}_G^P$):

$$T^* = \arg\min_T \rho\left(\sum_i^{N_P}\left[\vec{n}_i^P \cdot (\vec{c}_i^P - T\vec{p}_i^C)\right] + \sum_j^{N_C}\left[\vec{n}_j^C \cdot (\vec{c}_j^C - T^{-1}\vec{p}_j^P)\right]\right). \tag{4.17}$$

5. Update T by composition:

$$T = TT^*. \tag{4.18}$$

6. Check for convergence: if the sum of the differences between the pixel coordinates px after updating the transformation (δ) is below a threshold, the estimation has converged. Otherwise, repeat from step 2. In the following equation, k represents the iteration number:

$$\delta = \sum_i^{N_P}\left|px_{i,k-1}^C - px_{i,k}^C\right| + \sum_j^{N_C}\left|px_{j,k-1}^P - px_{j,k}^P\right|. \tag{4.19}$$

Finally, the relative pose between the two input scans, a full 3D transformation with 6 DOFs, has been estimated and the pose of the vehicle can be updated.

4.4.6 Experiments and results

In this section, we evaluate the performance of our method GND-LO on sequences of 3D lidar scans from the widely-used KITTI dataset [30]. The obtained results are compared with different state-of-the-art algorithms, including ELO [111], MULLS [67], SuMa [5], CLS [96], GICP and P2P-ICP [75]. For all the experiments, the test platform used is a standard PC with 8GB of RAM, running Ubuntu 20.04 with an Intel Core i7-7700HQ CPU with four cores and eight execution threads at 2.8 GHz. We also present an ablation study exploring different configurations of the proposed method.

Testbed

The proposed method was evaluated on the urban sequences included in the KITTI dataset [30]. The Velodyne HDL-64E sensor, running at 10 Hz, provides point clouds that are converted to 1000×64 range images. The error metric employed to compare the different localization algorithms is the one proposed by the dataset, in which the trajectory error is evaluated in all subsequences of different lengths (100, 200, ..., 800). The error is calculated as the translational drift [m] and angular error [°] accumulated every 100m of trajectory. Hence the measurement units are percentage in the case of translational error (m/100m = %), and °/100m in the case of rotational error.

Execution time performance

3D lidar sensors usually work at 10Hz, with some devices reaching up to 20Hz. Thus, for an odometry method to perform in real-time, it has to estimate the motion from the input scans in 100ms or less. It is desirable for odometry algorithms to operate even at a higher frequency since the computational resources of an autonomous vehicle have to be shared between the different processes running simultaneously.

The methods found in the literature are often parallelized and rely on a GPU to perform in real-time [106], especially deep-learning odometry algorithms [54, 98]. The proposed approach GND-LO, without any special parallelization or optimization, can run at 14 Hz on average, allowing for real-time operation even on resource-constrained platforms.

Table 4.5 shows a breakdown of the time taken by the different processes in the proposed workflow. It is evident how the matching and 2D motion estimation are the two most computationally demanding tasks, which is expected given they are part of an iterative process that takes 7.5 iterations on average before convergence.

Table 4.6 includes the average number of planar patches found in each scan, and their distribution based on size and label, to better establish the scale of the problem. Note that only wall patches are employed in the 2D motion optimization, the most time consuming part of the pipeline. Ground

Flatness image creation	6
Patch selection with quadtrees	5.5
Plane fitting to each patch	13
Patch labeling	0.14
Culling set of patches	7.5
Ground plane clustering	0.08
Ground planes registration	0.03
Patch-point Matching	10.5
2D motion estimation	28
Motion filter	0.10
Total	70.85 ms

Table 4.5: Time breakdown in milliseconds of the proposed method.

Size		2	4	8	16	32	**Total**
Outlier	[%]	95	0	0	0	0	55.67
Ground	[%]	0	74	81	96	86	31.32
Wall	[%]	5	26	19	4	14	13.01
All	[nº]	782.00	434.39	104.46	11.89	1.03	1333.77

Table 4.6: Average amount of extracted planar patches per frame, based on their size (pixels) and label. Culled patches are included as outliers.

patches are usually big and abundant, in contrast to wall patches. The effect of the culling is very noticeable, as 95% of the small patches are discarded, keeping only those that are needed to completely constrain the movement.

Accuracy results

Most of the localization algorithms that evaluate themselves via the KITTI dataset are LOAM or SLAM, which clearly outperform simple odometry methods by exploiting information from a map or even performing loop closure. Additionally, deep learning algorithms are usually trained with the same sequences that serve as evaluation, which influences the measured accuracy.

In favor of a fair comparison, and although GND-LO achieves competitive results with those methods in some cases, only non-DL odometry methods that estimate the motion in a frame-to-frame fashion have been considered in the study. This includes MULLS [67], CLS [96], SuMa [5], ELO [111], GICP, and P2P-ICP [75]. Note that while both MULLS and SuMa are SLAM methods, they also report the accuracy of their frame-to-frame odometry module, which is added to this comparison. Table 4.7 shows the error obtained from each method by applying the error metric mentioned before. It should be noted

		Sequence nº scans	00 4540	05 2760	06 1100	07 1100	Avg.
Method	MULLS	[%] [º/100m]	2.36 1.06	2.19 1.01	1.12 0.51	1.65 1.27	2.085 1.006
	CLS	[%] [º/100m]	2.11 0.95	1.98 0.92	0.92 **0.46**	1.04 0.73	1.811 0.859
	SuMa	[%] [º/100m]	2.1 0.9	1.5 0.8	1.0 0.6	1.8 1.2	1.764 0.871
	ELO	[%] [º/100m]	1.14 0.52	0.75 0.43	0.76 0.51	**0.60** **0.48**	0.920 0.488
	GICP	[%] [º/100m]	1.28 0.61	1.18 0.60	0.87 0.53	0.87 0.57	1.156 0.593
	P2P-ICP	[%] [º/100m]	1.57 0.77	1.32 0.76	1.12 0.76	0.95 0.62	1.373 0.749
	GND-LO	[%] [º/100m]	**0.85** **0.49**	**0.59** **0.36**	**0.71** 0.47	0.71 0.66	**0.744** **0.468**

Table 4.7: Accuracy comparison between the different methods evaluated on the KITTI dataset.

that the "Avg." column from the table is the result of averaging the different sequences weighted by their length, given by their number of scans.

It can be seen how the proposed method exhibits a higher accuracy in three of the four test sequences, and clearly outperforms the competing methods in the total average. As for a qualitative evaluation of the proposal, Figure 4.9 includes the trajectories of each sequence. Sequence 06 can be singled out for its higher drift, since less planar information is present along its trajectory. For short and well determined scenarios, as sequences 05 and 07 confirm, the drift is almost unnoticeable for an odometry method. Sequence 00 is the longest of the group, and thus significant drift can be observed near the end of the trajectory, but it is still a good result for odometry.

Ablation study

In this section we study the impact on the accuracy of GND-LO of the different modules of the pipeline. Specifically, four major components are evaluated: i) the motion filter, ii) the culling algorithm, iii) the usage of planar patches from both input scans, and iv) the decoupling of the motion estimation.

The results of this ablation study are displayed in Table 4.8. Although the unaltered method presents the lowest average error, some of the evaluated techniques slightly worsen the accuracy on some sequences. There is a

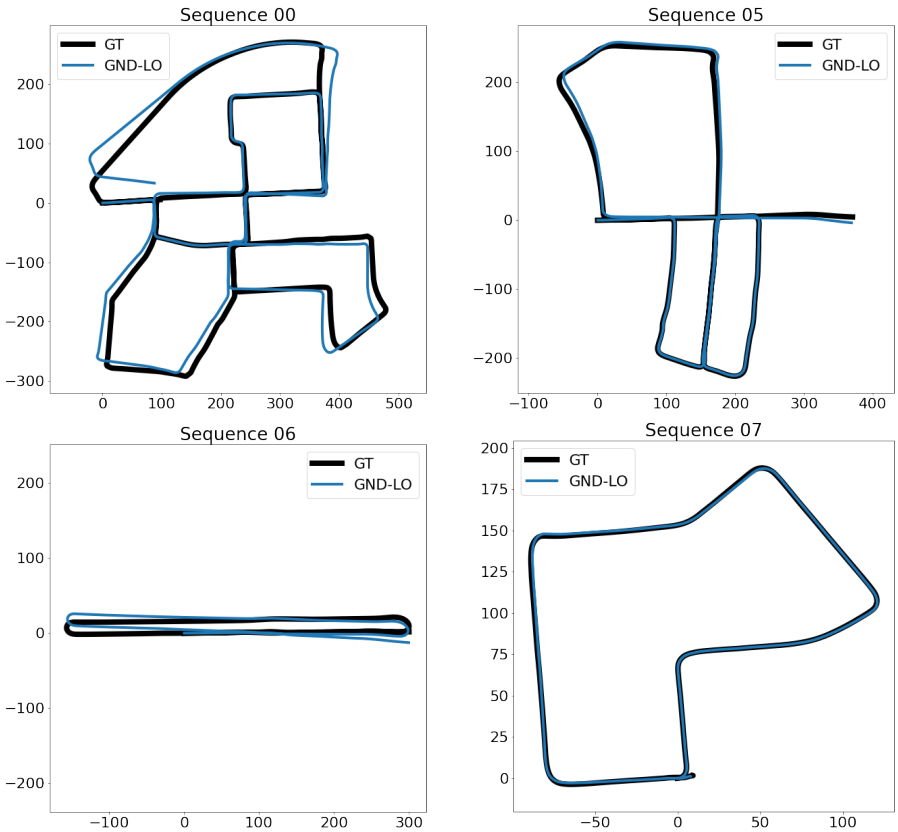


Figure 4.9: Top view of the resulting trajectories over each urban sequence of the dataset.

pattern where the evaluated components produce the most improvement on sequence 06. This sequence, despite being in an urban environment, presents the least flat structures along its trajectory, and thus estimating the motion becomes increasingly challenging. Every part of the algorithm helps dealing with these low-information scenarios, and thus are more noticeable on sequence 06. Adding information from both scans is the most noticeable improvement, more so on this sequence, where every little piece of information becomes crucial to correctly determine the relative pose.

Without decoupling, the method estimates the full 3D motion by minimizing the distance between every patch (ground and wall) and its corresponding point. It becomes a variation of the method explained in Section 4.3, with the addition of quadtree segmentation and processing patches from both scans. Although its higher accuracy on sequence 07, on average it is still worth decou-

		Sequence nº scans	00 4540	05 2760	06 1100	07 1100	Avg.
Method	GND-LO	[%]	**0.85**	**0.59**	**0.71**	0.71	**0.744**
	(original)	[º/100m]	0.49	**0.36**	**0.47**	0.66	**0.468**
	No motion	[%]	0.86	**0.59**	0.77	0.70	0.752
	filter	[º/100m]	**0.47**	0.37	0.48	0.67	**0.468**
	No culling	[%]	**0.85**	0.65	0.96	0.67	0.786
		[º/100m]	**0.47**	0.42	0.67	0.62	0.498
	Patches from	[%]	1.68	0.88	7.17	1.83	2.097
	single scan	[º/100m]	0.86	0.45	2.93	0.79	0.975
	No	[%]	1.06	0.87	1.18	**0.57**	0.961
	decoupling	[º/100m]	0.59	0.52	0.67	**0.53**	0.575

Table 4.8: Ablation study of the proposed method GND-LO.

pling. In addition, the execution time of this "no decoupling" implementation is 106ms, which exceeds the limit for real-time operation. To keep its frequency above 10Hz, only patches from a single scan must be employed, which drives the error up 27% and 53% on translation and rotation respectively. This evidences how the motion decoupling not only reduces the drift but also makes the method more efficient.

4.5 Conclusions

In this chapter, we have introduced a couple of methods for 3D lidar odometry that exploit the information found in the flat surfaces abundant in urban settings. The use of a flatness image is proposed to efficiently select planar patches, further enhanced by the application of quadtree segmentation. After this careful selection, the coplanarity constraint is imposed on each patch and its corresponding point. This way the motion is estimated by minimizing the point-to-plane distance between pairs of features. Correspondences are updated iteratively based on the current estimation of the relative pose, by employing the projection model of the sensor.

The second method goes one step further and decouples the motion estimation in two steps, improving the odometry in terms of both efficiency and drift. First, the ground planes of consecutive scans are registered; then, the 2D motion on the ground plane is recovered by minimizing the point-to-plane distance between pairs of features whose matching is iteratively updated.

To test their suitability, our approaches have been evaluated in popular datasets like Velodyne SLAM and KITTI, and compared to odometry methods from the literature. Not only do our proposals achieve a lower drift over the trajectory, but also run in real-time without the need for a GPU or multithreading like state-of-the-art competitors. This allows for accurate localization while sharing computational resources with other processes running simultaneously, as is desirable for odometry methods.

Based on the results obtained in this work, it is our opinion that these methods would benefit from implementing techniques inherited from LOAM or SLAM algorithms, like scan-to-map registration or loop closure, to further reduce the drift over long sequences.

Even though the use of planar features has proven to be worthwhile, it comes with the limitation of being unable to recover the motion in scenes lacking geometrical information from 3 independent directions. also known as the aperture problem [89]. A clear example of this is a tunnel, in which the movement along it remains undetermined since there are no flat surfaces perpendicular to the trajectory. This will be solved in future works by also exploiting a different type of sensor, like an IMU or an RGB camera.

5

Doppler-capable sensor odometry

This chapter presents a 3D odometry method that recovers the full motion of a vehicle only from a Doppler-capable range sensor. It leverages the radial velocities measured from the scene, estimating the sensor's velocity from a single scan. The vehicle's 3D motion, defined by its linear and angular velocities, is calculated taking into consideration its kinematic model which provides a constraint between the velocity measured at the sensor frame and the vehicle frame. Experiments carried out prove the viability of our single-sensor method compared to having an additional IMU. Our method provides a more reliable translation of the sensor, compared to the errors linked to IMUs due to noise and biases. Its short-term accuracy and fast operation (∼5ms) make it a proper candidate to supply the initialization to more complex localization algorithms or mapping pipelines. Not only does it reduce the error of the mapper, but it does so at a comparable level of accuracy as an IMU would. All without the need to mount and calibrate an extra sensor on the vehicle.

5.1 Introduction

Self-localization is one of the fundamental components of autonomous mobile robots and vehicles. This problem can be tackled by employing different sensors, but cameras and lidars are among the most widely used in contemporary methods. These sensors provide enough advantages to justify their relevance, though their performance can be severely hindered under challenging circumstances, like extreme lighting or weather conditions, or the presence of dust or mist [10].

Radar sensors, on the other hand, are a promising but underexplored alternative able to provide geometric information of its surroundings even in challenging low-visibility scenarios [39,62,97]. This makes them highly suitable for underground operations, construction sites, and other harsh environments. The relevance of radars in industrial and on-road applications has driven sensor technology forward, making them more affordable, accurate, and fast. Among the various advances, it is worth highlighting the introduction of radar sensors measuring in 3D (azimuth, elevation, and range), as well as their ability to estimate the radial velocity of each sensed 3D point. The latter is possible by leveraging precise phase measurements of the returning signal, which allows the sensor to measure the velocity of a point along the radar-point direction. Further on, we denote it as Doppler velocity. Note that this technology has also been implemented in certain lidars, so from now on we will refer to all of them as Doppler-capable range sensors.

The radial velocity provided by these sensors has proven to be advantageous for odometry methods, aiding in the segmentation of dynamic and static objects [49], as well as introducing more constraints to the movement estimation [38]. Given the right conditions, it can be as powerful as recovering the 2D ego-motion of the sensor from a single scan of its surroundings [46, 47]. Note that, to estimate the localization, these works rely upon either knowing the kinematic model of the vehicle or having multiple sensors mounted on the robot, since not all three degrees of freedom (DOFs) of a 2D movement are observable with just the radial velocity of points of the scene. Estimating the movement of the sensor this way removes the need to perform data association, bringing it closer to employing an Inertial Measurement Unit (IMU) or wheel odometry. These methods provide the relative motion of the sensor, and tend to accumulate error over time, also known as *drift*. However, their short-term accuracy obtained in a fast and simple way makes them widely adopted in different odometry solutions.

In this chapter, we propose a 3D odometry method that relies only on the Doppler velocity to estimate the vehicle motion from a single scan, hence avoiding data association. Doppler-capable range sensors can only capture the radial velocity of a point. Assuming most of the scene is static, the linear velocity of the sensor can be recovered via RANSAC [25]. However, there is a lack of information to obtain the 6 DOFs of a 3D movement. The observability

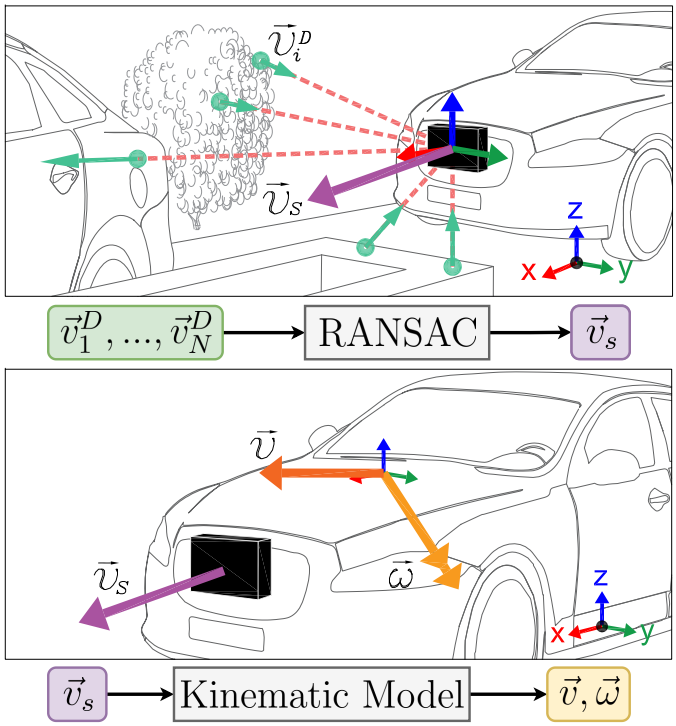


Figure 5.1: Representation of the working principle of the proposed method. Top, the radial velocity $\vec{v}_i^D$ of the observed points in the scene is leveraged to estimate the sensor's linear velocity $\vec{v}_s$. Bottom, the kinematic model provides the relation between $\vec{v}_s$ and both the linear $\vec{v}$ and angular $\vec{\omega}$ velocities of the vehicle.

of this problem was studied by Yoon et al. [107], where a gyroscope was added to completely determine the vehicle's 3D motion. However, in this work we propose a simpler solution closer to the one presented by Kellner [46] but in 3D: by taking into consideration the kinematic model of the vehicle, we can recover its 3D movement using only the Doppler velocities. In the following sections, we describe the proposed odometry algorithm, as well as the experiments performed to validate it. An overview of its working principles can be seen in Figure 5.1. The results obtained prove that the presented method yields a short-term accurate motion estimation. When it is employed as initialization of more advanced localization algorithms like mapper pipelines, it improves the accuracy without needing a different sensor mounted on the vehicle. The code is publicly available at https://github.com/andresgalu/doppler_odometry.

5.2 Related work

As explained in Chapter 2, the first publication to recover the ego-motion of a vehicle employing only the Doppler velocity disregarding the spatial information was due to Yokoo et al. [105]. They propose two sensor configurations to recover the velocity of an Ackermann vehicle, one with two radars in the front of the vehicle and another with a radar and a gyroscopic sensor. This concept was later expanded by Kellner et al. for a 2D radar [46], leveraging the increase in data points to remove the assumption of the scene being static. Still, this method works for vehicles with Ackermann steering. To recover the 2D movement of any vehicle multiple sensors must be mounted [47].

For 3D movement, the idea of employing an IMU (or gyroscope) introduced by Yokoo et al. [105] resurfaces, as seen in publications like Kramer et al. [50] and Doer and Trommer [20]. The observability of the motion is studied by Yoon et al. [107], explaining the need to mount an IMU along the Doppler-capable sensor to recover the 6 DOFs of the movement. In said work, the 3D motion is recovered in a decoupled manner, with the lidar and gyroscope yielding the translation and rotation respectively.

Apart from directly recovering the ego-motion, the Doppler velocity has also proven its value in traditional point cloud registration algorithms. The most common approach is to add a cost function to the optimization problem that compares the measured radial velocity of a point with its expected value given an estimation of the motion. This approach can be implemented into popular registration algorithms like ICP [7] variations [38, 78, 104], Normal Distribution Transform (NDT [8]) [74], and similar approaches [4]. It is worth highlighting the work of Monaco and Brennan [64] for decoupling the motion estimation, using the Doppler velocity to recover the translation and the spatial information for the rotation.

In contrast to the literature that is reviewed above, the method proposed in this article provides the 3D motion of a Doppler-capable range sensor by only leveraging the radial velocities from the scene. By taking into consideration the kinematic model of the vehicle the movement can be recovered without needing an extra IMU. The efficiency and short-term accuracy of this algorithm makes it more than helpful to provide an estimation of the motion, which is also valuable as initialization in more resource-hungry algorithms.

5.3 Method overview

In this section, we explain how the proposed method estimates the 3D movement of the vehicle from the Doppler velocity measurements of points in the scene. The method first estimates the sensor velocity and then finds the vehicle movement that fulfills a certain kinematic model and explains the observed sensor velocity. In Section 5.3.1 we will describe how the velocity of the sensor itself can be calculated from the captured radial velocities of the observed points from the scene. This is possible as long as part of the scene is static, and thus its observed velocity is the opposite of the sensor's. By employing RANSAC [25], dynamic objects can be identified as outliers and rejected. As explained before, from the sensor velocity we cannot obtain the 6 DOFs of the vehicle's 3D motion. By introducing the kinematic model of the vehicle in Section 5.3.2, the DOFs of the movement get reduced and hence it can be solved. Finally, Section 5.3.3 will analyze the variance of the estimated vehicle movement variables.

5.3.1 Sensor velocity

In this section, we will delve into the procedure that yields the sensor velocity from the Doppler velocities of the observed points. First, we will derive the equations assuming the scene is completely static, and then we will tackle the issue of removing dynamic objects.

Assuming a static scene, the velocity of an observed object is the opposite of the sensor velocity $\vec{v}_s$. Since the Doppler velocity from each point v_i^D is only measured along the radial direction, this value is the result of projecting the opposite of the sensor velocity on the radial direction:

$$-v_i^D = \begin{bmatrix} \cos\phi_i \cos\theta_i & \sin\phi_i \cos\theta_i & \sin\theta_i \end{bmatrix} \begin{bmatrix} v_{sx} \\ v_{sy} \\ v_{sz} \end{bmatrix}. \tag{5.1}$$

The azimuth ϕ and elevation angles θ of each point i are obtained by converting their cartesian coordinates to spherical. Applying Equation (5.1) to all sensed points results in a set of linear equations in the form of $B = A\vec{x}$ that can be solved with least squares regression. To reduce the influence of noise in the final result, we weight each point by its signal power. The sensor velocity is thus estimated as follows, with W being the diagonal matrix of the weights:

$$\vec{v}_s = (A^\top W A)^{-1}(A^\top W B). \tag{5.2}$$

In the case there are dynamic objects in the scene, not all the points will follow the same model. In our implementation, we chose RANSAC [25] to get rid of these outlier points that do not comply with the estimated sensor velocity. They are labeled as dynamic, which can be helpful for later processing of the data.

5.3.2 Vehicle kinematic model

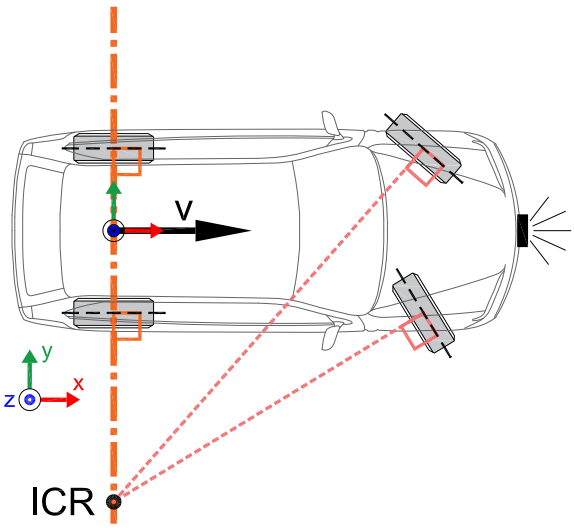
Now that the sensor velocity has been calculated from the Doppler velocity measurements, we can estimate the vehicle movement. There exists an infinite number of 3D motions that explain the calculated sensor velocity, and we lack the information to determine all 6 DOFs. However, by constraining the vehicle motion to a certain kinematic model, it is possible to simplify the movement enough to obtain an estimation.

First of all, we need to define the frame of reference used throughout this section. Its position in the vehicle is not particularly relevant, but its orientation is. As shown in Figure 5.2 we establish the Y-axis to be aligned with the rear wheel axis, pointing to the left of the vehicle. The X-axis is perpendicular to it and points toward the front wheel axis. Finally, the Z-axis points upwards, perpendicular to both X and Y. Without loss of generality, we will locate the frame of reference at the center of the rear wheel axis.

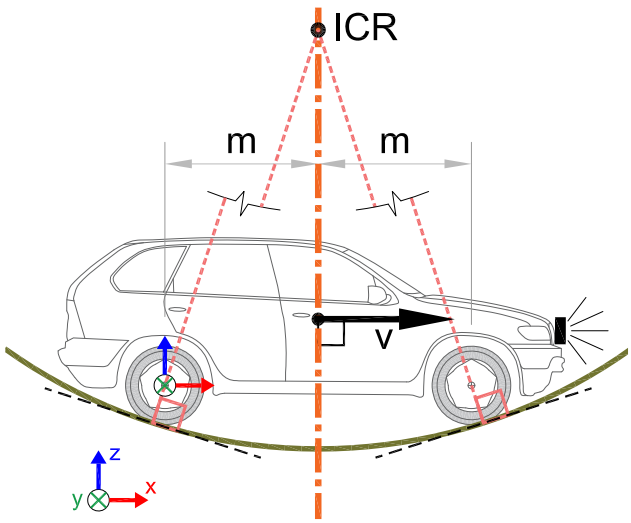
For the sake of clarity, we will start analyzing the movement of an Ackermann steering vehicle moving along a flat surface, which is simply a 2D motion. In that case, the Instant Center of Rotation (ICR) can be found somewhere along the line of contact between the rear wheels and the ground, but it can be well approximated as the line that goes through the rear wheel axis. From now on we will refer to this line as the Z-ICR axis, since it relates to the rotation around Z. The velocity along the Y direction of any point located in the Z-ICR axis is null by definition. See Figure 5.2a for a graphic representation. Note that the key concept here is to have the ICR restricted to a given line, and the Ackermann kinematic model is just an example. As long as the kinematic model fulfills this requirement, like in differential drive and skid-steer vehicles, the viability of the proposed method holds.

For a 3D motion, the movement along Z has to be analyzed too. Similar to the previous paragraph, we can first study the simplified movement where the vehicle only moves in the XZ plane, along a curved surface. The ICR in this case is located at the intersection of the lines perpendicular to the contact point between the wheels and the ground. Assuming the curvature of the ground is locally constant in the relatively short distance between both wheel axes, the ICR will be located in a line parallel to the Z-axis that goes through the midpoint between both wheel axes. Every point located along this Y-ICR axis will consequently have zero velocity in the Z direction. Figure 5.2b shows this concept for better clarity. This holds true even when driving on a flat plane, in which case every point in the vehicle has zero vertical velocity.

Until now, we have studied the rotation around the Z-axis (yaw) and Y-axis (pitch) through two basic movement examples (Figures 5.2a and 5.2b respectively). The roll, or rotation around the X-axis, can be assumed negligible for a vehicle moving on a road. The complete 3D motion can then be seen as a combination of these two basic movements, and the introduced restrictions will still be present in the resulting 3D motion model. Namely, the velocity



(a) Top view of the vehicle movement on the XY plane.



(b) Side view of the vehicle movement on the XZ plane.

Figure 5.2: Simple cases of movement of the vehicle showing the location of the line along which the ICR can be found (orange). The velocity of points located on the ICR axis is perpendicular to it. Black box represents an example of sensor location.

of points located in the Z-ICR axis will have no Y component, and similarly, points in the Y-ICR (the middle of the vehicle along the X-axis) will have zero Z velocity.

We can now recover the motion of the vehicle by applying the mentioned restrictions, knowing that the velocities of two points $\vec{p}, \vec{s}$ of the same rigid moving object are related through their relative position and angular velocity $\vec{\omega}$:

$$\vec{v}_p = \vec{v}_s + \vec{\omega} \times (\vec{p} - \vec{s}). \tag{5.3}$$

Using the previously estimated sensor velocity $\vec{v}_s$ (recall Equation 5.2), along with the restrictions, we can recover the angular velocity vector. To that end, point $\vec{s}$ is now the sensor position, and point $\vec{p}$ belongs to the Z-ICR axis, fulfilling the restriction of having zero Y velocity. Since all points on said axis comply with the requirement, we choose a point $\vec{p}_a$ located at the same Y coordinate as the sensor, for simplicity:

$$\begin{cases} \vec{p}_a = [0, s_y, 0]^\top \\ \vec{v}_a = [v_{ax}, 0, v_{az}]^\top \end{cases}, \tag{5.4}$$

$$0 = v_{sy} - \omega_z s_x + \omega_x s_z. \tag{5.5}$$

Similarly, we can apply the second restriction, which indicates that a point $\vec{p}_b$ located on the Y-ICR (in the middle of the vehicle) has zero Z velocity:

$$\begin{cases} \vec{p}_b = [m, 0, s_z]^\top \\ \vec{v}_b = [v_{bx}, v_{by}, 0]^\top \end{cases}, \tag{5.6}$$

$$0 = v_{sz} - \omega_x s_y - \omega_y (m - s_x), \tag{5.7}$$

where m is half the distance between the two wheel axes of the vehicle, see Figure 5.2b. Finally, we apply the restriction $\omega_x = 0$ since the rotation around the X-axis is negligible, and thus the angular velocity vector can be recovered:

$$\vec{\omega} = \begin{bmatrix} 0 & v_{sz}/(m - s_x) & v_{sy}/s_x \end{bmatrix}^\top. \tag{5.8}$$

The global pose of the sensor with respect to the reference frame of the world can be updated from the previous instance after a period of time Δt by employing the velocity $\vec{v}_s$ for the position $\vec{p}_s$, and the angular velocity $\vec{\omega}$ for the orientation $R \in \mathrm{SO}(3)$. Note that the velocity of any point of the vehicle can be calculated by applying Equation 5.3, and thus its pose can be updated:

$$\begin{cases} R(t + \Delta t) &= R(t) e^{\vec{\omega} \Delta t} \\ \vec{p}_s(t + \Delta t) &= \vec{p}_s(t) + R(t) \vec{v}_s \Delta t \end{cases} \tag{5.9}$$

5.3.3 Covariance analysis

In this section, we analyze the variance of the estimated sensor and angular velocities, to better understand how their accuracy can be improved. First, we start with the sensor velocity $\vec{v}_s$ estimated in Section 5.3.1. Since we employed least squares to find the velocity in Equation 5.2, its covariance matrix C_v can be estimated by propagating the covariance of the measured Doppler velocities C_B:

$$C_v = (A^\top W A)^{-1} C_B. \tag{5.10}$$

Assuming the variance of all measured velocities is the same and independent between them, their covariance matrix can be rewritten as $C_B = \sigma_B^2 I$. An estimation of the value of σ_B can be obtained from the residuals ρ [37]:

$$\rho = A\vec{v}_s - B, \tag{5.11}$$

$$\sigma_B^2 \approx \frac{\rho^\top W \rho}{N-3}. \tag{5.12}$$

Combining Equations 5.12 and 5.10 results in the final estimation of the covariance matrix of the sensor velocity:

$$C_v = \frac{\rho^\top W \rho}{N-3}(A^\top W A)^{-1}. \tag{5.13}$$

Other than the noise of the measurements, the distribution of the points in the scene is highly significant. The more spread they are, the higher the accuracy of the sensor velocity will be. The covariance matrix C_ω of the angular velocity vector can then be calculated via error propagation employing the Jacobian matrix J of Equation 5.8:

$$J = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1/(m - s_x) \\ 0 & 1/s_x & 0 \end{bmatrix}, \tag{5.14}$$

$$C_\omega = J C_v J^\top = \begin{bmatrix} 0 & 0 & 0 \\ 0 & \sigma_{vz}^2/(m - s_x)^2 & \sigma_{vyz}^2/(s_x m - s_x^2) \\ 0 & \sigma_{vyz}^2/(s_x m - s_x^2) & \sigma_{vy}^2/s_x^2 \end{bmatrix}. \tag{5.15}$$

Note the impact the X-location of the sensor on the vehicle has on the covariance. If the sensor's X coordinate coincides with that of either ICR axis, then the related angular velocity cannot be observed. Thus, for better accuracy it is advisable to place the sensor far from these axes along the X direction.

5.4 Experiments and results

In this section, we describe the experiments performed in order to validate the proposed method, and subsequently evaluate it based on the obtained results. The test platform used is a standard PC with 8GB of RAM, running Ubuntu 20.04 with an Intel Core i7-7700HQ CPU with four cores and eight execution threads at 2.8 GHz. The employed setup consists of a Hugin 4D imaging radar, an Ouster OS1-128 3D lidar, and an Xsens MTi-30 IMU all mounted on a Clearpath Husky robotic base. The proposed method makes use only of the radar, while the lidar and IMU are used here for comparison in the evaluation. A dataset was collected while the Husky was driven around sloped terrain. Six different sequences were recorded: 04 and 05 have the most hills and thus the most vertical movement; 06 represents driving on a flat ground; 07 and 08 are similar multifaceted trajectories, with the second having a higher velocity throughout; and 11 is a longer route. Both the environment and the sensor setup can be seen in Figure 5.3, and the dataset is publicly available at https://zenodo.org/record/8346769. Note how the sensor is mounted on a lever arm on the vehicle. This is because increasing the distance along X from the sensor to both ICR axes results in a more accurate angular velocity, as established in Section 5.3.3. On a larger vehicle, like a car, this can be achieved without a lever arm, as can be seen in Figure 5.2.

In contrast to the Ackermann vehicle used as an example in Section 5.3.2, the Husky is a skid-steer vehicle. As explained before, the proposed method works with every vehicle whose Z-ICR is in a consistent location. We are aware of the implications of working with this type of vehicle [61], but the results show that the kinematic model is accurate enough to recover the angular velocity.

5.4.1 Sensor calibration

For the proposed method to properly work, the sensor orientation and position with respect to the vehicle must be known. To that end, we propose a two-step calibration algorithm. First, the vehicle moves in a straight line, forward and backward. We then find the rotation that minimizes both Y and Z velocities calculated by the method, since only the X component should be non-zero. The second step is similar, but in this case, the rotation required is the one around the X-axis, which is not determined by the previous step. Therefore, the vehicle moves freely around a flat surface, to then find the rotation that minimizes the estimated Z velocity. By combining both rotations, the calculated sensor velocity can be transformed into the reference frame of the vehicle.

Just as important as the orientation of the sensor is its position, in particular, the distance along the X-axis from the sensor to the Z-ICR. This distance can be measured, but to avoid the possible error it would introduce, we decided to calibrate s_x employing the IMU data. The procedure is simple: the vehicle moves in a flat circle at a constant speed, and then s_x is chosen to minimize

Figure 5.3: Sensor setup (top) and test environment from sequence 04 (middle and bottom).

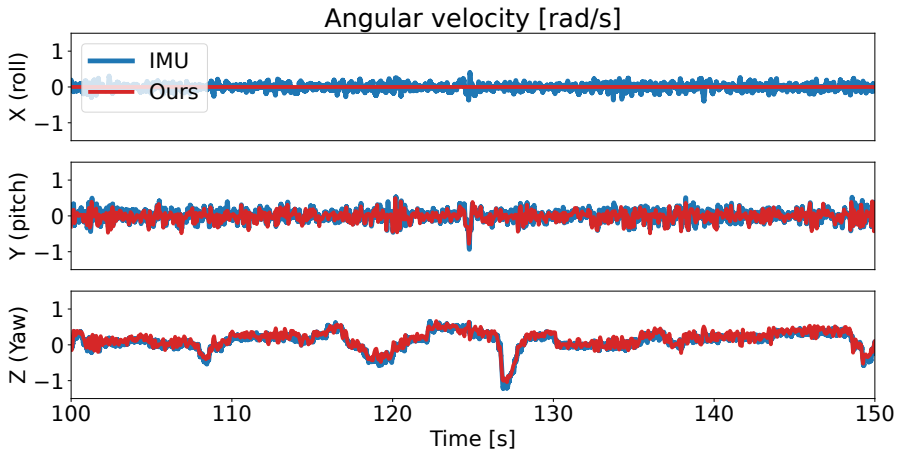


Figure 5.4: Angular velocity estimated from the Doppler velocities compared to IMU measurements, from sequence 11.

the difference between the calculated angular velocity and the one measured by the IMU. Note that this is not the only possible way to calibrate the sensor position, but information about the movement is required either way.

5.4.2 Accuracy results

In this section, we will evaluate the accuracy of the proposed method for providing single-scan and single-sensor 3D motion estimation. We compare the results to an IMU, since both provide a short-term accurate estimation of the motion, but show drift over time. As the ground truth for the comparison, we use the trajectory obtained by running an ICP-based mapper on the 3D lidar data.

An IMU provides information on the angular velocity and linear acceleration of the sensor. Its trajectory can be estimated by integrating these values over time. However, these measurements suffer from having noise and a non-constant bias. Combined with the high frequency at which these sensors operate, said trajectory shows significant drift over time. The position is impacted even more by these issues since two integration steps are performed from the measured linear acceleration [103]. Therefore, the translation estimated from the IMU is not reliable enough to be included in the results.

Our method, in contrast to the IMU integration, obtains the translation of the sensor directly from its estimated velocity. The rotation, however, depends also on the calibration previously mentioned, as well as the fulfillment of the kinematic hypothesis. Our simple approach allows for fast execution, taking on average 4.50 ± 1.94 ms per scan.

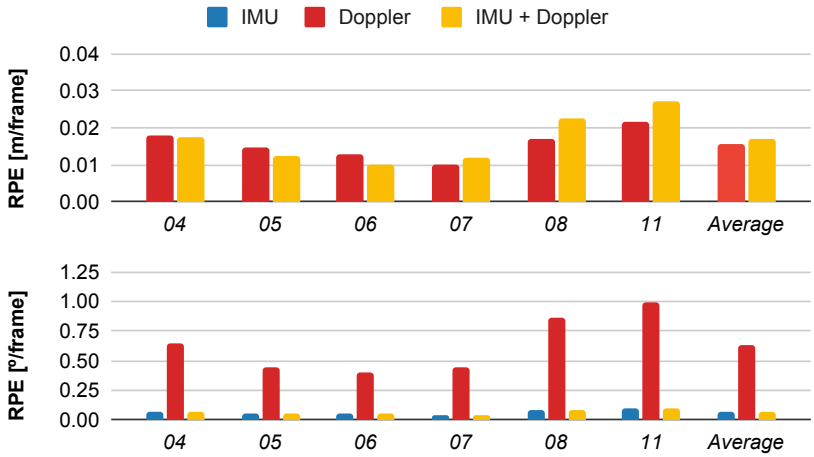


Figure 5.5: Translational and rotational RPE per frame.

Figure 5.4 shows the angular velocities estimated by the proposed method compared to those measured by the IMU. Despite the mentioned limitations, our method provides an estimation similar to the IMU. Notice also how the roll velocity measured by the IMU is not as significant as the yaw and pitch. It is normal to undergo some roll rotation in the rough terrain of the experiments, but it is small enough to consider it inconsequential. On smoother surfaces, like roads, it will be even smaller. This supports the restriction imposed by the kinematic model on the roll.

It makes sense to combine the advantages of both approaches into one, namely the translation from Doppler velocities with the rotation from the IMU, as done by Doer and Trommer [20]. However, this combined method needs information from two different sensors to recover the motion.

To compare these three approaches (IMU, Doppler, IMU + Doppler), we include in Figure 5.5 the resulting Relative Pose Error (RPE) per frame of the test sequences. This evaluates the short-term accuracy of the different methods, without taking into account their drift over time. Regarding the translation, IMU-only has been omitted because of its large error as discussed above, with the trajectory being hundreds of meters away from the ground truth. Both Doppler-only and IMU + Doppler recover the translation in a similar manner, and thus it makes sense for them to provide comparable accuracy. The difference is more apparent in the rotation. The accuracy of our method depends not only on the noise of the input data, unlike IMU. The fulfillment of the kinematic model and the calibration play an important role, hindering the rotation estimation. Sequences 04, 08 and 11 stand out as the most challeng-

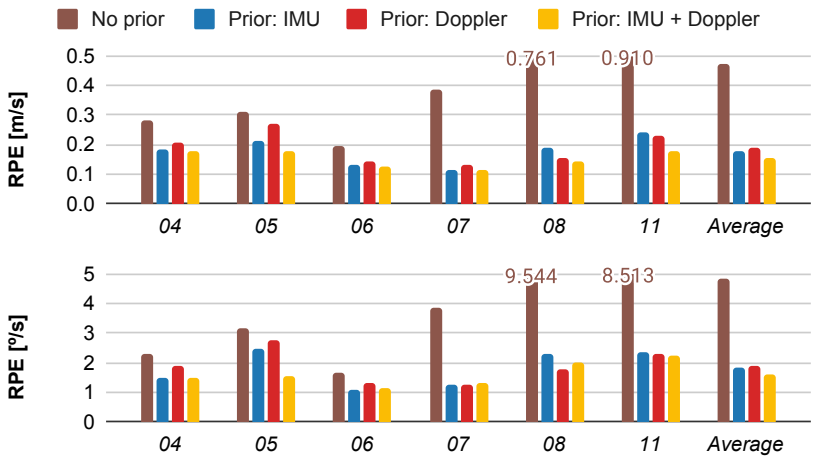


Figure 5.6: RPE per second of the mapper employing different prior data. Values out of the graph are labeled.

ing because of their uneven terrain, causing more vibrations and defying the no-roll assumption.

5.4.3 Initialization of SLAM

The proposed method's short-term accuracy and fast execution make it an interesting candidate to provide a motion prior to a more complex and accurate localization method, e.g. SLAM. ICP [7] is a widespread point cloud registration algorithm used among SLAM methods but is prone to falling into local minima. By providing a motion prior this limitation can be greatly reduced. It is already common in the literature to employ an IMU to initialize the motion.

To evaluate the impact of using the proposed method as initialization for SLAM, we feed the radar data and a motion prior to an ICP-based mapper and analyze its accuracy based on how the motion is initialized. We test 3 different prior estimators: IMU only, Doppler only (our method), and IMU combined with radar. The trajectory of the mapper without any prior is also included as the base case in the comparison. Again, the ground truth used for the comparison is the one obtained from running the mapper on the 3D lidar data. The evaluation metric in this case is the RPE per second, which focuses more on long-term accuracy, a desirable trait for SLAM algorithms.

The resulting error values of the experiments can be seen in Figure 5.6, and the trajectories in Figure 5.7. Despite the variation along different sequences, all methods perform at a similar level of accuracy. What is obvious is the need for a prior input to the mapper, without which it fails. Our method provides

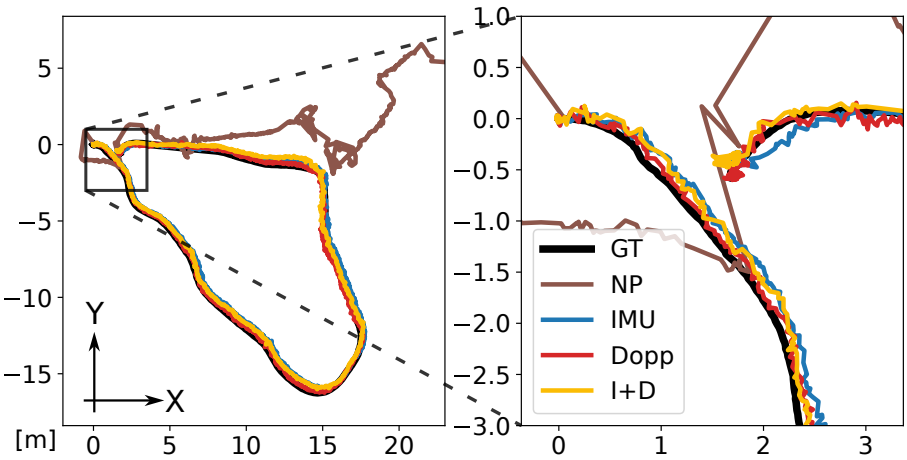


Figure 5.7: Trajectory in meters generated by the mapper with different prior data on sequence 07 (left), and a closer view of the start and finish points (right). Prior used: none (NP), IMU, our method (Dopp), and IMU and Doppler (I+D).

a reliable motion prior, resulting in a similar accuracy to IMU, without the need to mount and calibrate an extra sensor. Furthermore, the proposal can identify and remove dynamic points from the scene, which can help at later stages of the mapper pipeline.

5.5 Conclusions

In this chapter, we have introduced a fast 3D odometry that estimates the motion of a vehicle from a single scan leveraging the Doppler velocity. The method first obtains the sensor velocity based on the measured radial velocities of points from the scene, while rejecting dynamic objects. Based on the kinematic model of the vehicle, the relation between its velocity and the sensor's is established, and thus the 3D motion of the vehicle can be recovered.

A series of experiments have been carried out to validate our method, evaluating the accuracy of the odometry both when used as-is for pose tracking and when used as input to an ICP-based mapping pipeline. We compared its performance with an IMU, given their similar ability to efficiently provide short-term accurate odometry. These qualities make the proposal especially suitable as initialization for radar-only SLAM algorithms, as it boosts the accuracy and identifies dynamic objects at a very low cost. The results prove how our method provides similar accuracy to an IMU without needing an

external sensor mounted on the vehicle. Both the code and the dataset are publicly available.

At the core of the method lies the kinematic model of the vehicle. Granted that it simplifies the movement enough to be observable with a single sensor, it also sets the limitations needed for the proposal to accurately estimate the motion. Challenging scenarios include the Z-ICR being unstable, for example in slippery terrain, bumpy roads with abrupt and uneven vertical variation (potholes), and roads with considerable cant that generate roll rotation. These issues cannot be solved solely through software. On the other hand, the impact of having large amounts of outliers or noisy measurements on the accuracy can be reduced by correctly filtering the data, which should be the focus of future work.

6 Conclusions

This thesis has focused on the problem of the localization of mobile robots, from the perspective of odometry employing range sensors, in particular depth cameras, 3D lidar and Doppler-capable range sensors.

Localization is essential for the robot to safely and properly accomplish its high-level tasks. Among the multiple ways available to provide the localization of the robot, it is worth highlighting the role of odometry. Based on the sensory data captured, odometry can provide an estimation of the ego-motion. Apart from being used directly, it can also be the front end of a SLAM algorithm, increasing the long-term accuracy by leveraging a map of the environment. Odometry can employ different sensors, but the geometric information captured by range sensors comes with certain advantages. This information is less sensitive to lighting changes, and simplifies tasks like obstacle avoidance. In this thesis we proposed odometry methods for different range sensors, each taking advantage of their characteristics.

The first sensor to be addressed was depth cameras in Chapter 3. In it, we proposed an odometry method that takes advantage of the flat surfaces abundantly found in indoor environments, where these sensors operate best. The method exploits the image representation of the point cloud, and does so by combining techniques from both direct and indirect approaches. In summary, planar features are extracted, but they are matched with points of the other image based on reprojection, instead of the usually burdensome correspondence solver. The motion estimation is valid as long as the feature pairs belong to the same surface of the scene. To improve the chances of this happening, the matching is iteratively updated in the refinement step, based on the latest relative pose estimation. The resulting method achieves a higher accuracy than the competition, while maintaining real-time operation.

This odometry method was later adapted to work with 3D lidars. These sensors are commonly employed in autonomous vehicles because of their long-

range, wide field-of-view and dense point clouds. In Chapter 4 we presented the two proposed odometry methods that exploit the image representation of the point clouds provided by modern sensors. The first of the two methods recovers the motion employing planar patches extracted from the point clouds. The patch selection is performed by first calculating the flatness image of the scene, representing how flat the area around each pixel is. Patches are extracted from the pixels with the lowest curvature according to the flatness image. These selected patches are then matched with points from the other image, and the relative pose is estimated as the rigid transformation that minimizes the distance between feature pairs, assuming they fulfill the coplanarity constraint. The matches are iteratively updated based on the latest estimation of the motion, improving the accuracy of the solution. The second presented method builds upon the previous one, enhancing efficiency and accuracy by decoupling the motion estimation. This is possible by categorizing the extracted patches into ground and walls. Ground patches are combined into the ground plane, and compared to the previous instance to estimate three of the six DOFs of the 3D motion. The rest of the movement is recovered by employing the wall patches, and matching them with points of the other image. To further enhance feature extraction, quadtree segmentation is applied to the flatness image, resulting in a set of patches with different sizes located on the flat surfaces of the scene.

The last odometry method, described in Chapter 5, works with Doppler-capable range sensors. Thanks to precise phase measurements of the returning signal, these sensors are capable of capturing the radial velocity of points in the scene. The use of this information has proven to be helpful for odometry methods, from adding constraints to traditional approaches to providing an estimation of the ego-motion. Our method exploits the radial velocities to estimate the velocity of the vehicle equipped with the Doppler-capable sensor. Instead of employing an IMU to completely observe the 3D motion as is common in the literature, we leverage the kinematic model of the vehicle. The result is a very fast odometry method with high short-term accuracy. It can be used as is to recover the motion of the vehicle, or as the initialization of more resource-hungry localization algorithms that benefit from having a prior estimation of the movement.

Various routes could be researched in future works to advance the range odometry topic. First, it is worth stressing the importance of optimization and multi-threading. Execution time and computational resources are always limiting the performance of odometry algorithms. For example, the omnipresent ICP has trouble processing big point clouds, and thus its variants usually depend on downsampling them, losing information along the way. Ideally, having infinite resources and time translates into a higher accuracy by using more information, better preprocessing of the data, and going through more iterations. The optimization and multi-threading of odometry methods can help in this regard, improving the resulting localization.

With the most basic and obvious improvement out of the way, it is the turn for SLAM. Odometry is very interesting on its own, though combining it with SLAM algorithms provides better localization. Comparing new point clouds to the map, as well as loop closure, are the most significant SLAM component (apart from odometry), which enable long-term accuracy and little to no drift. We propose creating SLAM algorithms for the odometry methods presented in this thesis, exploiting their unique characteristics like the use of planar patches of different sizes, or the segmentation of the ground plane.

Another common limitation for most range odometry methods is the one defined as the aperture problem [89]. There will be scenarios where there is not enough information captured by the range sensors to make the motion fully observable. Examples of this include tunnels and corridors, where the movement of the sensor along it can be undetermined. To solve this, the use of other sensors could be studied. By mounting sensors like an IMU or a camera, more information about the motion would be available, reducing the probability of suffering the aperture problem.

Finally, we would like to include segmentation as another future work related to odometry. Segmentation algorithms tend to be slow, hindering the real-time capabilities of odometry methods. However, segmenting objects in the scene provides high-level information in comparison to an unordered point cloud. This can be useful to improve the accuracy and efficiency, as proven by the second article of Chapter 4, in which the ground plane was segmented for better performance.

Bibliography

[1] Philipp Adis, Nicolas Horst, and Mathias Wien. D3DLO: Deep 3D LiDAR odometry. In *2021 IEEE International Conference on Image Processing (ICIP)*, pages 3128–3132. IEEE, 2021.

[2] Daniel Adolfsson, Martin Magnusson, Anas Alhashimi, Achim J Lilienthal, and Henrik Andreasson. CFEAR radarodometry-conservative filtering for efficient and accurate radar odometry. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5462–5469. IEEE, 2021.

[3] Sameer Agarwal, Keir Mierle, and The Ceres Solver Team. Ceres Solver. https://github.com/ceres-solver/ceres-solver, March 2022.

[4] Michael Barjenbruch, Dominik Kellner, Jens Klappstein, Juergen Dickmann, and Klaus Dietmayer. Joint spatial- and Doppler-based ego-motion estimation for automotive radars. In *2015 IEEE Intelligent Vehicles Symposium (IV)*, pages 839–844. IEEE, 2015.

[5] Jens Behley and Cyrill Stachniss. Efficient surfel-based SLAM using 3D laser range data in urban environments. In *Proceedings of Robotics: Science and Systems*, Pittsburgh, Pennsylvania, June 2018.

[6] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9):509–517, sep 1975.

[7] Paul J Besl and Neil D McKay. Method for registration of 3-D shapes. In *Sensor fusion IV: control paradigms and data structures*, volume 1611, pages 586–606. International Society for Optics and Photonics, 1992.

[8] P. Biber and W. Strasser. The normal distributions transform: a new approach to laser scan matching. In *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, volume 3, pages 2743–2748 vol.3, 2003.

[9] Guillaume Bresson, Zayed Alsayed, Li Yu, and Sébastien Glaser. Simultaneous localization and mapping: A survey of current trends in autonomous driving. *IEEE Transactions on Intelligent Vehicles*, 2(3):194–220, 2017.

[10] Alexander Carballo, Jacob Lambert, Abraham Monrroy, David Wong, Patiphon Narksri, Yuki Kitsukawa, Eijiro Takeuchi, Shinpei Kato, and Kazuya Takeda. LIBRE: The multiple 3D LiDAR dataset. In *2020 IEEE Intelligent Vehicles Symposium (IV)*, pages 1094–1101. IEEE, 2020.

[11] Nicholas Carlevaris-Bianco, Arash K. Ushani, and Ryan M. Eustice. University of Michigan North Campus long-term vision and lidar dataset. *International Journal of Robotics Research*, 35(9):1023–1035, 2015.

[12] Ming-Fang Chang, John Lambert, Patsorn Sangkloy, Jagjeet Singh, Slawomir Bak, Andrew Hartnett, De Wang, Peter Carr, Simon Lucey, Deva Ramanan, et al. Argoverse: 3D tracking and forecasting with rich maps. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8748–8757, 2019.

[13] Baifan Chen, Chunfa Liu, Yu Tong, and Qian Wu. Robust RGB-D visual odometry based on planar features. In *2017 IEEE 7th Annual International Conference on CYBER Technology in Automation, Control, and Intelligent Systems (CYBER)*, pages 449–453, 2017.

[14] Pengxin Chen, Wenzhong Shi, Sheng Bao, Muyang Wang, Wenzheng Fan, and Haodong Xiang. Low-drift odometry, mapping and ground segmentation using a backpack LiDAR system. *IEEE Robotics and Automation Letters*, 6(4):7285–7292, 2021.

[15] Yang Chen and Gérard Medioni. Object modelling by registration of multiple range images. *Image and vision computing*, 10(3):145–155, 1992.

[16] Kok Seng Chong and L. Kleeman. Accurate odometry and error modelling for a mobile robot. In *Proceedings of International Conference on Robotics and Automation*, volume 4, pages 2783–2788 vol.4, 1997.

[17] Carlo Dal Mutto, Pietro Zanuttigh, and Guido M Cortelazzo. *Time-of-flight cameras and Microsoft KinectTM*. Springer Science & Business Media, 2012.

[18] Jean-Emmanuel Deschaud. IMLS-SLAM: Scan-to-model matching based on 3D data. In *IEEE International Conference on Robotics and Automation*, pages 2480–2485. IEEE, 2018.

[19] Albert Diosi and Lindsay Kleeman. Fast laser scan matching using polar coordinates. *The International Journal of Robotics Research*, 26(10):1125–1153, 2007.

[20] Christopher Doer and Gert F. Trommer. An EKF based approach to radar inertial odometry. In *2020 IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems (MFI)*, pages 152–159, 2020.

[21] Ivan Dryanovski, Roberto G. Valenti, and Jizhong Xiao. Fast visual odometry and mapping from RGB-D data. In *2013 IEEE International Conference on Robotics and Automation*, pages 2305–2310, 2013.

[22] Jakob Engel, Vladlen Koltun, and Daniel Cremers. Direct sparse odometry. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(3):611–625, 2018.

[23] Scott Ettinger, Shuyang Cheng, Benjamin Caine, Chenxi Liu, Hang Zhao, Sabeek Pradhan, Yuning Chai, Ben Sapp, Charles R Qi, Yin Zhou, et al. Large scale interactive motion forecasting for autonomous driving: The Waymo open motion dataset. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 9710–9719, 2021.

[24] Mark Fiala and Alex Ufkes. Visual odometry using 3-dimensional video input. In *2011 Canadian Conference on Computer and Robot Vision*, pages 86–93, 2011.

[25] Martin A Fischler and Robert C Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981.

[26] Patrick J Flynn and Anil K Jain. On reliable curvature estimation. In *CVPR*, volume 88, pages 5–9, 1989.

[27] Andres Galeote-Luque, Jose-Raul Ruiz-Sarmiento, and Javier Gonzalez-Jimenez. Efficient 3D lidar odometry based on planar patches. *Sensors*, 22(18):6976, 2022.

[28] Andres Galeote-Luque, Jose-Raul Ruiz-Sarmiento, and Javier Gonzalez-Jimenez. Método de odometría basada en planos para cámaras de profundidad. In *Jornadas Robótica, Educación Bioingeniería*, Málaga, 2022.

[29] Andres Galeote-Luque, Jose-Raul Ruiz-Sarmiento, and Javier Gonzalez-Jimenez. GND-LO: Ground decoupled 3D lidar odometry based on planar patches. *IEEE Robotics and Automation Letters*, 8(11):6923–6930, 2023.

[30] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics: The KITTI dataset. *International Journal of Robotics Research*, 2013.

[31] Javier Gonzalez and Rafael Gutierrez. Direct motion estimation from a range scan sequence. *Journal of Robotic Systems*, 16(2):73–80, 1999.

[32] W Shane Grant, Randolph C Voorhies, and Laurent Itti. Efficient velodyne SLAM with point and plane features. *Autonomous Robots*, 43(5):1207–1224, 2019.

[33] Giorgio Grisetti, Cyrill Stachniss, and Wolfram Burgard. Improved techniques for grid mapping with Rao-Blackwellized particle filters. *IEEE Transactions on Robotics*, 23(1):34–46, 2007.

[34] Seung-Jun Han, Jungyu Kang, Kyoung-Wook Min, and Jungdan Choi. DiLO: Direct light detection and ranging odometry based on spherical range images for autonomous driving. *ETRI Journal*, 43(4):603–616, 2021.

[35] Ankur Handa, Thomas Whelan, John McDonald, and Andrew J. Davison. A benchmark for RGB-D visual odometry, 3D reconstruction and SLAM. In *2014 IEEE International Conference on Robotics and Automation*, pages 1524–1531, 2014.

[36] Miles Hansard, Seungkyu Lee, Ouk Choi, and Radu Patrice Horaud. *Time-of-flight cameras: principles, methods and applications*. Springer Science & Business Media, 2012.

[37] Trevor Hastie, Robert Tibshirani, Jerome H Friedman, and Jerome H Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, volume 2. Springer, 2009.

[38] Bruno Hexsel, Heethesh Vhavle, and Yi Chen. DICP: Doppler iterative closest point algorithm. *arXiv preprint arXiv:2201.11944*, 2022.

[39] Ziyang Hong, Yvan Petillot, Andrew Wallace, and Sen Wang. RadarSLAM: A robust simultaneous localization and mapping system for all weather conditions. *The International Journal of Robotics Research*, 41(5):519–542, 2022.

[40] Albert S Huang, Abraham Bachrach, Peter Henry, Michael Krainin, Daniel Maturana, Dieter Fox, and Nicholas Roy. Visual odometry and mapping for autonomous flight using an RGB-D camera. In *Robotics Research: The 15th International Symposium ISRR*, pages 235–252. Springer, 2017.

[41] Xiaoshui Huang, Guofeng Mei, Jian Zhang, and Rana Abbas. A comprehensive survey on point cloud registration, 2021.

[42] Mariano Jaimez and Javier Gonzalez-Jimenez. Fast visual odometry for 3-D range sensors. *IEEE Transactions on Robotics*, 31(4):809–822, 2015.

[43] Mariano Jaimez, Javier Monroy, Manuel Lopez-Antequera, and Javier Gonzalez-Jimenez. Robust planar odometry based on symmetric range flow and multiscan alignment. *IEEE Transactions on Robotics*, 34(6):1623–1635, 2018.

[44] Mariano Jaimez, Javier G. Monroy, and Javier Gonzalez-Jimenez. Planar odometry from a radial laser scanner. a range flow-based approach. In *2016 IEEE International Conference on Robotics and Automation*, pages 4479–4485, 2016.

[45] Jinyong Jeong, Younggun Cho, Young-Sik Shin, Hyunchul Roh, and Ayoung Kim. Complex urban dataset with multi-level sensors from highly diverse urban environments. *International Journal of Robotics Research*, 38(6):642–657, 2019.

[46] Dominik Kellner, Michael Barjenbruch, Jens Klappstein, Jürgen Dickmann, and Klaus Dietmayer. Instantaneous ego-motion estimation using Doppler radar. In *16th International IEEE Conference on Intelligent Transportation Systems (ITSC 2013)*, pages 869–874. IEEE, 2013.

[47] Dominik Kellner, Michael Barjenbruch, Jens Klappstein, Jürgen Dickmann, and Klaus Dietmayer. Instantaneous ego-motion estimation using multiple Doppler radars. In *2014 IEEE International Conference on Robotics and Automation*, pages 1592–1597. IEEE, 2014.

[48] Christian Kerl, Jürgen Sturm, and Daniel Cremers. Robust odometry estimation for RGB-D cameras. In *2013 IEEE International Conference on Robotics and Automation*, pages 3748–3754, 2013.

[49] Aaron Kingery and Dezhen Song. Improving ego-velocity estimation of low-cost Doppler radars for vehicles. *IEEE Robotics and Automation Letters*, 7(4):9445–9452, 2022.

[50] Andrew Kramer, Carl Stahoviak, Angel Santamaria-Navarro, Ali-Akbar Agha-Mohammadi, and Christoffer Heckman. Radar-inertial ego-velocity estimation for visually degraded environments. In *2020 IEEE*

International Conference on Robotics and Automation, pages 5739–5746. IEEE, 2020.

[51] Seong Hun Lee and Javier Civera. What's wrong with the absolute trajectory error? *arXiv preprint arXiv:2212.05376*, 2023.

[52] Larry Li. Time-of-flight camera — An introduction. Technical Report SLOA190B, Texas Instruments Incorporated, Dallas, Texas, 2014.

[53] Qing Li, Shaoyang Chen, Cheng Wang, Xin Li, Chenglu Wen, Ming Cheng, and Jonathan Li. LO-Net: Deep real-time lidar odometry. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8473–8482, 2019.

[54] Zhichao Li and Naiyan Wang. DMLO: Deep matching lidar odometry. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6010–6017. IEEE, 2020.

[55] Shuang Liang, Zhiqiang Cao, Peiyu Guan, Chengpeng Wang, Junzhi Yu, and Shuo Wang. A novel sparse geometric 3-D lidar odometry approach. *IEEE Systems Journal*, 15(1):1390–1400, 2020.

[56] Yan Lu and Dezhen Song. Robust RGB-D odometry using point and line features. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, December 2015.

[57] Steve Macenski, Francisco Martín, Ruffin White, and Jonatan Ginés Clavero. The marathon 2: A navigation system. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020.

[58] Steve Macenski, Tom Moore, David V Lu, Alexey Merzlyakov, and Michael Ferguson. From the desks of ROS maintainers: A survey of modern & capable mobile robotics algorithms in the robot operating system 2. *Robotics and Autonomous Systems*, 2023.

[59] Martin Magnusson, Achim Lilienthal, and Tom Duckett. Scan registration for autonomous mining vehicles using 3D-NDT. *Journal of Field Robotics*, 24(10):803–827, 2007.

[60] Martin Magnusson, Andreas Nuchter, Christopher Lorken, Achim J Lilienthal, and Joachim Hertzberg. Evaluation of 3D registration reliability and speed-a comparison of ICP and NDT. In *2009 IEEE International Conference on Robotics and Automation*, pages 3907–3912. IEEE, 2009.

[61] Jorge L Martínez, Jesús Morales, Jesús M García, and Alfonso García-Cerezo. Analysis of tread ICRs for wheeled skid-steer vehicles on inclined terrain. *IEEE Access*, 11:547–555, 2022.

[62] Malcolm Mielle, Martin Magnusson, and Achim J Lilienthal. A comparative analysis of radar and lidar sensing for localization and mapping. In *2019 European Conference on Mobile Robots (ECMR)*, pages 1–6. IEEE, 2019.

[63] Sherif A. S. Mohamed, Mohammad-Hashem Haghbayan, Tomi Westerlund, Jukka Heikkonen, Hannu Tenhunen, and Juha Plosila. A survey on odometry for autonomous navigation systems. *IEEE Access*, 7:97466–97486, 2019.

[64] Chris D Monaco and Sean N Brennan. Radarodo: Ego-motion estimation from Doppler and spatial data in radar images. *IEEE Transactions on Intelligent Vehicles*, 5(3):475–484, 2020.

[65] Frank Moosmann and Christoph Stiller. Velodyne SLAM. In *Proceedings of the IEEE Intelligent Vehicles Symposium*, pages 393–398, Baden-Baden, Germany, June 2011.

[66] D. Nister, O. Naroditsky, and J. Bergen. Visual odometry. In *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2004. CVPR 2004.*, volume 1, pages I–I, 2004.

[67] Yue Pan, Pengchuan Xiao, Yujie He, Zhenlei Shao, and Zesong Li. MULLS: Versatile LiDAR SLAM via multi-metric linear least square. In *IEEE International Conference on Robotics and Automation*, pages 11633–11640, 2021.

[68] Prabin Kumar Panigrahi and Sukant Kishoro Bisoy. Localization strategies for autonomous mobile robots: A review. *Journal of King Saud University - Computer and Information Sciences*, 34(8, Part B):6019–6039, 2022.

[69] Yeong Sang Park, Young-Sik Shin, Joowan Kim, and Ayoung Kim. 3D ego-motion estimation using low-cost mmwave radars via radar velocity factor for pose-graph SLAM. *IEEE Robotics and Automation Letters*, 6(4):7691–7698, 2021.

[70] Sai Manoj Prakhya, Liu Bingbing, Lin Weisi, and Usman Qayyum. Sparse depth odometry: 3D keypoint based pose estimation from dense depth data. In *2015 IEEE International Conference on Robotics and Automation*, pages 4216–4223, 2015.

[71] Thinal Raj, Fazida Hanim Hashim, Aqilah Baseri Huddin, Mohd Faisal Ibrahim, and Aini Hussain. A survey on LiDAR scanning mechanisms. *Electronics*, 9(5):741, 2020.

[72] Srikumar Ramalingam and Yuichi Taguchi. A theory of minimal 3D point to 3D plane registration and its generalization. *International journal of computer vision*, 102(1):73–90, 2013.

[73] Carolina Raposo, Miguel Lourenco, Michel Antunes, and Joao Barreto. Plane-based odometry using an RGB-D camera. In *Proceedings of the British Machine Vision Conference*. BMVA Press, 2013.

[74] Matthias Rapp, Michael Barjenbruch, Markus Hahn, Jürgen Dickmann, and Klaus Dietmayer. Probabilistic ego-motion estimation using multiple automotive radar sensors. *Robotics and Autonomous Systems*, 89:136–146, 2017.

[75] Andrzej Reinke, Xieyuanli Chen, and Cyrill Stachniss. Simple but effective redundant odometry for autonomous vehicles. In *IEEE International Conference on Robotics and Automation*, pages 9631–9637. IEEE, 2021.

[76] Andrzej Reinke, Matteo Palieri, Benjamin Morrell, Yun Chang, Kamak Ebadi, Luca Carlone, and Ali-akbar Agha-mohammadi. LOCUS 2.0: Robust and computationally efficient LiDAR odometry for real-time underground 3D mapping. *arXiv preprint arXiv:2205.11784*, 2022.

[77] Fraser Rennie, David Williams, Paul Newman, and Daniele De Martini. Doppler-aware odometry from FMCW scanning radar. *arXiv preprint arXiv:2308.10597*, 2023.

[78] Kyle Retan, Frasher Loshaj, and Michael Heizmann. Radar odometry on $SE(3)$ with constant velocity motion prior. *IEEE Robotics and Automation Letters*, 6(4):6386–6393, 2021.

[79] Hermann Rohling. Ordered statistic CFAR technique - an overview. In *2011 12th International Radar Symposium (IRS)*, pages 631–638, 2011.

[80] Ricardo Roriz, Jorge Cabral, and Tiago Gomes. Automotive LiDAR technology: A survey. *IEEE Transactions on Intelligent Transportation Systems*, 23(7):6282–6297, 2021.

[81] Radu Bogdan Rusu and Steve Cousins. 3D is here: Point Cloud Library (PCL). In *IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China, May 9-13 2011.

[82] Hanan Samet. The quadtree and related hierarchical data structures. *ACM Computing Surveys (CSUR)*, 16(2):187–260, 1984.

[83] Davide Scaramuzza and Friedrich Fraundorfer. Visual odometry [tutorial]. *IEEE Robotics and Automation Magazine*, 18(4):80–92, 2011.

[84] A. Segal, D. Haehnel, and S. Thrun. Generalized-ICP. In *Proceedings of Robotics: Science and Systems*, Seattle, USA, June 2009.

[85] Dong-Uk Seo, Hyungtae Lim, Seungjae Lee, and Hyun Myung. PaGO-LOAM: Robust ground-optimized LiDAR odometry. In *2022 19th International Conference on Ubiquitous Robots (UR)*, pages 1–7, 2022.

[86] James Servos and Steven L Waslander. Multi-channel Generalized-ICP: A robust framework for multi-channel scan registration. *Robotics and Autonomous systems*, 87:247–257, 2017.

[87] Tixiao Shan and Brendan Englot. LeGO-LOAM: Lightweight and ground-optimized lidar odometry and mapping on variable terrain. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4758–4765. IEEE, 2018.

[88] Tixiao Shan, Brendan Englot, Drew Meyers, Wei Wang, Carlo Ratti, and Daniela Rus. LIO-SAM: Tightly-coupled lidar inertial odometry via smoothing and mapping. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5135–5142. IEEE, 2020.

[89] Shinsuke Shimojo, Gerald H Silverman, and Ken Nakayama. Occlusion and the solution to the aperture problem for motion. *Vision research*, 29(5):619–626, 1989.

[90] Merrill Ivan Skolnik. *Radar Handbook*. McGraw-Hill Professional Publishing, 1970.

[91] Frank Steinbrücker, Jürgen Sturm, and Daniel Cremers. Real-time visual odometry from dense RGB-D images. In *2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops)*, pages 719–722, 2011.

[92] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers. A benchmark for the evaluation of RGB-D SLAM systems. In *Proc. of the International Conference on Intelligent Robot Systems (IROS)*, Oct. 2012.

[93] Pei Sun, Henrik Kretzschmar, Xerxes Dotiwalla, Aurelien Chouard, Vijaysai Patnaik, Paul Tsui, James Guo, Yin Zhou, Yuning Chai, Benjamin Caine, et al. Scalability in perception for autonomous driving: Waymo open dataset. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 2446–2454, 2020.

[94] Qinxuan Sun, Jing Yuan, Xuebo Zhang, and Fengchi Sun. RGB-D SLAM in indoor environments with STING-based plane feature extraction. *IEEE/ASME Transactions on Mechatronics*, 23(3):1071–1082, 2018.

[95] Yuichi Taguchi, Yong-Dian Jian, Srikumar Ramalingam, and Chen Feng. Point-plane SLAM for hand-held 3D sensors. In *2013 IEEE International Conference on Robotics and Automation*, pages 5182–5189, 2013.

[96] Martin Velas, Michal Spanel, and Adam Herout. Collar line segments for fast odometry estimation from Velodyne point clouds. In *IEEE International Conference on Robotics and Automation*, pages 4486–4495. IEEE, 2016.

[97] Arthur Venon, Yohan Dupuis, Pascal Vasseur, and Pierre Merriaux. Millimeter wave FMCW radars for perception, recognition and localization in automotive applications: A survey. *IEEE Transactions on Intelligent Vehicles*, 7(3):533–555, 2022.

[98] Guangming Wang, Xinrui Wu, Shuyang Jiang, Zhe Liu, and Hesheng Wang. Efficient 3D deep lidar odometry. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2022.

[99] Cort J Willmott and Kenji Matsuura. Advantages of the mean absolute error (MAE) over the root mean square error (RMSE) in assessing average model performance. *Climate research*, 30(1):79–82, 2005.

[100] Benjamin Wilson, William Qi, Tanmay Agarwal, John Lambert, Jagjeet Singh, Siddhesh Khandelwal, Bowen Pan, Ratnesh Kumar, Andrew Hartnett, Jhony Kaesemodel Pontes, et al. Argoverse 2: Next generation datasets for self-driving perception and forecasting. *arXiv preprint arXiv:2301.00493*, 2023.

[101] Volker Winkler. Range Doppler detection for automotive FMCW radars. In *2007 European Radar Conference*, pages 166–169, 2007.

[102] Volker Winkler. Range doppler detection for automotive fmcw radars. In *2007 European Radar Conference*, pages 166–169, 2007.

[103] Oliver J Woodman. An introduction to inertial navigation. Technical report, University of Cambridge, Computer Laboratory, 2007.

[104] Yuchen Wu, David J Yoon, Keenan Burnett, Soeren Kammel, Yi Chen, Heethesh Vhavle, and Timothy D Barfoot. Picking up speed: Continuous-time lidar-only odometry using Doppler velocity measurements. *IEEE Robotics and Automation Letters*, 8(1):264–271, 2022.

[105] Kaoru Yokoo, Stephane Beauregard, and Martin Schneider. Indoor relative localization with mobile short-range radar. In *VTC Spring 2009-IEEE 69th Vehicular Technology Conference*, pages 1–5. IEEE, 2009.

[106] Masashi Yokozuka, Kenji Koide, Shuji Oishi, and Atsuhiko Banno. LiTAMIN2: Ultra light LiDAR-based SLAM using geometric approximation applied with KL-divergence. In *IEEE International Conference on Robotics and Automation*, pages 11619–11625, 2021.

[107] David J Yoon, Keenan Burnett, Johann Laconte, Yi Chen, Heethesh Vhavle, Soeren Kammel, James Reuther, and Timothy D Barfoot. Need for speed: Fast correspondence-free lidar odometry using Doppler velocity. *arXiv preprint arXiv:2303.06511*, 2023.

[108] Ji Zhang and Sanjiv Singh. LOAM: Lidar odometry and mapping in real-time. In *Proceedings of Robotics: Science and Systems (RSS '14)*, July 2014.

[109] Ji Zhang and Sanjiv Singh. Visual-lidar odometry and mapping: Low-drift, robust, and fast. In *IEEE International Conference on Robotics and Automation*, pages 2174–2181. IEEE, 2015.

[110] Zhengyou Zhang. Microsoft Kinect sensor and its effect. *IEEE MultiMedia*, 19(2):4–10, 2012.

[111] Xin Zheng and Jianke Zhu. Efficient LiDAR odometry for autonomous driving. *IEEE Robotics and Automation Letters*, 6(4):8458–8465, 2021.

[112] Qian-Yi Zhou, Jaesik Park, and Vladlen Koltun. Open3D: A modern library for 3D data processing. *arXiv:1801.09847*, 2018.

[113] David Zuñiga-Noël, J. R. Ruiz-Sarmiento, and Javier Gonzalez-Jimenez. *Intrinsic Calibration of Depth Cameras for Mobile Robots using a Radial Laser Scanner*, volume 11678 of *Computer Analysis of Images and Patterns. Lecture Notes in Computer Science*, pages 659–671. Springer International Publishing, 2019.