NATIONAL TECHNICAL UNIVERSITY OF ATHENS
SCHOOL OF ELECTRICAL AND COMPUTER ENGINEERING

DIVISION OF SIGNALS, CONTROL AND ROBOTICS

COMPUTER VISION, SPEECH COMMUNICATION AND SIGNAL PROCESSING GROUP

COMBINING SEMANTICS WITH UNIFIED GEOMETRIC REPRESENTATIONS FOR INDOOR SLAM

DIPLOMA THESIS

of

Ioannis Asmanis

Supervisor: Petros Maragos
Prof. NTUA

Athens, April 2021

National Technical University of Athens
SCHOOL OF ELECTRICAL AND COMPUTER ENGINEERING
DIVISION OF SIGNALS, CONTROL AND ROBOTICS
COMPUTER VISION, SPEECH COMMUNICATION AND SIGNAL PROCESSING GROUP

COMBINING SEMANTICS WITH UNIFIED GEOMETRIC REPRESENTATIONS FOR INDOOR SLAM

DIPLOMA THESIS

of

Ioannis Asmanis

Supervisor: Petros Maragos
Prof. NTUA

Approved by the examining committee on 2021-04-15.

.....	.....	.....
P. Maragos, Prof. NTUA	C. Tzafestas, Assoc. Prof. NTUA	H. Psilakis, Lect. NTUA

Athens, April 2021

.....
Ioannis Asmanis
Electrical and Computer Engineer

Abstract

One of the key problems of modern robotics research is Simultaneous Localization And Mapping (SLAM). This challenging area has been of great interest for mobile robotics, where agents rely on on-board sensors to accurately estimate their environments and robustly localize within them. From initial Visual Odometry (VO) solutions, which do not produce environment maps and suffer from accumulated trajectory drift, the state-of-the-art has advanced to SLAM theory, which executes mapping and loop-closing, enabling robots to self-correct their estimations when revisiting known locations. Probabilistic frameworks supported this progress, allowing for optimal responses to measurement noise. Today, with access to ever more potent hardware, the community uses a variety of sensors and sophisticated algorithms, including some from Machine Learning (ML), to extract a rich variety of additional data, which can be used to support core SLAM functions. This motivates us to create a mixed framework, efficiently combining two types of meta-data that are particularly popular in contemporary indoor SLAM research: geometric structures and semantic information. We therefore derive such a methodology, based on an RGB-D SLAM framework that internally uses *matchables* as its unified geometric structures. We enhance this system using our semantic framework, which in turn depends on a Deep Neural Network performing semantic segmentation. We show that, when geometry is combined with semantics, the system's performance consistently improves in our test datasets, namely the ICL-NUIM, InteriorNet and TUM RGB-D datasets, and compare with other state-of-the-art methods available in literature for reference.

Keywords: Indoor SLAM, Unified Geometric Representations, Semantic Segmentation

Περίληψη

Ένα από τα κύρια προβλήματα της σύγχρονης ρομποτικής έρευνας είναι η ταυτόχρονη εκτίμηση πόζας και χαρτογράφηση του περιβάλλοντος (Simultaneous Localization And Mapping - SLAM). Αυτή η απαιτητική περιοχή έχει ιδιαίτερη σημασία για την κινητή ρομποτική, όπου ρομποτικοί πράκτορες βασίζονται σε on-board αισθητήρες για την εκτίμηση τόσο της γεωμετρίας του περιβάλλοντος όσο και της τροχιάς τους μέσα σε αυτό. Ξεκινώντας με λύσεις οπτικής οδομετρίας, οι οποίες, μη έχοντας υποδομή για χαρτογράφηση, υποφέρουν από συσσωρευτικά σφάλματα, η σύγχρονη έρευνα έχει τελικά συγκλίνει στην θεωρία του SLAM, που με την χαρτογράφηση και το κλείσιμο βρόχων προσφέρει τη δυνατότητα στα ρομπότ να αυτοδιορθώνουν τις εκτιμήσεις τους όταν επισκέπτονται γνωστούς χώρους. Από μαθηματικής σκοπιάς, υπάρχουν πιθανοτικά εργαλεία που έχουν υποστηρίξει αυτήν την εξέλιξη, δίνοντας βέλτιστες λύσεις υπό τον θόρυβο των αισθητήρων. Σήμερα, με πρόσβαση σε ολοένα και πιο ισχυρό hardware, η κοινότητα χρησιμοποιεί μία ποικιλία από αισθητήρες και αλγορίθμους, συμπεριλαμβανομένων και αλγορίθμων μηχανικής μάθησης, για να αποσπά μεγάλο πλούτο πρόσθετων δεδομένων, που χρησιμοποιούνται για να υποστηρίξουν τις κύριες λειτουργίες SLAM. Αυτό μας κινητοποιεί να δημιουργήσουμε ένα μεικτό πλαίσιο, που θα συνδυάζει αποτελεσματικά δύο τύπους μεταδεδομένων που είναι ιδιαιτέρως δημοφιλείς στη σύγχρονη έρευνα του SLAM εσωτερικών χώρων: γεωμετρικές δομές και σημασιολογική πληροφορία. Επομένως, με βάση μία υπάρχουσα ερευνητική λύση RGB-D SLAM, που εισάγει την έννοια των γεωμετρικών matchable ως τις ενοποιημένες γεωμετρικές του αναπαραστάσεις, χτίζουμε τη μέθοδό μας. Εμπλουτίζουμε θεωρητικά το πλαίσιο με σημασιολογία, που βασίζεται σε ένα βαθύ νευρωνικό δίκτυο που εκτελεί σημασιολογική κατάτμηση των εικόνων. Δείχνουμε ότι, ο συνδυασμός με βάση τη λογική που ορίζουμε, οδηγεί σε καλύτερα αποτελέσματα, όπως αποδεικνύεται και από τα δεδομένα ελέγχου στα οποία πειραματιζόμαστε (ICL-NUIM, InteriorNet, TUM RGB-D), επιτυγχάνοντας εφάμιλλες επιδόσεις με άλλες state-of-the-art μεθόδους που χρησιμοποιούμε για σύγκριση.

Λέξεις-κλειδιά: SLAM Εσωτερικών Χώρων, Ενοποιημένες Γεωμετρικές Αναπαραστάσεις, Σημασιολογική Κατάτμηση

Acknowledgments

First and foremost, I would like to thank my supervisor, Prof. Petros Maragos, for inspiring me as an undergraduate student with his highly instructive lectures, and for his support and ideas throughout my thesis. In addition, I am deeply grateful to Georgia Chalvatzaki, who tirelessly stood by me from the very beginning of the thesis, mentoring and guiding me at every step, and to Panagiotis Mermigas, for the constructive discussions we had and his meticulous reviews of the work, in both mathematics and phrasing. In addition, I am indebted to Irvin Aloise of the Sapienza University of Rome, for his illuminating insights at the beginning of this thesis. Many thanks also to Panagiotis Filntisis and Christos Garoufis, for their crucial technical support. Finally, I would like to thank all my friends and fellow students, for making these past few years such an unforgettable experience.

I dedicate this work to my family, and especially my parents and brother, who always believed in me and have been by my side throughout my entire life.

Contents

List of Figures

List of Tables

List of Algorithms

Επεκτεταμένη Περίληψη

1. Εισαγωγή

Η ρομποτική όραση είναι ο τομέας της ρομποτικής που ασχολείται με εφαρμογές αισθητήρων και αλγορίθμων που επιτρέπουν στα ρομπότ να αντιλαμβάνονται οπτικά το περιβάλλον τους. Επομένως, ένα από τα βασικά προβλήματα στον τομέα είναι αυτό της ταυτόχρονης εκτίμησης πόζας και χαρτογράφησης περιβάλλοντος (*Simultaneous Localization And Mapping - SLAM*). Πρόκειται για μία ενεργή περιοχή ερευνητικού ενδιαφέροντος, με ιδιαίτερη αξία για την κινητή ρομποτική, όπου πράκτορες επαφίενται σε ενσωματωμένους αισθητήρες τόσο για την ακριβή χαρτογράφηση του περιβάλλοντός τους όσο και για την εύρωστη εκτίμηση της πόζας τους. Τα συστήματα SLAM έχουν εξελιχθεί έχοντας κατά νου βασικές έννοιες όπως η ανάπτυξη και μοντελοποίηση διαφόρων αισθητήρων για συλλογή μετρήσεων, ο ορισμός δομών δεδομένων για την αποθήκευση και χρήση χαρτών, και η θεωρία για την ανίχνευση και εκμετάλλευση βρόχων στις τροχιές των πρακτόρων [15, 16].

Τα διάφορα συστήματα που έχουν προταθεί στην πολυετή ιστορία του SLAM έχουν μεγάλη ποικιλομορφία. Ξεκινώντας από συστήματα *οδομετρίας*, τα οποία προσέφεραν μόνο εκτίμηση πόζας, η ερευνητική κοινότητα εξερεύνησε διάφορες μεθόδους βελτιστοποίησης, καταλήγοντας τελικά σε πιο ολοκληρωμένους μαθηματικούς φορμαλισμούς του προβλήματος, π.χ. ως πρόβλημα βελτιστοποίησης μη-γραμμικών τετραγώνων. Η συνένωση εκτίμησης πόζας και χαρτογράφησης οδήγησε σε πιο φιλόδοξους στόχους και σε ακόμα μεγαλύτερες δυνατότητες επέκτασης των υπαρχόντων θεωρητικών πλαισίων, με την εισαγωγή διαφορετικών τύπων χαρτογράφησης και εκτενέστερη χρήση σημασιολογίας.

Σε αυτό το πλαίσιο, σκοπεύουμε να συνεισφέρουμε μία μέθοδο φτιαγμένη για λειτουργία σε εσωτερικούς χώρους. Για αυτό και θέλουμε να εκμεταλλευτούμε τα ειδικά χαρακτηριστικά αυτών των περιβαλλόντων. Σε αντιδιαστολή με τους εξωτερικούς χώρους, στους εσωτερικούς μπορούμε να περιμένουμε με μεγαλύτερη σιγουριά κανονικές γεωμετρικές δομές, όπως επίπεδα και ευθείες. Επίσης, υπάρχει μεγάλη ποικιλία αντικειμένων και επίπλων, τα οποία έχουν σημασιολογική πληροφορία που μπορεί να ανιχνευθεί και να χρησιμοποιηθεί. Αυτά που χρειαζόμαστε λοιπόν είναι:

- να επιλέξουμε ένα γεωμετρικό μοντέλο για τα αντικείμενα του χώρου [17, 6, 7],
- να επιλέξουμε ένα τρόπο συλλογής σημασιολογικής πληροφορίας [18, 19, 9] και
- να θεμελιώσουμε και υλοποιήσουμε μία μέθοδο για το συνδυασμό γεωμετρίας και σημασιολογίας.

2. Σχετική Έρευνα

Υπάρχει μεγάλος πλούτος ερευνητικών προσπαθειών στον τομέα του SLAM. Όμως, είναι φυσικό να αναρωτηθεί κανείς αν μία λύση αυτόνομης κίνησης ενός ρομποτικού πράκτορα απαιτεί χαρτογράφηση. Για αυτό και θα ξεκινήσουμε μιλώντας για τις μεθοδολογίες της *οδομετρίας*, πριν προχωρήσουμε σε βασικές μεθόδους πιθανοτικού SLAM και ύστερα σε σύγχρονες μεθόδους με γράφους πόζας.

2.1 Βασικοί ορισμοί και συμβολισμοί

Πριν προχωρήσουμε στο περιεχόμενο, θα ξεκαθαρίσουμε κάποιες βασικές έννοιες και σύμβολα που θα φανούν πολύ χρήσιμα σε όλη την έκταση της εργασίας.

Χαρακτηριστικό (feature) του χώρου θα ονομάζεται οποιοδήποτε διακριτικό και αναγνωρίσιμο οπτικό ή γεωμετρικό στοιχείο ανιχνεύει ένα σύστημα. Για οπτικά χαρακτηριστικά έχει αναπτυχθεί πλούσια βιβλιογραφία [20, 21, 22, 23, 24].

Ως **μετασχηματισμός (transform)** $\boldsymbol{T}_B^A$ από το πλαίσιο συντεταγμένων A στο πλαίσιο συντεταγμένων B ορίζουμε τον 4×4 πραγματικό πίνακα με τις ακόλουθες ιδιότητες:

$$\boldsymbol{T}_B^A = \left[\begin{array}{c|c} \boldsymbol{R}_B^A & \boldsymbol{d}_B^A \\ \hline \boldsymbol{0}_{1\times 3} & 1 \end{array}\right], \ \boldsymbol{p^A} = \boldsymbol{T}_B^A \cdot \boldsymbol{p}^B, \ \boldsymbol{p} = \left[\begin{array}{ccc|c} p_x & p_y & p_z & 1 \end{array}\right]^T$$

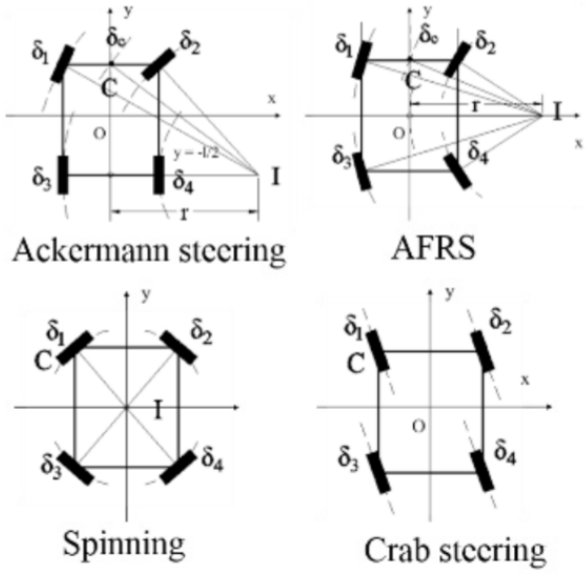
όπου R_B^A, d_B^A ο πίνακας στροφής και το διάνυσμα μετατόπισης ανάμεσα στα δύο πλαίσια αντίστοιχα και $\boldsymbol{p}$ οι επεκτεταμένες συντεταγμένες ενός σημείο, εκπεφρασμένες στο εκάστοτε πλαίσιο. Για πολλαπλά πλαίσια με γνωστούς μετασχηματισμούς σημειώνουμε ότι ισχύει: $\boldsymbol{T}_C^A = \boldsymbol{T}_B^A \cdot \boldsymbol{T}_C^B$.

2.2 Οδομετρία

Ο όρος οδομετρία [25, 26] αναφέρεται σε οποιαδήποτε προσπάθεια γίνεται από έναν ρομποτικό πράκτορα να εκτιμήσει την πόζα του σε σχέση με την απαρχή της κίνησής του. Αυτό επιτυγχάνεται με την εξής διαδικασία: αρχικοποιούμε την πόζα του πράκτορα (π.χ. στον ταυτοτικό μετασχηματισμό), συλλέγουμε διαδοχικά στοιχεία από τους αισθητήρες του πράκτορα και στη συνέχεια τα χρησιμοποιούμε για να εκτιμήσουμε διαδοχικούς μετασχηματισμούς πόζας. Επομένως, η τρέχουσα εκτίμηση της πόζας του πράκτορα είναι απλά το γινόμενο των επιμέρους μετασχηματισμών. Σημειώνουμε ότι οι αισθητήρες μας δίνουν δεδομένα σε κάποια συχνότητα που εξαρτάται από τον τύπο τους και ότι υπάρχουν πολλοί διαφορετικοί τύποι αισθητήρων.

Μία βασική μορφή οδομετρίας μπορεί να επιτευχθεί με την χρήση κωδικοποιητών στους τροχούς ενός ρομποτικού οχήματος. Μετρώντας δηλαδή τις περιστροφές των τροχών, μπορούμε να εκτιμήσουμε την μετατόπιση του ρομπότ με χρήση ενός μοντέλου για την κίνησή του. Συνιστώσες αυτού του μοντέλου μπορούν θα είναι η ακτίνα τροχών και ο τρόπος στροφής του. Υπάρχουν διάφοροι τρόποι στροφής όπως φαίνεται στο Σχ. 1, και οι αντίστοιχες

εξισώσεις για καθέναν από αυτούς δίνουν το τελικό μοντέλο κίνησης για την οδομετρία τροχών. Μία πιο γενική προσέγγιση στην οδομετρία χρησιμοποιεί αισθητήρες ακτινοβολίας, όπως LiDAR ή κάμερες. Η δύναμη αυτής της γενίκευσης φαίνεται στο ότι δεν υπάρχει πλέον περιορισμός στο τι είδους ρομπότ μπορεί να χρησιμοποιηθεί, φτάνει να υπάρχουν κάποια χαρακτηριστικά στο περιβάλλον που να ανιχνεύονται αξιόπιστα από το ρομπότ για να μπορεί να κάνει εκτίμηση της κίνησής του με κάποιον αλγόριθμο.



Σχήμα 1: Οπτικοποίηση διαφόρων τρόπων στροφής ενός τετράτροχου ρομπότ. [1]

Στις παραπάνω ιδέες παρατηρούμε ότι λείπει μία καίρια λεπτομέρεια: δεν υπάρχει δυνατότητα αυτοδιόρθωσης τυχαίων λαθών στην εκτίμηση (π.χ. λόγω θορύβου). Αυτή η βασική ιδιότητα των συστημάτων οδομετρίας είναι ένα από τα μεγαλύτερα μειονεκτήματά τους. Συγκεκριμένα, καθώς το σύστημα εκτιμά ολοένα και μεγαλύτερες 'αλυσίδες' από μετασχηματισμούς, πολλαπλασιάζοντάς τες για να πάρει την τρέχουσα πόζα, μικρά σφάλματα που συμβαίνουν σε κάθε εκτίμηση συσσωρεύονται, με αποτέλεσμα εν τέλει να υπάρχουν σοβαρές αποκλίσεις σε μεγάλες τροχιές. Φυσικά, υπάρχουν μέτρα κατά των αποκλίσεων αυτών, όπως τα ακόλουθα:

- προσαρμογή συστάδων (*bundle adjustment* - *BA*), με την οποία το σύστημα εκτελεί μία τοπική βελτιστοποίηση σε ένα κυλιόμενο παράθυρο από δεδομένα των αισθητήρων, εξομαλύνοντας τοπικά ως ένα βαθμό το θόρυβο [27]

- σύνθεση αισθητήρων (*sensor fusion*), όπου το σύστημα συνδυάζει τα δεδομένα από πολλαπλούς αισθητήρες για να διορθώσει λάθη που γίνονται από τον καθέναν ξεχωριστά

- σημασιολογικές προεκτάσεις, όπου με χρήση κάποιου μοντέλου ανίχνευσης σημασιολογικών οντοτήτων στο χώρο κερδίζουμε επιπρόσθετη πληροφορία, που βοηθάει στο

να διορθωθούν οι συσχετίσεις των βασικών δεδομένων των αισθητήρων [28, 29]

Τα παραπάνω αποτελούν *τοπικές* βελτιώσεις, σε ένα συστημικό πρόβλημα το οποίο δεν μπορούμε να ελπίζουμε ότι θα επιλυθεί πλήρως έτσι. Σε περίπτωση που η τροχιά του πράκτορα δεν ξαναπερνάει ποτέ από προηγούμενα σημεία, μία λύση οδομετρίας που χρησιμοποιεί τα παραπάνω εργαλεία βρίσκεται μεθοδολογικά στην καλύτερη έκδοση που θα μπορούσε να είναι. Αυτό που θέλουμε να πετύχουμε είναι το σύστημα να μπορεί να αναγνωρίζει τοποθεσίες που έχει ήδη επισκεφθεί και αυτόματα να διορθώνει τις εκτιμήσεις που έχει κάνει μέχρι εκείνο το σημείο. Αυτή η ικανότητα είναι θεμελιωδώς διαφορετική από τις παραπάνω, καθώς απαιτεί μία σειρά από επιμέρους βήματα, τα οποία πρακτικά μας οδηγούν στο SLAM.

2.3 Βασικά συστήματα SLAM

Ξεκινάμε με τον φορμαλισμό των πρώτων συστημάτων SLAM της βιβλιογραφίας, τα οποία ήταν στην βάση τους Bayes συστήματα εκτίμησης πιθανοτήτων. Τέτοια συστήματα υπέθεταν απλοποιημένους κόσμους για τα ρομποτικά τους μοντέλα, με γνωστά μοντέλα θορύβου για τους αισθητήρες και διάφορες άλλες υποστηρικτικές υποθέσεις, όπως η Μαρκοβιανή υπόθεση για τις διαδοχικές καταστάσεις του ρομπότ, με σκοπό η μαθηματική διαδικασία που προέκυπτε να μπορούσε να υπολογιστεί [2]. Η τελική διατύπωση του Bayesian φιλτραρίσματος διαχειρίζεται στην πραγματικότητα την συγγενική με τις πιθανότητες έννοια της *κατανομής πίστης*.

$$\mathrm{bel}(\boldsymbol{x}_t) := \mathbb{P}[\boldsymbol{x}_t|\boldsymbol{z}_{1:t}, \boldsymbol{u}_{1:t}],\ \overline{\mathrm{bel}}(\boldsymbol{x}_t) := \mathbb{P}[\boldsymbol{x}_t|\boldsymbol{z}_{1:t-1}, \boldsymbol{u}_{1:t}] \tag{1}$$

Με βάση τους παραπάνω ορισμούς προκύπτει και ο αντίστοιχος αλγόριθμος:

Algorithm 1: Bayesian φιλτράρισμα [2]

Input: $\mathrm{bel}(\boldsymbol{x}_{t-1}, \boldsymbol{u}_{1:t}, \boldsymbol{z}_{1:t})$
Output: $\mathrm{bel}(\boldsymbol{x}_t)$

forall $\boldsymbol{x}_t$ **do**
  $\overline{\mathrm{bel}}(\boldsymbol{x}_t) = \int \mathbb{P}[\boldsymbol{x}_t|\boldsymbol{z}_{1:t-1}, \boldsymbol{u}_{1:t}]\mathrm{bel}(\boldsymbol{x}_{t-1})dx_{t-1}$
  $\mathrm{bel}(\boldsymbol{x}_t) = \eta \cdot \mathbb{P}[\boldsymbol{z}_t|\boldsymbol{x}_t]\overline{\mathrm{bel}}(\boldsymbol{x}_t)$
end
return $\mathrm{bel}(\boldsymbol{x}_t)$

Όπως φαίνεται στον Αλγ. 1, η διατύπωσή του είναι αρκετά γενική, καθώς δεν γίνεται καμία υπόθεση για τις κατανομές των εμπλεκόμενων μεταβλητών. Οι κατανομές αυτές μπορούν να προσεγγιστούν με διάφορους τρόπους, όπως για παράδειγμα με την Γκαουσσιανή προσέγγιση. Αυτή η προσέγγιση, μαζί με τις μαθηματικές τροποποιήσεις που επιφέρει, δίνει εν τέλει τα φίλτρα Kalman. Η βασική διαδικασία για τα απλά φίλτρα Kalman περιγράφεται στον Αλγ. 2:

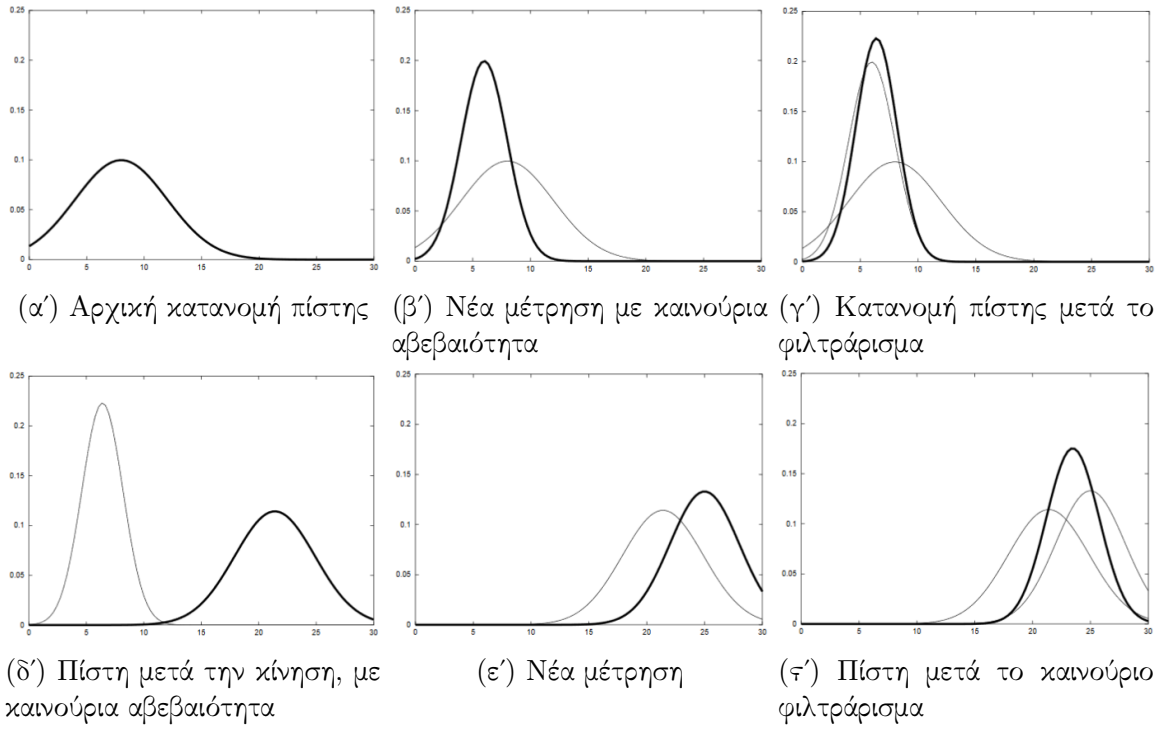
Algorithm 2: Φίλτρο Kalman [2]

Input: $\boldsymbol{\mu}_{t-1}, \boldsymbol{\Sigma}_{t-1}, \boldsymbol{u}_t, \boldsymbol{z}_t$
Output: $\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t$

$\overline{\boldsymbol{\mu}}_t = \boldsymbol{A}_t\boldsymbol{\mu}_t + \boldsymbol{B}_t\boldsymbol{u}_t$
$\overline{\boldsymbol{\Sigma}}_t = \boldsymbol{A}_t\boldsymbol{\Sigma}_{t-1}\boldsymbol{A}_t^T + \boldsymbol{R}_t$

$\boldsymbol{K}_t = \overline{\boldsymbol{\Sigma}}_t\boldsymbol{C}_t(\boldsymbol{C}_t\overline{\boldsymbol{\Sigma}}_t\boldsymbol{C}_t^T + \boldsymbol{Q}_t)^{-1}$
$\boldsymbol{\mu}_t = \overline{\boldsymbol{\mu}}_t + \boldsymbol{K}_t(\boldsymbol{z}_t - \boldsymbol{C}_t\overline{\boldsymbol{\mu}}_t)$
$\boldsymbol{\Sigma}_t = (\mathbb{I} - \boldsymbol{K}_t\boldsymbol{C}_t)\overline{\boldsymbol{\Sigma}}_t$
return $\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t$

Οι πίνακες $\boldsymbol{A}_t, \boldsymbol{B}_t, \boldsymbol{C}_t$ περιγράφουν τις γραμμικές μεταβολές κατάστασης του μοντέλου του συστήματος και οι πίνακες $\boldsymbol{Q}_t, \boldsymbol{R}_t$ είναι Γκαουσσιανοί πίνακες διασπορών θορύβου. Ο πίνακας $\boldsymbol{K}_t$ λέγεται και *κέρδος Kalman*. Στο Σχ. 2 φαίνεται το πώς η διαδικασία επηρεάζει την εκτίμηση κατάστασης για μία μεταβλητή.



(α′) Αρχική κατανομή πίστης (β′) Νέα μέτρηση με καινούρια αβεβαιότητα (γ′) Κατανομή πίστης μετά το φιλτράρισμα

(δ′) Πίστη μετά την κίνηση, με καινούρια αβεβαιότητα (ε′) Νέα μέτρηση (ϛ′) Πίστη μετά το καινούριο φιλτράρισμα

Σχήμα 2: Ενδιάμεσα βήματα του φίλτρου Kalman για εκτίμηση μίας μεταβλητής. [2]

Προεκτάσεις των παραπάνω ιδεών έχουν προταθεί, όπως είναι το επεκτεταμένο φίλτρο Kalman (*Extended Kalman Filter - EKF*) και το μη-καθοδηγούμενο φίλτρο Kalman (*Unscented Kalman Filter - UKF*). Η προσπάθειες αυτές έχουν επικεντρωθεί στο να γίνονται

καλές γραμμικοποιήσεις των μοντέλων σε κατάλληλα σημεία και στην μοντελοποίηση με δειγματοληψία αντίστοιχα. Ο σκοπός και των δύο είναι το πιθανοτικό μοντέλο να προσεγγίζει καλύτερα την πραγματικότητα, χωρίς όμως να εγκαταλείπει εντελώς τις ευνοϊκές ιδιότητες των Γκαουσσιανών κατανομών.

Υπάρχει όμως και μία κατηγορία συστημάτων SLAM ξέφυγε εντελώς από το Γκαουσσιανό μοντέλο, εισάγοντας την έννοια των μη-παραμετρικών συστημάτων SLAM. Τα συστήματα αυτά αντιμετώπισαν με καινούριες ιδέες το πρόβλημα των πολλαπλών μεγίστων στις κατανομές πίστης. Το πρόβλημα αυτό είναι υπαρκτό σε καταστάσεις όπου υπάρχει αμφισημία από τους αισθητήρες του ρομπότ για το πού ακριβώς βρίσκεται ο πράκτορας, δηλαδή όταν δύο διαφορετικές τοποθεσίες δίνουν πρακτικά ίδια εικόνα στην είσοδο του συστήματος. Τέτοιες καταστάσεις αφαιρούν πολλές Γκαουσσιανές ιδιότητες από την διατύπωση των κατανομών που προσπαθούμε να προσεγγίσουμε. Τα μη-παραμετρικά φίλτρα, όπως φίλτρο ιστογραμμάτων [30] και το φίλτρο σωματιδίων [5, 31, 32], δειγματοληπτούν με βάση τους εσωτερικού τους αλγόριθμους με τέτοιο τρόπο ώστε να προσεγγίσουν τις ζητούμενες κατανομές, χωρίς να χρειαστεί να βελτιστοποίησουν ρητά κάποια μαθηματική έκφραση για αυτές.

2.4 SLAM με γράφους πόζας

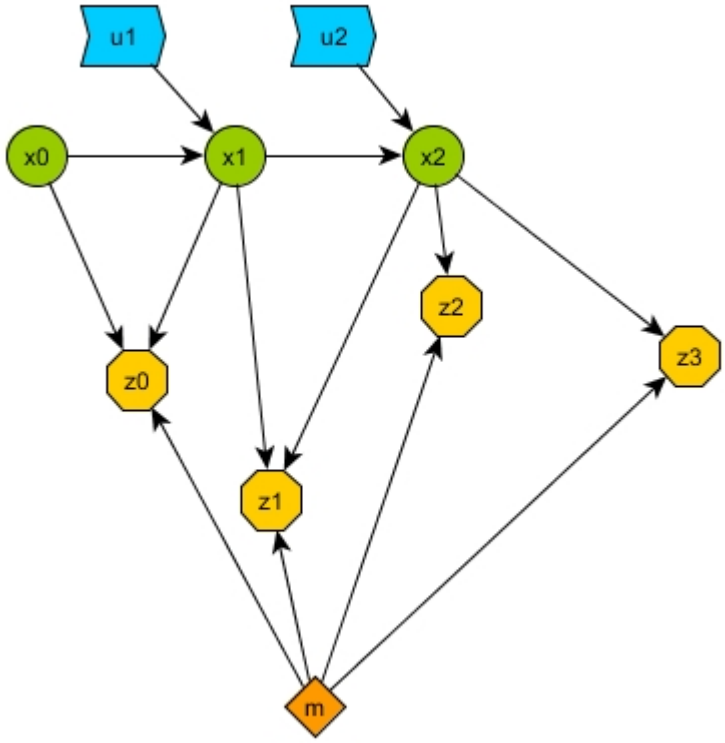
Τα συστήματα SLAM τα οποία έχουμε δει έως τώρα έχουν κάποιες ελλείψεις στην αναπαράσταση των χαρτών τους, καθώς δεν δίνουν ένα ενοποιημένο πλαίσιο για πόζες του πράκτορα και ορόσημα του περιβάλλοντος. Η βελτιστοποίηση λοιπόν είναι δέσμια της υλοποίησης και οι μέθοδοι δεν γενικεύονται, ενώ τα loop closures δεν έχουν κάποια γενική, διαισθητική περιγραφή. Αυτό το κενό έρχονται να καλύψουν οι μέθοδοι γράφων πόζας [3, 33]. Ο γράφος πόζας δεν είναι τίποτε άλλο παρά μία αφαιρετική αναπαράσταση του κόσμου του πράκτορα, που προσφέρει αρκετή ευελιξία και γενικότητα ώστε το πρόβλημα του SLAM να αναχθεί τελικά σε ένα πρόβλημα βελτιστοποίησης μη-γραμμικών ελαχίστων τετραγώνων.

Στο Σχ. 3 βλέπουμε μία γραφική αναπαράσταση για το πώς δομείται ένας γράφος πόζας. Με βάση αυτή τη διατύπωση, η ποσότητα που καλούμαστε να ελαχιστοποιήσουμε μπορεί να περιγραφεί ως το συνολικό τετραγωνικό σφάλμα πάνω σε ολόκληρο το διάνυσμα ποζών $\boldsymbol{X}$, όπου κάθε όρος προκύπτει ως η νόρμα Mahalanobis του σφάλματος e_{ij} της κάθε συνιστώσας ακμής $(i, j) \in \mathcal{C}$, σταθμισμένο με τον πίνακα πληροφορίας της μέτρησης $\boldsymbol{\Omega}_{ij}$. Συγκεκριμένα:

$$\boldsymbol{X}^* = \arg\min_{\boldsymbol{X}} \sum_{(i,j)\in\mathcal{C}} f_{ij} = \arg\min_{\boldsymbol{X}} \sum_{(i,j)\in\mathcal{C}} \boldsymbol{e}_{ij}^T \boldsymbol{\Omega}_{ij} \boldsymbol{e}_{ij} \tag{2}$$

όπου τα σφάλματα που ελαχιστοποιούμε προκύπτουν όπως στο Σχ. 4. Στην ουσία, η απόκλιση ανάμεσα στις παρατηρήσεις και το τι περιμέναμε να δούμε με βάση κάποια αρχική εκτίμηση ή μοντέλο κίνησης ποσοτικοποιείται και ελαχιστοποιείται από κοινού για την περίπτωση σφαλμάτων στην εκτίμηση πόζας και θορύβου παρατήρησης.

Η δύναμη της αναπαράστασης με γράφους πόζας δίνει ένα σχεδιαστικό προτέρημα, την αποσύμπλεξη front-end και back-end, που με τη σειρά της επιτρέπει την εύκολη και διαισθητική περιγραφή των κλεισιμάτων βρόχων. Συγκεκριμένα, από τα δύο αυτά υποσύνολα



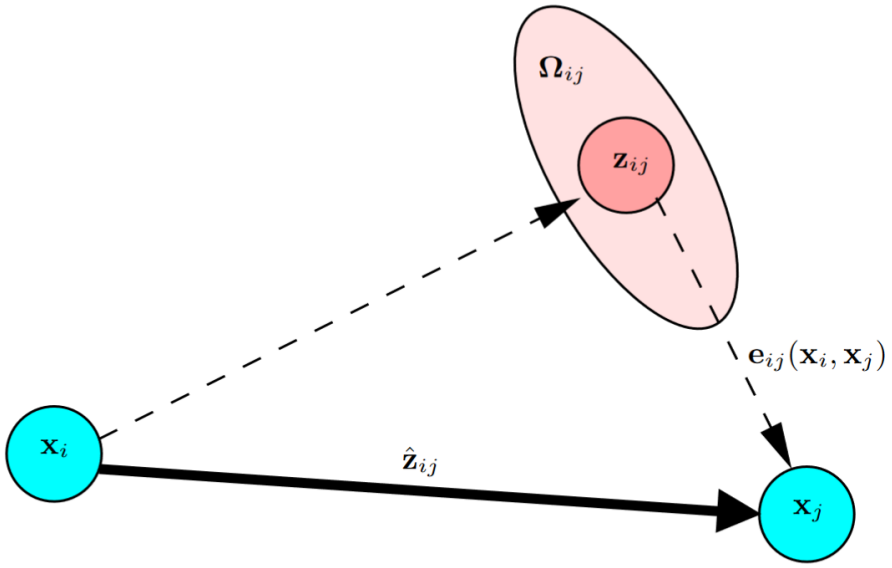
Σχήμα 3: Οπτικοποίηση δομής ενός γράφου πόζας: έλεγχοι u_i οδηγούν σε μεταβάσεις ανάμεσα στις πόζες x_i, από τις οποίες ο χάρτης m παρατηρείται με μετρήσεις z_i.

του συστήματος, το front-end είναι υπεύθυνο για την δημιουργία, ανανέωση και συντήρηση του χάρτη και του γράφου πόζας, ενώ το back-end είναι υπεύθυνο για την βελτιστοποίηση του γράφου αυτού. Η ανίχνευση κλεισιμάτων βρόχων εμπίπτει στις διαδικασίες του front-end, όπου αν ανιχνευθεί τέτοιο συμβάν προστίθενται οι απαραίτητες ακμές περιορισμών στον γράφο, οι οποίες θα επηρεάσουν έμμεσα την βελτιστοποίηση που θα γίνει στο back-end.

Η βελτιστοποίηση αυτή καθαυτή γίνεται με βάση το σφάλμα, το οποίο γραμμικοποιείται με χρήση Ιακωβιανών ανάμεσα στους εκάστοτε κόμβους του γράφου:

$$\boldsymbol{e}_{ij}(\breve{\boldsymbol{x}} + \Delta\boldsymbol{x}) \simeq \boldsymbol{e}_{ij} + \boldsymbol{J}_{ij}\Delta\boldsymbol{x} \tag{3}$$

όπου $\boldsymbol{J}_{ij}$ είναι η Ιακωβιανή της συνάρτησης σφάλματος $\boldsymbol{e}_{ij}$ γύρω από μία αρχική εκτίμηση κατάστασης $\breve{\boldsymbol{x}}$, την οποία έχουμε αλλάξει κατά μία τοπική μεταβολή Δx. Επομένως, με χρήση της παραπάνω διατύπωσης μπορούμε να γράψουμε τους επιμέρους όρους της βελτιστοποίησης



Σχήμα 4: Μία αβέβαιη μέτρηση. [3]

από την σχέση (2):

$$\begin{aligned} f_{ij}(\breve{\boldsymbol{x}} + \Delta\boldsymbol{x}) &\simeq (\boldsymbol{e}_{ij} + \boldsymbol{J}_{ij}\Delta\boldsymbol{x})^T \mathbf{\Omega}_{ij} (\boldsymbol{e}_{ij} + \boldsymbol{J}_{ij}\Delta\boldsymbol{x}) \\ &= \underbrace{\boldsymbol{e}_{ij}^T \mathbf{\Omega}_{ij} \boldsymbol{e}_{ij}}_{c_{ij}} + 2\underbrace{\boldsymbol{e}_{ij}^T \mathbf{\Omega}_{ij} \boldsymbol{J}_{ij}}_{\boldsymbol{b}_{ij}} \Delta\boldsymbol{x} + \Delta\boldsymbol{x}^T \underbrace{\boldsymbol{J}_{ij}^T \mathbf{\Omega}_{ij} \boldsymbol{J}_{ij}}_{\boldsymbol{H}_{ij}} \Delta\boldsymbol{x} \\ &= c_{ij} + 2\boldsymbol{b}_{ij}\Delta\boldsymbol{x} + \Delta\boldsymbol{x}^T \boldsymbol{H}_{ij} \Delta\boldsymbol{x} \end{aligned} \tag{4}$$

Αθροίζοντας τους παραπάνω όρους παίρνουμε:

$$\begin{aligned} f(\breve{\boldsymbol{x}} + \Delta\boldsymbol{x}) = \sum_{(i,j)\in\mathcal{C}} f_{ij}(\breve{\boldsymbol{x}} + \Delta\boldsymbol{x}) &= \sum_{(i,j)\in\mathcal{C}} c_{ij} + 2\boldsymbol{b}_{ij}\Delta\boldsymbol{x} + \Delta\boldsymbol{x}^T \boldsymbol{H}_{ij} \Delta\boldsymbol{x} \\ &= c + 2\boldsymbol{b}\Delta\boldsymbol{x} + \Delta\boldsymbol{x}^T \boldsymbol{H} \Delta\boldsymbol{x} \end{aligned} \tag{5}$$

από όπου η βέλτιστη λύση μπορεί να εκτιμηθεί επιλύοντας το σύστημα: $\boldsymbol{H}\Delta\boldsymbol{x}^* = -\boldsymbol{b}$.

Τα παραπάνω θεωρητικά βήματα αγνοούν ένα πολύ σημαντικό χαρακτηριστικό του μαθηματικού αντικειμένου υπό βελτιστοποίηση: ανήκει σε έναν εγγενώς μη-ευκλείδιο χώρο καταστάσεων. Η αξία αυτής της παρατήρησης είναι που οδηγεί στα πολύπτυχα (*manifolds*), μαθηματικούς χώρους που είναι τοπικά γραμμικοί αλλά ολικά μη-γραμμικοί. Η θεωρία που χρησιμοποιείται σε αυτές τις δομές είναι οι ομάδες Lie και οι άλγεβρες Lie. Οι ομάδες Lie αντιστοιχούν σε παραγωγίσιμα πολύπτυχα, ενώ οι άλγεβρες Lie αντιστοιχούν στις γραμμικοποιήσεις που γίνονται για να δώσουν τους εφαπτόμενους χώρους (*tangent spaces*) γύρω από ένα σημείο του πολύπτυχου. Οι εκθετικοί και λογαριθμικοί χάρτες επιτρέπουν την μετάβαση

Δεδομένα	**Τύπος**	**Παράμ.**	**Πόζα**	**Κίνηση**	**Σημ.**	**Όγκος**
ICL-NUIM [11]	συνθετικό	ναι	ναι	επαρκής	όχι	περιορ.
InteriorNet [34]	συνθετικό	ναι	ναι	καλή	ΣΦΚ	τεράστιος
TUM RGB-D [35]	πραγματικό	ναι	ναι	καλή	όχι	επαρκής
CoRBS [36]	πραγματικό	ναι	ναι	περιορ.	όχι	περιορ.
AVD [37]	πραγματικό	όχι	περιορ.	ανεπαρκής	ΣΦΚ	μεγάλος
MS 7-scenes [38, 39]	συνθετικό	όχι	ναι	καλή	όχι	περιορ.
NYU v2 [40]	πραγματικό	ναι	όχι	καλή	πυκνή	μεγάλο
RGB-D Scenes v2 [41]	πραγματικό	όχι	ναι	επαρκής	περιορ.	περιορ.
Coll. SLAM DS [42]	πραγματικό	περιορ.	ναι	καλή	όχι	περιορ.

Πίνακας 1: Διάφορα σύνολα δεδομένων και οι ιδιότητές τους. Ακολουθίες από τα σύνολα με **έντονη** γραφή χρησιμοποιήθηκαν στα πειράματά μας.

από το ένα στο άλλο, και ο τελεστής $\boxplus$ αντικαθιστά την απλή διανυσματική πρόσθεση για να δώσει τους καινούργιους τύπους.

2.5 Δεδομένα ελέγχου και υπάρχοντα συστήματα SLAM

Από την πολυσχιδή βιβλιογραφία στην περιοχή του SLAM, παρουσιάζουμε εν συντομία κάποιες συλλογές δεδομένων, καθώς και διάφορα συστήματα που συναντήσαμε ή χρησιμοποιήσαμε στην πορεία της εργασίας. Οι προδιαγραφές μας για τα δεδομένα ήταν να είναι σε μορφή $RGB-D$, με διαθέσιμες τις παραμέτρους της κάμερας (*calibration parameters*) και *ground-truth (GT)* δεδομένα πόζας. Επίσης ήταν επιθυμητό να υπάρχει ποικιλία στις κινήσεις που εκτελεί η κάμερα στις εκάστοτε ακολουθίες, ενώ αφήσαμε ευελιξία στο κατά πόσο τα δεδομένα ήταν φωτορεαλιστικά-συνθετικά ή αποτελέσματα από πραγματικούς αισθητήρες. Τα αποτελέσματά μας στον Πίν. 1 δείχνουν το τι σχετικό βρέθηκε στην βιβλιογραφία.

Πέρα από τα δεδομένα ελέγχου αξίζει προφανώς να αναφερθούμε και σε άλλες μεθόδους SLAM που υπάρχουν στη βιβλιογραφία. Τέτοιες μέθοδοι υπάρχουν πάρα πολλές και, σε αντίθεση με τα δεδομένα ελέγχου, επιδεικνύουν ριζικές μεθοδολογικές αποκλίσεις, τόσο στον σχεδιασμό τους όσο και στους αισθητήρες και τις εσωτερικές αναπαραστάσεις που χρησιμοποιούν. Επομένως θα τα κατατάξουμε αντίστοιχα με βάση αυτά τα χαρακτηριστικά τους: Παραπάνω βλέπουμε λοιπόν την μεγάλη ποικιλομορφία των λύσεων που υπάρχουν στη βιβλιογραφία, από λύσεις για LiDAR μέχρι πυκνές οπτικές λύσεις με σημασιολογική υποστήριξη, ένα φοβερά ευρύ φάσμα από προσεγγίσεις που έχουν προταθεί κατά καιρούς δίνει κατευθύνσεις για έρευνα. Στους εσωτερικούς χώρους, δουλειές με σημασιολογία έχουν εξελιχθεί εν πολλοίς ανεξάρτητα από προηγμένες γεωμετρικές προσεγγίσεις. Αυτό είναι το κενό που θέλουμε να εξερευνήσουμε με την δική μας πρόταση. Με τον Πίν. 2 ολοκληρώνεται η επισκόπηση της βιβλιογραφίας και μπορούμε να περάσουμε σε πιο συγκεκριμένα θέματα που αφορούν την συνεισφορά μας.

Σύστημα	**Αισθητήρες**	**Διαστ.**	**Περιβ.**	**Χάρτης**
Hector SLAM [43]	LiDAR	2D	όλα	πλέγμα κατ.
Cartographer [44]	LiDAR	2D & 3D	εσωτερικό	πλέγμα κατ.
LOAM [45]	LiDAR	3D	εσωτερικό	πλέγμα κατ.
OpenVSLAM [46]	mono/stereo/RGB-D	3D	όλα	αραιός
ProSLAM [47]	stereo	3D	εξωτερικό	αραιός
LSD-SLAM [48, 49]	mono/stereo	3D	όλα	ημίπυκνος
ElasticFusion [50]	RGB-D	3D	εσωτερικό	πυκνός
Bowman *et al.** [9]	stereo & IMU	3D	εσωτερικό	αραιός
Doherty *et al.** [51]	stereo & IMU	3D	εξωτερικό	αραιός
VSO* [28]	stereo	3D	εξωτερικό	όχι
SemVO* [29]	RGB-D	3D	εσωτερικό	όχι
SemanticFusion* [18]	RGB-D	3D	εσωτερικό	πυκνός
Li *et al.** [19]	stereo	3D	εξωτερικό	αραιός
ORB-SLAM2 [12]	mono/stereo/RGB-D	3D	όλα	αραιός
MaskFusion* [13]	RGB-D	3D	εσωτερικό	πυκνός
SASHAGO [7]	RGB-D	3D	εσωτερικό	αραιός

Πίνακας 2: Διάφορα συστήματ SLAM και οι ιδιότητές τους. Συστήματα με **έντονη** γραφή χρησιμοποιήθηκαν στην παρούσα εργασία. Ο αστερίσκος συμβολίζει σημασιολογικές μεθόδους.

3. Ενοποιημένες Γεωμετρικές Αναπαραστάσεις

Ένα βασικό χαρακτηριστικό των εσωτερικών χώρων είναι η γεωμετρική κανονικότητά τους. Συγκεκριμένα έχουμε χώρους που περιέχουν ευθείες γραμμές, επίπεδα και σημεία· τύπους γεωμετριών που θα θέλαμε να περιγράψουμε με ένα κοινό μαθηματικό πλαίσιο. Η ανάγκη αυτή προκύπτει με το εξής σκεπτικό: τα περισσότερα αραιά συστήματα οπτικού SLAM αναπαριστούν στην πραγματικότητα μεμονωμένα σημεία του περιβάλλοντός τους μέσα στους χάρτες που δημιουργούν. Αυτά τα σημεία είναι που δίνουν τους περιορισμούς που είδαμε στην προηγούμενη ενότητα και πάνω σε αυτά χτίζεται ολόκληρη η αντίληψη του πράκτορα για τον γύρω κόσμο. Αν μπορούσαμε όμως να εκμεταλλευτούμε ευθείες και επίπεδα, ορίζοντας κατάλληλα τις συναρτήσεις σφάλματος για αυτά τα νέα γεωμετρικά αντικείμενα, τότε θα κερδίζαμε σημαντικά ποσά εκφραστικότητας σε χώρους όπου τέτοιες γεωμετρίες εμφανίζονται συχνά, όπως ακριβώς είναι οι εσωτερικοί χώροι.

3.1 Μαθηματικές έννοιες και ορισμοί

Ξεκινάμε αυτήν την περιγραφή με τον ορισμό των τετραγωνικών υπερεπιφανειών (*quadric hypersurfaces*). Πρόκειται για σύνολα σημείων $\boldsymbol{x} \in \mathbb{R}^D$ που πληρούν την ακόλουθη εξίσωση:

$$\boldsymbol{x}^T \boldsymbol{Q} \boldsymbol{x} + \boldsymbol{p} \boldsymbol{x} + r = \boldsymbol{0} \tag{6}$$

όπου $\boldsymbol{Q}$ είναι ένας $D \times D$ πίνακας, $\boldsymbol{p}$ ένα D-διάστατο διάνυσμα-γραμμή και r ένας πραγματικός αριθμός. Τέτοια σχήματα έχουν χρησιμοποιηθεί σε εφαρμογές SLAM τόσο στην γενική μορφή του ορισμού τους [17, 6] όσο και σε μία πιο περιορισμένη έκδοση, τις λεγόμενες εκφυλισμένες τετραγωνικές επιφάνειες (*degenerate quadrics*) [7]. Τέτοιες επιφάνειες πληρούν την εξίσωση:

$$(\boldsymbol{x} - \boldsymbol{p})^T \boldsymbol{A} (\boldsymbol{x} - \boldsymbol{p}) = \boldsymbol{0} \tag{7}$$

όπου $\boldsymbol{p}$ είναι το κεντροειδές και $\boldsymbol{A}$ είναι ένας συμμετρικός πίνακας, ο οποίος τελικά καθορίζει και τη γεωμετρία μέσω της παραγοντοποίησής του: $\boldsymbol{A} = \boldsymbol{R} \boldsymbol{\Lambda} \boldsymbol{R}^T$, όπου ο $\boldsymbol{R}$ περιέχει τα ιδιοδιανύσματα (τους τοπικούς άξονες γύρω από τους οποίους εκτείνεται η υπερεπιφάνεια) και ο διαγώνιος πίνακας $\boldsymbol{\Lambda}$ περιέχει τις ιδιοτιμές, που επιβάλλουν το σχήμα της. Συγκεκριμένα, οι περιπτώσεις σχημάτων που μας ενδιαφέρουν είναι οι ακόλουθες:

$$\boldsymbol{\Lambda_M} = \mathrm{diag}(\boldsymbol{\lambda}), \qquad \boldsymbol{\lambda} = \begin{cases} (1,\ 1,\ 1) & \text{για σημεία} \\ (0,\ 1,\ 1) & \text{για ευθείες} \\ (1,\ 0,\ 0) & \text{για επίπεδα} \end{cases} \tag{8}$$

Η αιτιολόγηση για αυτήν τη γεωμετρική ερμηνεία προκύπτει καθαρά από τη μαθηματική λειτουργία τους. Αν ένα στοιχείο τις διαγωνίου είναι 0, πρακτικά αφήνει έναν βαθμό ελευθερίας, αφού αδρανοποιεί τον αντίστοιχο περιορισμό από τον ορισμό των τετραγωνικών επιφανειών. Αν είναι 1, τότε τον ενεργοποιεί, με αποτέλεσμα εν τέλει να αφαιρεί έναν βαθμό ελευθερίας.

Επεκτείνοντας τις παραπάνω έννοιες, θα ορίσουμε ένα matchable ως μία τριπλέτα με τα εξής δεδομένα:

$$M := (\boldsymbol{p}_M,\ \boldsymbol{R}_M,\ \boldsymbol{\Lambda}_M) \tag{9}$$

αντίστοιχα δηλαδή ένα κεντροειδές $\boldsymbol{p}_M$, έναν πίνακα στροφής $\boldsymbol{R}_M$ για το τοπικό πλαίσιο συντεταγμένων και έναν πίνακα σχήματος $\boldsymbol{\Lambda}_M$. Τέτοια αντικείμενα ανιχνεύουμε και παρακολουθούμε τελικά, όπως γίνεται στο σύστημα SASHAGO, όπως φαίνεται στο Σχ. 5. Μαθηματικά, χρειάζεται καταρχάς να ορίσουμε τα σφάλματα μεταξύ δύο matchables $\boldsymbol{M}_a$ και $\boldsymbol{M}_b$, ως εξής:

$$\boldsymbol{e}(\boldsymbol{M}_a, \boldsymbol{M}_b) = \begin{bmatrix} \boldsymbol{e}_p \\ \boldsymbol{e}_d \\ e_o \end{bmatrix} = \begin{bmatrix} \boldsymbol{R}_b^T(\boldsymbol{p}_a - \boldsymbol{p}_b) \\ \boldsymbol{d}_a - \boldsymbol{d}_b \\ \boldsymbol{d}_a^T \boldsymbol{d}_b \end{bmatrix} \tag{10}$$

$$\boldsymbol{\Omega} = \boldsymbol{C} \cdot \bar{\boldsymbol{\Omega}} \cdot \boldsymbol{C}^T,\ \boldsymbol{C} = \begin{bmatrix} \boldsymbol{C}_p & \boldsymbol{0}_{3\times3} & \boldsymbol{0}_{3\times1} \\ \boldsymbol{0}_{3\times3} & \boldsymbol{C}_d & \boldsymbol{0}_{3\times1} \\ \boldsymbol{0}_{1\times3} & \boldsymbol{0}_{1\times3} & C_o \end{bmatrix} \tag{11}$$

όπου $\boldsymbol{d}$ είναι το διάνυσμα του τοπικού άξονα x του matchable, οι πίνακες $\bar{\boldsymbol{\Omega}}, \boldsymbol{\Omega}$ είναι πίνακες θορύβου των μετρήσεων και $\boldsymbol{C}$ είναι ο πίνακας ενεργοποίησης, που επιλέγει ποιες διαστάσεις των σφαλμάτων θα ενεργοποιούνται ανάλογα με τη γεωμετρία του matchable, όπως φαίνεται στον Πίνακα 3.

$\hat{\boldsymbol{z}}_{ij} \setminus \boldsymbol{z}_{ij}$	Σημείο	Γραμμή	Επίπεδο
Σημείο	$(\mathbb{I}, \boldsymbol{0}, 0)$	$(\boldsymbol{\Lambda}_{\boldsymbol{z}_{ij}}, \boldsymbol{0}, 0)$	$(\boldsymbol{\Lambda}_{\boldsymbol{z}_{ij}}, \boldsymbol{0}, 0)$
Ευθεία	-	$(\boldsymbol{\Lambda}_{\boldsymbol{z}_{ij}}, \mathbb{I}, 0)$	$(\boldsymbol{\Lambda}_{\boldsymbol{z}_{ij}}, \boldsymbol{0}, 1)$
Επίπεδο	-	-	$(\boldsymbol{\Lambda}_{\boldsymbol{z}_{ij}}, \mathbb{I}, 0)$

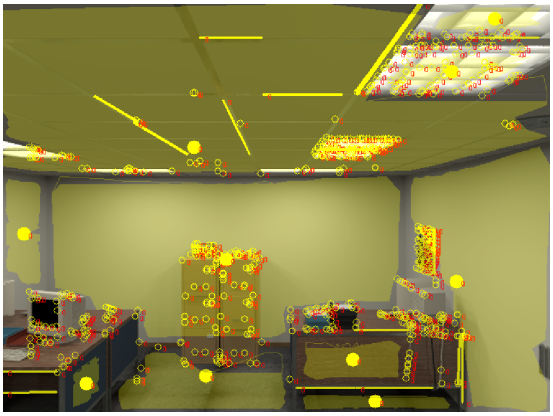
Πίνακας 3: Τύποι πινάκων ενεργοποίησης $(\boldsymbol{C}_p, \boldsymbol{C}_d, C_o)$

Ορίζοντας κατάλληλα τον τελεστή $\boxplus$ μπορούμε να αποκτήσουμε μία έκφραση για το τι συμβαίνει όταν μία μικρή διαταραχή επηρεάζει ένα matchable και εν τέλει με βάση αυτήν τη σχέση θα οριστεί αντίστοιχα η Ιακωβιανή του σφάλματος και θα προχωρήσει κανονικά η διαδικασία της βελτιστοποίησης. Ξεκινάμε με την περιγραφή της διαταραχής ενός matchable:

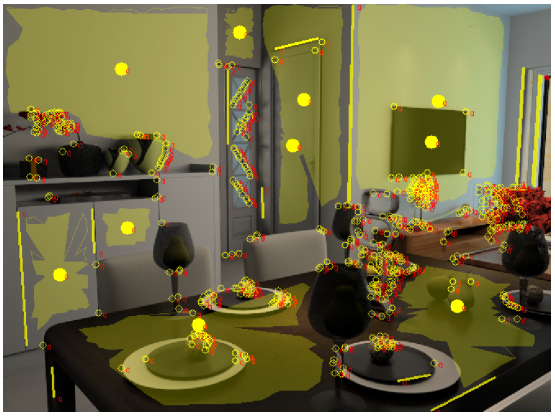
$$\Delta\boldsymbol{m} = \begin{bmatrix} \Delta\boldsymbol{p}^T & \Delta\alpha_y^T & \Delta\alpha_z^T \end{bmatrix}^T \tag{12}$$

$$\boldsymbol{M} \boxplus \Delta\boldsymbol{m} = (\Delta\boldsymbol{p}_{\boldsymbol{M}} + \Delta\boldsymbol{p},\ \boldsymbol{R}_{\boldsymbol{M}} \cdot \Delta\boldsymbol{R},\ \boldsymbol{\Lambda}_{\boldsymbol{M}}) \tag{13}$$

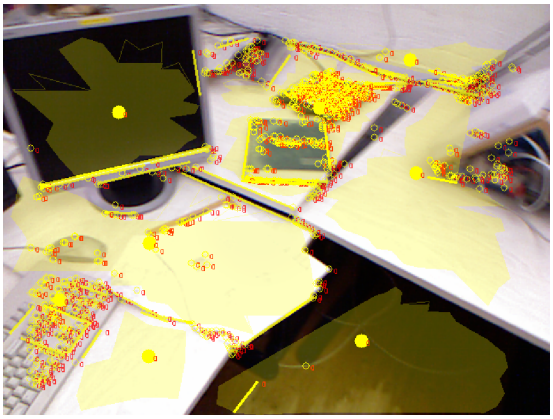
όπου $\Delta\boldsymbol{p}$ είναι η διαταραχή στο κεντροειδές και $\Delta\alpha_y$, $\Delta\alpha_z$ οι περιστροφικές διαταραχές. Στα παραπάνω σημειώνουμε ότι η περιστροφή γίνεται μόνο γύρω από τους μη-κύριους άξονες του matchable, καθώς υπάρχει συμμετρία γύρω από τον κύριο άξονα x. Επίσης η σειρά των στροφών ακολουθεί την σύμβαση: $\Delta\boldsymbol{R} = \boldsymbol{R}_y(a_y) \cdot \boldsymbol{R}_z(a_z)$.



(α′) ICL-NUIM tr0



(β′) InteriorNet



(γ′) TUM RGBD fr1

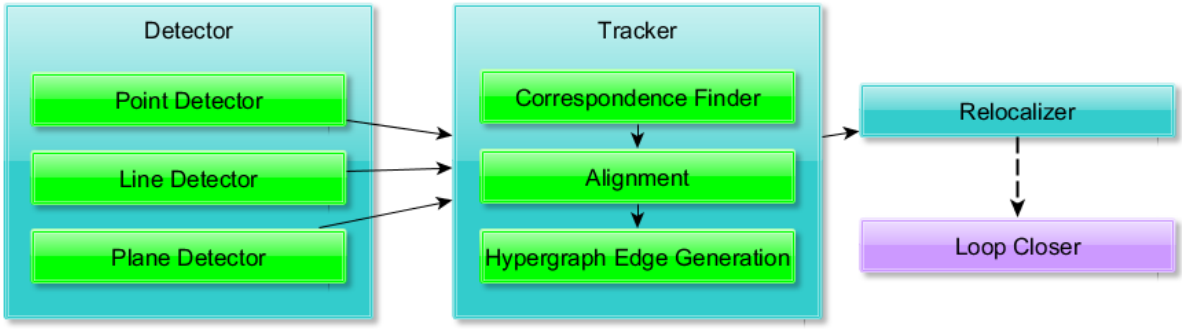
(δ′) TUM RGBD fr3

Σχήμα 5: Παραδείγματα γεωμετρικών ανιχνεύσεων σε διαφορετικά dataset. Παρατηρούμε ότι σε πραγματικά δεδομένα, ο θόρυβος δυσκολεύει την ανίχνευση επιπέδων (TUM dataset).

3.2 Το σύστημα SASHAGO

Όπως φαίνεται στο Σχ. 6, το σύστημα απαρτίζεται από ανιχνευτές των κύριων γεωμετρικών χαρακτηριστικών που χρησιμοποιούνται, από ανιχνευτές και ευθυγραμμιστές για τα χαρακτηριστικά αυτά και από διαχειριστές κλεισιμάτων βρόχων. Η αρχιτεκτονική αυτή μπορεί να αναλυθεί περαιτέρω ως εξής:

- ανιχνευτής σημείων (point detector): χρησιμοποιεί σημεία ORB [24] της `OpenCV` για την ανίχνευση και περιγραφή σημειακών χαρακτηριστικών του χώρου
- ανιχνευτής γραμμών (line detector): βασισμένος στον Line Segment Detector της `OpenCV` [52], ανιχνεύει ευθύγραμμα τμήματα σε εικόνες
- ανιχνευτής επιπέδων (plane detector): ειδικό σύστημα που συνενώνει σημεία από το



Σχήμα 6: Σύνοψη της εσωτερικής αρχιτεκτονικής του συστήματος SASHAGO.

τρισδιάστατο pointcloud της κάμερας (point clustering) με σκοπό να ανιχνεύση επίπεδα και τα κάθετα σε αυτά διανύσματα [53]

- ευρετής αντιστοιχιών (correspondence finder): βρίσκει, με χρήση δομών KD-tree, αντίστοιχα σημεία από ένα frame σε προηγούμενα σημεία του χάρτη
- ευθυγραμμιστής (aligner): υπολογίζει τον βέλτιστο μετασχηματισμό ανάμεσα στην καινούρια σκηνή και τον χάρτη για το τρέχον frame
- κλεισίματα βρόχων (relocalizer & loop closer): κομμάτια του συστήματος που έχουν να κάνουν με τη διαχείριση κλεισίματος βρόχων, το πρώτο για την προσθήκη των ακμών στον γράφο πόζας, το δεύτερο για την πρόσθετη βελτιστοποίηση

Τα παραπάνω συνοψίζουν τις κύριες λειτουργίες του front-end του συστήματος. Σε γενικές γραμμές, το back-end δεν έχει κάποια περίπλοκη εσωτερική δομή, καθώς η μόνη του λειτουργία είναι να βελτιστοποιεί τον γράφο πόζας με την βιβλιοθήκη g2o [33] σαν βάση (μαζί με τις απαραίτητες προεκτάσεις για τη διαχείριση matchables), που είναι σχεδιασμένη για να επιτελεί αυτό ακριβώς το έργο και έχει χρησιμοποιηθεί ευρέως από δημοφιλή συστήματα SLAM με γράφους πόζας.

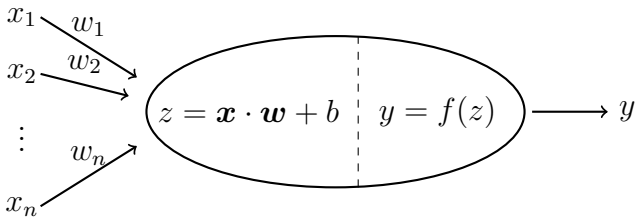
4. Σημασιολογική Κατάτμηση Εικόνων

4.1 Εισαγωγή

Το βασικό πρόβλημα που καλούμαστε να λύσουμε στον τομέα της σημασιολογικής κατάτμησης είναι η αναγνώριση του τι απεικονίζεται και πού στην εικόνα βρίσκεται. Εδώ υπάρχει μεγάλο περιθώριο για ερμηνεία του τι ακριβώς απαιτείται να βρεθεί. Κάποιες ερευνητικές δουλειές επικεντρώνονται στο να δώσουν ορθογώνια όρια σε συντεταγμένες pixel που περιγράφουν πού στην εικόνα βρίσκεται μία οντότητα. Η ορθογώνια περιγραφή συνοδεύεται και από μία σημασιολογική κλάση, που πρακτικά είναι ένας ακέραιος αριθμός που περιγράφει τον τύπο το ανιχνευθέντος αντικειμένου, με βάση μία αυθαίρετη αντιστοίχηση ανάμεσα σε κλάσεις και τους αριθμούς τους. Άλλες προσεγγίσεις γίνονται πιο συγκεκριμένες στα αποτελέσματά τους, δίνοντας μία πυκνή κατάτμηση. Μία τέτοια κατάτμηση επιστρέφει τόσες εξόδους όσα και τα pixel της εικόνας, με έναν αριθμό κλάσης για κάθε pixel. Τέλος, έχουν υπάρξει ειδικά συστήματα που στοχεύουν στην κατάτμηση στιγμιοτύπων, όπου η αναγνώριση δεν περιλαμβάνει μόνο την κλάση που ανιχνεύθηκε, αλλά και έναν ειδικό κωδικό που περιγράφει ποιο στιγμιότυπο αυτής της κλάσης (ξανα)παρατηρείται.

4.2 Νευρωνικά δίκτυα

Στην σύγχρονη βιβλιογραφία της Όρασης Υπολογιστών έχουν επικρατήσει τα τεχνητά νευρωνικά δίκτυα. Ο τομέας της σημασιολογικής κατάτμησης αποτελεί ιδιαίτερα εύφορο έδαφος για αυτές τις τεχνολογίες [54, 55], καθώς τα αντικείμενα του πραγματικού κόσμου έχουν μεγάλη ποικιλομορφία, επομένως η μηχανική μάθηση μπορεί να δώσει καλύτερες λύσεις σε αυτές τις συνθήκες, με τις γνωστές προϋποθέσεις: μεγάλο απόθεμα δεδομένων εκπαίδευσης, μία αρχιτεκτονική για το νευρωνικό δίκτυο και επαρκή υπολογιστική ισχύ για την εκπαίδευσή της, συνήθως με χρήση σύγχρονων GPU [56]. Η βασική μονάδα ενός νευρωνικού δικτύου είναι ο νευρώνας (βλ. Σχ. 7), ο οποίος ανασυνδυάζει τις εισόδους του για να παράξει μία έξοδο, πρώτα γραμμικά και μετά με μία μη-γραμμική συνάρτηση.

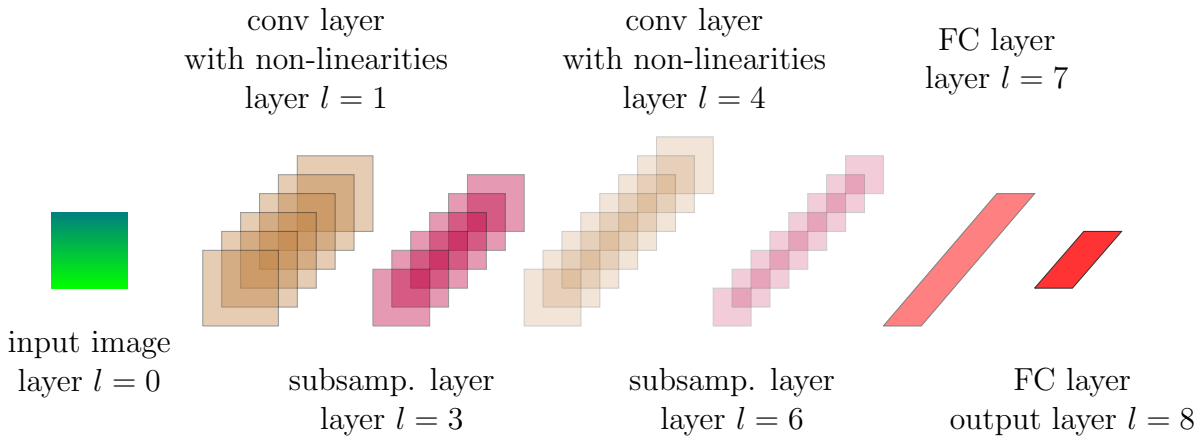


Σχήμα 7: Η δομή ενός νευρώνα. Τα βάρη w_i σταθμίζουν τις εισόδους x_i. Στη συνέχεια προστίθεται το bias b, και το αποτέλεσμα περνάει από την μη-γραμμική συνάρτηση ενεργοποίησης f πριν την έξοδο.

Η βασική ιδέα πίσω από την εκπαίδευση πολλών αρχιτεκτονικών σημασιολογικής κατάτμησης υπάγεται στην επιβλεπόμενη μάθηση. Συγκεκριμένα, τα δεδομένα εκπαίδευσης

έχουν επισημειωθεί από ανθρώπους και για την διαδικασία της εκπαίδευσης το σύστημα σε πρώτη φάση απλώς κάνει υπολογισμούς για ένα ή περισσότερα από τα δεδομένα εκπαίδευσης. Η απόκρισή του μετά συγκρίνεται με την αναμενόμενη, με βάση το τι έχει δοθεί στις επισημειώσεις των δεδομένων εκπαίδευσης. Η μεταξύ τους διαφορά ποσοτικοποιείται με χρήση μιας συνάρτησης απώλειας (*loss function*), και στη συνέχεια ένας αλγόριθμος οπισθοδιάδωσης (*back-propagation*) επηρεάζει διαδοχικά τα βάρη των νευρώνων με σκοπό να φέρει το σύστημα σε μία κατάσταση όπου η απόκρισή του να είναι κοντινότερα στην επιθυμητή. Με κατάλληλη επιλογή μη-γραμμικών ενεργοποιήσεων των νευρώνων, συνάρτησης απώλειας και των διαφόρων υπερπαραμέτρων της λογικής της οπισθοδιάδωσης το σύστημα θα συγκλίνει μετά από πολλές επαναλήψεις σε κατάλληλα βάρη, ώστε οι αποκρίσεις του ιδανικά να είναι σωστές ακόμα και για δεδομένα που δεν έχει δει ποτέ στη διαδικασία της εκπαίδευσης.

Μία σημαντική αρχιτεκτονική ιδέα για νευρωνικά δίκτυα που έχουν εφαρμογές στην Όραση είναι η συνέλιξη. Η βασική αυτή διεργασία για την κλασσική Όραση βρίσκει το ανάλογό της στον κόσμο των νευρωνικών δικτύων στα συνελικτικά νευρωνικά δίκτυα (*Convolutional Neural Networks - CNNs*). Μία επισκόπηση της γενικής αρχιτεκτονικής τους φαίνεται στο Σχ. 8. Η ιδέα είναι να οριστούν στρώσεις νευρώνων που να δουλεύουν με τοπική πληροφορία, επαναχρησιμοποιώντας βάρη και προσομοιάζοντας τελικά την λογική των κλασσικών συνελικτικών φίλτρων. Αυτή η σχεδίαση βοηθά στις επιδόσεις λόγω των λιγότερων συνδέσεων, άρα και βαρών προς ανεύρεση, αλλά επίσης περνάει την διαισθητική αντίληψη για την τοπικότητα της πληροφορίας στις εικόνες έμμεσα, μέσα στην ίδια την αρχιτεκτονική του δικτύου.



Σχήμα 8: Επισκόπηση βασικής αρχιτεκτονικής ενός συνελικτικού νευρωνικού δικτύου. Από αριστερά προς τα δεξιά, η αρχική εικόνα περνά από μία αλληλουχία συνελικτικών και υποδειγματοληπτικών στρώσεων, πριν φτάσει τελικά στις πλήρως συνδεδεμένες στρώσεις εξόδου.

Ένα συγκεκριμένο δίκτυο με ιδιαίτερη σημασία για την παρούσα εργασία είναι το Faster R-CNN [8]. Πρόκειται για ένα σύστημα που ακολουθεί μία σχεδίαση που είναι αρκετά διαδε-

δομένη στην βιβλιογραφία της σημασιολογικής κατάτμησης. Πρώτα, οι συνελικτικές στρώσεις ενός αρχικού δικτύου περνάνε πληροφορία σε ένα δίκτυο πρότασης περιοχών, *Region Proposal Network (RPN)*, που με τη σειρά του δίνει προτάσεις για το πού μπορεί να υπάρχει κάποιο αντικείμενο. Μετά γίνεται και η διαδικασία ανίχνευσης και ταξινόμησης για να προκύψει το τελικό αποτέλεσμα. Υπάρχουν προφανώς και πολλές άλλες αρχιτεκτονικές, τόσο για ΣΦΚ-κατατμήσεις [57, 58], αλλά και για πυκνές κατατμήσεις [59].

4.3 Εφαρμογές σε SLAM

Πολλές διαφορετικές προσεγγίσεις έχουν προταθεί για την σύμπλεξη σημασιολογίας και SLAM. Σε ένα επίπεδο πιο άμεσο από το SLAM, λύσεις οπτικής οδομετρίας έχουν χρησιμοποιήσει σημασιολογία για να βελτιώσουν τις εκτιμήσεις της κίνησης που προσφέρουν. Τέτοιες λύσεις μπορεί να προσθέτουν όρους βελτιστοποίησης με σημασιολογικό περιεχόμενο από την απόκλιση της σημασιολογίας σε διαδοχικά frame [28], ή μπορεί να αθροίζουν ανεξάρτητα σφάλματα φωτομετρίας, γεωμετρίας και αμιγούς σημασιολογίας [29]. Για το SLAM, μία ιδέα για αραιά συστήματα που χρησιμοποιούν ανιχνευτές με ΣΦΚ είναι να επιτραπεί στη σημασιολογία να επηρεάσει κάπως τον γράφο πόζας. Μία προσέγγιση θα ήταν η εισαγωγή νέων σημασιολογικών ακμών, με χρήση αλγορίθμων μεγιστοποίησης αναμενομένων (*Expectation Maximization - EM*), που όμως έρχεται με σημαντικούς υπολογιστικούς περιορισμούς σε κάποιες περιπτώσεις [9, 51]. Εμείς, όπως θα δούμε στη συνέχεια, αλλάζουμε τον γράφο με την εισαγωγή βαρών σε υπάρχουσες ακμές, μειώνοντας έτσι το υπολογιστικό φορτίο. Από την άλλη, σε πυκνά συστήματα SLAM, διάφορες ιδέες έχουν προταθεί για την ενημέρωση της πίστης του συστήματος ανά pixel, με βάση π.χ. Bayesian κανόνες ενημέρωσης πιθανοτήτων [60].

5. Η Συνεισφορά Μας

Είμαστε σε θέση να αναλύσουμε την δική μας επέμβαση στο σύστημα SASHAGO [7] για να εμπλουτιστεί με σημασιολογία. Η προσέγγισή μας χωρίζεται σε τρία στάδια: πρώτα, έχουμε κάποιους βασικούς ορισμούς για επεκτεταμένα matchable, έπειτα τον μηχανισμό διαχείρισης της σημασιολογικής πληροφορίας και τέλος κάποιες τροποποιήσεις για το αμιγώς γεωμετρικό κομμάτι του συστήματος που βοήθησαν πολύ στην ακρίβεια της τελική υλοποίησης.

5.1 Βασικοί ορισμοί

Ξεκινάμε ορίζοντας τι σημαίνει σημασιολογική μέτρηση στο πλαίσιο του προτεινόμενου συστήματος. Η i-στη σημασιολογική μέτρηση του frame t ορίζεται ως:

$$\boldsymbol{s}^{t,i} = (s_c^{t,i},\ s_s^{t,i},\ \boldsymbol{s}_b^{t,i}) \tag{14}$$

όπου το πρώτο στοιχείο της τριπλέτας είναι ο ακέραιος της κλάσης, το δεύτερο είναι ένας πραγματικός αριθμός στο διάστημα $[0.0, 1.0]$ που αντιστοιχεί στην αυτοπεποίθηση του σημασιολογικού ανιχνευτή και το τελευταίο είναι μία ελάχιστη παραμετροποίηση του ΣΦΚ της μέτρησης. Αυτή η παραμετροποίηση αποτελείται από δύο σημεία, ένα για την άνω αριστερά και ένα για την κάτω δεξιά γωνία του ορθογωνίου σε συντεταγμένες pixel, που επαρκούν για να προσδιορίσουν πλήρως το ΣΦΚ.

Με βάση τα παραπάνω, ορίζουμε την έννοια του σημασιολογικά επεκτεταμένου matchable ως εξής:

$$\boldsymbol{M}^s := (\boldsymbol{p}_M,\ \boldsymbol{R}_M,\ \boldsymbol{\Lambda}_M,\ \boldsymbol{s}_M) \tag{15}$$

όπου τα πρώτα τρία στοιχεία είναι τα γνωστά από την Εξ. 9 και το τελευταίο είναι η βέλτιστη σημασιολογική μέτρηση που αντιστοιχήθηκε με αυτό το matchable.

5.2 Μεθοδολογία

Μεθοδολογικά, η αντιστοίχηση αυτή γίνεται δύσκολη σε περιπτώσεις όπου ένα matchable περιέχεται σε δύο ή περισσότερες σημασιολογικές ανιχνεύσεις ταυτόχρονα. Αυτό μπορεί να συμβεί όταν ένα ΣΦΚ περιλαμβάνεται εξ' ολοκλήρου σε ένα άλλο, όπως για παράδειγμα ένα βάζο που λόγω της οπτικής γωνίας του πράκτορα πλαισιώνεται εντελώς από το τραπέζι πάνω στο οποίο είναι τοποθετημένο. Τα matchables λοιπόν αντιστοιχίζονται πάντα σε εκείνη την παρατήρηση που περιέχει το σημασιολογικά αντιπροσωπευτικό τους σημείο (π.χ. το κεντροειδές) και έχει το ελάχιστο pixel-εμβαδόν. Συγκεκριμένα:

$$\hat{\boldsymbol{s}}_b^{t,j} = \underset{\boldsymbol{s}_b \in S_b^{t,j}}{\arg\min}\ \mathrm{Area}(\boldsymbol{s}_b) \tag{16}$$

Έχοντας βρει προτεινόμενες αντιστοιχίες matchable με σημασιολογικές μετρήσεις, το επόμενο βήμα είναι να αναγνωρίσουμε ότι οι αντιστοιχίσεις που γίνονται είναι από τη φύση

τους θορυβώδεις. Ο θόρυβος προκύπτει από πολλαπλές πηγές: από τη διαδικασία ανίχνευσης και περιγραφής των matchable, από τη διαδικασία παρακολούθησής τους στο χρόνο, από τη διαδικασία της σημασιολογικής κατάτμησης και τέλος από την διαδικασία διασύνδεσης σημασιολογίας και γεωμετρίας που είδαμε παραπάνω. Όλα τα παραπάνω οδηγούν σε μία κατάσταση όπου αναπόφευκτα πρέπει να γίνει κάποια στάθμιση των δεδομένων που έχουμε συλλέξει, ώστε να μην γίνονται τεράστια λάθη στην αντίληψη του ρομποτικού πράκτορα. Άλλες δουλειές πετυχαίνουν μία αποθορυβοποίηση μέσω αλγορίθμων EM [9, 51], οι οποίοι όπως σχολιάσαμε και σε προηγούμενη παράγραφο έχουν υπολογιστικούς περιορισμούς. Εμείς, θέλοντας να κάνουμε πλούσια σημασιολογική ανίχνευση και με σκοπό να διοχετεύσουμε αυτήν την πληροφορία στο βασικό γεωμετρικό επίπεδο, θέλουμε μία πιο άμεση λύση, επομένως στρεφόμαστε στην ακόλουθη λογική της σημασιολογικής ψηφοφορίας:

$$\hat{l}_c^{T,j} = \arg\max_{v \in \mathcal{C}_+} \sum_{t=0}^{T} \mathbb{1}[s_c^{t,j'} = v] \cdot s_s^{t,j'} \tag{17}$$

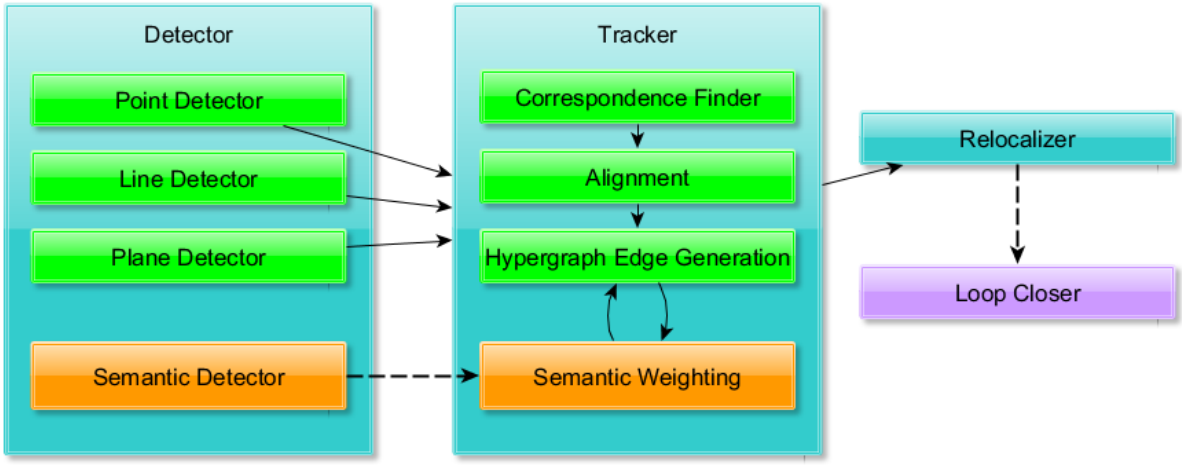
όπου έχουμε για το j-στο ορόσημο $l_c^{T,j}$ μία ψήφο για κάθε σημασιολογική παρατήρηση που βρήκε αυτό το ορόσημο στον δείκτη $j' = \text{idx}(j,t)$ του t-στου frame. Η ιδέα είναι να ενισχύουμε με μία υπολογιστικά απλή πράξη την υπόθεση μίας σημασιολογικής διασύνδεσης ανάλογα με το σκορ εμπιστοσύνης του ανιχνευτή για τη συγκεκριμένη διασύνδεση σε κάθε βήμα της παρακολούθησης του ορόσημου. Επομένως, αν γίνουν κάποια (σχετικά λίγα) λάθη, σε οποιοδήποτε στάδιο της διαδικασίας συσχέτισης, θα χαθούν μέσα στο πλήθος από σωστές ψήφους.

Η ανανέωση του γράφου πόζας γίνεται με τον υπολογισμό ενός κατάλληλου βάρους:

$$w_s^{t,j',j} = f\left(\mathcal{D}_C(s_c^{t,j'}, l_c^{t,j})\right) \simeq \begin{cases} 1 & , \; s_c^{t,j'} = l_c^{t,j} \\ W_p & , \; s_c^{t,j'} \neq l_c^{t,j} \end{cases} \tag{18}$$

Το βάρος αυτό δρα πολλαπλασιαστικά στον όρο σφάλματος και προκύπτει ως εξής: αν η τρέχουσα σημασιολογική μέτρηση συμφωνεί με την πλειοψηφία για το αντίστοιχο ορόσημο, τότε το βάρος είναι μονάδα, επομένως δεν έχει καμία επίπτωση στο γράφο. Αν δεν συμφωνεί, τότε παίρνει μία τιμή ανάλογη με το πόσο διαφορετική είναι η κλάση που ανιχνεύθηκε από την αναμενόμενη. Το σημείο αυτό είναι κάπως λεπτό, καθώς ιδανικά θα περιελάμβανε κάποιον πίνακα διάκρισης (*confusion matrix*) που να δίνει τις πιθανότητες να γίνει λάθος ανάμεσα σε οποιοδήποτε ζεύγος κλάσεων. Τέτοιοι πίνακες όμως εν γένει δεν είναι διαθέσιμοι για αυθαίρετες κατανομές δεδομένων εισόδου και έτσι κι αλλιώς δεν αναφέρονται για αρχιτεκτονικές σημασιολογικής κατάτμησης (συνήθως παρατίθενται στατιστικά mAP κτλ). Συνεπώς, προσαρμόζουμε την στρατηγική μας, δημιουργώντας υπερκλάσεις που εμπεριέχουν σύνολα κλάσεων που μπερδεύονται εύκολα μεταξύ τους. Η ομαδοποίηση γίνεται εμπειρικά και εξαλείφει κατά περίπτωση τα προβλήματα που προκύπτουν σε ένα συγκεκριμένο σύνολο δεδομένων.

5.3 Υλοποίηση



Σχήμα 9: Σύνοψη της αρχιτεκτονικής του τελικού συστήματος.

Πέρα από αυτές τις θεωρητικές προεκτάσεις, των οποίων η υλοποίηση ακολούθησε ακριβώς τη λογική που παρουσιάσαμε παραπάνω, έγινε και μία προσπάθεια να τροποποιηθεί η αλγοριθμική υλοποίηση του γεωμετρικού pipeline της λύσης. Εκτός από την βηματική προσαρμογή της παραμετροποίησης της λύσης για κάθε περιβάλλον, δουλέψαμε πάνω στις συναρτήσεις ευθυγράμμισης των διαδοχικών σκηνών. Συγκεκριμένα, τροποποιήσαμε την ευθυγράμμιση με τρεις τρόπους:

- αλλαγή του κώδικα αναζήτησης πλησιέστερων γειτόνων στην εσωτερική δομή KD-tree
- αλλαγή της λογικής υπολογισμού της απόσβεσης του βήματος της αριθμητικής μεθόδου πρώτης εκτίμησης ευθυγράμμισης
- πειράματα με μία υβριδική συνάρτηση υπολογισμού της εγγύτητας γειτονικών matchable

Ο συνδυασμός σημασιολογίας με τις παραπάνω βελτιώσεις είναι που δίνει την τελική μορφή στην προτεινόμενη λύση αυτής της εργασίας. Αναφορικά με την υλοποίηση της σημασιολογίας αυτής καθαυτής, παρουσιάζουμε συνοπτικά την τελική αρχιτεκτονική στο Σχ. 9.

5.4 Πειραματικά αποτελέσματα

Για την αξιολόγηση τροχιών στο SLAM υπάρχουν διάφορες τεχνικές και πολλές μετρικές που τις υποστηρίζουν. Εμείς, εφόσον έχουμε ένα αραιό σύστημα SLAM, θα ακολουθήσουμε την λογική πολλών αντίστοιχων συστημάτων, ελέγχοντας ποσοτικά την ακρίβεια της εκτίμησης πόζας με χρήση του απόλυτου σφάλματος τροχιάς (*Absolute Trajectory Error - ATE*). Σημειώνουμε ότι ο υπολογισμός αυτού του σφάλματος έχει ως απαραίτητη προϋπόθεση να έχει γίνει πρώτα βέλτιστη ευθυγράμμιση των συγκρινόμενων ακολουθιών, κάτι που επιτυγχάνεται μέσω διαδικασίας βασισμένης στην αποσύνθεση SVD [10, 14]. Επίσης, εισάγουμε μία

Πίνακας 4: Αποτελέσματα ICL-NUIM (RMS ATE [m])

Ακολουθία	Ours (w/o sem.)	SASHAGO	ORB-SLAM2
tr0	**0.0131**	0.0156	0.0212
lr0	**0.0238**	0.0519	0.0036
lrkt1	**0.0046**	0.0091	0.0392
lrkt2	**0.0067**	0.0148	0.0281

νέα μετρική για να ποσοτικοποιήσουμε την μέση σημασιολογική πληροφορία ανά frame, ως τη διάμεσο και μέση τιμή των συνολικών σημασιολογικών ανιχνεύσεων ανά εικόνα εισόδου (*SemInfo*). Με αυτόν τον τρόπο έχουμε μία διαισθητική προσέγγιση του πόσο πλούσια σε σημασιολογία είναι κάθε ακολουθία.

Επιβεβαιώνουμε την λειτουργικότητα του συστήματός μας σε τρία διαφορετικά σύνολα δεδομένων. Ξεκινάμε με ένα σύνολο αποκλειστικά για επιβεβαίωση των γεωμετρικών μας προεκτάσεων, το ICL-NUIM [11]. Αυτό το σύνολο δεδομένων είχε χρησιμοποιηθεί και για την αξιολόγηση του αρχικού συστήματος SASHAGO. Οι ακολουθίες περιέχουν φωτορεαλιστικές σκηνές από συνθετικούς εσωτερικούς χώρους, που παρουσιάζουν έντονη γεωμετρική κανονικότητα. Αυτό το οποίο κάνει τις σκηνές δύσκολες για σημασιολογικές προεκτάσεις είναι ότι δεν υπάρχει έτοιμο σημασιολογικό ground truth και δυστυχώς ανιχνευτές όπως ο Faster R-CNN αποτυγχάνουν, λόγω των κάπως απλοϊκών μοντέλων αντικειμένων που υπάρχουν στο χώρο, που ξεφεύγουν κατά πολύ από τις κατανομές εκπαίδευσής τους, και του ότι βρίσκονται σχετικά μακριά από την εικονική κάμερα, η οποία μάλιστα ξοδεύει σημαντικό κομμάτι της τροχιάς της παρατηρώντας τα όρια του δωματίου, που δεν δίνουν έτσι κι αλλιώς σημασιολογική πληροφορία.

Με βάση τα παραπάνω, ελέγχουμε μόνο το γεωμετρικό κομμάτι της λύσης μας και παραθέτουμε εν συντομία τα αποτελέσματα στον Πίνακα 4. Η βελτίωση από το SASHAGO είναι αισθητή και εκτός αυτού παίρνουμε και επιδόσεις εν γένει καλύτερες και από την μέθοδο σύγκρισης.

Το επόμενο σύνολο δεδομένων που μας ενδιαφέρει είναι το InteriorNet [34], το οποίο είναι συνθετικό, φωτορεαλιστικό και με αρκετά πιο πλούσιες σκηνές από το προηγούμενο. Μία κρίσιμη ιδιότητά του είναι ότι περιέχει σημασιολογικό ground truth, το οποίο χρησιμοποιούμε δίνοντας στο σύστημα τη δυνατότητα να το διαβάζει σαν να ερχόταν από κάποιον εξωτερικό ταξινομητή. Επίσης οι τροχιές είναι πιο απαιτητικές, καθώς οι κινήσεις που εκτελούνται έχουν προκύψει με αυτοματοποιημένες, τυχαίες διαδικασίες. Ακριβώς λόγω αυτής της τυχαιότητας, πολλά κομμάτια τους είναι δύσχρηστα, καθώς παρακολουθούν επίμονα κάποια αντικείμενα στα οποία έτυχε η κάμερα να πλησιάσει, ενώ σε μερικά σημεία η εικονική κάμερα περνάει μέσα από μικρά αντικείμενα (λάμπες, φυτά κτλ). Για αυτό έγινε προσεκτική επιλογή 'καθαρών' υπακολουθιών, με σκοπό να έχουμε ρεαλιστικά δεδομένα ελέγχου. Τα τεστ μας φαίνονται στον Πίνακα 5, όπου η λύση μας πετυχαίνει καλές επιδόσεις στον τομέα της σημασιολογίας, μένοντας ταυτόχρονα πολύ κοντά ή και ξεπερνώντας αμιγώς γεωμετρικές προσεγγίσεις.

Πίνακας 5: Αποτελέσματα InteriorNet (RMSE ATE [m])

Ακολουθία	SemInfo (Med/Avg)	Ours	Ours (w/o sem.)	SASHAGO	MaskFusion	ORB-SLAM2
133_open	8/8.2	**0.0075**	0.0091	0.0203	0.0190	0.0057
2r11_250-600	17/15.8	**0.0161**	0.0267	0.0772	0.0205	0.0147
3o11_open	9/8.9	0.0065	**0.0060**	0.0179	0.0179	0.0189
4o11_200-600	6/5.1	0.0343	**0.0319**	0.0565	0.4595	0.0447
5o11_full	12/12.7	**0.0099**	0.0111	0.0409	0.0110	[track lost]
5o33_200-675	16/15.7	**0.0292**	0.0539	0.0502	0.0575	0.0268
6o11_800-1000	12/11.4	**0.0128**	0.0135	0.0195	0.0164	0.0045

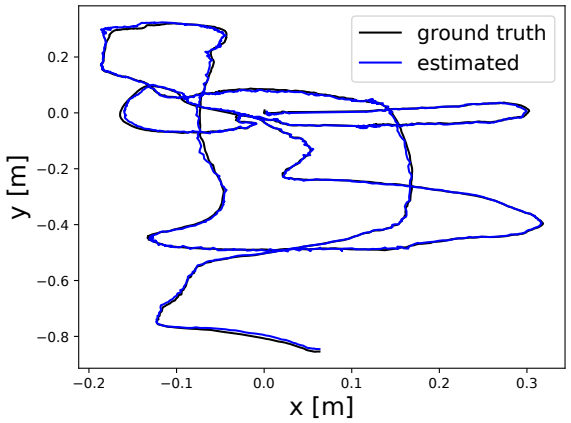
Σε αυτό το σύνολο δεδομένων μπορούμε να δούμε ξεκάθαρα και την βελτίωση που προσφέρεται από την σημασιολογία. Με εξαίρεση τις ακολουθίες `3o11_open` και `4o11_200-600`, που αποτελούνται από σχεδόν άδεια σκηνικά, επομένως η σημασιολογία δεν μπορεί να βοηθήσει ούτως ή άλλως, σε όλες τις άλλες υπάρχει βελτίωση. Επίσης, στις παραπάνω ακολουθίες φαίνεται η αξία των UGR, καθώς η ανίχνευση γραμμών και επιπέδων όντως δίνει καλύτερα αποτελέσματα.

Τέλος, αποφασίσαμε να πειραματιστούμε και με πραγματικά δεδομένα στο τρίτο και τελευταίο σύνολο δεδομένων μας, το TUM RGB-D dataset [35]. Σε αυτό το σύνολο υπάρχουν διάφορες ακολουθίες, στις οποίες δεν έχουμε σημασιολογικό ground truth, αλλά υπολογίζουμε σημασιολογικές κατατμήσεις με χρήση του Faster R-CNN. Το μεγαλύτερο πρόβλημα σε αυτά τα δεδομένα είναι ο θόρυβος. Οι εικόνες βάθους δεν έχουν μόνο ανακρίβειες, αλλά και κάποια pixel με τιμή 0, που αντιστοιχούν σε έλλειψη πληροφορίας. Για να καταπολεμήσουμε τον θόρυβο σε αυτές τις εικόνες καταλήξαμε τελικά στο φιλτράρισμα διαμέσου (median filtering). Επίσης, κάποια frame ορισμένων ακολουθιών παρουσιάζουν θόλωση λόγω κίνησης (*motion blur*), που επίσης δυσκολεύει το SLAM. Στον πίνακα 6 φαίνονται τα ποσοτικά αποτελέσματα.

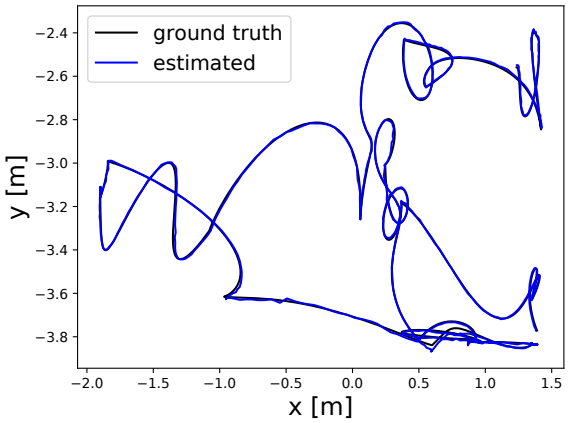
Πίνακας 6: Αποτελέσματα TUM RGB-D results (RMSE ATE [m])

Ακολουθία	SemInfo (Med/Avg)	Ours	Ours (w/o sem.)	SASHAGO	MaskFusion	ORB-SLAM2
fr1_desk	4/4.0	**0.0406**	0.0825	0.1188	0.9116	0.0203
fr2_desk_500	6/5.9	0.0183	**0.0178**	0.1350	0.0581	0.0053
fr2_desk_1000	6/5.9	0.0193	**0.0181**	0.1634	0.0581	0.0067
fr2_desk_full	4/4.6	0.0714	**0.0653**	0.2065	0.4559	0.0085
fr3_loh_200	11/10.2	**0.0161**	0.0267	0.0772	0.0337	0.0178
fr3_loh_1000	6/5.9	0.0521	0.0508	0.0867	**0.0272**	0.0087
fr3_loh_full	7/6.6	**0.1527**	0.1576	0.4145	0.2092	0.0097
fr3_sitting_static	10/10.4	0.0087	**0.0081**	0.0092	0.0117	0.0086
fr3_sitting_xyz	9/8.5	0.0441	**0.0436**	0.0506	0.0530	0.0092

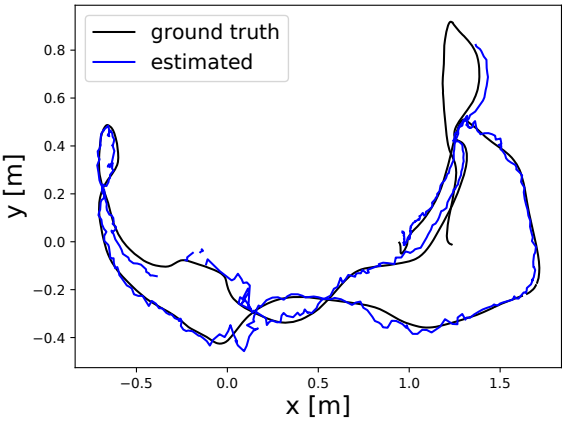
Βλέπουμε σαφή επιδείνωση της επίδοσης τόσο του δικού μας συστήματος, όσο και του MaskFusion [13]. Το ORB-SLAM2 [12] από την άλλη φαίνεται να πετυχαίνει καλύτερα αποτελέσματα, πιθανώς λόγω εσωτερικών μηχανισμών αποθορυβοποίησης. Σημειώνουμε ότι οι τελευταίες σειρές του πίνακα αντιστοιχούν σε ειδικές ακολουθίες του συνόλου δεδομένων που περιέχουν και δυναμικές οντότητες, συγκεκριμένα κινούμενοι άνθρωποι. Σημειώνουμε ότι αυτές είναι οι συνθήκες για τις οποίες είχε σχεδιαστεί ειδικά το MaskFusion και συνεπώς δείχνουν ακριβώς τους περιορισμούς της δικής μας μεθόδου, που δεν έχει ειδικές προβλέψεις για αυτά τα φαινόμενα.



(α′) Μία τροχιά από το ICL-NUIM.



(β′) Μία τροχιά από το InteriorNet.



(γ′) Μία τροχιά από το TUM RGB-D.

Σχήμα 10: Εκτιμήσεις τροχιών και ground truth για διάφορα σύνολα δεδομένων. Στο (γ′) σημειώνουμε την επίδραση του θορύβου, καθώς τα δεδομένα είναι από τον πραγματικό κόσμο.

6. Συμπεράσματα και Μελλοντική Έρευνα

Συμπεράσματα:

Στην παρούσα εργασία έχουμε προτείνει μία μέθοδο για το πρόβλημα του σημασιολογικού SLAM σε εσωτερικούς χώρους. Στα πλαίσια της προσέγγισης αυτής είδαμε διάφορα συστήματα οδομετρίας και SLAM, τη θεωρητική θεμελίωση των βασικών μεθόδων SLAM και τη μαθηματική διαδικασία μετατροπής του προβλήματος SLAM σε πρόβλημα βελτιστοποίησης μη-γραμμικών ελαχίστων τετραγώνων σε γράφο. Στη συνέχεια αναφέρθηκαν τα κύρια θεωρητικά σημεία που στηρίζουν τις ενοποιημένες γεωμετρικές αναπαραστάσεις και το πώς αυτές βοηθούν στο SLAM εσωτερικών χώρων. Ύστερα έγινε μία επισκόπηση της σημασιολογικής κατάτμησης εικόνων και των μεθόδων βαθιάς μάθησης που βοηθούν στην επίλυση αυτού του προβλήματος. Τέλος, αναλύθηκε η προτεινόμενη μεθοδολογία, από τις σκοπιές της θεωρίας, της υλοποίησης και του πειραματικού ελέγχου σε διάφορα περιβάλλοντα.

Από όλα τα παραπάνω μπορούμε να δούμε τα μεγάλα προτερήματα της υβριδικής προσέγγισης γεωμετρίας και σημασιολογίας στο SLAM εσωτερικών χώρων. Βέβαια, η σημασιολογία έχει πολλές εφαρμογές και σε εξωτερικούς χώρους, αλλά το σίγουρο είναι ότι ο συνδυασμός με τις ενοποιημένες γεωμετρίες βοηθά κυρίως για εσωτερικούς χώρους. Επιτρέπει σε αυτόνομα ρομποτικά συστήματα να προσαρμόζονται αυτόματα σε περιβάλλοντα με έντονα μεταβαλλόμενη ποικιλία αντικειμένων, από άδεια δωμάτια μέχρι χώρους γεμάτους έπιπλα. Επίσης, δίνει ένα νέο επίπεδο αντίληψης του πράκτορα για τον περιβάλλοντα χώρο, καθώς οι χάρτες που δημιουργούνται έχουν πρόσθετες σημασιολογικές πληροφορίες για αυτόν.

Μελλοντική έρευνα:

Ολόκληρη η ερευνητική περιοχή γενικά, αλλά και ειδικά η λύση μας, έχει ακόμα πολλά ανοιχτά ερωτήματα που χρήζουν διερεύνησης. Κατ΄ αρχάς, ο θόρυβος, όπως είδαμε και στα πειράματα, μειώνει σημαντικά την αποτελεσματικότητα των μεθόδων αυτών. Επομένως είναι κρίσιμο να βρεθούν στο μέλλον λύσεις για την αντιμετώπισή του, ειδικά για motion blur στα κανάλια RGB και θόρυβο αισθητήρα στα κανάλια βάθους. Επίσης, δυναμικές οντότητες στο πεδίο αντίληψης του πράκτορα μπορούν εύκολα να δημιουργήσουν λάθος εκτιμήσεις κίνησης. Εκεί η σημασιολογία μπορεί να βοηθήσει (όπως γίνεται στο MaskFusion), ή αλλιώς μπορεί να γίνει κάποια στατιστική ανάλυση τύπου RANSAC [61, 62] για να αφαιρεθούν outlier της κίνησης. Τέλος, ανοιχτό παραμένει το ερώτημα για το ποια είναι η καλύτερη αρχιτεκτονική για να επιτευχθεί η σημασιολογική κατάτμηση και το κατά πόσο αυτό μπορεί να γίνει σε πραγματικό χρόνο. Ειδικότερα, για το πρώτο ερώτημα μπορεί κανείς να αναρωτηθεί για το πώς θα συμπεριφερόταν το σύστημα αν είχε πρόσβαση σε πυκνή κατάτμηση. Για το δεύτερο ερώτημα, το αν θα χρειαστεί κάποια λογική διαλογής των frame για να επιλεχθούν keyframe στα οποία θα γίνεται περαιτέρω επεξεργασία είναι κρίσιμο, αλλά χρειάζεται ανάλυση για τη στρατηγική της διαλογής και την παραμετροποίηση της τελικής λύσης.

Chapter 1

Introduction

1.1 Basic Definitions

Robotics is the branch of technology dealing with the design, construction and operation of robots, also referred to as robotic agents. Of the many research areas that exist in this broad and ever-expanding field, perception is one of the most important. It encompasses all the different ways in which robots perceive their environments. Perception also exists in a multitude of different varieties in biological systems, including humans. However, unlike biological perception, robotic perception requires much stricter quantification of distances and shapes in the environment. Only then does it become possible to use the information gathered effectively, allowing the robot to interact safely and robustly with its surroundings, thus becoming truly useful. In visual perception systems, which are extremely popular in today's world, methods from *Computer Vision (CV)* are of paramount importance and are commonly employed to act as fundamental building blocks of complete systems. This highlights the interplay between robotics and CV, that has greatly helped in shaping the area of robotic perception.

Of course, the problem of automated spatial awareness is an extremely challenging one, with research spanning several disciplines over many decades. Not only is it theoretically difficult to account for the tremendous variability of possible environments a robotic agent can encounter, but other factors such as the design of sensors and noise modeling can seriously impact performance. For these reasons, initial approaches were confined to simplified abstract worlds and focused on probabilistic calculations. Later, as real-time (and increasingly often on-board) processing technologies became more prevalent in mobile robotics, the first implementations of trajectory estimators were designed, based on the concept of *odometry*, with an emphasis on multiple sensor fusion. These initial approaches mostly ignored the concept of maintaining an explicit mapping of the traversed environment, and instead focused on localization in a known map. Mapping solutions existed largely in separation, e.g. using occupancy grid representations for LiDAR sensors, assuming known localization. Naturally, the search for a framework that could execute both localization and mapping at the same time soon started; *Simultaneous Localization And Mapping (SLAM)*, as it came to be known, became an important problem in the community, with decades of research going into developing various solutions [15, 16]. With SLAM, we seek to enable robots to extract information about the topology of the surrounding space *and* their own trajectories, using on-board sensor data (e.g. LiDAR, cameras). SLAM was brought localization and mapping together, allowing robots to effectively and meaningfully negotiate unknown environments. Modern computational capabilities enabled real-time SLAM solutions to be deployed that were designed to use data streams from a large selection of possible sensor architectures, to both estimate robotic trajectories *and* generate a map representation of the traversed environment. The mathematical formulation also changed, as we will see later, with SLAM frequently being cast into a non-linear least squares optimization problem, and a variety of techniques

being developed both for its construction and its computational solution [3].

From the impressive variety of SLAM solutions available in the literature, one may discern some common structures and functions existing among them.

- **Sensor measurement collection:** sensor data can differ vastly depending on the specifics of the implementation, but the ultimate requirement is that the system be capable of calculating 3D coordinate estimates (initially with respect to the moving sensor's frame of reference) for a set of points. This set may be dense or sparse, reflecting the underlying logic of the algorithms involved.

- **Map structure:** different SLAM systems have different notions as to what exactly an environment map is; from lightweight structures, storing only sparse, salient features, to more descriptive ones, capturing a fully detailed 3D reconstruction of their surroundings. They typically account for the bulk of implementations' run-time memory consumption and are very frequently expanded, updated and queried, leading to strict performance requirements to maintain efficiency as they grow in size.

- **Loop closure detection:** routines in place to recognize when the agent revisits a known location. The optimization process takes this into account, thus alleviating accumulated pose estimation drift (seen in VO solutions - we discuss this further in Chap. 2) and improving accuracy.

1.2 Our Contribution

SLAM systems exist in many different varieties, each being tailored to function optimally under certain operating assumptions. For this work, we consider the scenario of indoor usage, inspired by assistive robotics applications for hospital patients. Let us first explain a key difference between indoor and outdoor spaces: geometric regularity. Indoor spaces typically contain clear lines and planes (floors, walls, ceilings etc.), which are not always so strikingly present in outdoor spaces. Accordingly, in this work we seek to utilize new technologies and theories for formulating and solving the SLAM problem, aiming to obtain improved results in challenging indoor spaces, by first exploring novel theories on *Unified Geometric Representations (UGRs)*. Over the past five years at least, research efforts have indeed been made in similar directions [63, 7], in an attempt to enrich the basic geometric units that an agent can detect utilize in describing the world around it.

Another promising research direction for SLAM systems is *Semantic SLAM*. This type of architecture incorporates some type of module that attempts to provide the robot with an understanding of *what* is in its vicinity. For instance, a semantic segmentation module can recognize that a cluster of pixels in the agent's camera feed in fact corresponds to a chair, a piece of information that could be used by the agent in various ways. Such powerful

semantic detectors are nowadays commonly implemented using *Machine Learning (ML)* approaches, specifically *Artificial Neural Networks (ANNs)*. In fact, there is quite a variety of them, with different accuracy and performance characteristics, and the literature is constantly evolving, as this is an independent problem from Computer Vision. Their success in advancing the state-of-the-art can be transferred to SLAM, provided that a system is capable of utilizing the detections made in a meaningful way.

Our approach is an attempt at combining the above ideas. We propose a system that can take advantage of both semantic richness and geometric regularities. Of the many imaginable ways such a novelty can be achieved, ours is primarily geared towards a fundamental coupling of the semantic and geometric information. In other words, we wish to imbue UGRs with a semantic component, and thus use the semantic information as directly as possible. We will describe the process we developed to achieve this, and then proceed to showcase an implementation of these ideas in an actual system. Finally, we shall verify our ideas in various indoor datasets. There, we show that our intuitions hold true: in excessively plain environments with little semantics, the geometric component of our solution proves robust. In busier scenes, where semantics are more abundant, measurable improvements are reported with our semantic integration.

1.3 Organization

The present document is structured as follows:

- In Chapter 2, we present some of the most relevant research to our work, along with a mathematical overview of SLAM optimization algorithms.
- In Chapter 3, we delve into the details of UGRs and how they tie into spatial perception for SLAM.
- In Chapter 4, we explore the key components of semantic segmentation, how the ongoing deep learning revolution has drastically altered this field, and what the SLAM community has used these technologies for.
- In Chapter 5, our solution is described in-depth, including the guiding design principles, mathematical foundations, implementation details and the results of our experimental evaluation.
- Finally, in Chapter 6 we conclude with a discussion on the above and proposals for future extensions of this work.

Chapter 2

Background and Related Work

This chapter is dedicated to presenting an overview of existing works in related areas of robotic perception. The systems that concern us cover a wide range of theory, starting with various ideas from odometry and extending to modern SLAM concepts. The central mathematical concepts that underpin our discussions mostly fall within the realms of statistical noise modeling, 3D spatial transforms and, finally, non-linear optimization. We will also attempt to give an overview of relevant datasets and their properties, explaining how to arrive at pertinent selection criteria for determining where a SLAM system's performance could be tested.

2.1 Key Terminology

Before we start with the main content of this chapter, let us first introduce some of the basic terminology that will be used henceforth. These terms are either assigned their standard meaning or defined slightly differently, in order to reflect the particular use cases of this work.

A *sensor frame* is a complete measurement coming from a sensor device for one specific moment in time, meaning that the data comes as one unit, with a unique timestamp associated with it.

A *feature* is a catch-all term used throughout this work, referring to an entity that can be detected in a frame, described using some kind of descriptor, and rediscoverable in subsequent frames observing the same physical region, albeit from a different vantage point. There is a wealth of computer vision literature concerning optical feature processes, both for detection and description ([20, 21, 22, 23, 24] to name but a few), but here we will extend the term to include all kinds of features that can serve in the SLAM pipeline, including optical, structured geometric and semantic.

A *LiDAR* sensor commonly uses a mechanically rotating laser beam to sweep a planar sector of the environment, measuring time-of-flight of the reflected beams to compute range estimates from the sensor to obstacles.

An *RGB-D* camera is a sensor that offers both RGB optical images and depth images. The depth image is encoded as a grayscale intensity image, with each pixel corresponding to a depth value. Depth is commonly captured using infrared rays, measuring time-of-flight. Note that RGB and depth data do not come perfectly synchronized from real-world sensors, which is why an association step to find the best matches between the two streams is always necessary to actually create the final frames to be passed on to a visual processing system.

A *spatial transform* describes how two coordinate frames are positioned relative to one another. It is comprised of two components: a translation vector $\in \mathbb{R}^3$ and a rotation $\in$ SO(3) (special orthogonal group of order 3, also known as the 3D rotation group). There are many ways to parameterize a transform's rotation, using mathematical tools such as the angle-axis representation, rotation matrices or quaternions. In the case of a 3D transform, as is typical in robotics, we will consider a transform T from a coordinate frame A to a coordinate frame B as a 4×4 matrix, defined as follows:

$$\boldsymbol{T}_B^A = \left[\begin{array}{c|c} \boldsymbol{R}_B^A & \boldsymbol{d}_B^A \\ \hline \boldsymbol{0}_{1\times 3} & 1 \end{array}\right], \ \boldsymbol{p^A} = \boldsymbol{T}_B^A \cdot \boldsymbol{p}^B, \ \boldsymbol{p} = \left[\begin{array}{ccc|c} p_x & p_y & p_z & 1 \end{array}\right]^T$$

where $\boldsymbol{R}_B^A$ and $\boldsymbol{d}_B^A$ are the rotation and translation from frame A to B respectively. So the transform matrix, when left multiplying the homogeneous coordinates of point $\boldsymbol{p}$ in frame B, will yield the homogeneous coordinates of that point in reference frame A. Note that in the multiplication, rotation is applied *first* to the point's coordinates, and then its coordinates are added to the result. In Fig. 2.1 we see a 2D transform, which is in the same as the 3D version in structure, but is one dimension smaller. The displacement is $[3.5, 2]$ and rotation is 45° counter-clockwise. So the transform in this case would be:

$$T = \left[\begin{array}{cc|c} \cos\theta & -\sin\theta & d_x \\ \sin\theta & \cos\theta & d_y \\ \hline 0 & 0 & 1 \end{array}\right] \simeq \left[\begin{array}{cc|c} 0.707 & -0.707 & 3.5 \\ 0.707 & 0.707 & 2 \\ \hline 0 & 0 & 1 \end{array}\right] \tag{2.1}$$

Transforms can also be chained together, simply by multiplying them in order: $\boldsymbol{T}_C^A = \boldsymbol{T}_B^A \cdot \boldsymbol{T}_C^B$.

2.2 Odometry

When considering the SLAM problem, it is natural to ask whether a system executing *only* localization is possible. After all, mapping adds considerable overhead to the implementation, and heavily constrained on-board CPUs do not have to be burdened with it if it is not essential to the application. Odometry solutions were developed to achieve just that.

Definition 2.2.1 (Odometry)**.** A framework that can estimate the motion of a robot relative to its starting pose, using data from its sensors.

In essence, it computes the transform between two consecutive frames along a sensor's trajectory. This way, the total transform between the origin of the motion and the current pose of the sensor can be estimated as the product of all the intermediate transforms that the system computed, using the chaining mechanism we saw in Sect. 2.1. This

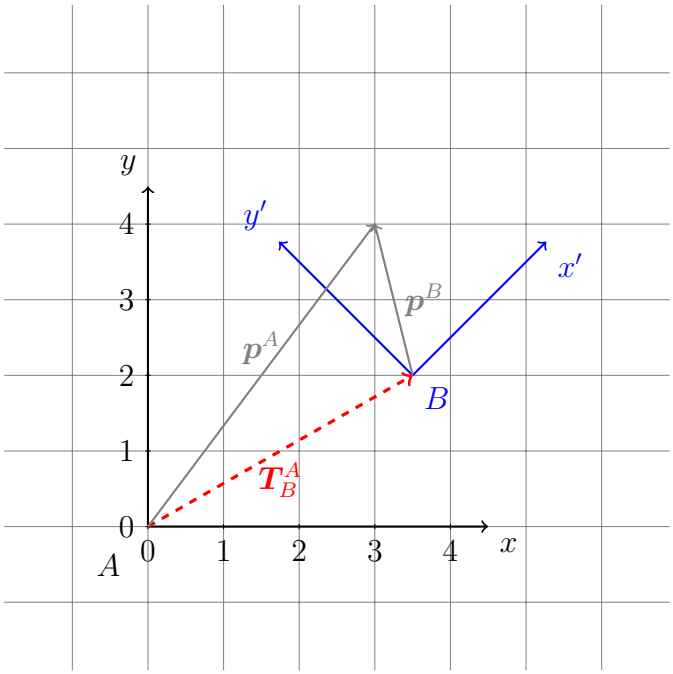


Figure 2.1: The 2D coordinate frame transform from the example in Eq. (2.1).

straightforward approach can function with a variety of sensors. For instance, in wheeled robots it is possible to use encoder hardware in the wheels of the robots to estimate the rotation of the wheels, thus getting an estimate for the total length of the trajectory. Additionally, the rotation of the robot can be computed by the difference in rotations between different wheels. This of course assumes that accurate measurements of wheel radii are available, and that these will not change over time. In addition, the steering mode of the robot needs to be known for estimating its rotations. Some examples of steering modes for four-wheel robots follow (with their visualizations in Fig. 2.2):

- **Ackermann steering:** this is the steering mode used almost universally by automobiles, where only the front wheels turn. Tires in this mode do not need to explicitly slip on the traversed surface when steering, because the inner wheel of the forward axle steers at a different angle than the outer one, so that the line perpendicular to the plane of the wheel, emanating from its central point toward the inside of the turn passes through the center of the turn circle.
- **Active Front and Rear Steering (AFRS):** the front and rear wheels work in tandem, turning in such a way as to achieve independent Ackermann geometries for the forward and rear axles, with the turn center located at the extension of their midline.
- **Spinning or Point Turn:** the front and rear wheels coordinate so that the center

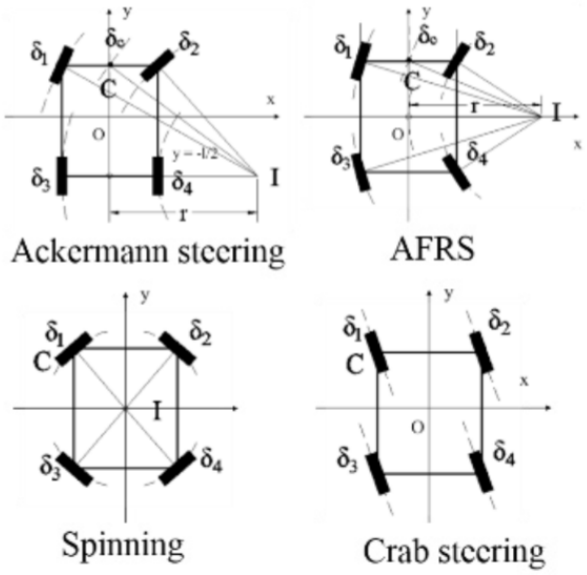


Figure 2.2: Visualizations of steering modes for a four-wheeled robot. [1]

of rotation coincides with the center of the robot's body, so that the final motion is an in-place rotation around the z-axis.

- **Crab steering:** in this mode, all wheels participate in steering, aligning themselves to be parallel to one another.

One thing that wheel odometry fails to account for is wheel slippage. If the terrain is uneven and/or exceptionally slippery, a wheel's rotation could easily stop corresponding to the actual motion of the robot. Another issue is that wheels can have inflatable tires around them, and the effective radius of a wheel with such a flexible padding is not necessarily constant; for instance, a higher platform weight could compress the tires, thus reducing the wheel radius. Wheel odometry also fails to generalize to robots with different locomotion methods, such as legged, aerial or underwater robots. This motivates one to consider using different sensors for odometry purposes, such as cameras, which led to the development of *Visual Odometry (VO)* [25, 26]. With VO, camera data is directly used to estimate frame-to-frame transformations, thus decoupling the estimation process from the locomotion mode of the agent.

2.2.1 Bundle Adjustment

The theoretical issue with VO, and indeed all forms of odometry, is that it is fundamentally unable to self-correct. Even when fusing with other sensors, such as an Inertial Measurement Unit (IMU), small estimation errors will occur, and due to the nature of the process of transform chaining, these errors will inevitably cause the estimate to drift

more and more. In a trajectory that never revisits previously seen locations, this drift is unavoidable, and has been statistically analyzed in [4]. Odometry solutions do in fact offer one possible local remedy for the problem, which derived from *Structure from Motion (SfM)* theory. The intuition behind it is that, instead of treating all frames independently, one can gain considerable accuracy simply by performing a joint optimization of multiple frames consecutive. This essentially defines a temporal window, that allows the system to *adjust* its estimates over a *bundle* of data:

Definition 2.2.2 (Bundle Adjustment (BA))**.** Given a window of images capturing a part of a 3D scene, BA is the process of simultaneously optimizing the geometry of the observed features, the motion of the camera and, optionally, the parameters of the camera's optics.

The success of BA demonstrates that using more data can help derive better estimates for movement; that is, by jointly optimizing over a time window, it is expected that, at least locally, the impact of noisy estimates can be reduced. This is presented in [27], where windowed BA achieves great results in long and challenging real-world traversals. The entire concept of BA highlights how temporally correlated data batches aid the estimation process. Spatial correlations, such as loop-closures, is what SLAM ultimately exploits, to give even better results.

2.2.2 Semantic extensions

Interestingly, works have tried to include semantics (more on this concept in Chap. 4) in VO. Roughly speaking, semantics here indicates that the system understands what *type* of objects exist in the environment (as opposed to just their shape), and then uses that information to improve its motion estimates. In [28], the authors generate medium-term semantic constraints, by getting a pixel-level segmentation of camera frames, which are then used to create additional semantic constraints for the estimation process. They penalize semantic mismatches by using a Gaussian distance transform of variance σ for the semantic label image around each detected region and comparing the estimated re-projection against what they receive in following measurements:

$$\mathbb{P}[S_k|T_k, X_i, Z_i = c] \propto e^{-\frac{1}{2\sigma^2} DT_k^{(c)}(\pi(T_k, X_i))^2} \tag{2.2}$$

$$\begin{aligned} e_{sem}(k,i) &= \sum_{c \in \mathcal{C}} w_i^{(c)} \log(\mathbb{P}[S_k|T_k, X_i, Z_i = c]) \\ &= -\sum_{c \in \mathcal{C}} w_i^{(c)} \frac{1}{2\sigma^2} DT_k^{(c)}(\pi(T_k, X_i))^2 \end{aligned} \tag{2.3}$$

These equations describe how the expected semantic measurement S_k from the k-th pose T_k, associated with point X_i and actual semantic measurement Z_i that returned classification c, and projected onto the image through the projection function π, impacts

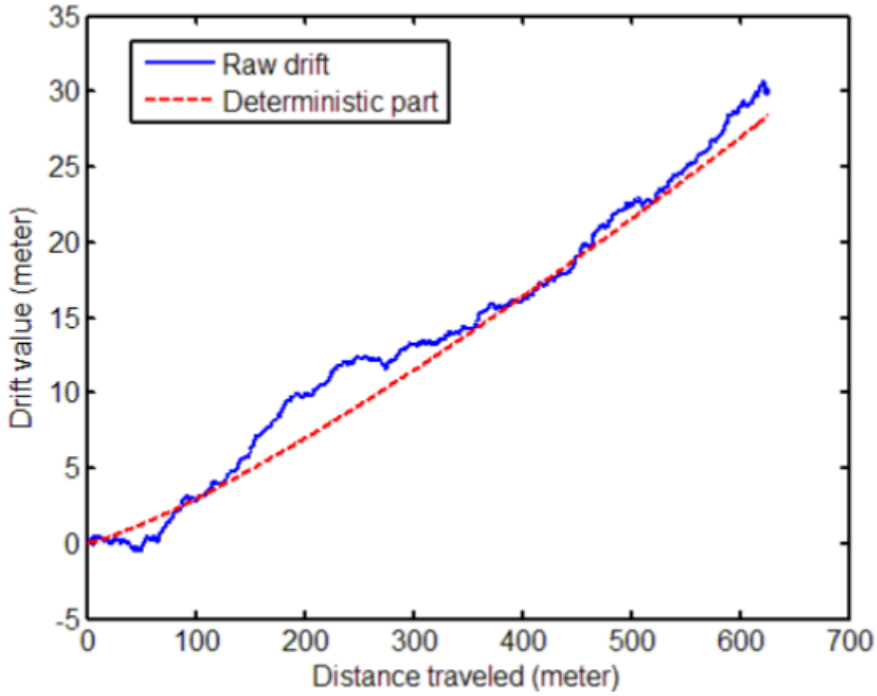


Figure 2.3: An example of VO drift as a function of distance traveled. [4]

neighboring areas through the distance transform $DT_k^{(c)}$. The pixels further away from the detection boundary of semantic measurement S_k belong to class c with a Gaussian distribution, and summing the logarithms of these yields the error term. The weight $w_i^{(c)}$ is simply the probability that X_i belongs to class c, in other words the confidence of the classification.

In contrast to the above, the authors of [29] take a different approach, in which they track motion using a purely semantic error component, in conjunction to both photometric and geometric components. They accomplish this by fine-tuning a deep neural network to perform dense segmentation of the image, and then optimize based on semantic constraints, which make use of purely semantic Jacobians for the optimization process (more on Jacobians later), that deviate from photometric standards, using the class prediction vector to guide the optimizer along the semantic error dimensions. The optimization tri-objective is:

$$\arg\min_{\boldsymbol{x}} \ \lambda_{\boldsymbol{S}}^2 \ \|e_{\boldsymbol{S}}\|^2 + \lambda_{\boldsymbol{I}}^2 \ \|e_{\boldsymbol{I}}\|^2 + \ \|e_{\boldsymbol{G}}\|^2 \tag{2.4}$$

where the λ terms are used to scale the error contributions and $e_{\boldsymbol{S}}$, $e_{\boldsymbol{I}}$ and $e_{\boldsymbol{G}}$ correspond to the semantic, photometric and geometric error terms respectively.

2.3 Initial SLAM Approaches

The SLAM problem was initially approached by means of probabilistic filtering, which we will discuss next. The overarching theme in the various mathematical formulations that have been proposed is the following: the agent is in a state that can be described using a state-vector $\boldsymbol{x}$, its sensors report measurements $\boldsymbol{z}$ and the actuators are fed control signals $\boldsymbol{u}$. All the above quantities are sampled at discrete time instances, and all are affected by some type of noise, which has commonly been assumed Gaussian. Finally, Markov assumptions are frequently made in these settings, as they massively ease both the theoretical derivation of the processes, as well as their practical implementations, without compromising their validity or accuracy.

2.3.1 Belief distributions and Bayesian filtering

The initial approaches explored by SLAM researchers attempted to address the following general problem: given a sequence of sensor measurement data $\boldsymbol{z}_{1:t}$ and control signals $\boldsymbol{u}_{1:t}$ of a robot, determine the probability distribution of its current pose $\boldsymbol{x}_t$. Following the convention of [2], let $bel(\boldsymbol{x}_t)$ be the belief of the agent that it is in state $\boldsymbol{x}_t$. We then have:

$$\text{bel}(\boldsymbol{x}_t) := \mathbb{P}[\boldsymbol{x}_t|\boldsymbol{z}_{1:t}, \boldsymbol{u}_{1:t}], \ \overline{\text{bel}}(\boldsymbol{x}_t) := \mathbb{P}[\boldsymbol{x}_t|\boldsymbol{z}_{1:t-1}, \boldsymbol{u}_{1:t}] \tag{2.5}$$

with the second definition being a shorthand for the posterior probability excluding the current measurement. Based on these definitions, the *Bayesian filter* for state estimation is defined as follows:

Algorithm 3: Bayesian Filtering Algorithm [2]

Input: $\text{bel}(\boldsymbol{x}_{t-1}, \boldsymbol{u}_{1:t}, \boldsymbol{z}_{1:t})$
Output: $\text{bel}(\boldsymbol{x}_t)$

forall $\boldsymbol{x}_t$ **do**
  $\overline{\text{bel}}(\boldsymbol{x}_t) = \int \mathbb{P}[\boldsymbol{x}_t|\boldsymbol{z}_{1:t-1}, \boldsymbol{u}_{1:t}]\text{bel}(\boldsymbol{x}_{t-1})dx_{t-1}$
  $\text{bel}(\boldsymbol{x}_t) = \eta \cdot \mathbb{P}[\boldsymbol{z}_t|\boldsymbol{x}_t]\overline{\text{bel}}(\boldsymbol{x}_t)$
end
return $bel(\boldsymbol{x}_t)$

We see that Alg. 3 computes belief recursively. Internally, the algorithm first computes a distribution without using $\boldsymbol{z}_t$ and then weighs and normalizes this distribution against the probability of measurement $\boldsymbol{z}_t$ being received given the target state.

2.3.2 Kalman filters

Basic Kalman Filter (KF)

Concrete implementations of Bayesian filtering make various assumptions in order to be efficient, without sacrificing accuracy or completeness. The KF family is a common class of recursive state estimators that have found many applications in SLAM. The basic KF is derived under the assumption that the posterior probability distributions involved in Bayesian filtering are linear with respect to their arguments and have only Gaussian additive noise applied to them. Although the derivation is somewhat involved, the final algorithm still follows a state update logic echoing the structure of Bayesian filtering.

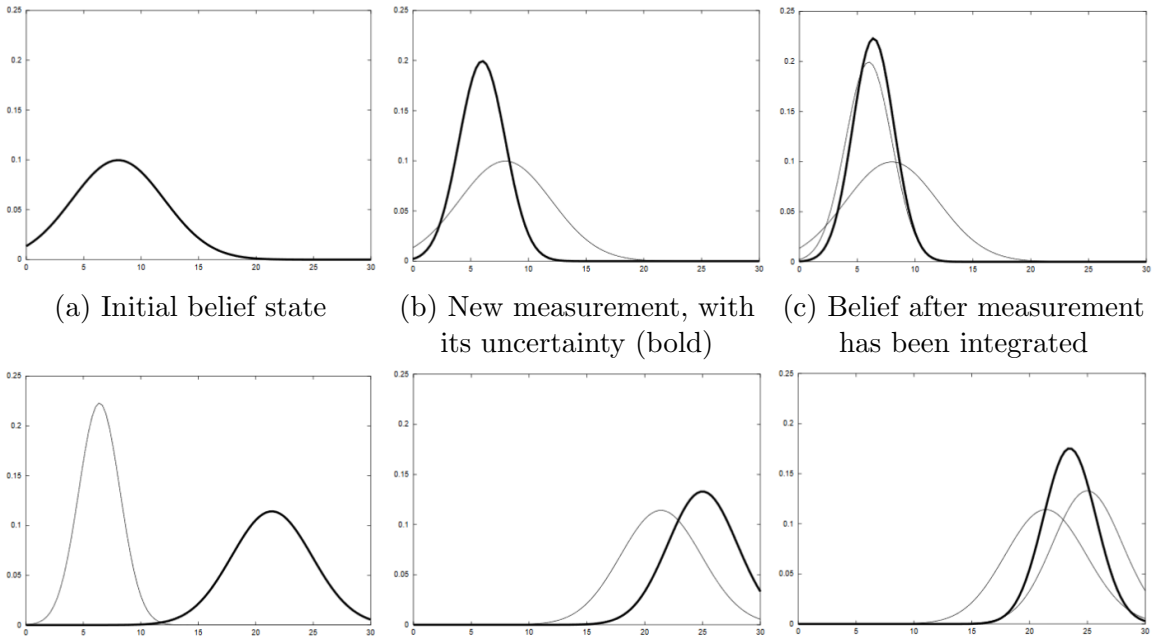
Algorithm 4: KF Algorithm [2]

Input: $\boldsymbol{\mu}_{t-1}, \boldsymbol{\Sigma}_{t-1}, \boldsymbol{u}_t, \boldsymbol{z}_t$
Output: $\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t$

$\overline{\boldsymbol{\mu}}_t = \boldsymbol{A}_t\boldsymbol{\mu}_t + \boldsymbol{B}_t\boldsymbol{u}_t$
$\overline{\boldsymbol{\Sigma}}_t = \boldsymbol{A}_t\boldsymbol{\Sigma}_{t-1}\boldsymbol{A}_t^T + \boldsymbol{R}_t$

$\boldsymbol{K}_t = \overline{\boldsymbol{\Sigma}}_t\boldsymbol{C}_t(\boldsymbol{C}_t\overline{\boldsymbol{\Sigma}}_t\boldsymbol{C}_t^T + \boldsymbol{Q}_t)^{-1}$
$\boldsymbol{\mu}_t = \overline{\boldsymbol{\mu}}_t + \boldsymbol{K}_t(\boldsymbol{z}_t - \boldsymbol{C}_t\overline{\boldsymbol{\mu}}_t)$
$\boldsymbol{\Sigma}_t = (\mathbb{I} - \boldsymbol{K}_t\boldsymbol{C}_t)\overline{\boldsymbol{\Sigma}}_t$
return $\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t$

Matrices $\boldsymbol{A}_t, \boldsymbol{B}_t, \boldsymbol{C}_t$ describe the system's *linear* state transition characteristics, while $\boldsymbol{Q}_t, \boldsymbol{R}_t$ are Gaussian noise covariance matrices. $\boldsymbol{K}_t$ is usually known as the *Kalman gain* (see [2] for the full mathematical derivation). The effect of a KF in state estimation can be seen for a simple case in Fig. 2.4, where a single state variable is updated.



(a) Initial belief state

(b) New measurement, with its uncertainty (bold)

(c) Belief after measurement has been integrated

(d) Belief after a motion, which re-introduces uncertainty

(e) A new measurement with its own uncertainty

(f) Belief after the new measurement is integrated

Figure 2.4: Intermediate steps of a KF, visualized for a single state variable. Darker curves indicate new belief distributions, whereas fainter curves indicate older ones. At each step, a combination is performed, and the estimate is updated according the KF equations. [2]

KF variants

Unsurprisingly, the Gaussian assumptions on which KF development was founded do not always hold in practice. In an attempt to better adapt KFs to a non-Gaussian reality, the *Extended Kalman Filter (EKF)* was devised. The primary difference between a KF and an EKF is that the state transition equations are replaced by a more accurate, *non-linear* model. The EKF takes this model into consideration by linearizing around the state average, using a Taylor expansion. This means that, instead of transferring a Gaussian state through a fixed linear model, the EKF in fact calculates a new locally linearized model on-the-fly, for each point in the iteration process. Given that the model will be processing Gaussian state estimates, the above linearization has a drawback. Computing

(a) Initial distribution. (b) Multimodality arises.

(c) Multimodality resolved.

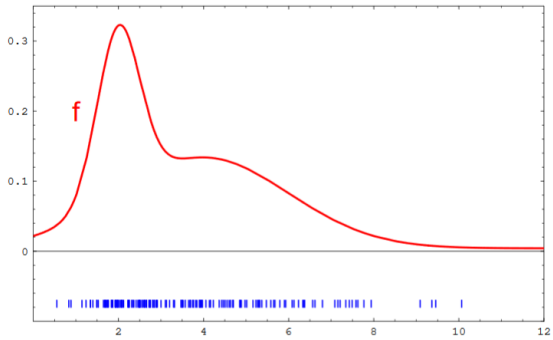
Figure 2.5: Particle filter iterations. Initially, the robot is highly uncertain of its whereabouts. Then, two highly likely locations are determined based on observations. Moving to collect more data, the robot finally localizes itself. [5]

the Jacobian at one point of the model function means that we ignore information of the area around the point of linearization. Since it is an entire Gaussian distribution that is in fact processed by our model, this deviation can actually reduce our accuracy. The *Unscented Kalman Filter (UKF)* stochastically selects a number of points to move through the model function, and from the outputs it reconstructs a more accurate estimation of the model's local behavior, taking a broader range of the model function into account.

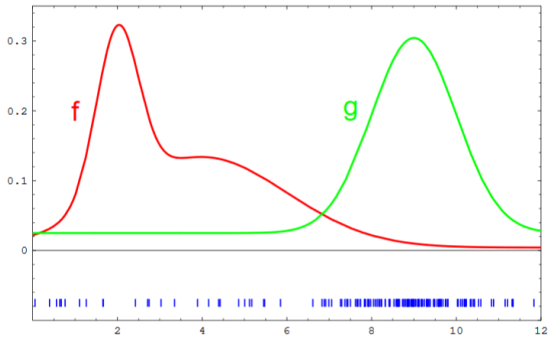
2.3.3 Non-parametric filtering and the Particle Filter (PF)

The filters that we have seen so far are all based on some form of Gaussian assumption. A different approach to filtering in general is non-parametric filtering. For Bayesian filters of this type, the posterior probability space is split, and then the probability mass is

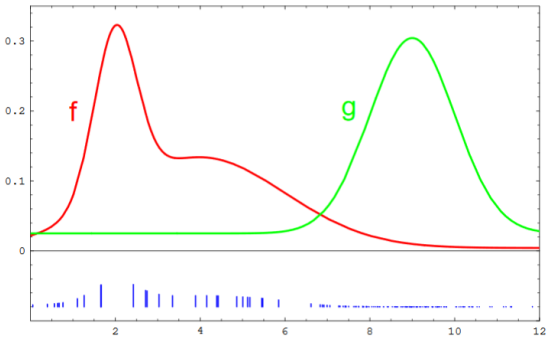
approximated by the local cumulative distribution. See the Histogram Filter [30] for a concrete implementation of these ideas. PFs instead generate samples for the posterior probability space and extract a discrete approximation by selecting particles to advance to the next iteration by a Monte Carlo process [31, 5]. FastSLAM 2.0 was a concrete implementation of this idea [32].



(a) Target posterior probability density f.



(b) Sampling based on a different distribution g.



(c) Samples are given weight $f(x)/g(x)$ and are selected accordingly (represented with blue dashes below the density graphs).

Figure 2.6: Importance factor sampling in PFs. In implementations, $f(x) := \text{bel}(x_t)$, $g(x) := \overline{\text{bel}}(x_t)$. [5]

While it is obvious that such a data-driven approach lends flexibility, it is instructive to consider why abandoning Gaussian models is desirable in the first place. One aspect of SLAM that KFs do not have any general way of dealing with is multimodality. Multimodality arises when a distribution has more than one salient local maxima, e.g. in localization distributions, when a robot visits an area that appears indistinguishable from another (at least within its sensors' error margins). This is a strongly non-Gaussian property, which cannot be addressed purely by mathematical modifications on the Kalman paradigm, which is part of the reason why PFs are viewed as an effective alternative.

Internally, PFs require the specification of a *sampling function*. This function is used to obtain samples from the general population of particles in the PF, so as to ultimately get a discrete approximation of the posterior $bel(\boldsymbol{x}_t)$. Note that, since we are directly sampling from the posterior, we do not constrain it to be a Gaussian distribution. This process in known as importance sampling, illustrated in Fig. 2.6.

2.4 Graph-based SLAM

All the filtering approaches that we have seen have an inherent limitation, in that they approach SLAM in isolated increments, under the Markov assumption. This fails to address the complete SLAM problem, as the probabilistic approximations made are only dependent on current measurements and controls. In fact, the full SLAM problem is a maximization over the *entire* sequence of poses, all considered at once. Graph-based SLAM offers a way to convert SLAM into a non-linear least squares optimization problem over a sparse graph, which contains the complete history of the robot's trajectory.

2.4.1 The pose graph

The central entity of any graph-based SLAM method is the pose graph [3]. This is the structure used to encode all the information necessary for an optimization framework to work with all the observations and pose estimates at once, aiming to converge to a globally optimal solution to the SLAM problem.

Definition 2.4.1 (Pose Graph)**.** A *pose graph* is a sparse representation of a robot's trajectory using nodes and edges. The nodes of a pose graph are split into two types: pose nodes and observation nodes, while edges represent transforms between two nodes they connect. An edge connecting two pose nodes corresponds to a step in the robot's trajectory, whereas an edge between a pose node and an observation node encodes the transform of the observed feature in the pose's reference frame.

In 2.7 we see a sample pose graph, with poses x subject to controls u, registering observations z of an unknown map m. A generalization of the above schema, which has been successfully used in [9, 51] among others, is the factor graph. A factor graph replaces edges with factors, which connect multiple nodes at once, changing the mathematical details of the optimization (see the GTSAM library [64] for a sample implementation).

The constraints imposed by the edges attempt to represent the geometry of the space around the robot, but noise will inevitably render the motion estimations inconsistent. To provide a simple example, one feature may indicate that the sensor's frame has been translated along the x-axis by 5cm, but another may instead suggest 6cm. To make matters worse, tracking features across consecutive frames is not trivial; false matches come up frequently, subject to descriptor efficacy, and can corrupt the process.

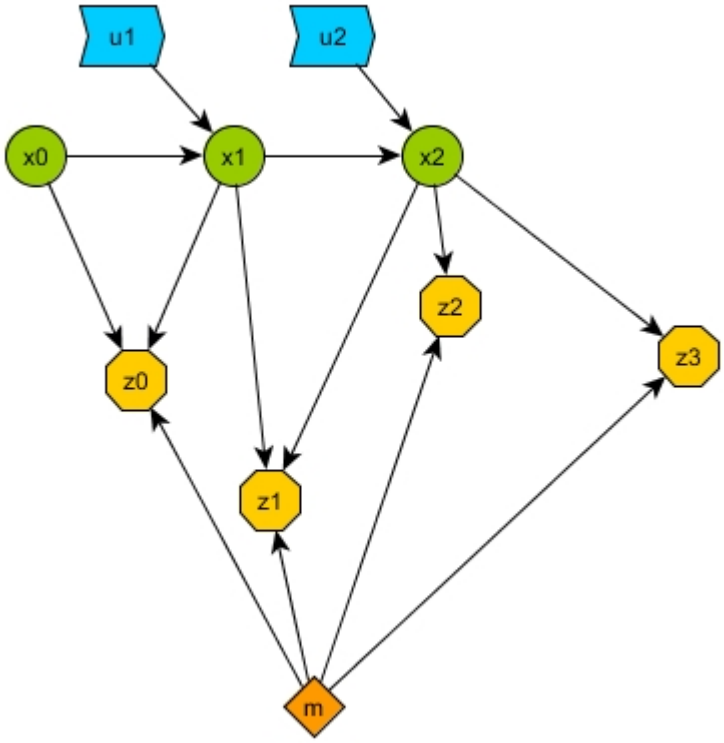


Figure 2.7: A visualization of how a pose graph is structured. The robot is controlled via signals u_i, and transitions between poses x_i. From each pose, it collects noisy measurements z_j of the unknown environment map m. The goal is to simultaneously approximate x_i and m.

General methods for optimizing graphs of this type have already been developed, such as [65], but our work is based on [33], which is also used by the standard ORB-SLAM2 [12]. The optimization process is generally compartmentalized in the **back-end** of the system. By contrast, the **front-end** is responsible for constructing a *meaningful* graph for the back-end to optimize. Typical operations therefore include:

- Feature correspondence: a feature detected across multiple frames should be identified as being the same physical structure. This is a crucial step, because without it all features would be considered new, making it impossible for the agent to localize, as every scene would appear disconnected from the rest.

- Outlier removal: this is essentially a way to counter the inevitable errors committed by the previous step. Each feature correspondence proposal implicitly suggests a transform that would justify it. The effects of noise of course eliminate any possibility for complete congruence among these suggestions, but correspondence errors typically fall way out of the noise's standard deviation. Therefore, using an outlier

detection method such as RANSAC [61, 62] can help cull such errors before they impact our motion estimations.

- Pose-graph maintenance: adding new nodes and edges (or factors), while also merging those that have correspondences. It is also possible to use some method to perform explicit loop-closure detection, although the choice of the similarity metric is a major design concern.
- Initial motion estimate: the result of the first two steps, it is just an initial guess of what motion was executed (akin to odometry).

2.4.2 Optimization

Setup

The graph optimization problem, as outlined, for example, in [3, 7, 33], is solved iteratively, using algorithms that aim to converge to a solution minimizing an error metric. As seen in Fig. 2.8, the robotic agent collects constraints $\boldsymbol{z}_{ij}$ by observing the world, namely from state $\boldsymbol{x}_i$ a feature is observed, and subsequently re-observed from state $\boldsymbol{x}_j$. This generates a constraint between the two, corresponding to the motion, and is then compared to the *expected* measurement $\hat{\boldsymbol{z}}_{ij}$ to define the error term. The measurement is of course impacted by noise, modeled here with a Gaussian distribution. Instead of parameterizing the Gaussian using the mean $\boldsymbol{\mu}_{ij}$ and covariance $\boldsymbol{\Sigma}_{ij}$, we use the canonical representation, which is based on the information matrix $\boldsymbol{\Omega}_{ij} := \boldsymbol{\Sigma}_{ij}^{-1}$ and the information vector $\boldsymbol{\xi}_{ij} := \boldsymbol{\Sigma}_{ij}^{-1}\boldsymbol{\mu}_{ij}$ ($\boldsymbol{z}_{ij}$ would then be the mean of this random variable). Thus, the error is defined to be: $\boldsymbol{e}_{ij}(\boldsymbol{x}_i, \boldsymbol{x}_j) := \boldsymbol{z}_{ij} - \hat{\boldsymbol{z}}_{ij}$, and we have the following expression for the optimization objective:

$$\boldsymbol{X}^* = \arg\min_{\boldsymbol{X}} \sum_{(i,j)\in\mathcal{C}} f_{ij} = \arg\min_{\boldsymbol{X}} \sum_{(i,j)\in\mathcal{C}} \boldsymbol{e}_{ij}^T \boldsymbol{\Omega}_{ij} \boldsymbol{e}_{ij} \tag{2.6}$$

where $\mathcal{C}$ is the set of indices for which there exists a constraint, f_{ij} is a shorthand for the corresponding quadratic summand and $\boldsymbol{X}$ is the vector of all states under optimization.

Approach in Euclidean spaces

Having defined the graph and the optimization objective, we now need a process for actually performing the optimization. This is where non-linear least-squares optimization routines become important. The key idea is to solve numerically, repeating certain steps iteratively until convergence. To do so, we first need to find a derivative (Jacobian) of the current solution estimate's error function. Then, we need to move our estimation in the direction that the Jacobian tells us will decrease the error. Through iterating, we should arrive closer to an optimum point, where the error is minimized.

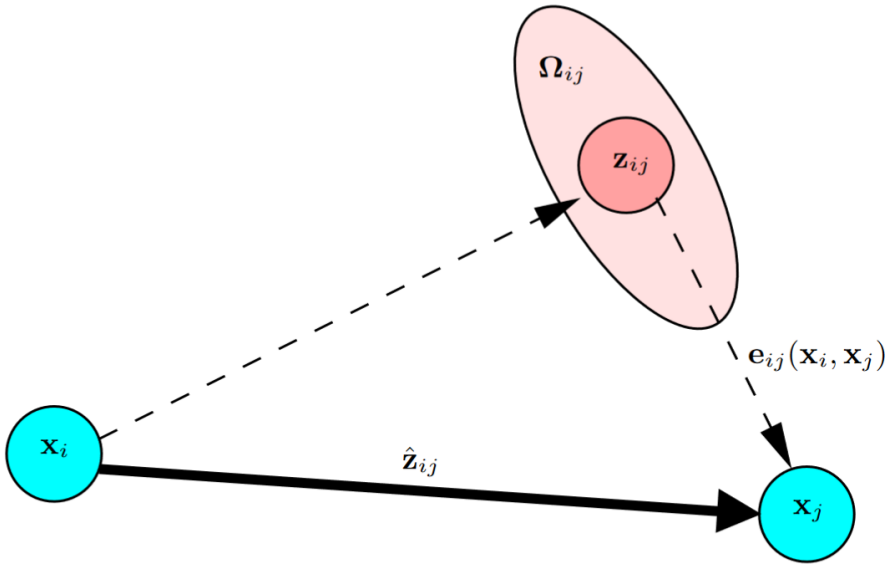


Figure 2.8: A noisy measurement, modeled with Gaussian uncertainty. [3]

Firstly, let us consider what happens to the error function $\boldsymbol{e}_{ij}$ of a single graph edge, when a small perturbation is applied to its arguments, initially at a starting guess of $\breve{\boldsymbol{x}}$:

$$\boldsymbol{e}_{ij}(\breve{\boldsymbol{x}} + \Delta\boldsymbol{x}) \simeq \boldsymbol{e}_{ij} + \boldsymbol{J}_{ij}\Delta\boldsymbol{x} \tag{2.7}$$

where $\boldsymbol{J}_{ij}$ is the error Jacobian of the error function computed at $\breve{\boldsymbol{x}}$. This linearization allows us to rewrite (2.6)'s terms as follows:

$$\begin{aligned} f_{ij}(\breve{\boldsymbol{x}} + \Delta\boldsymbol{x}) &\simeq (\boldsymbol{e}_{ij} + \boldsymbol{J}_{ij}\Delta\boldsymbol{x})^T\boldsymbol{\Omega}_{ij}(\boldsymbol{e}_{ij} + \boldsymbol{J}_{ij}\Delta\boldsymbol{x}) \\ &= \underbrace{\boldsymbol{e}_{ij}^T\boldsymbol{\Omega}_{ij}\boldsymbol{e}_{ij}}_{c_{ij}} + 2\underbrace{\boldsymbol{e}_{ij}^T\boldsymbol{\Omega}_{ij}\boldsymbol{J}_{ij}}_{\boldsymbol{b}_{ij}}\Delta\boldsymbol{x} + \Delta\boldsymbol{x}^T\underbrace{\boldsymbol{J}_{ij}^T\boldsymbol{\Omega}_{ij}\boldsymbol{J}_{ij}}_{\boldsymbol{H}_{ij}}\Delta\boldsymbol{x} \\ &= c_{ij} + 2\boldsymbol{b}_{ij}\Delta\boldsymbol{x} + \Delta\boldsymbol{x}^T\boldsymbol{H}_{ij}\Delta\boldsymbol{x} \end{aligned} \tag{2.8}$$

We can see that (2.8) is inherently an approximation, based on (2.7). Now we can rewrite (2.6)'s objective function $f(\boldsymbol{x}) := \sum_{(i,j)\in\mathcal{C}} f_{ij}$ to reflect this approximation by direct substitution:

$$\begin{aligned} f(\breve{\boldsymbol{x}} + \Delta\boldsymbol{x}) = \sum_{(i,j)\in\mathcal{C}} f_{ij}(\breve{\boldsymbol{x}} + \Delta\boldsymbol{x}) &= \sum_{(i,j)\in\mathcal{C}} c_{ij} + 2\boldsymbol{b}_{ij}\Delta\boldsymbol{x} + \Delta\boldsymbol{x}^T\boldsymbol{H}_{ij}\Delta\boldsymbol{x} \\ &= c + 2\boldsymbol{b}\Delta\boldsymbol{x} + \Delta\boldsymbol{x}^T\boldsymbol{H}\Delta\boldsymbol{x} \end{aligned} \tag{2.9}$$

where in the last step we define the new variables as the sum of the indexed terms in the summation. This approximation can indeed be directly minimized; the value of $\Delta\boldsymbol{x}^*$ that

achieves the minimum is the solution to the following linear system:

$$\boldsymbol{H}\Delta\boldsymbol{x}^* = -\boldsymbol{b} \tag{2.10}$$

After solving (2.10), one just needs to add the result to the existing solution approximation to complete a single iteration of the process. To solve this linear system efficiently, it is imperative to take the sparse nature of matrix $\boldsymbol{H}$ into consideration. Sparsity is guaranteed by the construction of $\boldsymbol{H}$, because it is a matrix built from Jacobians of edges in the pose graph. This means that the entries will be all zero, except for those entries corresponding to nodes i and j, since all other dimensions of the state do not directly affect the term in question. Therefore, using methods such as sparse Cholesky factorization can efficiently solve the system.

Approach in manifolds

In all the above, we have been using vector addition for state perturbations, without taking the unique, non-euclidean structure of their SO(3) components into account. This means that we are fundamentally ignoring the underlying mathematical structure. To address this, let us consider a space that is only *locally* euclidean, but not necessarily *globally* euclidean. The mathematically curious reader is encouraged to read [66] for further details on definitions. However, before we proceed, let us first review some basic facts and definitions about the related topic of Lie groups and Lie algebras.

Definition 2.4.2 (Lie group)**.** A *Lie group* is a group that is also a differentiable manifold.

Every Lie group has an associated Lie algebra linked to it. A Lie algebra must always have the following properties:

Definition 2.4.3 (Lie algebra)**.** A *Lie algebra* is a vector space $\mathfrak{g}$ over a field F, equipped with a binary operator: $[\cdot,\cdot] : \mathfrak{g} \times \mathfrak{g} \to \mathfrak{g}$, known as the Lie bracket. The algebra must satisfy the following:

- **bilinearity**, which stipulates that, $\forall m_1, m_2 \in F,\ x, y, z \in \mathfrak{g}$:

$$\begin{aligned}[m_1 x + m_2 y, z] &= m_1[x, z] + m_2[y, z],\\ [z, m_1 x + m_2 y] &= m_1[z, x] + m_2[z, y]\end{aligned}$$

- **alternativity**, meaning that $\forall x \in \mathfrak{g}$:

$$[x, x] = 0$$

- **Jacobi identity**, that is, $\forall x, y, z \in \mathfrak{g}$:

$$[x, [y, z]] + [z, [x, y]] + [y, [z, x]] = 0$$

A few very important examples of the above definitions involve groups we have already encountered. For instance, the 3D rotations group, SO(3), is in fact a Lie group, with the associated Lie algebra $\mathfrak{so}(3)$. Transforms also form a Lie group, known as SE(3), with the Lie algebra denoted by $\mathfrak{se}(3)$. In essence, the Lie algebra corresponds to the *tangent space* of the Lie group, that is, the vector space obtained by differentiating the group at the specified element, along predefined dimensions which form the basis of the algebra. For instance, in the case of $\mathfrak{so}(3)$, the basis is formed by the following three generators:

$$G_1 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix}, \; G_2 = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ -1 & 0 & 0 \end{bmatrix}, \; G_3 = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{2.11}$$

Comparing to what we see in Sect. 3.2.2, these generators are essentially the building blocks to skew-symmetric matrix logic, as applied for the $\times$ operator. The Lie algebra built on the skew-symmetric matrices is what corresponds to the Lie group SO(3) precisely because skew-symmetric matrices are the derivatives of rotation matrices along the primary axes of rotation.

Now, let us define an operator $\boxplus$ to abstract away from the actual mathematical operations and treat local and global operations in a uniform manner. What the operator does is first to convert the operand to an intermediate euclidean representation, perform an elementary addition in this euclidean space, and finally convert the result back to the manifold. In other words, it hides the transitions into and out of the local Lie algebra. In particular, assuming a local homeomorphic chart ϕ_x exists around a point x in our manifold, the $\boxplus$ operator will combine x with a local perturbation element δ as follows:

$$x \boxplus \delta := \phi_x^{-1}(\phi_x(x) + \delta) \tag{2.12}$$

As a special example, let us revisit SO(3), and explicitly formulate the maps that translate into and out of the manifold (commonly referred to as *exponential* and *logarithmic* maps), for the normalized quaternion representation of 3D rotations, that is: $q := (w, u)$, where: $w \in \mathbb{R}$, $u \in \mathbb{R}^3$ and $w^2 + \|u\|^2 = 1$. We define the mappings as follows:

$$\begin{aligned} &\phi(q) := \log(q), \; \log : \mathrm{SO}(3) \to \mathbb{R}^3 \\ &q \mapsto \begin{cases} \dfrac{u}{\|u\|} \arccos w, & w \neq 0 \\ 0, & w = 0 \end{cases} \end{aligned} \tag{2.13}$$

$$\begin{aligned} &\phi^{-1}(\delta) = \exp(\delta), \; \exp : \mathbb{R}^3 \to \mathrm{SO}(3) \\ &\delta \mapsto (\cos \|\delta\|, \delta \cdot \operatorname{sinc} \|\delta\|) \end{aligned} \tag{2.14}$$

Therefore, the tools for converting from manifold to local tangent spaces are analytically known. In the more general case of interest for SLAM, which is SE(3), $\boldsymbol{x}_1 \boxplus \boldsymbol{x}_2$ will first

apply $\boldsymbol{x}_2$'s rotation to $\boldsymbol{x}_1$ and then shift by the translation, and it will do so using similar logarithmic and exponential maps, to move from and back into the manifold respectively (see Sect. 1.5.2 in [66]).

The mathematical consequences for our optimization problem will now be studied. For starters, we rewrite the error term:

$$\boldsymbol{e}_{ij}(\breve{\boldsymbol{x}} \boxplus \Delta\widetilde{\boldsymbol{x}}) \simeq \breve{\boldsymbol{e}}_{ij} + \widetilde{\boldsymbol{J}}_{ij}\Delta\widetilde{\boldsymbol{x}} \tag{2.15}$$

where $\boldsymbol{e}_{ij}$ is the error term to be computed, $\breve{x}$ is the initial data estimate, and $\Delta\widetilde{x}$ is the perturbation. The above formula implicitly delegates the manifold operations to the computation of a new Jacobian $\widetilde{\boldsymbol{J}}_{ij}$. What we need to do is look at the the process of taking the derivative and, crucially, notice that the *chain rule* of calculus needs to be applied to manifold addition:

$$\begin{aligned}\widetilde{\boldsymbol{J}}_{ij} &= \left.\frac{\partial \boldsymbol{e}_{ij}(\breve{\boldsymbol{x}} \boxplus \Delta\widetilde{\boldsymbol{x}})}{\partial \Delta\widetilde{\boldsymbol{x}}}\right|_{\Delta\widetilde{\boldsymbol{x}}=\boldsymbol{0}} \\ &= \begin{bmatrix} \boldsymbol{0} & \dots & \left.\frac{\partial \boldsymbol{e}_{ij}(\breve{\boldsymbol{x}} \boxplus \Delta\widetilde{\boldsymbol{x}})}{\partial \Delta\widetilde{\boldsymbol{x}}_i}\right|_{\Delta\widetilde{\boldsymbol{x}}=\boldsymbol{0}} & \dots & \left.\frac{\partial \boldsymbol{e}_{ij}(\breve{\boldsymbol{x}} \boxplus \Delta\widetilde{\boldsymbol{x}})}{\partial \Delta\widetilde{\boldsymbol{x}}_j}\right|_{\Delta\widetilde{\boldsymbol{x}}=\boldsymbol{0}} & \dots & \boldsymbol{0} \end{bmatrix}\end{aligned} \tag{2.16}$$

Now the sparsity of the Jacobian construct, which we mentioned earlier, becomes clear, and we see that the non-zero terms are just the differentiations with respect to the directly involved i and j components. The chain rule is applied as follows:

$$\left.\frac{\partial \boldsymbol{e}_{ij}(\breve{\boldsymbol{x}} \boxplus \Delta\widetilde{\boldsymbol{x}})}{\partial \Delta\widetilde{\boldsymbol{x}}_i}\right|_{\Delta\widetilde{\boldsymbol{x}}=0} = \underbrace{\frac{\partial \boldsymbol{e}_{ij}(\breve{\boldsymbol{x}})}{\partial \breve{\boldsymbol{x}}_i}}_{\boldsymbol{A}_{ij}} \cdot \underbrace{\left.\frac{\partial \breve{\boldsymbol{x}}_i \boxplus \Delta\widetilde{\boldsymbol{x}}_i}{\partial \Delta\widetilde{\boldsymbol{x}}_i}\right|_{\Delta\widetilde{\boldsymbol{x}}=\boldsymbol{0}}}_{\boldsymbol{M}_i} \tag{2.17}$$

$$\left.\frac{\partial \boldsymbol{e}_{ij}(\breve{\boldsymbol{x}} \boxplus \Delta\widetilde{\boldsymbol{x}})}{\partial \Delta\widetilde{\boldsymbol{x}}_j}\right|_{\Delta\widetilde{\boldsymbol{x}}=0} = \underbrace{\frac{\partial \boldsymbol{e}_{ij}(\breve{\boldsymbol{x}})}{\partial \breve{\boldsymbol{x}}_j}}_{\boldsymbol{A}_{ij}} \cdot \underbrace{\left.\frac{\partial \breve{\boldsymbol{x}}_j \boxplus \Delta\widetilde{\boldsymbol{x}}_j}{\partial \Delta\widetilde{\boldsymbol{x}}_j}\right|_{\Delta\widetilde{\boldsymbol{x}}=\boldsymbol{0}}}_{\boldsymbol{M}_j} \tag{2.18}$$

In the above notice how $\boldsymbol{A}_{ij}$ is in fact completely disentangled from anything manifold-related. This means that we have successfully contained the manifold operator to only one direct application, before differentiating the result. This differentiation can be done analytically given the operator's formula (the logarithmic/exponential maps would provide that), so the problem is solved.

2.5 SLAM Resources

In this section, we wish to present various available datasets that exist in the bibliography, as well as several VO and SLAM solutions. We will restrict our datasets to

Dataset	Type	Calib.	Pose	Motion	Sem.	Size
ICL-NUIM [11]	synthetic	yes	yes	adequate	none	limited
InteriorNet [34]	synthetic	yes	yes	good	SBB	massive
TUM RGB-D [35]	real-world	yes	yes	good	none	sufficient
CoRBS [36]	real-world	yes	yes	limited	none	limited
AVD [37]	real-world	no	limited	insufficient	SBB	large
MS 7-scenes [38, 39]	synthetic	no	yes	good	none	limited
NYU v2 [40]	real-world	yes	no	good	dense	large
RGB-D Scenes v2 [41]	real-world	no	yes	adequate	limited	limited
Coll. SLAM DS [42]	real-world	limited	yes	good	none	limited

Table 2.1: Various datasets and their properties. Sequences from the datasets in **bold** were used in our experiments, see Chap. 5 for full details.

indoor spaces, and using RGB-D sensors, since these are the only ones that were relevant in our search. However, for the systems we allow arbitrary sensor specifications, in order to illustrate the available wealth of solutions in the bibliography.

2.5.1 RGB-D datasets

When examining an RGB-D dataset, several concerns become extremely important in assessing its usability. Technical concerns include availability of information such as calibration parameters, pose ground-truth etc. In addition to those, we need the actual trajectory executed by the sensor to be reasonably varied, including translations and rotations, and without irrationally sudden changes in direction. Especially for the case of real-world datasets, we need to be prepared for performance drops due to the presence of noise and motion blur. Finally, for our specific system idea, we explored datasets that additionally offered semantic ground-truth annotations, which can be pixel-level (dense) or in the format of *Semantic Bounding Boxes (SBBs)*. We discuss semantics in more detail in Chap. 4.

Tab. 2.1 is by no means an exhaustive enumeration of all datasets available in the SLAM literature, but it already highlights some interesting obstacles in the way of finding good datasets in this area, such as missing calibration parameters. Another complication is the cost of integrating the sequences in the datasets with the processing pipeline of the target SLAM system. This is not a trivial task, so much so that other strong candidate datasets (specifically, ScanNet [67] and Robot at Home [68]) were disregarded for such purely technical considerations and time constraints of the project.

The above table was put together to report on all the aspects of a dataset that would affect our tests. The "Type" column describes how the datasets were captured; synthetic datasets that we considered were all photorealistically rendered indoor spaces, whereas

the real-world datasets were captured by cameras inside various rooms/buildings. Next, the "Calibration" column indicates whether or not calibration parameters for the camera apparatus are available. We need them to be available, since otherwise most VO/vSLAM systems cannot interpret the visual data they receive properly. Note that there are works that perform SLAM along with automatic camera calibration (e.g. [69]), however we do not concern ourselves with such approaches here. The "Pose" column indicates whether pose GT was available, which we require for evaluation, since without it we cannot measure the accuracy of any localization system (see Sect. 5.5.1). In "Motion", we evaluate the extent to which the camera movements in the dataset are enough to thoroughly test a SLAM system. In [37] for instance, we found that the executed motions were only point-rotations, which does not capture the full range of transforms a system should be tested on. Next, the "Semantics" column indicates whether the dataset offers semantic annotations. Finally, the "Size" column gives a rough estimation as to the number of sequences available for use in each dataset.

2.5.2 Systems overview

Various SLAM solutions have been published over the years. In this section we will present some popular approaches, not only for visual SLAM, but also for LiDAR SLAM. There are several overviews of works in SLAM, for instance [70, 15, 71, 72]. Here we present some interesting methods from the many that are available, in a brief summary format.

Tab. 2.2 shows only the highlights of the features of each method. There are many modern approaches in SLAM, both for LiDAR sensors and for various camera architectures. Given that we are interested in SLAM for indoor spaces, and wish to include semantics in our approach, we also note the variability in those respects for the presented solutions. For the purposes of semantic SLAM, VO solutions were disregarded since they do not do mapping. LiDAR solutions also do not use semantics, so they were also abandoned. Dense solutions were not used, due to computational constraints. Finally, sparse solutions using EM were not applicable for the logic we wished to implement, because our detected entities were too numerous to be compatible with the exponential runtime of EM, and our semantics more diverse than previously explored (e.g. in [9, 51]). This is why SASHAGO, with its advanced geometric infrastructure (see Chap. 3 for more) was found to be ideal for our implementation, whereas ORB-SLAM2 and MaskFusion were two different, competitive alternatives, that we saw as truly meaningful comparison benchmarks for our system.

System	Sensor	Dimensions	Envir.	Map type
Hector SLAM [43]	LiDAR	2D	any	occ. grid
Cartographer [44]	LiDAR	2D & 3D	indoor	occ. grid
LOAM [45]	LiDAR	3D	indoor	occ. grid
OpenVSLAM [46]	mono/stereo/RGB-D	3D	any	sparse
ProSLAM [47]	stereo	3D	outdoor	sparse
LSD-SLAM [48, 49]	mono/stereo	3D	any	semi-dense
ElasticFusion [50]	RGB-D	3D	indoor	dense
Bowman *et al.** [9]	stereo & IMU	3D	indoor	sparse
Doherty *et al.** [51]	stereo & IMU	3D	outdoor	sparse
VSO* [28]	stereo	3D	outdoor	none
SemVO* [29]	RGB-D	3D	indoor	none
SemanticFusion* [18]	RGB-D	3D	indoor	dense
Li *et al.** [19]	stereo	3D	outdoor	sparse
ORB-SLAM2 [12]	mono/stereo/RGB-D	3D	any	sparse
MaskFusion* [13]	RGB-D	3D	indoor	dense
SASHAGO [7]	RGB-D	3D	indoor	sparse

Table 2.2: Various SLAM solutions and their properties. Solutions in **bold** were used in our work, see Chap. 5 for full details. An * is used to denote methods that use semantics.

Chapter 3

Unified Geometric Representations

3.1 Introduction and Motivation

Unified Geometric Representations (UGRs) are mathematical tools that give us a common way of describing heterogeneous geometric primitives. UGRs are described in detail in [63] and are subsequently used in [7], which is a big part of the basis for our work. The geometric primitives that we are interested in are points, lines and planes, all of which are ultimately expressed using the concept of degenerate quadrics. The indoor SLAM community has indeed been moving towards more descriptive geometric structures for perceiving indoor spaces for a while, including full quadrics, as seen in [17, 6].

Indoor spaces provide robots with multiple instances of UGRs to detect and track, which is especially important when considering navigation in mostly empty indoor spaces. An empty room can easily be almost completely devoid of reliable optical features, but if we are able to track lines and planes, then high-accuracy localization can still be achieved. In addition, since points are also being detected and tracked, a scene containing multiple objects and an abundance of optical features will still give good results. This intrinsic robustness against scene variability is very important for indoor SLAM, since indoor spaces can vary drastically even when just moving from one room to the next.

3.2 Mathematical Foundations

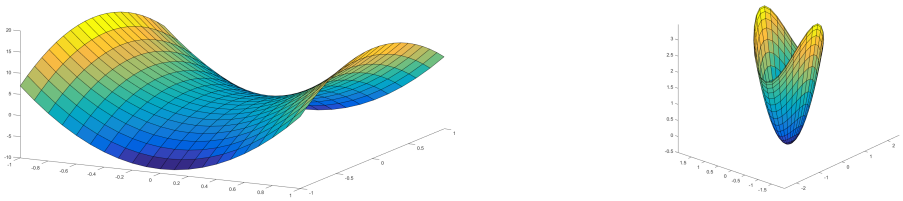
In this section we will present the background necessary for understanding UGRs and their purpose in SLAM. Before we begin, let us first review some basic 3D geometry:

Definition 3.2.1 (Quadric)**.** A *quadric* or *quadric hypersurface* is a generalization of conic sections in D dimensions, whose points $\boldsymbol{x} \in \mathbb{R}^D$ satisfy an equation of the following form:

$$\boldsymbol{x}^T \boldsymbol{Q} \boldsymbol{x} + \boldsymbol{p} \boldsymbol{x} + r = \boldsymbol{0}$$

where $\boldsymbol{Q}$ is a $D \times D$ matrix, $\boldsymbol{p}$ is a D-dimensional row vector and r is a scalar.

Such mathematical objects have been used in works like [17, 6] in the context of SLAM. In [17], the authors used the mathematical properties of quadrics to efficiently project them to conics on the image plane, later using their shape as additional constraints in their back-end factor graph. In the case of [6], the authors used dual quadric parameterizations to change the way that the constrained objects were represented. A dual quadric representation in their context is simply a regular quadric, expressed not via Def. 3.2.1, but instead via a set of tangential planes that constrain its geometry (see Fig. 3.2 for a visualization).



(a) A saddle quadric. (b) One sheet of an elliptical hyperboloid.

Figure 3.1: Visualizations of general quadrics.

3.2.1 Basic definitions

The following is an overview of the key theoretical points of [63, 7]. We are interested in deriving a mathematical model that can uniformly describe 3D geometric structures of three different varieties: points, lines and planes. This can be achieved by claiming that all points belonging to a UGR must satisfy an equation of the same form, the difference then being only a different parametrization for each type of structure. The following equation, corresponding to *degenerate quadrics*, serves this purpose:

$$(\boldsymbol{x} - \boldsymbol{p})^T \boldsymbol{A} (\boldsymbol{x} - \boldsymbol{p}) = \boldsymbol{0} \tag{3.1}$$

where $\boldsymbol{p}$ is the origin point of the quadric and $\boldsymbol{A}$ is a symmetric matrix. The matrix A is what will ultimately define the geometry of the quadric, and to see how that works we need to factorize it as: $\boldsymbol{A} = \boldsymbol{R}\boldsymbol{\Lambda}\boldsymbol{R}^T$. $\boldsymbol{R}$ contains the eigenvectors, that can be thought of as the primary axes along which the quadric extends. $\boldsymbol{\Lambda}$ is a diagonal matrix containing the eigenvalues dictating the shape of the degenerate quadric. In our case, we only need to use the following:

$$\boldsymbol{\Lambda} = \text{diag}(\boldsymbol{\lambda}), \ \boldsymbol{\lambda} = \begin{cases} (1,\ 1,\ 1) & \text{for points} \\ (0,\ 1,\ 1) & \text{for lines} \\ (1,\ 0,\ 0) & \text{for planes} \end{cases} \tag{3.2}$$

Essentially, setting an eigenvalue to 1 activates a constraint in (3.1) along the corresponding eigenvector. If it is 0, the corresponding eigenvector will of course yield 0 on the LHS of (3.1) immediately, so any coordinate for $\boldsymbol{x}$ will satisfy it; in other words, it corresponds to setting that axis free. This is why (3.2) uses the values it does for the different cases of $\boldsymbol{\lambda}$. All 1s leaves no degrees of freedom, therefore describing a point. All 1s except the first entry gives a line along the local x-axis. One 1 and 0s after it means that the local y and z axes are free, but x is constrained, so we get a plane whose normal is parallel to the x-axis.

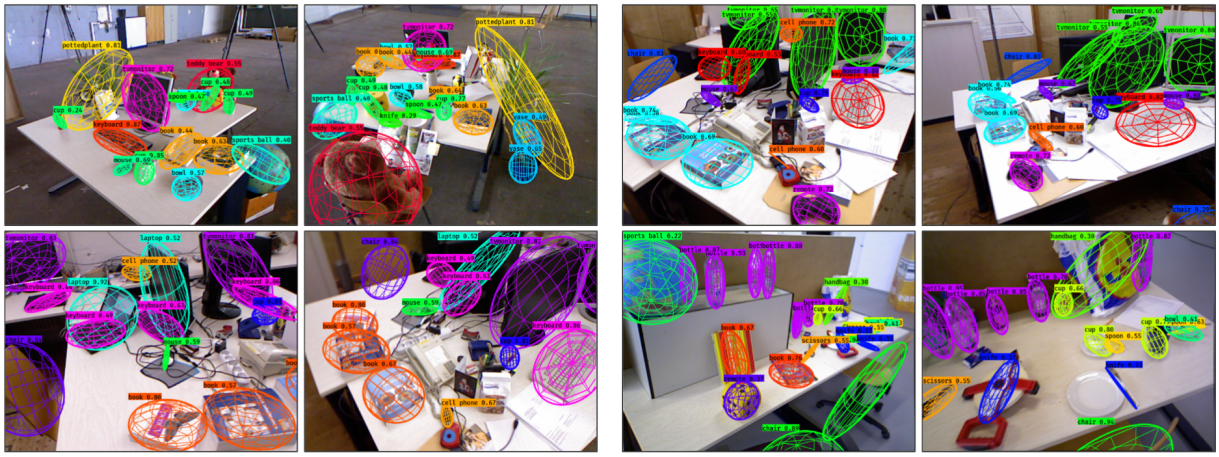


Figure 3.2: A visualization of dual quadrics detected and overlaid on an image from the TUM RGB-D dataset. [6]

To capture this information in a more structured way, the concept of a matchable is introduced.

Definition 3.2.2 (Matchable)**.** A *matchable* is defined as a triplet:

$$M := (\boldsymbol{p}_M,\ \boldsymbol{R}_M,\ \boldsymbol{\Lambda}_M) \tag{3.3}$$

where $\boldsymbol{p}_M \in \mathbb{R}^3$ is the centroid with respect to the world origin frame, $\boldsymbol{R}_M \in \mathrm{SO}(3)$ is the orientation matrix, again with respect to the world origin frame, and $\boldsymbol{\Lambda}_M$ is the diagonal shape matrix.

Matchables play a key role in the implementation seen in Sect. 3.3, which was used to give the detections in Fig. 3.3.

3.2.2 Optimization

Strategy and formalisms

Based on the above, we will now go through the mathematical steps necessary for pose-graph optimization when matchables are involved. As we have seen in Sect. 2.4, the graph-based SLAM optimization framework is quite generic and disconnected from the actual geometric entities to which its nodes correspond. Therefore, it stands to reason that one could extend nodes to represent different geometric entities, provided that error metrics are adjusted to account for the new information. Here, matchables are used as the basis, and the mathematics for the optimization process can be derived by considering

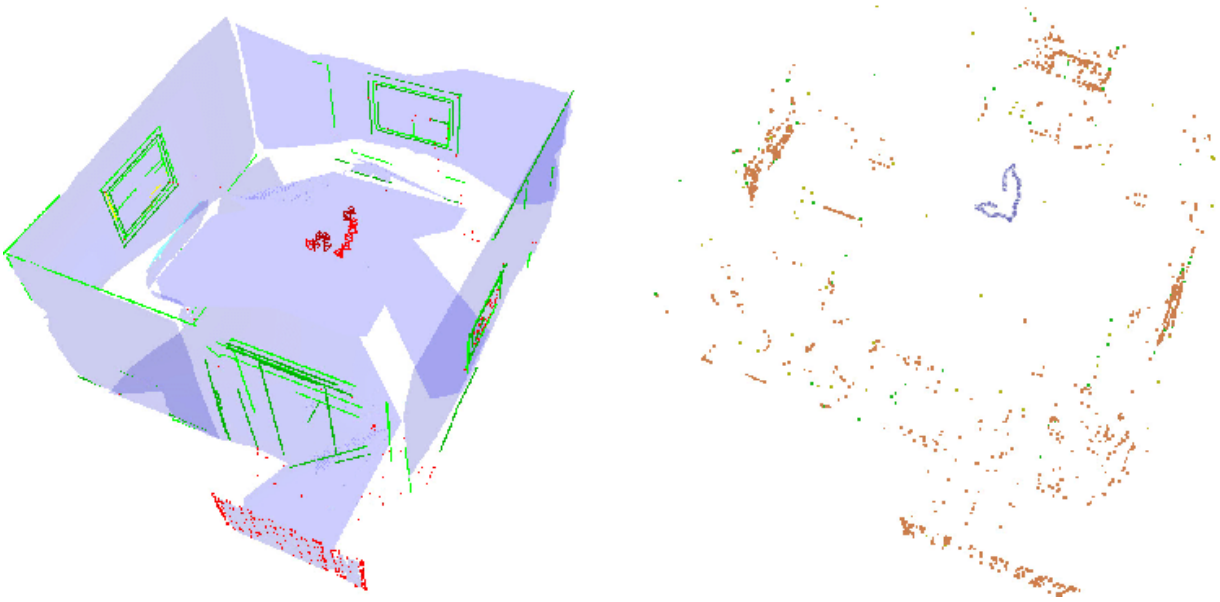


Figure 3.3: A demonstration of matchable detections. Point detections are displayed as red dots, lines are in green and planar regions in blue. Note the estimated camera trajectory in the center of the room. [7]

the error between a matchable $\boldsymbol{m}_j$ and a measurement $\boldsymbol{z}_{ij}$ from pose $\boldsymbol{x}_i$ as follows:

$$\boldsymbol{e}_M(\boldsymbol{x}_i, \boldsymbol{m}_j, \boldsymbol{z}_{ij}) = \begin{bmatrix} \boldsymbol{e}_p \\ \boldsymbol{e}_d \\ e_0 \end{bmatrix} = \begin{bmatrix} \boldsymbol{R}_j^T(\boldsymbol{p}_i - \boldsymbol{p}_j) \\ \boldsymbol{d}_i - \boldsymbol{d}_j \\ \boldsymbol{d}_i^T \boldsymbol{d}_j \end{bmatrix} \tag{3.4}$$

$$\boldsymbol{\Omega}_{ij} = \boldsymbol{C} \cdot \bar{\boldsymbol{\Omega}}_{ij} \cdot \boldsymbol{C}^T, \; \boldsymbol{C} = \begin{bmatrix} \boldsymbol{C}_p & \boldsymbol{0}_{3\times3} & \boldsymbol{0}_{3\times1} \\ \boldsymbol{0}_{3\times3} & \boldsymbol{C}_d & \boldsymbol{0}_{3\times1} \\ \boldsymbol{0}_{1\times3} & \boldsymbol{0}_{1\times3} & C_o \end{bmatrix} \tag{3.5}$$

In the above, we see the necessary extensions to the error function in (3.4) and the information matrix modifications in (3.5). Specifically, the error function has new dimensions added, with the following meaning: $\boldsymbol{e}_p$ for centroid position error, $\boldsymbol{e}_d$ for direction misalignment[1] and e_o for orthogonality. Furthermore, the information matrix needs to be extended to accommodate comparisons between different types of matchables. For this reason, in (3.5) it is edited by an *activation matrix* $\boldsymbol{C}$, which selectively activates error function components depending on matchable geometry, based on the logic of Tab. 3.1. Note that matchables of different geometries can still be compared, subject to some restrictions, because of the unitary eigenvalues of matrix $\boldsymbol{\Lambda}$ (see also [63], Sect. III. A).

[1]For simplicity, $\boldsymbol{d}$ here corresponds to the x-direction in the matchable's local frame (the x-eigenvector from matrix $\boldsymbol{R}$).

$\hat{\boldsymbol{z}}_{ij} \setminus \boldsymbol{z}_{ij}$	Point	Line	Plane
Point	$(\mathbb{I}, \mathbf{0}, 0)$	$(\boldsymbol{\Lambda}_{\boldsymbol{z}_{ij}}, \mathbf{0}, 0)$	$(\boldsymbol{\Lambda}_{\boldsymbol{z}_{ij}}, \mathbf{0}, 0)$
Line	-	$(\boldsymbol{\Lambda}_{\boldsymbol{z}_{ij}}, \mathbb{I}, 0)$	$(\boldsymbol{\Lambda}_{\boldsymbol{z}_{ij}}, \mathbf{0}, 1)$
Plane	-	-	$(\boldsymbol{\Lambda}_{\boldsymbol{z}_{ij}}, \mathbb{I}, 0)$

Table 3.1: Activation matrix types, given as $(\boldsymbol{C}_p, \boldsymbol{C}_d, C_o)$

Finally, the manifold perturbation outlined in Sect. 2.4.2 requires the definition of the perturbations induced by the $\boxplus$ operator. Firstly, the perturbation must be defined:

$$\Delta \boldsymbol{m} = \begin{bmatrix} \Delta \boldsymbol{p}^T & \Delta \alpha_y^T & \Delta \alpha_z^T \end{bmatrix}^T \tag{3.6}$$

Note that rotations along the primary axis of the matchable can *not* affect a matchable. Indeed, points have complete rotational invariance anyway, so rotations are ignored, lines rotated around their own axis are not changed, and planes rotated around their normal are also not changed. Therefore, (3.6) only contains rotations around the local y and z axes, not the primary x axis. Now, we can define the perturbation of a matchable $\boldsymbol{M}$ as follows:

$$\boldsymbol{M} \boxplus \Delta \boldsymbol{m} := (\boldsymbol{p}_{\boldsymbol{M}} + \Delta \boldsymbol{p},\ \boldsymbol{R}_{\boldsymbol{M}} \cdot \Delta \boldsymbol{R},\ \boldsymbol{\Lambda}_{\boldsymbol{M}}) \tag{3.7}$$

where $\Delta \boldsymbol{R} = \boldsymbol{R}_y(\alpha_y) \cdot \boldsymbol{R}_z(\alpha_z)$, meaning the total rotation around the non-primary axes of the matchable. Now, the problem can be cast directly into the form of (2.6), and the estimate can therefore be optimized, provided that we have a way of computing the necessary Jacobians.

Jacobians

We begin by defining what we seek to differentiate, namely the error terms of the pose graph edges. These terms will be differentiated using perturbations, such as those defined in (3.7). An error term $\boldsymbol{e}_{ij}$ is created by an edge linking pose $\boldsymbol{x}_i$ to a matchable $\boldsymbol{m}_j$, and has 7 dimensions, as defined in (3.4). The pose and matchable data vectors are concatenated, with the 6 first dimensions allocated to the pose and the 5 last to the matchable, so when trying to differentiate with respect to all variables involved in the error term, the result is:

$$\frac{\partial \boldsymbol{e}_{ij}(\boldsymbol{x} \boxplus \Delta \boldsymbol{x})}{\partial \Delta \boldsymbol{x}} = \frac{\partial \boldsymbol{e}_{ij}(\boldsymbol{x}_i \boxplus \Delta \boldsymbol{x}_i,\ \boldsymbol{M}_j \boxplus \Delta \boldsymbol{x}_j)}{\partial \Delta \boldsymbol{x}} =$$
$$= \left[\begin{array}{cccccc|ccccc} \mathbf{0}_{7\times6} & \dots & \mathbf{0}_{7\times6} & \boldsymbol{J}_i & \mathbf{0}_{7\times6} & \dots & \mathbf{0}_{7\times6} & \mathbf{0}_{7\times5} & \dots & \mathbf{0}_{7\times5} & \boldsymbol{J}_j & \mathbf{0}_{7\times5} & \dots & \mathbf{0}_{7\times5} \end{array}\right] \tag{3.8}$$

where the separate Jacobians are defined as:

$$\boldsymbol{J}_i = \frac{\partial \boldsymbol{e}_{ij}(\boldsymbol{x}_i \boxplus \Delta\boldsymbol{x}_i,\ \boldsymbol{M}_j)}{\partial \Delta\boldsymbol{x}_i} \tag{3.9}$$

$$\boldsymbol{J}_j = \frac{\partial \boldsymbol{e}_{ij}(\boldsymbol{x}_i,\ \boldsymbol{M}_j \boxplus \Delta\boldsymbol{x}_j)}{\partial \Delta\boldsymbol{x}_j} \tag{3.10}$$

Note the dimensions of the output matrix in (3.8). There are 7 rows, corresponding to the 7 dimensions of the error function that is being differentiated. There are also many columns, separated into blocks. Each column block corresponds to one specific pose from those registered in the pose graph or to one specific matchable from the ones that have been registered. This implies that, for a specific measurement, all poses but one are irrelevant, and their Jacobians are set to $\mathbf{0}_{7\times 6}$ accordingly, meaning there is no impact in any of the 7 error dimensions by any of the 6 pose dimensions. The same applies to the second part of the matrix, where observed matchables play no part in the Jacobian in question, except for the one that has actually been registered in the measurement.

The computation of the above Jacobians will require some mathematical context. The reason for this is that now we have reached the point where the error function will need to be analyzed in each dimension, replaced by the analytical formulation of its components from (3.4), and with the perturbations added in (which are not linear due to the existence of the $\boxplus$ operator).

Definition 3.2.3 (Skew Symmetric Matrix)**.** An $n \times n$ matrix $\boldsymbol{S}$ is said to be *skew-symmetric* if:

$$\boldsymbol{S} = -\boldsymbol{S}^T \Leftrightarrow s_{ii} = 0 \wedge s_{ij} = -s_{ji}, \forall i, j, \in \{1, 2, \ldots, n\}, i \neq j$$

where s_{ij} is an individual element of matrix $\boldsymbol{S}$ at row i and column j.

Based on the above, one can see that from the entries of a 3D vector, a total of $2^3 \cdot 3! = 48$ different 3×3 skew symmetric matrices can be generated, depending on how the signs are arranged and in what way the elements are placed in the matrix's off-diagonal elements. One special case is the following:

$$\boldsymbol{v} = [v_1\ v_2\ v_3]^T \Rightarrow \boldsymbol{S}(\boldsymbol{v}) := \begin{bmatrix} 0 & -v_3 & v_2 \\ v_3 & 0 & -v_1 \\ -v_2 & v_1 & 0 \end{bmatrix} \tag{3.11}$$

The above matrix has a very important property related to vector cross-products, namely:

$$S(\boldsymbol{v}) \cdot \boldsymbol{u} = \boldsymbol{v} \times \boldsymbol{u},\ \forall \boldsymbol{v}, \boldsymbol{u} \in \mathbb{R}^3 \tag{3.12}$$

Now, let a rotation matrix $\boldsymbol{R}(\theta)$ be given, and a derivative with respect to one angle of rotation for that matrix be required. The following holds:

$$\begin{aligned}
\frac{d}{dt}\left[\boldsymbol{R}(\theta)\boldsymbol{R}^T(\theta)\right] &= \frac{d}{dt}\mathbb{I}_{3\times 3} = \mathbf{0}_{3\times 3} \Leftrightarrow \\
\underbrace{\frac{d}{dt}\boldsymbol{R}(\theta)\boldsymbol{R}^T(\theta)}_{\boldsymbol{S}} + \underbrace{\boldsymbol{R}(\theta)\frac{d}{dt}\boldsymbol{R}(\theta)^T}_{\boldsymbol{S}^T} &= \mathbf{0}_{3\times 3} \underset{\substack{\boldsymbol{S}+\boldsymbol{S}^T=0\Rightarrow \\ -\boldsymbol{S}^T=\boldsymbol{S}}}{\Leftrightarrow} \\
\frac{d}{dt}\boldsymbol{R}(\theta)\boldsymbol{R}^T(\theta) &= \boldsymbol{S} \Leftrightarrow \\
\frac{d}{dt}\boldsymbol{R}(\theta) &= \boldsymbol{S}\boldsymbol{R}(\theta)
\end{aligned} \tag{3.13}$$

where $\mathbb{I}_{3\times 3}$ and $\mathbf{0}_{3\times 3}$ are the identity and zero matrices of size 3 respectively. The above relation shows that indeed, the derivative of a rotation matrix is given by left multiplying the rotation matrix with a skew-symmetric matrix. Assign a vector $\boldsymbol{\omega}$ to be generating $\boldsymbol{S}$, according to (3.11). This vector is in fact the definition of *angular velocity* for a time-varying rotation matrix. So ultimately, the time derivative with respect to a static frame A of the coordinates of a point $\boldsymbol{p}$, which is fixed with respect to a rotating frame B, can be computed as follows:

$$\begin{aligned}
\frac{d}{dt}\boldsymbol{p}^A(t) &= \frac{d}{dt}\boldsymbol{R}_B^A(t)\boldsymbol{p}^B \\
&= \left[\boldsymbol{S}(\boldsymbol{\omega}(t))\boldsymbol{R}_B^A(t)\right]\boldsymbol{p}^B \\
&= \boldsymbol{\omega}(t) \times \left[\boldsymbol{R}_B^A(t)\boldsymbol{p}^B\right] \\
&= \boldsymbol{\omega}(t) \times \boldsymbol{p}^A(t)
\end{aligned} \tag{3.14}$$

Now we can tackle the task of computing the Jacobians of (3.8). We begin with the pose Jacobian:

$$\boldsymbol{J}_i = \begin{bmatrix} \dfrac{\partial \boldsymbol{e}_p}{\partial \Delta \boldsymbol{x}_i} & \dfrac{\partial \boldsymbol{e}_d}{\partial \Delta \boldsymbol{x}_i} & \dfrac{\partial e_o}{\partial \Delta \boldsymbol{x}_i} \end{bmatrix}^T \tag{3.15}$$

The components of $\boldsymbol{e}$ are those of (3.4) after the perturbation is applied to the arguments, specifically here the robot pose perturbation $\Delta\boldsymbol{x}_i$. The results after the differentiations are:

$$\begin{aligned}
\frac{\partial \boldsymbol{e}_p}{\partial \Delta \boldsymbol{x}_i} &= \begin{bmatrix} \hat{\boldsymbol{R}}_{ij}^T & -\hat{\boldsymbol{R}}_{ij}^T\boldsymbol{S}(\boldsymbol{R}_i\boldsymbol{p}_{ij} + \boldsymbol{t}_i) \end{bmatrix} \\
\frac{\partial \boldsymbol{e}_d}{\partial \Delta \boldsymbol{x}_i} &= \begin{bmatrix} \mathbf{0}_{3\times 3} & -\boldsymbol{S}(\boldsymbol{R}_i\boldsymbol{R}_{ij}\boldsymbol{u}_x) \end{bmatrix} \\
\frac{\partial e_o}{\partial \Delta \boldsymbol{x}_i} &= \begin{bmatrix} \mathbf{0}_{1\times 3} & \boldsymbol{u}_x^T\boldsymbol{R}_i^T\boldsymbol{R}_{ij}^T\boldsymbol{S}(\hat{\boldsymbol{R}}_{ij}\boldsymbol{u}_x) \end{bmatrix}
\end{aligned}$$

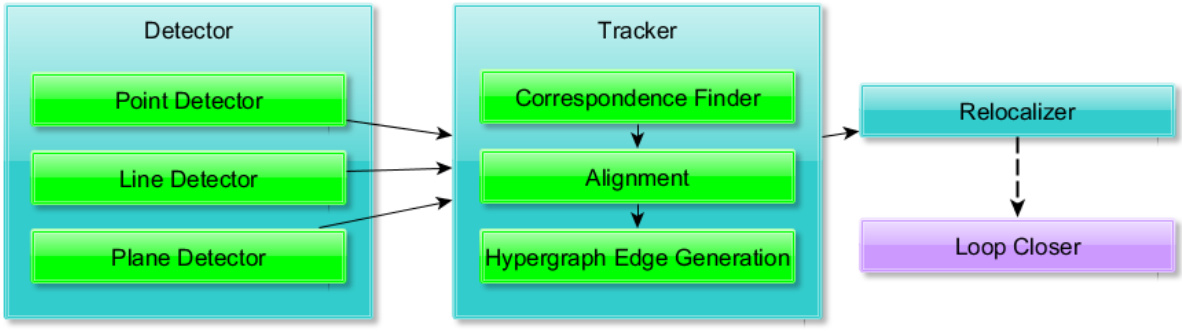


Figure 3.4: Overview of SASHAGO's internal architecture.

where $\boldsymbol{u}_x$ is the unit vector along the x-axis. In the above note the appearance of the $\boldsymbol{S}(\cdot)$ operator, for generating skew-symmetric matrices. Now, applying the perturbation of the matchable and differentiating yields the results for $\boldsymbol{J}_j$:

$$\frac{\partial \boldsymbol{e}_p}{\partial \Delta \boldsymbol{x}_j} = \begin{bmatrix} -\hat{\boldsymbol{R}}_{ij}^T & -\boldsymbol{S}(\hat{\boldsymbol{R}}_{ij}^T \cdot (\boldsymbol{R}_i \boldsymbol{p}_{ij} + \boldsymbol{t}_i)) \cdot \begin{bmatrix} \boldsymbol{u}_y & \boldsymbol{u}_z \end{bmatrix} \end{bmatrix}$$

$$\frac{\partial \boldsymbol{e}_d}{\partial \Delta \boldsymbol{x}_j} = \begin{bmatrix} \boldsymbol{0}_{3\times 3} & \hat{\boldsymbol{R}}_{ij} \boldsymbol{S}(\boldsymbol{u}_x) \cdot \begin{bmatrix} \boldsymbol{u}_y & \boldsymbol{u}_z \end{bmatrix} \end{bmatrix}$$

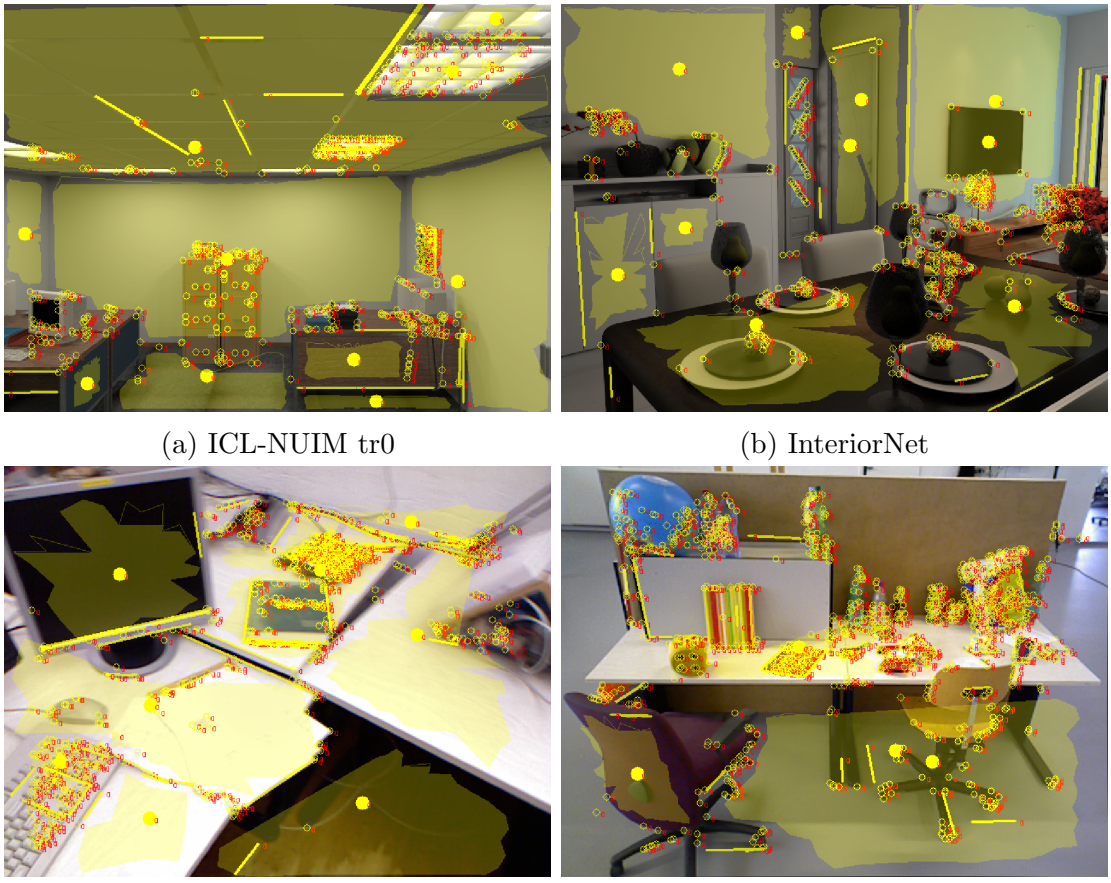
$$\frac{\partial e_o}{\partial \Delta \boldsymbol{x}_i} = \begin{bmatrix} \boldsymbol{0}_{1\times 3} & -\boldsymbol{u}_x^T \boldsymbol{R}_{ij}^T \boldsymbol{R}_i^T \hat{\boldsymbol{R}}_{ij} \boldsymbol{S}(\boldsymbol{u}_x) \cdot \begin{bmatrix} \boldsymbol{u}_y & \boldsymbol{u}_z \end{bmatrix} \end{bmatrix}$$

The reader may observe that the above are in fact quite simple expressions in comparison to what might be expected for the derivatives of such highly complex, non-linear forms. This simplification is due mostly to the definition of the *chordal manifold distance* as an approximation to full manifold large scale distance metrics. It serves, for instance, to simplify vector flattening operations. More details on the particulars of the derivations can be found in [73].

3.3 Implementation Details

This section will describe the system[2] on which we base our work in Chap. 5. The system can be described as performing *Systematic Handling of Heterogeneous Geometric Primitives in Graph-SLAM Optimization (SASHAGO)*, and belongs to the *Sapienza Robust Robotics Group (SRRG)* ecosystem. Let us first discuss the architecture and how it fits the graph-based SLAM model. To see how, we will draw on Sect. 2.4.1's outline, where a graph-based SLAM system is required to have a front-end and a back-end.

[2]The code for this project can be found here: https://gitlab.com/srrg-software/srrg_sashago



(a) ICL-NUIM tr0 (b) InteriorNet

(c) TUM RGBD fr1 (d) TUM RGBD fr3

Figure 3.5: Snapshots of geometric primitives detection (points, lines and planes) in different datasets. Notice how in TUM's real-world instances, plane segmentation is much harder due to corruption by noise.

In SASHAGO, the front-end is comprised of several modules, seen in Fig. 3.4. The detector modules are responsible for finding matchables in a frame. The tracker modules then attempt to link these matchables to others observed earlier, computing initial motion estimates and also updating the pose graph and map entities. Finally, the relocalizer and loop-closer modules handle loop-closure detection, and handle the logic for applying these constraints in the final optimization. Note that matchable detection is handled using different detectors for different types of matchables:

- `OpenCV`'s ORB [24] detector for points
- `OpenCV`'s line segment detector for lines (see [52] for algorithm details)
- clustering method for detecting planes, registered as in [53]

The back-end is implemented in g2o [33], a generic, non-linear least-squares graph optimization framework. Extended to incorporate matchables, it is installed along side the primary components of the frontend, that ultimately link against it to gain access to the available graph construction and optimization routines. The overall coordination of the setup is accomplished through the Robot Operating System (ROS) [74], which is an open-source framework allowing the structured development, testing and deployment of various robotic components in the form of packages.

Chapter 4

Semantic Segmentation

4.1 Introduction

Semantic segmentation is a field of Computer Vision. Given an image, the aim is to identify what is displayed and where in the image it is. This deliberately broad description tries to capture the incredible variety of approaches that exist in the area, both in terms of the methods and technologies developed to solve the problem, and in the parameters of the problem definition. Given the huge variability of entities in the real world, it becomes clear that the general problem is very difficult to solve, therefore all approaches to it require that some kind of prior knowledge be ingrained into the solution. This could be, for example, a geometric model of a typical instance for a class of objects.

Initial attempts at segmenting an image used flexible, probabilistic models, hand-crafted to capture the geometry of the entities they were trying to segment. Deformable Part Models (DPMs) [75] are a prime example. In DPMs, curves for a specific model expose parameters in the Fourier transform space. Matching the boundaries of the object in an image is done using a Bayesian inference rule, with deformations along the object model boundary being applied to fit the image boundaries. The image boundaries are computed by standard computer vision edge metrics, typically a Gaussian-smoothed gray-scale gradient, which adds yet another hand-crafted layer to the system. Later, approaches such as Markov Random Fields (MRFs), Conditional Random Fields (CRFs) and tree-based approaches were tried, using super-pixelization to get the outlines of separate objects. However, such approaches may still fail to capture the unique appearance of an object that, although still belonging to some class, is visually quite different from the norm of that class. This, among many other reasons, has motivated the Computer Vision community to move away from hand-crafted modeling. It has been developing automatic ways of ingesting large amounts of example data and having models implicitly generated, primarily using *deep neural networks (DNNs)* [54, 55].

4.2 Deep Neural Networks

4.2.1 Key concepts

DNNs are a popular machine learning architecture, in which an artificial neural network is extended to include multiple hidden layers (therefore adding to its depth). Although this straightforward extension seems natural, it had been computationally impossible to manage such complex models with conventional hardware. However, modern GPUs have increasingly supported neural network computations, thus enabling deep architectures to be brought in the limelight [56]. The basic building block of a DNN is the neuron, a simple mathematical entity that accepts an input vector, computes its dot product with an internal weight vector and adds a bias (the linear part of the neuron), then finally propagates the result through a non-linear function, known as the activation function

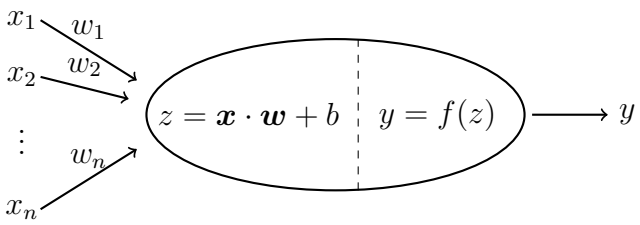


Figure 4.1: The structure of a neuron. The weights w_i scale the inputs x_i, after which the bias term b is added. Finally, the result is passed through the non-linear activation function f, thus yielding the neuron's output.

(see Fig. 4.1). There are many activation functions in the literature, e.g. sigmoid, tanh, Rectified Linear Unit (ReLU), leaky ReLU and more:

$$f_{sigmoid}(x) := \frac{1}{1 + e^{-x}} \tag{4.1}$$

$$f_{tanh}(x) := \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{4.2}$$

$$f_{relu}(x) := \max x, 0 = \begin{cases} x, & x \geq 0 \\ 0, & x < 0 \end{cases} \tag{4.3}$$

$$f_{leaky\ relu}(a, x) := \begin{cases} x, & x \geq 0 \\ ax, & x < 0 \end{cases}, \ a \simeq 0.01 \tag{4.4}$$

Neurons' inputs can either be raw data or, crucially, other neurons' outputs, meaning that they can be stacked one after the other, thus giving rise to layers in the architecture. A hidden layer is simply a layer whose inputs are outputs of previous neurons and whose outputs are inputs of following neurons.

4.2.2 Supervised training

Training a simple neural network usually falls within the realm of supervised learning. Firstly, a labeled dataset is gathered, containing a large number of examples and the solutions that a perfect system would give, typically put together by humans. For instance, this could involve a large number of hand-written digits stored as grayscale images, labeled with the integer digit that the images represent[1]. Then, the network's weights are initialized (see [76] for details on this crucial operation), so it can output something when an input example is given. However, its output is virtually random at this stage, so it

[1]There is a classic dataset in Optical Character Recognition (OCR), known as MNIST (http://yann.lecun.com/exdb/mnist/), which offers exactly this.

gives mostly wrong responses. It undergoes a training phase, where it computes answers for the examples, compares them to the expected answers provided by the dataset, using a loss function, and uses the result of that function to apply a back-propagation algorithm, slightly altering its weights in the process. The hope is that, if the back-propagation algorithm and loss function are parameterized appropriately, the architecture sufficiently complex to capture the difficulty of the problem and the data of quality good enough to allow the model to accurately generalize [2], the resulting network, after training, will give consistently good results when queried on data it has never seen before. Parameterizing a network can involve a wide variety of parameters that need to be set and tested for fitness, a process that is sometimes referred to as *hyperparameter tuning*.

Loss functions come in several different varieties. Three core categories exist, each being best suited for a specific type of task. Note that these categories have several members, each with a slightly different aim and mathematical formulation:

- **Probabilistic:** notably includes various cross-entropy and Kullback-Leibler divergence functions. This category is best suited for classification tasks, as the functions generally compare an output confidence distribution to an expected ideal classification output, scaling with how confident the model was in its error.

- **Regression:** notably includes MSE, Huber and cosine similarity functions. This is the category of choice for regression problems, as they are perfect for penalizing wrong predictions against expected outcomes, scaling with how wrong the prediction was.

- **Hinge:** contains various highly non-linear functions. The aim here is to help models determine similarity *and* dissimilarity between data. Useful for learning embeddings and in semi-supervised learning settings.

The training algorithm itself consists of iterations of two phases. In the forward pass, the training module selects an instance (or a batch of instances, for exploiting parallelization capabilities, common in modern GPUs) from the training data, and computes an output using its current weight configuration. Then, the output is compared with the perfect response (available in the dataset for supervised learning), using the selected loss function. The result of that comparison is then fed into the backward pass, using a tuned back-propagation routine to update the weights accordingly. The update first requires a method for interpreting the loss, which is done via the differentiation of the loss function at the error point. Many strategies exist to translate the derivative computed at the model's output into actionable feedback to be backpropagated through the model. A class

[2] Another widely used solution is **data augmentation**, a process in which copies of training data samples are artificially distorted and added back to the training dataset, to add robustness to the system, e.g. [77]

of learning approaches that are frequently used are the *Gradient Descent (GD)* methods. The basic GD operation works as follows:

$$\boldsymbol{w} := \boldsymbol{w} - \eta \nabla \boldsymbol{L}(w) = \boldsymbol{w} - \frac{\eta}{n} \cdot \sum_{i=1}^{n} \nabla L_i(\boldsymbol{w}) \tag{4.5}$$

where η is the learning rate, a very important hyper-parameter, which does not need to stay constant throughout the training session, and may in fact gradually be reduced automatically in order to help the system smoothly converge to the optimum. Note that in (4.5) the loss is averaged through the entire training set, before a single step can be taken by the optimizer. *Stochastic Gradient Descent (SGD)* performs one step for each individual training sample. *Minibatch Gradient Descent* strikes a tunable balance between the two, feeding a fixed-size batch of data per loss iteration, averaging their losses and taking an optimization step:

$$\boldsymbol{w} := \boldsymbol{w} - \eta \nabla \boldsymbol{L}(w) = \boldsymbol{w} - \frac{\eta}{b} \cdot \sum_{i=1}^{b} \nabla L_i(\boldsymbol{w}) \tag{4.6}$$

where b is the batch size. If b is equal to the training dataset size, we get regular GD, and if $b = 1$ we get SGD.

Other approaches also try to add sophistication to this schema, such as *Root Mean Square Propagation (RMSProp)*, which smooths the gradient over a time window of its iterations:

$$v(\boldsymbol{w}, t) := \gamma v(\boldsymbol{w}, t-1) + (1-\gamma) \left(\nabla L_i(\boldsymbol{w})\right)^2, \ \boldsymbol{w} := \boldsymbol{w} - \frac{\eta}{\sqrt{v(\boldsymbol{w}, t)}} \cdot \nabla L_i(\boldsymbol{w}) \tag{4.7}$$

where γ is the newly introduced forgetting factor. Finally, *Adaptive Momentum Estimation (Adam)* applies a similar smoothing concept, but approaches the problem differently, with an intermediate step to avoid bias errors:

$$m(t+1) := \beta_1 m(t) + (1-\beta_1) \nabla L(\boldsymbol{w}, t) \tag{4.8}$$

$$v(t+1) := \beta_2 v(t) + (1-\beta_2) \left(\nabla L(\boldsymbol{w}, t)\right)^2 \tag{4.9}$$

$$\hat{m} := \frac{m(t+1)}{1-\beta_1} \tag{4.10}$$

$$\hat{v} := \frac{v(t+1)}{1-\beta_2} \tag{4.11}$$

$$\boldsymbol{w}(t+1) := \boldsymbol{w}(t) - \eta \frac{\hat{m}}{\sqrt{\hat{v}} + \epsilon} \tag{4.12}$$

where m, v are the first and second moment, β_1, β_2 are the corresponding forgetting factors and ϵ is a small scalar to avoid dividing by zero.

4.2.3 Semantic segmentation and evaluation metrics

One of the most important works in semantic segmentation was [78], which introduced *Fully Convolutional Networks (FCNs)* to the community. Convolutional neural networks are a way of structuring a network's architecture so that pixels that are spatially close to one another are processed locally. The key idea is that each layer handles data volumes of dimensions $h \times w \times d$, where h and w are the dimensions and d is the number of channels. For an $H \times W$ RGB image in the input layer, the volume would be $H \times W \times 3$. The neurons deployed are batched together in small filters that are convolved with the images of each channel (see Fig. 4.2). As more layers are stacked, the final result is a non-linear filter applied to the initial image, working at multiple resolutions, and with weights shared locally at each step. Therefore, the architecture is lighter, as less weights are needed (since weights that would correspond to distant pixel connections are implicitly dropped) and easier to train for various visual tasks, because the local processing is better suited for semantic entities that are spatially confined to one small area of the image.

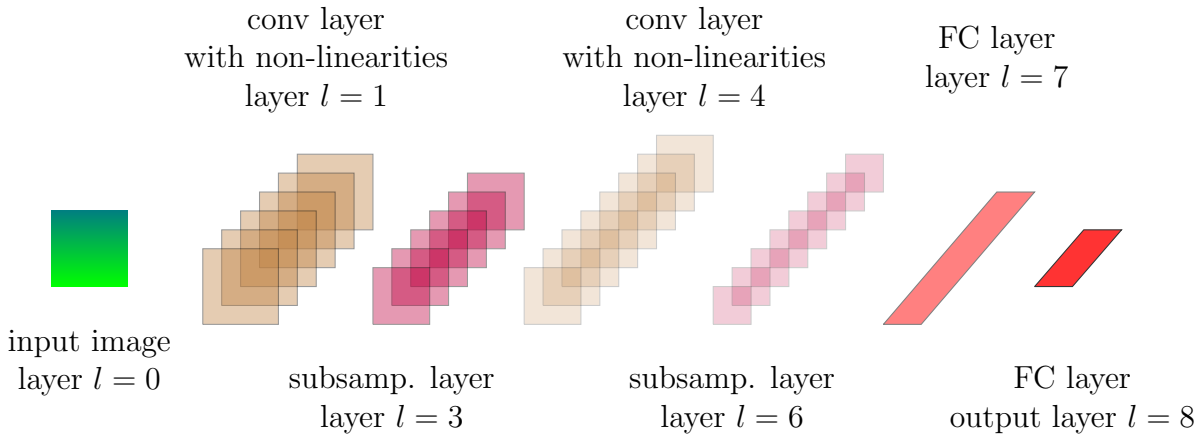


Figure 4.2: An overview of a basic convolutional neural network. The input image is fed into layer 0, after which a series of convolutional layers and subsampling layers are interwoven. Finally, *Fully Connected (FC)* layers are added before the output.

In defining semantic segmentation, we first need to consider what we are seeking to accomplish. For starters, there are several types of segmentation:

- pixel-level: the system attempts to label each pixel of an image with a class number[3]
- *semantic bounding boxes (SBBs)*: the system attempts to find a rectangular outline of semantic entities in the image

[3]Semantic classes (e.g. chair, dog etc.) are often represented as a simple integer, based on the convention used in the training dataset of the segmentation system.

- instance segmentation: this is a more complex task, where a system not only segments instances of classes, but additionally tracks the instances individually [79]

We also require a standardized method of measuring the efficacy of such solutions. There are several metrics to consider, such as pixel accuracy (mean pixel accuracy given in (4.13)), intersection over union (frequency-weighed intersection over union given in (4.14)) and precision-recall curves (precision, recall and F-score defined in (4.15), (4.16) and (4.17) respectively) [54]. Specifically, let us use $n(c_i, c_j)$ to denote the pixels that are labeled with class c_i by the model and actually belong to class c_j. Also, let $t(c_i)$ be the total number of pixels labeled as class c_i in the image. Now, metrics for accuracy can be quantified as follows:

$$mPA := \frac{1}{|\mathcal{C}|} \sum_{i=1}^{|\mathcal{C}|} \frac{n(c_i, c_i)}{t(c_i)} \tag{4.13}$$

$$FwIoU := \frac{1}{\sum_{i=1}^{|\mathcal{C}|} t(c_i)} \sum_{i=1}^{|\mathcal{C}|} t(c_i) \frac{n(c_i, c_i)}{\left(\sum_{j \neq i} n(c_i, c_j) + n(c_j, c_i)\right) + n(c_i, c_i)} \tag{4.14}$$

$$Prec := \frac{n(c_i, c_i)}{\left(\sum_{j \neq i} n(c_j, c_i)\right) + n(c_i, c_i)} \tag{4.15}$$

$$Rec := \frac{n(c_i, c_i)}{\left(\sum_{j \neq i} n(c_i, c_j)\right) + n(c_i, c_i)} \tag{4.16}$$

$$Fscore := 2 \cdot \frac{Prec \cdot Rec}{Prec + Rec} \tag{4.17}$$

4.2.4 Faster R-CNN

After considering many alternatives, we decided to proceed with Facebook Artificial Intelligence Research's (FAIR[4]) implementation of Faster R-CNN [8]. Here, R-CNN stands for *Region-based Convolutional Neural Network*, which simply indicates that the system first proposes regions of interest and then attempts to detect entities within them using a CNN architecture. This classifier, along with many others, such as Mask R-CNN [59], is implemented and packaged in an ecosystem provided by FAIR, which includes documentation and a model zoo. This last property enabled us to test various different implementations of Faster R-CNN, and would have even made it relatively straight-forward to test completely different architectures from the model zoo, which is why it was selected for the purposes of this thesis. Another strong contender was YOLO [58], but seeing as it is a standalone solution it was disregarded for utility reasons.

[4]Code available here: https://github.com/facebookresearch

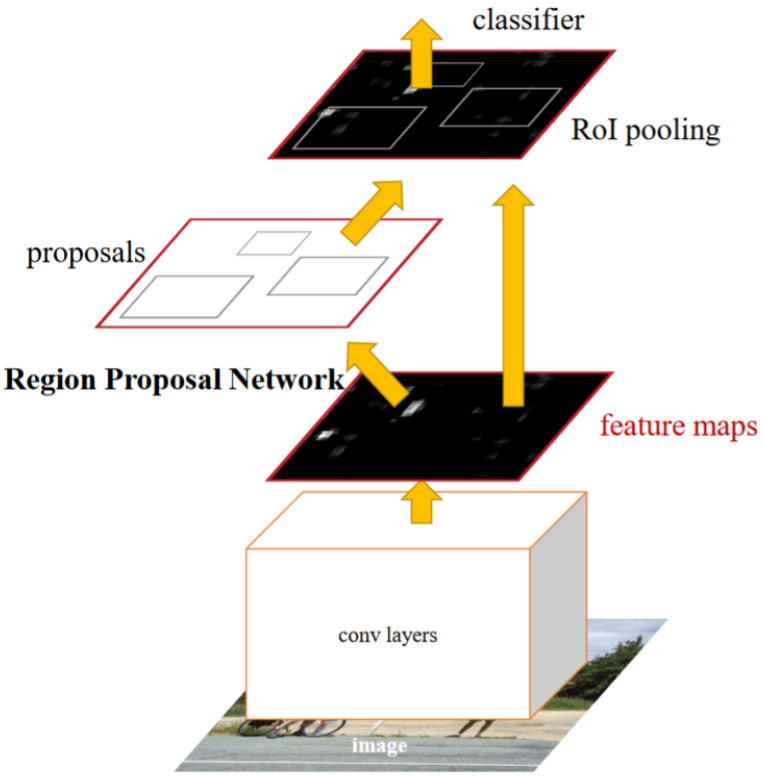


Figure 4.3: A basic overview of the Faster R-CNN architecture. [8]

Faster R-CNN is the successor of Fast R-CNN [57], and shares many similarities with it. The key idea that differentiates Faster R-CNN from Fast R-CNN is the use of a *Region Proposal Network (RPN)*, which is integrated in the system's architecture, directing the network to analyze specific regions of the image first (known as an attention mechanism), and only then attempt to classify them. In Fast R-CNN, proposals are computed by a separate module and are then taken as input by the main network. The architecture for Faster R-CNN is presented in Fig. 4.3, where we see that the system offers a truly end-to-end approach to object detection, since the RPN is an integrated path of neurons within the network. In addition, we see the structure of the convolutional layers, intertwined with *Region of Interest (RoI)* max-pooling, specifically designed for extracting reduced dimensionality feature maps. In our trials, we used it with RGB frames from the TUM dataset, and the results were found to be accurate (see Fig. 4.4 for a sample visualization and Sect. 5.5.2 for more details).

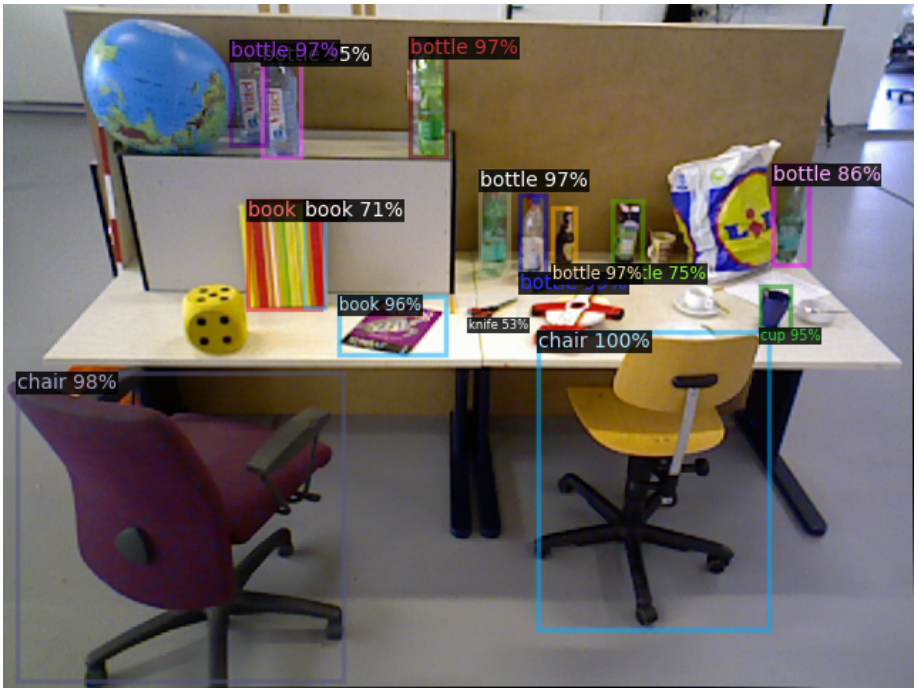


Figure 4.4: Example application of Faster R-CNN on a frame from TUM RGB-D's Freiburg 3 sequences.

4.3 Applications in SLAM

Semantic segmentation has found several applications in SLAM. One of the most inspiring for us was [9]. A lot of useful ideas can be found in this work. Firstly, note the process with which the SLAM pipeline is adapted. It begins by using the detector in the frontend, collecting data that will then be used to impact the pose graph in some way, prior to the back-end optimization step. It uses SBBs to identify objects in the environment, eventually adding semantic edges directly to the graph (see Fig. 4.5 for an illustration). We based initial attempts on this exact schema, but failed to create something meaningful. However, the general outline of having semantics in the frontend, used to directly affect the pose graph was very reasonable and was applied in the end, albeit with big differences in the underlying concepts. The formulation used in [9] is

therefore important. The semantic SLAM problem is described as follows:

$$\begin{aligned}
\mathcal{D}^{i+1} &= \arg\max_{\mathcal{D}} \mathbb{P}[\mathcal{D}|\mathcal{X}^i, \mathcal{L}^i, \mathcal{Z}] \Rightarrow \\
\mathcal{X}^{i+1}, \mathcal{L}^{i+1} &= \arg\max_{\mathcal{X},\mathcal{L}} \sum_{\mathcal{D}\in\mathbb{D}} \mathbb{P}[\mathcal{D}|\mathcal{X}^i, \mathcal{L}^i, \mathcal{Z}] \cdot \log \mathbb{P}[\mathcal{Z}|\mathcal{X}, \mathcal{L}, \mathcal{D}] \\
&= \arg\max_{\mathcal{X},\mathcal{L}} \sum_{k=1}^{K} \sum_{j=1}^{M} w_{kj}^i \log \mathbb{P}[z_k|x_{a_k}, l_j]
\end{aligned} \tag{4.18}$$

where the symbols mean the following: $\mathcal{D}, \mathcal{X}, \mathcal{L}, \mathcal{Z}$ stand for the sets of associations, poses, landmarks and measurements respectively, with $\mathbb{D}$ being the set of all possible associations, i denotes the frame number, k denotes the measurement number, a_k is the index of the associated pose and j the landmark index. The weight w_{kj}^i is computed by a probabilistic model and acts as an importance factor for the optimization, assigning different priorities to different terms according to the formulas that define it. The key observation in this schema, which we adapt to our own approach, is the encoding of priority and significance in a weight factor. This is an important theme that we will revisit in Chap. 5, when we define our own approach for the semantic SLAM problem.

One important detail of the framework in [9] for computing dedicated semantic graph edges, is that it is based on *Expectation Maximization (EM)*. While EM approaches are a great way to ensure theoretical optimality in the usage of semantic information, we see their computational limitations in Chap. 5. In a nutshell, these limitations stem from the system's intrinsic need to evaluate a very large number of potential associations of semantic labels to landmarks in the environment, which is only tractable if the number of landmarks and the number of possible classes are both very small. That is not the situation with our solution, so EM was not directly applicable. The approach was also extended in [51] to account for multimodality, where the authors generalized on the types of probability distributions the system considers, to give a more robust solution.

Other solutions have also been proposed. In SemanticFusion and MaskFusion [18, 13], we see two radically different approaches for semantic SLAM. Both are based on the ElasticFusion [50] dense SLAM framework, which uses small 3D patches called *surfels* to build a geometric reconstruction of its environment. ElasticFusion also does *not* use a pose graph, which makes it even more different from our approach; instead, a deformation graph is used to update the probability distributions of tracked surfels. This means that the generated map will be richer, but the computational requirements are much higher, which we explore in the case of MaskFusion in Chap. 5. On the semantic front, SemanticFusion uses [60] for segmentation, which ultimately yields a class probability distribution for each surfel, and then uses independence assumptions, along with a Bayesian rule, to update the distributions as more data comes in. This is done recursively according to the following

update rule:

$$\mathbb{P}[l_i|\boldsymbol{I}_{1,\dots,k}] = \frac{1}{\eta}\,\mathbb{P}[l_i|\boldsymbol{I}_{1,\dots,k-1}]\,\mathbb{P}[O_{\boldsymbol{u}(s,k)} = l_i|\boldsymbol{I}_k] \tag{4.19}$$

where l_i is the label to be updated, $\boldsymbol{I}$ are the various image detections, k is the current frame index, η is a normalization constant, $\boldsymbol{u}(s,k)$ is a pixel reprojection of a surfel s and $O_{\boldsymbol{u}}$ is the random variable corresponding to the label of pixel $\boldsymbol{u}$. We see here the logic of propagating semantic information based on some local history criterion, with probabilities encoding the model of observations backwards through the surfel's previous label estimations. This is another useful idea, which of course needs adapting for the sparse scenarios we explore in this work. The model here is pixel-based, and so adheres to an internal structure for initializing and propagating probabilities according to that basic constraint. We deviate from this formulation, instead basing our label estimations on the association history of each individual landmark using a weighted voting scheme.

On the other hand, MaskFusion uses Mask R-CNN [59] to extract semantics (see Fig. 4.6), and performs multiple object-level tracking, which are additionally classified as moving or stationary. This distinction is crucial, since it means that the system will be able to disregard moving entities, removing them from the generated map and not allowing them to influence motion estimations. This property lends robustness against dynamic noise, and in our experiments we found that trivial implementations that try to imitate such behaviors often deteriorate performance, so care must be taken to prevent the system from disregarding important information during data processing.

Another approach is [19], which was developed with autonomous driving applications

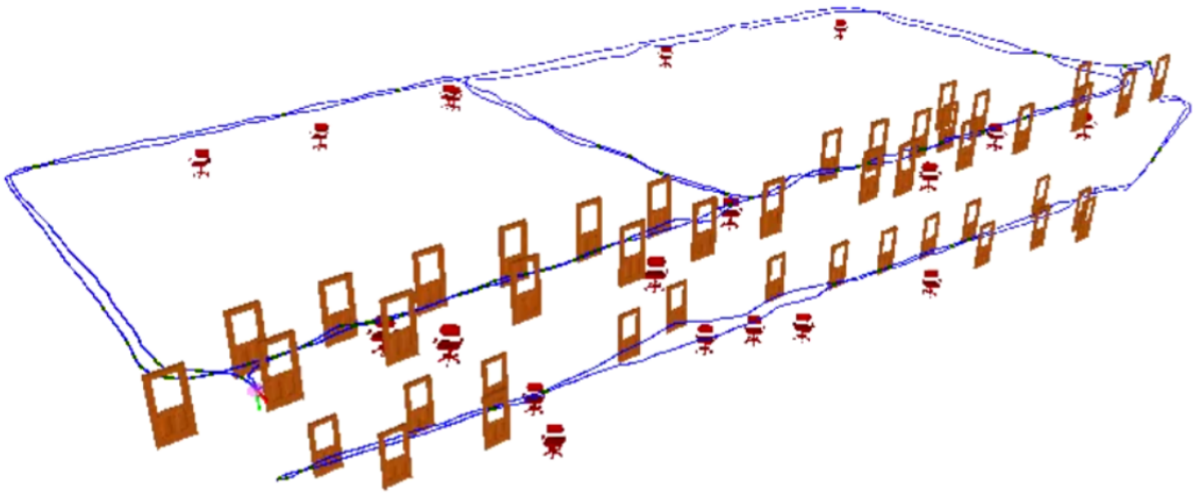


Figure 4.5: Bowman *et al.*'s trajectory estimates with semantics overlaid. Note the semantic classes used are only door and chair. [9]

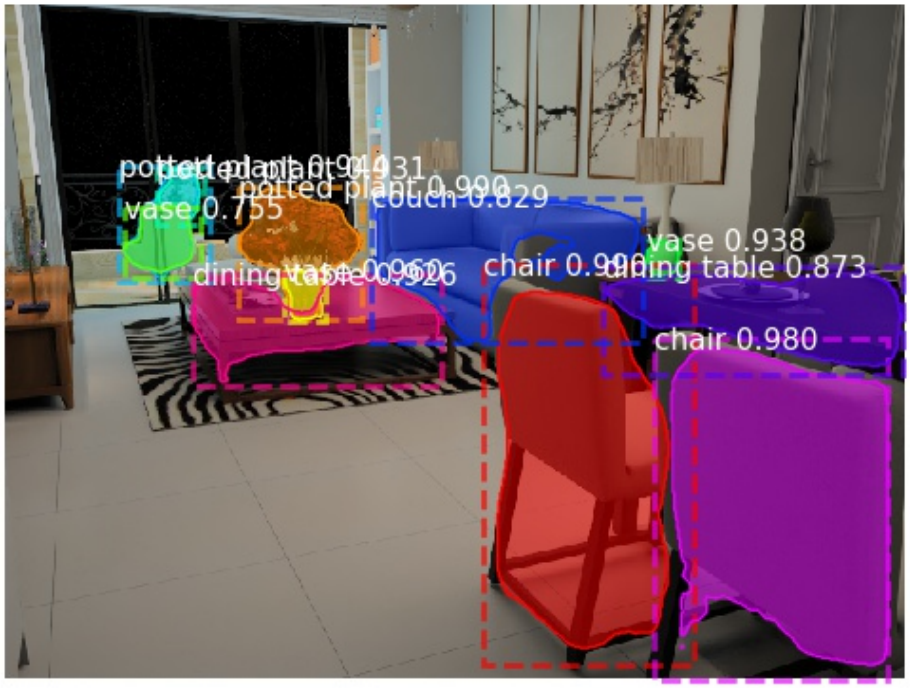


Figure 4.6: MaskFusion semantics for a sample image from the InteriorNet dataset.

in mind. The idea was to train a neural network so as to extract not just semantics, but also to classify the angle of the viewpoint. So in the end a 3D bounding volume of the object and an approximation of the viewpoint's angle toward the object are the output. As in [9], the observations are added to a pose graph and optimized along with geometric constraints. However, instead of EM, the constraints are computed using maximum-likelihood. SegMap [80] is yet another solution, which focuses on hybrid semantic-optical descriptors. Finally, MIT's Kimera library [81] offers a comprehensive framework for SLAM, with a complete stack of modules for visual-inertial odometry, mapping and 3D mesh reconstruction. Semantics are also present, and are propagated throughout the mapping pipeline.

Chapter 5

Our Indoor SLAM Approach

5.1 Introduction

In this section we will present our solution to the indoor SLAM problem. Our key objective is to merge matchables with semantics in a direct, computationally efficient way. The basic intuition is that this approach will try to improve system performance by guiding the optimization process to prioritize constraints that are semantically consistent, at the expense of constraints that break semantic continuity. The major difference is that we intend to do this at the *feature* level, rather than at the *object* level, as seen in other works [9, 51]. This immediately brings up a serious complexity issue of the EM algorithm employed by the aforementioned works, for two reasons: firstly, the possible detection classes are typically in the dozens for modern semantic classifiers, whereas the EM implementations we found were only detecting two, and secondly, the detected features for each frame can be in the hundreds, meaning it is possible to have up to two orders of magnitude of semantically labeled entities per frame than what EM has been designed to support. For these reasons, EM approaches are disregarded, due to the exponential number of possible semantic assignments they must explore to yield their final output.

5.2 Definitions

Let us begin by defining the basic concept of a semantic measurement in our context. The i-th *semantic measurement* of frame t, denoted $\boldsymbol{s}^{t,i}$, is defined as:

$$\boldsymbol{s}^{t,i} = (s_c^{t,i},\ s_s^{t,i},\ \boldsymbol{s}_b^{t,i}) \tag{5.1}$$

where $s_c^{t,i} \in \mathcal{C}$ is an integer representing the semantic class of the observation, $s_s^{t,i} \in [0.0, 1.0]$ is the confidence score of the detection[1], computed by the detector, and $\boldsymbol{s}_b^{t,i} = [(x_{tl}, y_{tl}),\ (x_{br}, y_{br})] \in \big([0, W-1] \times [0, H-1]\big)^2$ is the top-left-bottom-right-corner expression of the pixel region of an SBB in a $W \times H$-sized image. For instance, a simple example of a semantic measurement could be: $(4, 0.85, [(100, 200), (150, 330)])$, indicating that an entity of class 4 was detected with confidence 0.85 in the pixel rectangle from $(100, 200)$ to $(150, 330)$. With the above in mind, we can now proceed to define what a *semantically augmented matchable* $\boldsymbol{M}_s$ is:

$$\boldsymbol{M}^s := (\boldsymbol{p}_M,\ \boldsymbol{R}_M,\ \boldsymbol{\Lambda}_M,\ \boldsymbol{s}_M) \tag{5.2}$$

where $\boldsymbol{p}_M$ is the matchable's centroid, $\boldsymbol{R}_M$ is its orientation, $\boldsymbol{\Lambda}_M$ is its shape matrix and $\boldsymbol{s}_M$ is the semantic measurement selected to describe it (if such a measurement has been found).

[1] In practice, not all scores are accepted. A cutoff value is determined depending on the classifier, and any semantic measurements with confidence score less than the cutoff are ignored.

One note on the above: the values of t in our formulation and implementation are taken as consecutive integers, corresponding to *all* parsed frames. In many SLAM systems, a *keyframe* mechanism is employed, whereby only a few frames are heavily processed by additional SLAM components (including semantics), while all others are used mainly for basic odometry. This means the system only calls computationally expensive methods a fraction of the time, and if the keyframes are selected reasonably, it should incur a negligible accuracy penalty. For instance, [12] select keyframes generously at first, but then culls most of them, since keyframes whose fields of view overlap greatly give little extra information. This means that eventually, operations such as bundle adjustment occur only on very few keyframes, boosting performance. Another example is [9], where the authors preferred to select keyframes at a static rate, meaning that every 15-th frame was considered a keyframe in their system, without checking their content. In our base system, SASHAGO, keyframes are not implemented, so we follow their architecture and build on the implicit assumption that all frames are keyframes.

5.3 Methodology

5.3.1 Determining semantics

We begin by addressing a gap from the previous section, namely how a matchable receives its semantic label; this is where we will begin the analysis of our methodology. For starters, let us imagine the simple case of one point-matchable, whose reprojection on the image plane is contained within a single SBB. The answer is trivial - assign the SBBs label to the point. Now, if a line segment is detected within the SBB, we simply require the centroid of its reprojection to fall within the SBB. The same requirement holds for the planar patches that yield plane matchables. The above does not handle one important case: overlapping SBBs. This is quite common in practice, and is particularly pronounced when a large object is partially occluded by a significantly smaller one. Their SBBs not only intersect, but in fact the smaller is completely contained in the larger. To resolve this, we need an automated way to pick the SBB that will most likely link the matchable with the semantic measurement most closely So, from all the SBBs that contain the representative point of a matchable, we heuristically select the one of *smallest* pixel area, since it will almost always give the matchable the correct semantic label. Formally, we begin by defining the set $S_b^{t,j}$, containing all bounding boxes at frame t which contain the (reprojected) representative point of the j-th matchable, namely $(x^{t,j}, y^{t,j})$:

$$S_b^{t,j} := \left\{ \boldsymbol{s}_b^{t,i} \middle| \, x^{t,j} \in \left[x_{tl}^{t,i}, x_{br}^{t,i} \right] \wedge y^{t,j} \in \left[y_{tl}^{t,i}, y_{br}^{t,i} \right] \right\} \tag{5.3}$$

With the above definition in mind, we can directly define the labeling problem as follows:

$$\hat{\boldsymbol{s}}_b^{t,j} = \underset{\boldsymbol{s}_b \in S_b^{t,j}}{\arg\min}\ \text{Area}(\boldsymbol{s}_b) \tag{5.4}$$

where the Area$(\cdot)$ function is simply the pixel area of the semantic detection.

5.3.2 Semantic persistence

Assuming we have successfully associated semantics with geometric structures detected in the environment, we are now tasked with propagating these associations through time. This is trivial only if features are flawlessly tracked and semantics are perfectly detected, both spatially and in terms of the assigned classes; these conditions do not hold in practice. This is exactly what the EM approaches we saw in [9, 51] were attempting to address. Their idea was that optimizing semantic associations would help the system indirectly, through the added semantic constraints in the factor graph. Our aim is to use semantics to adapt existing geometric constraints, which employ *landmarks* stored in a map structure. Since landmarks are just features that have been reobserved multiple times, it makes sense to assign a class to them based on the semantics of those features. Again, due to noise or semantic detector failures, we cannot be sure that all registered features will perfectly align geometrically or semantically. At best we can hope that they will align *most* of the time. So we take the direct approach and implement **semantic voting**, as a way to approximate a Maximum Likelihood strategy for semantic associations.

Let $\hat{l}_c^{T,j}$ be the estimated class of the jth landmark at time T.

$$\hat{l}_c^{T,j} = \underset{v \in \mathcal{C}_+}{\arg\max} \sum_{t=0}^{T} \mathbb{1}[s_c^{t,j'} = v] \cdot s_s^{t,j'} \tag{5.5}$$

where $j' = \text{idx}(j,t)$ is the semantic measurement index corresponding to landmark j for each frame t, $s_s^{t,j'}$ is the confidence score for each measurement and $\mathcal{C}_+$ is the set of semantic classes, along with a special no-detection class, for frames where the landmark was not observed, which has a default confidence score of 1.0. Essentially, (5.5) uses a weighted voting scheme to determine which is the most probable class of each landmark, called the *dominant class* of the landmark, based on previous semantic detector output. This approach has negligible computational overhead, even when dozens of semantic classes are used, and hundreds of landmarks present in each frame.

5.3.3 Pose graph updating

The last problem we need to solve is determining a mechanism for propagating semantics to the pose graph, so that they can have some impact on the optimization process. As we have seen, computing pure semantic constraints would be problematic, since EM

ideas would have to be involved, resulting in computational overload. Instead, we edit the existing edges of the pose graph, updating them based on semantics. An edge, as we saw in Sect. 2.4.1, contains, among others, an information matrix, that is used to scale the error term. Our key idea is to edit this information matrix, according to the semantics of the observations that create it. We compute a multiplicative semantic weight $w_s^{t,j',j}$ for semantic detection j' of frame t, linked to landmark j, according to the following logic:

$$w_s^{t,j',j} = f\left(\mathcal{D}_C(s_c^{t,j'}, l_c^{t,j})\right) \simeq \begin{cases} 1 & , \; s_c^{t,j'} = l_c^{t,j} \\ W_p & , \; s_c^{t,j'} \neq l_c^{t,j} \end{cases} \tag{5.6}$$

What (5.6) captures is that if the detected semantic class for a feature is the same as the dominant class for the landmark the feature is matched with, then there is no penalty. Otherwise, the penalty is set to some other value W_p.

The exact mechanism for computing W_p is dependent on the combination of detector and environment. Ideally, if an approximate confusion matrix was available, the value of W_p would adapt, so as to be strict in the cases where the class mismatch was unlikely to occur, but more lenient where the classes are easily confused. However, systems such as [8] do not report such metrics at all, let alone for specific datasets. So instead, we fixed W_p to have a constant value, and performed manual semantic class clustering to help with some aberrant behavior in [35] (see Sect. 5.5 for more details).

5.4 Implementation

In this section we will go over some of the technical details of our implementation[2]. Our core system is based on SASHAGO (see Sect. 3.3), which has been changed in two major directions:

- edits to the base code, mostly to address issues with the alignment procedure
- extensions to the base code, to read semantic data, propagate it through the pipeline and eventually use it to effect the pose graph

The code itself is written in `C++`, with various supporting libraries being prevalent throughout, including generic libraries such as `OpenCV` and `Eigen`, as well as ROS packages, primarily from the SRRG ecosystem. In addition to these areas, supporting code was also developed in `Python` and `bash`, aiming to manage the various datasets we tested with (more on this in Sect. 5.5).

[2]Code available at: https://gitlab.com/_JackFrost_/sem-sashago

5.4.1 Geometry

We edited the base code's alignment procedures firstly by updating a background method for detecting the nearest neighbor of a point inside a pointcloud. The data structure used is a modified KD-Tree. KD-Trees are used as an efficient way to query pointclouds, since they store them internally in such a way as to facilitate, for example, nearest-neighbor queries. This is done by selecting a point and splitting the rest in left and right relative to that point, along one dimension, then cycling dimensions and recursively splitting the left and right subclouds. This essentially defines hyperplanes that fragment the space further and further. In SRRG, KD-Trees are built slightly differently, with hyperplanes being chosen using *Principal Component Analysis (PCA)*, instead. What we did was rewrite a part of the recursion for nearest neighbors, so as to make the search complete rather than approximate (which was the default). In addition, we updated the logic of the module responsible for the pointcloud alignment itself to use a dynamically computed number of damped iterations instead of just one.

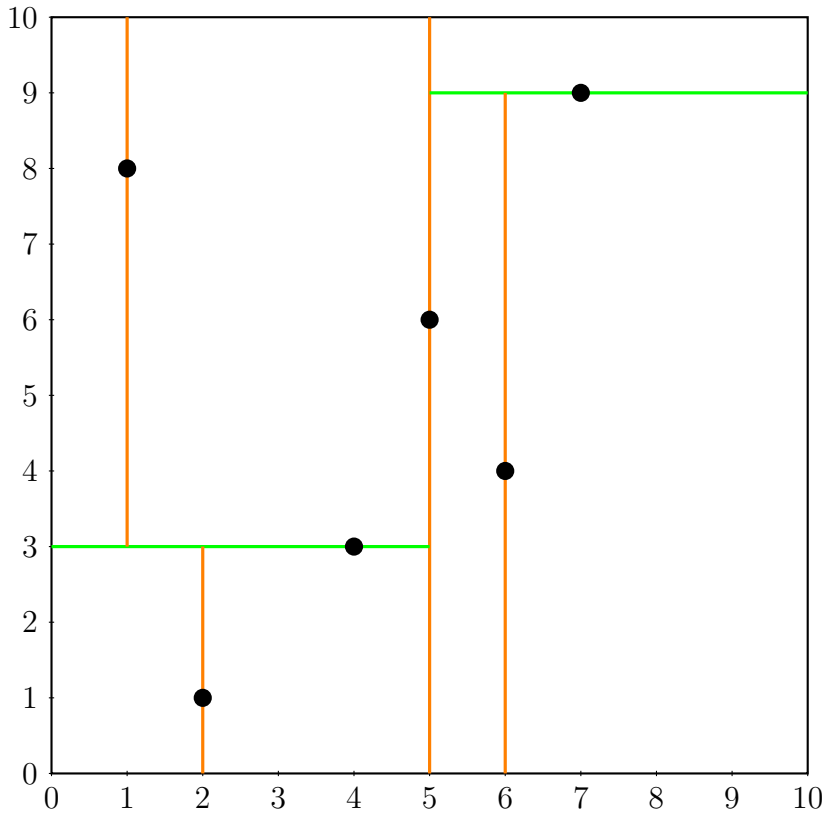


Figure 5.1: Visualization of a two-dimensional KD-Tree, where hyperplanes are reduced to lines. The splitting logic here is basic dimension cycling and pivots are selected as the median for each subtree.

Intuitively, geometric nearest-neighbor search is not always the soundest way of searching for good matches. Each detected feature is considered for association with a previously discovered landmark, by running a nearest-neighbor search over the internal KD-Tree structure. We implemented a tunable hybrid metric for matching detected features to mapped landmarks, which trades off between the geometric distance and the Hamming descriptor distance of the proposed associations. Specifically, we first detect all neighbors within a small geometric search radius, using a KD-Tree search operation, and then select the neighbor minimizing the hybrid metric:

$$f(\boldsymbol{m}_1, \boldsymbol{m}_2) = d_H\left(D(\boldsymbol{m}_1), D(\boldsymbol{m}_2)\right) + \lambda \left\|\boldsymbol{p}_1 - \boldsymbol{p}_2\right\| \tag{5.7}$$

where λ is a tunable weight parameter, $\boldsymbol{p}_1$, $\boldsymbol{p}_2$ are the 3D centroids of the matchables, $D(\cdot)$ is the descriptor function for the matchables (e.g. ORB or BRIEF binary descriptors) and $d_H(\cdot)$ represents the Hamming distance between the two resultant bitstrings, defined as the sum of logical XOR values between their elements:

$$d_H(\boldsymbol{v}_1, \boldsymbol{v}_2) := \sum_{i=1}^{L} v_1[i] \oplus v_2[i] \tag{5.8}$$

In this way, the system can give priority either to geometric proximity or descriptor similarity. The association problem for finding the best correspondence $\widetilde{\boldsymbol{m}}_d$ for each individual detected feature $\boldsymbol{m}_d$ from the set of mapped features $\mathcal{M}$ can be expressed in the form:

$$\widetilde{\boldsymbol{m}}_d = \underset{\boldsymbol{m} \in \mathcal{M} \cap \mathcal{N}_r(\boldsymbol{m}_d)}{\arg\min} f(\boldsymbol{m}_d, \boldsymbol{m}) \tag{5.9}$$

where $\mathcal{N}_r(\boldsymbol{m}_d)$ is the neighborhood of radius r around $\boldsymbol{m}_d$.

5.4.2 Semantics

A semantic SLAM system requires some way of externally receiving semantic labelings. In a keyframe setting (see Sect. 5.2), this could be a service existing in the ROS network, whereby a server-client architecture would enable the system to dispatch keyframes for semantic analysis and utilize the results in the pose graph later, thus computing semantics in real time. We deviate from this design, since keyframes are not implemented in our framework. We compute semantics during dataset preprocessing, for the entire sequence, and then export them to a standardized output file. This is an ordinary text file with a specific format for each line, containing a timestamp, the top-left bottom-right parameterization for the SBB, the semantic class ID (as an integer) and the confidence score for the measurement. Note that for the InteriorNet dataset [34] the computation is actually not necessary, since that dataset has groundtruth semantics in the COCO [82] JSON format, so they just need to be converted to our text format before being read by the system.

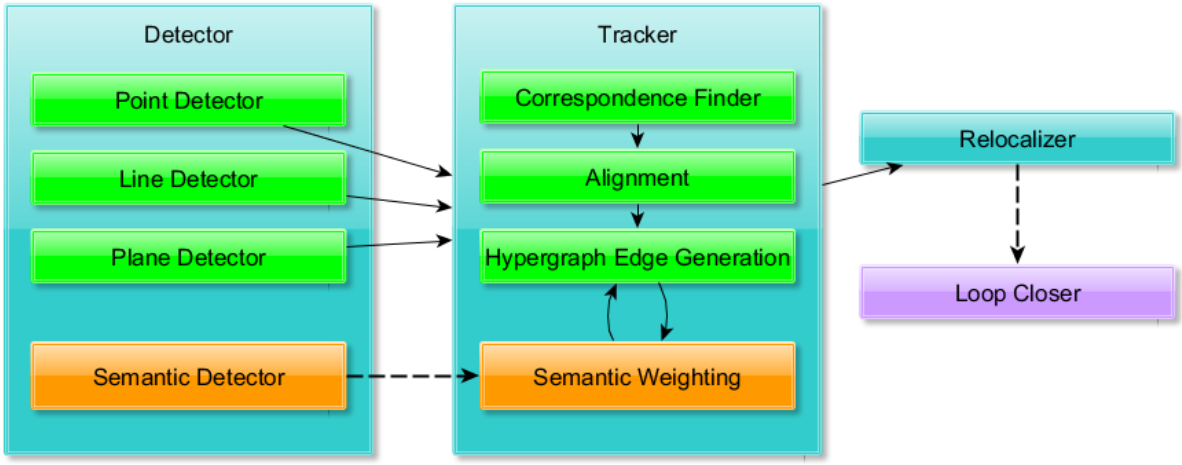


Figure 5.2: An overview of the final system architecture.

With the above functionalities in mind, we implement a `C++` class specifically for handling semantic detection. We concretely implement the I/O method for reading semantics from a file; the way in which the class is set up will allow for any future extensions to develop the logic necessary for real-time semantic computations. We then enrich several other areas of the code to account for semantics, specifically for computing, storing and propagating the values from Eqns. (5.4), (5.5) and (5.6). A brief overview of the final system is shown in Fig. 5.2, where the orange components are original additions.

5.5 Experimental Evaluation

Verifying a SLAM system's efficacy is not a straightforward task, as multiple components coexist in a pipeline that can be difficult to introspect. Outlining an evaluation framework requires first and foremost a robust, scalar metric for system performance. In other words, we need a method that quantifies how well a system is performing, that will not be implementation-specific, and will provide a fair and intuitive mechanism for evaluating and, crucially, comparing the accuracy of various different SLAM systems. Secondly, we need to collect varied datasets, with highly accurate trajectory ground truth available. It is important to differentiate between synthetic and real-world datasets, since the former tend to be perfectly accurate in their data stream synchronization and groundtruth, while also commonly displaying much less noise, than the latter. Finally, an optional requirement for our specific design is semantic groundtruth availability, which is decidedly rarer in datasets available in the literature. Finally, we require other SLAM systems to compare our performance against, so as to demonstrate not only the improvements achieved in comparison to the original base code of our project, but also to outline the relative

standing of the end result compared to other methods.

5.5.1 Evaluating SLAM performance

To systematically quantify SLAM precision, we focus on localization accuracy, since the system does not perform 3D reconstruction. The goal here is to derive a scalar metric computing the "distance" between the estimated trajectory (the SLAM system's output) and the ground truth trajectory (GT). Our evaluation process comes from [35]'s toolkit, which provides numerous error metrics; we use the *Absolute Trajectory Error (ATE)* metric in our evaluation. For ATE, the GT and estimated pose sequences are, first, optimally aligned to eliminate reference frame ambiguity and subsequently the error metrics are uniquely determined [10]. Then the Root Mean Square Error (RMSE) is computed, to convert the error to a scalar.

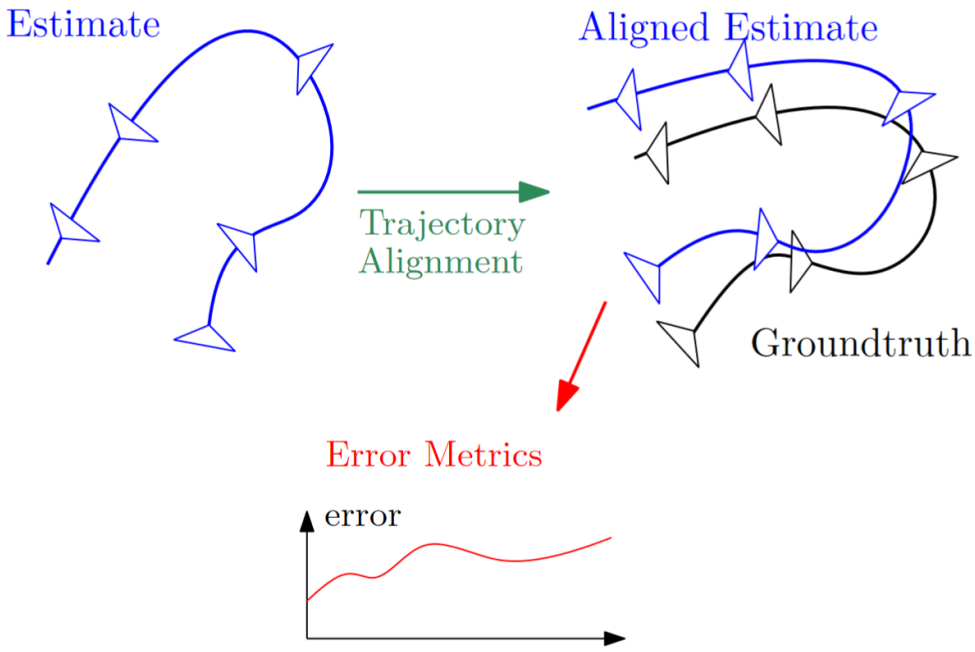


Figure 5.3: Outline of the process for trajectory error estimation. [10]

To be more specific, let us review the process of calculating the optimal parameters that align the estimated and groundtruth trajectories. The idea is to define a quadratic error term, then proceed to compute the statistical properties of the two sequences of data (averages, covariances etc.). Finally, these properties are used to give the optimal solution. The process is outlined in Alg. 5 and the reader is referred to [14] for the full mathematical derivation.

Algorithm 5: Sequence alignment algorithm, computing optimal scale, rotation and translation for two input sequences. [10, 14]

Input: estimated sequence: $\{\hat{\boldsymbol{p}}_i\}_{i=0}^{N-1}$, GT sequence: $\{\boldsymbol{p}_i\}_{i=0}^{N-1}$
Output: $\arg\min_{s,\boldsymbol{R},\boldsymbol{t}} \sum_{i=0}^{N-1} \|\boldsymbol{p}_i - s\boldsymbol{R}\hat{\boldsymbol{p}}_i - \boldsymbol{t}\|^2$

`# compute statistical properties`
$\boldsymbol{\mu_p} = \frac{1}{N}\sum_{i=0}^{N-1} \boldsymbol{p}_i$
$\boldsymbol{\mu_{\hat{p}}} = \frac{1}{N}\sum_{i=0}^{N-1} \hat{\boldsymbol{p}}_i$
$\sigma_{\boldsymbol{p}}^2 = \frac{1}{N}\sum_{i=0}^{N-1} \left\|\boldsymbol{p}_i - \boldsymbol{\mu_p}\right\|^2$
$\sigma_{\hat{\boldsymbol{p}}}^2 = \frac{1}{N}\sum_{i=0}^{N-1} \left\|\hat{\boldsymbol{p}}_i - \boldsymbol{\mu}_{\hat{\boldsymbol{p}}}\right\|^2$
$\boldsymbol{\Sigma} = \frac{1}{N}\sum_{i=0}^{N-1} (\boldsymbol{p}_i - \boldsymbol{\mu_p}) \cdot (\hat{\boldsymbol{p}}_i - \boldsymbol{\mu}_{\hat{\boldsymbol{p}}})^T$
`# perform singular value decomposition:` $\boldsymbol{\Sigma} = \boldsymbol{U}\boldsymbol{D}\boldsymbol{V}^T$
$\boldsymbol{U}, \boldsymbol{D}, \boldsymbol{V} = svd\_decomp(\boldsymbol{\Sigma})$
if $det(\boldsymbol{U}) \cdot det(\boldsymbol{V}) < 0$ **then**
  $\boldsymbol{W} = diag(1, 1, -1)$
else
  $\boldsymbol{W} = \mathbb{I}_{3\times 3}$
end
`# compute the values to return`
$s = \frac{1}{\sigma_{\hat{\boldsymbol{p}}}} trace(\boldsymbol{D}\boldsymbol{W})$
$\boldsymbol{R} = \boldsymbol{U}\boldsymbol{W}\boldsymbol{V}^T$
$\boldsymbol{t} = \boldsymbol{\mu_p} - s\boldsymbol{R}\boldsymbol{\mu}_{\hat{\boldsymbol{p}}}$
return $s, \boldsymbol{R}, \boldsymbol{t}$

What the above algorithm shows is the process for computing the optimal transform in the general case, where even scale is unknown. If it is known, then $s = 1$ is set at the beginning as a constant.

5.5.2 Datasets

There are many SLAM datasets available in the literature, as well as many semantic segmentation datasets. Selecting among them requires checking that at least a minimum set or requirements is met:

- RGB-D datasets only
- indoor environments
- consecutive frames of coherent, non-trivial motions
- complete calibration matrices for the cameras used

- full pose GT over the entire trajectory
- *(optional)* semantic segmentation GT for each frame

After examining a multitude of different datasets and listing their advantages and disadvantages, we selected three of them for use with testing, two synthetic and one real-world. See Tab. 2.1 for a more complete overview.

ICL-NUIM [11]

The ICL-NUIM dataset is a synthetic RGB-D dataset, featuring trajectories of indoor spaces. It was one of the very first datasets that we considered, since [7] actually evaluated their solution on it, showing promising results. After inspecting the dataset closely we observed that the environments within are relatively simple in structure, without much clutter, and, crucially, with many, clearly defined lines and planes for our pipeline to detect and track. Unfortunately, no semantic groundtruth is available, and the artificial environments contain few items, which are well outside the distributions learned by Faster R-CNN. So, instead of testing our full solution, we restricted the experiments to our pipeline *without* using semantics, so purely based on our geometric improvements, as a check that they were indeed improving system performance.

InteriorNet [34]

Another synthetic dataset, InteriorNet boasts an exceptionally large number of indoor sequences. For each scene, a trajectory is automatically generated, with the random movements being confined by different standard deviations for different sequences. Lower values for these deviations give slower, more gradual movements, whereas higher values result in more abrupt motions. There are also alternative versions of each scene, with random lighting instead of the default light sources. We found that lighting changes did not effect the system too much, provided that enough light was present to capture all the details of the environment.

Some technical elements that required our attention early on were the following:

- **depth scaling**: depth data for all datasets we have seen is stored in grayscale PNG images, usually with 16 or 32 bits per pixel. The values of these images are typically interpreted as integers by the input system, expressing the depth value *scaled* by some factor. For ICL-NUIM and TUM RGB-D, that value is 5000 (e.g. a depth of $1m$ would be represented by a pixel value of 5000). For InteriorNet, it is actually equal to 1000, which was not mentioned in the documentation.
- **depth convention**: in Fig. 5.4, we see the standard model for an RGB-D camera's measurement convention. For all datasets except this one, the PNG depth images report the z value for each pixel. InteriorNet does not follow that convention, instead

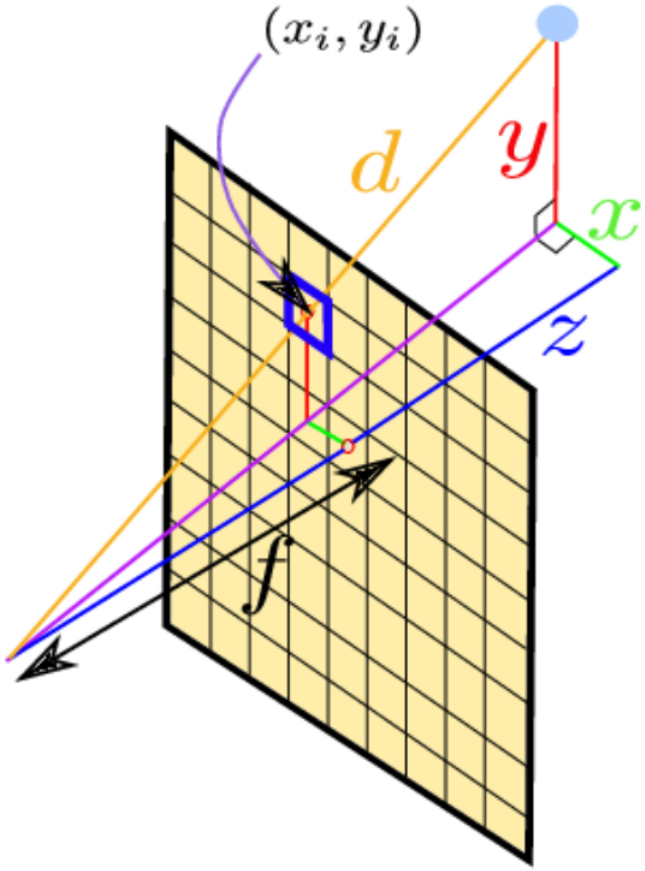


Figure 5.4: The camera depth registration model. [11]

reporting d. Consequently, we had to use the following method to preprocess the depth images:

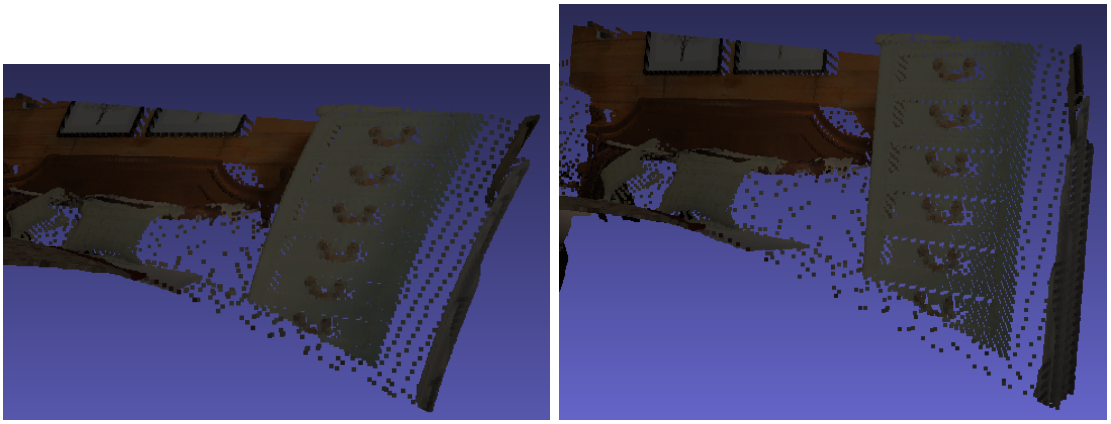
$$z := f \cdot \sqrt{\frac{d^2}{(x - x_c)^2 + (y - y_c)^2 + f^2}} \tag{5.10}$$

where x_c, y_c, f are all constants known from the calibration data of the dataset (see Fig. 5.5 for a visualized example).

- **trajectory failures**: generating random trajectories helps with bulk data generation, but unfortunately it sometimes leads to unrealistic or physically impossible simulations. For instance, the virtual camera would sometimes cross through minor objects, e.g. a lamp, a glass, a plant etc., which confused our systems tracker significantly. In addition, trajectories where frequently interspersed with long segments where the virtual camera would stare at a blank wall or door from a very close distance. Unable to detect anything other than a single plane, our system was prone

to losing its bearings, particularly with regards to orientation, in those cases; we found that the loop closer could not adequately recover from this. To avoid these complications, subsequences where manually selected that corresponded much closer to the quality of the trajectories in more realistic datasets.

- **semantic conversion**: semantics are captured in the COCO format, which is based on JSON. We convert to our text format from Sect. 5.4.2 beforehand, noting that class IDs are different from Faster R-CNN's, although that does not alter performance.



(a) Example MeshLab reconstruction of an InteriorNet scene without correction.

(b) Same area after correction; note that lines are straight as opposed to curved.

Figure 5.5: InteriorNet depth correction example.

TUM RGB-D [35]

This is a real-world dataset, almost universally used as part of the test suites for both outdoor and indoor SLAM systems. The dataset comes with a set of tools written in `Python`, which are very useful both for data preprocessing and the evaluation of results. There are multiple sequences to try, all of them of indoor spaces, using cameras with a depth channel, such as the Kinect or the Xtion. We focus on sequences without dynamic elements, but try our solution for some which include moving humans as an additional performance test, even though it was not within the system's primary specifications.

Two aspects of the data are worth pointing out. Firstly, since the sensors involved are real, the RGB and depth streams are not perfectly synchronized to start. We use the provided association script to achieve this, and then modify our preprocessing pipeline to work with such an association file. Secondly, the data itself is plagued by noise. From our preliminary overview of the data, the noise appears to fall into two primary categories:

motion blur for the RGB channels and depth noise. Motion blur only exists in sequences where movement of the camera was too rapid for the shutter speed. Depth noise, however, is present everywhere, which has to do with the time-of-flight infrared sensors used for such measurements. For some pixels, measurements are not registered at all (e.g. screens, or some types of floor), which leads to the depth value being set to 0 by default. So, in order to denoise the images, we preprocess them with a median filter of kernel size $k = 5$. Other kernel sizes were also tried, and we even attempted bilateral filtering of various parameterizations, following [83], but the performance was found to be poorer.

Finally, it is important to revisit (5.6), since now there a real detector is part of the system. The key problems with the detector are the following:

- **flickering**: this is the name we give to the case of a bounding box switching labels due to a missclassification. The object is framed properly by an SBB, but the classifier frequently alternates the labels it assigns to it. In indoor settings, this can occur for instance when a TV monitor is mistaken for a laptop, or a remote is mistaken for a cell phone. Since no confusion matrix is available, we instead resort to a hand-crafted hierarchical solution. Classes that are commonly confused are merged into one superclass, so that the flickering effect is completely nullified. So a laptop and a TV monitor are merged together in a "screen-like" class, and the remote and cell phones are now labeled as "hand-held electronics". This method requires no change in the architecture of the neural network or its data, merely a change in the output's interpretation.

- **false positives**: this is a rare phenomenon that we observed in some TUM RGB-D's Freiburg 3 subsequences. The detector would at times mistake background walls or panels that happened to align in a specific way as an instance of a class, e.g. a refrigerator. This was probably a minor problem, however we extended our detection filtering to remove instances of such problematic classes, lest they caused unexpected problems later on.

5.5.3 Testing other methods

In addition to testing our own method and its ancestor, we tested two different SLAM systems to get some comparative results on the same data. The first system is ORB-SLAM2 [12], a standard, highly accurate SLAM solution, that tracks ORB features to estimate movement, building a sparse map as it goes. The second system is MaskFusion, and is in many ways the polar opposite of ORB-SLAM2. It is a dense, surfel-based SLAM system, operating with a deformation graph rather than a pose graph, and uses semantics as well as geometry to optimize its estimations, as well as to detect and filter out dynamic entities (ORB-SLAM2 only achieves this passively via RANSAC).

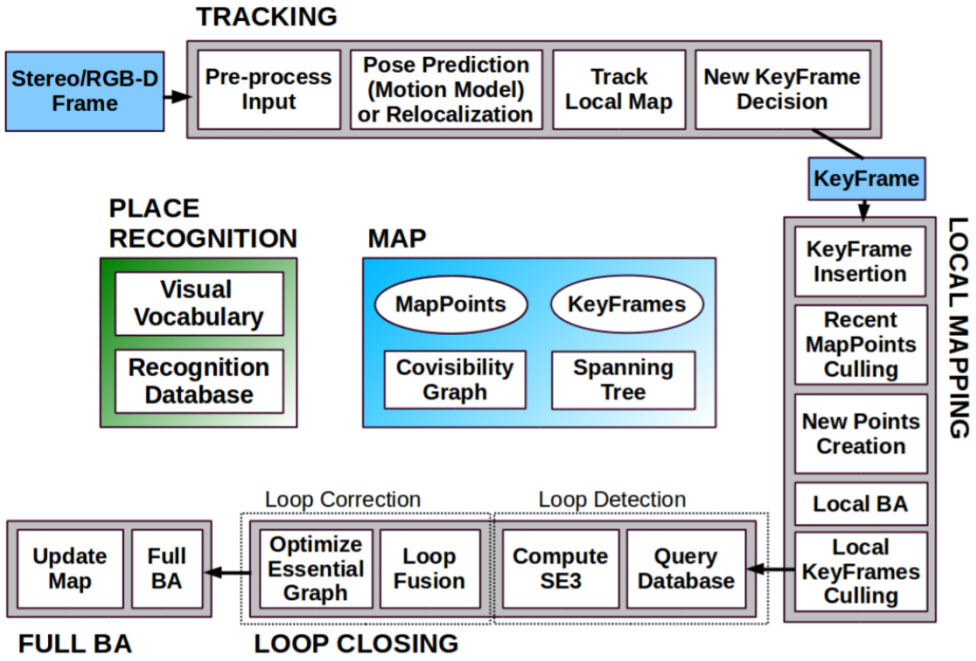
In terms of computational requirements, ORB-SLAM2 is by far the lightest, since its sparse methodology intrinsically reduces the CPU load, and in the absence of a semantic

component no classifier needs to run. SASHAGO follows; still a sparse methodology, but burdened with additional UGR detections and tracking. Our solution has no noticeable performance drop compared to SASHAGO, assuming that classification is offloaded to the preprocessing step. If it is done in real-time, then the classifier's performance will be the bottleneck, and for true real-time operation either keyframes would have to be implemented, or an extremely efficient detector would need to be deployed on very potent hardware. Finally, MaskFusion is the most resource intensive of all, requiring two high-end GPUs to run in full.

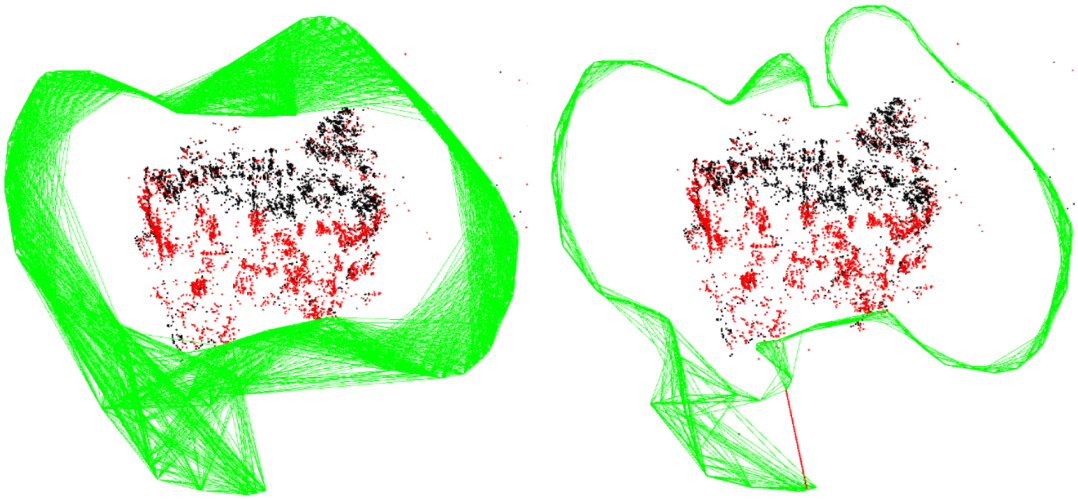
ORB-SLAM2

This SLAM system is based on the pipeline of Fig. 5.6a. The system can support a variety of sensor formats (monocular, stereo etc.), from which we use the RGB-D infrastructure, since this is the type of data we test on. Processing in ORB-SLAM2 is split into three main treads, with a fourth one being spawned to handle BA for loop closures. Features in ORB-SLAM2 are points detected using the FAST [22] detector, on a scale pyramid of the input images. The images are split into a grid, and each cell has a minimum number of required features that need to be detected, which in turn informs threshold adjustment for FAST. The results are then assigned an ORB descriptor and are used for matching.

The internal structure of the pose graph is hierarchical. A covisibility graph is used to store information for all frames, however this can grow quite large and complex (see Fig, 5.6b), so optimizing it fully can be computationally demanding. To avoid this, a more abstract essential graph is introduced, which includes only keyframes and some of the edges between them, including strong loop-closure edges; it is a connected subgraph of the covisibility graph (see Fig. 5.6c). Loop detection is done using a visual bag of words implementation, based on DBoW2 [84]. For efficiency, images are stored in a search tree and frames that are temporally adjacent are split up into *islands*, so that during a loop closure query only one representative is actually compared. Comparison is based on similarities of features extracted from the images and codified using an ORB descriptor.



(a) The ORB-SLAM2 pipeline. Note how the processing is split into four distinct threads, for tracking, local mapping, loop closing and Bundle Adjustment (BA).



(b) Covisibility graph.

(c) Essential graph.

Figure 5.6: System architecture for ORB-SLAM2, along with visualizations of the internal graphs it constructs. For the graphs, green edges correspond to the various pose constraints, red edges are the results of loop closure. The red and black points are parts of the environment map. [12]

MaskFusion

In contrast to ORB-SLAM2, MaskFusion is a dense SLAM framework, based on ElasticFusion and Mask R-CNN. The basic idea is to store a 3D model of each separate object tracked by the system. A model consists of all the surfels that are grouped into its point-cloud. The minimized metric is a weighted combination of geometric and photometric losses:

$$
\begin{aligned}
E_m &:= \min_{\xi_m}(E_m^{icp} + \lambda E_m^{rgb}) \\
&= \min_{\xi_m} \left(\sum_i \left((v^i - exp(\xi_m) v_t^i) \cdot \boldsymbol{n}^i \right)^2 + \lambda \sum_{\boldsymbol{u} \in \boldsymbol{\Omega}} \left(\mathcal{I}_t(\boldsymbol{u}) - \mathcal{I}_{t-1}^a (\pi(exp(\xi_m)\pi^{-1}(\boldsymbol{u}, \mathcal{D}_t))) \right) \right)
\end{aligned}
\tag{5.11}
$$

The Iterative Closest Point (ICP) term is the result of the error of aligning the surfel clouds of different frames, subject to ξ_m as the optimization parameter. To do so, the summation is carried over all surfel indices, with position vector errors projected along the normal of the surfel. This eliminates false errors that would arise in the case of a surface being split in different, but correct, surfels. The photometric term sums along all pixels $\boldsymbol{u}$ of the image, reprojecting the previous image, via the π function, to the current one, subject to the ξ_m optimization variable. The image $\mathcal{I}$ is in fact cast to grayscale for this term, and $\mathcal{D}$ corresponds to the depth channel of the image. Finally, the weight λ

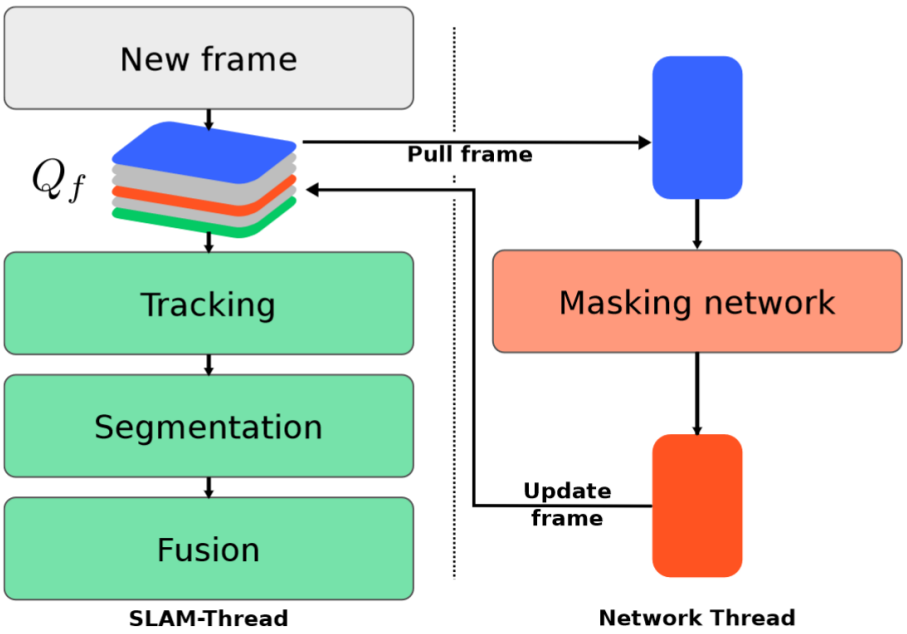


Figure 5.7: Overview of the multithreaded architecture of MaskFusion. [13]

determines the ratio of mixing between the two terms, acting as one of the many system parameters available to the user.

For semantic segmentation, the MaskFusion uses Mask R-CNN, but it goes a step further by integrating geometric criteria in its segmentation logic. It is mostly refined using an "edginess" metric, which attempts to use the 3D point coordinates of a neighborhood of each point to quantify if it belongs to an object's edge. An interesting detail is that, due to the computational complexity of running Mask R-CNN, it is in fact impossible to run the full segmentation pipeline at the full framerate. What happens instead, is that Mask R-CNN is ran as frequently as possible (using a multithreaded implementation), and in the intermediate frames, the models of the last available semantic labeling are used, along with geometric segmentation, to provide tentative labels to the new surfels, as illustrated in Fig. 5.7.

5.5.4 Results

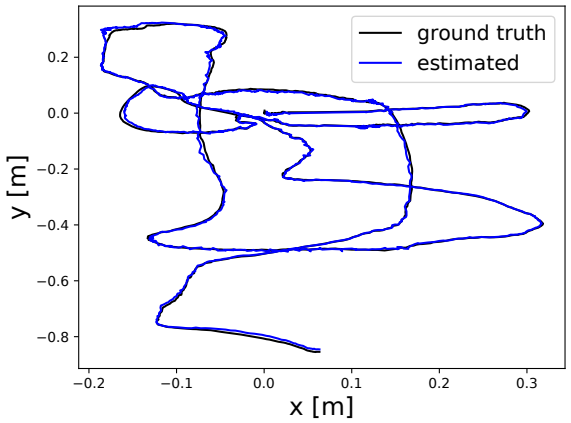
We shall now present our results in the three test datasets we ran on. For each dataset, we report the outcomes as evaluated by the evaluation scripts, ultimately resulting in an ATE RMSE value expressed in meters. In addition, we only report the best error values we could obtain, from the multiple trials executed with different values of the system parameters. Converging to a good parameterization is not trivial and required a fair amount of trial-and-error, with optimal parameters varying quite widely depending on the dataset in question. Finally, to aid in our analysis, we include a direct metric of semantic richness, *SemInfo*, expressed as the median and mean of semantic detections per frame (where applicable).

ICL-NUIM results

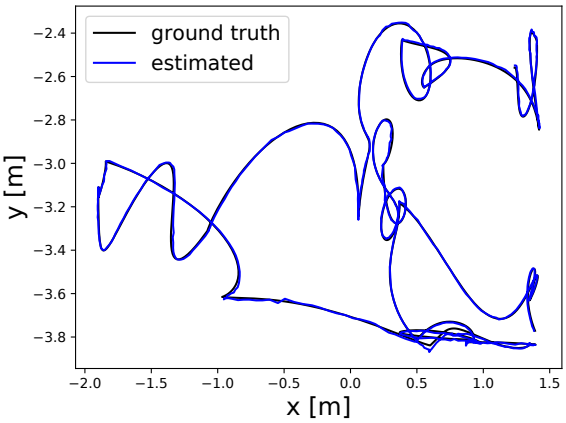
In Tab. 5.1 we see the results obtained on ICL-NUIM. Our system performs very well in these settings, since the environments are noise free (synthetic) and geometrically simple, so UGRs are already giving good results in SASHAGO. Our version has noticeable and consistent improvements, due to the tweaking we saw in Sect. 5.4.1. The trajectories in these sequences are involve both translations and rotations along all axes, meaning that the system also detects planes on the ceilings of the rooms in question, which allows for even better localization.

InteriorNet results

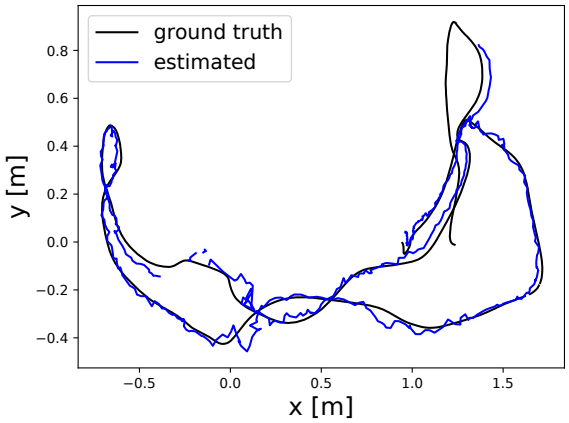
Tab. 5.2 is where we introduce semantics for the first time. We see that in all sequences where semantics are rich, improvements are observed between our solution with semantics disabled versus with semantics enabled. Here, semantic GT is used directly from the dataset, after being converted to our text format from the original COCO-JSON format.



(a) ICL-NUIM trajectory.



(b) InteriorNet trajectory.



(c) TUM RGB-D trajectory.

Figure 5.8: Estimated trajectories and GT for different datasets. In (c), note the effect of noise on the system's performance in a real-world dataset, as opposed to the synthetic photorealistic datasets (a) and (b).

ORB-SLAM2 is outperformed in sequences `3o11_open` and `4o11_200-600`, which are very simple geometrically; in fact, the latter is simply an empty room. This indicates that point tracking does not perform quite as well in such environments, so UGRs are shown again to be better suited here. Note also that semantics are not much help either, as there is almost nothing to detect, so our solution without the semantics is marginally better. We also run MaskFusion on these sequences, although the semantics are not fed from the GT, but rather computed by Mask R-CNN, as per the operations pipeline of the system. Running MaskFusion here revealed that, under noise-free conditions, the dense maps generated are impressively detailed.

Table 5.1: ICL-NUIM results (RMS ATE [m])

Sequence	Ours (w/o sem.)	SASHAGO	ORB-SLAM2
tr0	**0.0131**	0.0156	0.0212
lr0	**0.0238**	0.0519	0.0036
lrkt1	**0.0046**	0.0091	0.0392
lrkt2	**0.0067**	0.0148	0.0281

Table 5.2: InteriorNet results (RMS ATE [m])

Sequence	SemInfo (Med/Avg)	Ours	Ours (w/o sem.)	SASHAGO	MaskFusion	ORB-SLAM2
133_open	8/8.2	**0.0075**	0.0091	0.0203	0.0190	0.0057
2r11_250-600	17/15.8	**0.0161**	0.0267	0.0772	0.0205	0.0147
3o11_open	9/8.9	0.0065	**0.0060**	0.0179	0.0179	0.0189
4o11_200-600	6/5.1	0.0343	**0.0319**	0.0565	0.4595	0.0447
5o11_full	12/12.7	**0.0099**	0.0111	0.0409	0.0110	[track lost]
5o33_200-675	16/15.7	**0.0292**	0.0539	0.0502	0.0575	0.0268
6o11_800-1000	12/11.4	**0.0128**	0.0135	0.0195	0.0164	0.0045

Table 5.3: TUM RGB-D results (RMS ATE [m])

Sequence	SemInfo (Med/Avg)	Ours	Ours (w/o sem.)	SASHAGO	MaskFusion	ORB-SLAM2
fr1_desk	4/4.0	**0.0406**	0.0825	0.1188	0.9116	0.0203
fr2_desk_500	6/5.9	0.0183	**0.0178**	0.1350	0.0581	0.0053
fr2_desk_1000	6/5.9	0.0193	**0.0181**	0.1634	0.0581	0.0067
fr2_desk_full	4/4.6	0.0714	**0.0653**	0.2065	0.4559	0.0085
fr3_loh_200	11/10.2	**0.0161**	0.0267	0.0772	0.0337	0.0178
fr3_loh_1000	6/5.9	0.0521	0.0508	0.0867	**0.0272**	0.0087
fr3_loh_full	7/6.6	**0.1527**	0.1576	0.4145	0.2092	0.0097
fr3_sitting_static	10/10.4	0.0087	**0.0081**	0.0092	0.0117	0.0086
fr3_sitting_xyz	9/8.5	0.0441	**0.0436**	0.0506	0.0530	0.0092

TUM RGB-D results

In Tab. 5.3 we present the real-world data portion of our experiments. The TUM dataset is by far the most demanding, exactly because it is a real-world dataset, with serious noise impacting the measurements, particularly in the depth channel. Working with this data, even after preprocessing it to denoise it, proves quite challenging in sequences where the camera is moved quickly (`fr1_desk`), and the same is true for slower motions such as those of `fr2_desk` and `fr3_loh`. We segmented the latter sequences at intermediate frame counts, to compare the total amount of error present as more frames are accumulated. It becomes obvious that, although our method does improve over the

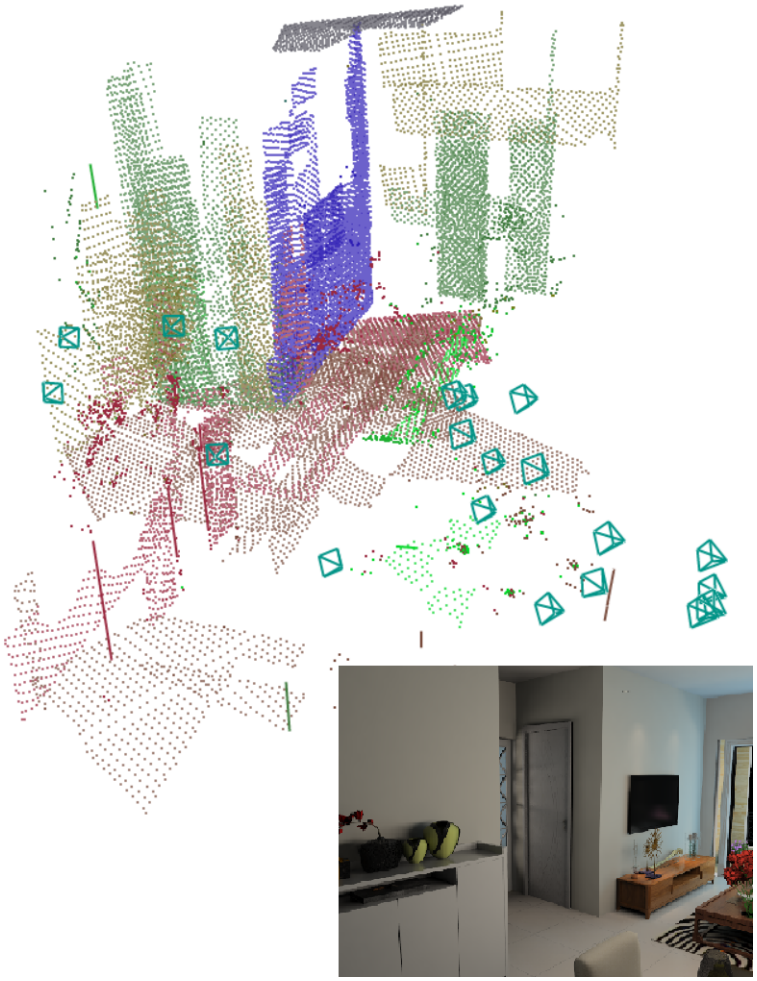


Figure 5.9: RGB-D point-cloud segmentation, showing different semantic classes in different colors, along with a sample frame from the InteriorNet sequence the detections were made in.

base method, the errors have a tendency to accumulate over time at such a rate that loop closure constraints eventually fail to provide the trajectory corrections necessary to correct the estimation substantially. One interesting feature of this experiment is that it uses Faster R-CNN (with class output clustering as discussed in Sect. 5.5.2), and there are measurable improvements observed between our solution with versus without semantics when plenty of detections are present. This also explains why `fr1_desk`'s detections are so low; there are actually more entities to detect in that sequence, but motion blur makes it extremely difficult for our classifier to do so. Finally, the last rows of the results table show the behavior of our system with regards to dynamic entities in the environment (seated, moving humans). Albeit tolerant to small movements and a static viewpoint, the system's

performance is drastically diminished when dynamics are combined with major camera movements, which is to be expected, since there is no provision in the base architecture to detect or filter out such artifacts.

Chapter 6

Conclusions

6.1 Brief Summary of Thesis Contributions

In this thesis we have proposed an approach for semantic indoor SLAM. We began by listing the main ideas we would explore and defined the context of our work. We then outlined the structure of the work in this document and our contribution. Then, we presented a brief overview of the history of localization and mapping research, both for pure odometry and for SLAM. From wheel and LiDAR odometry, onward to VO, and from Bayesian filtering to Kalman filters and beyond, we reviewed all major directions in the literature, before showing our exploration of concrete datasets and system implementations for reference.

Next, we gave the mathematical extensions to the fundamental work in graph-based SLAM, in the form of UGRs. From the basic definitions of Lie groups and algebras and the methods they facilitate in non-linear least squares optimization, we extended the theory to encompass more complex geometries. After seeing how degenerate quadrics can be expressed in a form amenable to consistent error terms, we discussed the advantages that these geometric structures offer to indoor SLAM applications, and presented our implementation base architecture.

We continued with semantics and a brief overview of one of the most active research areas in the world of technology: machine learning and neural networks. After outlining the definition of the problem, we showed some basic methodologies for tackling it. We also introduced the concept of a neural network, its mathematical principles and many architecture variants. Specifically, convolutional neural networks were analyzed, and the Faster R-CNN model was presented in more detail. We also delved into some example implementations of semantic SLAM, with their various different formulations and architectures, as well as their distinct pros and cons.

Finally, we presented our approach to semantic SLAM. After introducing the conventions and elementary definitions we would use, we showed how the use of semantics can be integrated with UGR-based pose-graphs. Then we highlighted the basic improvements in the fundamental workings of the base system, especially on the geometric front, before showcasing the final architecture of our system. We continued by presenting the common framework for SLAM systems evaluation, and discussing the various datasets that we used in our tests, along with the specifics for adapting the datasets for use with our pipeline. Next, we presented in detail the two other methods we tested with, to provide a broader picture of SLAM performance in these datasets. To conclude, we reported the results of the methodologies on the datasets, commenting on the performance profiles we observed and how they could be interpreted.

The contribution of this thesis can therefore be summarized as follows: to begin with, an overview of the fundamentals of odometry and SLAM, and more specifically, graph-based SLAM. To continue, a presentation of UGRs and their applications in SLAM. Then, an overview of machine learning for semantic segmentation and its adaptation in semantic

SLAM. Finally, the theoretical formulation and concrete implementation of a combined UGR and semantic SLAM pipeline, which is efficient enough to allow for integration of semantics at negligible computational cost and additionally yields competitive results in modern indoor datasets.

6.2 Future Research Directions

Semantic indoor SLAM is far from being a solved problem, and through our work we have highlighted some of the challenges that lay ahead in this field. The theoretical obstacles and practical limitations that we encountered, as well as those we found in the literature, have brought to light numerous directions of research that could prove to be instrumental for the advancement of both our understanding of this topic and the sophistication, robustness and safety of real-world implementations based on these foundations. Below we list a few crucial ones:

- **resilience to noise**: while it is true that sensors are becoming ever more accurate, solutions exposed to real-world data do not behave in the same way that they do with simulated datasets. This is especially true for the current UGR implementation we worked with, since registering large geometric structures from noisy pointclouds is a whole separate problem in and of itself.

- **keyframe selection**: systems that understand when a frame is important based on some criterion other than just counting frames could help drastically ease the computational load of running any SLAM pipeline. Work in this direction could lead to the development of more perceptive and responsive systems.

- **segmentation architectures**: with the overwhelming wealth of semantic segmentation choices available in the literature, choosing a framework for the future can be a challenging endeavor. Bleeding-edge technical solutions are becoming standardized that package AI solutions in efficient formats for deployment (e.g. Intel's OpenVINO toolkit[1]), using may different types of hardware, such as CPUs, GPUs and even FPGAs. It is therefore crucial to investigate how these architectures can tie into SLAM solutions in a modular way, and to make good trade-offs between accuracy and performance.

- **dynamics and variability**: humans are particularly adept at managing to localize and map complex environments, both indoors and outdoors, transitioning between the two quite seamlessly, even without digital aids such as GPS. In addition, we can plan our actions in these environments despite the existence of dynamic elements, within them, as well as other factors such as varying weather or lighting conditions.

[1] https://software.intel.com/content/www/us/en/develop/tools/openvino-toolkit.html

This flexibility is supported in great part by the semantic understanding of our surroundings, and advances towards the interplay between SLAM, semantics and automated planning is what will lead to more complete solutions for mobile robotics in the future.

Appendix A

Algorithm for Manifold Optimization

Algorithm 6: Manifold Graph Optimization [3]

Input: $\breve{\boldsymbol{x}}_{[1:T]}$, $\mathcal{C} = (\boldsymbol{e}_{ij}, \boldsymbol{\Omega}_{ij}) - \forall i, j$ s.t. $\exists$ constraint $\boldsymbol{z}_{ij}$
Output: $\boldsymbol{x}^*$, $\boldsymbol{H}^*$

while $\neg$ *converged* **do**

`# compute Jacobian projectors`

for $i = [1..T]$ **do**

$$\boldsymbol{M}_i \leftarrow \left. \frac{\partial \breve{\boldsymbol{x}}_i \boxplus \Delta\widetilde{\boldsymbol{x}}_i}{\partial \Delta\widetilde{\boldsymbol{x}}_i} \right|_{\Delta\widetilde{\boldsymbol{x}}=\boldsymbol{0}}$$

end

$\widetilde{\boldsymbol{b}} \leftarrow \boldsymbol{0}$, $\widetilde{\boldsymbol{H}} \leftarrow \boldsymbol{0}$

forall $(\boldsymbol{e}_{ij}, \boldsymbol{\Omega}_{ij})$ *in* $\mathcal{C}$ **do**

`# compute Jacobians and project on manifold`

$$\boldsymbol{A}_{ij} \leftarrow \left. \frac{\partial \boldsymbol{e}_{ij}(\boldsymbol{x})}{\partial \boldsymbol{x}_i} \right|_{\boldsymbol{x}=\breve{\boldsymbol{x}}}, \quad \widetilde{\boldsymbol{A}}_{ij} \leftarrow \boldsymbol{A}_{ij} \cdot \boldsymbol{M}_i$$

$$\boldsymbol{B}_{ij} \leftarrow \left. \frac{\partial \boldsymbol{e}_{ij}(\boldsymbol{x})}{\partial \boldsymbol{x}_j} \right|_{\boldsymbol{x}=\breve{\boldsymbol{x}}}, \quad \widetilde{\boldsymbol{B}}_{ij} \leftarrow \boldsymbol{B}_{ij} \cdot \boldsymbol{M}_j$$

`# compute Hessians and coefficient vectors`

$\widetilde{\boldsymbol{H}}_{ii} \mathrel{+}= \widetilde{\boldsymbol{A}}_{ij}^T \boldsymbol{\Omega}_{\boldsymbol{ij}} \widetilde{\boldsymbol{A}}_{ij}$, $\widetilde{\boldsymbol{H}}_{ij} \mathrel{+}= \widetilde{\boldsymbol{A}}_{ij}^T \boldsymbol{\Omega}_{\boldsymbol{ij}} \widetilde{\boldsymbol{B}}_{ij}$

$\widetilde{\boldsymbol{H}}_{ji} \mathrel{+}= \widetilde{\boldsymbol{B}}_{ij}^T \boldsymbol{\Omega}_{\boldsymbol{ij}} \widetilde{\boldsymbol{A}}_{ij}$, $\widetilde{\boldsymbol{H}}_{jj} \mathrel{+}= \widetilde{\boldsymbol{B}}_{ij}^T \boldsymbol{\Omega}_{\boldsymbol{ij}} \widetilde{\boldsymbol{B}}_{ij}$

$\widetilde{\boldsymbol{b}}_i \mathrel{+}= \widetilde{\boldsymbol{A}}_{ij}^T \boldsymbol{\Omega}_{\boldsymbol{ij}} \boldsymbol{e}_{ij}$, $\widetilde{\boldsymbol{b}}_j \mathrel{+}= \widetilde{\boldsymbol{B}}_{ij}^T \boldsymbol{\Omega}_{\boldsymbol{ij}} \boldsymbol{e}_{ij}$

end

`# system setup and solution - first node is fixed`

$\widetilde{\boldsymbol{H}}_{11} \mathrel{+}= \mathbb{I}$, $\Delta\boldsymbol{x} = \Delta\widetilde{\boldsymbol{x}} \leftarrow \text{cholesky\_solve}(\widetilde{\boldsymbol{H}} \Delta\widetilde{\boldsymbol{x}} = -\widetilde{\boldsymbol{b}})$

`# update estimates`

forall $\breve{\boldsymbol{x}}_i$ *in* $\breve{\boldsymbol{x}}$ **do**

$\breve{\boldsymbol{x}}_i \leftarrow \breve{\boldsymbol{x}}_i \boxplus \Delta\widetilde{\boldsymbol{x}}_i$

end

end

...

Algorithm 6: (contd.)

...

`# calculate final results outside manifold and return`

$\boldsymbol{x}^* \leftarrow \breve{\boldsymbol{x}},\ \boldsymbol{H}^* \leftarrow \mathbf{0}$

forall $(\boldsymbol{e}_{ij}, \boldsymbol{\Omega}_{ij})$ *in* $\mathcal{C}$ **do**

$\boldsymbol{H}^*_{ii} \leftarrow \boldsymbol{A}^T_{ij}\boldsymbol{\Omega}_{\boldsymbol{ij}}\boldsymbol{A}_{ij},\ \boldsymbol{H}^*_{ij} \leftarrow \boldsymbol{A}^T_{ij}\boldsymbol{\Omega}_{\boldsymbol{ij}}\boldsymbol{B}_{ij}$

$\boldsymbol{H}^*_{ji} \leftarrow \boldsymbol{B}^T_{ij}\boldsymbol{\Omega}_{\boldsymbol{ij}}\boldsymbol{A}_{ij},\ \boldsymbol{H}^*_{jj} \leftarrow \boldsymbol{B}^T_{ij}\boldsymbol{\Omega}_{\boldsymbol{ij}}\boldsymbol{B}_{ij}$

return $\boldsymbol{x}^*,\ \boldsymbol{H}^*$

Due to the importance of the manifold formulation, we have presented the full algorithm here, which takes an initial guess for the full state trajectory and a set of constraints as inputs and returns the optimized solution and information matrix as outputs.

Bibliography

[1] Y. Ye, L. He, and Q. Zhang, "Steering Control Strategies for a Four-Wheel-Independent-Steering Bin Managing Robot," *Proc. Int. Fed. Automatic Control (IFAC)*, 2016.

[2] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press, 2005.

[3] G. Grisetti, R. Kümmerle, C. Stachniss, and W. Burgard, "A Tutorial on Graph-based SLAM," *IEEE Trans. Intelligent Transportation Systems*, 2010.

[4] R. Jiang, R. Klette, and S. Wang, "Modeling of Unbounded Long-Range Drift in Visual Odometry," in *4th Pacific-Rim Symp. Image and Video Technology*, 2010.

[5] S. Thrun, "Particle Filters in Robotics," in *Proc. 17th Conf. Uncertainty in AI (UAI)*, 2002.

[6] L. Nicholson, M. Milford, and N. Sünderhauf, "QuadricSLAM: Dual Quadrics From Object Detections as Landmarks in Object-Oriented SLAM," *IEEE Robotics and Automation Letters*, 2019.

[7] I. Aloise, B. D. Corte, F. Nardi, and G. Grisetti, "Systematic Handling of Heterogeneous Geometric Primitives in Graph-SLAM Optimization," in *Proc. Robotics: Science and Systems (RSS XV)*, 2019.

[8] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," in *Advances in Neural Information Processing Systems 28*, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, Eds. Curran Associates, Inc., 2015.

[9] S. L. Bowman, N. Atanasov, K. Daniilidis, and G. J. Pappas, "Probabilistic Data Association for Semantic SLAM," *IEEE Trans. Robot. Autom.*, 2017.

[10] Z. Zhang and D. Scaramuzza, "A tutorial on quantitative trajectory evaluation for visual(-inertial) odometry," *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems (IROS)*, 2018.

[11] A. Handa, T. Whelan, J. McDonald, and A. Davison, "A benchmark for RGB-D visual odometry, 3D reconstruction and SLAM," in *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*, 2014.

[12] R. Mur-Artal and J. D. Tardos, "ORB-SLAM2: An Open-Source SLAM System for Monocular, Stereo, and RGB-D Cameras," *IEEE Trans. Robotics*, 2017.

[13] M. Rünz and L. Agapito, "MaskFusion: Real-Time Recognition, Tracking and Reconstruction of Multiple Moving Objects," *Proc. IEEE Int. Symp. Mixed and Augmented Reality (ISMAR)*, 2018.

[14] S. Umeyama, "Least-squares estimation of transformation parameters between two point patterns." *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 13, 1991.

[15] C. Cadena, *et al.*, "Past, Present, and Future of Simultaneous Localization and Mapping: Toward the Robust-Perception Age," *IEEE Trans. Robotics*, 2016.

[16] J. Aulinas, Y. Petillot, J. Salvi, and X. Llado, "The SLAM problem: a survey," *Frontiers in Artificial Intelligence and Applications*, vol. 184, 2008.

[17] M. Hosseinzadeh, Y. Latif, T. Pham, N. Suenderhauf, and I. Reid, "Structure Aware SLAM using Quadrics and Planes," *Proc. 14th Asian Conf. Computer Vision (ACCV)*, 2018.

[18] J. McCormac, A. Handa, A. Davison, and S. Leutenegger, "SemanticFusion: Dense 3D semantic mapping with convolutional neural networks," *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*, 2017.

[19] P. Li, T. Qin, and S. Shen, "Stereo Vision-based Semantic 3D Object and Ego-motion Tracking for Autonomous Driving," in *Proc. European Conf. Computer Vision (ECCV)*, 2018.

[20] D. G. Lowe, "Object recognition from local scale-invariant features," in *Proc. 7th IEEE Int. Conf. Computer Vision (ICCV)*, vol. 2, 1999.

[21] H. Bay, A. Ess, T. Tuytelaars, and L. Gool, "Speeded-Up Robust Features (SURF)," *Comput. Vis. Image Underst.*, vol. 110, 2008.

[22] E. Rosten and T. Drummond, "Machine learning for high-speed corner detection," in *Proc. European Conf. Computer Vision (ECCV)*, 2006.

[23] W. Huang, L. Wu, H. Song, and Y. Wei, "RBRIEF: a robust descriptor based on random binary comparisons," *IET Comput. Vis.*, 2013.

[24] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "ORB: an efficient alternative to SIFT or SURF," *Proc. IEEE Int. Conf. Computer Vision (ICCV)*, 2011.

[25] D. Scaramuzza and F. Fraundorfer, "Visual Odometry [Tutorial]," *IEEE Robot. Automat. Mag.*, vol. 18, 2011.

[26] F. Fraundorfer and D. Scaramuzza, "Visual Odometry: Part II - Matching, Robustness, and Applications," *IEEE Robot. Automat. Mag.*, vol. 19, 2012.

[27] K. Konolige, M. Agrawal, and J. Solà, "Large-Scale Visual Odometry for Rough Terrain," in *Proc. Int. Symp. Robotics Research (ISRR)*, 2007.

[28] K. Lianos, J. L. Schönberger, M. Pollefeys, and T. Sattler, "Vso: Visual semantic odometry," in *Proc. European Conf. Computer Vision (ECCV)*, 2018.

[29] H. Mahé, D. Marraud, and A. I. Comport, "Semantic-only visual odometry based on dense class-level segmentation," in *Proc. 24th Int. Conf. Pattern Recognition (ICPR)*, 2018.

[30] L. Banjanovic-Mehmedovic, E. Ivanjko, and I. Petrovic, "Histogram Based Mobile Robot Localization," in *Proc. Int. Electrotechnical and Computer Science Conf. (IEEE ERK)*, 2005.

[31] P. M. Djuric, *et al.*, "Particle filtering," *IEEE Signal Processing Mag.*, 2003.

[32] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit, "FastSLAM 2.0: An Improved Particle Filtering Algorithm for Simultaneous Localization and Mapping that Provably Converges," in *Proc. 16th Int. Joint Conf. Artificial Intelligence (IJCAI)*, 2003.

[33] R. Kümmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard, "g2o: A general framework for graph optimization," *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*, 2011.

[34] W. Li, *et al.*, "InteriorNet: Mega-scale Multi-sensor Photo-realistic Indoor Scenes Dataset," in *Proc. British Machine Vision Conf. (BMVC)*, 2018.

[35] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers, "A benchmark for the evaluation of RGB-D SLAM systems," in *Proc. Int. Conf. Intelligent Robot Systems (IROS)*, 2012.

[36] O. Wasenmüller, M. Meyer, and D. Stricker, "CoRBS: Comprehensive RGB-D Benchmark for SLAM using Kinect v2," in *Proc. IEEE Winter Conf. Applications of Computer Vision (WACV)*, 2016.

[37] P. Ammirato, P. Poirson, E. Park, J. Kosecka, and A. C. Berg, "A Dataset for Developing and Benchmarking Active Vision," in *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*, 2017.

[38] B. Glocker, S. Izadi, J. Shotton, and A. Criminisi, "Real-Time RGB-D Camera Relocalization," in *Proc. IEEE Int. Symp. Mixed and Augmented Reality (ISMAR)*, 2013.

[39] J. Shotton, *et al.*, "Scene Coordinate Regression Forests for Camera Relocalization in RGB-D Images," in *Proc. IEEE/CVF Conf. Computer Vision and Pattern Recognition (CVPR)*, 2013.

[40] P. K. Nathan Silberman, Derek Hoiem and R. Fergus, "Indoor Segmentation and Support Inference from RGBD Images," in *Proc. European Conf. Computer Vision (ECCV)*, 2012.

[41] K. Lai, L. Bo, and D. Fox, "Unsupervised feature learning for 3D scene labeling," in *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*, 2014.

[42] S. Golodetz, *et al.*, "Collaborative Large-Scale Dense 3D Reconstruction with Online Inter-Agent Pose Optimisation," *IEEE Trans. Visualization and Computer Graphics (TVCG)*, 2018.

[43] S. Kohlbrecher, O. von Stryk, J. Meyer, and U. Klingauf, "A flexible and scalable SLAM system with full 3D motion estimation," in *Proc. IEEE Int. Symp. Safety, Security, and Rescue Robotics*, 2011.

[44] W. Hess, D. Kohler, H. Rapp, and D. Andor, "Real-Time Loop Closure in 2D LIDAR SLAM," in *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*, 2016.

[45] J. Zhang and S. Singh, "LOAM : Lidar Odometry and Mapping in real-time," *Proc. Robotics: Science and Systems Conf. (RSS)*, 2014.

[46] S. Sumikura, M. Shibuya, and K. Sakurada, "OpenVSLAM: A Versatile Visual SLAM Framework," *Proc. 27th ACM Int. Conf. Multimedia*, 2019.

[47] D. Schlegel, M. Colosi, and G. Grisetti, "ProSLAM: Graph SLAM from a Programmer's Perspective," *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*, 2018.

[48] J. Engel, T. Schöps, and D. Cremers, "LSD-SLAM: Large-Scale Direct Monocular SLAM," in *Proc. European Conf. Computer Vision (ECCV)*, 2014.

[49] J. Engel, J. Stückler, and D. Cremers, "Large-scale direct SLAM with stereo cameras," in *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems (IROS)*, 2015.

[50] T. Whelan, R. F. Salas-Moreno, B. Glocker, A. J. Davison, and S. Leutenegger, "ElasticFusion: Real-Time Dense SLAM and Light Source Estimation," *Int. Journal of Robotics Research (IJRR)*, 2016.

[51] K. Doherty, D. Fourie, and J. Leonard, "Multimodal Semantic SLAM with Probabilistic Data Association," *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*, 2019.

[52] R. Grompone von Gioi, J. Jakubowicz, J.-M. Morel, and G. Randall, "LSD: a Line Segment Detector," *Image Processing On Line*, 2012.

[53] B. Della Corte, I. Bogoslavskyi, C. Stachniss, and G. Grisetti, "A General Framework for Flexible Multi-Cue Photometric Point Cloud Registration," in *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*, 2018.

[54] I. Ülkü and E. Akagündüz, "A Survey on Deep Learning-based Architectures for Semantic Segmentation on 2D images," *ArXiv*, 2019.

[55] S. Minaee, *et al.*, "Image Segmentation Using Deep Learning: A Survey," *ArXiv*, 2020.

[56] D. Steinkraus, I. Buck, and P. Y. Simard, "Using GPUs for machine learning algorithms," in *Proc. 8th Int. Conf. Document Analysis and Recognition (ICDAR)*, vol. 2, 2005.

[57] R. Girshick, "Fast R-CNN," in *Proc. IEEE Int. Conf. Computer Vision (ICCV)*, 2015.

[58] J. Redmon, S. Divvala, R. B. Girshick, and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," *Proc. IEEE/CVF Conf. Computer Vision and Pattern Recognition (CVPR)*, 2016.

[59] K. He, G. Gkioxari, P. Dollár, and R. B. Girshick, "Mask R-CNN," *Proc. IEEE Int. Conf. Computer Vision (ICCV)*, 2017.

[60] H. Noh, S. Hong, and B. Han, "Learning Deconvolution Network for Semantic Segmentation," in *Proc. IEEE Int. Conf. Computer Vision (ICCV)*, 2015.

[61] R. Raguram, J.-M. Frahm, and M. Pollefeys, "A Comparative Analysis of RANSAC Techniques Leading to Adaptive Real-Time Random Sample Consensus," *Lect. Note. Comput. Sci.*, 2008.

[62] S. Choi, T. Kim, and W. Yu, "Performance evaluation of RANSAC family," *Proc. British Machine Vision Conf. (BMVC)*, 2009.

[63] F. Nardi, B. D. Corte, and G. Grisetti, "Unified Representation and Registration of Heterogeneous Sets of Geometric Primitives," *IEEE Robotics and Automation Letters*, 2019.

[64] L. Carlone, Z. Kira, C. Beall, V. Indelman, and F. Dellaert, "Eliminating conditionally independent sets in factor graphs: A unifying perspective based on smart factors," in *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*, 2014.

[65] S. Agarwal and K. Mierle, "Ceres solver," http://ceres-solver.org.

[66] C. Hertzberg, "A Framework for Sparse, Non-Linear Least Squares Problems on Manifolds," Master's thesis, Universität Bremen, 2008.

[67] A. Dai, *et al.*, "Scannet: Richly-annotated 3d reconstructions of indoor scenes," in *Proc. IEEE/CVF Conf. Computer Vision and Pattern Recognition (CVPR)*, 2017.

[68] J. R. Ruiz-Sarmiento, C. Galindo, and J. González-Jiménez, "Robot@Home, a Robotic Dataset for Semantic Mapping of Home Environments," *Int. Journal of Robotics Research*, 2017.

[69] L. Heng, G. Lee, and M. Pollefeys, "Self-Calibration and Visual SLAM with a Multi-Camera System on a Micro Aerial Vehicle," in *Autonomous Robots*, 2014.

[70] B. Huang, J. Zhao, and J. Liu, "A survey of simultaneous localization and mapping with an envision in 6g wireless networks," in *ArXiv*, 2020.

[71] M. Filipenko and I. Afanasyev, "Comparison of Various SLAM Systems for Mobile Robot in an Indoor Environment," *Proc. Int. Conf. Intelligent Systems (IS)*, 2018.

[72] K. Krinkin, A. Filatov, A. Filatov, A. Huletski, and D. Kartashov, "Evaluation of Modern Laser Based Indoor SLAM Algorithms," *Proc. XXth Conf. Open Innovations Association (FRUCT)*, 2018.

[73] I. Aloise, G. Grisetti, and D. Schlegel, "Extended Measurements in Pose Graph Optimization," Master's thesis, Sapienza University of Rome, 2017.

[74] M. Quigley, "ROS: an open-source Robot Operating System," in *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*, 2009.

[75] L. H. Staib and J. S. Duncan, "Boundary finding with parametrically deformable models," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 14, 1992.

[76] K. He, X. Zhang, S. Ren, and J. Sun, "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification," *Proc. IEEE Int. Conf. Computer Vision (ICCV)*, 2015.

[77] S. C. Wong, A. Gatt, V. Stamatescu, and M. D. McDonnell, "Understanding Data Augmentation for Classification: When to Warp?" in *Proc. Int. Conf. Digital Image Computing: Techniques and Applications (DICTA)*, 2016.

[78] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proc. IEEE/CVF Conf. Computer Vision and Pattern Recognition (CVPR)*, 2015.

[79] A. Hafiz and G. M. Bhat, "A survey on instance segmentation: state of the art," *Int. Journal of Multimedia Information Retrieval*, 2020.

[80] R. Dubé, *et al.*, "SegMap: Segment-based mapping and localization using data-driven descriptors," *Int. Journal of Robotics Research*, 2020.

[81] A. Rosinol, M. Abate, Y. Chang, and L. Carlone, "Kimera: an Open-Source Library for Real-Time Metric-Semantic Localization and Mapping," *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*, 2020.

[82] T.-Y. Lin, *et al.*, "Microsoft COCO: Common Objects in Context," in *Proc. European Conf. Computer Vision (ECCV)*, D. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars, Eds., 2014.

[83] T. Schops, T. Sattler, and M. Pollefeys, "BAD SLAM: Bundle Adjusted Direct RGB-D SLAM," in *Proc. IEEE/CVF Conf. Computer Vision and Pattern Recognition (CVPR)*, 2019.

[84] D. Gálvez-López and J. D. Tardós, "Bags of Binary Words for Fast Place Recognition in Image Sequences," *IEEE Trans. Robotics*, 2012.