

Vector Map-Based, Non-Markov Localization for Long-Term Deployment of Autonomous Mobile Robots

Joydeep Biswas

CMU-RI-TR-14-25

*Submitted in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Robotics*

The Robotics Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

December 2014

Thesis Committee:

Manuela M. Veloso, Chair

Reid Simmons

David Wettergreen

Dieter Fox (University of Washington)

Abstract

As robots become increasingly available and capable, there has been an increased interest in having robots continue to perform autonomously over time despite changes in their environment. This thesis introduces several algorithms for localization of autonomous mobile robots in real human environments with the goal of having them autonomously deployed over extended periods of time.

Monte Carlo Localization with Sampling-Importance Resampling (MCL-SIR) is commonly used for indoor mobile robot localization, with the frequently prescribed suggestion of increasing the number of particles to increase accuracy or robustness. Furthermore, most variants of MCL-SIR sample from the odometry model of a robot, which are far less accurate than modern range sensors. We address both these challenges by introducing Corrective Gradient Refinement (CGR), which, instead of relying on more particles, does more with fewer particles. In particular, it uses the analytically computed state space derivatives of the observation likelihood function to refine the proposal distribution, thus improving the accuracy as well as robustness while requiring fewer particles than MCL-SIR. For robots equipped only with inexpensive depth cameras, we introduce the Fast Sampling Plane Filtering algorithm to extract dominant planar features from observed depth images, to use with CGR.

Going beyond MCL, we recognize that human environments have objects that are either permanent, like walls, movable, like tables and chairs, or moving, like humans. We introduce Episodic non-Markov Localization, which reasons about the nature of such observations, and accounts for correlations between observations even if they are of unmapped objects, to provide location estimates that are accurate globally with respect to the long-term features, as well as locally, with respect to the short-term features. By examining the short-term features detected by the robot over multiple deployments, the robot is further able to build a Model-Instance map of its environment, reasoning about the shapes or models of common movable objects separately from the specific occurrences or instances.

We extensively demonstrate the accuracy and robustness of the localization algorithms introduced in this thesis over a “1000km Challenge”: to deploy a team of robots, over multiple floors of multiple buildings, spanning a duration of a few years. We present quantitative and qualitative results from the 1000km Challenge, and the data collected in the process.

Acknowledgements

While it would be difficult to name all the people who have helped me through the PhD, I nonetheless attempt to thank the few who especially stand out.

I would like to thank my family foremost, especially my parents, for supporting and encouraging me as I chose the potentially suboptimal, high-risk path of travelling far to pursue what I enjoy working on.

I would like to thank my advisor, Manuela Veloso, for all her encouragement, support, infectious enthusiasm, and words of wisdom over the last seven years as she advised my work on multiple robot projects, patiently let me explore a wide range of research topics, and generously supported the construction of a team of four CoBots and a team of twelve CMDragons robots, all for science (and fun).

I thank Reid Simmons, David Wettergreen and Dieter Fox for graciously agreeing to serve on my committee, and providing valuable guidance and feedback for the thesis.

I would like to thank the present and past members of the CORAL research groups, in particular Mike Licitra (thank you for the amazing robots!), Stefan Zickler (the CMDragons lab still misses you), Brian Coltin (wouldn't have had deployable CoBots without you) and Stephanie Rosenthal (the CoBots thank you for their people-skills!). I would also like to thank Aijun Bai, Susana Brandão, Tom Charlie, Benjamin Choi, Fatma Faruq, Steven Klee, Tom Kollar, Max Korein, Somchaya Liemhetcharat, Juan Pablo Mendoza, Çetin Meriçli, Tekin Meriçli, Michael Murphy, Tiago Pereira, Vittorio Perera, Prashant Reddy, Mehdi Samadi, Grant Strimel, Yichao Sun, Junyun Tay, Felipe Trevizan, Rodrigo Ventura, Richard Wang, and Danny Zhu. I would like to thank the members of the NSF-SMRobots group for the many discussions on robots, and in particular Eleanor Avrunin and Heather Knight.

I am very grateful to have a band of friends to hang out, discuss ideas, share meals, and commiserate over life as grad students. The “dinnertrain”, as it were, consists of Peter Anderson-Sprecher, Rika Antonova, Eleanor Avrunin, Brian Becker, Nathan Brooks, Ellen Cappel, Michael Furlong, Brina Goyette, Pyry Matikainen, Umashankar Nagarajan, Heather Jones, Heather Justice, Prasanna Velagupudi, and Rachel Vistein. Thank you, Theresa, Frankie and Shadow Fab-

rizio, for the welcome breaks from work. I would also like to thank Laura Brocklebank, Monica Garcia, and the many alums of the Animal Rescue League for many hours of puppy play-time.

Finally, thank you, Neontetra, Barracuda, Pleco, Bluefin, CoBots1-4, and CMDragons robots 0-A for the countless CPU-cycles you've spent running my code, scoring goals, and running unreasonably long 1000km marathons!

Contents

1	Introduction	1
1.1	Thesis Question	2
1.2	Approach	3
1.3	Contributions	6
1.4	Reading Guide to the Thesis	7
2	Background and Related Work	11
2.1	Markov Localization	11
2.2	Extensions to Monte Carlo Localization	13
2.3	Simultaneous Localization and Mapping	14
2.4	The Data Association Problem	15
2.5	Non-Markov Nature of Observations	16
2.6	Varying Environments	17
2.6.1	Dynamic Pose Graph SLAM	17
2.6.2	Independent Markov Chain Occupancy Grid Maps	18
2.6.3	Temporary Maps	18
2.6.4	Dynamic Maps	19
2.6.5	Patch Maps	19
2.6.6	Hierarchical Object Maps	20
2.6.7	Detection and Tracking of Moving Objects	20
2.6.8	Mapping of Objects	20
2.7	Autonomous Service Mobile Robots	21
2.8	Contrast with Related Work	22
2.8.1	Corrective Gradient Refinement	22
2.8.2	Episodic non-Markov Localization	22

3	Sensors and Sensor-Specific Maps	25
3.1	Odometry	25
3.2	Range Sensors	27
3.2.1	Laser Rangefinders	28
3.2.2	Depth Cameras	28
3.3	Processing Depth Images	29
3.3.1	Fast Sampling Plane Filtering	30
3.3.2	Obstacle Scan	32
3.4	WiFi Signal Strength Sensing	32
3.5	Map Representations	34
3.5.1	Vector Maps	34
3.5.2	Graph-WiFi Maps	36
3.6	Summary	37
4	Markov Localization	39
4.1	Particle Filters for Mobile Robot Localization	39
4.2	Corrective Gradient Refinement for Localization	40
4.2.1	The Corrective Gradient Refinement Algorithm	41
4.2.2	Vector Map Observation Models	44
4.2.3	Experimental Results	46
4.3	Graph-Map WiFi Localization	52
4.3.1	WiFi Observation Model and Constrained Particle Filter	52
4.3.2	Experimental Results	57
4.4	Multi-Sensor Localization for Diverse Environments	58
4.4.1	Multi-Sensor Localization	59
4.4.2	Comparing Localization Algorithms	60
4.5	Summary	61
5	Episodic Non-Markov Localization	63
5.1	The Varying Graphical Network Representation	64
5.2	Episodes in Non-Markov Localization	65
5.3	Classification of Observations	67
5.3.1	Classification of Long-Term Features	67
5.3.2	Classification of Short-Term Features	69
5.3.3	Classification of Dynamic Features	69
5.4	Cost Function Representation of Belief	70

5.5	Computation of Sub-Cost Functions	73
5.6	Structure Analysis	74
5.7	Results	79
5.7.1	Freiburg-Parkinglot Dataset	80
5.7.2	Örebro-Longterm Dataset	81
5.7.3	Deployments of the CoBots	82
5.8	Summary	83
6	Updating Maps	89
6.1	Updating Plane-Polygon Maps	89
6.1.1	Plane Polygon Construction and Merging	90
6.1.2	Experimental Results	93
6.2	Model-Instance Object Mapping	98
6.2.1	Finding Object Instances	98
6.2.2	Building Object Models	100
6.2.3	Updating the Model-Instance Map across Deployments	103
6.2.4	Results	105
6.3	Summary	105
7	The 1000km Challenge	109
7.1	The Scope of Deployments	110
7.2	Data Collected	111
7.3	Quantitative Results	114
7.3.1	Accuracy Compared to Scan Matching	115
7.3.2	Accuracy Compared to Sparse Ground-Truth	120
7.3.3	Robustness	121
7.4	Lessons Learned	126
7.4.1	Sensors	126
7.4.2	Localization	126
7.4.3	Navigation	127
7.4.4	System Integration	127
7.5	Summary	128
8	Conclusion and Future Work	129
8.1	Contributions	129
8.2	Future Work	131

8.3	Summary	132
A	Maintaining Perpetual Deployability of Mobile Robots	133
A.1	Software Architecture	133
A.2	Software Automation	135
A.2.1	Startup Automation	135
A.2.2	Log Management Automation	136
A.3	Log Explorer	138
A.4	Source Code Management	138

List of Figures

1.1	CoBots 1,2,3 and 4	5
1.2	Organization of the thesis	9
2.1	Dynamic Bayesian network for Markov localization.	12
2.2	Non-Markov nature of robot observations	16
3.1	Results of FSPF on a sample image	31
3.2	Example obstacle scan generated from a depth image	33
3.3	Analytic ray casting line occlusion cases	36
4.1	Error in localization using MCL-SIR vs. CGR	47
4.2	Confidence Intervals for localization using MCL-SIR vs. CGR	48
4.3	Success rates of the CGR and MCL-SIR algorithms	48
4.4	Tradeoff between CGR localization error and run time	49
4.5	Comparison of location estimates using FSPF v.s. other approaches	50
4.6	Failure rates over time for FSPF v.s. other approaches	51
4.7	Localization error using FSPF v.s. other approaches	51
4.8	Uncertainty of WiFi localization	56
4.9	Variation in Wifi Localization Error with visible access points	56
4.10	Locations visited by robot using WiFi localization	57
4.11	WiFi localization uncertainty over time	58
4.12	Cumulative histogram of WiFi localization error	58
4.13	Errors in localization using different sensors	61
4.14	Open-area localization errors using different sensors	62
5.1	An example Varying Graphical Network for non-Markov localization	64
5.2	Example episode in non-Markov localization	66
5.3	Classification of observations by EnML	68
5.4	Example robot scenario running EnML	76

5.5	Sample trajectory estimated by EnML from Freiburg-Parkinglot dataset	83
5.6	Sample DFs classified by EnML in Freiburg-Parkinglot dataset	84
5.7	Location estimates using EnML on Orëbro-Longterm dataset	85
5.8	Snapshots of EnML running in the GHC4 atrium	86
5.9	Snapshots of EnML running on GHC5	87
6.1	Correspondence matching between depth images and polygon maps	92
6.2	Errors compared to ground truth for planar polygon mapping without clutter . . .	94
6.3	Cumulative histogram of errors for planar polygon mapping without clutter . . .	95
6.4	Errors compared to ground truth for planar polygon mapping with clutter	95
6.5	Cumulative histogram of errors for planar polygon mapping with clutter	96
6.6	Planar polygon mapping test scenes	97
6.7	Model-Instance Object Mapping, Step 1: STfs estimated by EnML	99
6.8	Model-Instance Object Mapping, Step 2: Observation clustering	99
6.9	Model-Instance Object Mapping, Step 3: Model building	100
6.10	P-R curves for object similarity classification	102
6.11	Alignment and similarity comparison grid between four object instances	103
6.12	Model-Instance Object Mapping, Step 4: Similarity graph construction	104
6.13	Model-Instance Object Mapping, Step 5: Instance cataloging	104
6.14	Door states, as discovered by Model-Instance Object Mapping	106
6.15	Example objects discovered by Model-Instance Object Mapping	107
7.1	The CoBot task scheduling website	111
7.2	The CoBot remote telepresence website	112
7.3	The on-board task scheduler on CoBot	113
7.4	An example email from CoBot4, asking for assistance	113
7.5	Localization and navigation in varied scenarios	114
7.6	Reliable robot positioning at task locations	115
7.7	Progress of the 1000km Challenge	115
7.8	Combined traces of the paths traversed by the CoBots over the 1000km Challenge	116
7.9	Histogram of scan matching errors on each floor during the 1000km Challenge .	118
7.10	Scatter plots of localization errors evaluated by scan matching	119
7.11	Scatter plots of localization errors over all maps during the 1000km Challenge .	120
7.12	Histogram of errors evaluated by sparse ground truth on each map	122
7.13	Scatter plots of localization errors evaluated by sparse ground truth	123
7.14	Distance traversed before first operator intervention	125

7.15	Distance traversed between successive operator interventions	125
A.1	CoBot Software Architecture	134
A.2	A synopsis email generated by the nightly processing script	137
A.3	Log Explorer website	139
A.4	Log Explorer website, showing details of a single deployment log	140
A.5	The Save The Robots Fund	142

List of Tables

2.1	Comparison of CGR with related work	23
2.2	Comparison of EnML with related work	24
3.1	Commonly used range sensors for autonomous robots	27
3.2	Configuration parameters for FSPF	31
4.1	Accuracy of WiFi localization	56
4.2	Open-area localization errors using different sensors	61
5.1	EnML Jacobian and information matrix for example robot scenario	77
5.2	EnML Jacobian and information matrix for no-map example robot scenario . . .	79
5.3	Pose graph SLAM Jacobian and information matrix for example robot scenario .	80
5.4	EnML Jacobian and information matrix for all-mapped example robot scenario .	81
5.5	Localization errors using EnML vs. other approaches	82
5.6	Fraction of different floors traversed by EnML	83
6.1	Results of planar polygon mapping from test scenes	96
6.2	Unique object models discovered by Model-Instance Object Mapping	105
7.1	Cumulative data from the 1000km Challenge	114
7.2	Distances traversed by each robot during the 1000km Challenge	117
7.3	Deployments per floor during the 1000km Challenge	117
7.4	Localization errors evaluated by scan matching at Landmark Checkpoints	118
7.5	Localization accuracy by sparse ground truth	121
7.6	Operator interventions over the 1000km Challenge	121
7.7	Mean distance traversed between interventions using CGR and EnML	124

Chapter 1

Introduction

We seek the ultimate goal of having ever-present autonomous mobile robots in our everyday human environments, performing user-requested tasks on demand. In order to be able to meaningfully perform user-requested tasks, autonomous mobile robots need to be able to reliably localize and navigate in the environment.

A mobile robot relies on observations of its surroundings, estimates of its ego-motion, and an internal model or *map* of the environment, to reason about its location in the environment. The problem of mobile robot localization thus involves reasoning about how the actual observations of the environment relate to the map, how to incorporate the estimates of ego-motion with updates based on the observations, and how to reason about the uncertainty in the location estimates due to imperfect sensing and ego-motion estimates, and limited observability of the environment.

Human environments exhibit significant variations in the types of objects that are observable to robots. Office corridors may have abundant visible permanent features and few moving or movable objects. Open areas, like atria and cafés, have numerous moving obstacles like humans, carts and other robots, as well as movable objects, like chairs, tables and bins. Such fine details, like the exact placements of chairs, tables, trash cans, planters, and other movable objects, are impossible to know for the entire environment, for arbitrary times. Therefore, in this thesis we introduce a map representation that ignores fine details: the map will represent only the permanent features in the environment as a set of lines, which we term a *vector map* representation.

Human environments also change over time. The exact locations of movable objects vary continuously, and in general, such variations are not periodic. Thus, it is impossible to predict what the state of the environment will be like in the future, based on its present and past states. This is the challenge in *long-term* deployments of mobile robots: the environment is bound to change, and the robot must remain robust and accurate in its localization even when the objects that it observes move in unpredictable ways over time. Long-term deployments are thus in stark

contrast to short-term deployments, where a robot may capture an instantaneous snapshot of the state of the environment, and may assume that all its observations will be explainable by the instantaneous snapshot.

To date, considerable work has been done on mobile robot localization, and the approaches proposed commonly assume the Markov independence of the observations: that given a map of the environment and an estimate of the robot’s location, all the observations of the robot can be explained by the map, and are thus independent of past observations. Such approaches are thus broadly categorized under the heading of *Markov Localization*. However, as we have observed, human environments change over time, and a robot’s observation of movable objects will not always match the map. Therefore, in this thesis we propose to relax the assumption of the Markov independence of observations, and introduce *non-Markov Localization*, which explicitly reasons about the correlations between observations from different time-steps due to the presence of unmapped, movable objects in the environment.

This thesis thus culminates in the introduction of a vector map-based non-Markov localization algorithm for the long-term deployments of autonomous mobile robots. However, in the process, we also made significant contributions to the map representations for mobile robot localization, and introduced several variants to Markov Localization. In particular, driven by the need to accurately and robustly localize a mobile robot while simultaneously minimizing the computational requirements of localization, we introduced algorithms to efficiently reason with depth image observations, to efficiently and accurately update location estimates, and to utilize different sensor modalities. Therefore, the contributions of the thesis include several complete localization algorithms that are suited to human environments.

1.1 Thesis Question

This thesis seeks to answer the question,

How can a robot with *sensor limitations robustly* and *accurately* estimate its location while deployed over *extended periods of time* in a *real-world varying human environment*?

Sensor limitations are inherent to robot design, since long-range, wide field-of-view, low-noise sensors are expensive. Therefore, localization algorithms on deployed robots need to appropriately handle the uncertainty and errors arising from the inexpensive sensors at hand.

Localization algorithms need to be **robust** to unexpected sequences of observations, action uncertainty, as well as changes in the environment. Furthermore, they need to maintain **accurate** location estimates of the robot in order to effectively perform tasks.

We seek to investigate localization algorithms that in particular can robustly and accurately localize autonomous robots deployed over **extended periods of time**, extending over years of deployments. Such deployments would be in **real-world varying human environments**, characterized by a variety of operating conditions, moving objects, and frequent changes to the environment.

While this thesis introduces algorithms for long-term deployments of mobile robots in varying environments, the same algorithms are equally applicable for short-term deployments, or deployments in static environments. Furthermore, the algorithms introduced will be empirically evaluated using data collected from several different types of robots, both indoors and outdoors.

1.2 Approach

The thesis question is investigated from a number of fronts, tackling the localization problem in a number of different approaches.

Mobile robot localization is a well-researched problem, and there currently exist many proposed algorithmic solutions. Nearly all such algorithms are variants of Markov Localization [35]. Markov Localization assumes that given a map of the environment and the latest pose estimate, observations made by a robot are independent of the past observations. This assumption of Markov independence between observations provides tractability to online localization algorithms, and has experimentally been shown to be largely successful. Therefore, in this thesis we first explore extensions to Markov Localization with the goal of developing accurate, robust localization algorithms for use in human environments with limited movable or unmapped objects.

We investigate a 2D vector map representation of the environment, where features such as walls are represented as 2D line segments. We develop an observation likelihood model to efficiently predict the observable features of a map given a robot pose estimate by a novel analytic ray-casting algorithm. The observation likelihood function can thus be analytically computed, along with its state space derivatives. By using these analytically computed state space derivatives of the observation likelihood function, we develop an algorithm aimed to reduce the computational requirements of Monte Carlo Localization (MCL) by using fewer samples, but *refining* the samples to better account for the observations.

The next challenge we explored is that of localization using the recently introduced inexpensive depth cameras. Laser rangefinders have long been accepted as the *de-facto* sensors of choice for mobile robot localization, but they are expensive and do not provide feedback of the world in 3D. Recently, a number of depth cameras, like the Microsoft Kinect sensor and the PrimeSense

Sensor, has become available at less than one tenth the cost of the cheapest laser rangefinders, which makes them economically attractive as robot sensors. However, there are challenges to using depth cameras for mobile robot localization. Depth cameras typically generate voluminous data that cannot be processed in its entirety in real time for localization. Furthermore, inexpensive depth cameras have a much narrower field of view, shorter range, and are noisier compared to laser rangefinders. We tackle these challenges in a two-fold approach: We first introduce a novel plane filtering algorithm that samples the depth image to produce a set of points corresponding to planes, along with the plane parameters (normals and offsets). The volume of data to be processed is thus significantly reduced, and non-planar objects in the scene, which are unlikely to correspond to the map, are filtered out. We then introduce an observation model in our localization algorithm that matches the plane filtered points to the existing 2D vector maps. The two main objectives in developing these new algorithms are *speed* and *reuse*. The plane filtering algorithm that we introduce is capable of running orders of magnitude faster than the sensor actually provides frames, thus minimizing latency and computational requirements. By re-using the same 2D vector map representation, our overall localization algorithm still retains its benefits: fast analytic evaluations of the observation likelihood function, and its state space gradients.

Using the localization algorithms developed thus far, we successfully deployed our robots, across multiple floors and multiple buildings to perform user-requested tasks. The CoBots successfully traversed hundreds of kilometers in our indoor environments, to a large degree thanks to their robust localization algorithms. However, part of the success was due to the scope of the deployments. Deployments rarely extended to open areas where moving and movable objects comprised the majority of the robots' observations. We quickly realized that in order to be able to successfully deploy the robots in environments that exhibited significant changes over time, the localization algorithms would have to reason better about the unexpected observations.

Previous approaches to robot localization in changing environments built new updated maps of the environment. We sought an alternate solution that did not rely on accurate up to date maps, but instead reasoned about whether the observations of the robot arose from long-term, short-term, or dynamic features. We introduce the long-term features as the unchanging parts of the environment, like the walls and columns; the short-term features as the movable objects like chairs, which would change locations over time, but might be perceived to be static by the robot for a short period of time; and dynamic features as moving objects, which the robot would perceive as actively changing locations across time steps. By accounting for the correlations between observations and the map due to long-term features, between observations from different time steps due to short-term features, and between consecutive poses due to robot odometry,

the robot could use all its observations, both expected and unexpected, to better estimate its location in a varying environment. Since this approach involves explicit correlations between observations from different time steps, it is inherently non-Markovian in nature, and we explore a few strategies to limit the history, or *episodes* of such non-Markov observations.

This thesis includes extensive experimental evaluation of the localization algorithms developed. We demonstrate the robots performing tasks in a variety of environments, over a time period spanning four years, and using the different localization algorithms developed in this thesis. The latest approach, Episodic non-Markov Localization, is capable of handling wide variations in the environment, while the Markov localization variants are less successful in open areas, but are still successful for robot deployments in areas with little or no variations over time.

The robots, which we use for the experimental results in this thesis, are the CoBots, or *Collaborative Robots*. The CoBots are custom-built robots¹ with four-wheel omnidirectional drive bases and off-the-shelf tablet computers as the computational platforms to run all the algorithms necessary for them to operate autonomously. We have four CoBots in total, shown in Figure 1.1, which were used for the experiments in this thesis including the 1000km Challenge. The CoBots are primarily service mobile robots, and perform user-requested tasks allocated by a Multi-agent Pickup and Delivery Planner with Transfers [25], and interact with humans in the environment for Symbiotic Autonomy [73].



Figure 1.1: CoBots 1, 2, 3 and 4, which were used for the experiments in the thesis.

¹Thanks to Mike Licitra (mikelicitra@gmail.com) who designed and built the robots.

1.3 Contributions

The key contributions of this thesis are as follows.

MCL-CGR for accurate and robust localization with low computational cost in indoor human environments with limited variations over time. We introduce Monte-Carlo Localization with Corrective Gradient Refinement (MCL-CGR) to use an analytic sensor likelihood function with a vector map representation of the environment. Instead of directly using samples from the odometry model of the robot as the proposal distribution of MCL, MCL-CGR first refines the samples using the analytically computed state space gradients of the observation likelihood function before using them as the proposal distribution for MCL. This results in more efficient sampling of the robot’s localization belief by sampling sparsely along directions of low uncertainty, while densely sampling along directions of high uncertainty. We experimentally show that MCL-CGR thus has greater localization accuracy, lower variance, and lower computational requirements than Monte-Carlo Localization with Sampling-Importance Resampling (MCL-SIR).

FSPF for plane-filtering of depth images, and FSPF-MCL-CGR for localization with inexpensive depth cameras in human indoor environments with limited variations over time. We introduce the Fast Sampling Plane Filtering (FSPF) algorithm that samples the depth image to produce a set of points corresponding to planes, along with the plane parameters (normals and offsets). We then introduce an observation model that matches the plane filtered points to the existing 2D vector maps. Finally, we use MCL-CGR to localize the robot using the plane filtered points. We experimentally demonstrate that a robot with an inexpensive depth camera running MCL-CGR with FSPF to accurately and robustly localize it in common office environments with comparable results to localization using MCL-CGR and the much more expensive laser rangefinders.

EnML for accurate and robust localization in varying environments with many moving objects and frequent changes. We introduce Episodic non-Markov Localization (EnML) to localize a robot in environments with many dynamic and movable objects. EnML classifies observations into long-term, short-term, and dynamic features. Long-term features are matched with the permanent map, and provide global, absolute localization corrections, while short term features observed from different time steps provide local, relative localization corrections. The Varying Graphical Network (VGN), a novel graphical model that we introduce, is used to keep

track of correlations between successive poses based on odometry, between pairs of poses from short-term features, and between each pose and the map from long-term features. We introduce a cost function representation of the belief to efficiently compute the maximum likelihood estimate of the robot's location by performing non-linear least-squares optimization of the cost function. We demonstrate EnML accurately and robustly localizing robots even in environments which differ substantially from the map due to unexpected observations of dynamic and movable objects.

Extensive experimental results of over 1000km traversed by robots using the proposed localization algorithms. We present results from continual deployments of four robots, CoBots 1-4 in real academic buildings at Carnegie Mellon University. The robots were deployed in multiple buildings, over multiple floors, and exposed to a variety of indoor environments.

1.4 Reading Guide to the Thesis

Figure 1.2 presents an overview of the chapters of the thesis, and the outline below summarizes the chapters:

Chapter 2 - Background and Related Work We define the problem of mobile robot localization and the terms used. We discuss previous work in a number of areas including Markov Localization, simultaneous localization and mapping, the data association problem, localization and mapping approaches for varying environments, and autonomous service mobile robots.

Chapter 3 - Sensors and Sensor-Specific Maps We describe the sensors that we shall use: WiFi, laser rangefinders, depth cameras, and wheel encoders for odometry. We introduce the algorithms for processing sensor data including plane filtering and obstacle scan generation. Furthermore, we introduce the sensor-specific map representations that will be used by the localization algorithms introduced in subsequent chapters: a topological graph representation for a WiFi map, and a vector map representation for range sensors.

Chapter 4 - Markov Localization We introduce several extensions of Markov localization. First, we introduce Corrective Gradient Refinement, which takes into account the state space derivatives of the observation likelihood functions to refine the proposal distributions of particle filters. We also introduce a projected 3D point cloud model to use depth cameras for localization by extracting planes from observed depth images and relating them to the vector map. Next, we

introduce a WiFi localization algorithm that uses a constrained particle filter to localize a robot on a WiFi map. Finally, we present the tradeoffs in accuracy and robustness between each of these algorithms across different types of environments.

Chapter 5 - Episodic non-Markov Localization We introduce an approach to localization that classifies observations into long-term, short-term, and dynamic features, and represents the varying correlations between such observations from different time steps with a new graphical model, the Varying Graphical Network. We introduce the concept of *episodes* to reason about the history of observations that will be required to be maintained in order to correctly update the belief. Representing the belief in terms of a cost function, we introduce the complete Episodic non-Markov Localization algorithm that maintains the belief of localization over the latest episode of observations, while explicitly reasoning about correlations between observations from different time steps due to unexpected observations that are not explained by a long-term static map.

Chapter 6 - Updating Maps We present algorithms to update maps in varying environments by processing observations from logs across multiple robot deployments. We present how a robot may model the environment in 3D using a compact 3D polygon map by mapping the dominant planes observed. Furthermore, we present how a robot may reason about movable objects by *Model-Instance Object Mapping*, which separately reasons about the shapes of common objects from the duplicate instances of the objects observed at different locations.

Chapter 7 - The 1000Km Challenge We present experimental results of the localization algorithms introduced, over deployments with multiple robots, in multiple buildings, and spanning four years of deployments.

Chapter 8 - Conclusion and Future Work We conclude the dissertation with a summary of its contributions, and expected directions of future work leading from this thesis.

Chapter A - Maintaining Perpetual Deployability of Mobile Robots The thesis further includes an appendix, which describes the procedures, automation, and methodology involved in ensuring that all the robots remain functional all the time.

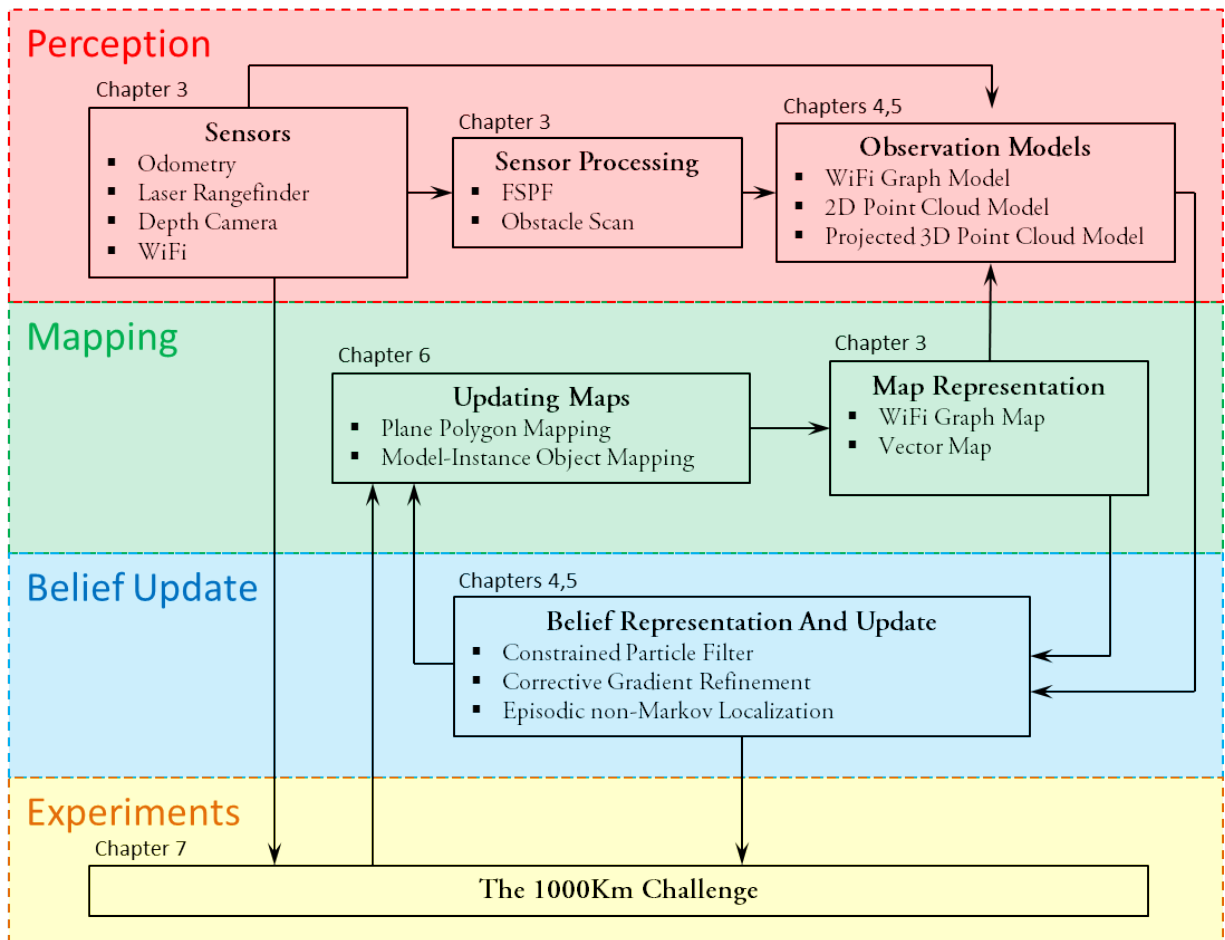


Figure 1.2: Organization of the chapters in the thesis.

Chapter 2

Background and Related Work

The field of localization and mapping for robots is vast - many approaches have been proposed, using various sensors, observation models, map features, and update algorithms. This chapter is not intended to be an exhaustive review of all existing localization and mapping algorithms. In this chapter we review the background related to this thesis, and detail select relevant approaches.

Suppose a robot makes observations of the environment, $S = \{s_1, \dots s_n\}$ and accumulates robot odometry $U = \{u_1, \dots u_n\}$ over n time-steps $t_1 \dots t_n$. We follow the convention that at time t_i , observation s_i is made, and odometry u_i indicates relative motion of the robot from timestep t_{i-1} to t_i . Given a prior map of the environment M , the problem of mobile robot localization is then stated as estimating the probability distribution over the robot pose at the latest time step x_n , or the “belief” Bel ,

$$Bel(x_n) = P(x_n | s_1, \dots s_n, u_1, \dots u_n, x_0, \dots x_{n-1}, M). \quad (2.1)$$

This equation is general to all localization algorithms, assuming only that there exists a prior map environment M .

2.1 Markov Localization

Markov Localization [35] makes two independence assumptions in order to simplify the computation of the belief:

1. **Markov independence of odometry:** Given the estimate of the robot pose x_{n-1} and the latest odometry u_n , the estimate of x_n is independent of past pose estimates $x_0, \dots x_{n-2}$

and odometry u_1, \dots, u_{n-1} , so that

$$\begin{aligned} P(x_n | s_1, \dots, s_{n-1}, u_1, \dots, u_n, x_0, \dots, x_{n-1}, M) \\ = P(x_n | x_{n-1}, u_n). \end{aligned} \quad (2.2)$$

2. **Markov independence of observations:** Given the estimate of robot pose x_n , observation s_n is independent of past pose estimates x_0, \dots, x_{n-1} , observations s_1, \dots, s_{n-1} and odometry u_1, \dots, u_n , so that

$$\begin{aligned} P(s_n | s_1, \dots, s_{n-1}, u_1, \dots, u_n, x_0, \dots, x_n, M) \\ = P(s_n | x_n, M). \end{aligned} \quad (2.3)$$

Using the Markov assumptions and applying Bayes rule, the recursive update of the belief simplifies to (see [37] for a complete derivation),

$$\begin{aligned} Bel(x_n) = \\ \eta P(s_n | x_n, M) \int P(x_n | x_{n-1}, u_n) Bel(x_{n-1}) dx_{n-1}, \end{aligned} \quad (2.4)$$

where η is a normalizing constant. Figure 2.1 shows the dynamic Bayesian network (DBN) for Markov localization. Based on this DBN, for the “prediction” step (before s_n becomes available) the Markov blanket [70] of x_n consists of u_n (which is observed) and x_{n-1} (estimated by the belief $Bel(x_{n-1})$). For the “update” step, the Markov blanket of s_n consists of x_n (estimated by the predict step), u_n (which is observed), x_{n-1} (estimated by the belief $Bel(x_{n-1})$) and the map M (which is known). Thus the belief update for $Bel(x_n)$ does not require storing the history of observations and states prior to t_{n-1} .

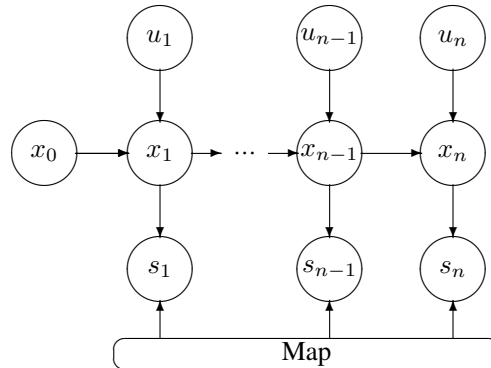


Figure 2.1: Dynamic Bayesian network for Markov localization.

By varying the belief representation, the map representation, how the update steps are computed, and what sensors are used, numerous variants of Markov localization have been proposed. The belief could be represented by Position Probability Grids [35], Partially Observable Markov Decision Processes (POMDPs) [48], Normal distributions [52], or discrete samples called “particles” in Monte Carlo Localization [29].

2.2 Extensions to Monte Carlo Localization

Monte Carlo Localization (MCL) [29] has been shown to be effective at localizing a robot in a number of scenarios. However, one limitation of MCL is that it requires a very large number of particles to simultaneously *accurately* and *robustly* track the location of a robot. Several extensions to MCL have therefore been proposed to improve its accuracy while using fewer particles, by better accounting for the observations during the sampling step of MCL.

KLD-Sampling [36] adapts the number of samples based on the Kullback-Liebler Distance between the belief and the posterior, thus reducing the computational requirements when the belief accurately models the posterior, and increasing the number of particles when the posterior distribution is likely to spread out. However, this approach does not refine the samples based on observations, but only varies the number of samples.

Sensor Resetting Localization (SRL) [50] takes into account mismatch between observations and the belief by drawing samples from the observation model, $p(s_t|x_t)$ to incorporate into the belief. However, this assumes that the observation model can be sampled efficiently over the map, which may not be possible (*e.g.*, for observation models of laser rangefinder scans on a map).

Monte Carlo Markov Chain (MCMC) [57, 43] based approaches, and in particular, the Hybrid Monte Carlo (HMC) filter [31, 23] have been shown to be successful at maintaining the belief with small sample sets, even in large dimensional state spaces. In the HMC filter, the posterior $p(x_t|s_{1:t}, u_{1:t-1})$ is modeled as a hypothetical potential energy term $E = -\log(p(x_t|s_{1:t}, u_{1:t-1}))$, and for each sample x_t^i , a Markov Chain (MC) is evolved using Hamiltonian dynamics over the Hamiltonian $H = E + K$ where K is the potential energy of the sample, $K = \frac{1}{2}\dot{x}^T\dot{x}$. The disadvantages of this method are that it is slow (the MCMCs requires a minimum number of steps of evolution before they reach equilibrium), and it assumes that the state space gradient of the full posterior, $\frac{\delta}{\delta x}p(x_t|s_{1:t}, u_{1:t-1})$ can be computed, which is often impossible or computationally infeasible.

The Auxiliary Particle Filter (APF) [71] improves the proposal distribution by “simulating” the evolution of the samples x_t^i using the observation s_t to generate a new sampling x_{t-}^i , which is

then used to generate samples x_t^i using the motion model. Since the simulation step is performed prior to applying the motion model, the simulation model must account for the motion model as well.

In FastSLAM 2.0 [60], the proposal distribution is refined by linearizing the observation model with respect to pose and landmark locations, and then performing a single EKF-like update step. The GMapping [41] algorithm refines its proposal distribution by sampling the observation likelihood around every particle, and fitting a Gaussian to the sampled points. The proposal distribution is then updated using this Gaussian estimate.

2.3 Simultaneous Localization and Mapping

So far we have assumed that a map M of the environment exists, and is used for robot localization. This map could be built by collecting sensory data and batch processing it offline, or (as is more common in recent work) online on the robot while simultaneously tracking the location of the robot on the map. “Simultaneous Localization And Mapping” (SLAM) is the problem of building a map M while concurrently maintaining a belief of the robot’s trajectory x_0, \dots, x_n .

The DBN for SLAM with a static map is similar in structure to that of Markov localization. The difference is that in Markov localization, the map M is known and only the latest pose estimate x_n is updated at every step, but in SLAM the map M is unknown a priori, and is updated at every step, along with the full history of pose estimates x_0, \dots, x_n . Despite the updates to the history of pose estimates, the model is still Markovian: it is assumed that were x_{n-1} and u_n accurately known, x_n would be independent of all other prior states and observations, and that were x_n and M accurately known, observation s_n would be independent of all other prior states and observations. Bailey and Durant [4, 32] review some of the popular approaches to SLAM. Pose graph SLAM [88] frames the SLAM problem as a graph optimization problem where robot poses are represented as nodes on a graph, and edges joining the nodes indicate relations between them based on odometry or observation matching. “Factor Graphs” [30] offer an alternative representation of the correlations between the pose history and the map in SLAM, replacing measurements by “factor nodes”. Some approaches split up the SLAM problem into algorithms for building a pose graph (*e.g.*, by scan matching [67, 54]) and algorithms for efficiently updating and optimizing the pose graph by iterative non-linear optimization [68].

2.4 The Data Association Problem

Whether localizing a robot on a pre-built map or building a map while concurrently localizing the robot online, the robot needs to reason about which parts of its observations correspond to which parts (or features) of the map, or which parts of its latest observation match which parts of past observations. This is called the “Data Association” problem. Suppose the map M consists of m features, and observation s_i for timestep t_i has n_i features. The total possible number of feature pair associations that need to be considered for timestep t_i are $m \times n_i$. For time-steps t_0 to t_i , the total combination of possible associations is $\prod_{j=0}^{i-1} m \times n_j$, which is $O((m \times N)^i)$ where N is the maximum number of features in any observation s_j . Given the exponential nature of the number of the space of possible data associations, it is infeasible for any algorithm to reason about *all* possible associations. There are three most commonly used approximations to the data association problem that are most commonly used by localization and mapping algorithms today:

1. **Nearest Neighbor:** For every feature in s_i , the single feature from M that is closest to it is chosen as its match.
2. **Individual Compatibility:** For pose x_i , the uncertainty in x_i is used to compute a matching threshold for each feature independently in s_i . All matches to features in s_i from M within the corresponding thresholds are considered.
3. **Joint Compatibility:** For pose x_i , the uncertainty in x_i is used to compute a joint matching threshold for all features in s_i considered jointly. The joint threshold is then used to assign matches to the features in s_i from M .

The nearest neighbor test is the simplest to perform, since it does not require the computation of any uncertainty estimates. The individual compatibility test requires computation of covariances between x_i and individual features from s_i , and the joint compatibility test requires computation of the joint covariance between x_i and all features in s_i . An algorithm for recovering the exact covariance matrix for compatibility tests is discussed in [46]. The joint compatibility test is implemented most commonly in a branch and bound algorithm [61].

Due to the computational cost of the joint compatibility test, in practice it is rarely used for online algorithms. Instead, either the nearest neighbor or individual compatibility tests are performed online to reason about local data associations, and periodic “loop closure” checks are performed to reason about global data associations. “Scan matching” [54] is a widely used algorithm for loop closure detection using rangefinder sensors. Scan matching estimates the optimal relative pose between pairs of arbitrary rangefinder scans by aligning the pair for maximum overlap. Visual features extracted from color images may also be used for loop closure detection [28].

2.5 Non-Markov Nature of Observations

In reality, neither the odometry nor the observations of a robot deployed in a real environment are truly Markovian. In a human environment, as shown in Figure 2.2, observations made by the robot include parts that do indeed correspond to the map, but also include moving objects (*e.g.*, humans and other robots) and movable objects (*e.g.*, tables, chairs, recycling bins) that change locations on a daily basis. Since Markov localization assumes that observations correspond only to features from a static map, unexpected observations of static objects will cause two potential problems:

1. If there are no locations on the map that match the observations, the localization algorithm will be unable to track the location of the robot with low uncertainty.
2. If the observations of movable objects coincidentally match a feature on the map (*e.g.*, if the robot matches an observed flower pot to a pillar on the map), then the localization algorithm will estimate the pose of the robot incorrectly and with overconfidence.

If unexpected observations are explicitly removed from the computation of the belief by filtering the observations [35], then the incorrect and overconfident estimation of pose will be less likely, but the uncertainty of the robot's location will still grow.



Figure 2.2: Non-Markov nature of robot observations, shown by plotting laser rangefinder observations over multiple time steps, and registered by non-Markov localization to a static map (blue lines). The observations include observations that can be explained by the map (orange points), observations that can be explained by movable, but currently static objects like the potted plant and doors (purple points), and moving objects like humans (green points). The robot trajectory is plotted in gray.

2.6 Varying Environments

We introduce the existing related work and contrast it with the proposed approach, by introducing a varying human environment test scenario. Suppose a robot is operating in an open area, and makes the following sequence of observations:

1. Monday, t_0 : The robot observes two cylindrical shaped objects, although the map only has a single pillar near that location.
2. Monday, t_1 : The robot observes more of the cylindrical objects.
3. Monday, t_2 : The robot observes a wall beyond the two cylindrical objects.
4. Tuesday, t_3 : The robot again observes two cylindrical objects, but with different relative locations.
5. Wednesday, t_4 : The robot yet again observes two cylindrical objects with new relative locations.

In this scenario, one of the observed cylindrical objects is an unmapped planter, while the other is a pillar that exists on the robot’s map. At time-steps t_0 , t_3 and t_4 , the robot is unable to discern which of the cylindrical objects is the planter and which the pillar, due to its initial pose uncertainty. We now discuss how selected approaches in related work address this scenario.

2.6.1 Dynamic Pose Graph SLAM

The approach of Dynamic Pose Graph SLAM (DPG-SLAM) [91] is a long-term variant of pose graph SLAM intended to maintain an up-to date map of a changing environment. In addition to the usual pose nodes in pose graph SLAM, DPG-SLAM also maintains indicators for every node to indicate parts of the observations from that node as being “static”, “added”, or “removed”. The indicators for every node are updated after every pass of the robot through the environment. At any given time the latest estimate of the map, called the “active map”, can be recovered by transforming the static and new observations from every pose node to the global reference frame as related by the poses of the nodes. The “dynamic map”, indicating the changes to the environment is similarly recovered by transforming the new and outdated observations from every node to the global reference frame.

In the test scenario, from time-steps t_0 to t_2 , the observations corresponding to the planter are added to both the active as well as dynamic map as “added” observations. The responsibility of disambiguating the pillar from the planter would be borne by pose graph SLAM, but the assignment would not be revisited in future, since existing edges in the pose graph are considered static. The wall and pillar are labelled “static” as their observations have not changed, and hence are only

part of the active map. In t_3 , the older observations of the planter are labeled as “removed”, and the new observations labelled as “added”. The same procedure is performed at t_4 , thus maintaining an up to date estimate of the map. However, at t_3 and t_4 , the observations of the planter are treated entirely independently of the older observations of the planter from time-steps t_0 to t_2 .

2.6.2 Independent Markov Chain Occupancy Grid Maps

Saarinen et al. [78] model a dynamic environment as an occupancy grid with associated independent Markov chains (iMac) with every cell on the grid. Each independent Markov Chain models the transition probabilities of the associated cell for transitioning from unoccupied to occupied, and vice-versa. The iMac occupancy grid thus models the environment by learning the probabilities of each cell remaining occupied vs. unoccupied.

The iMac occupancy grid of the test scenario would add new occupancy values to the grid from time-steps t_0 to t_2 for the planter. The classification of which cylindrical observation corresponded to the planter would be performed once, and would not be re-evaluated again in case of inconsistencies. From time-steps t_3 onwards, the iMac occupancy grid would update the transition probabilities of each individual cell covered by the planter locations to indicate that those cells are likely to transition from unoccupied to occupied as well as vice-versa. The cells occupied by the walls and pillar would be updated to indicate being occupied and having low probabilities of transitioning to being unoccupied. Since each cell in the iMac occupancy grid is considered independently, the planter would not be considered as a whole movable object.

2.6.3 Temporary Maps

The approach of “Temporary Maps” [58] models the effect of temporary objects by performing local SLAM, using the latest global map estimate as an initial estimate for the local map. Using these locally static maps, they perform localization with a particle filter.

In the test scenario, the approach of temporary maps would start off with a global map with the walls and the pillar, and as it performed local SLAM during time-steps t_0 to t_2 , it would add the planter to the local map. However, the problem of disambiguating the pillar from the planter would not be revisited. In time-steps t_3 and t_4 it would again perform local SLAM using the last mapped location of the planter as initial estimates, subsequently removing the old mapped locations of the planter and adding the new locations of the planter.

2.6.4 Dynamic Maps

Dynamic Maps [7] extends SLAM and maintains estimates of the map over several timescales. In order to accommodate changes on the order of different timescales, the approach of Dynamic Maps maintains recency weighted samples of the map at several timescales simultaneously. The state of the environment is saved as multiple local maps at all predefined timescales. During deployments, the robot picks the local map for its current location from that timescale that best matches its present observations.

The approach of dynamic maps in the test scenario would start out with an initial global map consisting of only the walls and the pillar. From time-steps t_0 to t_2 it would add new samples of the locations of the planter to the maps at different timescales. Between the time-steps t_0 to t_4 the problem of disambiguating the pillar from the planter would be visited only once for every time a sample were added. In subsequent time-steps, for localization, the algorithm would select that past map sample that best matched the latest observations. Since the location of the planter changes on a daily basis, the algorithm would be likely to select samples from the maps with the timescale closest to a day. Thus, to account for the locations of temporary objects, the algorithm would require sufficient samples of all possible combinations of locations of all the temporary objects in the environment.

2.6.5 Patch Maps

To account for changes to the environment due to movable objects, one proposed approach maintains multiple maps of the different possible states of the environment [83]. The resulting set of local maps representing observed configurations of the environment is termed a “patch-map”. As the robot localizes in the environment it reasons about the probability of each local patch map as best representing the state of the environment.

In the test scenario, from time-steps t_0 to t_2 the approach of Patch Maps would create a new patch map with the observed location of the planter. At time steps t_3 onwards the algorithm would create new patch maps with the new locations of the planter. The problem of disambiguating the planter from the pillar would be addressed once for every time a new patch map were created. When localizing the robot using past patch maps, the robot would be able to account for the observations of the planter if it had previously created a patch map with the same location of the planter. If there were additional movable objects, the number of patch maps required to represent the possible states of the environment would grow exponentially with the number of such movable objects.

2.6.6 Hierarchical Object Maps

Recognizing that many objects in indoor human environments are of similar shapes, the approach of hierarchical object maps [2] assumes certain classes of shapes of objects that are matched to observed unmapped objects. When unmapped objects are observed and added to the occupancy grid map, the hierarchical algorithm estimates which class the object is most likely to belong to, and then finds the optimal parameters of the object class to best fit the size and orientation of the observed object.

In the test scenario, for each time that the robot encountered the planter object in a new location, the approach of object maps would select that object class that best matched the planter, and would estimate its size. Thus it would be able to reason about subsequent observations of the different sides of the planter on the same day. However, the approach of hierarchical object maps does not address the uncertainty in resolving which observed cylinder was the planter and which the pillar - it would assign the pillar to the closest observation, and not revisit the assignment even if the subsequent observation of the wall did not match the map.

2.6.7 Detection and Tracking of Moving Objects

An alternate view of a dynamic environment is to partition the static elements from the non-static elements, and consider each set separately. The approach of Simultaneous localization and mapping with detection and tracking of moving objects (SLAM + DTMO) [92] is an example of such an algorithm. SLAM + DTMO does not distinguish between LTFs and STFs, but actively tracks DFs while maintaining a map of the environment as the union of LTFs and STFs.

In the test scenario, the approach of SLAM + DTMO would update the map with the new location of the planter every time it encountered it at a new location. SLAM + DTMO would additionally segment out observed moving objects like humans and would avoid adding them to the map.

2.6.8 Mapping of Objects

There have been a few approaches to detecting movable objects in the environment. Recognizing that many objects in indoor human environments are of similar shapes, the approach of hierarchical object maps [2] assumes certain classes of shapes of objects that are matched to observed unmapped objects. The Robot Object Mapping Algorithm [16] detects moveable objects by detecting differences in the maps built by SLAM at different times. Detection and Tracking of Moving Objects [92] is an approach that seeks to detect and track moving objects while

performing SLAM. Relational Object Maps [53] reasons about spatial relationships between objects in a map. Bootstrap learning for object discovery [59] is similar to the “model” part of our work on Model-Instance Object Mapping (Chapter 6.2) in that it builds models of unmapped objects, but does not reason about instances. Generalized Approach to Tracking Movable Objects (GATMO) [38] is an approach to tracking the movements of movable objects that is similar to our work in that it models movable objects. Our work however, further tracks every observed instance of the movable objects, thus allowing it to reason about the most likely poses of objects.

2.7 Autonomous Service Mobile Robots

There have been a number of autonomous service mobile robots deployed in real indoor human environments, which we review here. Shakey the robot [64] was the first robot to actually perform tasks in human environments by decomposing tasks into sequences of actions. Rhino [18], a robot contender at the 1994 AAAI Robot Competition and Exhibition, used SONAR readings to build an occupancy grid map, and localized by matching its observations to expected wall orientations. Minerva [85] served as a tour guide in a Smithsonian museum. It used laser scans and camera images along with odometry to construct two maps, the first being an occupancy grid map, the second a textured map of the ceiling. For localization, it explicitly split up its observations into those corresponding to the fixed map and those estimated to have been caused by dynamic obstacles. Xavier [49] was a robot deployed in an office building to perform tasks requested by users over the web. Using observations made by SONAR, a laser striper and odometry, It relied on a Partially Observable Markov Decision Process (POMDP) to reason about the possible locations of the robot, and to reason about the actions to choose accordingly. Robox [81] was a team of autonomous robots deployed at the Swiss National Exhibition Expo.02 to act as tour guides for visitors. The Robox robots operated in an exhibition area of size $315m^2$ and localized using geometric features extracted from laser rangefinder scans matched to a vector map. A number of robots named Chips, Sweetlips and Joe Historybot [65] were deployed as museum tour guides at the Carnegie Museum of Natural History, Pittsburgh Pennsylvania. Artificial fiducial markers were placed in the environment to provide accurate location feedback for the robots. The PR2 robot at Willow Garage [69] has been demonstrated over a number of “milestones” where the robot had to navigate over 42 km and perform a number of manipulation tasks. The PR2 used laser scan data along with Inertial Measurement Unit (IMU) readings and odometry to build an occupancy grid map using GMapping [42], and then localized itself on the map using KLD-sampling [36]. The RoboCup@Home [90] is a special competition league in the annual RoboCup event, where robots are evaluated over challenges involving navigating

in realistic human environments, manipulating everyday objects, and interacting with humans while performing various tasks.

2.8 Contrast with Related Work

With the preceeding review of the state of the art in mobile robot localization and mapping algorithms, we now contrast our work with the related work, highlighting the key contributions of the thesis.

2.8.1 Corrective Gradient Refinement

Table 2.1 contrasts the features of our approach, Corrective Gradient Refinement (CGR) [8] with the related work. KLD-Sampling varies the number of samples, but does not directly refine the locations of the existing particles in the particle filter. SRL requires a global estimate of the observation likelihood function. HMCs knowledge of the gradients of the full posterior, which is not available in most robot localization problems. APFs similarly require knowledge of the gradient, or predicted evolution of the motion model, which is not available in most robot localization problems. FastSLAM and GMapping both refine the proposal distriction of their particle filters, but FastSLAM assumes a linear model of the gradients, while GMapping fits a Gaussian to the local samples to perform such refinement. CGR, in contrast:

1. Is analytic in nature since it uses a vector map,
2. Assumes no prior model for the local distribution of the observation likelihood function,
3. Actively refines the location estimates of the particles,
4. Is independent of the gradients of the motion model and the posterior, and
5. Does not require any global computations of the observation likelihood function.

2.8.2 Episodic non-Markov Localization

Table 2.2 contrasts the features of our approach, Episodic non-Markov Localization (EnML) [14] with the related work. Markov Localization assumes a static world and does not account for observations of unmapped objects. DPG-SLAM does not reason about movable objects or the difference between permanant and short-term objects. iMac maps explicitly reason about movable objects and, but treat permanent objects similarly to movable objects. Temporary maps reasons about, and distinguishes movable objects from permanent objects, but is a stochastic

	ANA	MF	RL	IND	NG
KLD [36]		x			x
SRL [50]	*	x	x	x	
HMC [31, 23]	*	x	x		x
APF [71]	x	x	x		x
FastSLAM 2.0 [60]			x	x	x
GMapping [41]			x	x	x
CGR	x	x	x	x	x

Table 2.1: Comparison of CGR with Related Work, highlighting whether each algorithm: is analytic (ANA), model-free (MF), refines locations (RL), is independent of gradients of motion model and posterior (IND), and requires no global computations (NG). The applicability of a feature for an algorithm is indicated by a ‘x’, while a ‘*’ indicates that the feature depends on the implementation.

algorithm based on Mont-Carlo Localization. Dynamic maps maintains an up-to-date map consistent with the perceived changes in the environment, but does not distinguish permanent from movable objects. SLAM+DTMO attempts to map the static state of the environment, and does not distinguish permanent from movable objects. None of the above approaches revise their data associations in light of newer observations. EnML, in contrast:

1. accounts for observations of unmapped objects,
2. explicitly reasons about observations as arising from long-term, short-term, or dynamic features,
3. uses the short-term features for relative corrections while simulatenously using the long-term features for global corrections,
4. does not require any up-to-date maps and reasons in real-time about observations of unmapped objects,
5. is a deterministic algorithm, and
6. revises the observation classifications whenever new data is available.

	UNM	DLS	MOV	NUM	DET	RDA
Markov Localization [35]			x		*	
DPG-SLAM [91]	x				x	
iMac Maps [78]	x		x			
Temporary Maps [58]	x	x	x	x		
Dynamic Maps [7]	x		x			
SLAM+DTMO [92]	x				*	
EnML	x	x	x	x	x	x

Table 2.2: Comparison of EnML with Related Work, highlighting whether each algorithm: uses unmapped objects for location estimation (UNM), distinguishes long-term from short-term objects (DLS), explicitly reasons about observations of movable objects (MOV), does not need update-to-date maps (NUM), is deterministic (DET), and revises data associations (RDA). The applicability of a feature for an algorithm is indicated by a ‘x’, while a ‘*’ indicates that the feature depends on the implementation.

Chapter 3

Sensors and Sensor-Specific Maps

In this thesis we shall introduce localization algorithms that use a number of different sensors, including wheel odometry, laser rangefinders, depth cameras, and WiFi signal strength. Each of these sensors has its own strengths and limitations, and are processed differently by the localization algorithms. This chapter introduces the specifications and constraints of the different types of sensors used in this thesis. We further introduce algorithms to efficiently process the sensor data to be usable for mobile robot localization. We also introduce the sensor-specific map representations that will be used along with the sensors in the rest of the thesis.

3.1 Odometry

Wheel odometry is important for robot localization, especially in indoor environments since there exist featureless stretches like corridors, along which odometry is the only source of information available to localize a robot. Furthermore, odometry provides a prior for correspondence matching between sensor observations and the map. We view wheel encoders as sensors, and in this section, we review the computation of robot odometry from individual wheel encoders.

We assume that the robot has two or more identical wheels of radius r_w , arranged radially about the center of the robot, at a distance R_w from the center such that the face of each wheel is perpendicular to the radial direction. The robot coordinate reference frame is arranged such that the local x axis points forward on the robot, and the y axis points to the left of the robot. The direction of rotation of each wheel is considered positive for counter-clockwise rotation about the radial vector from the center of the robot to the wheel, in accordance with the right-hand rule. Let there be n_w wheels in total, where wheel i is arranged with its axis at an angle θ_i to the robot coordinate frame, and the encoder-reported rate of rotation of the wheel is \dot{e}_i . Note that this is not the only form of wheel arrangement found on mobile robots: there are other forms,

like the four-wheel steered form, but we do not encounter such forms among the robots used in this thesis.

There are two types of wheel configurations that we encounter in the experimental robot data presented in this thesis: the two wheel diff-drive configuration, and the four-wheel omnidirectional drive configuration. A diff-drive robot has two wheels, with wheel 0, the left wheel at $\theta_0 = 90^\circ$ and wheel 1, the right wheel at $\theta_1 = -90^\circ$. The CoBots have four omnidirectional wheels, arranged symmetrically about the robot with the first wheel at $\theta_0 = 45^\circ$. Thus, given the wheel configuration of the robot and the encoder rates, the velocity (v_x, v_y, v_r) of the robot is computed as

$$v_x = \frac{r_w \sum_i \dot{e}_i \cos(\theta_i)}{n_w} \quad (3.1)$$

$$v_y = \frac{r_w \sum_i \dot{e}_i \sin(\theta_i)}{n_w} \quad (3.2)$$

$$v_r = \frac{r_w \sum_i \dot{e}_i}{R_w n_w}. \quad (3.3)$$

The infinitesimal robot odometry $(\delta_x, \delta_y, \delta_r)$ is given by

$$\delta_x = \frac{r_w \sum_i \delta e_i \cos(\theta_i)}{n_w} \quad (3.4)$$

$$\delta_y = \frac{r_w \sum_i \delta e_i \sin(\theta_i)}{n_w} \quad (3.5)$$

$$\delta_r = \frac{r_w \sum_i \delta e_i}{R_w n_w}. \quad (3.6)$$

Here, δe_i is the infinitesimal encoder increment observed. In practice, the infinitesimal changes are measured as increments over successive time periods of constant duration, which is 50ms for the CoBots. Thus, if the robot has the pose x, y, θ in global coordinates, the odometry updated pose $x^\dagger, y^\dagger, \theta^\dagger$ for odometry $\delta_x, \delta_y, \delta_\theta$ is given by

$$x^\dagger = x + \delta_x \cos(\theta) - \delta_y \sin(\theta) \quad (3.7)$$

$$y^\dagger = y + \delta_x \sin(\theta) + \delta_y \cos(\theta) \quad (3.8)$$

$$\theta^\dagger = \theta + \delta_\theta. \quad (3.9)$$

Sensor	Range (m)	FoV (°)	Accuracy (mm)	Refresh Rate (Hz)	Cost (USD)
Hokuyo URG-04lx laser rangefinder	4	240	10	10	1,995
Hokuyo UTM-30lx laser rangefinder	30	270	30	40	4,975
SICK LMS100 laser rangefinder	20	270	30	50	5,776
SICK LMS500 laser rangefinder	80	190	35	25-75	7,018
Microsoft Kinect / Primesense depth sensor	5	57×43	40	30	96
Mesa Imaging SR4000 depth sensor	5	43.6×34.6	5	10-30	4,295

Table 3.1: Specifications for some commonly used range sensors for autonomous robots.

3.2 Range Sensors

Range sensors provide distance measurements to observable objects placed in the sensing area. SONAR sensors, laser rangefinders, and depth cameras are among the most commonly used range sensors for robots. Out of these sensors, in this thesis we shall restrict our use to laser rangefinders and depth cameras.

There are several factors involved in selecting an appropriate range sensor for an autonomous robot, including cost, maximum range, accuracy, refresh rate, field of view, and operating environment. Table 3.1 lists some common range sensors used on autonomous mobile robots and their specifications. Except for the cost of the sensor, the range and field of view of a sensor have the greatest impact on the accuracy and robustness of mobile robot localization.

The two types of range sensors used in this thesis are short-range laser rangefinders and depth cameras. Laser rangefinders have wider fields of view and greater sensing range than depth cameras, but are limited to sensing in a single plane. Depth cameras on the other hand sense the world in 3D, and generate a far greater volume of data. For comparison, the Hokuyo URG-04lx laser rangefinder generates 6.8K 2D points per second, whereas the Microsoft Kinect Sensor generates 9.2M 3D points per second. The proceeding sections discuss the characteristics of each of the two in detail.

3.2.1 Laser Rangefinders

Laser Rangefinders (LRfs) provide range measurements along rays spaced at equidistant angular increments around the sensor. LRfs usually have a wide field of view of at least 180° . The sensor data returned from an LRf is a fixed size list of range values, where the size of the list is determined by the field of view and the angular resolution of the sensor. As Table 3.1 shows, there is a wide variation in the range, field of view, accuracy and refresh rate among different models of LRfs, with a corresponding variation in the cost of the sensors.

The accuracy of LRfs is affected by a number of factors, including the reflectivity and polarizing nature of the surfaces of the objects being observed, the ambient light, and the presence of transparent, reflecting, or refracting objects in the pathway of the laser beam from the sensor. Since these factors are dependent on the specific environment and conditions of operation of the sensor, there is a corresponding wide variation in the accuracy of sensing.

LRfs provide an array of n_R range readings $S_R = \{r_i\}_{i=1:n_R}$ which are the range readings observed at the corresponding angles $S_\theta = \{\theta_i\}_{i=1:n_R}$. Given these range readings, we can construct a 2D point cloud $S_{P_{2D}} = \{p_i\}_{i=1:n_{2D}}$ where each point p_i is reconstructed as

$$p_i = r_i(\cos(\theta_i), \sin(\theta_i)). \quad (3.10)$$

3.2.2 Depth Cameras

Depth cameras provide depth images as observations, where each pixel value in the images corresponds to the observed depth from the camera along the ray passing through that pixel. This depth information, along with the camera intrinsics (horizontal field of view Ω_H , vertical field of view Ω_V , image width w and height h in pixels) can be used to reconstruct a 3D point cloud. Let the depth image of size $w \times h$ pixels provided by the camera be I , where $I(\zeta)$ is the depth of a pixel at location $\zeta = (i, j)$. Note that the pixel value $I(\zeta)$ is the *depth*, not the *range* at the pixel location ζ . This is true for depth cameras that operate on disparity, like the Microsoft Kinect sensor, which we use in this thesis. Other depth cameras return the *range* for the pixel values instead of their *depth*. The corresponding 3D point $s = (s_x, s_y, s_z)$ is reconstructed using the depth value $I(\zeta)$ as $s_x = I(\zeta) \left(\frac{j}{w-1} - 0.5 \right) \tan \left(\frac{\Omega_H}{2} \right)$, $s_y = I(\zeta) \left(\frac{i}{h-1} - 0.5 \right) \tan \left(\frac{\Omega_V}{2} \right)$, $s_z = I(\zeta)$.

There exist depth cameras with different sensing technologies. The Microsoft Kinect and Primesense sensors use disparity matching of a known, projected infrared pattern to compute the depth at every pixel. These sensors generate depth images at a resolution of 640×480 pixels at a frame rate of 30fps. The Mesa Imaging time-of-flight (ToF) sensors, on the other hand, compute

the time of flight of light pulses from every pixel in the image to compute the depth value at the pixel. The resultant depth images have a resolution of 176×144 pixels at a frame rate of 30fps. The disparity based depth cameras, compared to the ToF depth cameras, have lower accuracy and suffer from degradation of sensitivity with depth, but are substantially cheaper: USD 90 for the Microsoft Kinect, compared to USD 4,295 for the Mesa Imaging SR4000.

3.3 Processing Depth Images

The recent availability of inexpensive depth cameras, like the Microsoft Kinect, the Asus Xtion and the Primesense sensor, has made available dense 3D point clouds, which were previously only accessible using much more expensive sensors like time-of-flight cameras or scanning 3D laser rangefinders. We are interested in using these depth cameras for ground based indoor mobile robots. We consider mobile robots with limited on-board computational power, and address two immediate challenges to using the depth cameras for mobile robot localization and navigation:

1. Depth cameras typically generate voluminous data that cannot be processed in its entirety in real time for localization (*e.g.*, the Microsoft Kinect sensor produces 9.2 million 3D pts/sec, compared to the 6800 2D pts/sec of the Hokuyo URG-04lx laser rangefinder).
2. Given that we already have existing 2D maps of our indoor environments, the observed 3D point clouds should be matched with the 2D maps.

We tackle both these challenges. We first introduce the Fast Sampling Plane Filtering (FSPF) algorithm that samples the depth image to produce a set of points corresponding to planes, along with the plane parameters (normals and offsets). The volume of data to be processed is thus significantly reduced, addressing the first challenge with the additional advantage that non-planar objects in the scene, which are unlikely to correspond to map features are filtered. We then address the second challenge later in Chapter 4.2.2 by introducing an observation model in our localization algorithm that matches the plane filtered points to the lines in the 2D maps, making it possible to reuse our existing 2D maps. This approach of using the plane-filtered point cloud for mobile robot localization is in contrast to, and more effective than (as we shall show in Chapter 4.2.3) down-projecting the 3D point cloud and binning it to simulate a conventional laser rangefinder. Our combined approach interestingly uses only the depth image and does not require the RGB images, thus eliminating any possible privacy issues that might arise in the use and logging of RGB camera images.

Algorithm 1 Fast Sampling Plane Filtering

```

1: procedure PLANEFILTERING( $I$ )
2:    $S_P \leftarrow \{\}$  ▷ Plane filtered points
3:    $S_R \leftarrow \{\}$  ▷ Normals to planes
4:    $S_O \leftarrow \{\}$  ▷ Outlier points
5:    $n \leftarrow 0$  ▷ Number of plane filtered points
6:    $k \leftarrow 0$  ▷ Number of neighborhoods sampled
7:   while  $n < n_{\text{filt}} \wedge k < n_{\text{neighbor}}$  do
8:      $k \leftarrow k + 1$ 
9:      $\zeta_0 \leftarrow (\text{rand}(0, h - 1), \text{rand}(0, w - 1))$ 
10:     $\zeta_1 \leftarrow \zeta_0 + (\text{rand}(-g, g), \text{rand}(-g, g))$ 
11:     $\zeta_2 \leftarrow \zeta_0 + (\text{rand}(-g, g), \text{rand}(-g, g))$ 
12:    Reconstruct  $s_0, s_1, s_2$  from  $\zeta_0, \zeta_1, \zeta_2$ 
13:     $r = \frac{(s_1 - s_0) \times (s_2 - s_0)}{\|(s_1 - s_0) \times (s_2 - s_0)\|}$  ▷ Compute plane normal
14:     $\bar{z} = \frac{s_{0z} + s_{1z} + s_{2z}}{3}$ 
15:     $w' = w \frac{\Delta}{\bar{z}} \tan(\Omega_H)$ 
16:     $h' = h \frac{\Delta}{\bar{z}} \tan(\Omega_V)$ 
17:     $[\text{numInliers}, \hat{S}_P, \hat{S}_R] \leftarrow \text{RANSAC}(\zeta_0, w', h', n_{\text{local}}, \epsilon_{\text{plane}})$ 
18:    if  $\text{numInliers} > \alpha_{\text{in}} n_{\text{local}}$  then
19:      Add  $\hat{S}_P$  to  $S_P$ 
20:      Add  $\hat{S}_R$  to  $S_R$ 
21:       $\text{numPoints} \leftarrow \text{numPoints} + \text{numInliers}$ 
22:    else
23:      Add  $\hat{S}_P$  to  $S_O$ 
24:    end if
25:  end while
26:  return  $S_P, S_R, S_O$ 
27: end procedure

```

3.3.1 Fast Sampling Plane Filtering

FSPF takes the depth image I as its input, and creates a list S_P of 3D points, a list S_R of corresponding plane normals, and a list S_O of outlier points that do not correspond to any planes. Algorithm 1 outlines the plane filtering procedure. It uses the helper subroutine $[\text{numInliers}, \hat{S}_P, \hat{S}_R] \leftarrow \text{RANSAC}(\zeta_0, w', h', n_{\text{local}}, \epsilon_{\text{plane}})$, which performs the classical RANSAC algorithm over the window of size $w' \times h'$ around location ζ_0 in the depth image, and returns inlier points and normals \hat{S}_P and \hat{S}_R respectively, as well as the number of inlier points found. The configuration parameters required by FSPF are listed in Table 3.2.

FSPF proceeds by first sampling three locations $\zeta_0, \zeta_1, \zeta_2$ from the depth image (lines 9-11). The first location ζ_0 is selected randomly from anywhere in the image, and then ζ_1 and ζ_2 are

Parameter	Value	Description
n_{filt}	2000	Maximum total number of filtered points
n_{neighbor}	20000	Maximum number of neighborhoods to sample
n_{local}	80	Number of local samples
g	60	Neighborhood for global samples (in pixels)
Δ	$0.5m$	Plane size in world space for local samples
ϵ_{plane}	$0.02m$	Maximum plane offset error for inliers
α_{in}	0.8	Minimum inlier fraction to accept local sample

Table 3.2: Configuration parameters for FSPF

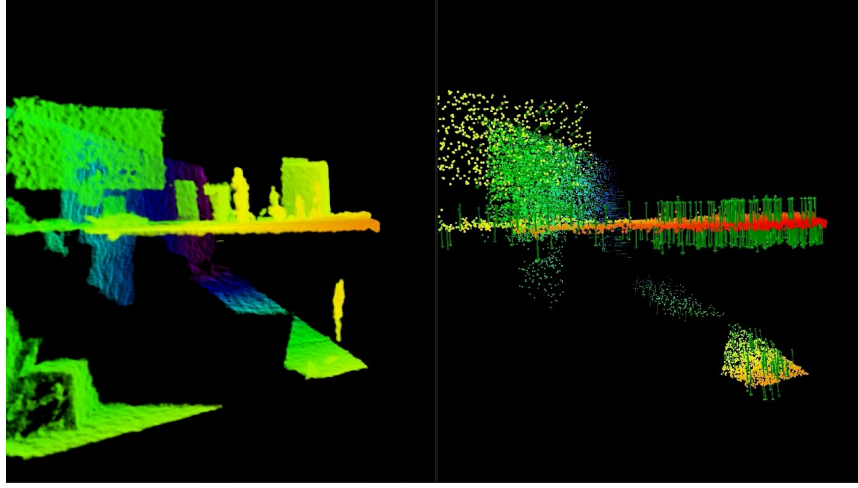


Figure 3.1: Fast Sampling Plane Filtering in a scene with a cluttered desktop. The complete 3D point cloud is shown on the left, the plane filtered points and the corresponding normals on the right. The table clutter is rejected by FSPF while preserving the large planar elements like the monitors, the table surface, the walls and the floor.

selected from a neighborhood of size g around ζ_0 . The 3D coordinates for the corresponding points s_0, s_1, s_2 are then computed (line 12). A search window of width w' and height h' is computed based on the mean depth (z -coordinate) of the points s_0, s_1, s_2 (lines 14-16) and the minimum expected size Δ of the planes in the world. Local RANSAC is then performed in the search window. If more than $\alpha_{\text{in}} n_{\text{local}}$ inlier points are produced as a result of running RANSAC in the search window, then all the inlier points are added to the list S_P , and the associated normals (computed using a least-squares fit on the RANSAC inlier points) to the list S_R . This algorithm is run a maximum of m_{max} times to generate a list of maximum n_{filt} 3D points and their corresponding plane normals. Figure 3.1 shows an example scene with the plane filtered points and their corresponding plane normals.

Algorithm 2 Obstacle Scan Generation

```

1: procedure GENRATEOBSTACLESCAN( $S$ )
2:    $S_N[j] \leftarrow \infty \forall j \in [0, n_N]$ 
3:   for all  $s_i \in S$  do
4:     if  $s_i^z < \epsilon_N$  then
5:       continue
6:     end if
7:      $\theta \leftarrow \text{atan2}(s_i^y, s_i^x)$ 
8:      $r \leftarrow (s_i^x^2 + s_i^y^2)^{\frac{1}{2}}$ 
9:      $j \leftarrow \text{floor} \left( n_N \frac{\theta - \frac{\Omega_N}{2}}{\Omega_N} \right)$ 
10:     $S_N[j] \leftarrow \min(S_N[j], r)$ 
11:   end for
12:   return  $S_N$ 
13: end procedure

```

3.3.2 Obstacle Scan

In addition to plane filtering by FSPF, depth images observed by the robot are also processed to generate an *Obstacle Scan* S_N . The obstacle scan is a list of n_N entries, where value $S_N[i]$ is the distance of the closest obstacle along the ray from the robot in the direction of $\Omega_N \left(\frac{i}{n_N} - \frac{1}{2} \right)$. The obstacle scan thus has a format identical to a laser rangefinder scan: it covers an angular field of view Ω_N with n_N bins, thus providing an angular resolution of $\frac{\Omega_N}{n_N}$. The obstacle scan is used for safe navigation of the mobile robot, to avoid obstacles in its path, and also for localization using Episodic non-Markov Localization (Chapter 5). Algorithm 2 lists the algorithm to generate the obstacle scan S_N given a set of 3D points S . Elements $s_i = [s_i^x, s_i^y, s_i^z] \in S$ are 3D points in the reference frame of the robot drawn from the set of points $S_P \cup S_R \cup S_O$ generated by FSPF. Note that these points may be generated from multiple depth cameras on the robot, but are used to generate a single, combined obstacle scan. Figure 3.2 shows an example obstacle scan in a scene including a table and four chairs.

3.4 WiFi Signal Strength Sensing

With the increasing prevalence of wireless LAN, “WiFi” in indoor environments, signal strength sensing of WiFi has recently emerged as an inexpensive sensing mode for localization. Office environments frequently deploy “Infrastructure Mode” WiFi networks, which advertise the network using a Service Set ID (SSID) (*e.g.*, “CMU-Secure” at Carnegie Mellon University) over multiple stations called Access Points (APs). The SSID is a human-readable text string used

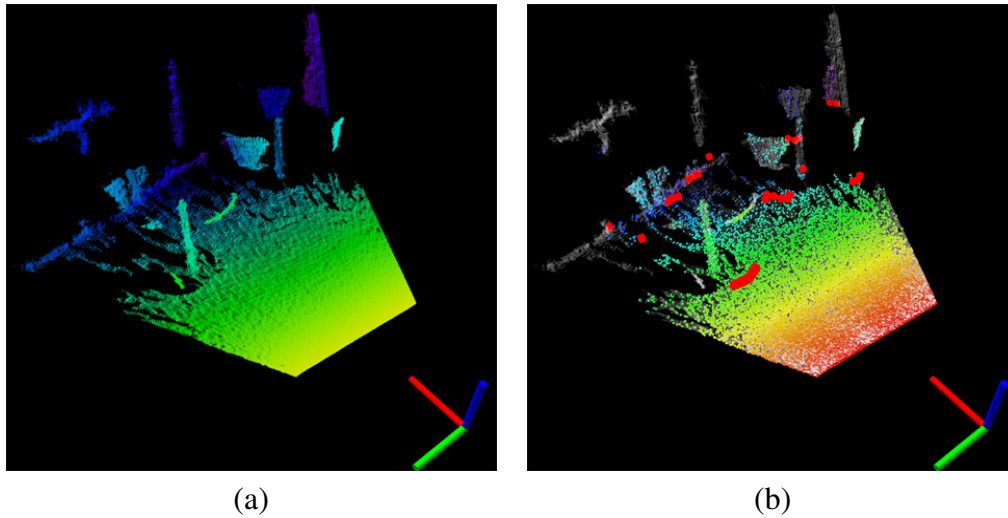


Figure 3.2: Obstacle scan generated from a depth image: The full 3D point cloud (a) and (b) the sampled points (shown in color), along with the obstacle scan (red boxes). The robot location is marked by the axes.

by end users to identify which network they are connecting to. APs are WiFi network radios distributed over the building, to provide even radio coverage everywhere. Each AP has a unique ID called the Base Service Set ID (BSSID), which is a 48-bit integer that is unique among all APs, across all networks and all manufacturers in the world. End-users usually select the SSID of the network that they wish to connect to, and the computer's operating system and network card driver automatically polls for a list of APs in range, and selects the best AP to connect to.

In order for a robot to use WiFi signal strengths as sensing for localization, the robot must be able to query all APs in range. In our work we use the Wireless Tools For Linux ¹ to perform these queries. The result of such queries is a list of APs observed, with the following fields per AP:

1. SSID: The SSID of the network that the AP provides access to.
2. BSSID: The unique BSSID of the AP
3. RSSI: The received radio signal strength of the last radio packet received from the AP
4. Age: The age in seconds, of the radio packet last received from the AP

Given these entries per AP observed, the robot can construct a WiFi map of the environment, and use the observations and the map to localize in the building.

¹<http://wireless.kernel.org/>

3.5 Map Representations

Map representations are crucial to the accuracy, robustness, and computational complexity of mobile robot localization algorithms. We introduce here the two map representations that we use with the localization algorithms in this thesis: the Vector Map representation, and the Graph-WiFi Map representation.

3.5.1 Vector Maps

The most common map representation for robot localization using range sensors is the occupancy grid map [33]. An occupancy grid map discretizes the world into grid cells and assigns an occupancy value to each cell to indicate whether that cell in the world is occupied by an obstacle, or is free space. The map representation directly affects the computational complexity of correlating actual observations to the map. This correlation is evaluated by performing a *ray cast* from the robot’s location estimate in the map. The result of the ray cast is the expected range reading for each observation angle from the robot’s range sensor. In this thesis we use a map representation called the vector map, which instead models only the permanent features of the environment as a set of line segments, and which, as we shall show, can be used to efficiently compute ray casts analytically.

The vector map M_{vector} is a set of s 2-dimensional line segments l_i corresponding to all the permanent features like walls in the environment: $M_{\text{vector}} = \{l_i\}_{i=1:s}$. Such a representation may be acquired by mapping (*e.g.*, [94]) or (as in our case) taken from the blueprints of the building. The reason we use a vector map for localization is three-fold:

1. A vector map allows higher precision than an occupancy grid map, in a more compact representation.
2. Ray casts can be performed analytically on a vector map, and thus are much faster than ray casts on an occupancy grid map.
3. The gradients of the observation model (which are required for the localization algorithms introduced later) can be computed analytically on a vector map.

Given a vector map, to compute the observation likelihoods based on observed planes, the first step is to estimate which lines on the map are likely to be observed (the *scene lines*), given the pose estimate of the robot. This ray casting step is analytically computed using the vector map representation.

The procedure to analytically generate a ray cast at location x given the map M_{vector} is outlined in Algorithm 3. The returned result is the scene lines L : a list of non-intersecting, non-

occluded line segments visible by the robot from the location x . This algorithm calls the helper procedure $\text{TrimOcclusion}(x, l_1, l_2, L)$ that accepts a location x , two lines l_1 and l_2 and a list of lines L . TrimOcclusion trims line l_1 based on the occlusions due to the line l_2 as seen from the location x . The list L contains lines that yet need to be tested for occlusions by l_2 . There are in general 4 types of arrangements of l_1 and l_2 , as shown in Figure 3.3:

1. l_1 is not occluded by l_2 . In this case, l_1 is unchanged.
2. l_1 is completely occluded by l_2 . l_1 is trimmed to zero length by TrimOcclusion .
3. l_1 is partially occluded by l_2 . l_1 is first trimmed to a non occluded length, and if a second disconnected non occluded section of l_1 exists, it is added to L .
4. l_1 intersects with l_2 . Again, l_1 is first trimmed to a non occluded length, and if a second disconnected non occluded section of l_1 exists, it is added to L .

Algorithm 3 Analytic Ray Cast Algorithm

```

1: procedure ANALYTICRAYCAST( $M_{\text{vector}}, x$ )
2:    $\hat{L} \leftarrow M_{\text{vector}}$ 
3:   Sort  $\hat{L}$  in order of angle subtended from  $x$ 
4:    $L \leftarrow \{\}$ 
5:   for  $l_i \in \hat{L}$  do
6:     for  $l_j \in L$  do
7:        $\text{TrimOcclusion}(x, l_i, l_j, \hat{L})$ 
8:     end for
9:     if  $\|l_i\| > 0$  then  $\triangleright l_i$  is partly non occluded
10:      for  $l_j \in L$  do
11:         $\text{TrimOcclusion}(x, l_j, l_j, \hat{L})$ 
12:      end for
13:       $L \leftarrow L \cup \{l_i\}$ 
14:    end if
15:  end for
16:  return  $L$ 
17: end procedure

```

The analytic ray casting algorithm (Algorithm 3) proceeds as follows: A list \hat{L} of all possible lines is made from the map M_{vector} . Every line $l_i \in \hat{L}$ is first trimmed based on occlusions by lines in the existing scene list L (lines 6-8). If at least part of l_i is left non occluded, then the existing lines in \hat{L} are trimmed based on occlusions by l_i (lines 10-12) and l_i is then added to the scene list L . The result is a list of non occluded, non-intersecting scene lines in L .

Given the list of non occluded, non-intersecting scene lines in L , to compute correspondences between observed points and the lines in L , the line segments in L are sorted in order of their

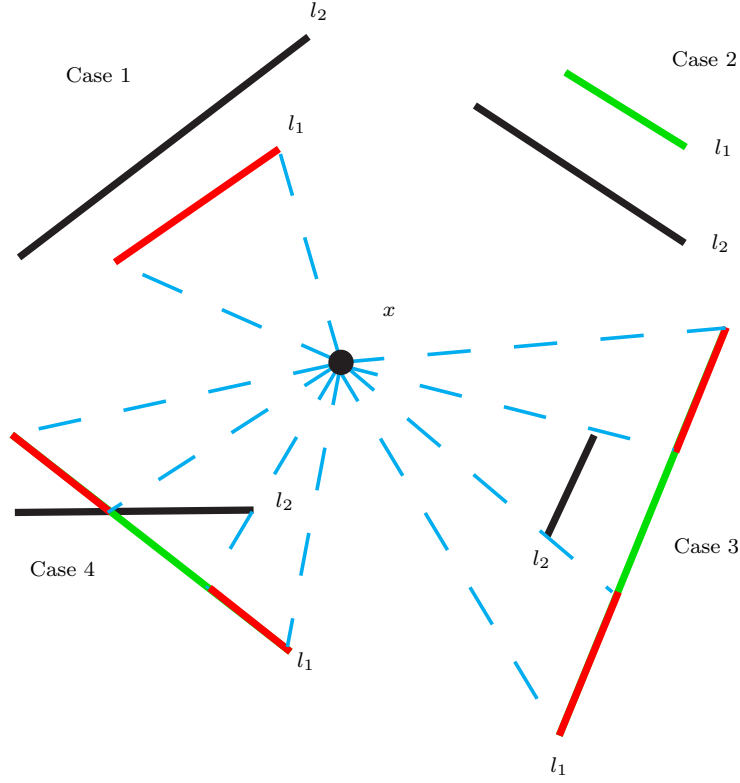


Figure 3.3: Line occlusion cases. Line l_1 is being tested for occlusion by line l_2 from location x . The occluded parts of l_1 are shown in green, and the visible parts in red. The visible ranges are bounded by the angles demarcated by the blue dashed lines.

angle relative to the robot's location x . Note that the angles subtended by every line segment in L is unique and non-overlapping, since the line segments in L are non-intersecting and non-occluding from the robot's location. The correspondence of an observed point s is then found by computing the angle θ_s subtended by s from the robot location x , and searching for that line segment in L that has an angular extent that θ_s falls within. Since the lines in L have been sorted in order of their angles, and since the angles subtended by the line segments are not overlapping, this search takes at best $O(1)$ time if using a hash table lookup without any collisions, and at worst $O(\log(n_L))$ time if using binary search over the list of n_L sorted line segments in L .

3.5.2 Graph-WiFi Maps

We denote the WiFi map of the building as M_{wifi} , with $M_{\text{wifi}} = \langle V_{\text{wifi}}, E_{\text{wifi}}, A_{\text{wifi}}, \mathbf{W}, \Sigma_{\text{wifi}} \rangle$ where V_{wifi} is a set of vertices, E_{wifi} a set of edges to connect them, and A_{wifi} the set of WiFi access points in the environment. \mathbf{W} and Σ_{wifi} are matrices representing the WiFi signal strength means and standard deviations across the map. Each vertex $v \in V_{\text{wifi}}$ corresponds to a unique

location in the building. Edges $e = \langle v_a, v_b, \Gamma \rangle, e \in E_{\text{wifi}}$ indicate that there is a navigable path between vertices v_a and v_b . For every edge, the width of the corridor in that section and the length of the edge are represented as $\Gamma = \langle \text{width}, \text{length} \rangle$. Element i, j of \mathbf{W} is the mean WiFi signal strength (in dBm) of access point a_j as measured from vertex v_i . Similarly, element i, j of Σ_{wifi} is the observed standard deviation of the WiFi signal strength of access point a_j as measured from vertex v_i .

Matrices \mathbf{W} and Σ_{wifi} are enumerated during the *Learning Phase*, which needs to be performed once. The Learning Phase starts with a manual definition of a “skeleton graph” of the map, where the longest straight line segments in the building are defined as the edges of the skeleton graph. Once the learning phase is done, a path is planned to traverse every edge of the skeleton graph. The robot then follows this path, and along each edge, it stops at regularly interspaced sample locations of a user-defined maximum spacing, and defines a new vertex. Thus, each edge of the skeleton graph is split up into multiple smaller edges. At each sample location (which is also a vertex in the newly generated complete graph), the robot collects WiFi signal strength readings for a pre-determined (and user-defined) duration. During this process, the robot uses odometry alone for estimating its localization, and errors in localization are corrected manually using a graphical user interface. Thus, the skeleton graph is used to generate the complete graph, and the matrices \mathbf{W} and Σ_{wifi} are populated.

3.6 Summary

In this chapter we presented the tradeoffs in different forms of sensing, and the format of the data generated by the different sensors. We introduced the Fast Sampling Plane Filtering (FSPF) algorithm to generate plane-filtered point clouds from depth images, and an algorithm to generate Obstacle Scans from depth images. We further introduced graph-WiFi maps, and vector map representations, along with the analytic ray cast algorithm to efficiently evaluate correspondences between observations and the vector map.

Chapter 4

Markov Localization

In this chapter, we present a number of variants of Markov Localization for mobile robots. We first review particle filters for mobile robot localization. Next, we introduce Corrective Gradient Refinement (CGR), which improves upon Monte Carlo Localization by using the gradients of the observation likelihood function to improve the samples in the proposal distribution of a particle filter. CGR can use either points observed by a laser rangefinder, or plane-filtered points generated by FSPF (Chapter 3.3.1) along with robot odometry to localize a robot on a vector map (Chapter 3.5.1). We further introduce a WiFi signal strength based localization algorithm using a graph-based WiFi map (Chapter 3.5.2). Finally, we compare localization using the different variants of Markov Localization introduced in this chapter, highlighting the tradeoffs in accuracy and robustness in different indoor environments.

4.1 Particle Filters for Mobile Robot Localization

Bayesian filters are used to track the probability distribution of a d -dimensional state variable $x \in \mathbb{R}^d$ over time given the history of observations $s_{1:t}$ and control inputs $u_{1:t-1}$. The resulting probability distribution of x is called the *belief*, $Bel(x_t) = p(x_t|y_{1:t}, u_{1:t-1})$. The belief is recursively updated using the equation

$$Bel(x_t) = \eta p(y_t|x_t) \int p(x_t|x_{t-1}, u_{t-1}) Bel(x_{t-1}) dx_{t-1}. \quad (4.1)$$

Particle filters, and in particular, Monte Carlo Localization (MCL)[29] represent the belief by a set of weighted samples, or “particles,” $Bel(x_t) = \{x_t^i, w_t^i\}_{i=1:m}$, where x_t^i is the state of the particle i at time t and w_t^i its associated weight. Recursive updates of the particle filter can be implemented via a number of methods, notably among which is the sampling/importance

resampling method (SIR) [79]

In MCL-SIR, m samples x_{t-}^i are drawn with replacement from the prior belief $Bel(x_{t-1})$ proportional to their weights w_{t-1}^i . These samples x_{t-}^i are used as priors to sample from the motion model of the robot $p(x_t|x_{t-1}, u_{t-1})$ to generate a new set of samples \hat{x}_t^i that approximates $p(x_t|x_{t-1}, u_{t-1})Bel(x_{t-1})$. The samples \hat{x}_t^i are called the “proposal distribution”. Importance weights for each sample \hat{x}_t^i are then computed as

$$w_t^i = \frac{p(s_t|\hat{x}_t^i)}{\sum_i p(s_t|\hat{x}_t^i)}. \quad (4.2)$$

Here, the term $p(s_t|\hat{x}_t^i)$ is computed by the “observation model”, which computes the probability of making the observation s_t at location \hat{x}_t^i . Finally, samples \hat{x}_t^i are drawn from the proposal distribution \hat{x}_t^i proportional to the importance weights w_t^i . The location hypotheses in a particle filter are thus updated iteratively as follows:

1. *Predict*: Samples from the previous time step, x_{t-1}^i are propagated through the robot’s motion model $p(x_t|x_{t-1}, u_{t-1})$ to generate proposal distribution samples \hat{x}_t^i .
2. *Update*: The observation model is then used to compute weights for each sample in the proposal distribution using Equation 4.2.
3. *Resample*: The posterior distribution is then estimated by samples \hat{x}_t^i , drawn from the proposal distribution \hat{x}_t^i proportional to the importance weights w_t^i .

With this general background of particle filters for mobile robot localization, we now look at the variants that we introduced.

4.2 Corrective Gradient Refinement for Localization

Particle filters have proven to be successful at state estimation and tracking for non-parametric stochastic processes. However, for every particle filter implementation, the mantra to reach higher accuracy has traditionally been “add more particles.” This approach does work, but the required number of particles scales exponentially with the number of dimensions. Therefore, significant work (as discussed in Chapter 2) has gone into developing algorithms that work with fewer particles but *do more* with the same particles, such that the desired accuracy is still met.

In this thesis we introduce one such novel algorithm, Corrective Gradient Refinement (CGR), which reduces the required number of particles for a particle filter by using the *gradients* of the observation model. This results in more efficient sampling of the robot’s localization belief by sampling less along directions of low uncertainty given an observation, while densely sampling

along directions of high uncertainty. For example, if the LIDAR scans of the robot detect parallel walls, there will be fewer samples in the direction perpendicular to the walls (the direction of low uncertainty), and more samples parallel to the walls (the direction of high uncertainty).

We apply Corrective Gradient Refinement to the task of indoor mobile robot localization with point cloud sensors. By point cloud sensors, we refer to sensors that generate 2D or 3D points corresponding to the surfaces of detected obstacles. We use both a 2D LIDAR scanner and a 3D Microsoft Kinect sensor to generate point clouds. Although MCL-SIR (Chapter 4.1) is largely successful, several issues may still create problems:

1. Observations could be “highly peaked,” *i.e.* $p(s_t|x_t^i)$ could have very large values for a very small subspace $\epsilon \subseteq \mathbb{R}^d$ and be negligible everywhere else. In this case, the probability of samples x^i overlapping with the subspace ϵ is diminishingly small.
2. If the posterior is no longer being approximated well by the samples x^i , recovery is only possible by chance, if the motion model happens to generate new samples that better overlap with the posterior.
3. The required number of samples necessary to overcome the aforementioned limitations scales exponentially in the dimension of the state space.

The Corrective Gradient Refinement (CGR) algorithm, which we introduce, addresses these problems in a computationally efficient manner. It refines samples locally to better cover neighboring regions with high observation likelihood, even if the prior samples had poor coverage of such regions. Since the refinement takes into account the gradient estimates of the observation model, it performs well even with highly peaked observations. Thus, CGR tracks the location of the robot with fewer particles than MCL-SIR, although for the task of global localization, the number of particles would still have to be scaled up as in the case of MCL-SIR to cover the entire map. We formulate the gradient estimates of the observation model in a manner that allows analytical computation of the gradient using a vector map. We show experimentally that CGR has higher accuracy and lower variance across trials than MCL-SIR. Furthermore, for the same level of desired accuracy, CGR requires far fewer particles than MCL-SIR for localization tracking.

4.2.1 The Corrective Gradient Refinement Algorithm

The CGR algorithm iteratively updates the past belief $Bel(x_{t-1})$ using observation s_t and control input u_{t-1} to estimate the latest belief $Bel(x_t)$ as follows:

1. **Predict:** Samples of the belief $Bel(x_{t-1})$ are evolved through the motion model, $p(x_t|x_{t-1}, u_{t-1})$ to generate a first stage proposal distribution q^0 .

2. **Refine:** Samples of q^0 are “refined” in r iterations (which produce intermediate distributions $q^i, i \in [1, r - 1]$) using the gradients $\frac{\delta}{\delta x}p(s_t|x)$ of the observation model $p(s_t|x)$.
3. **Update:** Samples of the last generation proposal distribution q^r and the first stage proposal distribution q^0 are sampled using an acceptance test to generate the final proposal distribution q .
4. **Resample:** Samples x_t^i of the final proposal distribution q are weighted by corresponding importance weights w_t^i , and resampled with replacement to generate $Bel(x_t)$.

We now explain the four steps of the CGR algorithm in detail.

The Predict Step

Let the samples of the past belief, $Bel(x_{t-1})$ be given by x_{t-1}^i . These samples are then evolved using the motion model of the robot to generate a new set of samples $q^0 = \{x_{q^0}^i\}_{i=1:m}$ as $x_{q^0}^i \sim p(x_t|x_{t-1}^i, u_{t-1})$. This sample set q^0 is called the first stage proposal distribution, and takes time complexity $\mathcal{O}(m)$ to compute.

The Refine Step

The Refine step is central to the CGR algorithm. It corrects sample estimates that contradict the observations (*e.g.*, when the LIDAR scan shows the robot to be in the center of the corridor, but the sample is closer to the left wall). This results in the CGR algorithm sampling less along directions that have low uncertainty in the observation model while maintaining samples along directions of high uncertainty.

For the CGR algorithm, estimates of the first order differentials (the gradients) of the observation model, $\frac{\delta}{\delta x}p(s_t|x)$ must to be computable. Given these gradients, the Refine step performs gradient descent for r iterations, generating at iteration i the i -th stage proposal distribution. Algorithm 4 outlines the Refine step.

Performing multiple iterations of gradient descent allows the estimates of the gradients of the observation model to be refined between iterations, which results in higher accuracy. The computation time required for the Refine step scales as $\mathcal{O}(rm)$. In general, smaller values of the step size η , paired with larger values of r result in higher accuracy.

The Refine step performs rm total iterations, each of which requires the computation of the gradient of the observation model. Therefore, the time complexity of the refine step is $\mathcal{O}(rmf(\dots))$ where f is a function that depends on the observation model.

To generate the final distribution q , Samples $x_{q^r}^i$ from the r -th stage distribution are probabilistically chosen over the corresponding samples $x_{q^0}^i$ of the first stage distribution proportional

Algorithm 4 The Refine step of CGR

```

1: Let  $q^0 = \{x_{q^0}^j\}_{j=1:m}$ 
2: for  $i = 1$  to  $r$  do
3:    $q^i \leftarrow \{\}$ 
4:   for  $j = 1$  to  $m$  do
5:      $x_{q^i}^j \leftarrow x_{q^{i-1}}^j + \eta \left[ \frac{\hat{\delta}}{\delta x} p(s_t|x) \right]_{x=x_{q^{i-1}}^j}$ 
6:    $q^i \leftarrow q^i \cup x_{q^i}^j$ 
7:   end for
8: end for

```

to the value of the acceptance ratio ρ^i , as:

$$\rho^i = \min \left\{ 1, \frac{p(s_t|x_{q^r}^i)}{p(s_t|x_{q^0}^i)} \right\} \quad (4.3)$$

This acceptance test allows the algorithm to probabilistically choose samples that better match the observation model $p(s_t|x)$. If the Refine step does not produce samples with higher observation likelihood than the samples in the first stage distribution, the final proposal distribution q will have a mixture of samples from q^0 and q^r . Furthermore, if the Refine step results in most samples having higher observation likelihood than the samples in the first stage distribution, the final proposal distribution q will consist almost entirely of samples from q^r .

Samples from the final proposal distribution q thus generated are weighted by importance weights, and resampled to compute the latest belief $Bel(x_t)$ in the Update step. The acceptance test step has time complexity $\mathcal{O}(m)$ since it reuses the cached values of the observation likelihood computed on the first and last iterations of the Refine step.

The Update Step

The importance weights for the CGR algorithm are different from those of the MCL-SIR algorithm, since in the CGR algorithm, the proposal distribution q is not the same as the samples of the motion model. To derive the expression for the importance weights of the CGR algorithm, we first factor out the belief update (Equation 4.1) as:

$$Bel(x_t) \propto p(s_t|x_t)p(x_t|u_{t-1}, Bel(x_{t-1})) \quad (4.4)$$

$$p(x_t|u_{t-1}, Bel(x_{t-1})) = \int p(x_t|x_{t-1}, u_{t-1})Bel(x_{t-1})dx_{t-1} \quad (4.5)$$

The proposal distribution from which the belief $Bel(x_t)$ is computed, is q . Hence, the (non-normalized) importance weights w_t^i for samples x_t^i in q are given by:

$$w_t^i = \frac{p(s_t|x_t^i)p(x_t^i|u_{t-1}, Bel(x_{t-1}))}{q(x_t^i)} \quad (4.6)$$

In this expression, $p(s_t|x_t^i)$ is computed using the observation model, and the remaining two terms $p(x_t^i|u_{t-1}, Bel(x_{t-1}))$ and $q(x_t^i)$ are computed by kernel density estimates at the locations x_t^i using the samples from q^0 and q for support of the kernel density respectively. Since the importance weight accounts for the motion model (from the term $p(x_t^i|u_{t-1}, Bel(x_{t-1}))$) as well as the refined proposal distribution ($q(x_t^i)$), the CGR algorithm avoids “peak following” behavior which contradict the motion model.

The Resample Step

The samples in q are resampled in proportion to their importance weights using low variance resampling [87] to obtain the samples of the latest belief, $Bel(x_t)$. The update step has time complexity $\mathcal{O}(m^2)$. It is quadratic in the number of particles since it requires computation of kernel density functions.

Thus, given the motion model $p(x_t|x_{t-1}, u_{t-1})$, the observation model $p(s_t|x)$, the gradients of the observation model $\frac{\delta}{\delta x}p(s_t|x)$ and the past belief $Bel(x_{t-1})$, the CGR algorithm computes the latest belief $Bel(x_t)$. Adding the runtime of all the steps together, the CGR algorithm is performed in $\mathcal{O}(rmf(\dots) + m^2)$ time. The formulation of the observation model and its gradients is the subject of the next Section.

4.2.2 Vector Map Observation Models

We are interested in localizing an indoor mobile robot using the CGR algorithm applied to point cloud sensors and a vector map M_{vector} (Chapter 3.5.1) Using the vector map, we develop an observation model for 2D point clouds and projected 3D point clouds.

2D Point Cloud Observation Model

The 2D point cloud observation model is used for sensors like a planar LIDAR scanner, which generate point clouds in 2D space.

Let the 2D point cloud generated by the sensor (*e.g.*, 2D LIDAR scanner) be denoted by the set of 2D points $S_{P_{2D}} = \{p_i\}_{i=1:n_{2D}}$. Without loss of generality, we assume that the origin of the sensor coincides with the origin of the robot. Let the observation be $s = S_{P_{2D}}$, and the pose of the robot x be given by $x = \{x_l, x_\theta\}$ where x_l is the 2D location of the robot, and x_θ its orientation angle. The observation likelihood $p(y|x)$ is then computed as follows:

1. For every point p_i in $S_{P_{2D}}$, line l_i ($l_i \in M_{\text{vector}}$) is found by ray casting such that the ray in the direction of $p_i - x_l$ and originating from x_l intersects l_i before any other line.
2. The perpendicular distance d_i of p_i from the (extended) line l_i is computed.
3. The total (non-normalized) observation likelihood $p(y|x)$ is then given by:

$$p(S_{P_{2D}}|x) = \prod_{i=1}^n \exp \left[-\frac{d_i^2}{2f\Sigma_S^2} \right] \quad (4.7)$$

Here, Σ_S is the standard deviation of the distance measurements of a single ray, and f (where $f > 1$) is a discounting factor to discount for the correlation between rays. The gradient of the observation model is therefore given by:

$$\frac{\delta}{\delta x} p(S_{P_{2D}}|x) = -\frac{p(S_{P_{2D}}|x)}{f\Sigma_S^2} \sum_{i=1}^n \left[d_i \frac{\delta d_i}{\delta x} \right] \quad (4.8)$$

The term $\frac{\delta d_i}{\delta x}$ in this equation has two terms, corresponding to the translation component $\frac{\delta d_i}{\delta x_l}$ and the rotational component $\frac{\delta d_i}{\delta x_\theta}$. These terms are computed by rigid body translation and rotation of the point cloud P respectively. Due to errors in the pose of the robot, the point to line correspondences may be incorrect for some points, which would lead to errors in the gradient estimate. Hence, some estimated outlier rejection is necessary before evaluating Equation 4.8 for all the points. Outlier rejection is performed by two checks:

1. If the distance d_i is greater than a threshold ϵ_{dist} , the corresponding point p_i is rejected.
2. If the angle between the line joining successive points p_i, p_{i+1} and either of the corresponding lines l_i, l_{i+1} is greater than a threshold ϵ_{ang} , then the points p_i, p_{i+1} are rejected.

Performing the ray cast operation from the estimated pose takes time $\mathcal{O}(s + n)$, and the computation of the observation likelihood and gradients is $\mathcal{O}(n)$. So the total complexity for the observation model is $f(s, n) = \mathcal{O}(s + n)$. Hence the complexity of vector-based 2D point cloud

localization with CGR is $\mathcal{O}(rm(s+n) + m^2)$. In comparison, the time complexity of MCL-SIR using a vector map is $\mathcal{O}(m + m \cdot (s+n))$. Thus, for the same number of particles, CGR is generally slower than MCL-SIR. However, since CGR-localization provides greater accuracy than MCL-SIR, it can be run with far fewer particles, as we will demonstrate experimentally.

Projected 3D Point Cloud Observation Model

For localization using depth cameras, we use the FSPF-generated plane filtered point cloud S_P and the corresponding plane normal estimates S_R (Chapter 3.3.1). Since the map M_{vector} on which the robot is localizing is in 2D, the 3D filtered point cloud S_P and the corresponding plane normals S_R are first projected onto 2D to generate a 2D point cloud S'_P along with the corresponding normalized normals S'_R . Points that correspond to ground plane detections are rejected at this step. The projected 3D point cloud S'_P is then treated identically to the 2D point cloud, and its observation likelihood and space gradients are computed using the 2D point cloud observation model.

4.2.3 Experimental Results

Our experiments were performed on our custom built omnidirectional indoor mobile robot CoBot, which is equipped with a Hokuyo URG-04LX planar LIDAR scanner (which produces 2D point clouds) and a Microsoft Kinect sensor (which produces 3D point clouds). We separately evaluated CGR using the 2D point cloud model, and CGR with the projected 3D point cloud model.

CGR with 2D Point Cloud Model

We ran a series of experiments to compare the performance of the CGR algorithm with the MCL-SIR algorithm and to test the effectiveness of the CGR algorithm alone over extended periods of time. The MCL-SIR algorithm was implemented using the same map representation and observation models as those used for the CGR algorithm. The only difference is that for the MCL-SIR algorithm, there were no refine step, and the usual MCL-SIR update equations were used.

In the first series of experiments, we logged sensor data while driving the robot around the map, traversing a path 374m long. The environment is the 7th floor of the Gates-Hillman Center at Carnegie Mellon University. As in a typical human office environment, there were a few unmapped objects in the environment like tables, chairs, planters, and trash cans. There was also regular human traffic in the corridors, acting as dynamic obstacles in the path of the robot. The true robot location was manually annotated by aligning the sensor scans to the map. This data

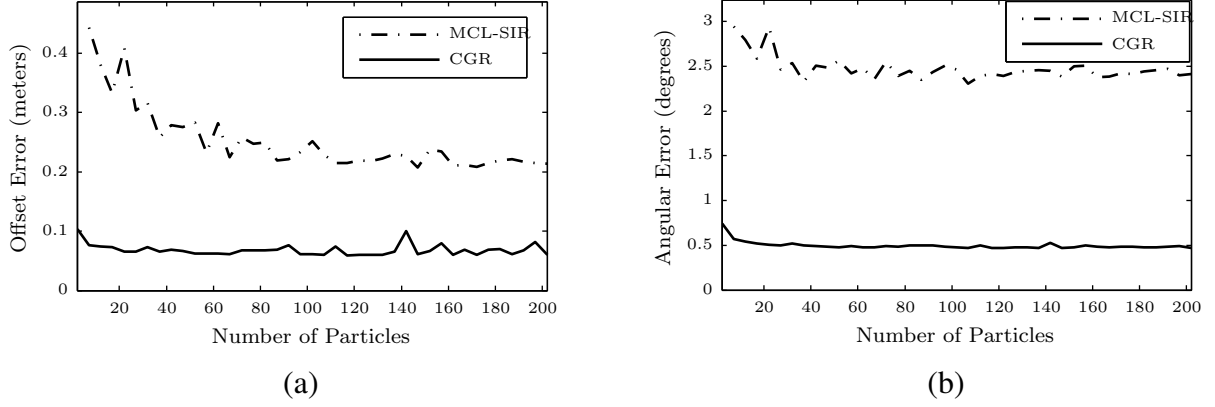


Figure 4.1: The offset (a) and angular (b) errors from the true robot locations for the MCL-SIR and CGR algorithms

was then processed offline with varying number of particles to compare success rates, accuracy and run times for CGR as well as MCL-SIR. At the beginning of every trial, the particles were randomly initialized with errors of up to $\pm 4m$ and $\pm 40^\circ$. The number of particles m was varied from 2 to 302, with 80 trials each for CGR and MCL-SIR for each value of m . The values of f (the observation discount factor) and r (the number of iterations in the Refine step) were 2000 and 3 respectively for all trials. The step size η used was 0.1. The logged data was replayed offline for MCL-SIR and MCL-CGR for 100 times per approach, with a unique random seed each time.

Accuracy Figure 4.1 shows the mean localization errors for the two algorithms for different numbers of particles. Both graphs show that localization using CGR is consistently more accurate than when using MCL-SIR. In particular, localizing using CGR with 20 particles has smaller mean errors than localizing using MCL with even 200 particles.

Another advantage of the CGR algorithm over MCL-SIR is that the CGR algorithm has lower variance across trials. Figure 4.2 shows the 70% confidence interval sizes for the two algorithms, for different numbers of particles. The variance across trials is consistently smaller for the CGR algorithm.

Success Rates A trial was counted as being “successful” if the estimated end location was within $1m$ of the true robot location and if the error at every time step was less than $1m$. Figure 4.3 shows the success rates of the CGR and MCL-SIR algorithms for varying numbers of particles.

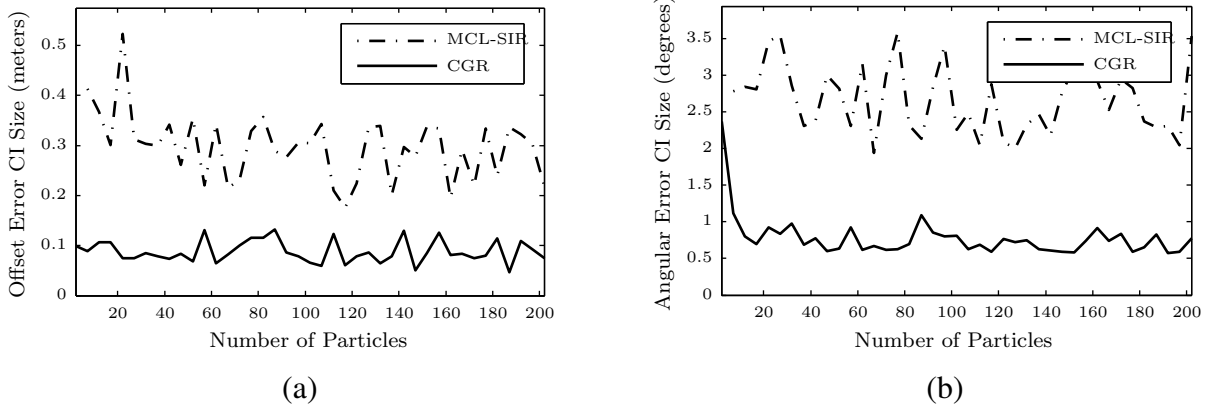


Figure 4.2: The size of the 70% confidence intervals for the offset (a) and angular (b) errors for the MCL-SIR and CGR algorithms

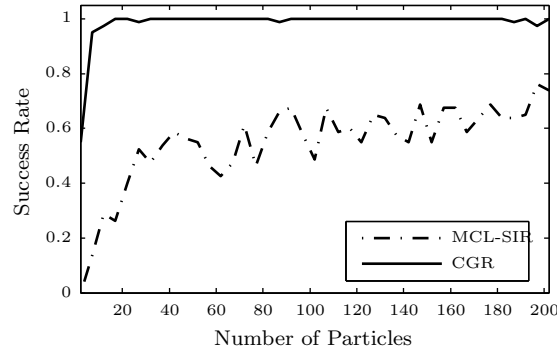


Figure 4.3: Success rates of the CGR and MCL-SIR algorithms

Computational Tradeoffs Figure 4.4 shows the computational tradeoff for running CGR and MCL-SIR as a scatter plot of the localization offset error and the average run time of the complete belief update. Different data points represent different numbers of particles. This figure indicates that for any desired maximum computational runtime, CGR provides lower mean localization errors than MCL-SIR.

CGR with projected 3D Point Cloud Model

We evaluated the performance of our depth camera based localization and navigation algorithms over two sets of experiments. The first set of experiments compare our approach using FSPF CGR localization to three other approaches that use the Kinect for localization by simulating laser rangefinder scans from the Kinect sensor. The second set of long run trials test the effectiveness

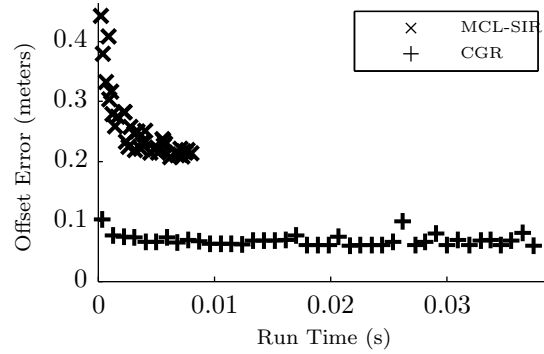


Figure 4.4: Tradeoff between mean localization offset error and normalized computational run times.

of our complete localization and navigation system over a long period of time.

To compare the accuracy in localization of the different approaches using the Kinect, we also used a Hokuyo URG-04LX 2D laser rangefinder scanner as a reference. The autonomous long run trials were run using the Kinect alone for localization and navigation. All trials were run single threaded, on a single core of an Intel Core i7 950 processor.

Comparison of FSPF to Simulated Laser Rangefinder localization We compared our approach using FSPF CGR localization to the following three other CGR based laser rangefinder localization algorithms where the data from the Kinect sensor was used to simulate laser rangefinder scans:

1. Extracting a single raster line from the Kinect depth image, reconstructing the corresponding 3D points, and then down-projecting into 2D to generate the simulated laser rangefinder scans. We call this approach the Kinect-Raster (KR) approach.
2. Randomly sampling locations in the Kinect depth image, and using the corresponding 3D points to simulate the laser rangefinder scan. We call this approach the Kinect-Sampling (KS) approach.
3. Reconstructing the full 3D point cloud from the entire Kinect depth image, and using all these points to generate the simulated laser rangefinder scan. We call this approach the Kinect-Complete (KC) approach.

In each of these cases, we used simulated laser rangefinder readings with the CGR algorithm for localization.

For estimating the error in localization using the Kinect, we used the localization estimates produced by the laser rangefinder CGR localization algorithm for reference. In the KR, KS



Figure 4.5: Combined traces of all successful trials of all approaches: green (KR), red (KS), blue (KC) and black (FSPF). FSPF CGR localization is seen to have the least variation across trials.

and KC approaches, the simulated laser rangefinder scan had a simulated angular range of 180° and an angular resolution of 0.35° , although only part of this scan was populated with useful information due to the limited angular field of view of the Kinect. The number of points sampled in the KS approach was limited to 2000 points, the same as the number of plane filtered points generated by FSPF.

We recorded a log with odometry and data from the Kinect sensor and the laser rangefinder while traversing a path consisting of corridors as well as open areas with unmapped obstacles. There was significant human traffic in the environment, with some humans deliberately blocking the path of the robot. This log was replayed offline for the different localization approaches, running 100 times per approach, with randomly added 20% noise to the odometry data. Figure 4.5 shows the combined traces of the localization estimates for all the successful trials of each of the approaches. A trial was said to be “successful” if the error in localization was less than $1m$ at every time-step of the trial.

Figure 4.7 shows a cumulative histogram of the error in localization using the different approaches. The FSPF approach has significantly less error than the other three approaches. Figure 4.6 shows the cumulative failure rate as a function of the elapsed run time. The FSPF approach has a 2% failure rate at the end of all the trials, whereas all the other approaches start failing after around $20s$ into the trials. The KR, KS, and KC approaches have total failure rates of 82%, 62% and 61% respectively. The abrupt increase in failures around the $20s$ mark corresponds to the time when the robot encounters unmapped objects in the form of humans, tables

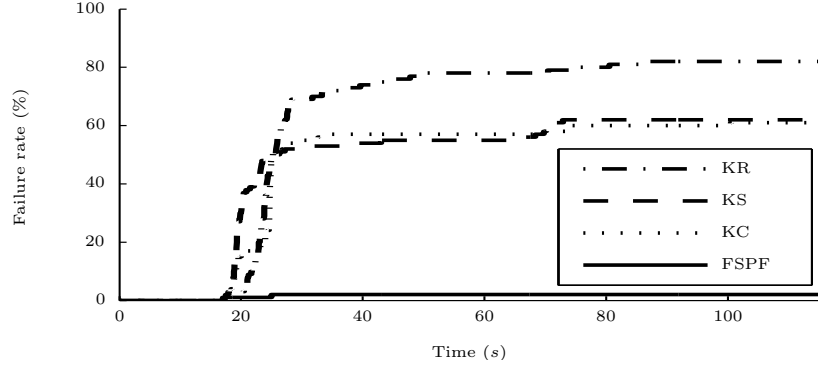


Figure 4.6: Cumulative fractions of the failure rates as a function of elapsed time for all four approaches

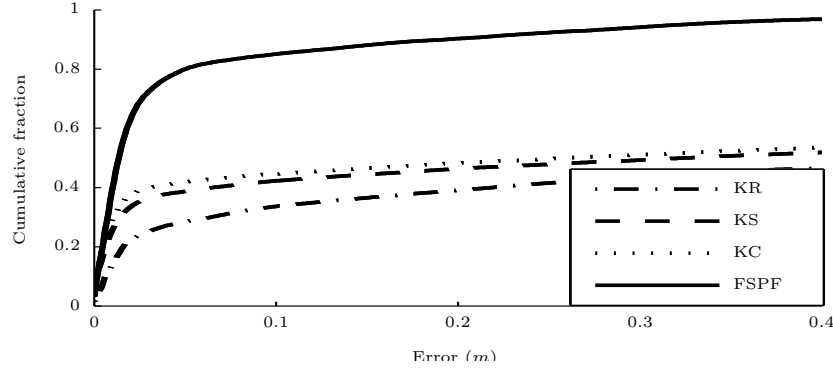


Figure 4.7: Cumulative histogram of errors in localization of the different approaches. For each point (x, y) on the curves, the y value indicates the fraction of time that the algorithm had an error less than the x value.

and chairs in an open area.

To compare the execution times, we kept track of the time taken to process the Kinect depth images (including disk access times), and calculated the ratio of these execution times to the duration of the trials. These values are indicative of the mean CPU processing load on the robot. The values for the different algorithms were 0.01%, 3.6%, 56.6% and 16.3% for the KR, KS, KC, and FSPF approaches respectively.

Insights into Experimental Results We have shown that CGR-FSPF is both more accurate as well as more robust compared to CGR with random sampling (KR), scan extraction (KS), or even using the complete point cloud (KC). We provide here some insights into these results.

The Kinect depth sensor, as described in Chapter 3.2, has a very limited field of view. Therefore, when humans in the environment walking into the path of the robot, they occlude the sen-

sor’s view of map features. Given that all the algorithms compared here are variants of Markov Localization, observations are either evaluated to match the map, or ignored as not matching the map. Therefore, for the robot’s localization estimates to be robust to observations of unmapped objects, the localization algorithm needs to correctly reject observations that do not match the map while still accepting observations that do match the map. The approaches of localizing using random sampling, scan extraction, and the complete point cloud all rely *exclusively* on outlier rejection (Chapter 4.2.2) to reject observations that do not match the map.

However, observations that do not match the map, like humans, chairs, planters, and trash cans are all non-planar objects. FSPF thus effectively ignores all such observations by filtering them out since they are non-planar. Therefore, CGR-FSPF is especially robust to observations of unmapped objects since FSPF efficiently rejects such observations, making it more robust than the alternative approaches.

4.3 Graph-Map WiFi Localization

With the increasing prevalence of wireless LAN, “WiFi,” considerable work has been done on using signal strength measurements from WiFi access points for localization. Approaches based on wireless signal strength propagation models have been proposed [80], but due to the complex interactions of WiFi signals in particular in indoor environments, data-driven signal map based methods have been found more suitable [66]. WiFi-based localization has been shown to be successful in different scenarios [3, 20, 34, 95] in terms of its ability to support humans or robots to identify their locations. We present a WiFi signal strength based localization algorithm that uses a parametric graph based representation of the environment, similar to [34]. In our case, the graph is composed of the discrete points for which we collect the WiFi signal strengths, and the observation model as well as the location hypotheses are constrained to lie strictly on the graph.

4.3.1 WiFi Observation Model and Constrained Particle Filter

Our WiFi Localization algorithm uses Monte Carlo Localization as reviewed earlier to maintain a set of hypotheses of the robot location in real time. As part of the particle filter, we need an observation model which can be used to calculate the probability of making a particular signal strength measurement S at a location x , $P(S|x)$.

The WiFi Observation Model

The WiFi signal strength mean and standard deviations of every access point are measured from each vertex $v \in V_{\text{wifi}}$. Using this data, we model the WiFi signal strength mean and standard deviation as being piecewise linear along the graph, with Gaussian noise. Let x be a location on the edge e ($e = \langle v_i, v_j, \Gamma \rangle$) of the graph, between vertices v_i and v_j . Let $v_i = \langle x_i \rangle$, $v_j = \langle x_j \rangle$. Let \mathbf{W}^x denote the vector of mean signal strengths of every access point as seen from location x . The component \mathbf{W}_k^x of vector \mathbf{W}^x is the mean WiFi signal strength of access point a_k ($a_k \in A_{\text{wifi}}$) as seen from location x . Similarly, let Σ_{wifi}^x denote the standard deviations of the signal strengths of each access point as seen from location x , with $\Sigma_{\text{wifi}k}^x$ being the standard deviation of the signal strength of access point a_k as seen from location x . Hence, the linearly interpolated mean signal strengths vector $\mathbf{W}^x = [\mathbf{W}_1^x \dots \mathbf{W}_{|A_{\text{wifi}}|}^x]$, and the standard deviations vector $\Sigma_{\text{wifi}}^x = [\Sigma_{\text{wifi}1}^x \dots \Sigma_{\text{wifi}|A_{\text{wifi}}|}^x]$ at location x are given by:

$$\mathbf{W}_k^x = \frac{\|x - x_j\| \mathbf{W}_{ik} + \|x - x_i\| \mathbf{W}_{jk}}{\|x_i - x_j\|} \quad (4.9)$$

$$\Sigma_{\text{wifi}k}^x = \frac{\|x - x_j\| \Sigma_{\text{wifi}ik} + \|x - x_i\| \Sigma_{\text{wifi}jk}}{\|x_i - x_j\|} \quad (4.10)$$

During the operation of the robot, let $S = [S_1 \dots S_{|A_{\text{wifi}}|}]$ be a WiFi signal strength observation set, where S_i is the measured WiFi signal strength of access point i . Hence, the probability of making this observation S from a location x , ignoring unobserved access points (*i.e.* $S_i : S_i = 0$) is given by:

$$P(S|x) = \prod_{i=1, S_i \neq 0}^{i=|A_{\text{wifi}}|} \left(\frac{2\epsilon}{\sqrt{2\pi}d_{xi}} \exp \frac{(S_i - \mathbf{W}_i^x)^2}{2(\Sigma_{\text{wifi}i}^x)^2} \right) \quad (4.11)$$

With this estimate of the observation model $P(S|x)$ and the motion model of the robot $P(x_t|x_{t-1}, u_{t-1})$, the localization belief of the robot can be recursively updated with a particle filter.

Constrained Particle Filter

Chapter 4.1 reviewed the implementation of a particle filter. For robot localization using WiFi sensing, we use the observation model introduced in the previous section. We further make one modification to the particle filter. After the update step but before the resample step of the particle filter, we introduce an additional ‘‘Constrain’’ step. The constrain step ensures that the location

Algorithm 5 ‘Constrain’ Algorithm

```

1: for  $i = 1$  to  $|\mathbb{P}|$  do
2:   Let  $p_i = \langle ep_i, d_i, o_i, x_i, y_i, \theta_i, w_i, wc_i \rangle$ 
3:    $fr \leftarrow d_i / (\text{length of edge } ep_i)$ 
4:   if  $fr > 1 - thresh$  or  $fr < thresh$  then
5:     Find edge  $e$  best associated with  $p_i$ 
6:     if  $e \neq ep_i$  then
7:        $ep_i \leftarrow e$ 
8:       Calculate new  $d_i$ 
9:        $o_i \leftarrow 0$ 
10:    end if
11:  end if
12:   $width_i \leftarrow \text{width of edge } ep_i$ 
13:  if  $|o_i| > width_i$  then
14:     $wc_i \leftarrow w_i * \exp(-(abs(o_i) - width_i/2)^2)$ 
15:  else
16:     $wc_i = w_i$ 
17:  end if
18: end for
19: Renormalize weights  $wc_i$ 

```

hypotheses do not violate the graph constraints of the M_{wifi} , in particular the widths and lengths of the edges. Algorithm 5 outlines the Constrain step. Here, the *thresh* term is a threshold term, which is set to 0.1.

After every Update step, the particles are resampled using Sensor Resetting Localization (SRL) [51], subject to a maximum N_{max} and minimum N_{min} number of particles. Algorithm 6 implements this.

To infer the location of the robot based on the particles p_t^i , we perform K-Means clustering of the particles by modifying the algorithm of [17] to take into account the constrained weights wc_i of the particles. The reported location of the robot is then the location of the cluster with the largest weight. During this inference step, we also estimate the “uncertainty” (σ) of the location hypothesis as the weighted standard deviation of all the particles in the cluster with the maximum weight. The “confidence” (c) of the location estimate is estimated as the weight of the cluster with the maximum weight.

To infer the orientation of the robot, we use a mixture of dead reckoning with global wall orientation data, similar to the method described in [86]. The property θ_i of each particle p_i is allowed to evolve using the robot’s motion model when no wall is detected by the robot. When walls are detected by the robot (using the LIDAR sensor), the property θ_i of every particle p_i is

Algorithm 6 ‘Resample’ Algorithm

```

1:  $N \leftarrow 0$ 
2: Sort Particles  $p_i$  in increasing order of  $wc_i$ 
3: for  $i = 1$  to  $|\mathbb{P}|$  do
4:   Let  $p_i = \langle ep_i, d_i, o_i, x_i, y_i, \theta_i, w_i, wc_i \rangle$ 
5:   if  $wc_i < wc_{min}$  then
6:      $N \leftarrow N + 1$ 
7:   end if
8: end for
9:  $N \leftarrow N + |\mathbb{P}| * \min(\frac{N_{max}}{|\mathbb{P}|}, \max(\frac{N_{min}}{|\mathbb{P}|}, wc_{sum}/\kappa))$ 
10: for  $i = 1$  to  $N$  do
11:   Draw  $(x_i, y_i)$  with probability  $P(S|x_i, y_i)$ 
12:    $wc_i \leftarrow P(S|x_i, y_i)$ 
13:    $w_i \leftarrow P(S|x_i, y_i)$ 
14:    $o_i \leftarrow 0$ 
15:   Find edge  $e$  best associated with  $p_i$ 
16:   calculate  $d_i$  for particle  $p_i$  on edge  $e$ 
17: end for

```

updated using global wall orientation information and the location of the particle.

Experimental Evaluation

We evaluate the performance of our WiFi localization algorithm based on a number of parameters.

Convergence In this test, the robot was stopped at 20 randomly chosen locations on the map, and the particles initialized with equal weights randomly distributed across the map. Figure 4.8 shows a plot of the mean uncertainty σ in localization with time. It took a mean of 8s for the location uncertainty to converge to less than 1m.

Accuracy In this test, the robot was again stopped at 20 randomly chosen locations on the map, and the particles initialized with equal weights randomly distributed across the map. The location estimates were allowed to converge till the uncertainty of localization σ stopped decreasing. The reported location estimates were then compared to the ground truth. Table 4.1 sums up the results of this test.

“Incidental” Observations While the robot is performing various tasks, it is reasonable to expect that it would drop WiFi Signal Strength readings from some access points. We wish to

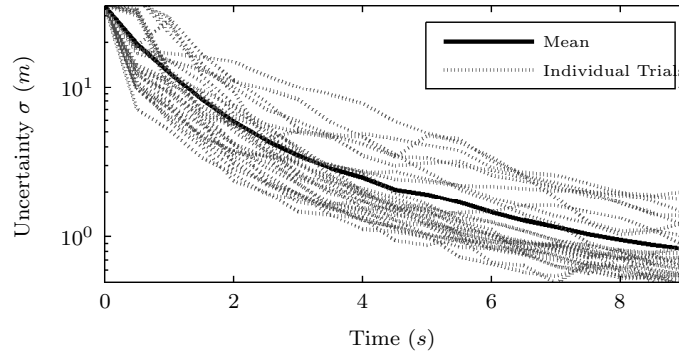


Figure 4.8: Uncertainty σ vs. t : Convergence of Location Estimates. Black, solid: Mean of all trials. Grey, dotted: Individual trials.

Value	Mean	Minimum	Maximum
Localization Error	0.7m	0.2m	1.6m
Localization Uncertainty	0.6m	0.2m	0.9m
Convergence Time	11s	7.4s	16.5s

Table 4.1: Accuracy of WiFi Localization

investigate the impact of dropped signals on the localization accuracy in this experiment. To do so, the robot is stopped at a fixed location on the map, and is allowed to collect at least one signal strength measurement from all WiFi access points in range. Next we selectively and deliberately drop signal strength measurements from all permutations of the access points to simulate dropped signals. Figure 4.9 shows the results of this test. A total of 15 access points were accessible from the location, and even with 9 dropped signals (6 visible access points), the mean error in localization is less than 2m.

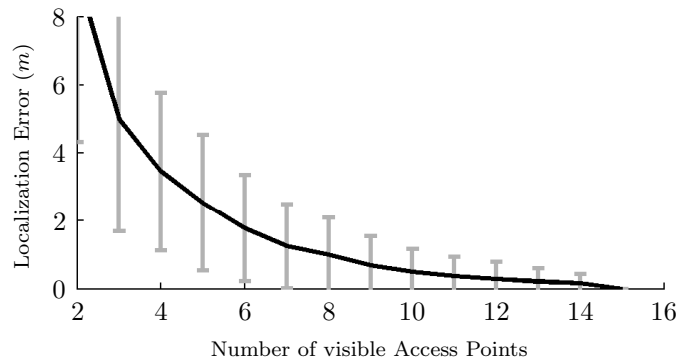


Figure 4.9: Localization Error vs. Number of visible Access Points

4.3.2 Experimental Results

Our WiFi localization and navigation algorithms were tested using CoBot, deployed on the third floor of Wean Hall at Carnegie Mellon University.

Parameters of The Tests and Test Methodology

The graph representation of the map built during the Learning Phase had 223 vertices and 222 edges. There were a total of 106 unique WiFi access points. The particle filter used 500 particles that were initialized with random orientations and locations, and equal weights.

In order to test the localization and navigation system, we ran a series of experiments, each of which were conducted as follows. At the start of each experiment, a list of 12 random locations over the map were generated such that no two successive locations were from the same corridor, and the robot had to autonomously navigate to these locations sequentially.

Results

Each experiment lasted for an average of 28 minutes, and covered an average distance of 818 meters. In total, this experiment was repeated 8 times, adding up to over 3.5 hours of robot navigation time, and a total path length of over 6.5 km. Out of these runs, for 2 of these runs, while the robot navigated between these locations, its true location was manually recorded periodically to use as ground truth for comparison with the localization estimates.

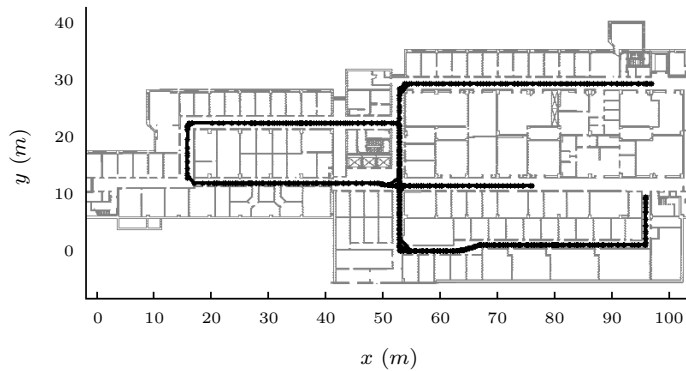


Figure 4.10: Trace of the path traversed by the robot

Figure 4.10 shows a trace of the path followed by the robot during one of the experiments. This trace was reconstructed using the localization estimate of the WiFi localization algorithm. Figure 4.11 shows the evolution of the robot's uncertainty σ over the first ten minutes of this experiment. During all the 8 experiments, the robot encountered a total of 33 action failures,

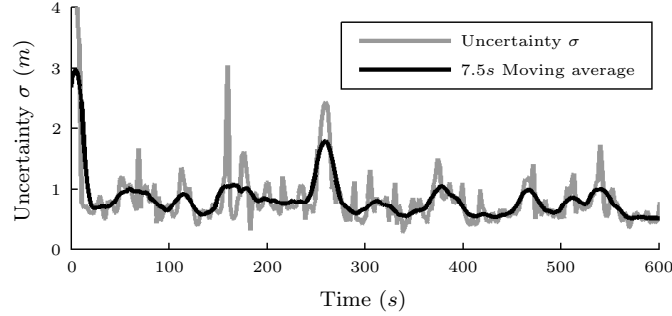
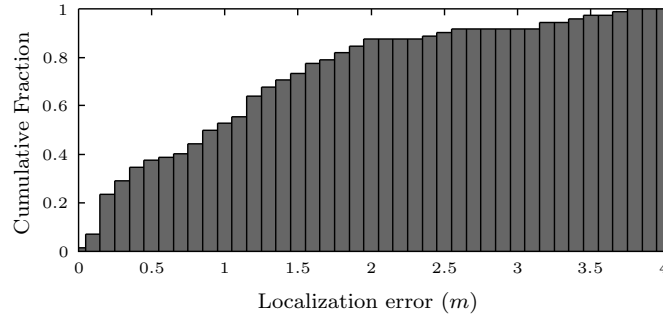
Figure 4.11: σ vs. t : Uncertainty of localization with time

Figure 4.12: Cumulative histogram of localization error

but autonomously recovered from all of them. Figure 4.12 shows a cumulative histogram of the location accuracy of the robot for the 2 experiments with the manually annotated ground truth data. The robot's localization error is a mean of $1.2m$, and the error is less than $1.8m$ for greater than 80% of the time. It is worth noting that while the robot is in motion, the localization has more errors compared to when the robot was stopped. This can be attributed to odometry errors, latency in signal strength readings, and unobserved signal strengths.

4.4 Multi-Sensor Localization for Diverse Environments

In order to overcome individual sensor limitations, a number of algorithms for mobile robot localization have been proposed, which fuse multiple sensor feeds. There are a number of approaches to sensor fusion [47] for robot localization, including merging multiple sensor feeds at the lowest level before being processed homogeneously, and hierarchical approaches to fusing state estimates derived independently from multiple sensors.

In this chapter, we present an approach that instead of merging sensor feeds into a single algorithm, selects one out of a set of localization algorithms, each of which independently process

different sensory feeds [13]. The selection is driven by the results of evaluating the performance of the individual algorithms over a variety of indoor scenarios that the mobile robot will encounter over the course of its deployment. We use three localization algorithms, each using a different sensor: a laser rangefinder based localization algorithm that uses a Corrective Gradient Refinement [8] particle filter, a depth camera based localization algorithm [11], and a WiFi localization algorithm that is a cross between two previously demonstrated approaches [10, 34]. Algorithm selection [72] is a general problem that has been applied to problems like SAT solving [93], and spawned the field of “meta-learning” [82]. Here we apply the problem of algorithm selection to the problem of mobile robot localization.

By running each localization algorithm independently, we can decide to pick one over the others for different scenarios, thus avoiding the additional complexity of fusing dissimilar sensors for a single algorithm. To determine which algorithm is most robust in which scenario, we collected sensor data over the entire map, and evaluated each of the algorithms (with respect to ground truth) over the map and over multiple trials. This data is used to determine which algorithm is most robust at every location of the map. Additionally, by comparing the performance of the algorithms while the robot navigates in corridors and while it navigates in open areas, we show that different algorithms are robust and accurate for different locations on the map.

4.4.1 Multi-Sensor Localization

We use three sensors for localization - a laser rangefinder, a depth camera, and the WiFi received signal strength indicator (RSSI). Each sensor, along with robot odometry, is processed independently of the others. For the laser rangefinder, we use a Corrective Gradient Refinement (CGR) [8] based particle filter. The depth camera observations are processed by the Fast Sampling Plane Filtering [11] algorithm to detect planes, which are then matched to walls in the map to localize the robot using a CGR particle filter. The WiFi RSSI values observed by the robot are used for localization using a Gaussian Processes-Learned WiFi Graph. This WiFi localization algorithm combines the strengths of graph-based WiFi localization [10] with Gaussian Processes for RSSI based location estimation [34].

WiFi localization using a graph based WiFi map [10] provides fast, accurate robot localization when constrained to a graph. In this approach, the mean and standard deviations of WiFi RSSI observations are approximated by linear interpolation on a graph. This leads to a computationally efficient observation likelihood function, with the computation time of each observation likelihood being independent of the number of training instances. The disadvantage is that to construct the WiFi graph map, the robot needs to be accurately placed at every vertex location of the graph to collect WiFi RSSI training examples.

RSSI based localization using Gaussian Processes [34], on the other hand, does not require training observations from specific locations. Given an arbitrary location x_* and the covariance vector k_* between the training locations and x_* , the expected mean μ_{x_*} and variance is given by,

$$\mu_{x_*} = k_*^T (K + \sigma_n^2 I)^{-1} y \quad (4.12)$$

$$\sigma_{x_*}^2 = k(x_*, x_*) - k_*^T (K + \sigma_n^2 I)^{-1} k_*. \quad (4.13)$$

Here, σ_n^2 is the Gaussian observation noise, y the vector of training RSSI observations, and $k(\cdot, \cdot)$ the kernel function used for the Gaussian Process (most commonly a squared exponential). The drawback with this approach is that the computational complexity for Eq.4.12-4.13 grows quadratically with the number of training examples, which is more than 100,000 per access point for our single floor map. Our approach combines the advantages of both algorithms while overcoming each of their limitations. In our approach, training examples are first collected across the map while driving the robot around (not from any specific locations). Next, using Gaussian Processes, the WiFi mean and variance for the nodes of a WiFi Graph (with regularly spaced sample locations on the map) are learned offline. Once the WiFi graph is learned, WiFi mean and variance at locations during run time are estimated by bicubic interpolation across nodes of the graph.

4.4.2 Comparing Localization Algorithms

To evaluate the robustness of the different localization algorithms over a variety of environments, we collected laser scanner, depth camera, and WiFi sensor data while our mobile robot autonomously navigated around the map. The navigation trajectory covered each corridor multiple times, and also covered the open areas of the map. The sensor data thus collected was then processed offline for 100 times by each of the algorithms (since the algorithms are stochastic in nature), and the mean localization error across trials evaluated for each sensor. Figure 4.13 shows the errors in localization when using the three different sensors. It is seen that in corridors, the laser rangefinder based localization algorithm is the most accurate, with mean errors of a few centimeters. The depth camera (Kinect) based localization has slightly higher errors of about ten centimeters while the robot travels along corridors. The WiFi localization algorithm consistently had errors of about a meter across the map.

We then ran additional trials in a large open area of the map. In these trials, the robot started out from the same fixed location, and was manually driven around the open area. The robot was then driven back to the starting location, and the error in localization noted. Figure 4.14a shows part of such a trajectory as the robot was driven around the open area. This was done 20 times

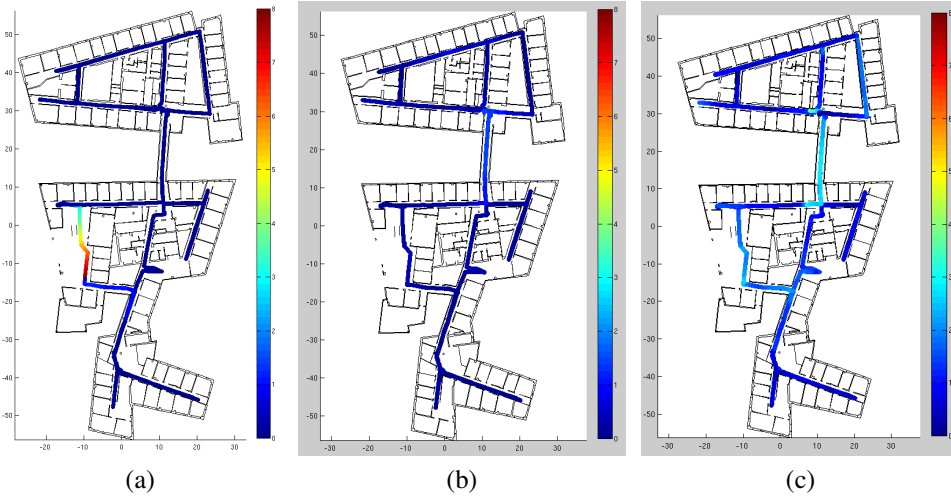


Figure 4.13: Errors in localization using (a) a laser rangefinder, (b) the Kinect, and (c) WiFi, while autonomously driving across the map. The error in localization for every true location of the robot is coded by color on the map. All units are in meters.

for each localization algorithm, and the combined length of all the trajectories was over 2Km, with approximately 0.7Km per algorithm. Figure 4.14b shows a histogram of the recorded errors in localization at the end of the driven trajectories for each localization algorithm. Table 4.2 tabulates the minimum, maximum, and median errors for the different algorithms. It is seen that the laser rangefinder has the smallest minimum error during trajectories that the algorithm tracked successfully, but also has the largest maximum error for trajectories that the algorithm failed to track. The WiFi localization algorithm, however, was consistently able to track the robot's trajectory for every trial, with a median error of less than a meter.

	Laser Rangefinder	Kinect	WiFi
Min Error	0.01	0.17	0.20
Max Error	14.77	3.47	1.88
Median Error	1.83	1.08	0.76

Table 4.2: Errors in localization while using laser rangefinder, Kinect, and WiFi while driving the robot in the open area

4.5 Summary

In this chapter, we introduced two new variants to Markov Localization: Corrective Gradient Refinement (CGR), and Graph-Map WiFi localization. For CGR, we introduced two observation models, one for planar laser rangefinders, one for depth cameras with plane-filtered 3D points

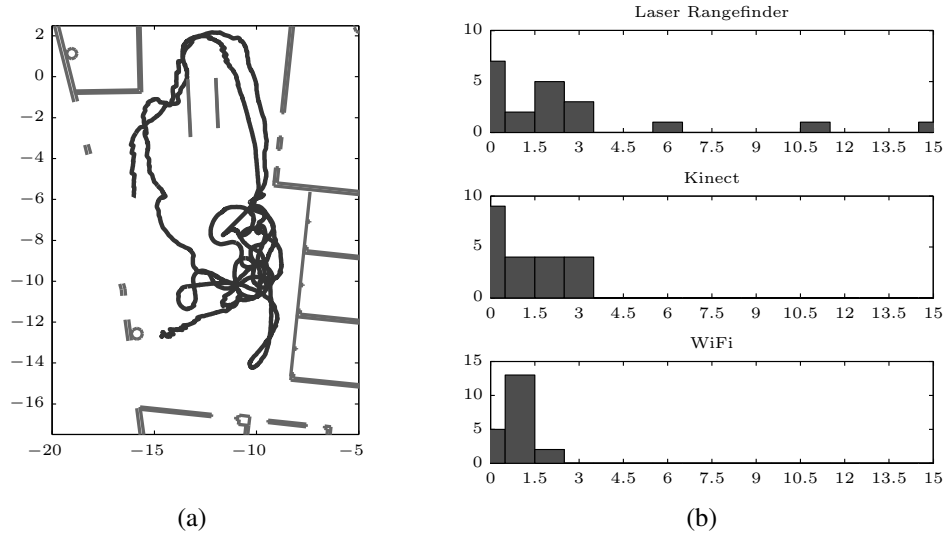


Figure 4.14: Trials in the open area: (a) part of a test trajectory of the robot, and (b) histogram of errors in localization across trials for each algorithm.

generated by FSPF (Chapter 3.3.1). We presented experimental results of localization using the algorithms introduced. Finally, we compared the tradeoffs in localization accuracy in different areas using the different algorithms, and showed that the different algorithms complement each other in terms of accuracy and robustness in different scenarios.

Chapter 5

Episodic Non-Markov Localization

Human environments, particularly indoor ones, have elements of permanence - the architectural structure of a building is unlikely to change over time. However, such environments also include movable and moving objects. Movable objects like furniture and doors appear static, but are likely to be moved around frequently.

We introduce a novel localization algorithm that explicitly reasons about observations of non-mapped objects without saving locally static maps. We call this “non-Markov localization” because it relaxes the Markov assumption that observations are independent given the map, which is the central assumption of Markov localization. Episodic non-Markov localization maintains a belief of the history of pose estimates of the robot over “episodes” of observations of unmapped objects. For every time-step, it classifies observations into those arising from the map (“Long Term Features” (LTFs)), from unmapped static objects (“Short Term Features” (STFs)), or from moving objects (“Dynamic Features” (DFs)). Observations made from LTFs are matched to a static map, while correlations between observations from STFs at different time steps are computed. The belief is framed in terms of a cost function over odometry observations, observations of LTFs related to the static map, and correlations between observations of STFs from different time steps. The maximum likelihood estimate of the belief is incrementally computed over successive steps by functional non-linear least-squares (NLLS) optimization over the cost function. The cost function representation of the belief is distinct from other belief representations such as Particle Filters [29], Extended Kalman Filters (EKF) and Unscented Kalman Filters (UKFs) [45], and Partially Observable Markov Decision Processes (POMDPs) [48]. As we shall show, the cost function representation allows Episodic non-Markov Localization to tractably compute the maximum likelihood estimate of the belief even over long episodes, which would not have been possible using particle filters, EKF, UKF, or POMDPs.

5.1 The Varying Graphical Network Representation

To represent the varying nature of this problem, we introduce a new graphical model, the “Varying Graphical Network” (VGN). As in a dynamic Bayesian network (Chapter 2.1), a VGN includes certain periodically repeating nodes and edges that do not change with the belief. We term these the non-varying nodes and edges. A VGN includes two additional structural elements: varying nodes and varying edges. The presence and structure of the varying nodes and varying edges are not known a-priori, and are estimated jointly with the belief. Since the estimates of the structure may change with the belief, the structure is likely to change as new observations become available.

VGNs provide an accurate representation for non-Markov localization. The presence of LTFs and their relations to the map, and the correlations between successive poses of the robot due to odometry observations are encoded by the non-varying edges and nodes. The presence of STFs and DFs is encoded by the presence of associated varying nodes. The correlations between STFs observed at different time-steps is encoded by the varying edges. Figure 5.1 shows an example instance of a VGN for non-Markov localization.

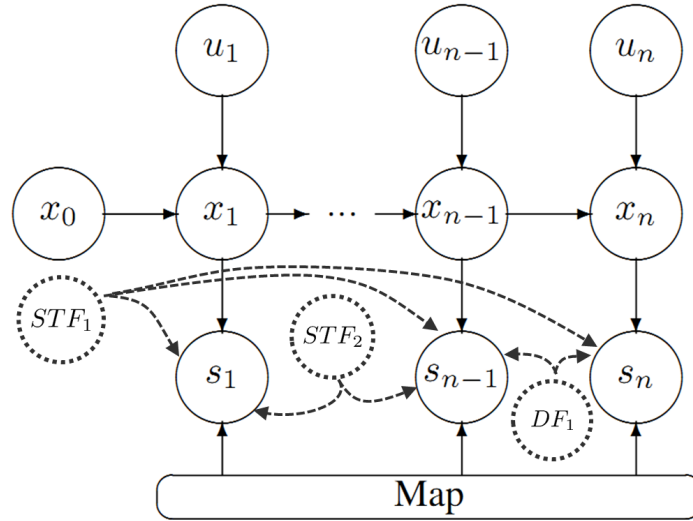


Figure 5.1: An example instance of a Varying Graphical Network (VGN) for non-Markov localization. The non-varying nodes and edges are denoted with solid lines, and the varying nodes and edges with dashed lines. Due to the presence of short term features (STFs) and dynamic features (DFs), the structure is no longer periodic in nature. The exact structure of the graph will depend on the STFs and DFs present.

The varying structure of the VGN, including the varying nodes consisting of the STFs and DFs and the varying edges indicating correlations between STFs and observations, is not enu-

merable a priori since there is no way of predicting beforehand the state of the unmapped static and dynamic objects in an environment. The structure of the VGN is dependent on the exact locations of the STFs, the trajectories of the DFs, and from which poses of the robot's trajectory the STFs and DFs are visible. The STFs and DFs cannot, in general, be added to the map since for any pose in the world, the robot will encounter different STFs and DFs at different times.

In order to explicitly account for the effect of the past robot pose estimates $x_{1:n}$ on future observations and pose estimates, we solve for the belief over the complete history of robot poses, $Bel(x_{1:n})$. The dimension of the state space of the belief over the complete history of robot poses $x_{1:n}$ is dn , where d is the dimension of the state space of each pose x_i . For ground robots, $d = 3$, corresponding to the Cartesian coordinates of the robot location, and the robot angle. Due to the prohibitively large state space, we concentrate on evaluating the belief in the neighborhood of its maximum likelihood estimate (MLE). The MLE of the belief is given by $x_{1:n}^*$ such that

$$x_{1:n}^* = \arg \max_{x_{1:n}} (Bel(x_{1:n})). \quad (5.1)$$

Since the VGN for non-Markov localization has no predefined structure, it might seem that computation of the belief would require storing the complete history of all states and observations since the robot was turned on. However, in practice this is not necessary, as we shall now show.

5.2 Episodes in Non-Markov Localization

Suppose there exists a time step t_i such that all observations and state estimates made after t_i , given x_i , are independent of all prior observations and state estimates:

$$\begin{aligned} P(x_{1:n}|x_0, s_{1:n}, u_{1:n}, M) = \\ P(x_{1:i}|x_0, s_{1:i}, u_{1:i}, M) \times P(x_{i+1:n}|x_i, s_{i+1:n}, u_{i+1:n}, M). \end{aligned} \quad (5.2)$$

This conditional independence implies that there are no STF observations after t_i that correspond to STF observations before t_i . In such a case, the history of states and observations prior to t_i , called the “episode” $t_{0:i-1}$, can be discarded when estimating $Bel(x_{i:n})$ over the episode $t_{i:n}$. We assume such episode-boundary time-steps like t_i exist, allowing real-time non-Markov localization with limited computational resources. Episode-boundary time-steps frequently occur in practice when a robot either does not observe any STFs for one or more time-steps, or if all the STFs prior to the episode-boundary are unrelated to the STFs after, for example when a robot leaves one room and enters another through a doorway. Figure 5.2 shows an example VGN near an episode boundary, highlighting the absence of any varying edges crossing the episode

boundary.

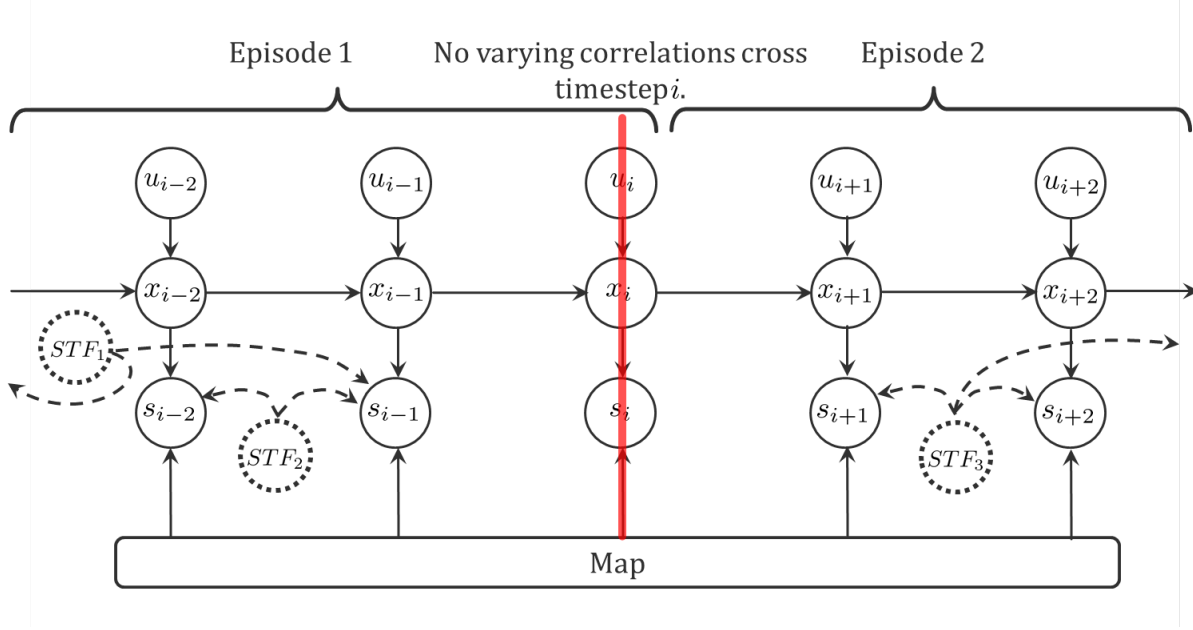


Figure 5.2: An example VGN demonstrating the presence of an episode in non-Markov localization. Note the absence of any varying edges that cross the red line indicating the episode boundary. Hence the pose x_i is an episode boundary, where all previous poses up to x_{i-1} are in the previous episode, and poses x_i and later are in the latest episode. Observations s_{i-1} and older thus no longer need to be stored.

A special case of such a transition is when the robot does not encounter any STFs or DFs for a sequence of time steps from t_i to t_j , so that

$$\begin{aligned}
 P(x_{1:n}|x_0, s_{1:n}, u_{1:n}, M) &= P(x_{1:i}|x_0, s_{1:i}, u_{1:i}, M) \times \\
 &\quad \prod_{k=i+1}^{k=j} P(x_k|x_{k-1}, u_k, s_k, M) \times \\
 &\quad P(x_{j+1:n}|x_j, s_{j+1:n}, u_{j+1:n}, M).
 \end{aligned} \tag{5.3}$$

In this case, from t_i to t_j , the Markov assumptions hold, and Markov localization can be used during this sequence.

Thus, the history of observations and state estimates of the robot can be divided into multiple episodes such that only observations and state estimates of the latest episode need be considered to estimate the latest robot pose x_n .

For the rest of this chapter, to simplify notation, we will refer to observations of LTFs, STFs, and DFs directly as LTFs, STFs, and DFs, respectively, with the understanding that henceforth

they refer to the *observations*, not the actual *objects* that cause them.

5.3 Classification of Observations

For every time-step, the structure of the VGN, based on the classification of the observations into LTFs, STFs and DFs, is re-evaluated prior to updating the MLE of the belief. In this work the sensor we use is a laser rangefinder, so each observation s_i is a set of n_i 2D points $s_i = \{p_j^i\}_{j=1:n_i}$ observed by the robot. We represent the pose x_i of the robot on the map at time-step i as an affine transform T_i that consists of a 2D rotation followed by a 2D translation. The affine transform T_i is computed from the latest estimate of the pose x_i for all poses in the latest episode except for the newest pose x_n . For x_n , the corresponding affine transform T_n is estimated by forward-predicting pose x_{n-1} using odometry observations u_n . Thus, for every 2D point p_j^i observed by the robot in its own reference frame, the corresponding location of the point in the global reference frame of the map is given by $T_i p_j^i$. Each point p_j^i has to be classified as an LTF or as an STF or as a DF, and depending on its classification, results in addition of varying edges to the VGN for non-Markov localization. Figure 5.3 demonstrates the steps in the classification of observations in an example with two poses.

5.3.1 Classification of Long-Term Features

We use a vector map (Chapter 3.5.1) representation $M_{\text{vector}} = \{l_i\}_{i=1:s}$ for the permanent map, consisting of a set of s line segments l_i . To evaluate which of the observed points p_k^i are LTFs, an analytic ray cast (Chapter 3.5.1) is performed from the latest MLE of x_i . The result of the analytic ray cast is a mapping from $p_j^i \rightarrow l_j \in M_{\text{vector}}$, indicating that the line segment l_j from map M_{vector} is the most likely line in the map to be observed by the point p_j^i . Let $\text{dist}(p, l)$ denote the perpendicular distance of point p from the line segment l where both p and l are in the reference frame of the map. The observation likelihood $P(p_j^i | T_i, M_{\text{vector}})$ of the point p_j^i is then given by

$$P(p_j^i | x_i, M_{\text{vector}}) = \exp \left(-\frac{\text{dist}(T_i p_j^i, l_j)^2}{\Sigma_s} \right), \quad (5.4)$$

where Σ_s is the scalar variance of observations, which depends on the accuracy of the sensor used. Thus, given the location of the observed points in the reference frame of the map, observations are classified as LTFs if the observation likelihood of the point given the map is greater than a threshold, $P(p_j^i | x_i, M_{\text{vector}}) > \epsilon_{\text{LTF}}$. Observed points p_j^i that satisfy this condition are

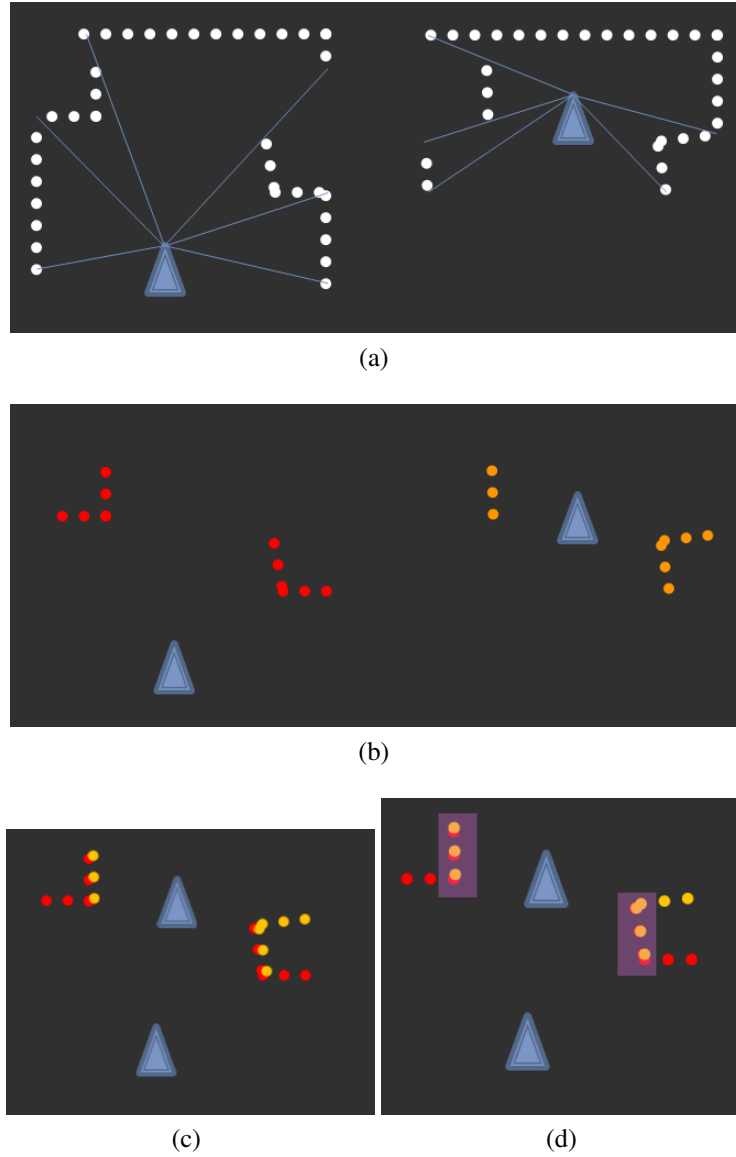


Figure 5.3: An example of classification of observations by EnML from two poses. The robot poses are shown as blue triangles, and the observations are shown as dots. Observations from two poses (a) are first independently compared to the vector map to find LTF observations. The remaining non-LTF observations (b) are aligned (c) with the last pose prior, and (d) matching observations are classified as STFs. The remaining observations are classified as DFs.

classified as LTFs, and those that do not are classified as non-LTFs. The set $\text{LTF}_i \subseteq s_i$ denotes the set of points in s_i that have been classified as LTFs, and $\overline{\text{LTF}}_i$ the set of points that have been classified as non-LTFs. The sets LTF_i and $\overline{\text{LTF}}_i$ are thus given by

$$\text{LTF}_i = \{p_j^i \in s_i | P(p_j^i | x_i, M_{\text{vector}}) > \epsilon_{\text{LTF}}\}, \quad (5.5)$$

$$\overline{\text{LTF}}_i = s_i \setminus \text{LTF}_i. \quad (5.6)$$

5.3.2 Classification of Short-Term Features

Observed points that are classified as non-LTFs could potentially be STFs. To check if an observed point $p_j^i \in \overline{\text{LTF}}_i$ is an STF, it is compared to all non-LTF points observed prior to time-step i to check if they correspond to observations of the same point. Given a point $p_j^i \in \overline{\text{LTF}}_i$ observed at time-step i and another point $p_k^l \in \overline{\text{LTF}}_l$ observed at a previous time-step l , the probability that both the observations correspond to the same point is given by the STF observation likelihood function,

$$P(p_j^i, p_k^l | x_i, x_l) = \exp \left(-\frac{\|T_i p_j^i - T_l p_k^l\|^2}{\Sigma_s} \right) \quad (5.7)$$

where Σ_s is the scalar variance of observations. Therefore, a non-LTF point $p_j^i \in \overline{\text{LTF}}_i$ is classified as an STF if there exists a point $p_k^l \in \overline{\text{LTF}}_l$ from a time-step $l, l < i$ such that $P(p_j^i, p_k^l | x_i, x_l) > \epsilon_{\text{STF}}$:

$$\text{STF}_i = \{p_j^i \in \overline{\text{LTF}}_i | \exists p_k^l \in \overline{\text{LTF}}_l : P(p_j^i, p_k^l | x_i, x_l) > \epsilon_{\text{STF}}\}. \quad (5.8)$$

To speed up the correspondence check, points $p_j^i \in s_i$ from every observation s_i are stored in KD-trees [6]. The STF observation likelihood function for a pair of matching STF observations p_j^i and p_k^l introduces a correlation between observations s_i and s_l since $p_j^i \in s_i$ and $p_k^l \in s_l$. For every such pair of matching STF observations, the VGN of non-Markov localization thus includes a varying node representing the unmapped static object observed as STFs at different time-steps, and a pair of varying edges joining the varying node and each of the observations s_i and s_l .

5.3.3 Classification of Dynamic Features

Observations that are classified as neither LTFs nor STFs correspond to objects that were observed at one particular location for only one particular time-step, and were not observed at any

other time-step at the same location. This implies that these objects are not static, and hence their observations are classified as DFs, $DF_i = s_i \setminus LTF_i \setminus STF_i$. In this work, we do not actively track DFs, so the correlations between observations due to DFs are not used to further refine the belief. However, points that are classified as DFs in one pose, at one iteration of the EnML update may subsequently be matched with points observed at future poses, at which time they will become STFs.

5.4 Cost Function Representation of Belief

As has been described earlier in Chapter 5.1, the state space of the belief in EnML is dn , where d is the dimension of the state space of each pose, and n is the number of poses in the latest episode. There are two challenges to maintaining the belief as thus described:

1. The dimensionality of the state space is *considerably larger* than for other localization algorithms. For example, for a ground robot, while the state space of other localization algorithms is 3 dimensional (x, y, rotation), for an episode length of 10 poses EnML has a state space of 30 dimensions.
2. The dimensionality of the state space is *variable* for EnML. The dimensions of the state space of EnML increases as new poses are added to the latest episode, and decreases when a new episode boundary is discovered, resulting in the latest episode being shortened.

Due to these challenges, commonly used belief representations such as Particle Filters, EKF, UKFs, and POMDPs are ill-suited to being used for EnML. Particle filters and POMDPs are particularly poor choices for the belief representation of EnML since the number of particles required by particle filters, and the size of the POMDP, both scale exponentially with the dimensions of the state space. To tractably estimate the MLE of the belief in EnML, we therefore introduce a cost function representation of the belief. The components of cost function representation, as we shall present, relate directly to the different terms in the expression of the belief, and the MLE can be tractably estimated by functional non-linear least-squares (NLLS) optimization of the cost function.

We now examine the form of the belief in EnML. The belief is given by

$$Bel(x_{1:n}) = P(x_{1:n}|x_0, s_{1:n}, u_{1:n}, M). \quad (5.9)$$

Applying Bayes' rule and separating out the terms involving sensing and odometry, we get

$$Bel(x_{1:n}) \propto P(s_{1:n}|x_{1:n}, M)P(x_{1:n}|x_0, u_{1:n}). \quad (5.10)$$

Note that the odometry observations are independent at each step, so the pose update terms are independent of each other, resulting in

$$Bel(x_{1:n}) \propto P(s_{1:n}|x_{1:n}, M) \prod_{i=1}^{i=n} P(x_i|x_{i-1}, u_i). \quad (5.11)$$

Each observation s_i is either a laser rangefinder scan, or a depth camera obstacle scan. Therefore, s_i is a set of multiple points. Furthermore, these points could be from Long-Term Features, Short-Term Features, or Dynamic Features:

$$s_i = s_i^{\text{LTF}} \cup s_i^{\text{STF}} \cup s_i^{\text{DF}} \quad (5.12)$$

Exploiting the fact that different classes of observations (LTFs, STFs and DFs) are independent of each other, the belief for Episodic non-Markov Localization thus becomes:

$$Bel(x_{1:n}) \propto P(s_{1:n}^{\text{LTF}}|x_{1:n}, M) P(s_{1:n}^{\text{STF}}|x_{1:n}) P(s_{1:n}^{\text{DF}}|x_{1:n}) \prod_{i=1}^{i=n} [P(x_i|x_{i-1}, u_i)] \quad (5.13)$$

We do not track DFs, so the terms involving DFs are ignored and assumed to be take on a constant value. The observations of LTFs, given the pose estimate and the map, are independent of each other, and the observations of different STFs are independent of each other as well, resulting in,

$$Bel(x_{1:n}) \propto \prod_{i=1}^{i=n} [P(s_i^{\text{LTF}}|x_i, M)] P(s_{1:n}^{\text{STF}}|x_{1:n}) \prod_{i=1}^{i=n} [P(x_i|x_{i-1}, u_i)] \quad (5.14)$$

$$\propto P(s_{1:n}^{\text{STF}}|x_{1:n}) \prod_{i=1}^{i=n} [P(s_i^{\text{LTF}}|x_i, M) P(x_i|x_{i-1}, u_i)] \quad (5.15)$$

$$\propto \prod_{j=1}^{i=m} [P(s_{1:n}^{\text{STF}_j}|x_{1:n})] \prod_{i=1}^{i=n} [P(s_i^{\text{LTF}}|x_i, M) P(x_i|x_{i-1}, u_i)] . \quad (5.16)$$

Note that in this expression, there are two parts: the first part consists of a product of m terms $P(s_{1:n}^{\text{STF}_j}|x_{1:n})$ corresponding to the observation of STFs, and the second part consists of a product of n terms $P(s_i^{\text{LTF}}|x_i, M) P(x_i|x_{i-1}, u_i)$. The first part consists of the terms that arise from the *varying nodes* and *varying edges* of the Varying Graphical Network, while the second part consists of the terms that arise from the *repeating nodes* and *repeating edges* of the Varying Graphical Network.

Recall from Equation 5.1 that we wish to compute the Maximum Likelihood Estimate (MLE) $x_{1:n}^*$ such that

$$x_{1:n}^* = \arg \max_{x_{1:n}} (Bel(x_{1:n})) \quad (5.17)$$

$$= \arg \max_{x_{1:n}} \left(\prod_{j=1}^{i=m} [P(s_{1:n}^{\text{STF}_j} | x_{1:n})] \prod_{i=1}^{i=n} [P(s_i^{\text{LTF}} | x_i, M) P(x_i | x_{i-1}, u_i)] \right). \quad (5.18)$$

To estimate the MLE of the belief, we first convert the belief from a probability distribution representation to a cost function representation C such that

$$\begin{aligned} Bel(x_{1:n}) &= P(x_{1:n} | x_0, s_{1:n}, u_{1:n}, M) \\ &\propto \exp(-C(x_{1:n} | x_0, s_{1:n}, u_{1:n}, M)). \end{aligned} \quad (5.19)$$

The cost function C consists of a sum of m sub-cost functions c_j^{STF} corresponding to the STF terms $P(s_{1:n}^{\text{STF}_j} | x_{1:n})$, n sub-cost functions c_i^{LTF} corresponding to the LTF terms $P(s_i^{\text{LTF}} | x_i, M)$, and n sub-cost functions c_i^{odom} corresponding to the odometry terms $P(x_i | x_{i-1}, u_i)$. The complete expression for C is thus given by

$$\begin{aligned} C(x_{1:n} | x_0, s_{1:n}, u_{1:n}, M) &= \sum_{j=1}^{j=m} (c_j^{\text{STF}}(s_{1:n} | x_{i:n})) + \\ &\quad \sum_{i=1}^{i=n} (c_i^{\text{LTF}}(s_i | x_i, M) + c_i^{\text{odom}}(x_i | x_{i-1}, u_i)). \end{aligned} \quad (5.20)$$

The MLE is therefore computed by minimizing the cost function as

$$x_{1:n}^* = \arg \min_{x_{1:n}} (C(x_{1:n} | x_0, s_{1:n}, u_{1:n}, M)). \quad (5.21)$$

There are two important properties of the cost-function representation of the belief that make it a better representation for computing the MLE rather than the probability distribution representation.

Quadratic Nature of Sub-Cost Functions An important property of the sub-cost functions that we exploit is that they are all *purely quadratic in form*. The quadratic forms of the sub-cost functions are introduced in the next section. Since all the sub-cost functions are purely quadratic, Equation 5.21 can be solved by functional non-linear least-squares (NLLS) optimization.

Numerical Stability The sub-cost functions scale as the log of the probability values of the corresponding terms in the belief, and are thus less affected by numerical precision errors. Furthermore, the sub-cost values are *added* instead of *multiplied*, as in the case of probability values, thus further avoiding numerical precision errors.

Before we proceed to present the forms of the sub-cost functions, we first summarize here the steps we took in deriving a cost-function form of the belief.

1. The belief is first factored into the odometry and sensing terms (Equation 5.10).
2. The sensing terms are further factorized into terms corresponding to the STFs and LTFs (Equation 5.13).
3. The LTF and odometry terms are decomposed into independent terms for each pose (Equation 5.16).
4. The belief is then represented in terms of a cost function, with each independent term in the belief corresponding to a sub-cost function (Equation 5.20).
5. The MLE is then computed by functional NLLS optimization of the cost function (Equation 5.21).

The computation of each individual sub-cost function is the subject of the next section.

5.5 Computation of Sub-Cost Functions

The sub-cost functions are computed and stored as *function objects* [5], not as function values, thus allowing the $\arg \min$ over \mathcal{C} to be computed in real time by non-linear optimization of cost functions expressed as function objects [1]. Since the optimizer performs algorithmic differentiation [40] of the function objects, it provides the same accuracy as symbolic optimization, with much lower computational requirements.

The odometry sub-cost function c_i^{odom} relating poses x_i and x_{i-1} with observed odometry u_i is given by

$$c_i^{\text{odom}} = (\mathbf{x}_i - \mathbf{x}_{i-1} - \mathbf{u}_i)^T \Sigma_o^{-1} (\mathbf{x}_i - \mathbf{x}_{i-1} - \mathbf{u}_i) \quad (5.22)$$

where \mathbf{x}_i denotes the vector representation of pose x_i , and Σ_o is the covariance on odometry based on the motion model of the robot.

The sub-cost function for LTFs for pose i , c_i^{LTF} depends on the observations of LTFs $s_i^{\text{LTF}} \in s_i$ made at time-step t_i . Let there be m LTF observations in s_i^{LTF} , and for each point $p_i^j \in s_i^{\text{LTF}}$, $j \in [1, m]$, let the corresponding line segment in the vector map M_{vector} be $l_j \in M_{\text{vector}}$. Let $\text{dist } p, l$

indicate the perpendicular distance of point p from line segment l . The LTF sub-cost function for pose i is then given by

$$c_i^{\text{LTF}} = \sum_{j=1}^{j=m} \left[\frac{\text{dist}(T_i p_i^j, l_j)^2}{\Sigma_s} \right], \quad (5.23)$$

where T_i denotes the affine transform for pose x_i as described in Chapter 5.3, and Σ_s is the variance of the range sensor.

Let STF j consist of a pair of points from poses i and k , with point $p_i \in s_i^{\text{STF}}$ being the point from pose i and point $p_k \in s_k^{\text{STF}}$ from pose k . The sub-cost function c_j^{STF} is then given by

$$c_j^{\text{STF}} = \frac{\|T_i p_i - T_k p_k\|^2}{\Sigma_s}. \quad (5.24)$$

Note that all the sub-cost functions (Equations 5.22, 5.23, 5.24) are purely quadratic in form, as required for non-linear least-squares optimization as described in Chapter 5.4.

5.6 Structure Analysis

Thus far we have introduced the EnML algorithm as agnostic of the algorithm that performs the non-linear functional optimization of the cost function C . In principle, any general non-linear optimization algorithm may be used to solve Equation 5.21 to estimate the MLE $x_{1:n}^*$. We use the Ceres-Solver [1] to solve Equation 5.21 using the Levenberg-Marquardt (LM) [55, 56] algorithm with Sparse Normal Cholesky [21] as the linear solver. In this section, we examine the structure of EnML from the view of the non-linear solver solving Equation 5.21, how the concept of episodes in the VGN translates to the solutions of the non-linear solver, and how the structure of the problem relates to SLAM and Markov localization.

The cost function C can be re-cast as

$$C(x_{1:n}) = \frac{1}{2} \|F(x_{1:n})\|^2. \quad (5.25)$$

This construction is possible because all of the sub-cost functions c_i (Equations 5.22, 5.23, 5.24) that comprise C as $C = \sum_i (c_i)$ are of the form $c_i = \mathbf{a}^T \Sigma \mathbf{a}$. Therefore, the corresponding entries in F would be $\Sigma^{-\frac{1}{2}} \mathbf{a}$. In this form, Equation 5.21 is rephrased as

$$x_{1:n}^* = \arg \min_{x_{1:n}} \frac{1}{2} \|F(x_{1:n})\|^2. \quad (5.26)$$

The solution to this equation is thus a non-linear least squares optimization. Global solutions to this equation are computationally intractable due to the presence of local minima arising from non-linearities in the arguments \mathbf{a} for each sub-cost function c_i . However, the local optimum may be found in the neighborhood of an initial estimate. For EnML, the initial estimate for the solution of $x_{1:n}^*$ is taken from the last estimates of $x_{1:n-1}^*$ and the odometry-predicted estimates of x_n .

Non-linear least-squares solutions to Equation 5.26 by LM are computed iteratively, with each iteration updating x^* by Δx , where Δx is the solution of the equation

$$J^T J \Delta x = J^T F. \quad (5.27)$$

Here, J is the Jacobian of F , the elements of which are given by

$$J_{ij}(x) = \frac{\delta F_i}{\delta x_j}(x). \quad (5.28)$$

The matrix $J^T J$ is called the information matrix \mathcal{I} . The solution to Equation 5.27 may be found by inverting the information matrix, although in practice alternative solutions like the QR decomposition or LU decomposition of \mathcal{I} yield more numerically stable as well as computationally efficient solutions, in particular for sparse matrices. The structure of the information matrix \mathcal{I} , as we shall show, indicates the correlations between the poses of the robot in the EnML episode. We shall also show that the computational complexity of EnML scales from being comparable to Markov Localization in the absence of short-term features, to being comparable to pose graph SLAM when only short-term features are visible.

We present here a simple scenario of EnML running on a robot. Figure 5.4 shows a robot navigating in an environment with four static unmapped objects. There are six poses of the robot over time, and at each pose the robot observes one or more of the unmapped objects, as well as part of the static map. Thus, in this scenario, there would be 15 sub-cost functions: 5 sub-cost functions $c_1^o - c_5^o$ for the odometry between successive poses, 4 sub-cost functions for the STFs $c_0^{\text{STF}} - c_3^{\text{STF}}$, and 6 sub-cost functions $c_0^{\text{LTF}} - c_5^{\text{LTF}}$ for the LTFs from each pose.

The structure of the Jacobian and the information matrix is thus as shown in Table 5.1. There are several interesting features about the structure of the information matrix. It is sparse and nearly block-diagonal with two major blocks $x_{0:2} \times x_{0:2}$ and $x_{3:5} \times x_{3:5}$. In fact, the only non-block diagonal elements arise from the relation between x_2 and x_3 due to the odometry sub-cost functions. Note also that in this scenario, x_3 would be an episode boundary: all the observations of STFs at and after pose x_3 are independent of the STFs observed before x_3 . This is not just coincidence. The existence of episode boundaries enforce this almost block-diagonal structure to

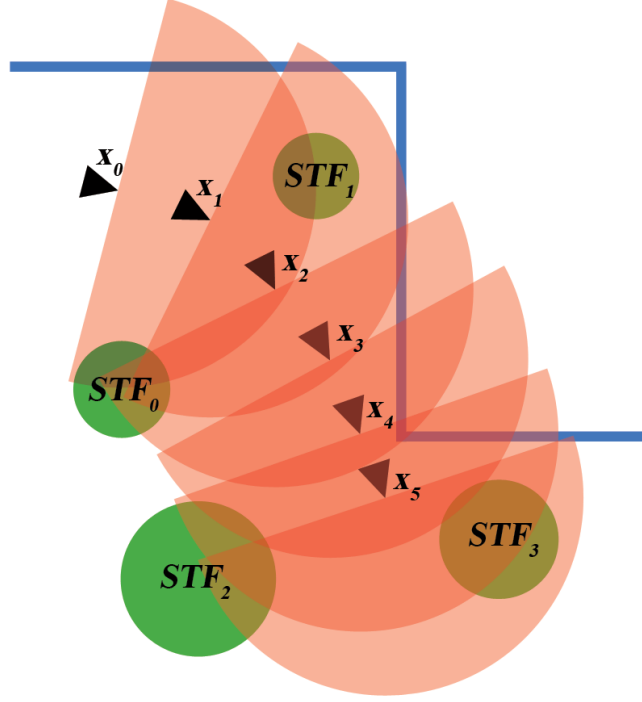


Figure 5.4: An example robot scenario running EnML. There are 6 poses $x_0 - x_5$ denoted by the triangular robot markers, along with the maximum sensor range denoted by the translucent orange semicircles. There are four STFs, $STF_0 - STF_3$, shown by the green circles. The long-term static map is shown by the blue lines.

the information matrix by virtue of the fact that *except for odometry, correlations between poses arise only from STFs*, and by definition, no such correlations may cross the episode boundary. This nearly block-diagonal structure has a particularly simple sub-solution. Consider the information matrix \mathcal{I} as composed of blocks A, B, C, D such that

$$\mathcal{I} = \begin{bmatrix} A & B \\ B^T & D \end{bmatrix}, \quad (5.29)$$

$$A = J_{0:14,0:2}^T J_{0:14,0:2}, \quad (5.30)$$

$$B = J_{0:14,0:2}^T J_{0:14,3:5}, \quad (5.31)$$

$$D = J_{0:14,3:5}^T J_{0:14,3:5}. \quad (5.32)$$

Here, the notation $J_{a:b,c:d}$ refers to the sub-matrix of J consisting of rows a to b and columns c to

d. The information matrix \mathcal{I} may then be inverted by the matrix block-inversion formula,

$$\mathcal{I}^{-1} = \begin{bmatrix} A & B \\ B^T & D \end{bmatrix}^{-1} \quad (5.33)$$

$$= \begin{bmatrix} S_D^{-1} & -A^{-1}BS_A^{-1} \\ -D^{-1}B^TS_D^{-1} & S_A^{-1} \end{bmatrix} \quad (5.34)$$

where S_A and S_D are the Schur complements of A and D respectively, given by

$$S_A = D - B^T A^{-1} B, \quad (5.35)$$

$$S_D = A - BD^{-1}B^T. \quad (5.36)$$

Let the only non-zero element in the lower left corner of B be given by b . We now investigate the structure of S_A .

	x_0	x_1	x_2	x_3	x_4	x_5
c_1^o	X	X
c_2^o	.	X	X	.	.	.
c_3^o	.	.	X	X	.	.
c_4^o	.	.	.	X	X	.
c_5^o	X	X
c_0^{STF}	X	X	X	.	.	.
c_1^{STF}	X	X
c_2^{STF}	.	.	.	X	X	X
c_3^{STF}	.	.	.	X	X	X
c_0^{LTF}	X
c_1^{LTF}	.	X
c_2^{LTF}	.	.	X	.	.	.
c_3^{LTF}	.	.	.	X	.	.
c_4^{LTF}	X	.
c_5^{LTF}	X

	x_0	x_1	x_2	x_3	x_4	x_5
x_0	X	X	X	.	.	.
x_1	X	X	X	.	.	.
x_2	X	X	X	X	.	.
x_3	.	.	X	X	X	X
x_4	.	.	.	X	X	X
x_5	.	.	.	X	X	X

Table 5.1: The structure of the Jacobian (left) and the information matrix (right) for EnML in the example robot scenario. Non-zero entries are marked with a 'x', while zero entries are marked with a '.'.

$$S_A = D - \begin{bmatrix} 0 & 0 & b \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} A^{-1} \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ b & 0 & 0 \end{bmatrix} \quad (5.37)$$

$$= D - b^2 \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} A_{00}^{-1} & A_{01}^{-1} & A_{02}^{-1} \\ A_{10}^{-1} & A_{11}^{-1} & A_{12}^{-1} \\ A_{20}^{-1} & A_{21}^{-1} & A_{22}^{-1} \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix} \quad (5.38)$$

$$= D - b^2 \begin{bmatrix} A_{20}^{-1} & A_{21}^{-1} & A_{22}^{-1} \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix} \quad (5.39)$$

$$= D - b^2 \begin{bmatrix} A_{22}^{-1} & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad (5.40)$$

Hence the computation of the Schur complements S_A, S_D are trivial single-operation computations, given A^{-1} and D^{-1} . Furthermore, for cases where $b^2 \ll D_{00}$, the Schur complement may be approximated as $S_A \approx D$, further simplifying the inversion of \mathcal{I} . The most computationally expensive operations in the inversion of \mathcal{I} are therefore the computations of A^{-1} and D^{-1} , which can be computed independently of each other. Thus, *across an episode boundary, the computations of the inverse of the dominant sub-matrices of \mathcal{I} , $S_D^{-1} \approx A^{-1}$ and $S_A^{-1} \approx D^{-1}$, are independent of each other.*

We next examine the example scenario if the robot did not have a static map. We call this the “no-map” example scenario. The three linear features that would otherwise have been considered LTFs would instead be considered by EnML to be STFs. The associated cost functions are denoted by $c_4^{\text{STF}}, c_5^{\text{STF}}, c_6^{\text{STF}}$. The Jacobian and information matrix for the no-map example scenario are thus listed in Table 5.2. Note that the LTF cost functions are zero for all the poses. Alternatively, if the robot were running pose graph SLAM (*e.g.*, [68]), all the features, including what EnML considered to be STFs and LTFs would be considered landmark features, and give rise to correlations between the poses. The corresponding Jacobian and information matrix are thus shown in Table 5.3. Note that the structure of the information matrix is identical between EnML in the no-map scenario, and that of SLAM. Thus, *in the absence of a static map, the structure of the information matrix, and hence the computational complexity of EnML is identical to that of pose graph SLAM.*

Finally, we consider the case where the STFs are not present in the example scenario. This

	x_0	x_1	x_2	x_3	x_4	x_5
c_1^o	X	X
c_2^o	.	X	X	.	.	.
c_3^o	.	.	X	X	.	.
c_4^o	.	.	.	X	X	.
c_5^o	X	X
c_0^{STF}	X	X	X	.	.	.
c_1^{STF}	X	X
c_2^{STF}	.	.	.	X	X	X
c_3^{STF}	.	.	.	X	X	X
c_4^{STF}	X	X
c_5^{STF}	.	X	X	X	X	.
c_6^{STF}	X
c_0^{LTF}
c_1^{LTF}
c_2^{LTF}
c_3^{LTF}
c_4^{LTF}
c_5^{LTF}

	x_0	x_1	x_2	x_3	x_4	x_5
x_0	X	X	X	.	.	.
x_1	X	X	X	X	X	.
x_2	X	X	X	X	X	.
x_3	.	X	X	X	X	X
x_4	.	X	X	X	X	X
x_5	.	.	.	X	X	X

Table 5.2: The structure of the Jacobian and the information matrix for EnML in the no-map example robot scenario.

is a scenario perfect for Markov localization. There would be no sub-cost functions of EnML for STFs, and the corresponding Jacobian and information matrix for EnML would thus be as given in Table 5.4. Note here that the information matrix is tridiagonal in nature, thus allowing efficient inversion of \mathcal{I} by the tridiagonal matrix algorithm in $O(n)$ operations for n poses, just like Markov Localization, since Markov Localization operates at each pose independently of the others. Thus, *in the absence of short-term features, the computational complexity of EnML is similar to that of Markov Localization.*

Thus, the computational complexity of EnML scales from being comparable to Markov Localization in the absence of short-term features, to being comparable to pose graph SLAM when only short-term features are visible.

5.7 Results

We present two sets of experiments to evaluate the performance of episodic non-Markov localization and compare it with alternative approaches to robot localization in varying environments.¹

¹ The authors would like to thank Gian Diego Tipaldi and Daniel Meyer-Delius for sharing the Freiburg-Parkinglot dataset, and Tom Duckett for sharing the Örebro-Longterm dataset.

	x_0	x_1	x_2	x_3	x_4	x_5
c_1^o	X	X
c_2^o	.	X	X	.	.	.
c_3^o	.	.	X	X	.	.
c_4^o	.	.	.	X	X	.
c_5^o	X	X
c_0^l	X	X	X	.	.	.
c_1^l	X	X
c_2^l	.	.	.	X	X	X
c_3^l	.	.	.	X	X	X
c_4^l	X
c_5^l	.	X	X	X	X	.
c_6^l	X

	x_0	x_1	x_2	x_3	x_4	x_5
x_0	X	X	X	.	.	.
x_1	X	X	X	X	X	.
x_2	X	X	X	X	X	.
x_3	.	X	X	X	X	X
x_4	.	X	X	X	X	X
x_5	.	.	.	X	X	X

Table 5.3: The structure of the Jacobian matrix and the information matrix for pose graph SLAM in the example robot scenario.

5.7.1 Freiburg-Parkinglot Dataset

We compared localization using our approach to localization using Temporary Maps [58] and RBPFs with dynamic occupancy grids [89] by running episodic non-Markov localization on the Freiburg-Parkinglot dataset collected on an outdoor robot driven around a parking lot at University of Freiburg [89]. This dataset consists of 12 runs in an outdoor parking lot over the course of the day totaling over 5.8 km traversed and presents a challenging environment where there are very few permanent features and many changes over the course of the day due to the arrival and departure of cars in the lot. For every run, a ground truth estimate was determined independently of the other runs by running static SLAM offline with manual corrections. To run episodic non-Markov localization on the Freiburg-Parkinglot dataset we extracted the dominant linear features from the results of running SLAM on run 01 of the dataset. This map was then used as the static map for all subsequent runs.

As a baseline, the ground truth maps for each of the runs were used along with Monte-Carlo Localization to localize the robot for every run. Table 5.5 lists the errors in localization with respect to ground truth for localization using the baseline (MCL-GT), Monte-Carlo Localization using a static map (MCL-S), Temporary Maps [58] (MCL-TM), RBPFs with dynamic occupancy grids [89] (RBPF), and episodic non-Markov localization (EnML). The entries for MCL-GT, MCL-S, MCL-TM and RBPF are reproduced from the results of [89]. Episodic non-Markov localization has smaller localization errors for every run compared to the other online algorithm including MCL-S, MCL-TM and RBPF and even outperforms the Monte-Carlo Localization baseline (MCL-GT) for some of the runs. Furthermore, as a deterministic algorithm, our

	x_0	x_1	x_2	x_3	x_4	x_5
c_1^o	X	X
c_2^o	.	X	X	.	.	.
c_3^o	.	.	X	X	.	.
c_4^o	.	.	.	X	X	.
c_5^o	X	X
c_0^{LTF}	X
c_1^{LTF}	.	X
c_2^{LTF}	.	.	X	.	.	.
c_3^{LTF}	.	.	.	X	.	.
c_4^{LTF}	X	.
c_5^{LTF}	X

	x_0	x_1	x_2	x_3	x_4	x_5
x_0	X	X
x_1	X	X	X	.	.	.
x_2	.	X	X	X	.	.
x_3	.	.	X	X	X	.
x_4	.	.	.	X	X	X
x_5	X	X

Table 5.4: The structure of the Jacobian and the information matrix for EnML in the all-mapped example robot scenario.

approach does not have any variance in localization over different runs with the same log, unlike the stochastic algorithms (MCL-S, MCL-TM, RBPF) which exhibit variance across trials [89]. Figure 5.5 shows the trajectory of the robot during run 11 of the Freiburg-Parkinglot dataset, as estimated by episodic non-Markov localization and compared to ground truth.

Figure 5.6 shows two selected snapshots of episodic non-Markov localization running on the Freiburg-Parkinglot dataset. The snapshots show the classification of the observations as originating from LTFs, STFs and DFs. The snapshots demonstrate episodic non-Markov localization correctly identifying the parked cars as STFs and the pedestrian and moving car as DFs.

5.7.2 Örebro-Longterm Dataset

We ran episodic non-Markov localization on the Örebro-Longterm dataset, which has previously been used to empirically evaluate localization using Dynamic Maps [7], and Dynamic Pose Graph SLAM [91]. This dataset was collected over a span of five weeks by driving a robot around an office-like environment, covering a total distance of 9.6 km.

As in the previous experiment, we estimated the static long-term map by running SLAM on the first run. Figure 5.7 shows the traces of the robot as estimated using episodic non-Markov localization over all the runs. Despite the changes in the environment over the five-week period, our approach was successfully able to localize the robot without having to maintain up-to-date maps of the environment. This experiment demonstrates that even in a varying environment, using only a static map of LTFs, episodic non-Markov localization is successfully able to localize a robot without having to maintain maps of the exact state of the environment.

Run	Length	MCL-GT	MCL-S	MCL-TM	RBPF	EnML
01	503.33	0.04	0.18	0.25	0.09	0.015
02	497.74	0.03	0.18	0.16	0.08	0.018
03	496.25	0.04	0.09	0.63	0.05	0.016
04	487.80	0.02	0.08	0.63	0.04	0.018
05	494.78	0.02	0.06	0.51	0.03	0.024
06	489.89	0.02	0.09	0.21	0.02	0.024
07	488.10	0.02	0.07	0.44	0.03	0.023
08	488.39	0.02	0.09	0.59	0.02	0.022
09	479.84	0.02	0.07	0.49	0.03	0.026
10	484.06	0.02	0.09	0.32	0.03	0.036
11	484.88	0.03	0.10	0.47	0.05	0.037
12	479.97	0.03	0.15	0.23	0.06	0.048
Total	5875.0	0.03	0.10	0.41	0.04	0.025

Table 5.5: Localization Squared Errors (m^2) for the Freiburg-Parkinglot Dataset

5.7.3 Deployments of the CoBots

Episodic non-Markov Localization has been deployed on all the CoBots over part of the 1000km Challenge (Chapter 7), and has been used to localize the robots in many different environments spanning multiple floors across multiple buildings. We present here some selected results, and the impact of using EnML over the robot deployments.

Two floors in our environment, GHC4 and GHC5 are particularly challenging for autonomous robots deployed over long periods of time. These floors have significant pedestrian traffic, being on the ground level and connecting different parts of the campus. GHC4 has an atrium with several unusual large movable lounge chairs which obstruct the robot’s view of the walls. GHC5 has large areas with tiles which provide for very poor robot odometry due to wheel slippage. Due to these challenges, prior to the introduction of EnML, the deployments of the CoBots in GHC4 and GHC5 were severely limited - the robots frequently required operator intervention as they got lost or uncertain of their localization. However, since January 2014 when EnML was deployed on the robots, they have been deployed successfully without trouble. Table 5.6 shows the marked difference in the scope of deployments of the CoBots in these challenging areas using CGR (Chapter 4.2.1) and after deploying EnML. While EnML contributed to 68.1% of the total autonomous distance traversed by the CoBots over the 1000km Challenge, on the GHC4 and GHC5 floors, EnML contributed over 93% of the autonomous distance traversed. The disproportionately large contributions of EnML to the distances traversed by the CoBots on GHC4 and GHC5 are entirely due to the increased reliability and robustness of localization of the robots in the presence of the abundant moving and movable objects on those floors.

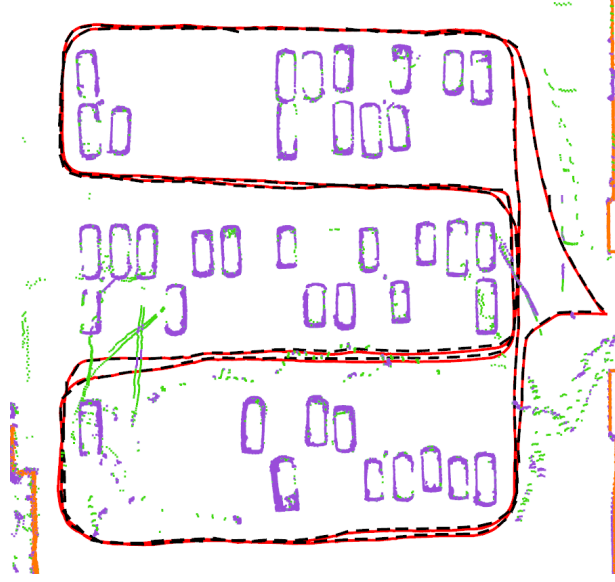


Figure 5.5: The trajectory of the robot during run 11 of the Freiburg-Parkinglot dataset, as estimated by episodic non-Markov localization (red trace), and compared to ground truth (dashed black trace). The LTFs, STFs and DFs classified by episodic non-Markov localization are drawn as orange, purple, and green points, respectively.

Floor	CGR	EnML	EnML Fraction (%)
GHC4	6.4km	88.1km	93.2
GHC5	3.0km	42.9km	93.5
All	320.9km	685.2km	68.1

Table 5.6: The contribution of EnML to the deployment of the CoBots on two challenging floors, GHC4 and GHC5. Over 93% of the distance traversed by the CoBots on GHC4 and GHC5 were with EnML, while only 68.1% of the distance traversed over the entire 1000km Challenge over all floors were with EnML.

Figure 5.8 shows some snapshots of EnML running on the CoBots while deployed on GHC4, at different times of the year. Note that the STFs are at different positions over the different deployments. Figure 5.9 shows two examples of EnML accounting for large unmapped features, a wooden panel of lockers, and the walls of the helix in the Gates-Hillman Center.

5.8 Summary

In this chapter, we introduced a representation of the world that classifies observations as arising from long-term features, short-term features, or dynamic features. We introduced a graphical model called the Varying Graphical Network (VGN) to represent correlations between observa-

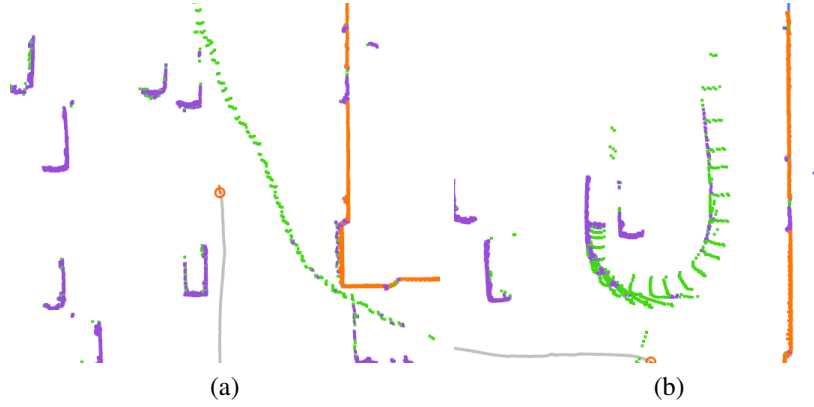


Figure 5.6: Non-Markov localization on run 12 of the Freiburg-Parkinglot dataset showing *a)* a pedestrian and *b)* a moving car in the parking lot amidst static parked cars. LTFs are plotted in orange, STFs in purple, and DFs in green. The robot’s location is shown by the orange marker and its trajectory as grey lines. Both images are $35m$ wide.

tions from different time steps due to the different classes of observations. We further introduced the Episodic non-Markov Localization (EnML) algorithm that explicitly reasons about the different classes of observations to maintain accurate location estimates of the robot even in the presence of observations of unexpected objects and the absence of observations of the static map. We explored the structure of EnML, and thus compared its computational complexity to SLAM and Markov Localization. Finally, we presented results of running EnML on several datasets and comparing its results to alternate approaches.

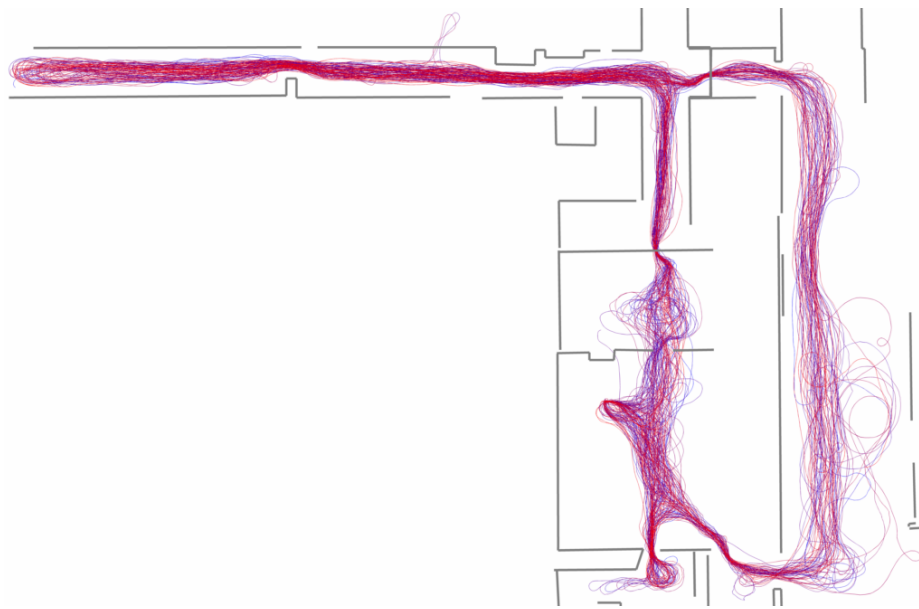


Figure 5.7: Combined traces of localization using episodic non-Markov localization on all runs of the Orrebro-Longterm dataset. The traces are color-coded by time, from blue (oldest) to red (newest).

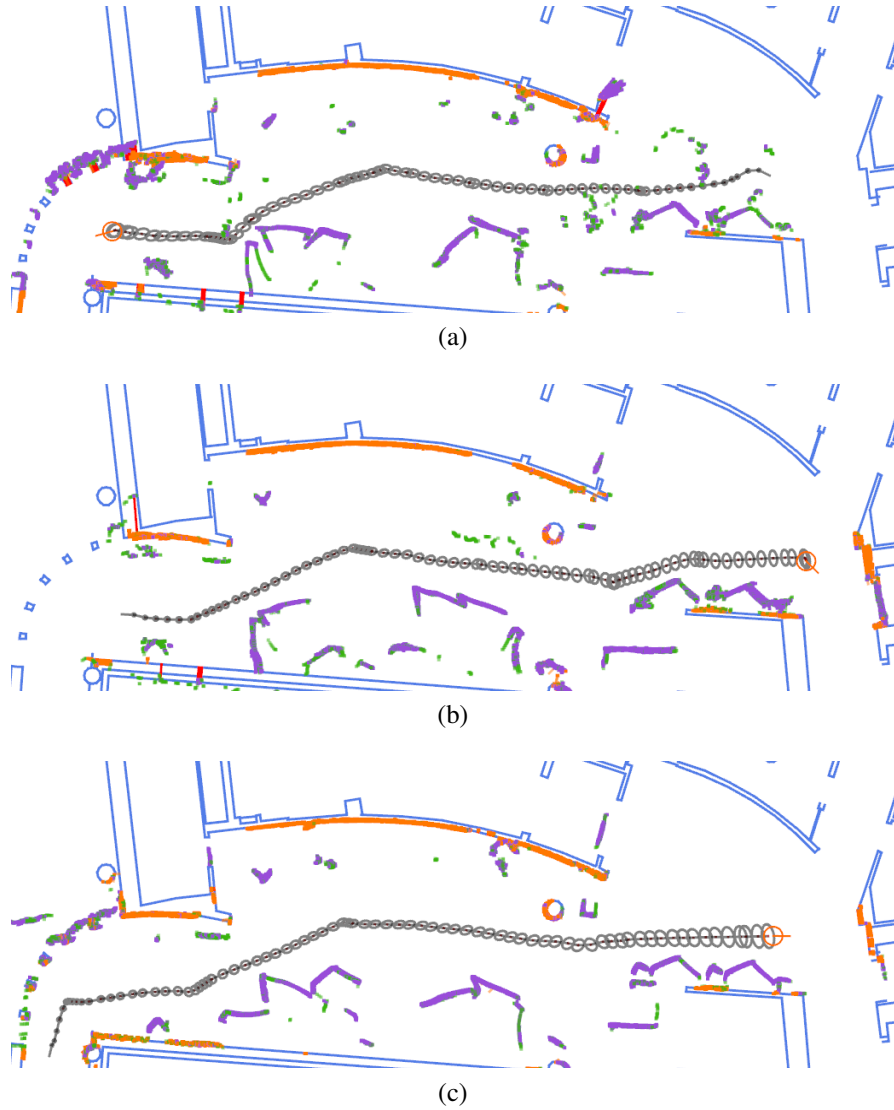


Figure 5.8: Snapshots of EnML running in the GHC4 atrium on different days. The trajectory of the robot over the episode is shown in grey, along with the covariance ellipses. LTF observations are shown as orange points, STF observations as purple points, and DF observations as green points. The long-term static map is shown as blue lines. Note the different placements of the STFs in the different runs, in particular the lounge chairs at the lower end. All images are 26m wide.

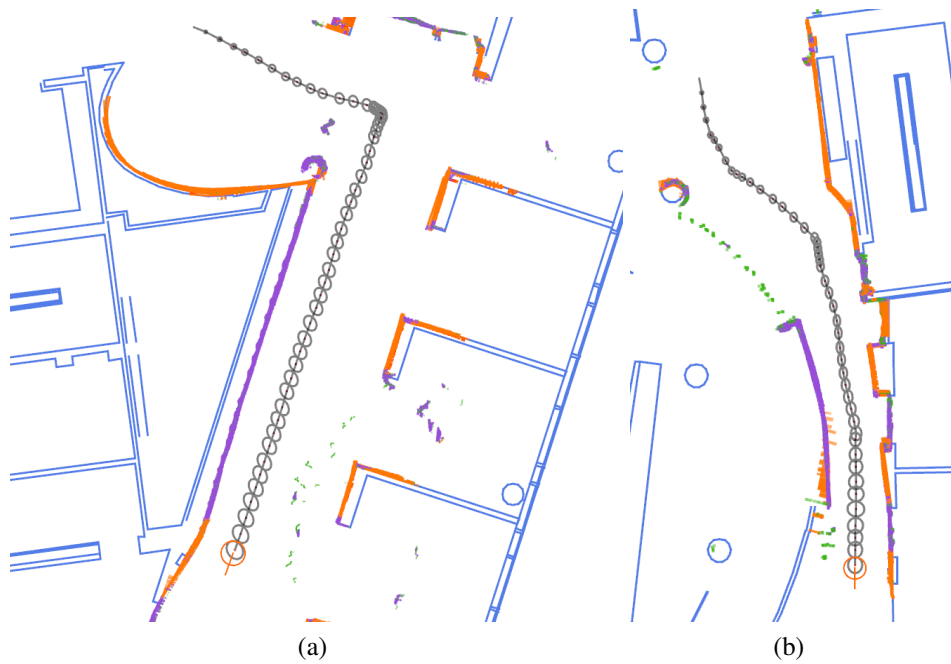


Figure 5.9: Snapshots of EnML running on GHC5, showing unmapped features that are (a) the walls of the helix ramp, and (b) a wooden panel of lockers. LTF observations are shown as orange points, STF observations as purple points, and DF observations as green points. The long-term static map is shown as blue lines.

Chapter 6

Updating Maps

As robots are deployed over extended periods of time, the sensor logs collected over the deployments may be processed to better understand the changes in the environment. In this chapter, we present two algorithms to update maps over time: Updating Plane-Polygon maps, and Model-Instance Object Mapping.

6.1 Updating Plane-Polygon Maps

With the availability of low-cost depth cameras, such as the Microsoft Kinect sensor, there has been renewed interest in building 3D models of indoor environments. Much of the current work towards the goal of 3D mapping (*e.g.*, [44, 62, 63]) has focused on generating detailed, dense 3D models of environments. However, typical indoor environments consist of large planar features that could instead be used to build simplified models of indoor environments at a fraction of the memory requirements of dense 3D models. Hence, we are interested in modeling such indoor environments by extracting the dominant planar features as a *Plane-Polygon Map* [12]. Such an approach is similar in spirit to that of 2D Vector Maps (Chapter 3.5.1) in that the dominant planar features in the 3D Plane-Polygon Map, just like the line segments in the Vector Maps, correspond to the permanent features in the environment. Just as the Vector Maps ignore fine details of movable objects, the Plane-Polygon Maps omit detailed 3D models of movable objects.

Mapping of planar features could be performed as a post-processing step by mesh simplification [24] of a detailed 3D model. Instead we introduce an incremental, on-line technique where for every observed depth image, only planar regions are extracted to be merged into the model of the environment. Our approach consists of the following steps:

1. For every observed depth image, compute polygons to best approximate the dominant planes in that image.

2. Find correspondences between the polygons in the current image with the polygons in the current plane-polygon map.
3. Iteratively merge corresponding polygons from successive images, thus growing the plane-polygon map.

Our approach uses the Fast Sampling Plane Filtering (FSPF) algorithm (Chapter 3.3.1), which extracts points belonging to local neighborhoods of planes from depth images, even in the presence of clutter. Given the FSPF generated neighborhoods of local planar points, the algorithm first computes the plane parameters of each neighborhood by eigenvector decomposition of the scatter matrix, and then construct a convex hull over the points to define the polygon boundary. The scatter matrix for each polygon is stored in a decoupled manner which, as we shall show, allows us to merge polygons over successive observations and compute the least square fit over *all* observed points over time without having to explicitly maintain a list of them. We introduce an image render based ray-casting method for determining correspondences between the polygons observed in the latest depth image, to the polygons in the plane-polygon map. Polygons in the latest depth image that do not correspond to any existing polygons in the plane-polygon map are added as new polygons to the map.

6.1.1 Plane Polygon Construction and Merging

Given an observed raw depth image I , the FSPF algorithm returns a plane filtered point cloud P of n 3D points that belong to local planes and a list R of the associated plane normals.

Polygon Construction

From the plane filtered point cloud P , the Plane-Polygon Mapping algorithm merges local neighborhoods of points to form local convex polygons. A convex polygon is denoted by the tuple $c = \{S_L, n_S, \psi, \nu, b_1, b_2, B\}$ where S_L is the set of 3D points used to construct the convex polygon, n_S the number of points in S_L , ψ the centroid of the polygon, ν the normal to the polygon plane, b_1 and b_2 the 2D basis vectors on the plane of the polygon and B the set of 3D points that define the convex boundary of the polygon. Given a neighborhood of plane filtered 3D points S_L , the Plane-Polygon Mapping algorithm constructs a convex polygon c as follows:

1. The polygon centroid ψ as $\psi = \frac{1}{n_S} \sum_{p_i \in S_L} p_i$.
2. The scatter matrix χ of the points in S_L as $\chi = \sum_{p_i \in S_L} (p_i - \psi)(p_i - \psi)^T$.
3. The normal ν as then the eigenvector of χ corresponding to its smallest eigenvalue.

4. The orthogonal basis vectors that span the plane, b_1 and b_2 as the remaining two eigenvectors of χ .
5. The boundary set B found using Graham scan [39] over the points of S_L projected onto the plane.

We assume the pose of the depth camera sensor has already been registered with respect to a global reference frame. Such registration could be done (for example) using one of the variants of iterative closest point (ICP) based approaches [76]. After pose registration, in order to construct a meaningful geometric representation of a scene, it is necessary to merge the observed polygons from multiple scenes to form a single, “global scene”, which the the Plane-Polygon Mapping algorithm does in two steps:

1. Correspondence Matching: Polygons in the latest frame are matched to polygons in the global scene.
2. Polygon Merging: Matched polygons are merged to update the plane parameters and the convex hull of the polygons in the scene.

Correspondence Matching

Given a set $M_{\text{polygon}} = \{p_i^M\}$ of polygons that represent the plane-polygon map constructed so far and the set $O = \{p_j^O\}$ of observed polygons in the latest depth image, the correspondence matching problem is to find out which polygons p_j^O in the current depth image overlap with which polygons p_i^M in the map. In our approach, we solve this problem using ray casting:

1. A unique color is assigned to each polygon p_i^M in the map M_{polygon} .
2. A virtual camera with the same optical parameters (pixel size, field of view) as the actual depth camera is set up at the location found by pose registration.
3. The color-coded polygons are then rendered to an image as viewed by the virtual camera.
4. The pixel location of each plane filtered point in the depth image is then used to look up the color of the corresponding pixel in the rendered image, thus indicating the map polygon corresponding to that point.

Figure 6.1 shows the steps of the correspondence matching procedure with plane-polygon maps.

The image rendering step, when performed using OpenGL and accelerated by an nVidia GeForce GTX 560 GPU, runs at an average of 3800 frames per second. After estimating observation to map polygon correspondences, these correspondences are further checked to ensure that matched polygons have an offset distance of less than ϵ_{dist} , and that the angle difference between their normals is at most ϵ_{ang} . ϵ_{dist} and ϵ_{ang} are configurable parameters, and in our experiments

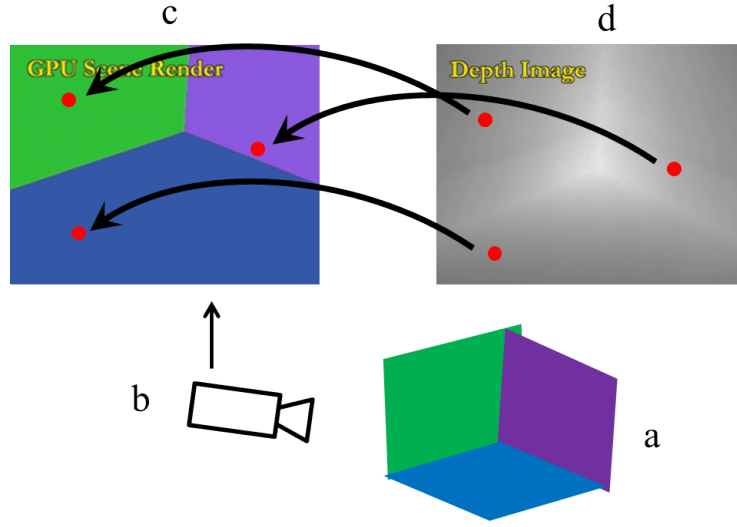


Figure 6.1: Correspondence matching: Polygons in the plane-polygon map are uniquely color-coded (a), and from the perspective of a virtual camera (b) rendered in an image (c). Correspondences of pixels in the observed depth image (d) are then determined by inspecting the color of the same pixel in the rendered image.

are set to 0.01m and 1.0° respectively. Observed polygons p_j^O that do not correspond to any map polygon are assumed to be new polygons, and added to the map M_{polygon} without merging with any existing polygon.

Polygon Merging

When merging the latest depth image observation with the map built so far, the polygons extracted from the latest depth image need to be merged with the corresponding polygons in the map built so far. The problem of polygon merging is thus: given two convex polygons c^1, c^2 , we wish to construct a merged polygon c^m using the 3D points from both planes c^1 and c^2 .

A naïve approach to merging polygons $c^1 = \{S_L^1, n_S^1, \psi^1, \nu^1, b_1^1, b_2^1, B^1\}$ and $c^2 = \{S_L^2, n_S^2, \psi^2, \nu^2, b_2^2, b_2^2, B^2\}$ to form merged polygon $c^m = \{S_L^m, n_S^m, \psi^m, \nu^m, b_m^m, b_2^m, B^m\}$ is to combine the 3D points as $S_L^m = S_L^1 \cup S_L^2$ and then compute all other polygon parameters from S_L^m . In practice, this is infeasible since it requires a complete history of all 3D points ever observed to be maintained and the amount of available memory is finite.

It is, however, possible to perform polygon merging without maintaining a complete history of all observed 3D points by in a manner analogous to the parallel computation of covariance matrices [19]. Scatter matrices from two different polygons cannot be directly combined due to the coupling in χ between the points p_i and the centroid. However, χ^1 may be decoupled as $\chi^1 = \chi_1^1 - n_S^1 \chi_2^1$ where χ_1^1 depends on the 3D points in S_L^1 , and χ_2^1 on the centroid ψ^1 . χ_1^1 and

χ_2^1 are then given by $\chi_1^1 = \sum p_i^1 p_i^{1T}$ and $\chi_2^1 = \psi^1 \psi^{1T}$. Given the centroids ψ^1 and ψ^2 of the two polygons, the centroid ψ^m of the merged polygon is computed as their weighted mean. Thus, the matrix χ_2^m of the merged polygon can be computed from ψ^m and the scatter matrix of the combined polygon is given by $\chi^m = \chi_1^1 + \chi_1^2 + n_S^m \chi_2^m$.

Therefore, the matrix χ^1 along with centroid ψ and the number of points n is sufficient for merging polygons based on the complete history of observed 3D points, and the individual 3D points no longer need to be maintained for each polygon.

One implementation detail remains: to ensure numerical stability over time, the algorithm performs the eigenvector decomposition on the normalized matrix $\frac{1}{n_S} \chi$, and maintains normalized matrices $\frac{1}{n_S} \chi_1$ instead of χ_1 .

The convex boundary point set B^m of the merged polygon is computed by running Graham scan on the union of the points from the boundary point sets B^1 and B^2 of the constituent polygons.

The complete set of steps of our Plane-Polygon Mapping algorithm are thus:

1. Compute centroid ψ^m
2. Compute scatter matrix χ^m
3. Compute normal ν^m and basis vectors b_1^m, b_2^m by eigenvector decomposition of χ^m
4. Compute boundary point set B^m using Graham scan on plane projected points from $B^1 \cup B^2$

The merged polygons in the map are thus updated using all observed points so far that correspond to them.

6.1.2 Experimental Results

We perform two sets of experiments to demonstrate the computational efficiency and effectiveness of FSPF and the polygon merging algorithms. First, we collect data for a controlled scene with and without clutter, with ground truth. This ground truth data evaluates the accuracy of the merged polygons, both with respect to the observed depth image, as well as with respect to the measured ground truth. We then test the FSPF and polygon merging algorithms on a set of real indoor scenes.

Ground Truth Comparison

We construct a scene with planar features (Figure 6.2a) including planes at various angles and different levels of occlusion. We then record depth image data from the scene and built a map of the scene using our algorithm. The errors in raw observation are evaluated by comparing

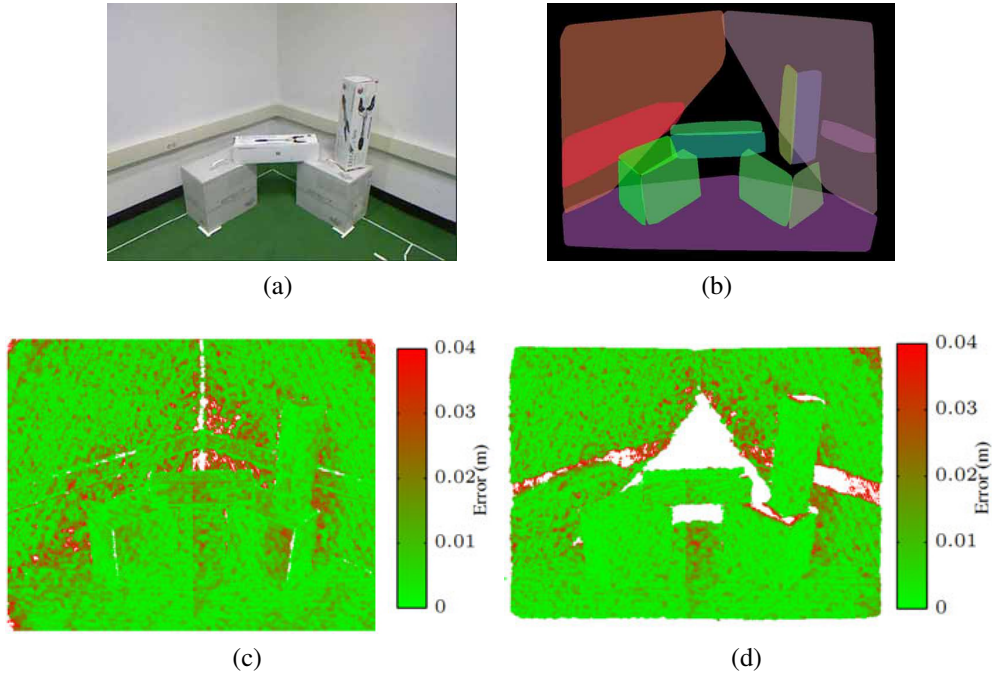


Figure 6.2: Ground truth in a clutter-free scene: (a) RGB Image of the scene, (b) The map built by the polygon extraction and merging algorithm, (c) Errors between points in depth image compared to ground truth, and (d) compared to built map. Errors are indicated in color, ranging from green (0m) to red (0.04m).

the reconstructed 3D points with the ground truth values. We refer to this error comparison as the “point to ground truth” (PG) error. Figure 6.2c shows the point to ground truth errors in observation for different parts of the image. Figure 6.2b shows the map of polygons reconstructed from the scene. Except for the top surfaces of the two boxes on the right, all the dominant planes in the image are extracted by the algorithm. Figure 6.2d shows the errors between the 3D points reconstructed from the depth image and the built map. We refer to this error comparison as the “point to map” (PM) error.

To evaluate the accuracy of the polygons in the built map, points on the polygons are uniformly sampled at intervals of 0.5cm , and these 3D points are compared to the ground truth. We refer to this error comparison as the “map to ground truth” (MG) error. Figure 6.3 shows a cumulative histogram of the three error comparisons.

We next evaluate the effect on the error in polygon map building due to the presence of clutter in the scene. Figure 6.4a shows a controlled scene with added objects of clutter. Figure 6.4c shows that the point to ground truth error is noticeably large in the regions of the scene corresponding to the objects of clutter. Figure 6.4d shows that the built map (Figure 6.4b) has

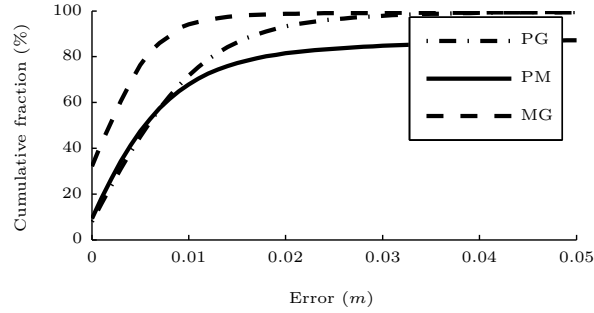


Figure 6.3: Cumulative histogram of errors in mapping the clutter-free scene: (PG) depth image vs. ground truth, (PM) depth image vs. built map and (MG) built map vs. ground truth.

lower errors than the point to ground truth case, but parts of the scene are unmapped. In spite of the local occlusions due to clutter on the boxes, the surfaces of the boxes are still reconstructed by the FSPF and polygon merging algorithms, and as Figure 6.5 shows, the polygons that are constructed in the scene are still observed to have a maximum error of $2cm$.

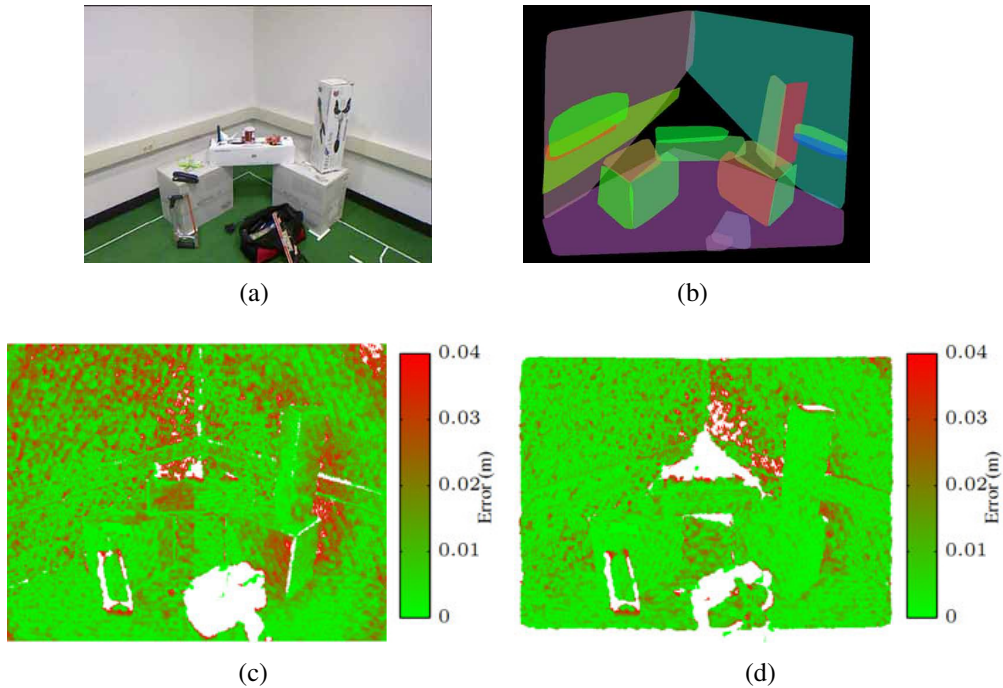


Figure 6.4: Ground truth in a scene with clutter: (a) RGB Image of the scene , (b) The map built by the polygon extraction and merging algorithm, (c) Errors between points in depth image compared to ground truth, and (d) compared to built map.

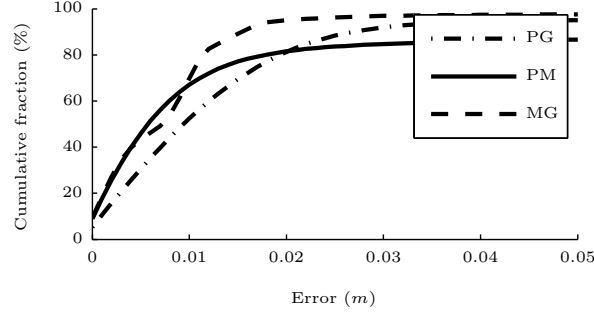


Figure 6.5: Cumulative histogram of errors in mapping the scene with clutter: (PG) depth image vs. ground truth, (PM) depth image vs. built map, and (MG) built map vs. ground truth

Real World Scene Tests

We test the FSPF and Plane-Polygon Mapping algorithms on data collected from 5 scenes that we select as representative of our environment. At each scene, the Kinect sensor is kept static, so no pose update is necessary. For each scene, the polygons from 15 consecutive frames are merged to generate a set of polygons that represent the entire scene. Figure 6.6 shows the scenes and the processed output from each scene, while Table 6.1 summarizes the output and processing times for the algorithms for each scene. The sampling efficiency for each scene is computed as the ratio of the mean number of plane filtered points per frame to the mean sampled locations per frame. The FSPF run time is per frame, and the polygon merge processing time is for all the polygons per frame.

Scene 1 has three boxes in the corner of a room. The merged scene polygons correctly include all the planes in the scene except two (the tops of the boxes on the left and center). Scene 2 is set in a corridor, and lacks clutter. The merged scene polygons correctly include all the polygons in the scene near the Kinect. Scene 3 is set in an open area next to a staircase. The merged scene polygons include the ground polygon, the side of the staircase, and the overhanging staircase. Scenes 4 and 5 show work areas with chairs and tables. Despite the clutter, all table surfaces are correctly identified among the merged scene polygons. Scene 5 had a toy chair on top of the table, and the merged scene includes its corresponding polygons as well.

Scene	Merged Polygons	Plane Filtered Points Per Frame	Sampled Points Per Frame	Sampling Efficiency (%)	Polygons Per Frame	FSPF Run Time (ms)	Polygon Merge Time (ms)
1	14	2002	5434	36.84	71	1.86	0.37
2	5	2006	8620	23.28	81	1.66	0.71
3	3	2001	11009	18.18	82	2.06	0.38
4	14	2004	8260	24.27	70	1.39	1.12
5	7	2001	6861	19.17	67	1.66	0.95

Table 6.1: Results of planar polygon mapping from test scenes

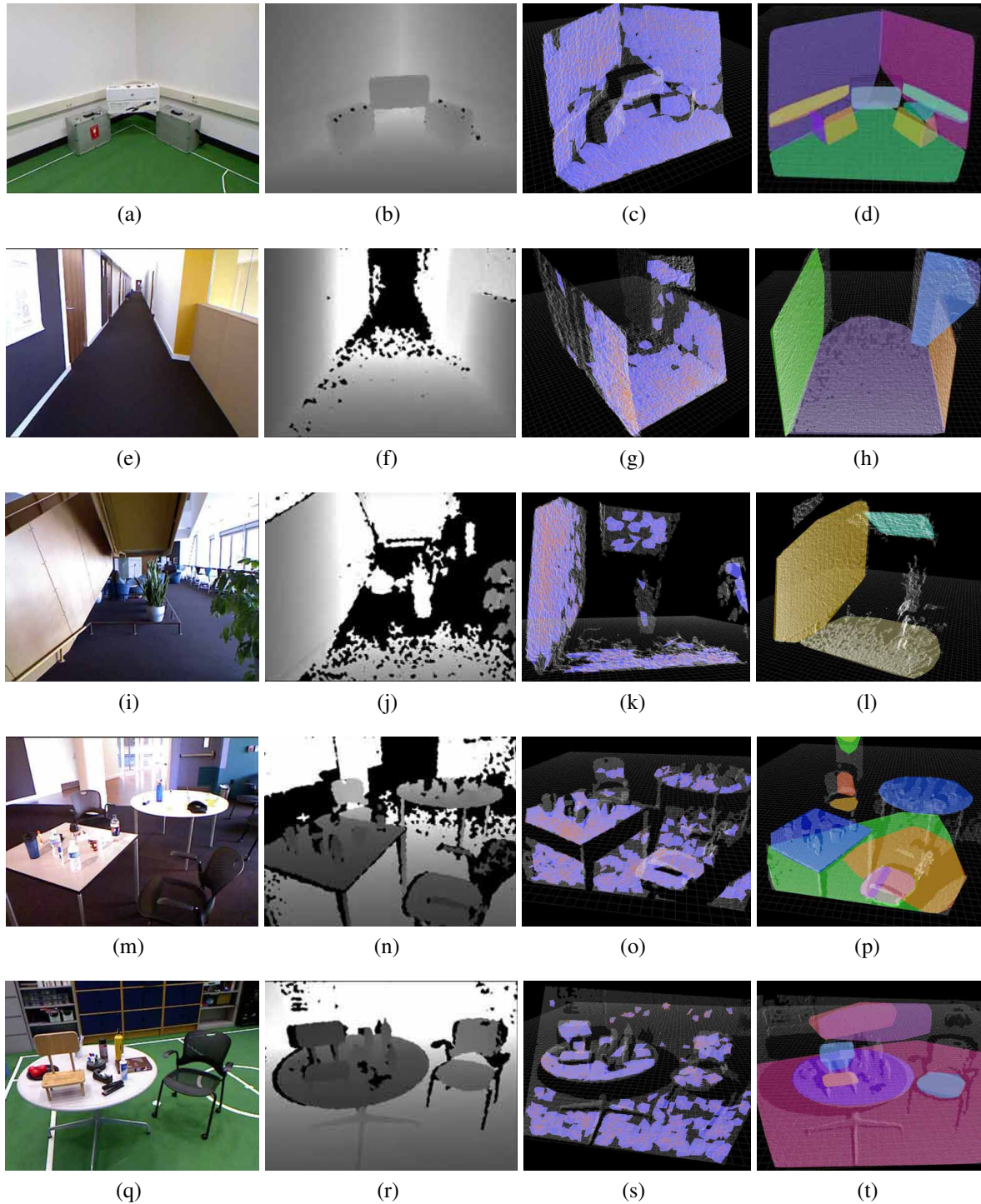


Figure 6.6: Test scenes 1-5 (top to bottom). First column shows the RGB image of the scene, second column the depth image, third column the plane filtered points (orange points) and polygons (lilac) for a single depth image frame, fourth column the polygons generated by merging 15 consecutive processed depth image frames. Columns 2 and 3 include overlays of the raw 3D point cloud (translucent white) for reference.

6.2 Model-Instance Object Mapping

Robots deployed in human environments frequently encounter movable objects like tables and chairs. These movable objects pose a challenge for the long-term deployment of robots since they obstruct features of the map, and are difficult to map persistently. Areas where the majority of the observations of the robot are of unmapped objects are especially challenging, making it harder for a robot to accurately estimate its location globally.

We propose an approach to modelling a changing environment by separating the models of the movable objects from their distribution of poses in the environment. This proposed approach is borne of the realization that movable objects in human environments are often different instances of the same type of object. Furthermore, even though movable objects do move around, they still tend to be situated in roughly the same locations. For example, a work chair will most likely be observed in front of its accompanying work desk, even if the exact location of the chair might change over time. Our proposed approach, *Model-Instance Object Mapping* [15], models the objects independently of their instances in the environment. Each Model thus represents a unique *type* of object, while Instances represent the different *copies* of it that are observed. Observed short-term features, as classified by Episodic non-Markov Localization (EnML) (Chapter 5), are first clustered into separate point clouds for each object instance. An occupancy grid is built for each object instance, and are then compared in pairs to form a directed object similarity graph. Common object models are then detected by looking for strongly connected components in the graph.

EnML estimates the robot's pose, as well as the pose of the unmapped STFs over each episode. EnML does not maintain a persistent history or database of the STFs that have been observed in the past. However, in a real human environment, the set of movable objects observable by a robot in a given area is finite, and the rest of this section is devoted to building models of these STFs, and estimating their pose distributions in the world. EnML provides (aside from the poses of the robot) a set S_{STF} of all the points corresponding to STFs observed in the world during the robot's deployment, and the corresponding poses X_{STF} of the robot from where the points were observed. Figure 6.7 shows an example set S_{STF} of STFs as detected by EnML.

6.2.1 Finding Object Instances

The first step to building models of the objects is to cluster the observed points. The goal of this step is, given S_{STF} (the set of points in global coordinates corresponding to STFs) and X_{STF} (the set of poses of the robot from which points in S_{STF} were observed), to form clusters $\tilde{S} = \{\tilde{s}_i\}_{i=1:n}$ where each cluster \tilde{s}_i is a non-overlapping subset of S_{STF} , and every point in \tilde{s}_i

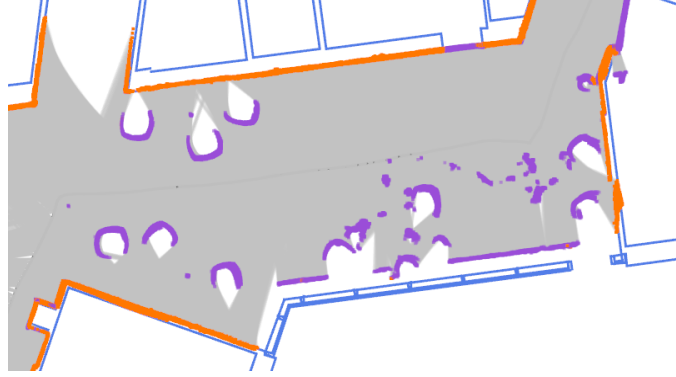


Figure 6.7: Step 1 of Model-Instance Object Mapping: Observations S_{STF} classified as Short-Term Features (purple) by Episodic non-Markov Localization are extracted.

is within a distance of $\epsilon_{\text{cluster}}$ of at least one other point in \tilde{s}_i . Here, $\epsilon_{\text{cluster}}$ is a configurable parameter that determines how close two objects can be, in order to be considered to be part of the same object. Note that each cluster \tilde{s}_i may contain points observed from different poses of the robot, as long as they are spatially separated by at most $\epsilon_{\text{cluster}}$ from other points in the cluster. We use the point cloud clustering algorithm from [77] for this step, with a distance threshold of $\epsilon_{\text{cluster}} = 3\text{cm}$. Figure 6.8 shows the clusters that were extracted from the example set S_{STF} of Figure 6.7

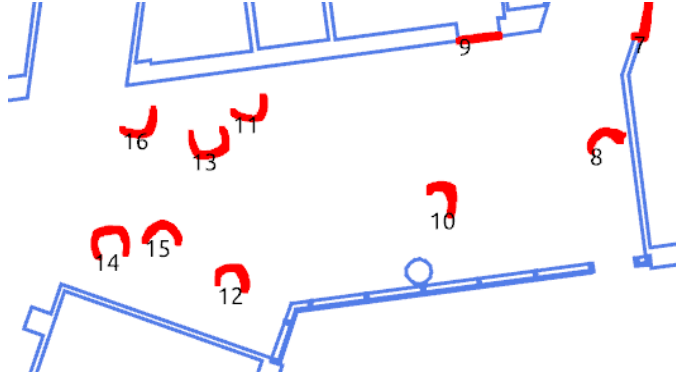


Figure 6.8: Step 2 of Model-Instance Object Mapping: Observations S_{STF} are clustered into set $\tilde{S} = \{\tilde{s}_i\}_{i=1:n}$. Each distinct cluster is denoted by its cluster index. Note that points in S_{STF} that were sparsely distributed and not within distance $\epsilon_{\text{cluster}}$ of other points have been discarded.

For each cluster \tilde{s}_i , we build an occupancy grid [33] map m_i from the observed points to model the shape of the object. Algorithm 7 lists the algorithm to build an occupancy grid map m given cluster c , the associated set of poses x , and the width w and height h of the cluster. The grid map m and occupancy counts n_m are first initialized to zero matrices. For every pixel location ζ , for every observation that was made at that location, the occupancy value $m(\zeta)$ and observa-

tion count $n(\zeta)$ are both incremented. For every pixel location ζ that is observed to be vacant by observing a point beyond it, the observation count is incremented without incrementing the occupancy value. Finally, the occupancy value at every pixel location is normalized by the observation count for that pixel. Figure 6.9 shows three example occupancy grid maps constructed from the clusters of Figure 6.8.

Algorithm 7 Build Object Model

```

1: procedure BUILDOBJECTMODEL( $c, x, w, h$ )
2:    $m \leftarrow w \times h$  zero matrix
3:    $n \leftarrow w \times h$  zero matrix
4:   for each pixel  $\zeta$  in  $m$  do
5:     for each point  $p$  in  $c$ , pose  $o$  in  $x$  do
6:       if  $p$  is observed at pixel  $\zeta$  then
7:          $m(\zeta) \leftarrow m(\zeta) + 1$ 
8:          $n(\zeta) \leftarrow n(\zeta) + 1$ 
9:       else if  $\zeta$  is between  $p$  and  $o$  then
10:         $n(\zeta) \leftarrow n(\zeta) + 1$ 
11:      end if
12:    end for
13:  end for
14:  for each pixel  $\zeta$  in  $m$  do
15:     $m(\zeta) \leftarrow m(\zeta)/n(\zeta)$ 
16:  end for
17:  return  $m$ 
18: end procedure

```

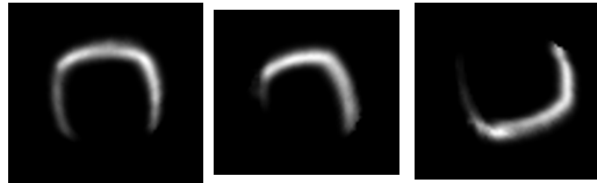


Figure 6.9: Step 3 of Model-Instance Object Mapping: Each object instance is used to build an occupancy grid model, and three such example models are shown here.

6.2.2 Building Object Models

In human environments, and in particular in office environments, movable objects frequently occur as multiple instances of the same model. For example, in a common study area, there may be many chairs, but there will likely be either a single, or a small number of *types* of

chairs. Hence, given the occupancy grid maps m_i for each object instance observed, there may be multiple instances of the same model of object, and the subject of the next section is on how to find these common object models.

As a robot encounters different instances of the same object model, it is likely to observe them at different locations and rotations. Furthermore, due to the presence of other objects, they may be partially occluded. Therefore, before comparing the models of different instances, we first align pairs of objects. For a given relative transform $T = \{\delta_x, \delta_y, \delta_\theta\}$ between the centroids of two object instances m_i and m_j , we define an alignment objective function F_{Align} ,

$$F_{\text{Align}}(m_i, m_j, T) = \sum_{\zeta \in m_i} m_i(\zeta) m_j(T \times \zeta) \quad (6.1)$$

computed over all pixel locations ζ in m_i . Given this alignment object function, the optimal alignment T_{ij}^* between object instances m_i and m_j is defined as

$$T_{ij}^* = \arg \max_T F_{\text{Align}}(m_i, m_j, T). \quad (6.2)$$

This optimal alignment is found by exhaustive search over all possible relative transforms within a search window $|\delta_x| < \Delta_t, |\delta_y| < \Delta_t, |\delta_\theta| < \Delta_\theta$.

Given an optimal alignment T_{ij}^* between object instances m_i and m_j , we define a similarity metric F_{Similar} to compare the object instances:

$$F_{\text{Similar}}(m_i, m_j, T_{ij}^*) = \frac{\sum_{\zeta \in m_i} I_{\epsilon_{\text{occ}}}(m_i(\zeta)) I_{\epsilon_{\text{occ}}}(m_j(T_{ij}^* \times \zeta))}{\sum_{\zeta \in m_i} I_{\epsilon_{\text{occ}}}(m_i(\zeta))}. \quad (6.3)$$

Here, $I_{\epsilon_{\text{occ}}}(\cdot)$ is the thresholded indicator function that returns 1 when the value passed to it is greater than the threshold ϵ_{occ} . Note that F_{Similar} is not a symmetric metric: $F_{\text{Similar}}(m_i, m_j, T_{ij}^*) \neq F_{\text{Similar}}(m_j, m_i, T_{ji}^*)$. This is because the similarity metric is computed over all pixel locations ζ in the first object instance m_i , and the two object instances need not necessarily be of the same size. Furthermore, the similarity metric returns values in the range $0 \leq F_{\text{Similar}} \leq 1$. Given two object instances m_i and m_j , and a similarity threshold Δ_{Similar} , m_i is classified as being similar to m_j if

$$F_{\text{Similar}}(m_i, m_j, T_{ij}^*) \geq \Delta_{\text{Similar}}. \quad (6.4)$$

Similarity comparisons between pairs of objects thus depend on two parameters, ϵ_{occ} and Δ_{Similar} . ϵ_{occ} serves as an outlier rejector, ignoring low occupancy values in the object instance models,

while Δ_{Similar} serves as a confidence threshold.

To determine the values of ϵ_{occ} and Δ_{Similar} to be used, we hand-labelled similarity comparisons between 25 object instances. Next, for several values of ϵ_{occ} , we varied Δ_{Similar} to evaluate the precision and recall of the similarity classification. Figure 6.10 shows the precision-recall curves for the different values of ϵ_{occ} . The values $\epsilon_{\text{occ}} = 0.05$ and $\Delta_{\text{Similar}} = 0.7$ yield a precision as well as recall of 90%, and we use these values for the subsequent sections.

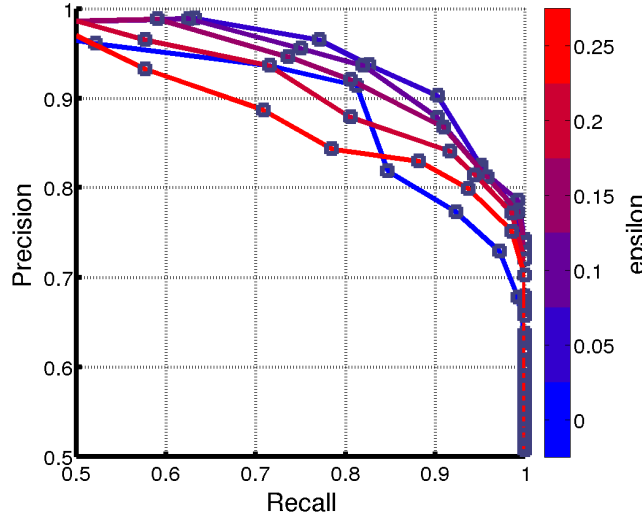


Figure 6.10: Precision-recall curves for object similarity classification between pairs drawn from 25 objects, compared to hand-labeled classification.

Figure 6.11 shows sample results of alignment and similarity comparison between four objects. Each object is classified as being similar to itself as expected, while different instances of the same model (instances 1 and 2, and instances 3 and 4) are correctly aligned and classified as being similar. Objects 1 and 2 were doors, and objects 3 and 4 were two instances of the same type of chair observed by the robot. Note that object instances 1 and 2, as shown by their occupancy grids, were observed in different poses, but still aligned correctly with respect to each other.

The similarity metric F_{Similar} is used to build a directed similarity graph $G_{\text{similar}} = \langle V_{\text{similar}}, E_{\text{similar}} \rangle$ where vertices $v_i \in V_{\text{similar}}$ denote the object instances the corresponding object instances m_i , and directed edges $e_k \in E_{\text{similar}}$ indicate similarities between object instances. The presence of an edge $e_k : v_i \rightarrow v_j$ denotes that object instance m_i , corresponding to vertex v_i , is similar to object v_j , as evaluated by Equation 6.4. Given this directed similarity graph G_{similar} , we find sets of object instances corresponding to the same object model by computing the strongly connected components of G_{similar} . Let $\Lambda = \{\lambda_i\}_{i=1:n_\Lambda}$ be the set of N_s strongly connected components

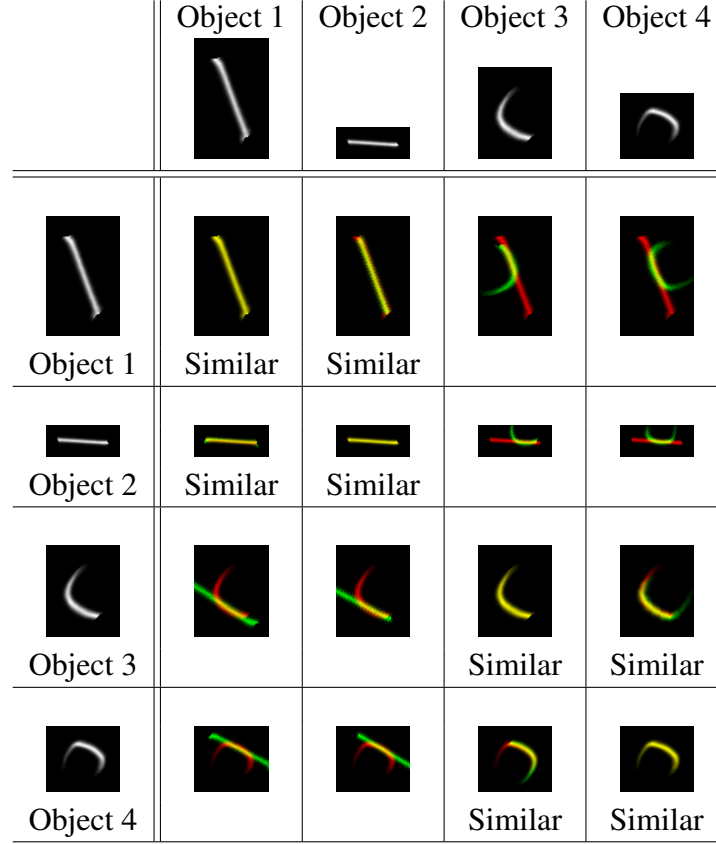


Figure 6.11: Alignment and similarity comparison grid between four object instances. The top row and left-most column show the occupancy grids of the object instances. In each cell, the instance corresponding to the row is drawn in red, and the optimal alignment T^* of the instance corresponding to the column is drawn in green. Similarity matches are labelled in each cell.

extracted from graph G_{similar} . Each strongly connected component s_i thus corresponds to a common model that is shared by the object instances represented by the vertices in s_i . Figure 6.12 shows an example similarity graph constructed by comparing the observed object instances of Figure 6.8.

6.2.3 Updating the Model-Instance Map across Deployments

So far we have focussed on how common models, detected as strongly connected components s_i , are extracted from the observations of STFs during a single deployment log of the robot. However, as a lifelong learning algorithm, Model-Instance Object Mapping needs to reason about the persistence of object models and their instances across *all* deployments of the robot. In this section we present how models s_i from the latest deployment of the robot are merged with older persistent models p_j from all past deployments of the robot. observed.

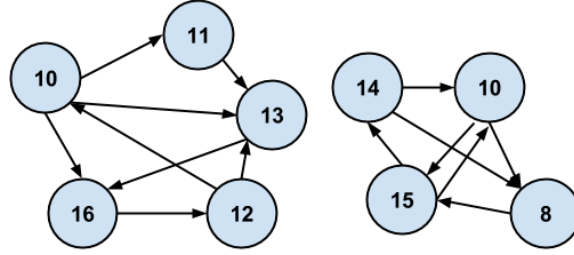


Figure 6.12: Step 4 of Model-Instance Object Mapping: Every object instance is compared to every other object instance, and a similarity graph is constructed by connecting nodes (representing object instances) by directed edges to indicate similarity between them. Note that in this example, there are two strongly connected components, and hence the Model-Instance Object Mapping algorithm will find two common object models.

Persistent models p_j , just like models s_i , include a common occupancy grid model of the object, and a list of poses of the observed instances of the object. For the first deployment of the robot, the list of persistent objects is empty, and thus all models s_i are saved as unique persistent objects. For every subsequent deployment, every model s_i is aligned (Equation 6.2) and compared to every persistent model p_j using the similarity metric F_{Similar} (Equation 6.3). For each object s_i , every persistent object p_j that satisfies the similarity test of Equation 6.4 bidirectionally is declared to be of the same model. All such sets of objects that are declared to be of the same model are then merged. Note that this merging step also attempts to merge persistent objects that were previously distinct, but which are determined to be bidirectionally similar to a newly observed object. This procedure for updating and merging persistent objects preserves the property that all instances of that object form a strongly connected component. Figure 6.13 shows the pose instances of an example persistent object, as extracted from the similarity graph of Figure 6.12, which was constructed by comparing the object instances of Figure 6.8.

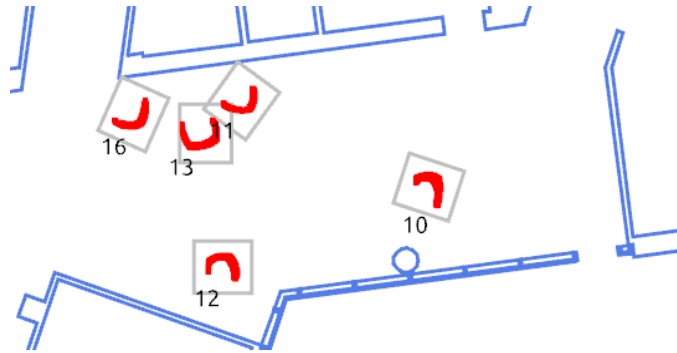


Figure 6.13: Step 5 of Model-Instance Object Mapping: Poses for every instance of one example persistent objects are visualized here by duplicating the model of the persistent object at the poses of the instances on the map.

6.2.4 Results

We have been deploying our robot, CoBot in an actual office environment since 2011, and have gathered extensive logs of the robot's deployments (Chapter 7). We processed a subset of the logs thus gathered between the January and April of 2014, in one of the floors in our building, through the Object-Instance Mapping algorithm. There were 133 such deployments in total, over the course of which the robot traversed a total distance of more than 55km. We present here the resultant Model-Instance map thus learned.

There were a total of 2526 objects observed, out of which there were 221 unique object models discovered. Out of the 221 unique object models, 50 had 2 or more observed instances. Table 6.2 lists the distribution of the frequency of the unique object models. The most commonly observed object model is that of doors, and the algorithm detected 1440 instances of them. Figure 6.14 shows the poses of the doors that were detected by Model-Instance Object Mapping without any supervision. The algorithm correctly detected all the doors in the environment, and also catalogued all the observed poses of the doors observed. From the figure, it is easy to discern the opening edges of the doors. Figure 6.15 shows some of the most commonly observed object models and their instances.

Num Instances	1	2-5	6-10	11-20	21-40	41-80	81-160	161-1440
Num Objects	171	25	11	2	5	4	1	2

Table 6.2: Frequency counts of the unique object models discovered. The first row lists the range of the number of instances, and the second row lists the number of objects that were discovered within that range of instances.

6.3 Summary

In this chapter we presented algorithms for updating maps over time, and in particular an algorithm to update planar polygon maps, and the Model-Instance Object Mapping algorithm to model the shapes of commonly observed movable objects and their observed instances. We presented results of running the planar polygon map update in some example scenes, and Model-Instance Object Mapping over logs gathered over the duration of a month.



Figure 6.14: The most commonly observed object, doors, as discovered by Model-Instance Object Mapping. Our algorithm correctly detected all the doors in the environment without any supervision.

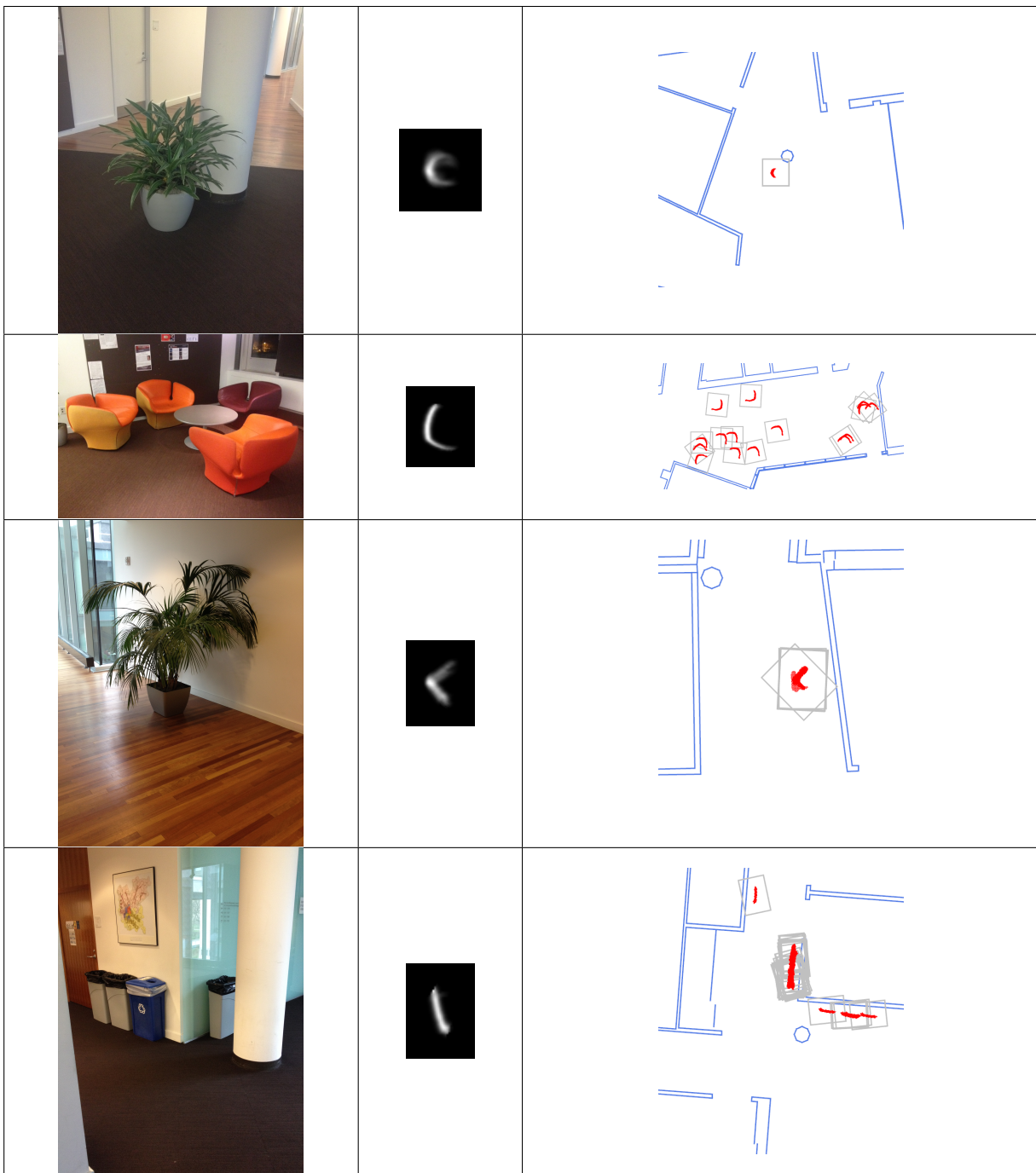


Figure 6.15: Some examples of objects discovered by Model-Instance Object Mapping: (from top to bottom) a round planter, a commonly found chair, a square planter, and trashcans in the kitchen area. The first column shows a photograph of the object, the second column shows the occupancy grid model built, and the third column shows the instances that were observed for each object model, across all the deployments. Note that instances of these objects were detected at multiple locations, but only one location is shown for each object here.

Chapter 7

The 1000km Challenge

This thesis introduced a number of new algorithms and representations for mobile robot localization, belief representation, map representation, as well as observation models for different sensors. To demonstrate the robustness and accuracy of localization using these contributions, a few years ago, we proposed the *1000km Challenge*.

The 1000km Challenge: Demonstrate, on a team of deployed autonomous mobile robots, in multiple real-world human environments, the robustness and accuracy in localization over long-term deployments spanning multiple years and covering a total distance of more than 1000km¹ over all such deployments.

The challenge is formulated to have the following properties:

1. **Robustness and accuracy:** To concretely evaluate the robustness and accuracy of the localization algorithms.
2. **Multiple real-world human environments:** To be performed in real environments, so as to expose the localization algorithms to a potential wide variation in environments.
3. **Long-term deployments:** To be performed over deployments spanning multiple years, thus exposing the localization algorithms to the variations that would normally occur in real human environments over such a time-span.
4. **Team of autonomous robots:** To be performed collectively by multiple robots, each with its own unique sensing abilities.

The 1000km Challenge was started on May 17, 2011 and completed on November 18, 2014. In this chapter, we present the scope of the deployments of the 1000km Challenge, the data collected, and corresponding quantitative and qualitative results.

¹1000km is 621.37 miles

7.1 The Scope of Deployments

The robots used in the 1000km Challenge are the CoBots (Chapter 1.2). Since September 2011, four CoBots have been autonomously performing various tasks for users on multiple floors of our buildings, including escorting visitors, transporting objects, and engaging in semi-autonomous telepresence. The CoBots vary in their sensing capabilities. CoBots 1, 2 and 3 have a short-range laser rangefinder, the Hokuyo URG-04lx (Chapter 3.2). All the CoBots have one forward-facing depth camera, the Microsoft Kinect. The forward-facing depth camera is also used for human detection [22] in addition to sensing for localization and obstacle avoidance. CoBots 2 and 4, in addition to the forward-facing depth camera have a second depth camera each. On CoBot 2 the second depth camera is placed low near the base, and facing upwards, allowing the robot to perceive tall chairs and tables. On CoBot 4 the second depth camera is mounted high on the robot and facing downwards, allowing the robot to detect small obstacles in its path.

The robots have been deployed on several buildings, including the Gates-Hillman Centers (GHC) and Newell-Simon Hall (NSH) at Carnegie Mellon University, and The Center for Urban Science and Progress at New York University (NYU). There are 12 floors in total across all the buildings that the CoBots have been deployed on, including floors 3-9 in GHC, 1-4 in NSH, and floor 19 in NYU.

Occupants of the buildings may schedule tasks for the CoBots from our online scheduling interface [27], shown in Figure 7.1. Additionally, the robots may also be interrupted by bystanders, who can then schedule tasks on the robot directly using the on-board scheduling interface [84], shown in Figure 7.3. The task scheduler assigns and distributes the user-requested tasks among the deployed robots, taking into account transfers between the robots [25]. In addition to user-requested tasks, when the robots do not have any pending tasks, they perform self-assigned tasks commanding the robots to visit randomly chosen locations along the navigable paths in the buildings.

The CoBots navigate through the environment unchaperoned and largely unmonitored. The robots exhibit symbiotic autonomy, as they autonomously seek human assistance to perform tasks [74, 75] that involve manipulation, since the CoBots do not have manipulators. Execution monitoring scripts on the robot track the task execution progress and send email to the administrators in the rare cases when it needs assistance. Figure 7.4 shows one such email, where the robot had waited for more than 5 minutes for human help at the elevators. When required, CoBot developers can use the web-based remote monitoring and telepresence interface of CoBot [26], shown in Figure 7.2, to inspect the state of the robot and remotely send it commands.

Over the course of their regular deployments, the CoBots traverse the public areas in the

Figure 7.1: The CoBot task scheduling website. Users may select from a drop-down list of tasks, shown in the inset.

building, encountering varied scenarios (Figure 7.5), and yet accurately reach task locations (Figure 7.6). Using localization alone, the robot is repeatably and accurately able to slow down before traversing bumps on the floor (Figure 7.5c), stop at the right location outside offices (Figure 7.6a-b) and the kitchen (Figure 7.6c), as well as autonomously get on and off the elevators in the building (Figure 7.6d). There are areas in the building where the only observable long-term features are beyond the range of the robot’s sensors, requiring the robot to navigate these areas using only the short-term features as reasoned by EnML (Chapter 5). For safe navigation, the CoBots have to detect tall chairs and tables undetectable to the laser rangefinder sensors but visible to the depth cameras. Despite these challenges, the CoBots remain autonomous in our environment, and rarely require human intervention.

7.2 Data Collected

During every deployment, the CoBots log the following data streams:

1. The drive commands sent to the motors at 20Hz
2. Software exceptions (if any) from all nodes running on the robot
3. Operator interventions (if any) using the on-board touch-screen as well as the remote telepresence controls on the website
4. Humans detected by the depth cameras at 10Hz
5. Joystick commands received
6. Localization estimates at 20Hz

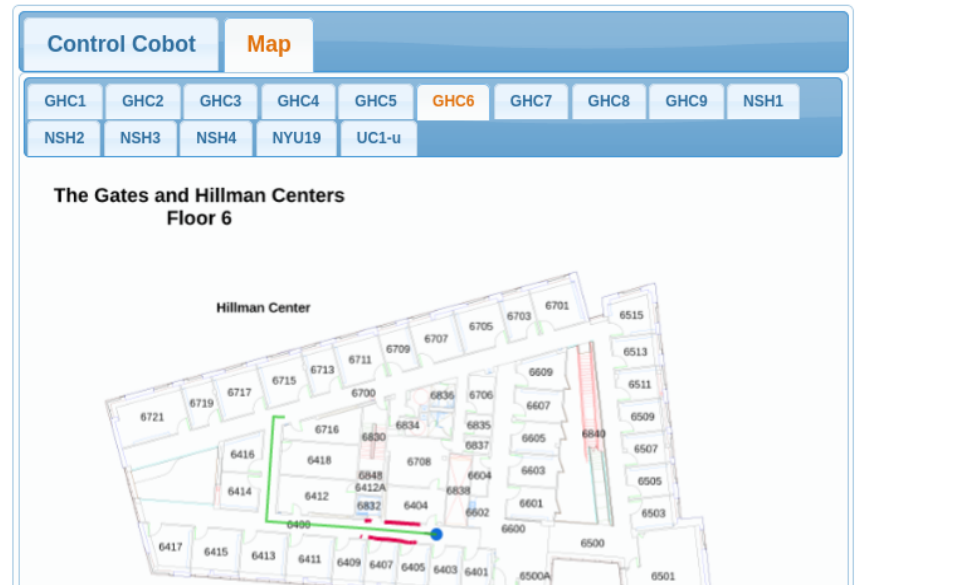
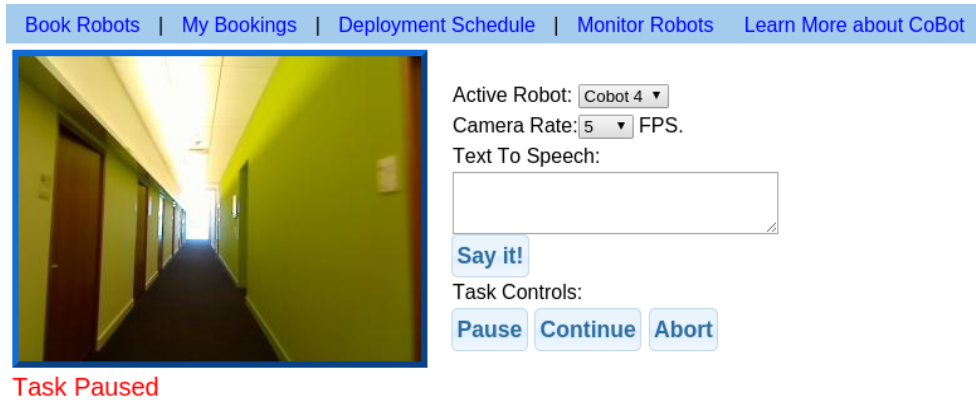


Figure 7.2: The CoBot remote telepresence website, which administrators can use to remotely monitor and control the robots.

7. Odometry feedback from the wheel encoders at 20Hz
8. StarGazer observations when received
9. Navigation status, including planned path and current command at 20Hz
10. Laser rangefinder scans at 10Hz
11. Obstacle scans computed using the depth cameras at 30Hz
12. Raw depth images from the depth cameras, at a reduced frame rate of 0.1Hz

All data streams are logged at the rates that they are generated, except for the raw depth images, which are logged only at a reduced frame rate of 0.1Hz to keep the log file sizes manageable.

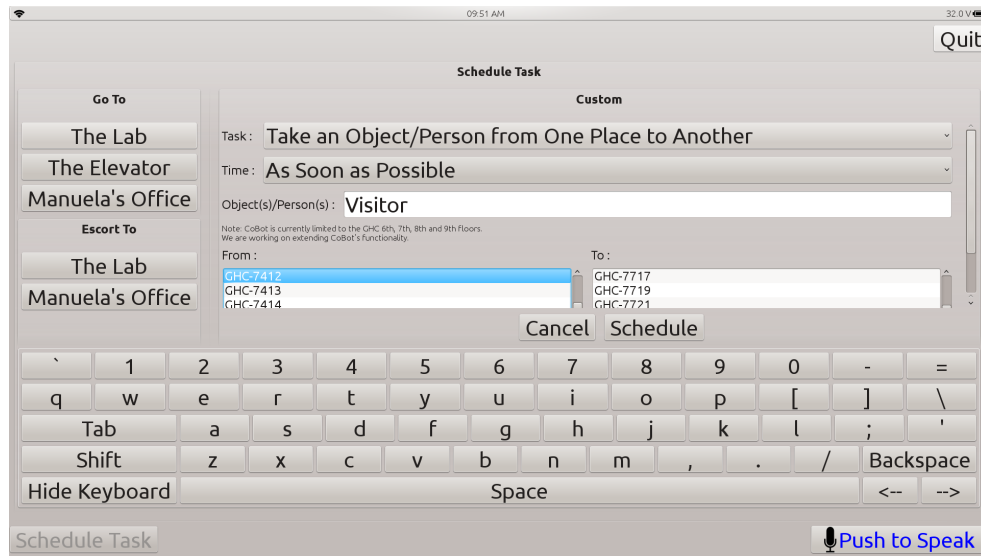


Figure 7.3: The on-board CoBot task scheduling interface. Bystanders may use this interface on the robot's touch-screen to schedule tasks locally on the robots.

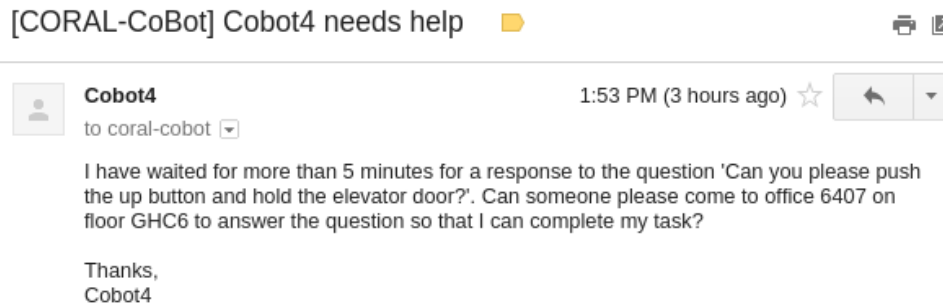


Figure 7.4: An example email from CoBot4, asking for assistance.

Figure 7.7 shows the progress of the 1000km Challenge as a graph of the cumulative distance traversed by the robots over time. CoBot2, as the most sensor-rich robot, as well as the primary test and development platform, initially accounted for a large share of the total autonomous distance traversed, but CoBot3 and CoBot4 have been deployed more extensively, and over more extended deployments more recently. Figure 7.8 shows the combined traces of all the locations visited by all the CoBots over all the maps. The 1000km Challenge has resulted in the collection of over 168GB of compressed data logs. Table 7.1 lists the cumulative contents of all the logs, and Table 7.2 and Table 7.2 list the contributions to the 1000km Challenge per robot. Note that the different robots have uneven deployments due to the variations in their availability: the robots were completed at different times, and were otherwise occupied for other research. Figure 7.8 shows the combined traces of the paths traversed by the CoBots on the different floors. The

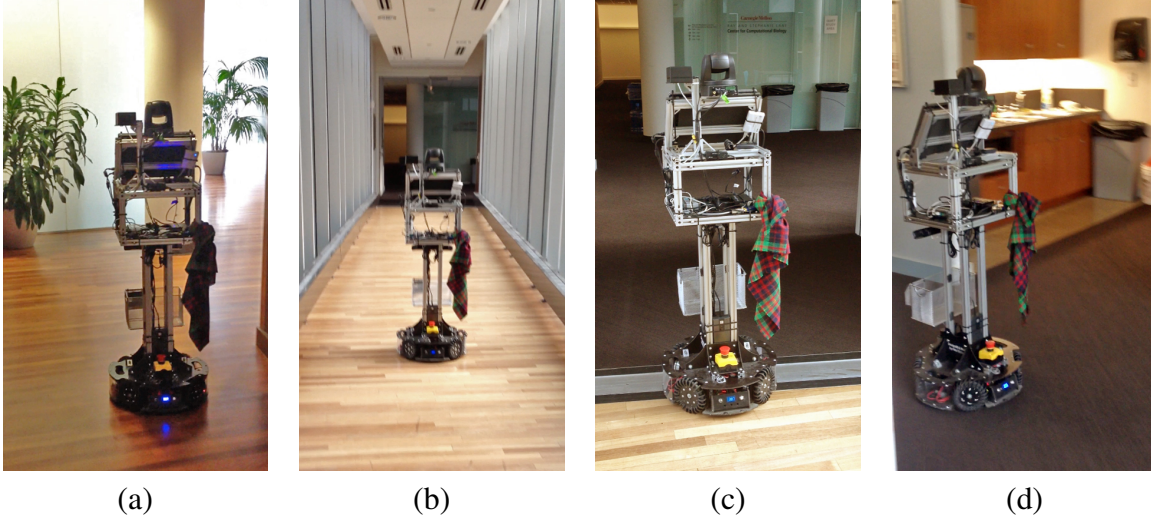


Figure 7.5: Localization and navigation in varied scenarios, including (a) in an open area with glass walls, (b) while crossing a bridge, (c) slowing down to cross a bump, and (d) negotiating a sharp turn.

CoBots visited different floors with different frequencies, as listed in Table 7.3. Complete sensor logs collected by the CoBots over the 1000km Challenge are available online at <http://www.cs.cmu.edu/~coral/cobot/data.html>

Property	Value
Duration	1279.5 h
Distance traversed	1006.1 km
Deployments	3199
Laser rangefinder scans	42,815,389
Depth camera obstacle scans	54,932,523

Table 7.1: Cumulative totals from the data logs collected from all the CoBots over the 1000km Challenge.

7.3 Quantitative Results

We evaluated the error in the localization estimates over the 1000km Challenge by two methods: comparison to scan matching, and comparison to sparse ground truth.

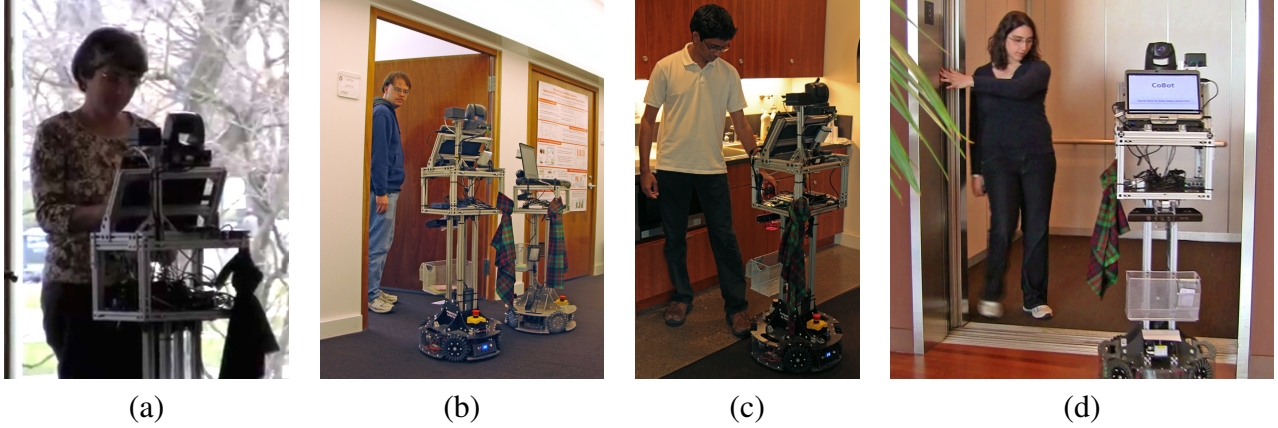


Figure 7.6: Reliable robot positioning at task locations, including (a) outside an office door, (b) at a transfer location, (c) at the kitchen, and (d) while taking the elevator.

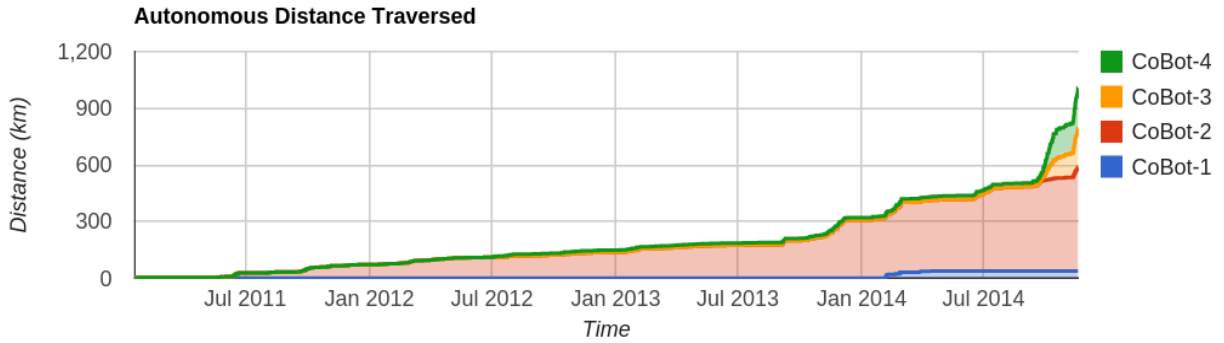


Figure 7.7: The progress of the 1000km Challenge as a stacked cumulative graph. The contributions of the CoBots 1, 2, 3 and 4 are represented by the areas shaded in blue, red, yellow and green, respectively.

7.3.1 Accuracy Compared to Scan Matching

There are locations in the environment where there exist abundant full-rank long-term map features such that the robot's location may be uniquely determined by scan matching [67]. We call these locations "Landmark Checkpoints". Corridor intersections are good examples of some Landmark Checkpoints. We use these Landmark Checkpoints to process the deployment logs offline and estimate the localization errors of the CoBots. The scan matching algorithm is a computationally intensive operation, since it is evaluated over all possible locations and orientations in a $2\text{m} \times 2\text{m}$ search window at a resolution of 0.02m and 5° . Therefore the scan matching can only be performed offline for the evaluation of localization accuracy. From the deployment logs, at every instant that a CoBot estimated that it was near a Landmark Checkpoint, the last observed laser rangefinder scan or depth image obstacle scan (depending on what was available on that particular CoBot) is used to estimate the instantaneous most probable location of the

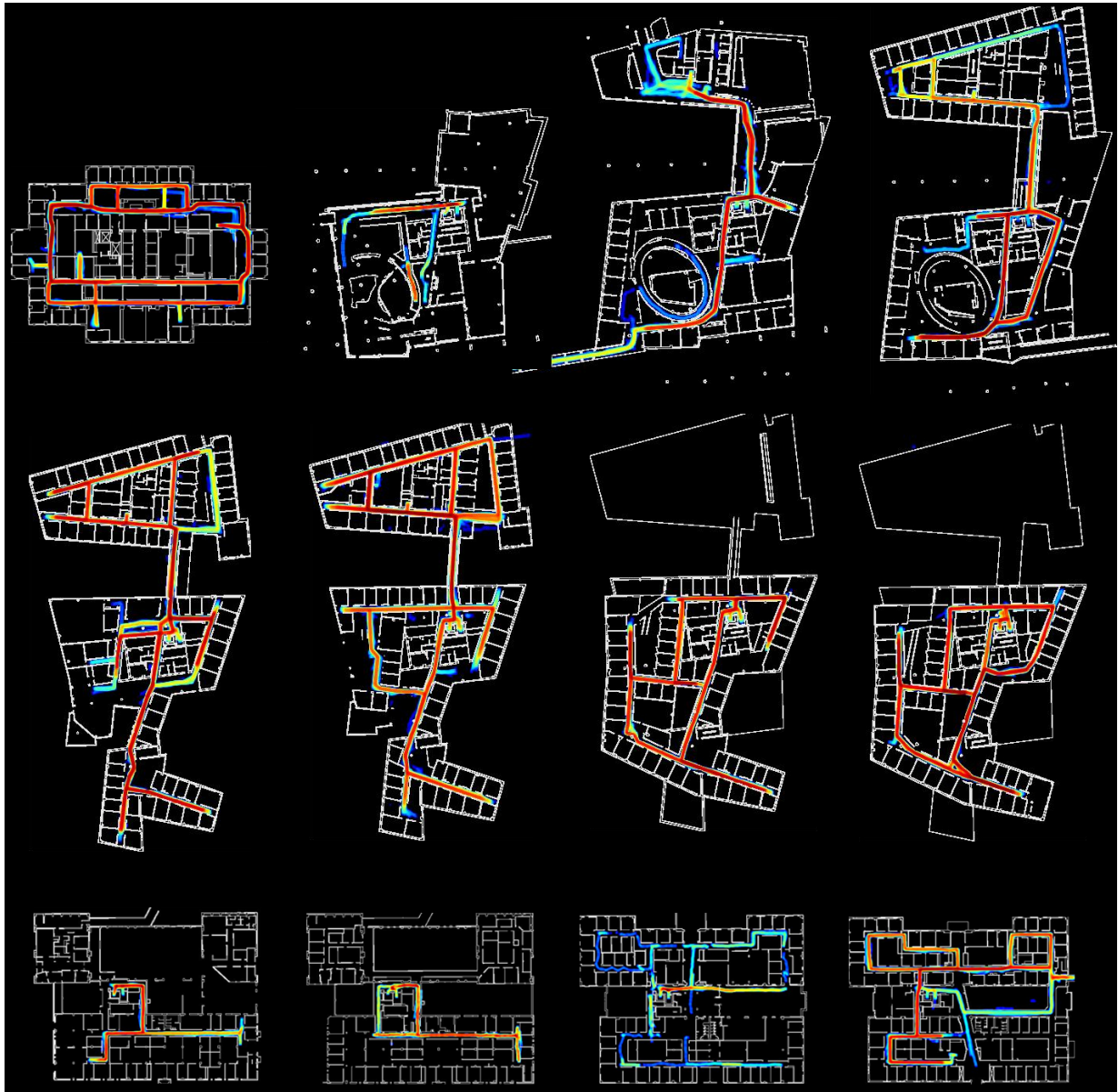


Figure 7.8: Combined traces of the paths traversed by the CoBots over the 1000km Challenge: (from left to right, top to bottom) NYU19, GHC3, GHC4, GHC5, GHC6, GHC7, GHC8, GHC9, NSH1, NSH2, NSH3, and NSH4. Locations on the map are color-coded by the frequency of visits, varying from dark blue (least frequently visited) to red (most frequently visited). Since the CoBots visited each floor a different number of times, the color scales are different for each floor.

Robot	Distance (km)
CoBot1	36.6
CoBot2	548.2
CoBot3	205.1
CoBot4	216.1

Table 7.2: Distances traversed by each robot during the 1000km Challenge

Floor	Deployments
GHC3	15
GHC4	158
GHC5	50
GHC6	315
GHC7	2135
GHC8	150
GHC9	164
NSH1	9
NSH2	6
NSH3	9
NSH4	80
NYU19	108

Table 7.3: Breakup of the times each floor was visited by the CoBots during the 1000km Challenge

CoBot by scan matching. When the robots are deployed, they do not slow down or stop specifically at the Landmark Checkpoints: the online localization and navigation algorithms are even unaware of the locations of the Landmark Checkpoints. By comparing the instantaneous most probable location computed by scan matching to the localization estimates of the robot from the deployment log, we estimate the error in localization at that particular instant.

Table 7.4 lists the localization errors evaluated by scan matching at Landmark Checkpoints on the different maps. There are 167 LandMark Checkpoints in total over all the floors. Maps NSH1-4, GHC3 and NYU19 have been omitted since there were insufficient Landmark Checkpoints encountered on those maps. Figure 7.9 shows the histograms of errors in localization computed by scan matching at the Landmark Checkpoints. Figure 7.10 shows scatter plots of the errors of the robot localization estimates relative to the corresponding Landmark Checkpoints, for each of the different maps. Figure 7.11 shows a similarly computed scatter plot of the localization errors combined for all the maps.

Map	Samples	Mean error (m)	Median error (m)	Std. Dev. (m)
GHC4	294	0.165	0.104	0.159
GHC5	218	0.132	0.101	0.113
GHC6	669	0.105	0.075	0.113
GHC7	6626	0.132	0.083	0.140
GHC8	527	0.107	0.055	0.145
GHC9	806	0.136	0.080	0.164

Table 7.4: Localization accuracy evaluated by scan matching at Landmark Checkpoints for the different maps over the course of the 1000km Challenge.

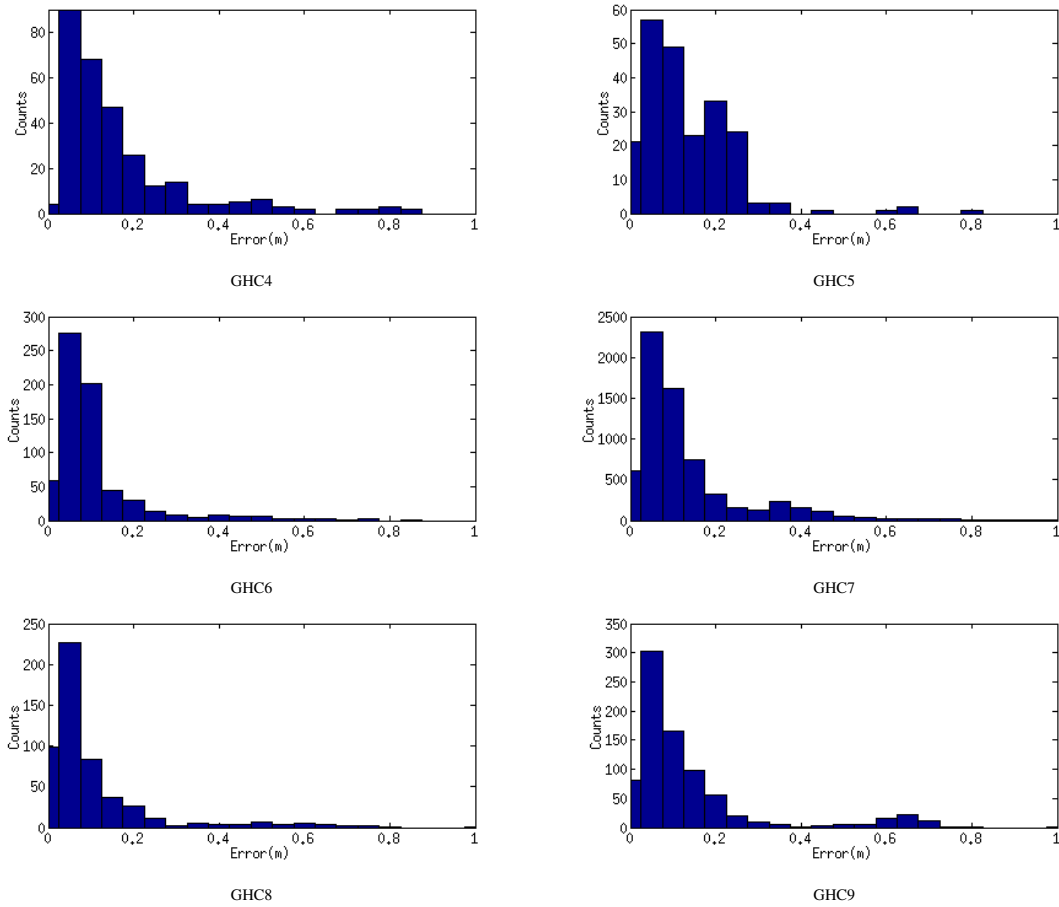


Figure 7.9: Histogram of scan matching errors on each floor during the 1000km Challenge

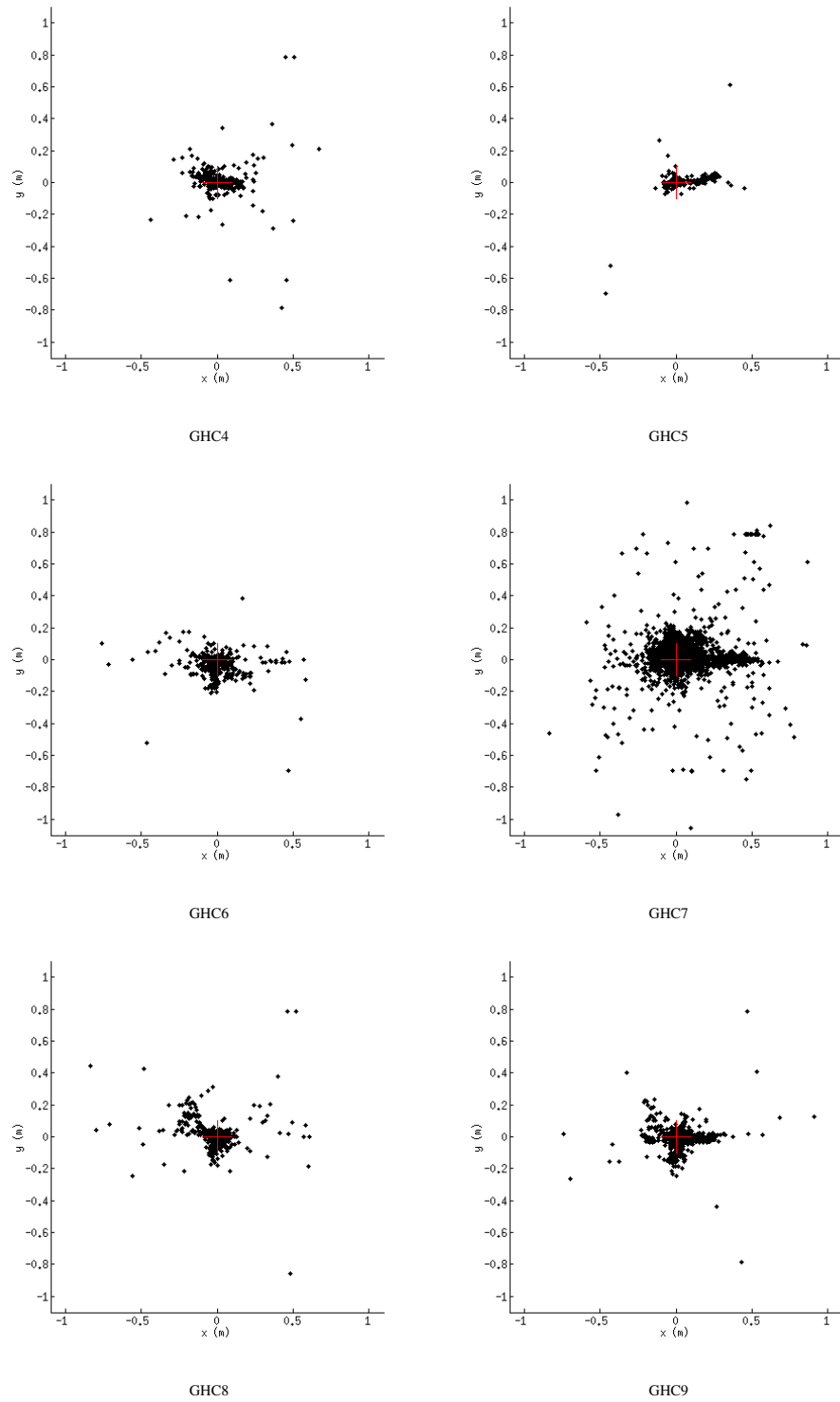


Figure 7.10: Scatter plots of localization errors evaluated by scan matching during the 1000km Challenge

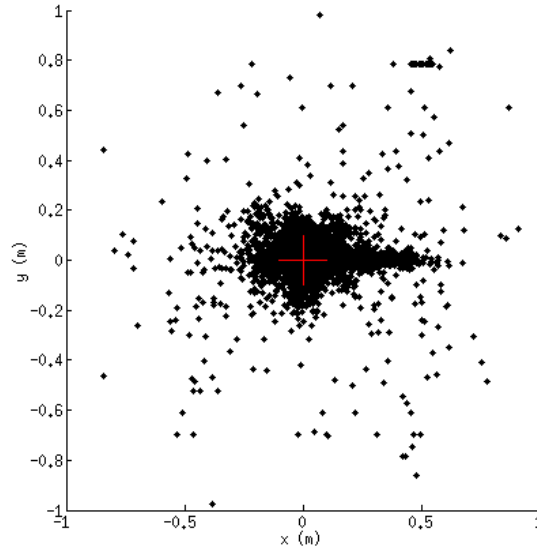


Figure 7.11: Scatter plots of localization errors over all maps during the 1000km Challenge

7.3.2 Accuracy Compared to Sparse Ground-Truth

CoBot2 and CoBot3 are equipped with Hagisonic StarGazer² sensors, which detect StarGazer marker patterns mounted on the ceiling and provide relative locations of detected markers with respect to the sensor. Each StarGazer marker consists of four or more retro-reflective dots arranged on a 4×4 grid. The absence or presence of specific dots on the grid encodes a unique ID number for each marker. We have 46 StarGazer markers placed on the ceiling throughout the Gates-Hillman Center to provide sparse ground-truth location estimates. The locations of the markers were tabulated manually by measuring their locations relative to nearby map features. While providing absolute ground truth information, the StarGazer markers have a few known limitations:

1. The marker locations were tabulated manually, and are subject to human placement and measurement errors.
2. The StarGazer sensor reports observed markers at 5Hz, along with some processing latency. The location estimates of the robot would thus differ when the robot is moving.
3. The StarGazer sensor assumes it is always parallel to the markers, but the sensor would in practice tilt with the acceleration and deceleration of the robot.

When a CoBot detects a StarGazer marker, the localization estimates are compared to the global location of the observed StarGazer marker to compute the error in localization. Fig-

²<http://eng.hagisonic.kr/cnt/prod/prod010102?uid=11&cateID=2>

Figure 7.12 shows the histograms of errors in localization evaluated by sparse ground truth, and Table 7.5 lists the mean, median and standard deviations of errors for the different maps.

Map	Samples	Mean error (m)	Median error (m)	Std. Dev. (m)
GHC4	7656	0.352	0.343	0.218
GHC5	5123	0.394	0.400	0.181
GHC6	1235	0.296	0.314	0.122
GHC7	17489	0.368	0.341	0.190
GHC8	50	0.234	0.167	0.185
GHC9	276	0.464	0.402	0.232

Table 7.5: Localization accuracy by sparse ground truth for the different maps over the course of the 1000km Challenge.

7.3.3 Robustness

While deployed, the CoBots log the operator interventions provided. To gauge the real-world robustness of our localization algorithms, we count the number of times operator interventions were required to reset localization while the robot was operating autonomously. Table 7.6 lists the number of times operator interventions were required for all the deployments for each map during the 1000km Challenge. More than 93% of the deployments proceeded without any localization interventions, and there were no deployments with more than three interventions.

Map	Interventions				
	0	1	2	3	> 3
GHC3	14	1	0	0	0
GHC4	126	31	0	1	0
GHC5	44	6	0	0	0
GHC6	290	21	3	1	0
GHC7	2047	77	9	2	0
GHC8	140	9	1	0	0
GHC9	156	8	0	0	0
NSH1	7	2	0	0	0
NSH2	4	0	1	1	0
NSH3	8	1	0	0	0
NSH4	66	13	0	1	0
NYU19	101	7	0	0	0
Total	3003	176	14	6	0

Table 7.6: Operator interventions over the 1000km Challenge, listed as the number of deployments with 0, 1, 2, 3, and > 3 interventions per deployment, for each map.

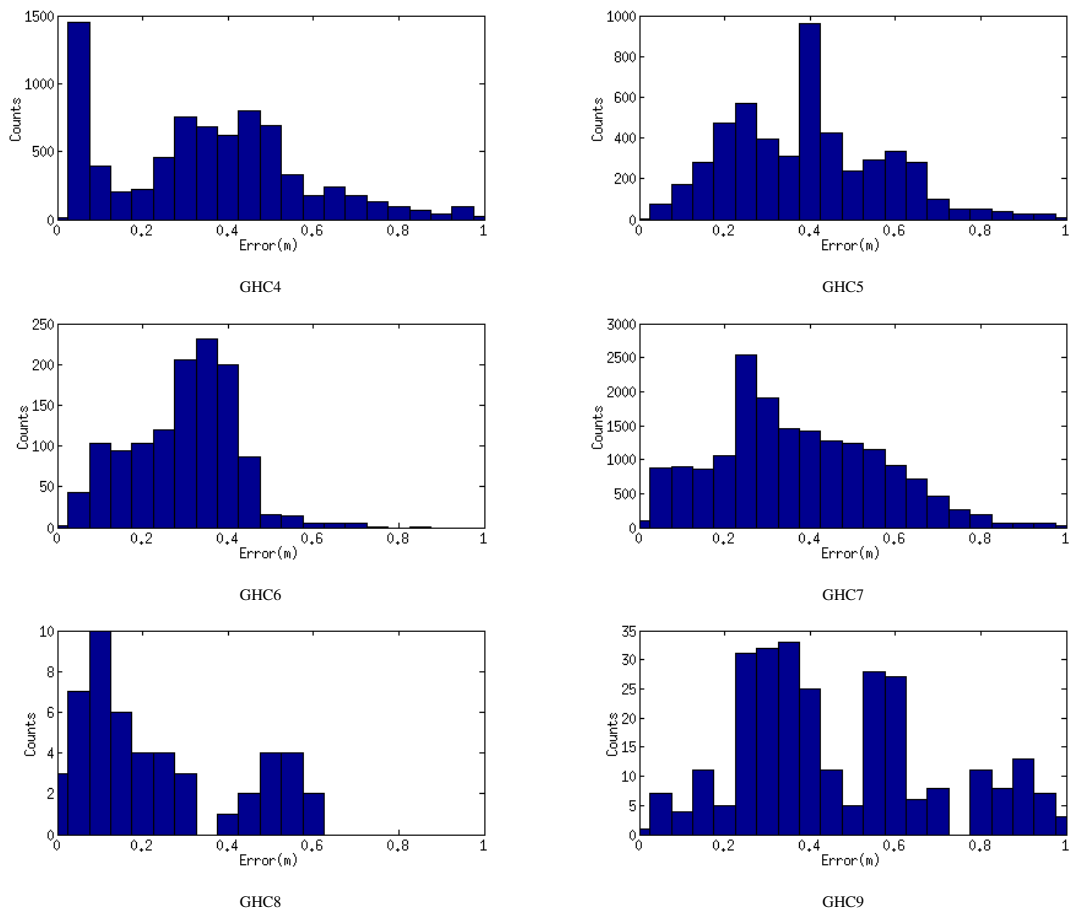


Figure 7.12: Histogram of errors evaluated by sparse ground truth on each map

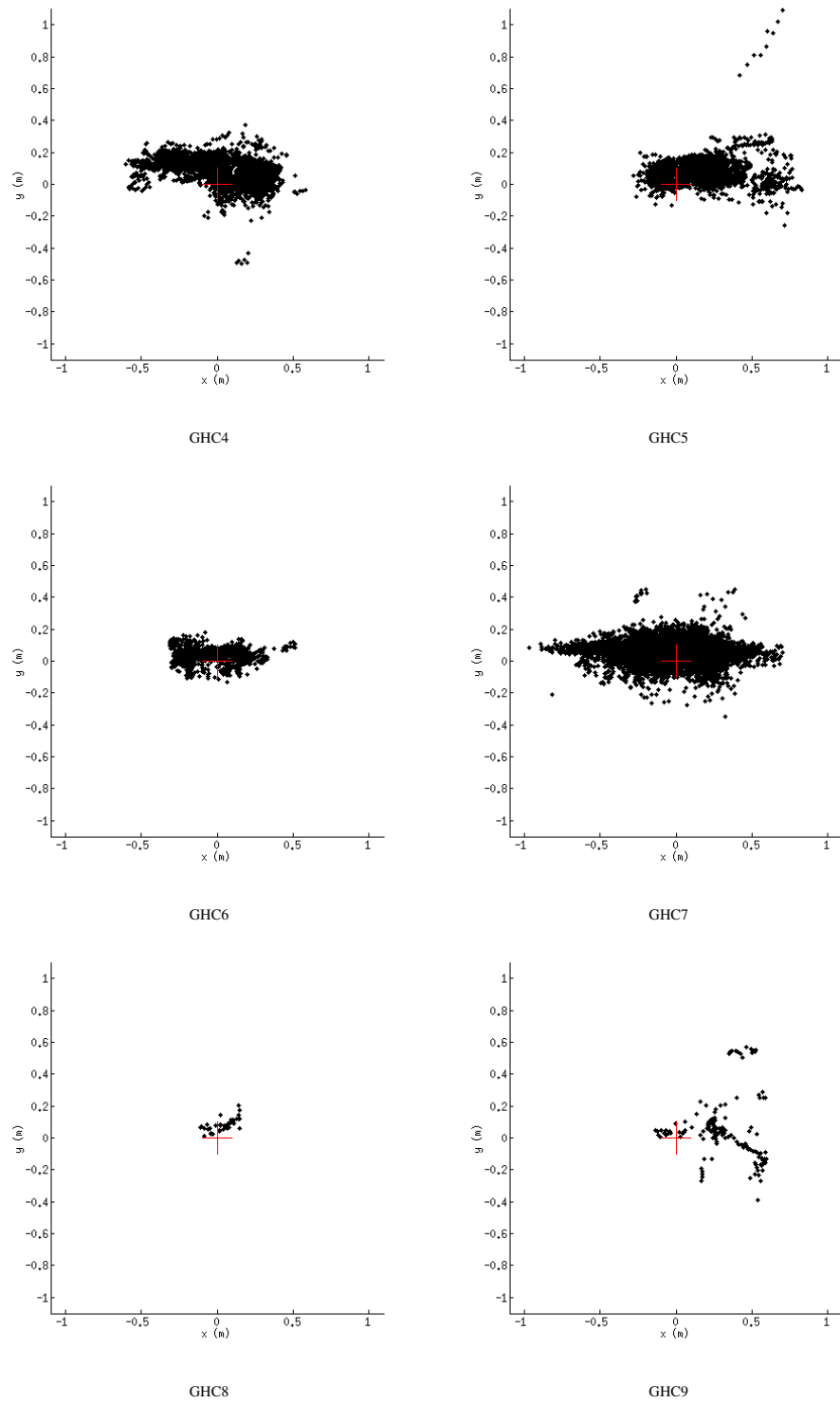


Figure 7.13: Scatter plots of localization errors evaluated by sparse ground truth during the 1000km Challenge

Over the duration of the 1000km Challenge, Corrective Gradient Refinement (Chapter 4.2.1) was used for localization between September 2011 and January 2014, while Episodic non-Markov Localization (Chapter 5) was used February 2014 onwards. Table 7.7 compares the mean distance traversed by the robot between interventions, for each map, when using CGR, and when using EnML for localization. The mean distance traversed between interventions is significantly higher for EnML than for CGR, thus demonstrating the higher reliability of EnML for localization in real-world human environments. Note that the maps GHC3, GHC5, NSH1, NSH2, NSH3 and NYU19 are omitted from this analysis since we have insufficient data for both algorithms to compare the mean distance between interventions.

	CGR	EnML
GHC4	0.62	4.42
GHC6	8.61	9.48
GHC7	5.58	9.02
GHC8	6.04	19.36
GHC9	5.33	20.05
NSH4	0.56	2.65
All	4.79	8.13

Table 7.7: Mean distance (in km) traversed between interventions using CGR and EnML per map over the 1000km Challenge.

We also logged how far the robots traversed autonomously before operator interventions were required. As listed in Table 7.6, there were 196 deployments with one or more deployments. For those deployments, Figure 7.14 shows a histogram of how far the robot traversed autonomously before the first operator intervention. We noticed that most of the interventions were required shortly after the robot started moving, within the first 0.5km. This is largely due to the operator providing inaccurate localization initialization at the beginning of the deployments. Furthermore, of the 26 instances where two successive operator interventions were required, Figure 7.15 shows a histogram of how far the robot traversed autonomously between successive operator interventions. Again, it is seen that the vast majority of successive interventions happen before the robot traverses more than 0.5km after the first intervention. This is because when an intervention is required, in many cases it is hard even for the operator to correctly re-initialize the robot's location, thus requiring further interventions shortly thereafter.

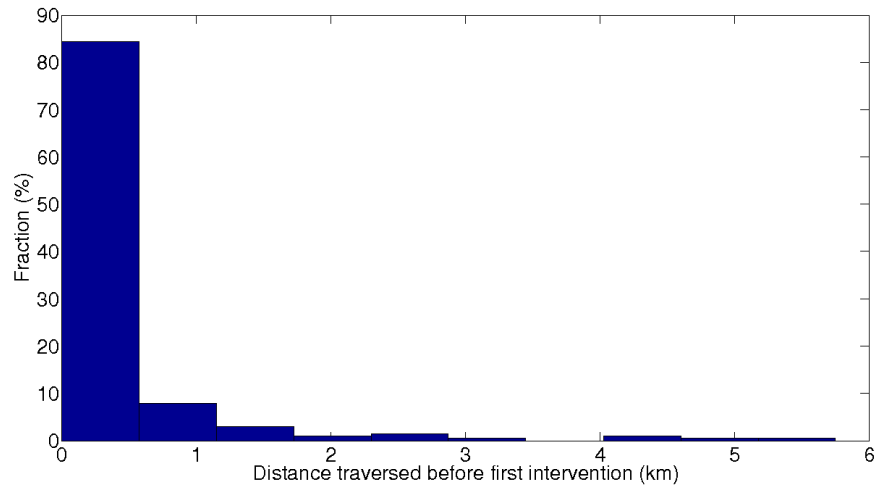


Figure 7.14: Histogram of the distance traversed by the robot before the first operator intervention, in terms of the percentage of the number of times that operator interventions were required in a deployment.

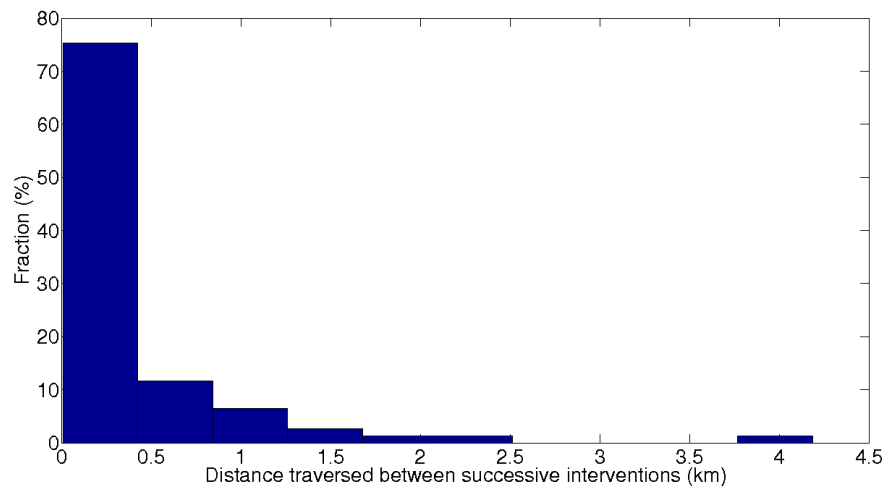


Figure 7.15: Histogram of the distance traversed by the robot between successive operator interventions, in terms of the percentage of the number of times that more than one operator intervention was required in a single deployment.

7.4 Lessons Learned

The continued deployments of the CoBots provide us with valuable insights into the limitations and challenges of deploying service mobile robots in real human environments. We list the lessons learned under the sub-categories of Sensors, Localization, Navigation and System Integration.

7.4.1 Sensors

As mentioned earlier in Chapter 7.1, the different CoBots are equipped with different sensor combinations. Due to the variations in the sensing abilities, the different CoBots vary in their ability to handle different types of environments.

With the upward-facing depth camera, CoBot2 is well-suited for deployments in areas with tall chairs and tables. CoBot4, with its downward-facing depth camera, is capable of detecting small obstacles on the ground, like laptop power adapters or cables.

The laser rangefinder, with its larger field of view than the depth cameras, make CoBot1, CoBot2 and CoBot3 better adapted to deployments in open areas where the only observable features are often far off to the side of the robot instead of within the narrow field of view of the depth cameras.

Neither the laser rangefinder, nor the depth camera sensors on the CoBots, are rated for use in direct sunlight. As a result, occasionally, when the CoBots encounter direct bright sunlight, they detect false obstacles and the obstacle avoidance algorithm brings the robots to a stop.

7.4.2 Localization

The sensors used for localization on the CoBots have limited sensing range (Chapter 3.2), and cannot observe the entire length of the hallway that the CoBot is in, for most of the time. Therefore, it is up to the localization algorithms to reason about the uncertainty parallel to the direction of the hallway. This is in stark contrast to a scenario where a robot with a long-range sensor (like the SICK LMS-200 laser rangefinder, with a maximum range of 80m) is able to observe the entire length of every hallway (the longest hallway in the GHC building is about 50m in length), and hence is able to accurately compute its location with a single reading. The decision to use inexpensive short-range sensors is motivated by cost, since our goal included deploying several CoBots, and by the goal to investigate algorithms robust to sensor limitations. Despite the limitation of the sensor range, the CoBots repeatedly stop at exactly the right destination locations in front of office doors, and always follow the same path down hallways (when there are no obsta-

cles). In fact, in a number of hallways, the repeated traversal of the CoBots along the exact same paths has worn down tracks in the carpets.

As presented in Chapter 7.3.3, Episodic non-Markov Localization has dramatically increased the robustness in localization of the CoBots, and has further permitted deployments in challenging areas like the floors GHC4 and GHC5, where most of the observations made by the robots consist of movable objects.

7.4.3 Navigation

In our experiences with extended deployment of the CoBots, we have come to realize that a conservative approach to navigation is more reliable in the long term as compared to a more unconstrained and potentially hazardous approach. In particular, the obstacle avoidance algorithm uses a local greedy planner, which assumes that paths on the navigation graph will always be navigable, and can only be blocked by humans. As a result, the planner will not consider an alternative route if a corridor has a lot of human traffic, but will stop before the humans and ask to be excused. Furthermore, due to the virtual corridors, the robot will not seek to side-step obstacles indefinitely. While this might result in longer stopped times in the presence of significant human traffic, it also ensures that the robot does not run into invisible obstacles in the pursuit of open paths. Although there exist many hallways with glass walls, the robot has never come close to hitting them, thanks to its reliable localization and virtual corridors.

One drawback of the constrained navigation is that if the localization estimates are off by more than half the width of a corridor intersection (which is extremely rare, but has occurred), the obstacle avoidance algorithm will prevent the robot from continuing, perceiving the corner of the walls at the intersection as an obstacle in its path. In such circumstances, issuing a remote command to the robot (via its telepresence interface) to turn and move to the side is sufficient for the robot to recover its localization (by observing the true locations of the walls), and hence the navigation as well.

7.4.4 System Integration

As described in Appendix A, CoBots rely on significant automation in order to ensure continued reliable operation. During deployments, the CoBots are accessible remotely via a telepresence interface that allow members of the research group to examine the state of the robot from the lowest (sensor) levels to the highest (task execution) levels. When a CoBot is blocked for task execution due lack of human responses to interaction, it automatically sends an email to the research group mentioning its latest location estimate, task status, and reason for being blocked.

The ability of the CoBots to proactively email for assistance when required has proven to be invaluable on a few unexpected occasions. In one such case, one building occupant was annoyed by the noise of the robots passing by his office, and sought to stop the robot by trapping it by placing cardboard boxes around it. The robot detected that its path was blocked from all sides, and emailed the developers for assistance. Since then, we have limited the maximum speed of the robots while passing by that office, in order to reduce its noise.

We are currently at the point where the physical intervention required to manually unplug the charger from the robot is the bottleneck in deploying the CoBots. Therefore we are exploring designs for an automatic charging dock that will be robust to small positioning errors of the robot, durable in order to withstand thousands of cycles per year, and yet be capable of transferring sufficient power to charge the robot base as well as the laptop at the same time.

7.5 Summary

In this chapter we presented the results of running the CoBots during the 1000km Challenge, spanning deployments over several years in varied environments across multiple floors and buildings. Despite variations and changes in the environments, the robots successfully autonomously traversed all the public areas over the years of deployments of the robots. We further presented quantitative results of the localization accuracy and robustness over the course of all their deployments, and discussed lessons learned.

Chapter 8

Conclusion and Future Work

In this chapter we summarize the contributions of the thesis, discuss some possible avenues for future work, and conclude with a brief summary of the thesis.

8.1 Contributions

The key contributions of this thesis are as follows.

MCL-CGR for accurate and robust localization with low computational cost in indoor human environments with limited variations over time. We introduced Monte-Carlo Localization with Corrective Gradient Refinement (MCL-CGR) to use an analytic sensor likelihood function with a vector map representation of the environment. We experimentally showed that MCL-CGR has greater localization accuracy, lower variance, and lower computational requirements than MCL-SIR.

FSPF for plane-filtering of depth images, and FSPF-MCL-CGR for localization with inexpensive depth cameras in human indoor environments with limited variations over time. We introduced the Fast Sampling Plane Filtering (FSPF) algorithm that samples the depth image to produce a set of points corresponding to planes, along with the plane parameters (normals and offsets). We then introduced an observation model that matches the plane filtered points to the existing 2D vector maps. Finally, we used MCL-CGR to localize the robot using the plane filtered points. We experimentally demonstrated that a robot with an inexpensive depth camera running MCL-CGR with FSPF to accurately and robustly localize it in common office environments with comparable results to localization using MCL-CGR and the much more expensive laser rangefinders.

EnML for accurate and robust localization in varying environments with many moving objects and frequent changes. We introduced Episodic non-Markov Localization (EnML) to localize a robot in environments with many dynamic and movable objects. We introduced the Varying Graphical Network (VGN), a novel graphical model to keep track of correlations between successive poses based on odometry, between pairs of poses from short-term features, and between each pose and the map from long-term features. We introduced a cost function representation of the belief to efficiently compute the maximum likelihood estimate of the robot's location by performing non-linear least-squares optimization of the cost function. We experimentally demonstrated that EnML is able to localize accurately and reliably even in challenging varying environments, and performs favorably compared to alternative approaches.

Plane-Polygon Map Update and Model-Instance Object Mapping to updated maps with sensor logs gathered over extended deployments We introduced the Plane-Polygon Map update algorithm to build and update sparse 3D maps by extracting the dominant planar features of the environment. The Plane-Polygon Map thus constructed are considerable more compact than alternative, high-resolution 3D maps. We further introduced Model-Instance Object Mapping, an approach that reasons about the shapes, or *models* of objects independently of their observed copies, or *instances*. Model-Instance Object Mapping detects models as well as the instances of commonly observed movable objects in an entirely unsupervised manner. We demonstrated the effectiveness of Model-Instance Object Mapping by presenting the learnt models and their instances from sensor logs of deployments of the robots autonomously traversing more than 55km.

Extensive experimental results of over 1000km traversed by robots using the proposed localization algorithms. We presented results from continuous deployments of four robots, CoBots 1-4 in real academic buildings at Carnegie Mellon University as well as New York University. The robots were deployed in multiple buildings, over multiple floors, and exposed to a variety of indoor environments. Unchaperoned and unmonitored, they successfully completed numerous tasks over their deployments, while maintaining accurate location estimates using the algorithms presented in this thesis. We contributed data analysis along multiple dimensions, including data visualization, evidence of algorithm correctness, and statistics of performance.

8.2 Future Work

We introduced Episodic non-Markov Localization as an approach to reason about the nature of the observations made by a robot, in terms of the permanence of the objects being observed. There are several possible interesting areas of future work that are possible by extending EnML and relaxing its assumptions. Ideally, a long-term localization algorithm would not only reason about the nature of its observations, it should also build its maps of such expected observations over time without requiring an initial map estimate. Such an approach would be similar to SLAM in that it would result in building a map, but different in that not every observation would be added to the map, in particular when multiple observations contradict each other. Tracking dynamic features, which EnML currently does not do, is another logical next step. Accounting for Joint compatibility, and revisiting conflicting data associations would make EnML more robust to initial localization and data association errors. Finally, as an eventual goal, an approach to *Generalized Perception* should reason about object detection, tracking and mapping at the same level as mobile robot localization.

Extending EnML to initialize and build long-term maps Currently, EnML requires an initial estimate of the long-term static map of the environment. Extending EnML to be able to robustly generate its own initial estimate of the long-term static map would require reasoning about loop closures with observations of only short-term features. In the absence of any long-term features (as would be the case if EnML had no initial long-term static map), the episodes would span the full duration of the initial deployment of the robot, thus requiring computationally efficient ways of optimizing over the Variable Graphical Network. Furthermore, building a long-term static map would require intelligent conflict resolution between observations when the robot observes movable objects at different locations at different times, requiring the robot to reason about objects necessarily being short-term features if the space occupied by them was previously observed to be vacant.

Tracking dynamic features in EnML There has been some work in tracking moving objects while performing SLAM [92]. Such tracking of moving objects could also be performed by EnML. Tracking moving objects would allow EnML to better hypothesize about the expected observations arising from moving objects in future time-steps, thus further improving its data associations and consequently its location tracking.

Joint compatibility for correspondence matching in EnML Joint compatibility is inherently computationally expensive, since it involves reasoning about the exponential number of possible

data association pairings. However, using a strict bounding threshold in Joint Compatibility with Branch and Bound (JCBB) [61] could potentially allow EnML to reason about a limited number of most likely alternate data associations. This would lead to increased robustness of location tracking in the presence of poor initial estimates.

Generalized Perception Currently robots perform object detection and tracking independently from localization. Furthermore, there are even specialized detectors for certain classes of objects like humans or doors. Ideally, the information extracted from object detection and tracking could better guide localization by reasoning about whether a particular object is likely to be a long-term feature, short-term feature or dynamic feature. For example, a table is more likely to be an short-term feature than an long-term feature, and a human is likely to be a dynamic feature. The location estimates of such a generalized perception algorithm would, in return, aid in better tracking of objects.

8.3 Summary

This thesis introduced several new map representations, observation models, and mobile robot localization algorithms. We introduced a vector map representation, and the analytic ray cast algorithm to efficiently compute observation likelihood models on the vector map. We introduced the Fast Sampling Plane Filtering algorithm that processes depth images to extract planar features from them even in the presence of clutter. We developed the Corrective Gradient Refinement algorithm that used the vector map representation to analytically refine the proposal distributions of particle filters used for localization user laser range finders or depth cameras processed by FSPF, thus resulting in more accurate and robust location estimates while requiring fewer particles than MCL-SIR. We introduced Episodic non-Markov Localization to reason about observations arising from long-term, short-term, or dynamic features, and thus explicitly reason about correlations between observations of unmapped objects from different time steps. We presented algorithms for updating planar polygon maps, as well as a Model-Instance map representation of movable objects in the environment. Finally, we presented extensive results from the 1000km Challenge, demonstrating the accuracy and robustness of the localization algorithms over robot deployments in real human environments spanning multiple buildings and several years.

Appendix A

Maintaining Perpetual Deployability of Mobile Robots

This thesis introduced a number of theoretical contributions to mobile robot localization that have enabled the long term deployment of the CoBots in increasingly challenging environments. The 1000km Challenge demonstrated the accuracy and robustness of our algorithms over extensive deployments. However, on a scale of experiments this large, spanning multiple years, multiple robots, and multiple buildings, algorithms alone are not enough to maintain perpetual deployability. The goal of this chapter is to try to explain some of the ingredients of the secret sauce that enabled us to have the CoBots be perpetually deployable.

While there are specific techniques that we used for the individual components of the robot, a common theme is to keep it as simple as possible. A non-functional system requires significantly more knowledge and insight than building the same system. Therefore, the cleverer that a designer attempts to be while designing a system, the more frustrating it is to debug the overly clever system. In contrast, if the system is bare minimal, and designed in the simplest manner possible to get the task done, the simpler it will be to understand and debug.

A.1 Software Architecture

The software architecture on the CoBots is designed to be modular and easily extensible. Figure A.1 shows the software architecture on the CoBots. There are multiple software nodes, including the robot hardware drivers, localization, navigation, the graphical user interface, the task planner, the server interface, and the scheduling server. All software nodes except the scheduling server run on the onboard computer on the CoBots. The scheduling server runs on a central

off-board server. The software nodes communicate with each other using ROS¹. There are two forms of inter-node communications: *topics* and *services*. Topics are data streams, which may have multiple publishers and multiple subscribers. Services are one-to-one interfaces for sending dedicated commands from one node to another. The services on the CoBots include odometry, laser rangefinder scans, depth image obstacle scans, filtered point clouds, robot location estimates, and drive commands.

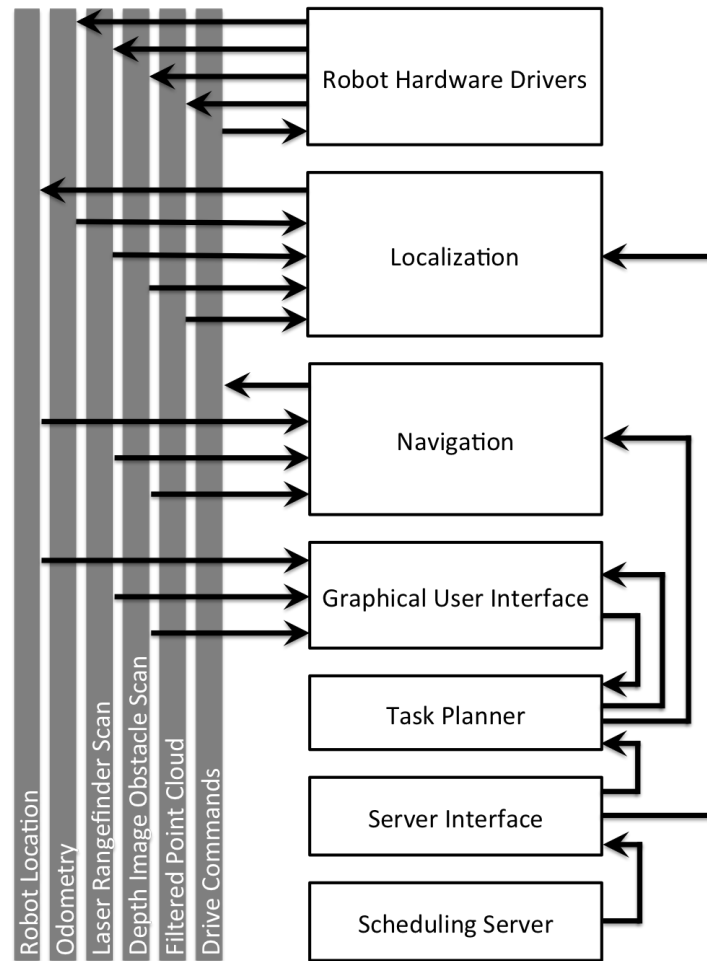


Figure A.1: CoBot Software Architecture. Software nodes, including the robot hardware drivers, localization, navigation, the graphical user interface, the task planner, the server interface, and the scheduling server send and receive data via multiple topics (columns on the left). The different software nodes send control commands to each other via services (arrows on the right).

Due to the modular nature of the software architecture, it is easy to add and remove functionality to the CoBots. For example, the localization software node may be running EnML (Chap-

¹<http://www.ros.org>

ter 5) or CGR (Chapter 4.2.1) interchangeably, and the rest of the software is agnostic to exactly which algorithm is running, since software nodes communicate via the pre-defined topics.

A.2 Software Automation

A number of steps involved in maintaining the deployability of a mobile robot can be automated, including the software startup, log collection and management, and maintaining the source code on the robots.

A.2.1 Startup Automation

Starting up and deploying a mobile robot involves a long sequence of steps that quickly becomes tedious when deploying the robot frequently. To make it as easy as possible to deploy the robot, we created startup scripts to automate the robot startup process as well as run diagnostics checks once started up to detect startup failures. The startup script is launched from a one-touch icon on the desktop of the robot's touch-screen tablet, thus allowing deployers to start up the robot with a single press of a button. The startup script performs the following tasks:

1. Detect the robot type from the configuration file, and select the appropriate set of sensors and software nodes to launch.
2. Load the configuration files specific to the robot and publish the parameters for all software nodes.
3. Enumerate all connected USB devices, and auto-detect which USB device corresponds to which expected sensor.
4. When multiple sensors are detected, distinguish between them by their serial number, and load their specific calibration files.
5. Start up all sensor drivers with command-line parameters specifying the automatically detected port numbers of the devices.
6. Start up all software nodes.
7. Check for internet connectivity to the central scheduling server, and attempt to restart internet connection if necessary.
8. Check that all sensors are publishing at the expected rates.
9. Check if the depth sensors detect the ground plane at the location expected by their extrinsic calibration.

If any of the startup steps fail, the startup is terminated, and a spoken error message played on the robot's computer speakers describes what the cause of the failure was. An example such error could be "The Kinect is not publishing at the expected rate", or "Failed to auto-detect the laser rangefinder."

A.2.2 Log Management Automation

The CoBots record logs during every deployment, collecting over a Gigabyte of sensor data per hour of deployment. The volume of data logs thus generated would understandably be challenging to manage manually. Instead, on every robot, we have nightly log management scripts which perform the following tasks at 4AM everyday:


1. Compress every data log
2. Transfer the compressed data logs to the central server
3. If the data logs transferred successfully, delete the logs from the robot.




Once the data logs have all been transferred to the central server, they are further processed on the server by a separate log processing script running on the server. This log processing script performs the following tasks at 5AM everyday:

1. Compute the total autonomous distance traversed for each deployment
2. Generate a synopsis image plotting the trajectory of the robot over the deployment
3. Compute localization errors over the deployments using StarGazer observations as well as scan matching (Chapter 7.3)
4. Extract the locations and times of operator interventions, if any, during the deployments
5. Convert the data logs into a standard format such that it may be readily shared with researchers
6. Generate synopses of the deployments including the tasks performed and sensor data logged
7. Email a synopsis (Figure A.2) of the deployments to the developer

Occasionally a robot may fail to transfer its logs automatically, if it has a poor wireless internet connection. In such a case, an error log is locally saved, and the deployer may check the logs and transfer them manually later.

The vast majority of the data logs collected by the CoBots are automatically managed by the log management scripts presented here without any developer intervention. Occasionally the developer may refer to the email synopses and inspect specific logs in case of an unusual occurrence, for example unusually long deployments.

Cron <joydeepb@neontetra> /home/joydeepb/cobot/cobot_linux/scripts/nightly_generate_synopsis_and_trajectory 

 **Cron Daemon** root@neontetra.coral.cs.cmu.edu [via](#) ri.cmu.edu Nov 8 (4 days ago) 
to joydeepb 

Processing logs from CoBot1
Processing logs from CoBot2
/data/cobot2_logs/2014-11-07-11-16-08.bag 268.4s 100.2m 2673 8042 100.2m 3014 3345 1
/data/cobot2_logs/2014-11-06-15-18-16.bag 348.5s 0.0m 3475 10443 0.0m 67 4343 0
/data/cobot2_logs/2014-11-06-14-56-22.bag 726.1s 0.0m 7240 21753 0.0m 434 9053 0
Total distance traversed: 0.100165 km
Total laser scans: 2673
Total Kinect scans: 40238
Total stargazer sightings: 948
Total duration: 268.418150
Not interesting: /data/cobot2_logs/2014-11-06-15-18-16.bag
Checking /data/cobot2_logs/2014-11-07-11-16-08.bag
Not interesting: /data/cobot2_logs/2014-11-06-14-56-22.bag
Checking /data/cobot2_logs/2014-11-06-14-56-22.bag.synopsis
Checking /data/cobot2_logs/2014-11-07-11-16-08.bag.synopsis
Checking /data/cobot2_logs/2014-11-06-15-18-16.bag.synopsis
Converting /data/cobot2_logs/2014-11-07-11-16-08.bag
Writing to /data/cobot2_logs/converted/2014-11-07-16-17-32.bag
Processing logs from CoBot3
/data/cobot3_logs/2014-11-07-11-51-57.bag 6.5s 0.0m 66 190 0.0m 82 82 0
/data/cobot3_logs/2014-11-07-11-51-35.bag 6.6s 0.0m 66 194 0.0m 81 81 0
Total distance traversed: 0.000000 km
Total laser scans: 0
Total Kinect scans: 384
Total stargazer sightings: 0
Total duration: 0.000000
Not interesting: /data/cobot3_logs/2014-11-07-11-51-35.bag
Not interesting: /data/cobot3_logs/2014-11-07-11-51-57.bag
Checking /data/cobot3_logs/2014-11-07-11-51-57.bag.synopsis
Checking /data/cobot3_logs/2014-11-07-11-51-35.bag.synopsis
Processing logs from CoBot4
/data/cobot4_logs/2014-11-07-15-50-02.bag 4.5s 0.0m 0 131 0.0m 56 56 0
/data/cobot4_logs/2014-11-07-15-50-32.bag 4.5s 0.0m 0 129 0.0m 56 56 0
/data/cobot4_logs/2014-11-07-15-51-09.bag 396.2s 0.0m 0 11869 0.0m 678 4943 0
/data/cobot4_logs/2014-11-07-15-57-54.bag 1864.4s 164.6m 0 55864 164.6m 23250 23250 1
Total distance traversed: 0.164554 km
Total laser scans: 0
Total Kinect scans: 67993
Total stargazer sightings: 0
Total duration: 1864.353963
Not interesting: /data/cobot4_logs/2014-11-07-15-50-02.bag
Not interesting: /data/cobot4_logs/2014-11-07-15-50-32.bag
Not interesting: /data/cobot4_logs/2014-11-07-15-51-09.bag
Checking /data/cobot4_logs/2014-11-07-15-57-54.bag
Checking /data/cobot4_logs/2014-11-07-15-50-02.bag.synopsis
Checking /data/cobot4_logs/2014-11-07-15-50-32.bag.synopsis
Checking /data/cobot4_logs/2014-11-07-15-51-09.bag.synopsis
Checking /data/cobot4_logs/2014-11-07-15-57-54.bag.synopsis
Converting /data/cobot4_logs/2014-11-07-15-57-54.bag
Writing to /data/cobot4_logs/converted/2014-11-07-20-58-17.bag
Writing to /data/cobot4_logs/converted/2014-11-07-21-07-06.bag
Total autonomous distance: 810.948km

Figure A.2: A synopsis email generated by the nightly processing script running on the central server, summarizing the results of processing the deployment logs from the previous day.

A.3 Log Explorer

Continuously deploying the CoBots quickly generates Gigabytes of data logs, which are hard to search through and process in order to answer simple queries like “How often did the robots visit the 4th floor in Newell-Simon Hall?”, or “How much distance has CoBot3 traversed since 24th October 2013 on the 6th floor of Gates-Hillman Center?” To address this, we developed a web-based Log Explorer interface to be able to browse through the data logs collected so far, and to answer simple deployment-related queries ². The Log Explorer has three components: a log to database importer, the website server, and the client-side Log Explorer website. The log to database importer processes all the deployment logs and imports their summaries into an SQL database for fast query and retrieval. The website server runs a Django-based interface to respond to queries from clients with summaries of logs extracted from the database using the client’s search queries. The client-side Log Explorer website, shown in Figure A.3, provides user-selectable query fields, which can be used to query the server. Once the server responds with the list of the summaries of the matching deployment logs, the Log Explorer website tabulates a list of the deployments and graphically shows the cumulative distances traversed by the CoBots over those deployments, their durations, as well as the latest progress of the 1000km Challenge. The user may also select any of the entries in the list of deployments to view detailed information about that deployment, as shown in Figure A.4.

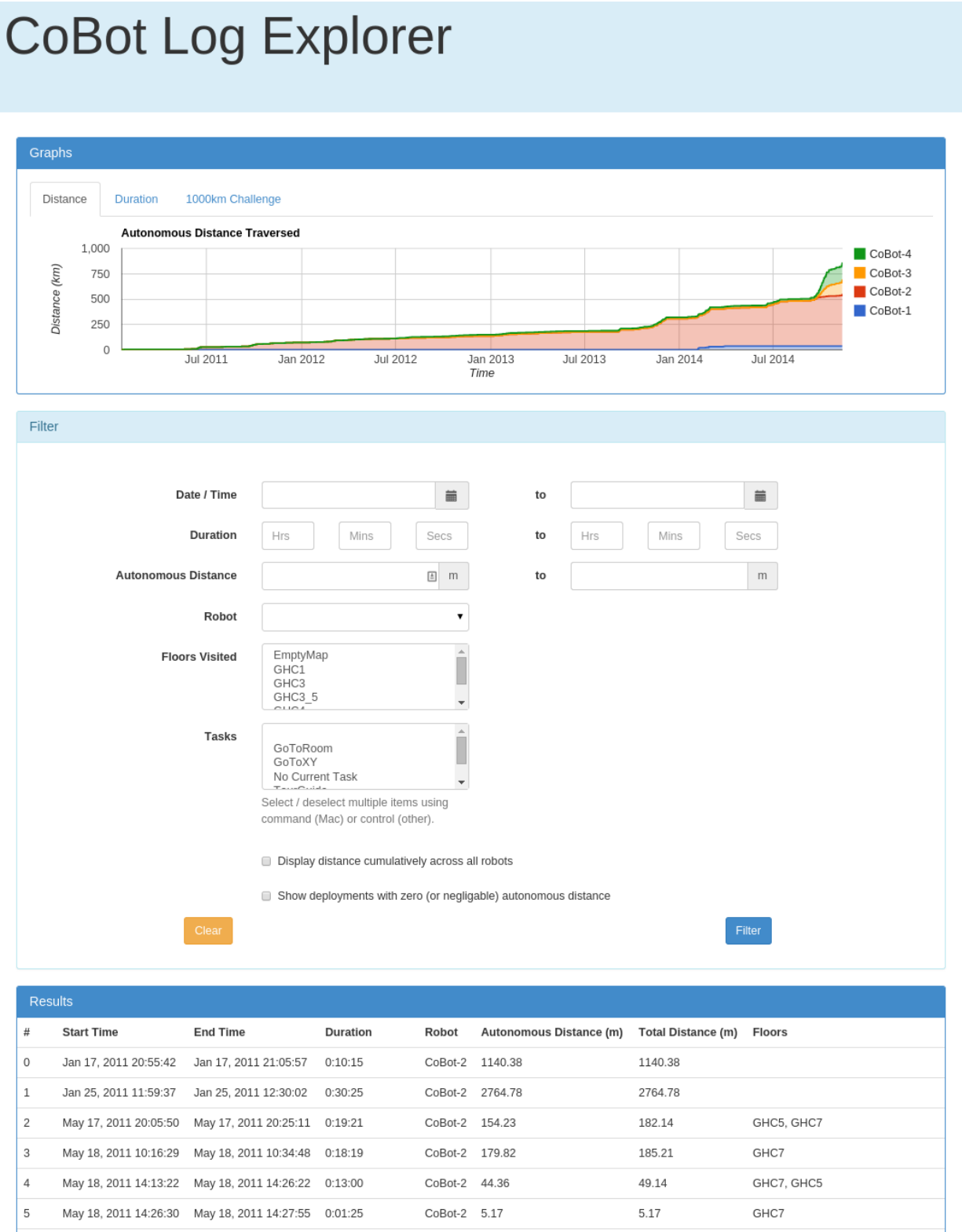
A.4 Source Code Management

As a research platform, the CoBots undergo software updates daily, as contributed by researchers working on different aspects of the autonomous service mobile robot problem. Hence, it is a continual challenge to ensure that code updates do not break core functionality of the CoBots. To help maintain a working code-base, we follow a number of code management procedures.

Code Version Control We use a Subversion³ repository to track changes to the code-base. While distributed version control systems like Git and Mercurial are popular in the larger community, in our experience Subversion works better for large teams of developers with a wide variety of skill levels. On other projects like the CMDragons RoboCup Small Size Soccer team [9] we have observed that inexperienced developers are easily bewildered by the branching and merging procedures that are inevitable with distributed version control systems. Another useful feature of

²Thank you to Michael Murphy (mjmurphy@andrew.cmu.edu) for developing the log explorer.

³<https://subversion.apache.org/>



Topic	Type	Messages
/Cobot/ArduinoSensors	cobot_msgs/ArduinoSensorMsg	3516
/Cobot/Drive	cobot_msgs/CobotDriveMsg	108385
/Cobot/Events	cobot_msgs/CobotEventsMsg	3261
/Cobot/FollowPath	cobot_msgs/CobotFollowPathMsg	39
/Cobot/HumanDetect/ClassifiedHumans	cobot_msgs/CobotHumansClassified	19190
/Cobot/HumanDetect/Humans	cobot_msgs/CobotHumansDetected	23058
/Cobot/Joystick	cobot_msgs/CobotJoystickMsg	108030
/Cobot/Kinect/DepthThrottled	sensor_msgs/Image	434
/Cobot/Kinect/Scan	sensor_msgs/LaserScan	130219
/Cobot/Kinect/Status	cobot_msgs/CobotKinectStatusMsg	260445
/Cobot/Localization	cobot_msgs/CobotLocalizationMsg	86733
/Cobot/Odometry	cobot_msgs/CobotOdometryMsg	86735
/Cobot/Status	cobot_msgs/CobotStatusMsg	54192
/Cobot/TalkerStatus	cobot_msgs/CobotTalkerMsg	86712
/Cobot/TaskPlannerStatus	cobot_msgs/CobotTaskPlannerMsg	41218
/Cobot/VectorLocalization/Gui	cobot_msgs/LidarDisplayMsg	57100

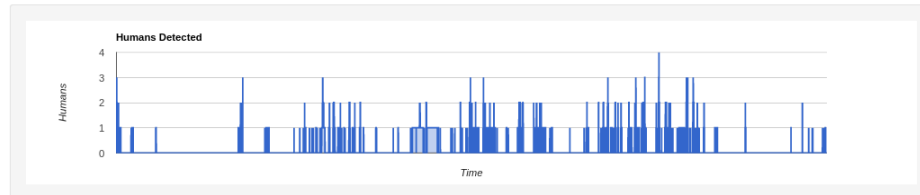
[illegible]

Figure A.4: Log Explorer website showing details of a single deployment log

Subversion, which we rely on, is that every subdirectory in the repository is a valid repository in itself, which permits user access control down to individual sub-parts of the repository.

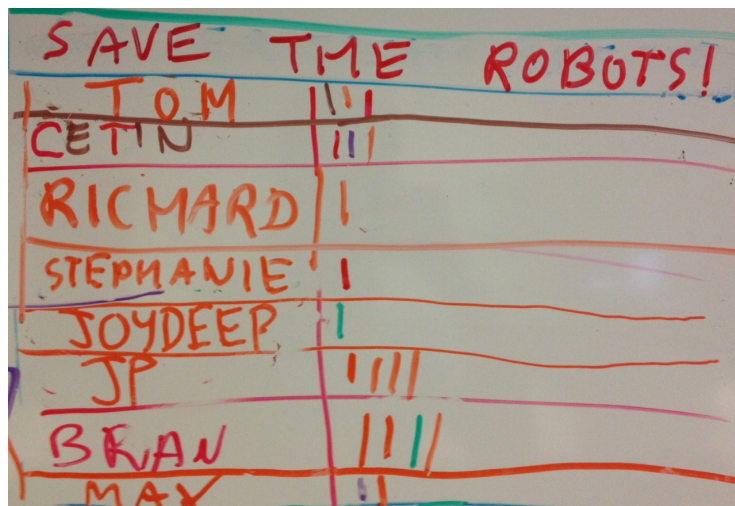
Redmine Interface In addition to Subversion, we use Redmine⁴ to browse through changesets committed to the repository. Redmine allows a web-browser based interface to inspect changesets, the committing developers, and visual diffs of the changes made. It also provides an interface to query changesets to specific subdirectories in the repository. This interface has been invaluable for tracing down origins of bugs and to perform code reviews. The Redmine interface additionally provides a wiki system for documentation, which we use to document the usage procedures of the different software blocks of CoBot, and also usage procedures for the hardware.

Development and Testing When new changes are made to the CoBot source code they are first tested in simulation. The simulator replaces the hardware drivers of the CoBots, but all the remaining software nodes, including localization and navigation, run exactly as if they were executing on the CoBot. CoBot developers use the simulator to verify that the robot can still successfully perform user-requested tasks as expected after changes are made to the source code.

The 5-Minute Demo Policy We strive to maintain a “5-Minute Demo” policy in the CoBot lab, where the robots should always be in a state to be started up for a live demo within 5 minutes’ warning. While it is understandable that the demo may not include the most experimental features, we still strive to be able to give a demo of the research of our last submitted publications. Having the 5-Minute Demo policy has allowed us to give demos to a large number of announced as well as unannounced visitors, and it also reinforces the confidence of the CoBot researchers in the deployability of the robots, thus making it easier for them to collect real world experimental data for their research with minimal setup time.

The Save The Robots Fund Despite the best efforts of the CoBot researchers, occasionally mistakes are made which result in longer down-times of the robots, be it due to discharged batteries, broken builds, or sensor misalignments. We keep a tally of the number of times each researcher is responsible for the downtime of the robot. After a researcher is responsible for five such incidents, he or she is expected to treat the remaining researchers to dinner. The tally is thus appropriately titled the “Save The Robots Fund”, and is shown in Figure A.5.

⁴<http://www.redmine.org/>



A handwritten ledger on lined paper titled "SAVE THE ROBOTS!". The ledger lists names in the left column and corresponding tally marks in the right column. The names are written in orange and blue ink, while the tally marks are in blue and green ink. The names listed are TOM, CETH, RICHARD, STEPHANIE, JOYDEEP, JP, BRIAN, and MAX.

SAVE THE ROBOTS!	
TOM	
CETH	
RICHARD	
STEPHANIE	
JOYDEEP	
JP	
BRIAN	
MAX	

Figure A.5: The Save The Robots Fund

Bibliography

- [1] S. Agarwal and K. Mierle. *Ceres Solver: Tutorial & Reference*. Google Inc. 73, 74
- [2] D. Anguelov, R. Biswas, D. Koller, B. Limketkai, and S. Thrun. Learning hierarchical object maps of non-stationary environments with mobile robots. In *Proceedings of the Eighteenth conference on Uncertainty in artificial intelligence*, pages 10–17. Morgan Kaufmann Publishers Inc., 2002. 20
- [3] P. Bahl and V. Padmanabhan. RADAR: An in-building RF-based user location and tracking system. In *IEEE infocom*, pages 775–784, 2000. 52
- [4] T. Bailey and H. Durrant-Whyte. Simultaneous localization and mapping (SLAM): Part II. *Robotics & Automation Magazine, IEEE*, 13(3):108–117, 2006. 14
- [5] J. Barton and L. Nackman. *Scientific and Engineering C++: an introduction with advanced techniques and examples*. Addison-Wesley Longman Publishing Co., 1994. 73
- [6] J. Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975. 69
- [7] P. Biber and T. Duckett. Dynamic maps for long-term operation of mobile service robots. In *Robotics: Science and Systems*, pages 17–24, 2005. 19, 24, 81
- [8] J. Biswas, B. Coltin, and M. Veloso. Corrective gradient refinement for mobile robot localization. In *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*, pages 73–78, Sept 2011. 22, 59
- [9] J. Biswas, J. P. Mendoza, D. Zhu, B. Choi, S. Klee, and M. Veloso. Opponent-driven planning and execution for pass, attack, and defense in a multi-robot soccer team. In *Proceedings of the 2014 international conference on Autonomous agents and multi-agent systems*, pages 493–500. International Foundation for Autonomous Agents and Multiagent Systems, 2014. 138
- [10] J. Biswas and M. Veloso. Wifi localization and navigation for autonomous indoor mobile robots. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages

- 4379–4384. IEEE, 2010. 59
- [11] J. Biswas and M. Veloso. Depth camera based indoor mobile robot localization and navigation. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 1697–1702. IEEE, 2012. 59
- [12] J. Biswas and M. Veloso. Planar polygon extraction and merging from depth images. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 3859–3864, Oct 2012. 89
- [13] J. Biswas and M. Veloso. Multi-sensor mobile robot localization for diverse environments. *RoboCup 2013: Robot Soccer World Cup XVII*, 2013. 59
- [14] J. Biswas and M. Veloso. Episodic non-markov localization: Reasoning about short-term and long-term features. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pages 3969–3974, May 2014. 22
- [15] J. Biswas and M. Veloso. Model-instance object mapping. *RoboCup 2014: Robot Soccer World Cup XVIII*, 2014. 98
- [16] R. Biswas, B. Limketkai, S. Sanner, and S. Thrun. Towards object mapping in non-stationary environments with mobile robots. In *Intelligent Robots and Systems, 2002. IEEE/RSJ International Conference on*, volume 1, pages 1014–1019. IEEE, 2002. 20
- [17] P. Bradley and U. Fayyad. Refining initial points for k-means clustering. In *Proceedings of the 15th International Conference on Machine Learning*, pages 91–99, 1998. 54
- [18] J. Buhmann, W. Burgard, A. Cremers, D. Fox, T. Hofmann, F. Schneider, J. Strikos, and S. Thrun. The mobile robot rhino. *AI Magazine*, 16(2):31, 1995. 21
- [19] T. Chan, G. Golub, and R. LeVeque. Updating formulae and a pairwise algorithm for computing sample variances. *COMPSTAT 1982. Pt. 1. Proceedings*, 5:30, 1982. 92
- [20] Y. Chen, J. Chiang, H. Chu, P. Huang, and A. Tsui. Sensor-assisted wi-fi indoor location system for adapting to environmental dynamics. In *Proceedings of the 8th ACM international symposium on Modeling, analysis and simulation of wireless and mobile systems*, pages 118–125, 2005. 52
- [21] Y. Chen, T. A. Davis, W. W. Hager, and S. Rajamanickam. Algorithm 887: Cholmod, supernodal sparse cholesky factorization and update/downdate. *ACM Transactions on Mathematical Software (TOMS)*, 35(3):22, 2008. 74
- [22] B. Choi, C. Meriçli, J. Biswas, and M. Veloso. Fast human detection for indoor mobile robots using depth images. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 1108–1113. IEEE, 2013. 110

- [23] K. Choo and D. Fleet. People tracking using hybrid Monte Carlo filtering. In *Proc. of Int. Conf. on Computer Vision (ICCV)*. IEEE, 2001. 13, 23
- [24] P. Cignoni, C. Montani, and R. Scopigno. A comparison of mesh simplification algorithms. *Computers & Graphics*, 22(1):37–54, 1998. 89
- [25] B. Coltin. *Multi-Agent Pickup and Delivery Planning with Transfers*. PhD thesis, The Robotics Institute, Carnegie Mellon University, USA, 2014. 5, 110
- [26] B. Coltin, J. Biswas, D. Pomerleau, and M. Veloso. Effective semi-autonomous telepresence. In *RoboCup 2011: Robot Soccer World Cup XV*, pages 365–376. Springer, 2012. 110
- [27] B. Coltin, M. M. Veloso, and R. Ventura. Dynamic user task scheduling for mobile robots. In *Automated Action Planning for Autonomous Mobile Robots*, 2011. 110
- [28] M. Cummins and P. Newman. FAB-MAP: Probabilistic localization and mapping in the space of appearance. *The International Journal of Robotics Research*, 27(6):647–665, 2008. 15
- [29] F. Dellaert, D. Fox, W. Burgard, and S. Thrun. Monte carlo localization for mobile robots. In *ICRA*, volume 2, pages 1322–1328, 1999. 13, 39, 63
- [30] F. Dellaert and M. Kaess. Square root sam: Simultaneous localization and mapping via square root information smoothing. *The International Journal of Robotics Research*, 25(12):1181–1203, 2006. 14
- [31] S. Duane, A. Kennedy, B. Pendleton, and D. Roweth. Hybrid monte carlo. *Physics Letters B*, 195(2):216 – 222, 1987. 13, 23
- [32] H. Durrant-Whyte and T. Bailey. Simultaneous localization and mapping: part I. *Robotics & Automation Magazine, IEEE*, 13(2):99–110, 2006. 14
- [33] A. Elfes. Using occupancy grids for mobile robot perception and navigation. *Computer*, 22(6):46–57, 1989. 34, 99
- [34] B. Ferris, D. Haehnel, and D. Fox. Gaussian processes for signal strength-based location estimation. In *Proceedings of Robotics: Science and Systems*, August 2006. 52, 59, 60
- [35] D. Fox. *Markov localization: A probabilistic framework for mobile robot localization and navigation*. PhD thesis, Dept. of Computer Science, University of Bonn, Germany, 1998. 3, 11, 13, 16, 24
- [36] D. Fox. KLD-sampling: Adaptive particle filters and mobile robot localization. *Advances in Neural Information Processing Systems (NIPS)*, 2001. 13, 21, 23

- [37] D. Fox, S. Thrun, W. Burgard, and F. Dellaert. Particle filters for mobile robot localization. *Sequential Monte Carlo methods in practice*, pages 499–516, 2001. 12
- [38] G. Gallagher, S. S. Srinivasa, J. A. Bagnell, and D. Ferguson. Gatmo: A generalized approach to tracking movable objects. In *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*, pages 2043–2048. IEEE, 2009. 21
- [39] R. L. Graham. An efficient algorithm for determining the convex hull of a finite planar set. *Information Processing Letters*, 1(4):132 – 133, 1972. 91
- [40] A. Griewank and A. Walther. *Evaluating derivatives: principles and techniques of algorithmic differentiation*. Society for Industrial and Applied Mathematics (SIAM), 2008. 73
- [41] G. Grisetti, C. Stachniss, and W. Burgard. Improved techniques for grid mapping with rao-blackwellized particle filters. *Robotics, IEEE Transactions on*, 23(1):34–46, 2007. 14, 23
- [42] G. Grisetti, C. Stachniss, and W. Burgard. Improved techniques for grid mapping with rao-blackwellized particle filters. *Robotics, IEEE Transactions on*, 23(1):34–46, 2007. 21
- [43] W. Hastings. Monte Carlo sampling methods using Markov chains and their applications. *Biometrika*, 57(1):97, 1970. 13
- [44] P. Henry, M. Krainin, E. Herbst, X. Ren, and D. Fox. RGB-D mapping: Using depth cameras for dense 3d modeling of indoor environments. In *the 12th International Symposium on Experimental Robotics*, 2010. 89
- [45] S. J. Julier and J. K. Uhlmann. Unscented filtering and nonlinear estimation. *Proceedings of the IEEE*, 92(3):401–422, 2004. 63
- [46] M. Kaess and F. Dellaert. Covariance recovery from a square root information matrix for data association. *Robotics and Autonomous Systems*, 57(12):1198–1210, 2009. 15
- [47] M. Kam, X. Zhu, and P. Kalata. Sensor fusion for mobile robot navigation. *Proceedings of the IEEE*, 85(1):108–119, 1997. 58
- [48] S. Koenig and R. Simmons. Xavier: A robot navigation architecture based on partially observable markov decision process models. *Artificial Intelligence Based Mobile Robotics: Case Studies of Successful Robot Systems*, pages 91–122, 1998. 13, 63
- [49] S. Koenig and R. Simmons. Xavier: A robot navigation architecture based on partially observable markov decision process models. In *Artificial Intelligence Based Mobile Robotics: Case Studies of Successful Robot Systems*, pages 91 – 122. MIT Press, 1998. 21
- [50] S. Lenser and M. Veloso. Sensor resetting localization for poorly modelled mobile robots.

- In *Int. Conf. on Robotics and Automation*, 2000. 13, 23
- [51] S. Lenser and M. Veloso. Sensor resetting localization for poorly modelled mobile robots. In *IEEE International Conference on Robotics and Automation*, pages 1225–1232, 2000. 54
- [52] J. Leonard, H. Durrant-Whyte, and I. Cox. Dynamic map building for an autonomous mobile robot. *The International Journal of Robotics Research*, 11(4):286–298, 1992. 13
- [53] B. Limketkai, L. Liao, and D. Fox. Relational object maps for mobile robots. In *IJCAI*, pages 1471–1476, 2005. 21
- [54] F. Lu and E. Milios. Robot pose estimation in unknown environments by matching 2d range scans. *Journal of intelligent & robotic systems*, 18(3):249–275, 1997. 14, 15
- [55] D. W. Marquardt. A method for the solution of certain nonlinear problems in least squares. *Quart. Appl. Math*, 2(2):164–168, 1944. 74
- [56] D. W. Marquardt. An algorithm for least-squares estimation of nonlinear parameters. *Journal of the Society for Industrial & Applied Mathematics*, 11(2):431–441, 1963. 74
- [57] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, E. Teller, et al. Equation of state calculations by fast computing machines. *The journal of chemical physics*, 21(6):1087, 1953. 13
- [58] D. Meyer-Delius, J. Hess, G. Grisetti, and W. Burgard. Temporary maps for robust localization in semi-static environments. In *IROS*, pages 5750–5755, 2010. 18, 24, 80
- [59] J. Modayil and B. Kuipers. Bootstrap learning for object discovery. In *Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, volume 1, pages 742–747. IEEE, 2004. 21
- [60] 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 *International Joint Conference on Artificial Intelligence*, volume 18. Citeseer, 2003. 14, 23
- [61] J. Neira and J. Tardós. Data association in stochastic mapping using the joint compatibility test. *Robotics and Automation, IEEE Transactions on*, 17(6):890–897, 2001. 15, 132
- [62] R. Newcombe and A. Davison. Live dense reconstruction with a single moving camera. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 1498–1505. IEEE, 2010. 89
- [63] R. A. Newcombe, A. J. Davison, S. Izadi, P. Kohli, O. Hilliges, J. Shotton, D. Molyneaux,

- S. Hodges, D. Kim, and A. Fitzgibbon. Kinectfusion: Real-time dense surface mapping and tracking. In *Mixed and augmented reality (ISMAR), 2011 10th IEEE international symposium on*, pages 127–136. IEEE, 2011. 89
- [64] N. Nilsson. Shakey the robot. Technical report, DTIC Document, 1984. 21
- [65] I. Nourbakhsh, C. Kunz, and T. Willeke. The mobot museum robot installations: A five year experiment. In *Intelligent Robots and Systems, 2003.(IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on*, volume 4, pages 3636–3641. IEEE, 2003. 21
- [66] M. Ocana, L. Bergasa, M. Sotelo, J. Nuevo, and R. Flores. Indoor Robot Localization System Using WiFi Signal Measure and Minimizing Calibration Effort. In *Proceedings of the IEEE International Symposium on Industrial Electronics*, pages 1545–1550, 2005. 52
- [67] E. Olson. Real-time correlative scan matching. In *ICRA*, pages 4387–4393, 2009. 14, 115
- [68] E. Olson, J. Leonard, and S. Teller. Fast iterative alignment of pose graphs with poor initial estimates. In *ICRA*, pages 2262–2269, 2006. 14, 78
- [69] A. Oyama, K. Konolige, S. Cousins, S. Chitta, K. Conley, and G. Bradski. Come on in, our community is wide open for robotics research! In *The 27th Annual conference of the Robotics Society of Japan*, volume 9, page 2009, 2009. 21
- [70] J. Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan Kaufmann, 1988. 12
- [71] M. Pitt and N. Shephard. Filtering via simulation: Auxiliary particle filters. *Journal of the American Statistical Association*, 1999. 13, 23
- [72] J. Rice. The algorithm selection problem. *Advances in Computers*, 15:65–118, 1976. 59
- [73] S. Rosenthal. *Human-Centered Planning for Effective Task Autonomy*. PhD thesis, Computer Science Department, Carnegie Mellon University, USA, 2012. 5
- [74] S. Rosenthal, J. Biswas, and M. Veloso. An effective personal mobile robot agent through symbiotic human-robot interaction. In *AAMAS 2010*, pages 915–922, 2010. 110
- [75] S. Rosenthal and M. M. Veloso. Mobile robot planning to seek help with spatially-situated tasks. In *AAAI*, volume 4, page 1, 2012. 110
- [76] S. Rusinkiewicz and M. Levoy. Efficient variants of the icp algorithm. In *3-D Digital Imaging and Modeling, 2001. Proceedings. Third International Conference on*, pages 145–152. IEEE, 2001. 91
- [77] R. B. Rusu. Semantic 3d object maps for everyday manipulation in human living environments. *KI-Künstliche Intelligenz*, 24(4):345–348, 2010. 99

- [78] J. Saarinen, H. Andreasson, and A. Lilienthal. Independent markov chain occupancy grid maps for representation of dynamic environment. In *IROS*, pages 3489–3495, 2012. 18, 24
- [79] D. Salmond, N. Gordon, and A. Smith. Novel approach to nonlinear/non-gaussian bayesian state estimation. In *IEE Proc. F, Radar and signal processing*, volume 140, 1993. 40
- [80] O. Serrano, J. Canas, V. Matellan, and L. Rodero. Robot localization using WiFi signal without intensity map. *WAF04, March*, 2004. 52
- [81] R. Siegwart, K. Arras, S. Bouabdallah, D. Burnier, G. Froidevaux, X. Greppin, B. Jensen, A. Lorotte, L. Mayor, and M. Meisser. Robox at expo. 02: A large-scale installation of personal robots. *Robotics and Autonomous Systems*, 42(3):203–222, 2003. 21
- [82] K. Smith-Miles. Cross-disciplinary perspectives on meta-learning for algorithm selection. *ACM Computing Surveys (CSUR)*, 41(1):1–25, 2008. 59
- [83] C. Stachniss and W. Burgard. Mobile robot mapping and localization in non-static environments. In *AAAI*, pages 1324–1329, 2005. 19
- [84] Y. Sun, B. Coltin, and M. Veloso. Interruptible autonomy: Towards dialog-based robot task management. In *Workshops at the Twenty-Seventh AAAI Conference on Artificial Intelligence*, 2013. 110
- [85] S. Thrun, M. Bennewitz, W. Burgard, A. Cremers, F. Dellaert, D. Fox, D. Hahnel, C. Rosenberg, N. Roy, J. Schulte, et al. Minerva: A second-generation museum tour-guide robot. In *Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on*, volume 3. IEEE, 1999. 21
- [86] S. Thrun, A. Bucken, W. Burgard, D. Fox, T. Frohlinghaus, D. Hennig, T. Hofmann, M. Krell, and T. Schimdt. Map learning and high-speed navigation in RHINO. *AI-based Mobile Robots: Case studies of successful robot systems*, 1998. 54
- [87] S. Thrun, W. Burgard, and D. Fox. *Probabilistic robotics*. MIT press Cambridge, MA, 2005. 44
- [88] S. Thrun and M. Montemerlo. The graph slam algorithm with applications to large-scale mapping of urban structures. *The International Journal of Robotics Research*, 25(5-6):403–429, 2006. 14
- [89] G. D. Tipaldi, D. Meyer-Delius, M. Beinhofer, and W. Burgard. Lifelong localization and dynamic map estimation in changing environments. In *RSS Workshop on Robots in Clutter*, 2012. 80, 81
- [90] T. van der Zant and T. Wisspeintner. Robocup@ home: Creating and benchmarking tomor-

- rows service robot applications. *Robotic Soccer*, pages 521–528, 2007. 21
- [91] A. Walcott-Bryant, M. Kaess, H. Johannsson, and J. Leonard. Dynamic pose graph slam: Long-term mapping in low dynamic environments. In *IROS*, pages 1871 –1878, 2012. 17, 24, 81
- [92] C. Wang and C. Thorpe. Simultaneous localization and mapping with detection and tracking of moving objects. In *Robotics and Automation, 2002. Proceedings. ICRA’02. IEEE International Conference on*, volume 3, pages 2918–2924. IEEE, 2002. 20, 24, 131
- [93] L. Xu, F. Hutter, H. Hoos, and K. Leyton-Brown. SATzilla: portfolio-based algorithm selection for SAT. *Journal of Artificial Intelligence Research*, 32(1):565–606, 2008. 59
- [94] L. Zhang and B. Ghosh. Line segment based map building and localization using 2D laser rangefinder. In *IEEE Int. Conf. on Robotics and Automation*, 2000. 34
- [95] S. Zickler and M. Veloso. RSS-Based Relative Localization and Tethering for Moving Robots in Unknown Environments. In *IEEE International Conference on Robotics and Automation*, 2010. 52