Map Learning and Coverage Planning for Robots in Large Unknown Environments

Grant P. Strimel

August 2014
CMU-CS-14-129

School of Computer Science
Computer Science Department
Carnegie Mellon University
Pittsburgh, PA 15317

Thesis Committee:
Manuela M. Veloso, *chair*
Avrim Blum

Submitted in partial fulfillment of the requirements for the degree of Master of Science of Computer Science

Keywords: Service Robots, SLAM, Robotic Coverage, Motion and Path Planning, Path Planning for Multiple Mobile Robots, Kinect Sensor Feature Extraction

ABSTRACT

The robotics field has seen indoor robots that are increasingly capable of accurately navigating in buildings and performing service tasks, such as cleaning and transporting items. Given the advances in accurate navigation and robust motion planning, large scale industrial applications become feasible tasks. Two common tasks are the mapping of large unknown structured spaces and using learned maps for coverage planning. In this thesis, methods are presented for robotic mapping of large spaces and coverage planning under finite energy constraints. The thesis is presented in two parts. Part I focuses on the mapping component. We present a basic Kinect-based Simultaneous Localization And Mapping (SLAM) system for CoBot (mobile service robot in CMU's Gates-Hillman Center) in predominantly planar environments. The SLAM solution is Kinect-based in the sense that observations come only from odometry measurements and three Kinect sensors. The system is designed for the motivating scenario of mapping a large room or floor with aisles and shelves for the purposes of a robot in a store or warehouse. We present our feature extraction techniques, describe the graph SLAM method used and show and compare SLAM results with and without parallel-orthogonal geometric constraints on the planar environment.

Part II addresses the coverage problem. The robot coverage problem, a common planning problem, consists of finding a motion path for the robot that passes over all points in a given area or space. In many robotic applications involving coverage, e.g., industrial cleaning, mine sweeping, and agricultural operations, the desired coverage area is large and of arbitrary layout. In this portion of the work, we address the real problem of planning for coverage when the robot has limited battery or fuel, which restricts the length of travel of the robot before needing to be serviced. We consider several alterations of the problem with varying objectives. We introduce new sweeping planning algorithms, which build upon the boustrophedon cellular decomposition coverage algorithm to include a fixed fuel or battery capacity of the robot. We show illustrative examples of the planned coverage outcome in a real building floor maps and run timed computational experiments for each of the methods.

Introduction

The robotics field has seen indoor robots that are increasingly capable of accurately navigating in buildings and performing service tasks, such as cleaning and transporting items. Given the advances in accurate navigation and robust motion planning, large scale industrial applications become feasible tasks. One such task is floor scrubbing in large indoor environments. Commercial buildings such as schools, airports, stores, and malls have their floors cleaned on a regular basis. Over a billion dollars a year is committed to these tasks. This fact naturally raises the question whether these scrubbing tasks can be automated and completed more efficiently using artificial intelligence and robots. Addressing the problem poses two interesting research challenges: the mapping of large unknown structured spaces and using learned maps for coverage planning. In this thesis, methods are presented for robotic mapping of large spaces and coverage planning under finite energy constraints. The thesis is presented in two parts. Part I focuses on the mapping component. We present a basic Kinect-based Simultaneous Localization And Mapping (SLAM) system for CoBot (mobile service robot in CMU's Gates-Hillman Center) in predominantly planar environments. The SLAM solution is Kinect-based in the sense that observations come only from odometry measurements and three Kinect sensors. The system is designed for the motivating scenario of mapping a large room or floor with aisles and shelves for the purposes of a robot in a store or warehouse. We present our feature extraction techniques, describe the graph SLAM method used and show and compare SLAM results with and without parallel-orthogonal geometric constraints on the planar environment.

Part II addresses the coverage problem. The robot coverage problem, a common planning problem, consists of finding a motion path for the robot that passes over all points in a given area or space. In many robotic applications involving coverage, e.g., industrial cleaning, mine sweeping, and agricultural operations, the desired coverage area is large and of arbitrary layout. In this portion of the work, we address the real problem of planning for coverage when the robot has limited battery or fuel, which restricts the length of travel of the robot before needing to be serviced. We consider several alterations of the problem with varying objectives. We introduce new sweeping planning algorithms, which build upon the boustrophedon cellular decomposition coverage algorithm to include a fixed fuel or battery capacity of the robot. We show illustrative examples of the planned coverage outcome in a real building floor maps and run timed computational experiments for each of the methods.

Part 1

SLAM for CoBot

1.1 Introduction

Carnegie Mellon's CORAL research group led by Professor Manuela Veloso works on autonomous indoor mobile robots named CoBot (Collaborative Robot). Currently, CoBot is able to navigate through the Gates-Hillman Center by non-markov localization using 3D depth camera information [BV12] [BV14]. The localization uses a blueprint map of each floor of the environment. With the map, it is able to combine sequences of odometry and Kinect depth camera observations to predict its position on the map with relatively high accuracy.

We address here a method to build maps for use in localization and navigation. We explore and implement a basic Kinect-based Simultaneous Localization And Mapping (SLAM) system for CoBot. The SLAM solution is be Kinect-based in the sense that observations come only from odometry and the Kinect camera.

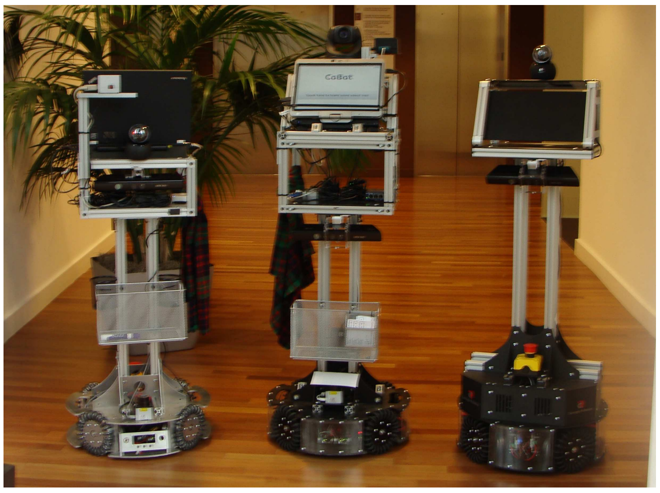


Fig. 1.1: CoBots 1, 2 and 4 (left to right) in the Gates Hillman Center at Carnegie Mellon University

1.1.1 Motivation

The motivating scenario for this thesis is the mapping of a large room or floor for the purposes of a service robot. For example, CoBot should be able to map the floor of a store or warehouse with many aisles and shelves. The mapping solution needs to map walls and shelves and ignore temporary insignificant features of the environment.

The SLAM solution proposed relies on identifying features in an environment and operates under the assumption of a zero-mean Gaussian error on the observations. The proposed algorithm can then solve for the maximum a posteriori (MAP) of the trajectory of the robot and the map of the environment which is in the form of a non-linear least squares optimization problem to minimize the reprojection error of the observations. The solution uses the Ceres Non-Linear Least Squares solver [AMO] to perform this optimization. The solution proposed is reasonable in the sense that it is capable of generating relatively accurate maps retaining only the major features of the motivating environments in an efficient amount of time.

1.1.2 Related work

The SLAM problem is one of the most well-studied problems in robotics. The goal of SLAM is to combine localization measurements to aid in mapping and concurrently use the mapping to aid in the localization. In most scenarios, a robot will have several types of sensors. One will have readings for location but give no information about the map. And others like range sensors will give data about landmarks in the map, but without any localization, these readings have little value. Both sensors have errors and lead to drift. The aim of SLAM is to combine the data and perform both processes at once to improve accuracy for both.

SLAM techniques usually come in one of two forms. The first is an online approach where the map and predicted locations grow with every observation. Most of these use the Extended Kalman Filter approach [GGR+05],[LHD07],[PJTN08],[WS05]. The method is able to incrementally estimate the "joint posterior distribution over the robot pose and landmark positions"[MT03]. From this procedure, accurate predictions on the updated pose and map landmarks can be made. There exists alternative online approaches like that of *FastSLAM* [MT03], [SEGL05] based on particle filters (each particle represents a pose of the robot) to incrementally solve the SLAM problem.

The second SLAM form is the batch problem where instead of incrementally solving the problem as more data is observed, all data and observations are gathered and stored and then all variables (localization and map) are solved for at once offline [FC04], [OLT06], [Thr06], [TBF05]. This approach is commonly referred to as *Graph SLAM* or *Graphical SLAM* because it is convenient to view the problem as a large graph where each vertex is a pose of the robot and constraint edges exist between the poses representing the observed measurements of consecutive frames. We take this second approach and solve the complete batch problem offline with our *graph SLAM-sd* algorithm. *Graph SLAM-sd* is much like the standard graph slam with a shelf detection (sd) component.

In many scenarios, such as the inside of offices or academic buildings, it is often appropriate to make a planar world assumption on the environment being explored (i.e. all landmarks are large planar surfaces) [GGR+05], [NHS07], [SRD06], [TIC12], [WS05]. We, too, make this assumption and solve only the two dimensional case. Methods which exploit this assumption are often referred to as *planar SLAM* or *line SLAM*.

We use both odometry observations and planar observations when solving SLAM unlike occasional techniques [PVP+09] when only large plane registration is used to correct the robot's pose. However, the only depth data we use comes from the relatively inexpensive Kinect sensors instead of the typical more expensive laser scanner.

1.2 Features

As typical with most SLAM solutions, the data we use to learn the map comes in two forms: **odometry features** and **landmark features**.

1.2.1 Odometry

CoBot is equipped with an omniwheel base. The omniwheel base of CoBot is composed of a set of four omniwheels arranged in a square formation. We have motion models which collect data from the actuators of each of the wheels. The data gives us the x, y, θ distance traveled by the robot at each timestep. The data measurements as well as the model are susceptible to noise, slippage, and other inaccuracies. Because data inaccuracies, relying on only odometry for localization for long stretches, will lead to large drift and dramatic error in estimating the global position of the robot. The tendency for dramatic drift is a fundamental motivation for the SLAM solution. SLAM uses the x, y, θ observation data from odometry for initial pose estimates in our SLAM solution.

1.2.2 Landmarks

The landmarks observed by CoBot at each timestep are those features which construct the map of the environment. For the purposes of this project, we have arranged three Kinect sensors on the robot base. The sensors are arranged to increase the total field of view of the robot so that the most features can be extracted. Each Kinect has a depth sensor which in turn produces point cloud objects as observations. With three Kinects working simultaneously (as opposed to just a single sensor), the point cloud observations are then combined into single point cloud for observation. This process requires a transform between each of the Kinects. Determining these transforms is part of a calibration procedure and can be expressed as a learning problem in itself. See Appendix B for the procedure on automatic calibration of the Kinects for the purposes of this project.

With the learned transforms from the calibration process, a single composite point cloud is observed at each timestep. The solution assumes the environment of the floor or store is a predominantly planar environment. Thus, the algorithm must be able to detect both walls (large planar surfaces) and shelves.

Plane Extraction

From each point cloud, we extract all large planar surfaces. We use the Fast Sampling Plane Filtering algorithm [BV12] to extract the planar surfaces from the data of the Kinect depth cameras. The method, which utilizes a local iterative RANSAC [FB81] approach, is efficient and robustly identifies the large planes at each timestep.

Once the planes are identified, they are filtered so that only those perpendicular to the ground plane remain (i.e. remove the floor, ceiling and table planes). The remaining planes are then projected onto the xy-plane. The exteme point observations of the projected plane define the line segment endpoints. See Figure 1.2.

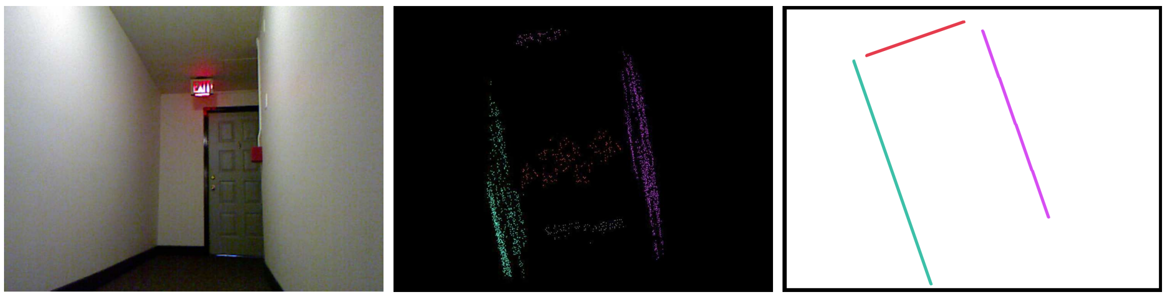
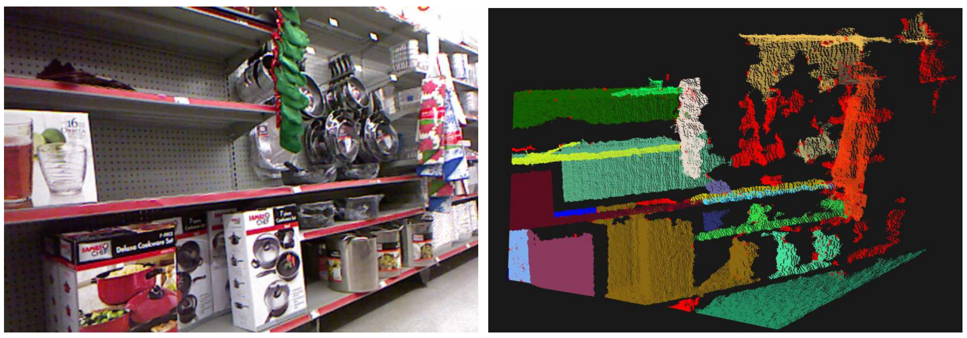


Fig. 1.2: RGB image of hallway (left); Identified planes from point cloud data (center); Filtered planes as line segments projected on xy-plane (right)

Shelf Detection

In addition to walls, for the purposes of our motivating example, there is a need to detect shelves in the same manner. When the shelves are full with many items (ex. boxes, books, etc), the plane extraction method described in Section 1.2.2 often is sufficient. However, when the shelves are sparse, it is less effective. Thus, we additionally apply a shelf detection technique based on a region growing RGBD segmentation. Our implementation is a variant based on the method presented in [RVvdH09].

After the initial segmentation, we keep only those segments with length $> \ell$ and width w within an ϵ error, where ℓ, w, and ϵ are pre-defined parameters for the particular shelves in the environment. After filtering, we check that several parallel segments are identified (i.e. greater than or equal to some n) before determining that shelves are actually present in the scene. Finally, as with plane extraction, we project the observed shelves onto the ground plane to get the observations in the form of 2D line segments.



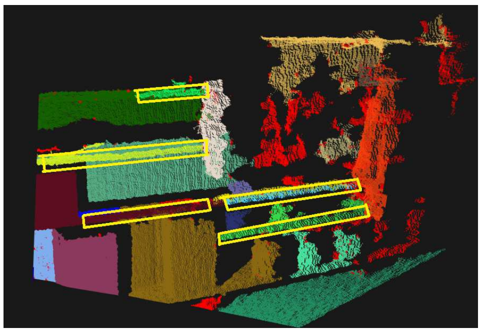


Fig. 1.3: RGB image of store shelves (top-left); RGBD segmented image (top-right); identified shelves (bottom)

1.3 Graph Slam-SD

Graph SLAM solves the full SLAM problem in the sense that it solves for the entire map and all robot poses at once. For this setting, it is often convention to represent the SLAM problem as a graph. Each vertex of the graph represents a pose of the robot defined by its x, y position and its orientation θ. Each vertex also contains observations taken by sensors at that pose. Edges in the graph come in two forms: *odometry edges* and *observation or correspondence edges.* Odometry edges connect consecutive poses by measured odometry. Correspondence edges connect observations of various vertices which are measurements of the same planar surface.

1.3.1 Notation

Figure 1.4 shows an example graph for a SLAM. The notation that will be used follows the conventions in [TBF05]

We define a pose at time instance t as x_t.

$$x_t = \begin{pmatrix} x \\ y \\ \theta \end{pmatrix}$$

x and y are the coordinates of the robot from the origin and θ is its orientation measure from the x-axis.

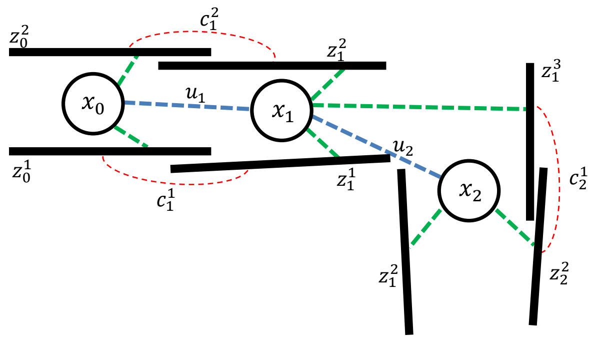


Fig. 1.4: Graph for SLAM problem. There are 3 robot poses each with 2-3 observations. Red edges show observation correspondences while blue edges show odometry correspondences.

We define u_t to be the measured distance between poses x_{t-1} and x_t.

$$u_t = \begin{pmatrix} x_t - x_{t-1} \\ y_t - y_{t-1} \\ \theta_t - \theta_{t-1} \end{pmatrix} = \begin{pmatrix} \Delta x \\ \Delta y \\ \Delta \theta \end{pmatrix}$$

The observations extracted from the Kinect data come in the form of 2D line segments. We define the i-th observation at timestep t as z_t^i which is defined by its two points.

$$z_t^i = \begin{pmatrix} p_{t,i,1} \\ p_{t,i,2} \end{pmatrix} = \begin{pmatrix} p_{t,i,1_x} \\ p_{t,i,1_y} \\ p_{t,i,2_x} \\ p_{t,i,2_y} \end{pmatrix}$$

We use m to represent the entire map of the environment. m is the planar description of the environment with which each observation corresponds:

$$m = \begin{pmatrix} m_1 & m_2 & \cdots & m_N \end{pmatrix}^\top$$

where each m_j is a landmark (planar surface/partial planar surface). Each m_j is again defined by two points $m_j = (p_1 \; p_2)^\top = \begin{pmatrix} p_{1_x} \; p_{1_y} \; p_{2_x} \; p_{2_y} \end{pmatrix}^\top$.

Each observation corresponds to a feature of the map. We define c_t^i to be the corresponding map feature index of z_t^i. Thus observation z_t^i corresponds with $m_{c_t^i}$.

Finally, it is useful to define the variable y to denote an augmented vector combining both pose variables x and and map m. Hence, $y_{0:t}$ will be the vector composed of both trajectory $x_{0:t}$ and map m through time t. At the same time, y_t is the vector composed of the current pose at time t and the map m.

$$y_{0:t} = \begin{pmatrix} x_0 & x_1 & \cdots & x_t & m \end{pmatrix}^\top \qquad \text{and} \qquad y_t = \begin{pmatrix} x_t & m \end{pmatrix}^\top$$

1.3.2 The Posterior

The slam problem involves optimizing the following posterior.

$$P(y_{0:t}|u_{1:t}, z_{1:t}, c_{1:t})$$

We can factor this posterior:

$$P(y_{0:t}|u_{1:t}, z_{1:t}, c_{1:t}) \quad = \quad \eta_1 \cdot P\left(z_t|y_{0:t}, u_{1:t}, z_{1:t-1}, c_{1:t}\right) \cdot P\left(y_{0:t}|u_{1:t}, z_{1:t-1}, c_{1:t}\right) \tag{1.1}$$

Equation (1) can be reduced further via the following observations:

$$P\left(z_t|y_{0:t}, u_{1:t}, z_{1:t-1}, c_{1:t}\right) = P\left(z_t|y_t, c_t\right) \tag{1.2}$$

since the observations at the t timestep (z_t) is independent of all states (poses, observations and correspondences) prior to t. The second probability in (1) can be broken down by splitting $y_{0:t}$ into the most recent state x_t and all prior augmented states $y_{0:t-1}$.

$$P\left(z_t|y_{0:t}, u_{1:t}, z_{1:t-1}, c_{1:t}\right) \quad = \quad P(x_t|x_{t-1}, u_t) \cdot P(y_{1:t-1}|z_{1:t-1}, u_{1:t-1}, c_{1:t-1}) \tag{1.3}$$

[TBF05]. As done in the derivation of (2), irrelevant variables were removed.

By substituting (2) and (3) back into (1), one arrives at the following recursive definition.

$$P(y_{0:t}|u_{1:t}, z_{1:t}, c_{1:t}) \quad = \quad \eta_1 \cdot P\left(z_t|y_t, c_t\right) \cdot P(x_t|x_{t-1}, u_t) \cdot P(y_{1:t-1}|z_{1:t-1}, u_{1:t-1}, c_{1:t})$$

With induction, one can obtain the closed form:

$$P(y_{0:t}|u_{1:t}, z_{1:t}, c_{1:t}) \quad = \quad \eta_2 \cdot P(x_0) \cdot \prod_t \left[P\left(x_t|x_{t-1}, u_t\right) \cdot \prod_i P(z_t^i|y_t, c_t^i) \right]$$

Note that $P(y_0)$ was split into $P(x_0)$ and $P(m)$ and since there is no knowledge about the map, $P(m)$ is consumed by η_2.

1.3.3 Gaussian Noise

As is typical, we assume that the noise and general inaccuracy from the odometry and sensors follows normal distribution probability density functions.

Keeping notation with [TBF05], it is typical to define a motion function for odometry. Recall that u_t is the measured odometry distances between the pose at $(t-1)$ and pose at t. We can define the motion function g like so.

$$g(u_t, x_{t-1}) = x_{t-1} + u_t$$

Thus, the model with Gaussian noise becomes

$$x_t = g(u_t, x_{t-1}) + \mathcal{N}(0, R)$$

where $R = \begin{pmatrix} \sigma_p^2 & 0 & 0 \\ 0 & \sigma_p^2 & 0 \\ 0 & 0 & \sigma_\omega^2 \end{pmatrix}$. Thus we have $\mathcal{N}(g(u_t, x_{t-1}), R)$

With this, we use the multivariate normal distribution PDF to determine $P(x_t|x_{t-1}, u_t)$.

$$P(x_t|x_{t-1},u_t) \quad = \quad \eta_3 \cdot \exp\left\{-\frac{1}{2}(x_t - g(u_t,x_{t-1}))^\top R^{-1}(x_t - g(u_t,x_{t-1}))\right\} \tag{1.4}$$

The observations extracted from the Kinect data come in the form of 2D line segments. The line segments observed z_t^i are defined by the two end points

$$z_t^i = \begin{pmatrix} p_{t,i,1} \\ p_{t,i,2} \end{pmatrix} = \begin{pmatrix} p_{t,i,1_x} \\ p_{t,i,1_y} \\ p_{t,i,2_x} \\ p_{t,i,2_y} \end{pmatrix}$$

For this SLAM project, we define the distance vector δ between line segment a and line segment b as the sum of the projected distances of the endpoints of a onto the line through b. See the Figure 1.5(a). This planar SLAM project assumes that the δ distances are distributed normally with variance σ_d^2. See 1.5(b).

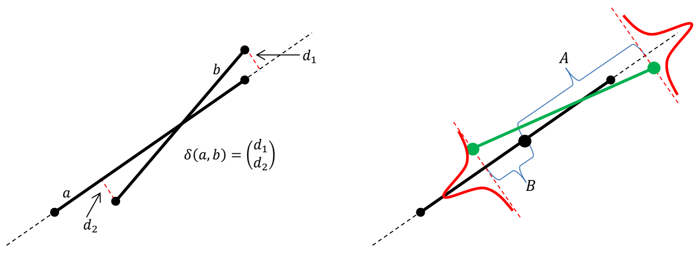


Fig. 1.5: Landmark distance metric δ (left); Gaussian observation noise (right)

$$z_t^i = \begin{pmatrix} \frac{x_2-x_1}{2} \\ \frac{y_2-y_1}{2} \\ \frac{x_2-x_1}{2} \\ \frac{y_2-y_1}{2} \end{pmatrix} + \begin{pmatrix} \frac{x_2-x_1}{||p_2-p_1||} \\ \frac{y_2-y_1}{||p_2-p_1||} \\ 0 \\ 0 \end{pmatrix} A - \begin{pmatrix} 0 \\ 0 \\ \frac{x_2-x_1}{||p_2-p_1||} \\ \frac{y_2-y_1}{||p_2-p_1||} \end{pmatrix} B + \begin{pmatrix} C_1\frac{y_1-y_2}{||p_2-p_1||} \\ C_1\frac{x_2-x_1}{||p_2-p_1||} \\ C_2\frac{y_1-y_2}{||p_2-p_1||} \\ C_2\frac{x_2-x_1}{||p_2-p_1||} \end{pmatrix}$$

$C = \begin{pmatrix} C_1 \\ C_2 \end{pmatrix} \sim \mathcal{N}(0, \begin{pmatrix} \sigma_d^2 & 0 \\ 0 & \sigma_d^2 \end{pmatrix})$ where $A, B \sim Uniform(0,k)$ for some positive finite constant k. Then defining $h(y_t, c_t^i) = m_{c_t^i}$ we have

$$P(z_t^i|y_t, c_t^i) \quad = \quad \eta_3 \cdot \exp\left\{-\frac{1}{2\sigma_2^2}\delta(z_t^i, h(y_t, c_t^i))^\top \delta(z_t^i, h(y_t, c_t^i))\right\}$$

*Note that our motion models and sensor experiments give us estimates σ_p, σ_ω, and σ_d.

1.3.4 Final Optimization

We want to maximize the probability of the model given the observations and correspondences under the conditions stated above. Thus, we wish to solve

$$\operatorname*{argmax}_{y_{0:t}} P(y_{0:t}|u_{1:t}, z_{1:t}, c_{1:t}) \quad = \quad \operatorname*{argmax}_{y_{0:t}} \left(\eta_2 \cdot P(x_0) \cdot \prod_t \left[P\left(x_t|x_{t-1}, u_t\right) \cdot \prod_i P(z_t^i|y_t, c_t^i)\right]\right)$$

Because the logorithm is a monatonically increasing function, we can apply it to the function being optimized without it changing its nature under optimization.

$$\begin{aligned} & \operatorname*{argmax}_{y_{0:t}} P(y_{0:t}|u_{1:t}, z_{1:t}, c_{1:t}) \\ = \; & \operatorname*{argmin}_{y_{0:t}}[\text{const.} + \sum_t \frac{1}{2}(x_t - g(u_t, x_{t-1}))^\top R^{-1}(x_t - g(u_t, x_{t-1})) \\ & \qquad + \sum_t \sum_i \frac{1}{2\sigma_2^2}\delta(z_t^i, h(y_t, c_t^i))^\top \delta(z_t^i, h(y_t, c_t^i))] \end{aligned} \tag{1.5}$$

Equation (5) is the non-linear least squares function to be optimized. We use the Ceres Non-Linear Least Squares solver [AMO] as an off-the-shelf optimization solution to solve this problem.

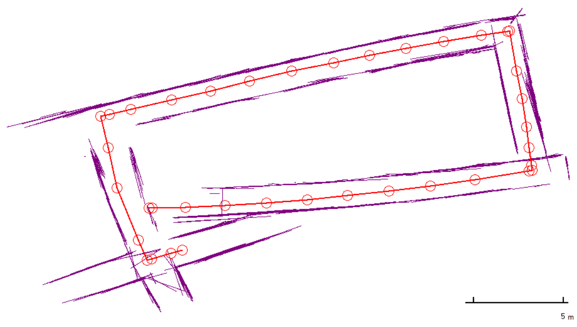
Parallel-Orthogonal Constraint

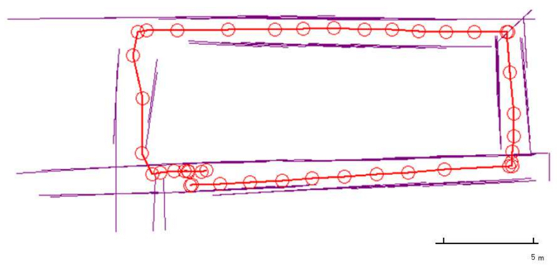
With the motivating example for this project as an indoor environment with many corridors, walls, and/or aisles, it is reasonable to make the assumption that all landmark line segments are parallel or perpendicular to each other. If we make this assumption, it is an additional "constraint" on the map generated from the procedure. All landmarks in the ground truth map are to be parallel to either the x-axis or y-axis. Adding this constraint to the optimization process is as simple as modifying the observational error function $\delta(a,b)$ to a function $\delta_{po}(a,b)$. $\delta_{po}(a,b)$ with line segments a and b is defined as the sum of the projected distances of b onto the line $y = \text{mid}_y(a)$ if a is angularly closer to the x-axis than the y-axis, otherwise $\delta_{po}(a,b)$ is the sum of the projected distances of b onto the line $x = \text{mid}_x(a)$. Here $\text{mid}_x(a)$ and $\text{mid}_y(a)$ are the x and y components of the midpoint of line segment a respectively. We compare the maps produced using both error metrics δ and δ_{po}.

1.4 Experimental Results

The graph-SLAM method described in this paper was implemented and tested on four separate data sets: a single hallway with the robot moving in a swervy path, a rectangular loop path in Wean Hall (WEH) Floor 8, a longer traversal of Wean Hall Floor 8, and seven aisles of a library. The robot was driven throughout the environment on each dataset via joystick and the appropriate data was collected and bagged. We also recorded the true global positions for start and finish of the robot path. The smallest data set (the hallway) contained 2705 poses and 3983 planar observations while the largest data set (the floor) contained 8912 poses and 20023 planar observations.

The SLAM algorithm was run both with and without the parallel-orthogonal constraint. The results of the rectangular loop data set is shown in Figure 1.6. Plots of the mapping relying only on dead-reckoning (odometry only), mapping using planar SLAM, mapping using planar SLAM with the parallel-orthogonal contraint, and the ground truth map of the environment are shown. Similar figures for the remaining data sets are shown in Figures 2.22, 2.18, 2.19 (Appendix A).





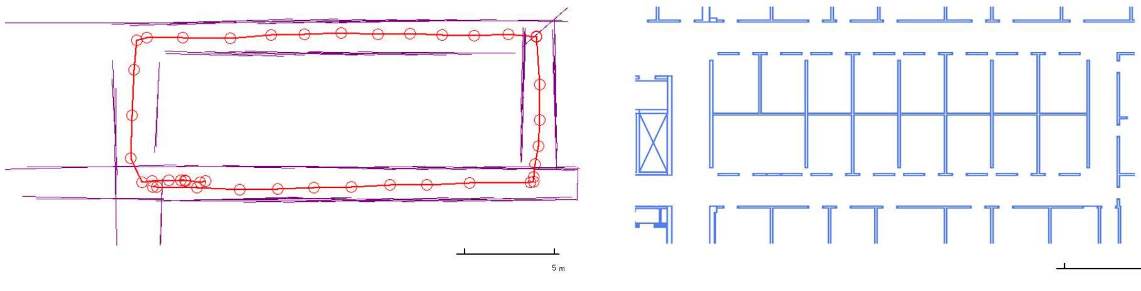


Fig. 1.6: **WEH Loop Data Set**: raw data map produced using odometry only (top-left); planar SLAM map (top-right); planar SLAM map [parallel-orthogonal constraint] (bottom-left); ground truth map (bottom-right)

A qualitative analysis shows us that our SLAM solution significantly reduces the drift of the robot's navigation path and the resulting maps are dramatically closer to the ground truth. This drift is most dramatic in the largest dataset (Figure 2.22 WEH Hall) where relying simply on dead reckoning leads to a final pose prediction that is approximately 37 m away from the true final pose in global coordinates while the SLAM solution reduces this error to only 1.5 m.

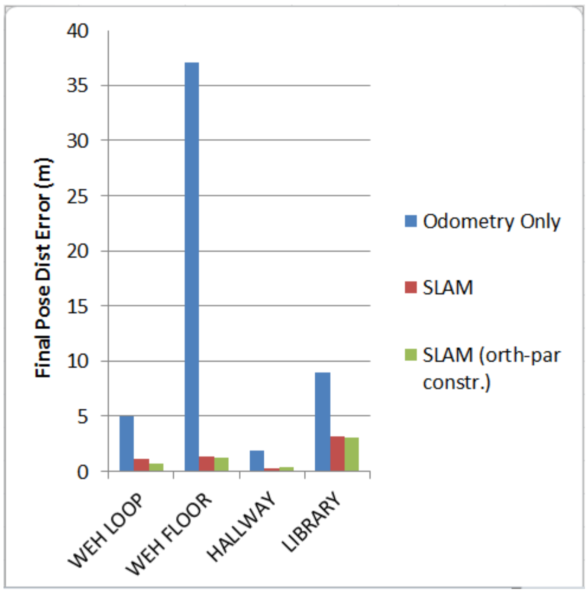


Fig. 1.7: Final robot position distance errors

We also note that while some minor effects can be seen by adding the parallel-orthogonal constraint, a major improvement cannot be concluded and any corrections that the constraint may add are highly dependent on the data set collected. The final position errors (measured by euclidean distance from the recorded global position) of dead-reckoning vs. SLAM vs. SLAM (with constraints) for the four datasets are shown in Figure 1.7. On average there is only a 3% percent decrease in overall error by applying the parallel-orthogonal constraint. In one case, the Hallway Dataset (smallest map), the error actually increased by adding the constraint.

From the maps generated by the graphical SLAM approach implemented here, we find our solution is general and able to reconstruct reasonable maps to use for future navigation. We attribute this relative robustness to the method's simplicity and direct approach to solving the problem. Though there exists more advanced and complete methods, for our needs, graphical SLAM demonstrates sufficiency for the motivating scenario. The design of the mapping system as an offline approach to mapping and unknown indoor environment proves adequate.

1.5 Conclusion

In this work, we explored graphical SLAM for CoBot. We proposed, implemented, and tested a straight-forward offline graph SLAM solution called *graph slam-sd* for the motivating scenario of mapping a large room or floor with aisles and shelves for the purposes of a service robot in a store. We gave two methods for detecting and measuring landmarks of walls and shelves and used them as features when solving the problem. They are sufficient for the presented scenario. We implemented the solution with the efficient Ceres solver.

We collected four data sets of varying sizes and tested our methods on each. We find that the graphical planar SLAM method can be successful and relatively accurate when mapping a route of corridors or aisles with shelves. The solution leads to dramatic reductions in navigation and mapping errors and yields maps which are close to ground truth. Additionally, we analyzed the effects of adding a parallel-orthogonal constraint to the optimization and conclude that adding the constraints can lead to minor corrections for orientational errors of the map but overall yields little significant improvements.

There are several possible directions for future work for this CoBot mapping project. Future work includes incorporating additional data sources such as vision or wifi data to aid in the localization and mapping. Also, it would be interesting to add a form of loop closure to the SLAM project and compare the results to those achieved already.

Using an advanced automatic loop closure technique or something as simple as placing select QR codes in known locations in the environment are possibilities. Though not directly addressed in this report, the time to perform the optimization and solve for the larger maps is significant. Several optimizations including heuristics and compressions should be explored to reduce the amount of computation required.

Part 2

Methods for Coverage Planning with Finite Resources

2.1 Introduction

We have extensively experienced mobile indoor robots that are capable of accurately navigating in buildings and performing service tasks, such as transporting items or accompanying people to locations [BV13]. Given their accurate localization and navigation and their reliable motion planning, we investigate their service to further include a complete sweeping task. In this work, we address a robot space coverage problem.

Coverage planning has been commonly studied in robotics (e.g., [Cho01]). The goal is to plan a path in which the robot covers all points in a given map, i.e., the robot's work space. Many approaches have been explored for a variety of applications, including item search [SXS$^+$00a], floor cleaning [DH93], large scale agriculture [FSTC01], mowing and milling [AM00], and painting [?]. In all of these applications, it is essential that the robot path or sensor paths are guaranteed to cover the surface in a robust and efficient manner to complete the objective.

These tasks inspire various methods in addressing the coverage problem. Techniques used can be categorized by how they address altering objectives. Some works only require the robot's sensors to cover an area while many force the robot base to pass over the entire region [ACZS03]. There is much work on randomized approaches without knowledge of the environment [Gag93] as well as approaches with a predefined map like those which use cellular decomposition [Lat91, CARL00, CP97].

Often these approaches are motivated to minimize some objective. A common choice is to minimize the total distance traveled during the cover of the area. However, finding the optimal route in this regard is an NP-Hard problem. This can be seen by its close relation to solving the geometric Traveling Salesman Problem (TSP) with neighborhoods [AH94]. Hence, approximation heuristics are often used. Even through a cellular decomposition where the problem is broken down into smaller regions, planning an efficient tour through the regions requires some form of approximation. Other optimizations like minimizing the number of turns required in a decomposition have also been studied [HY01] [YKPS14].

In this work, motivated by our real robot, we investigate an additional component to the coverage problems by incorporating a consideration for a fixed battery or fuel source. Accounting for the limited battery life is important as, in many applications, the area to cover is too large for the robot to completely cover in a single non-interrupted charge of a battery. Coverage planning with energy constraints and timing restrictions has been addressed in previous works [SKPY10] [YKPS14] [HL06]. Of these works, some consider the problem of sensor-based multi-robot coverage in narrow environments and use a heuristic algorithm to reduce the number of robots needed under energy constraints [SKPY10] [YKPS14]. Another scenario addressed is a multi-robot deployment problem to determine the number of groups unloaded by a carrier, the number of robots in each group and the initial locations of those robots for coverage tasks under both timing and energy constraints [HL06].

We focus here on coverage planning under several objectives all with fixed energy capacity. We present both heuristic algorithms and integer programming to reduce total distances traveled during coverage. To investigate this problem, we also assume that the space has a service For our heuristic method, we contribute a new *battery-constrained* sweep algorithm (*BC Sweep*) which extends the boustrophedon cellular decomposition coverage algorithm to reason about a battery capacity constraint. We then present integer linear programming formulations to the problem to solve coverage routing solutions optimally.

2.2 Problem Definitions

There has been limited prior work mentioned for planning an entire path for the robot that considers the total distance the robot travels or total energy used in relation to a fixed battery life or fuel capacity. We consider this extra constraint during path planning coverage.

In all of the problem variations addressed here, we make the assumption that given some finite path for a robot, an estimate can be made on the energy used while executing the path. We denote this estimation function f. We now define three variations of the problem definition for the coverage scenario addressed in this work.

1. **Single Robot, Single Depot Coverage (SRSD)**
   The most basic of the coverage variations is the single robot, single depot (SRSD) case. The problem formulation can be stated as follows: Given a two dimensional map M with a service center location s_0, a fuel capacity λ, and a fuel consumption function f, plan a route such that the robot covers the map and respects the robot's fuel capacity constraint. The robot begins and ends at s_0 and returns to s_0 to refuel. M is represented as a closed figure (typically a polygon) and will usually have obstacles. See Figure 2.1.

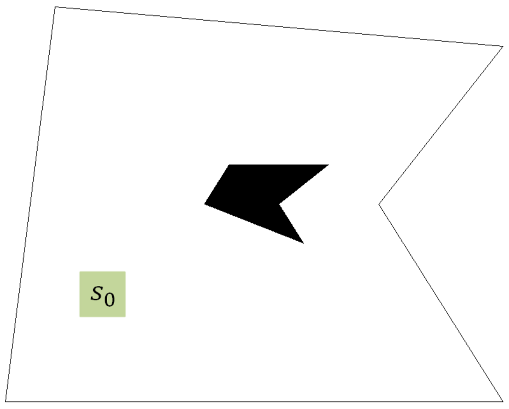


Fig. 2.1: An example polygonal map M with a single obstacle (black) and a single service station (green).

2. **Single Robot, Multi-Depot Coverage (SRMD)**
   The single robot, multi-depot coverage (SRMD) scenario is identical to the SRSD problem description except that there are several service stations present represented by a depot set D. The robot must originate from s_0 and may finish at any other service station. Under both SRSD and SRMD, the objective is to find a feasible solution which reduces the total distance traveled.

3. **Multi-Robot, Single Depot Coverage (MRSD)**
   The multi-robot, single depot coverage (MRSD) has a variation in objective from SRSD and SRMD. The ultimate goal is to achieve complete coverage, however, we now have a fleet of robots. The robots are identical robots and originate from a

single source depot and then perform coverage in parallel. Each robot has a fixed capacity but recharges are not permitted. The objective here is minimizing the total of robots needed.

2.3 Boustrophedon Coverage

Our algorithms make use of the boustrophedon cellular decomposition [CARL00] [CP97] for bounded planar environments with obstacles. This decomposition breaks the map into disjoint regions called *cells*. The individual cells can be covered simply by back-and-forth or "ox-plow" motions. See Figure 2.2. To cover the entire free space or map, a tour through each region is constructed and the robot visits and covers the cells sequentially along this tour.

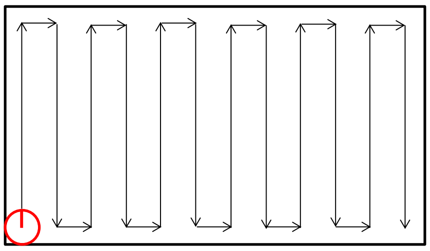
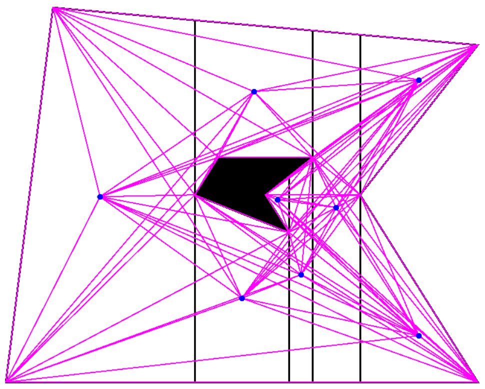
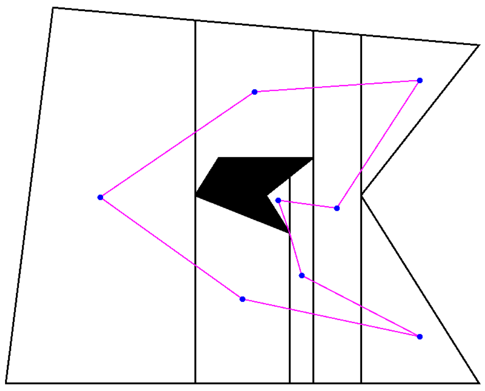


Fig. 2.2: Back-and-forth ox-plow motions.

Specifically, the boustrophedon decomposition uses a verticle line sweep approach to construct the cells. The *slice* sweeps from left to right across the map. At any point, if the continuity of the sweep line changes count, then a new cell is spawned or two adjacent cells are merged. In the case of connectivity increasing, a new cell is added. For the instances when connectivity decreases, then adjacent cells are merged [CARL00] [CP97].

After decomposition, the algorithm constructs some graph (commonly a complete or adjacency graph) between the regions. The weight on an edge is the euclidean shortest distance path between the two points on the map. To determine the shortest paths on a map between all cells, a *visibility graph* is constructed where obstacles are expanded by the robot radius [LpW79]. The visibility graph is simply a graph where an edge exists between two points of interest if the robot can travel between by straight line with touching an obstacle. Figure 2.3a shows the visibility graph for our example.


(a) Visibility Graph


(b) TSP tour through cells

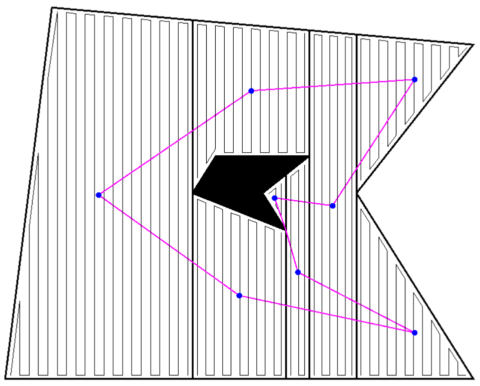

(c) Cell cover paths with tour

Fig. 2.3: Components of boustrophedon coverage

Diijkstra's algorithm is then run on this graph for all pairs of cells to determine the weight of the shortest distance between them. This gives us our complete shortest path graph.

With the complete graph, the goal becomes finding a minimal cost tour through all regions and reduces to solving the TSP on the graph. Typically an approximation algorithm is used as a heuristic to solve for a reasonable tour. Figure 2.3b shows a TSP tour through the cells and 2.3c shows the complete cover plan.

Our algorithms make use of this decomposition technique to create cellular regions and extends touring the regions to account for a fixed fuel or battery life.

2.4 Approximate Solutions

As discussed in the previous section, determining a minimum route through the cell graph created by the boustrophedon decomposition requires solving or approximating the traveling salesman problem. As an extension to the decomposition algorithm, we wish to solve for a minimum route such that the robot (s) abides by the fuel capacity constraint. This problem becomes more difficult than TSP. We begin with an approximation algorithm. We present now the *BC Sweep* algorithm.

2.4.1 BC Sweep for SRSD

Again, we presume that given some route r of the robot that we have a function f such that $f(r)$ is an estimation of the energy used over that route. Often, this is directly related to the length of r and could also incorporate the turns (could be expensive). f must also be linear (ie $f(r_1 \to r_2) = f(r_1)+f(r^*_{12})+f(r_2)$) where r^*_{12} is the shortest direct route connecting r_1 to r_2. We present now the *BC Sweep* algorithm. The intuition of *BC Sweep* is straight forward. We construct a graph expressing the cellular decomposition of the space with a refueling location. The representation accounts for the costs of covering each cell and traveling between them. The algorithm then simply approximates a minimum cost walk through the graph which circles back to the service station when necessary to refuel. Our approach needs the minimum requirement that the fuel capacity λ is large enough that the robot can depart from s_0, cover any cell, and return without running out of fuel. Under these conditions, we have the following algorithm.

BC Sweep:

1. *Decomposition*: Perform boustrophedon decomposition on the map M into cell set X. Plan all back-and-forth cover-paths $\{r_1, ..., r_n\}$ for respective cells $\{x_1, ..., x_n\}$. For simplicity, we assume all cover-paths begin and end in the same location. Add a special cell s_0 of 0 size and a null cover-path r_0 representing the service station s_0.

2. *Graph Construction*: Construct a complete graph $G = (V, E)$ between all cells including the service station s_0. Thus $V = \{s_0\} \cup \{x_1, ..., x_n\}$. For every edge $e_{ij} = (v_i, v_j) \in E$ let $r^*_{ij} = r^*_{ji}$ be the shortest direct route on M between cell (or station) i and j. Assign the weight $w(e_{ij}) = f(r^*_{ij}) + \frac{1}{2}(f(r_i) + f(r_j))$ to each edge.

This procedure is shown in Figure 2.4.

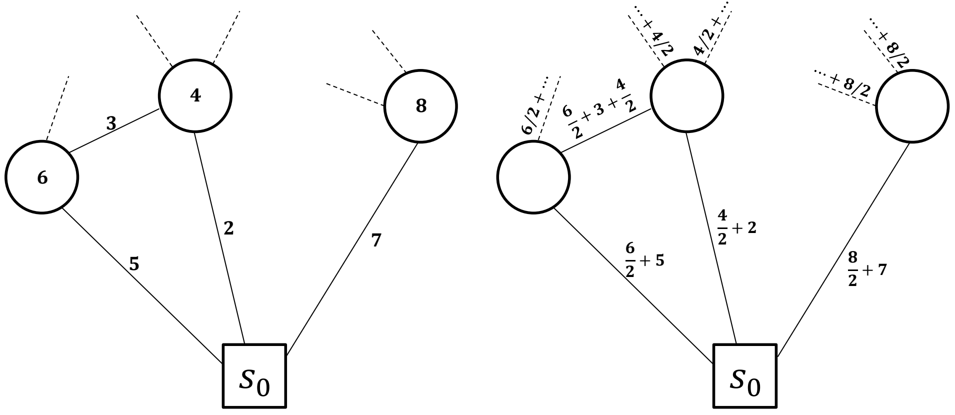


Fig. 2.4: *(Left)* graph cells (nodes) with intra and inter cell fuel costs. *(Right)* modified graph G of *BC Sweep* algorithm

3. *Solve for Giant Tour*: Use Christofides Algorithm [Chr76] to generate a TSP tour $T = v_0^T \to v_1^T \to \cdots \to v_n^T \to v_0^T$ starting and ending at s_0 (ie $s_0 = v_0^T$).

4. *Tour Partitioning*: We now *optimally* partition T into subroutes which meet the fuel capacity constraint. Define a cost matrix C as follows. $\forall\ i, j \in \{0, ..., n\}$

$$C_{ij} = \begin{cases} f(v_0^T \to v_{i+1}^T \to \cdots \to v_j^T \to v_0^T) & \text{if this cost} \\ & \text{is } \leq \lambda \text{ and } i < j \\ \infty & \text{otherwise} \end{cases}$$

   This cost matrix defines a new directed graph H. We now use Dijkstra's algorithm on H to find the shortest path from *node* 0 to *node* n. Since each edge of this path represents a subroute, we append these subroutes together to get our tour. This gives us the shortest route using T which abides by the battery capacity constraint.

The *BC Sweep* steps are shown in Figure 2.5.

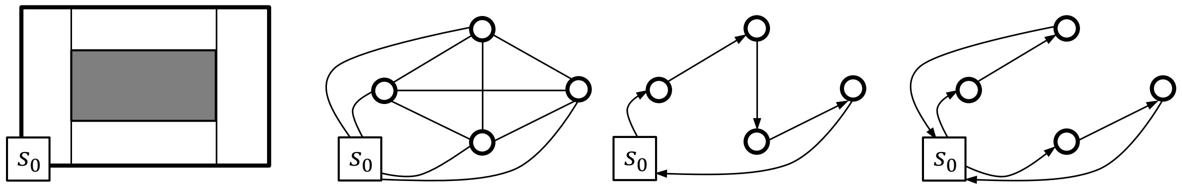


Fig. 2.5: Steps 1-4 (*left to right*) of the *BC Sweep* Algorithm

Steps 3) and 4) can be seen as an approximation of a reduction to the *distance constrained vehicle routing problem* (DCVRP). Vehicle routing with constraints is a variant of TSP and has been studied in several works including [Bea83] [LSL90] [LSLD92] [NR]. The heuristic used here is described and analyzed in [Bea83] and [LSLD92].

We present the following theorem on completeness and correctness.

Theorem 1. *BC Sweep covers M and obeys the fuel capacity constraint.*

Proof. By boustrophedon decomposition, if each cell is visited, it will be covered. We show that each cell is visited and obeys the fuel constraint. When G is constructed, we add half of every cell's covering fuel cost to all incident edges of that cell. See Figure 2.4. Hence any path which passes through the cell will pick up half the weight on the way in and the other half on the way out. Because of this set-up, our TSP tour T accounts for all cell costs. With all fuel costs accounted for in T and H giving infinite weight to any subroutes violating the fuel constraint, our final route abides by the constraint. Note because we assumed a feasible solution exists, a finite cost path will always be possible. Since our final route is the concatenation of adjacent subroutes beginning and ending at s_0, the route visits all cells. $\square$

2.4.2 Extensions for SRMD

Recall that the problem formulation for the single robot, multi-depot setting is the same as SRSD except that we consider the ability of the robot to refuel at several refueling stations. The robot must originate from s_0 and may finish at any service station in our depot set $\{s_0, s_1, \ldots, s_{m-1}\} = D$ of size m. We extend *BC Sweep* to handle this scenario by modifying a few of the steps.

1. *Decomposition*: Again, perform boustrophedon decomposition on the map M into cell set X. Plan all back-and-forth cover-paths $\{r_1, ..., r_n\}$ for respective cells

$\{x_1, ..., x_n\}$. Add a special cells $\{s_0, s_1, \ldots, s_{m-1}\}$ of 0 size and a null cover-path r_0 representing the service stations.

2. *Graph Construction*: Construct a complete graph $G = (V, E)$ between all cells including the service stations in D. Thus $V = D \cup \{x_1, ..., x_n\}$. As before, for every edge $e_{ij} = (v_i, v_j) \in E$ let $r^*_{ij} = r^*_{ji}$ be the shortest direct route on M between cell (or station) i and j. Assign the weight $w(e_{ij}) = f(r^*_{ij}) + \frac{1}{2}(f(r_i) + f(r_j))$ to each edge.

3. *Solve for Giant Tour*: Generate an approximate TSP tour $v^T_0 \rightarrow v^T_1 \rightarrow \cdots \rightarrow v^T_n \rightarrow v^T_0$ starting and ending at v_0 (ie $s_0 = v^T_0$). Truncate the tour of the service station $T = v^T_1 \rightarrow \cdots \rightarrow v^T_n$. The important component is the trail through the cell vertices. See Figure 2.6.

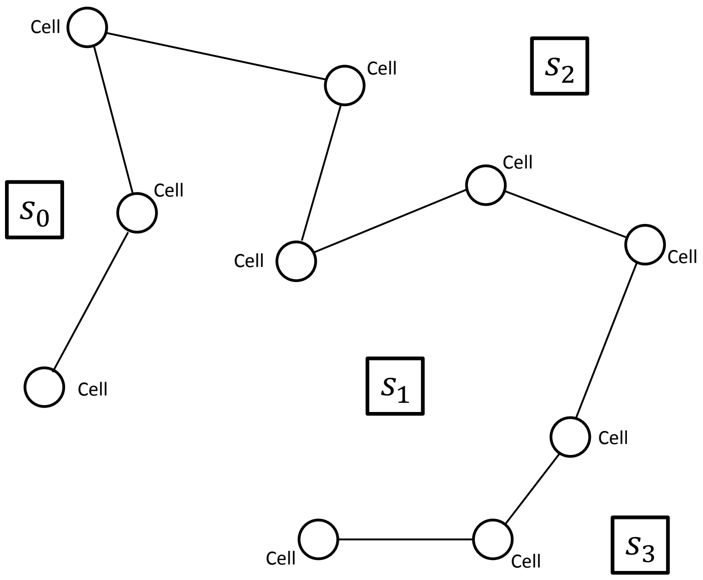


Fig. 2.6: Giant tour through cell vertices in a multi-service station environment.

4. *Tour Partitioning*: We now *optimally* partition T into subroutes which meet the fuel capacity constraint. Define a cost matrix C as follows. $\forall i, j \in \{0, \ldots, (n +$

$1)m\}$ if we let

$$i' = i \bmod (n+1)$$
$$j' = j \bmod (n+1)$$
$$p = \lfloor i/(n+1) \rfloor$$
$$q = \lfloor j/(n+1) \rfloor$$

$$C_{ij} = \begin{cases} f(s_p \to v^T_{i'+1} \to \cdots \to v^T_{j'} \to s_q) & \text{if this cost is } \leq \lambda \text{ and } i' < j' \\ \infty & \text{otherwise} \end{cases}$$

This cost matrix defines a new directed graph H. We now use Dijkstra's algorithm m times on H to find the minimum cost path of all shortest paths from *node* 0 to *node* $(k(n+1)+n)$ for all $k \in \{0, \ldots, m-1\}$. Since each edge of this path represents a subroute from a depot to depot, we append these subroutes together to get our tour. We take the minimum over the m instances of Dijkstra's algorithm to determine the best depot at which to finish. This procedure gives us the shortest route using T which abides by the battery capacity constraint.

2.4.3 BC Sweep for MRSD

The *BC Sweep* heuristic algorithm need not be modified for the multi-robot, single depot setting. Each depot-to-depot subtour of the partitioned path becomes the entire path for an individual robot of the multiple robots deployed. The number of robots needed becomes the number of partitions used by *BC Sweep*.

2.4.4 Performance Analysis

We now show results on the performance of this optimal tour partitioning heuristic with respect to the objectives in each scenario. The results and proofs were originally presented in 1992 [LSLD92]. We will use a variant of their notation and proofs when giving the results here.

Bounds for the *BC Sweep* Heuristic

Since the *BC Sweep* algorithm is the same for both the SRSD and MRSD cases, we begin with their performance bounds. For convenience, we define the following variable relations.

T - the TSP tour through all cells that is partitioned by *BC Sweep*

S - the complete route through all the cells produced by *BC Sweep*

k - the number of subroutes of S starting and ending at a service station

$L = f(S)$ - the total length of our route S

L^* - the minimum (optimal) possible length of a tour possible over all feasible routes

k^* - the minimum (optimal) possible number of subtours possible over all feasible routes

$s_{p\{v\}}$ - the closest service station vertex to node v in G

$d_m = \max\{w(e_{ij})\}$ where e_{ij} are edges in G where $i \in D, j \in L$

Lemma 2. *$L \leq k\lambda$.*

Proof. L is composed of k subtours each of which must be less than the fuel capacity λ thus the total length must be $\leq k\lambda$. □

Theorem 3. *$k \leq \min\{n, \frac{w(T)-2d_m}{\lambda-2d_m} + 1\}$.*

Proof. Consider a greedy tour partitioning of T instead of the optimal tour partitioning. For the greedy approach, we go to the nearest service station to refuel only when we can go no further without violating the capacity constraint. Let the greedy partitions (without detours to service stations) of T be $\{T_1, T_2, \ldots, T_{k^G}\}$ where k^G is the numberof partitions used and

$$\begin{aligned}
&T_1 = s_0 - v_1^T - v_2^T - \cdots - v_{\ell(1)}^T \\
&T_2 = v_{\ell(1)+1}^T - v_{\ell(1)+2}^T - \cdots - v_{\ell(2)}^T \\
&\vdots \\
&T_{k^G} = v_{\ell(k^G-1)+1}^T - v_{\ell(k^G-1)+2}^T - \cdots - v_{\ell(k^G)}^T
\end{aligned}$$

Also let $w(T_i) = w(v_{\ell(i-1)+1}^T - v_{\ell(i-1)+2}^T - \cdots - v_{\ell(i)}^T)$. Thus we have

$$w(T_1) + w(v^T_{\ell(1)} - v^T_{\ell(1)+1}) + w(v^T_{\ell(1)+1} - s_{p\{v^T_{\ell(1)+1}\}}) \geq \lambda \tag{2.1}$$

$$w(s_{p\{v_{\ell(i-1)}\}} - v^T_{\ell(i-1)+1}) + w(T_i) + w(v^T_{\ell(i)} - v^T_{\ell(i)+1}) + w(v^T_{\ell(i)+1} - s_{p\{v_{\ell(i)}\}}) \geq \lambda$$
$$\forall i \text{ s.t. } 1 < i < k^G \tag{2.2}$$

$$w(T_{k^G}) - w(s_{p\{v_{\ell(k^G-1)}\}} - v^T_{\ell(k^G-1)+1}) \geq 0 \text{ by triangle inequality} \tag{2.3}$$

Using inequalities (1), (2) and (3), we can sum the inequalities over all i from 1 to k^G:

$$\begin{aligned}
& w(T_1) + w(v^T_{\ell(1)} - v^T_{\ell(1)+1}) + w(v^T_{\ell(1)+1} - s_{p\{v^T_{\ell(1)+1}\}}) + \\
& \sum_{i=2}^{k^G-1} \Big[w(s_{p\{v_{\ell(i-1)}\}} - v^T_{\ell(i-1)+1}) + w(T_i) + w(v^T_{\ell(i)} - v^T_{\ell(i)+1}) + w(v^T_{\ell(i)+1} - s_{p\{v_{\ell(i)}\}}) \Big] + \\
& w(T_{k^G}) - w(s_{p\{v_{\ell(k^G-1)}\}} - v^T_{\ell(k^G-1)+1}) \geq (k-1)\lambda \\
\implies (k^G - 1)\lambda & \leq \sum_{i=1}^{k^G} w(T_i) + \sum_{i=1}^{k^G-1} w(v^T_{\ell(i)} - v^T_{\ell(i)+1}) + 2\sum_{i=2}^{k^G-2} d_m \\
& = w(T) + 2\sum_{i=2}^{k^G} d_m \\
& \leq w(T) + 2(k^G - 2)d_m
\end{aligned}$$

We simply rearrange $(k-1)\lambda \leq w(T) + 2(k^G - 2)d_m$ to arrive at $k^G \leq \frac{w(T)-2d_m}{\lambda - 2d_m} + 1$. Since our optimal tour partitioning heuristic will always yield at most as many subtours as the greedy partition (because of triangle inequality), we have $k \leq k^G$. Thus, we can simply conclude that $k \leq \min\{n, \frac{w(T)-2d_m}{\lambda-2d_m} + 1\}$.

$\square$

Theorem 4. $\frac{L}{L^*} \leq 1 + (1.5)\frac{\lambda/d_m}{\lambda/d_m - 2}$

Proof.

$$\begin{aligned}
k &\leq \frac{w(T) - 2d_m}{\lambda - 2d_m} + 1 \quad \text{by Theorem 3} \\
\implies L/\lambda &\leq \frac{w(T) - 2d_m}{\lambda - 2d_m} + 1 \quad \text{since } L \leq k\lambda \\
\implies L &\leq \left(\frac{w(T) - 2d_m}{\lambda - 2d_m} + 1\right)\lambda \\
\implies L/L^* &\leq \left(\frac{w(T) - 2d_m}{\lambda - 2d_m} + 1\right)\lambda/w(T^*) \quad \text{since } w(T^*) \leq L^* \\
\implies L/L^* &\leq \left(\frac{(1.5)w(T^*) - 2d_m}{\lambda - 2d_m} + 1\right)\lambda/w(T^*) \quad \text{by Christofides bound} \\
\implies L/L^* &\leq (1.5)\frac{\lambda}{\lambda - 2d_m} + 1 \quad \text{by } \lambda < w(T^*) \\
\implies L/L^* &\leq (1.5)\frac{\lambda/d_m}{\lambda/d_m - 2} + 1
\end{aligned}$$

□

Note that if we don't assume $\lambda < w(T^*)$ in the second to last step, then only a single subtour is needed and the TSP approximation is the solution. Hence, there would be a bound of 1.5 which is tighter than the above bound.

Theorem 5. $\frac{k}{k^*} \leq 1 + (1.5)\frac{\lambda/d_m}{\lambda/d_m - 2}$

Proof.

$$\begin{aligned}
k &\leq \frac{w(T) - 2d_m}{\lambda - 2d_m} + 1 \quad \text{by Theorem 3} \\
\implies k/k^* &\leq \left(\frac{w(T) - 2d_m}{\lambda - 2d_m} + 1\right)\lambda/w(T^*) \quad \text{since } k^* \geq w(T^*)/\lambda \\
\implies k/k^* &\leq \left(\frac{(1.5)w(T^*) - 2d_m}{\lambda - 2d_m} + 1\right)\lambda/w(T^*) \quad \text{by Christofides bound} \\
\implies k/k^* &\leq (1.5)\frac{\lambda}{\lambda - 2d_m} + 1 \quad \text{by } \lambda < w(T^*) \\
\implies k/k^* &\leq (1.5)\frac{\lambda/d_m}{\lambda/d_m - 2} + 1
\end{aligned}$$

□

Theorem 4 and Theorem 5 give us performance bounds for the tour partitioning heuristic used by *BC Sweep* for all three cases, SRSD, SRMD, and MRSD cases. Specifically, the length of the tour produced by *BC Sweep* for SRSD and SRMD is within $1+(1.5)\frac{\lambda/d_m}{\lambda/d_m-2}$ times of optimal and the number of vehicles used in the MRSD case is also within $1+(1.5)\frac{\lambda/d_m}{\lambda/d_m-2}$ times of optimal. As is also noted in [LSLD92], this bound is worth more when $\lambda \gg 2d_m$.

2.4.5 Atomic Regions

One of the nice aspects of this algorithm as its stands is that if f does not underestimate energy usage and navigation is flawless, then each of the decomposed cells will be covered in an *atomic* nature. By atomic, we mean the covering of a cell will not be interrupted by a need for a recharge. This component is a useful feature in many applications where a room or designated area must be swept or covered all at once with no intermission guaranteed.

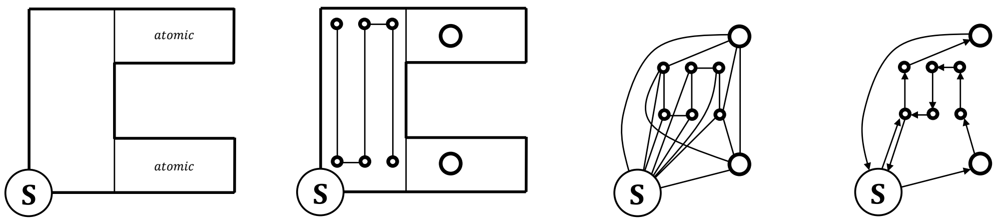


Fig. 2.7: Modified *BC Sweep* illustration for non-atomic regions. The environment contains two atomic regions and a single interruptable region. (Far-left) map decomposition, (Center-left) key routing nodes identified, (Center-right) routing graph constructed, (Far-right) traversal route determined.

However, there are applications where every decomposed cell need not have an atomic covering. In such cases, one can optimize the above algorithm to make less service trips and only perform them in designated regions. We extend the original *BC Sweep* to handle the scenario where there is a set of cells A which we want to be atomically covered and a set of cells B which does not have this constraint. Note $A \cup B = X$ and $A \cap B = \emptyset$.

We need only redefine $G = (V, E)$ and cost matrix C and the rest of the algorithm remains the same.

Define $G_1 = (V_1, E_1)$ to be

- x_0 is a vertex in G_1.
- All cells $x_k \in A$ are vertices in G_1.
- For every ox-plow cover-path
  $y_1^k \to \cdots \to y_m^k = r_k$ corresponding to $x_k \in B$, the first and last nodes become vertices: $y_1^k, y_m^k \in V_1$. Like the depot, y_1^k, y_m^k will have a null cover-paths.
- There is an edge between all atomic cell vertices. There is an edge between all atomic cell vertices and start/finish cover-path vertices.
- Assign the weight $w(e_{ij}) = f(r_{ij}^*) + \frac{1}{2}(f(r_i) + f(r_j))$ to each edge.

Define $G_2 = (V_2, E_2)$ to be

- x_0 is a vertex in G_2.
- For all cover-paths
  $y_1^k \to \cdots \to y_m^k = r_k$ corresponding to $x_k \in B$, all nodes become vertices: $\{y_1^k, ..., y_m^k\} \subseteq V_2$.
- For every vertex y_i^k there is an edge e between y_i^k and y_{i+1}^k with weight $w(e) = f(y_i^k \to y_{i+1}^k)$.
- For every vertex y_i^k there is an edge e between x_0 and y_i^k with weight of the shortest path between the two.

Our final graph G is the union of these two graphs: $G = G_1 \cup G_2$.

We must now slightly change our cost matrix C after we approximate a TSP tour

$T = v_0^T \to v_1^T \to \cdots \to v_0^T$ on G.

$$C_{ij} = \begin{cases} f(v_0^T \to v_i^T \to \cdots \to v_j^T \to v_0^T) & \text{if this cost is } \leq \lambda \text{ and } i < j \text{ and } v_i^T \text{ is spawned from } B \\ f(v_0^T \to v_{i+1}^T \to \cdots \to v_j^T \to v_0^T) & \textbf{else} \text{ if this cost is } \leq \lambda \text{ and } i < j \\ \infty & \text{otherwise} \end{cases}$$

This change in C is necessary because in the non-atomic regions, the robot must return to the same spot it left off so that it may complete coverage in that area instead of advancing to the next region.

We now simply plan the entire route by running *BC Sweep* on the newly defined G and C. The route respects the λ-capacity constraint and guarantees regions $x_k \in A$ remain atomic. Figure 2.7 demonstrates the modified approach.

Theorem 6. *The modified BC Sweep covers M, obeys the fuel capacity constraint, and does not service in atomic regions.*

Proof. The algorithm behaves the same in the atomic regions so the proof follows through in the same manner as the original theorem for atomic regions. It is only necessary to argue that complete coverage occurs in non-atomic regions. Since the cost matrix C was constructed in a manner that if a non-atomic cover path was divided by a service trip that the tour would return to the same node after refueling, we know that no legs of the sweep will be skipped. And since we assumed a feasible solution, after resuming sweeping, progress will always be made in non-atomic regions until complete. Thus, complete coverage occurs in non-atomic regions. □

2.4.6 Dynamic BC Sweep

The theoretic algorithms proposed so far are not entirely feasible in practice. *BC Sweep* leveraged unrealistic liberties when planning its covering route.

First, we assumed that when in operation, the robot has perfect navigation in the environment. Due to uncertainty in sensors, actuators and imperfect navigation algorithms, this assumption is an impractical one.

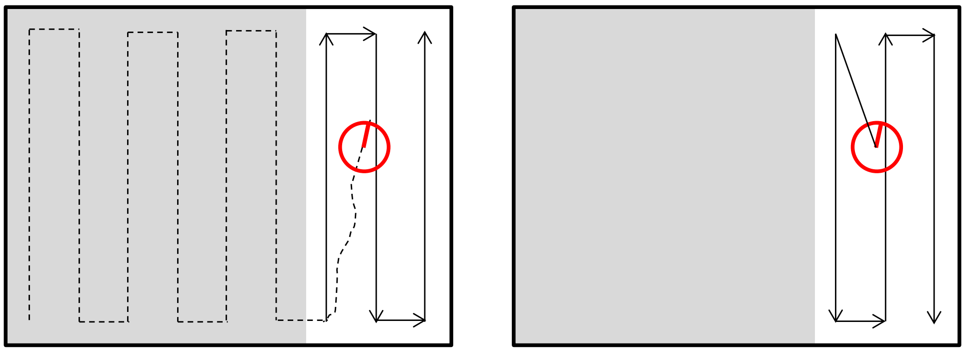
Second, *BC Sweep* relies on a function f which does not underestimate the energy used over a given path. Though one could exaggerate f to meet this requirement, the more accurate f is, the more energy efficient and coverage effective *BC Sweep* becomes.

To relax some of these assumptions, we propose extensions to the *BC Sweep* algorithm to be more reliable in practice. The key here is that the algorithm needs to be adaptive while executing.

Variability in Navigation

To account for imperfection in navigation while executing a sweeping route, we dynamically adjust the route taken. Consider the scenario where while performing the back-and-forth motions in a cell, the robot drifts along one of the lengths. Figure 2.8a. If the navigation realizes the error, we can modify the route to cover the missing area. Thus we can dynamically adjust our *BC Sweep* route. Figure 2.8b. In such a case, we note that it is possible the atomic regions may need to be interrupted depending on how much rerouting is necessary. Denoting Q_λ to be the current fuel life, the online dynamic algorithm:

```
while covering cell $x_k$ do
    if off path then
        recalc. ox-plow path $r$ for the remainder of $x_k$;
        if $x_k$ is atomic and $f(r) + f(x_k \to x_0) > Q_\lambda$ then
            make $x_k$ interruptible;
        end
        rerun BC Sweep;
    end
end
```



(a) Navigation error. (b) Navigation error recovery.

Fig. 2.8: Variability in navigation.

Updating f

One of the more difficult aspects of *BC Sweep* is determining the energy consumption function f. Any offline theoretic function f estimate could change depending where you are in the route, map, or on any other factors. To account for a variable function f, it is possible to recalculate an f estimate dynamically while executing the sweeping. For example, one could consider a moving average approach evaluated on some past window size w for dynamically updating f. After each online update, *BC Sweep* can be rerun. Similar to accounting for variability in navigation, depending on how much f changes at any point, atomic regions may become interruptible or even revert back to being atomic.

2.4.7 Simulations

We simulated *BC Sweep* on a test convenience store environment requiring covering. Figure 2.9 shows the floor plan of a convenience store for which *BC Sweep* was run.

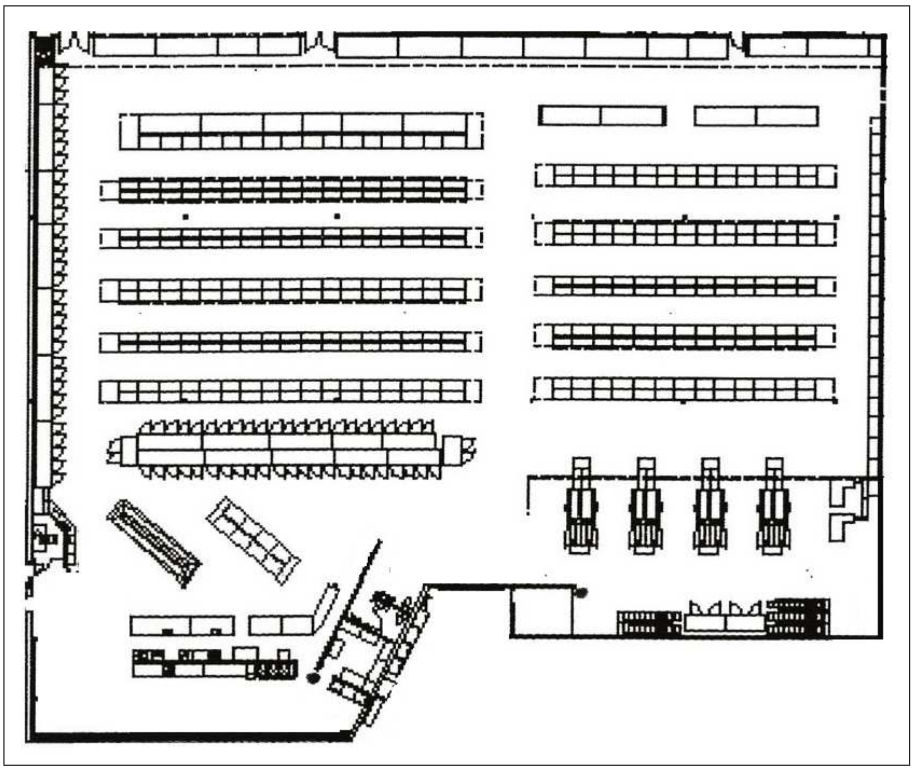


Fig. 2.9: An example convenience store layout.

We ran the *BC Sweep* algorithm with the following parameters. We used a circular robot of radius 0.75 *meters*. The fuel consumption function f was a one-to-one correspondence with the total distance of a path traveled. For example, if the robot traversed a path of 10 *meters* then the robot would have consumed 10 *units* of fuel. Each cell constructed from the boustrophedon decomposition was designated as an atomic region. Figure 2.10 shows the final cover route that *BC Sweep* generates for one of the parameter settings tested with a fuel capacity of $5d_m$ and a single service station in the top-left corner.

Appendix D shows the full set of test runs under several fuel capacities λ and varying locations and numbers of the service station. It can be seen from these experiments that *BC Sweep* is able to route the robot efficiently through a real test environment with minimal wasted travel time. Additionally, in Appendix D, Figures 2.24 - 2.26 show the basic *BC Sweep* steps while Figures 2.27 - 2.31 show the algorithm run with different fuel capacities λ and between one and three service stations.

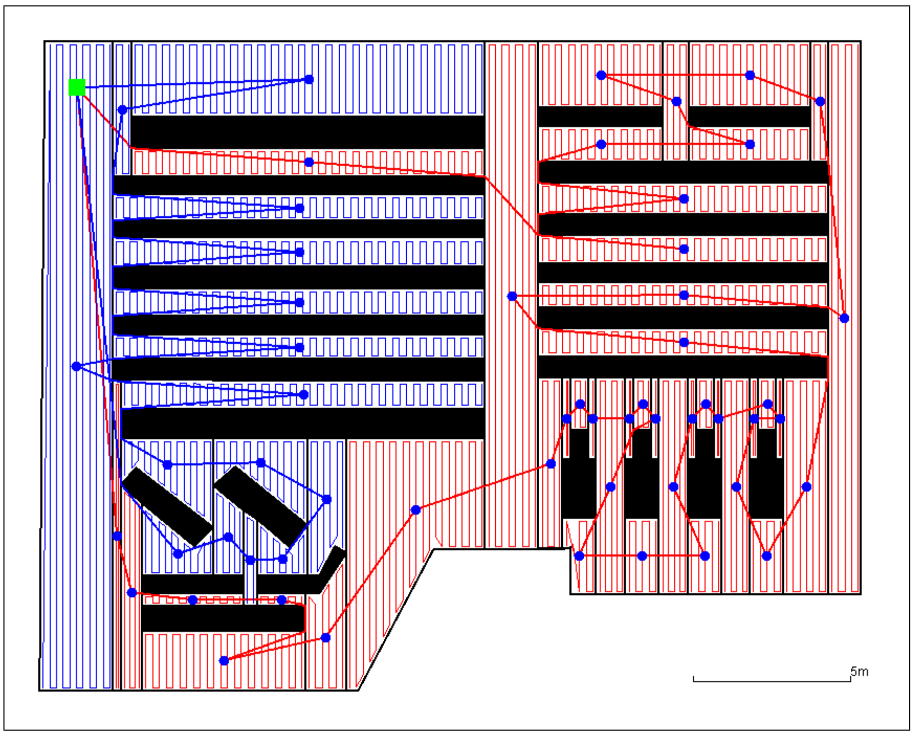


Fig. 2.10: Single depot, $\lambda = 5d_m$.

2.4.8 BC Sweep Timed Experiments

We also ran timed experiments using the tour partitioning heuristic for the single depot case on the VRP problems given in [ABB+]. The problems were transformed into *BC Sweep* appropriate problems by treating the item count demanded at each node as the cover cost distance of a cell. We ran the heuristic with a fuel capacity of $2d_m$, $4d_m$, $6d_m$ and $10d_m$. We compared the total fuel used on tours generated by the *BC Sweep* with lower bound costs generated using the integer programming formulation presented in the next section.

Both *BC Sweep* and the integer programs were executed with a 2.40GHz Intel quad-core Core 2 Quad Q6600 processor. The integer programs were run with ILOG CPLEX 8.1 [ILO] with a time limit of 12000 CPU seconds. A summary of results are presented in Table 2.1.

As can be seen, the smaller the fuel capacity the weaker the heuristic performs. However, overall we find the heuristic is competitive in the sense that the bench mark problems run are all within 2.2% of optimal. We also found that all *BC Sweep* instances (up to 80

Fuel Capacity (λ)	Best Ratio	Worst Ratio	Average Ratio
$2d_m$	1.2068	2.1808	1.5362
$4d_m$	1.0164	1.2472	1.1349
$6d_m$	1.0090	1.1276	1.0629
$10d_m$	1.0008	1.0995	1.0289

Table 2.1: Summary of *BC Sweep* executed on VRP benchmark problems. Ratios are computed by dividing the fuel consumption of the *BC Sweep* solution by the lower bound generated by the ILP.

nodes and 3160 edges) executed in under 0.5 CPU seconds showing that the heuristic is not only competitive but also fairly time efficient. For the full set of computational results see Appendix 2.5.

2.5 Integer Programming Methods

In *BC Sweep*, after graph construction, we use a DCVRP tour partitioning heuristic to solve for a feasible route abiding by the fuel capacity constraint. Instead of using a heuristic, it is also possible to express the problem as a binary integer linear program and to solve using studios such as CPLEX.

Though there is a wealth of work studying general forms of vehicle routing problems where each node has some demand for a specific commodity, as stated in [Kar11], there is limited work on ILP formulations for the DCVRP variation.

Of the existing proposed methods, the most studied component of the formulation is that which ensures the tour produced is not disjoint and is a continuous path through locations. This is referred to in the literature as *subtour elimination.* Subtour elimination is accomplished in one of three ways. One method is set partitioning which requires an exponential number of constraints [LDN84]. Another method is MTZ formulations which require $O(n)$ extra variables and $O(n^2)$ additional constraints [KB85] [AC91]. There are also commodity flow network models [Wat88] [Kar11].

Here, we use a solution which is a variant based of the MTZ formulation proposed in [KB85].

2.5.1 MTZ TSP Formulation

Our solution is based on the Miller-Tucker-Zemlin (MTZ) formulation [MTZ60] for solving TSP problems using integer programming. We begin by presenting this formulation. We let d_{ij} be the cost of edge e_{ij} in our graph G. $z_{ij} \in \{0, 1\}$ will be variables we need to solve for. $z_{ij} = 1$ when the edge is used in the TSP tour and 0 otherwise. Note that we would never need to use an edge more than once because of the triangle inequality on G. A natural starting place would be the following.

$$\min_{\mathbf{z} \in \mathbb{R}^{n^2}} \sum_{i,j} e_{ij} z_{ij}$$

$$\text{s.t.} \sum_{j} z_{ij} = 1 \quad \forall i \in V, \tag{2.4}$$

$$\sum_{i} z_{ij} = 1 \quad \forall j \in V, \tag{2.5}$$

$$z_{ij} \in \{0, 1\} \quad \forall i, j \tag{2.6}$$

This set-up ensures every vertex has both an in-degree and out-degree of 1 and consequently, that every vertex is visited exactly once. This formulation as is, however, allows disjoint subtours or cycles as in Figure 2.11.

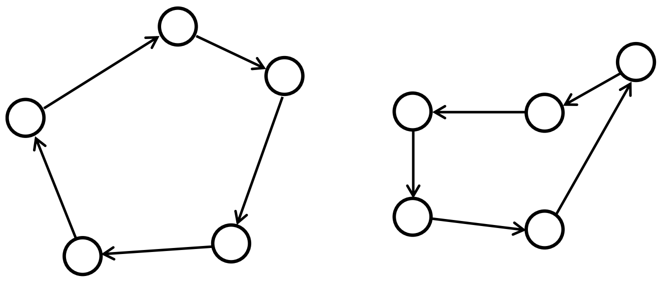


Fig. 2.11: An example of two subcycles in a graph. Each vertex has in-degree and out-degree of 1 and thus visited exactly once.

To force there to be a single cycle, we must add extra variables and extra constraints. We must solve for variables $\{u_1, u_2, \ldots, u_n\}$ which are continuous but can viewed as taking on the position of each vertex in the cycle; if $u_i < u_j$ implies vertex i comes before vertex j in the tour. We add the following contraints known as the MTZ constraints.

$$u_1 = 1, \tag{2.7}$$

$$2 \leq u_i \leq n \quad \forall i \neq 1, \tag{2.8}$$

$$u_i - u_j + 1 \leq (n-1)(1 - z_{ij}) \quad \forall i \neq 1, \forall j \neq 1. \tag{2.9}$$

Note we added an extra $O(n)$ variables and $O(n^2)$ constraints to the formulation. To understand why this eliminates disjoint cycles, see that if every cycle in the graph contains the start node, then there is only one cycle. This formulation forces this idea on the tour. We force the start vertex to have position 1 ($u_1 = 1$) and all other positions be between 1 and n. When we use edge e_{ij}, it forces $u_i + 1 \leq u_j$ which implies $u_i < u_j$. Now if we don't use e_{ij}, then the inequality is not constraining and simply forces the difference to be less than some finite value. There cannot be cycles which do not contain the start vertex because the vertex positions u_i on that cycle will be forced to increase to infinity which would violate the upper bounds on u_i.

2.5.2 SRSD IP Formulation

The integer programming formulation for the single-robot, single-depot case is based on the MTZ formulation. For convenience, let our cell vertices be set X and our depot vertices be D. $X \cup D = V$, $X \cap D = \emptyset$.

$$
\begin{aligned}
\min_{\mathbf{z}\in\mathbb{R}^{n^2}} \sum_{i,j} & e_{ij} z_{ij} \\
\text{s.t.} \sum_{j} z_{ij} &= 1 \quad \forall i \in X, \\
\sum_{i} z_{ij} &= 1 \quad \forall j \in X, \\
z_{ij} &\in \{0,1\} \quad \forall i, j,
\end{aligned}
$$

$$
\begin{aligned}
u_i &= 0, \quad i \in D, && \text{(2.10a)} \\
0 \le u_i &\le \lambda \quad \forall i \in X, && \text{(2.10b)} \\
u_i - u_j + e_{ij} &\le (\lambda + e_{max})(1 - z_{ij}) \quad \forall i \in V, \forall j \notin D, && \text{(2.10c)} \\
u_i + e_{ij} z_{ij} &\le \lambda \quad \forall i \in X, j \in D. && \text{(2.10d)}
\end{aligned}
$$

See that we asserted that each cell is visited exactly once. We need not assert any constraint on the degrees of the service station vertex for the degree constraints on the cell vertices is enough to ensure its in-degree equals its out-degree.

In this formulation, the u_i variables represent the concept of the amount of fuel consumed by the robot at vertex i. Thus, we set this to be 0 for the service station vertex. To ensure no vertex consumes more than our capacity λ, we bound this appropriately in (2.10b). We must also assert that the last leg/edge of each subtour abides by the capacity constraint. This is accomplished in (2.10d).

Eliminating disjoint cycles is done in the same manner as the MTZ formulation. In this case, however, if we use an edge e_{ij}, we add the associated energy e_{ij}. See this as constraint (2.10c) reduces to $u_i + e_{ij} \le u_j$ when edge e_{ij} is used. When it's not used, we just assert that the difference between energy consumption variables u_i and u_j is at most the finite value of $(\lambda + e_{max})$ when e_{max} takes on the value of the maximum edge weight in G.

2.5.3 SRMD ILP Formulation

For the single-robot, multi-depot scenario the ILP involves a bit extra work. We make n copies of each service station. We denote this augmented depot set as D^*. The augmented service station set has the same function but each can only be visited at most once (hence why we need n of each). The graph remains a complete graph. All depot copies get the same weight on edges incident with cell vertices. We put zero weight edges between all copies of a service station vertex and intraservice station weights remain the same. Last, all edges coming from a depot into our special starting service station vertex have weight 0. This allows the robot to finish at any depot with no additional cost to complete the tour. See Figure 2.12 on this transformation. With this in place, we present the ILP formulation for our problem.

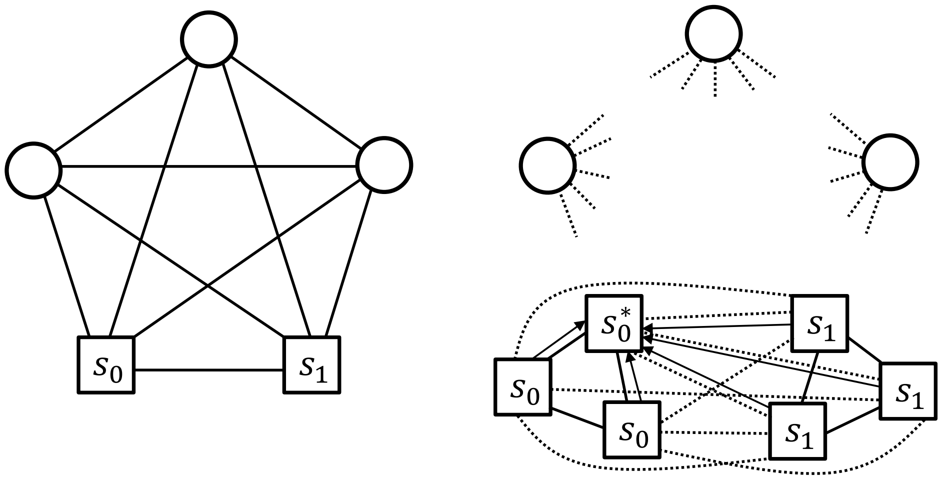


Fig. 2.12: An example of a three-cell, two-node graph into the augments graph for the ILP. (*Left*) The original graph. (*Right*) The augment graph with 3 copies of each depot. The solid arcs all have 0 weight. Dashed arcs retain their original weight.

$$
\begin{aligned}
\min_{\mathbf{z}\in\mathbb{R}^{(n+nm)^2}} \sum_{i,j} e_{ij}z_{ij} & & \text{(2.11a)}\\
\text{s.t.} \sum_{j} z_{ij} = 1 & \quad \forall i \in X, & \text{(2.11b)}\\
\sum_{i} z_{ij} = 1 & \quad \forall j \in X, & \text{(2.11c)}\\
\sum_{j} z_{ij} = \sum_{j} z_{ji} & \quad \forall i \in D^*, & \text{(2.11d)}\\
\sum_{j} z_{ij} \leq 1 & \quad \forall i \in D^*, & \text{(2.11e)}\\
z_{ij} \in \{0,1\} & \quad \forall i,j, & \text{(2.11f)}
\end{aligned}
$$

$$
\begin{aligned}
u_i = 0, & \quad \forall i \in D^*, & \text{(2.11g)}\\
0 \leq u_i \leq \lambda & \quad \forall i \in X, & \text{(2.11h)}\\
u_i - u_j + e_{ij} \leq (\lambda + e_{max})(1 - z_{ij}) & \quad \forall i \in V, \forall j \notin D^*, & \text{(2.11i)}\\
u_i + e_{ij}z_{ij} \leq \lambda & \quad \forall i \in V, \forall j \in D^*. & \text{(2.11j)}
\end{aligned}
$$

$$
\begin{aligned}
q_{s_0^*} = 1, & & \text{(2.11k)}\\
2 \leq q_i \leq n + mn & \quad \forall i \neq 1, & \text{(2.11l)}\\
q_i - q_j + 1 \leq (n + mn - 1)(1 - z_{ij}) & \quad \forall i \in V, \forall j \neq 1. & \text{(2.11m)}
\end{aligned}
$$

Our objective remains the same in (2.11a) while (2.11b) and (2.11c) retain that every cell is visited exactly once. (2.11d) and (2.11e) assert that the in-degree and out-degree of each service station vertex is the same as the degree and is at most one. This forces each depot to be visited at most once. Constraints (2.11g) - (2.11j) as in the SRMD and MRSD formulations ensure that the robot capacity constraint is met. And as with the MTZ TSP formulation constraints, 2.11k - 2.11m make certain that the tour is a single continuous tour from start depot to start depot. Since we used a 0 weight back home to the start service station, we can simply eliminate that edge and end on the optimal depot.

2.5.4 MRSD ILP Formulation

For the multi-robot, single-depot objective, only minor modifications are made from the SRSD IP Formulation. We change the objective to minimize the number of robots k (subtours of the route) needed. We then simply force the in-degree and out-degree of the depot vertex to be k in (2.12a) and (2.12b).

$$\begin{aligned}
\min_{\mathbf{z}\in\mathbb{R}^{n^2}} \ & k \\
\text{s.t.} \sum_j z_{ij} &= 1 \quad \forall i \in X, \\
\sum_i z_{ij} &= 1 \quad \forall j \in X, \\
\sum_j z_{ij} &= k \quad i \in D, && (2.12a) \\
\sum_i z_{ij} &= k \quad j \in D, && (2.12b) \\
z_{ij} &\in \{0,1\} \quad \forall i, j,
\end{aligned}$$

$$\begin{aligned}
u_i &= 0, \quad i \in D, \\
0 \le u_i &\le \lambda \quad \forall i \in X, \\
u_i - u_j + e_{ij} &\le (\lambda + e_{max})(1 - z_{ij}) \quad \forall i \in V, \forall j \notin D, \\
u_i + e_{ij} z_{ij} &\le \lambda \quad \forall i \in X, j \in D.
\end{aligned}$$

However because we use triangle inequality, we can use the above formulation to get an equally good result. This formulation is able to be solved easier due to the number of varying optimal solutions.

On Conflicting Objectives

One may initially question whether the objective for minimizing total distance in the SRSD case naturally would reduce the number of robots (refuels/loops back to the depot) in the MRSD case. If it were the case, then the MRSD ILP formulation is worth little for we could just reuse the SRSD formulation. This is not the case however. The objectives are separate and can often conflict. To demonstrate this we give the example in Figure 2.13. We will show that for this example, increasing the number of loops (i.e. increasing the robot count for MRSD) can actually decrease the total distance.

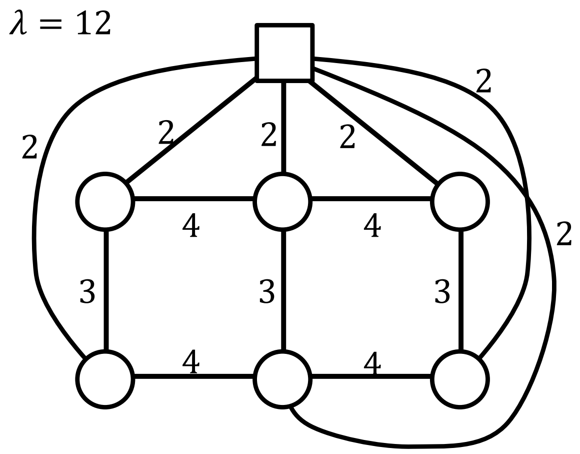


Fig. 2.13: An example graph where the objectives conflict. Fuel capacity $\lambda = 12$.

We note first that the TSP tour on this graph has a length of 21. Figure 2.14 shows such a tour.

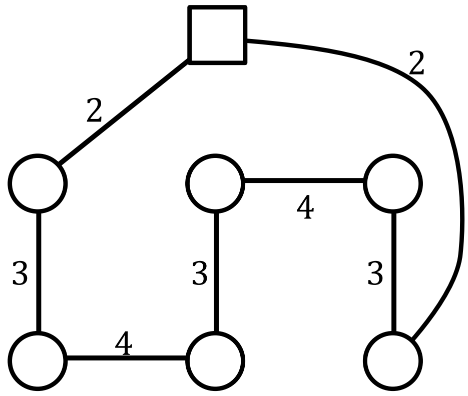


Fig. 2.14: TSP tour of the example graph. Total cost = 21.

There is a solution using three loops (refuels/robots) that abides by the fuel constraint and also has a cost of 21. This route is shown in Figure 2.15. We know this solution is optimal with respect to minimizing distance since it matches the cost of the TSP solution.

When the objective switches to minimize the number of loops, though, we find that there exist solutions that use just two loops. One such solution is shown in Figure 2.16

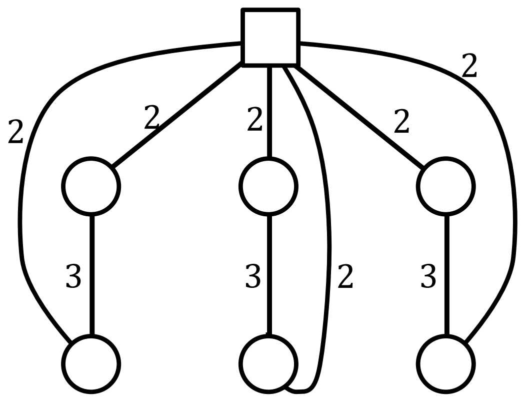


Fig. 2.15: Distance minimizing optimal solution abiding by the fuel capacity. Total cost = 21.

which has a total cost of 24. However, any solution which uses two loops cannot match the total distance of 21. Hence this example shows that the objectives are distinct since minimizing the total distance cost does not necessarily reduce the number of refuels needed.

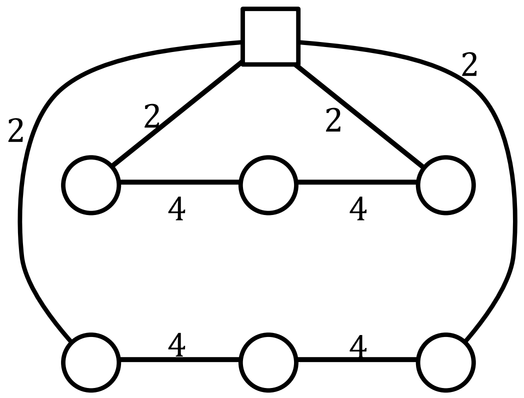


Fig. 2.16: Refuel count minimizing optimal solution abiding by the fuel capacity. Total cost = 24.

2.5.5 ILP Timed Experiments for SRSD

We ran timed experiments of the integer linear programs for the benchmark problems of [ABB+]. We ran experiments on all the problems with fuel capacities $2d_m$, $4d_m$, $6d_m$

and $10d_m$. The integer programs were executed with a 2.40GHz Intel quad-core Core 2 Quad Q6600 processor and were run with ILOG CPLEX 8.1 [ILO] with a time limit of 12000 CPU seconds.

We find that the smaller the fuel capacity the more difficult the problem becomes to solve. For example, with a fuel capacity $10d_m$ nearly all the problems were solved within the time limit while only a few of those with capacities $2d_m$ could be solved optimally within the time limit. For full table computational results and times, see Appendix 2.6.

2.6 Conclusion

In this work, we have introduced the *BC Sweep* algorithm to address the real problem of robot path coverage, with a battery or fuel capacity constraint. We build upon previous coverage research using boustrophedon decomposition, and contribute the *BC Sweep* heuristic planning algorithm that has the property of complete coverage, under the assumption that there is a limited amount of space that can be covered on a single battery charge, and the assumption that there is a recharging service station (s). We show the algorithm is adaptable for three distinct but related problem definitions. We presented a proofs of correctness that verify the complete coverage under the resource constraint. *BC Sweep* runs on arbitrary geometrical physical layouts, and we have demonstrated it in simulation using a real world map and a real simulated coverage robot. We tested the algorithm with varying parameters for the fuel capacity and service station locations. We also demonstrate that the three problem scenarios can also be expressed by integer linear programs. We compare the optimality of the the *BC Sweep* heuristic using benchmark problems in the literature. After having addressed the real battery constraint, our future work includes to continue to bring coverage algorithms closer to real situations faced by real robots in the real world.

Appendices

A Graph SLAM-SD Experimental Results - Extended

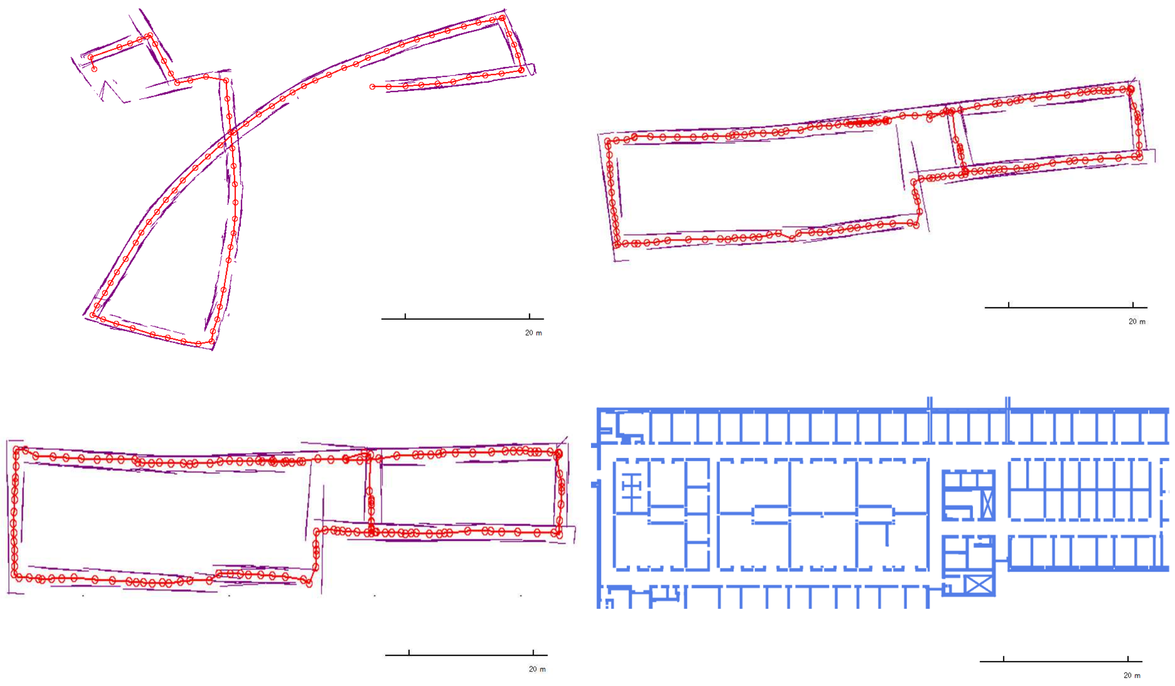


Fig. 2.17: **WEH Floor Data Set**: raw data map produced using odometry only (top-left); planar SLAM map (top-right); planar SLAM map [parallel-orthogonal constraint] (bottom-left); ground truth map (bottom-right)

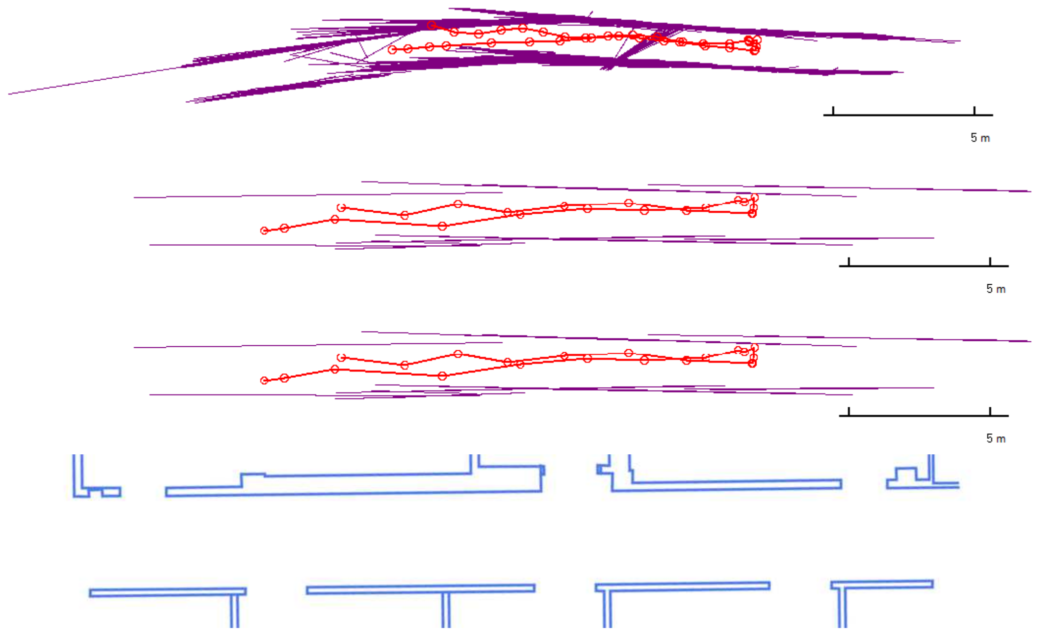


Fig. 2.18: **Hallway Data Set**: raw data map produced using odometry only (top-top); planar SLAM map (middle-top); planar SLAM map [parallel-orthogonal constraint] (middle-bottom); ground truth map (bottom-bottom)

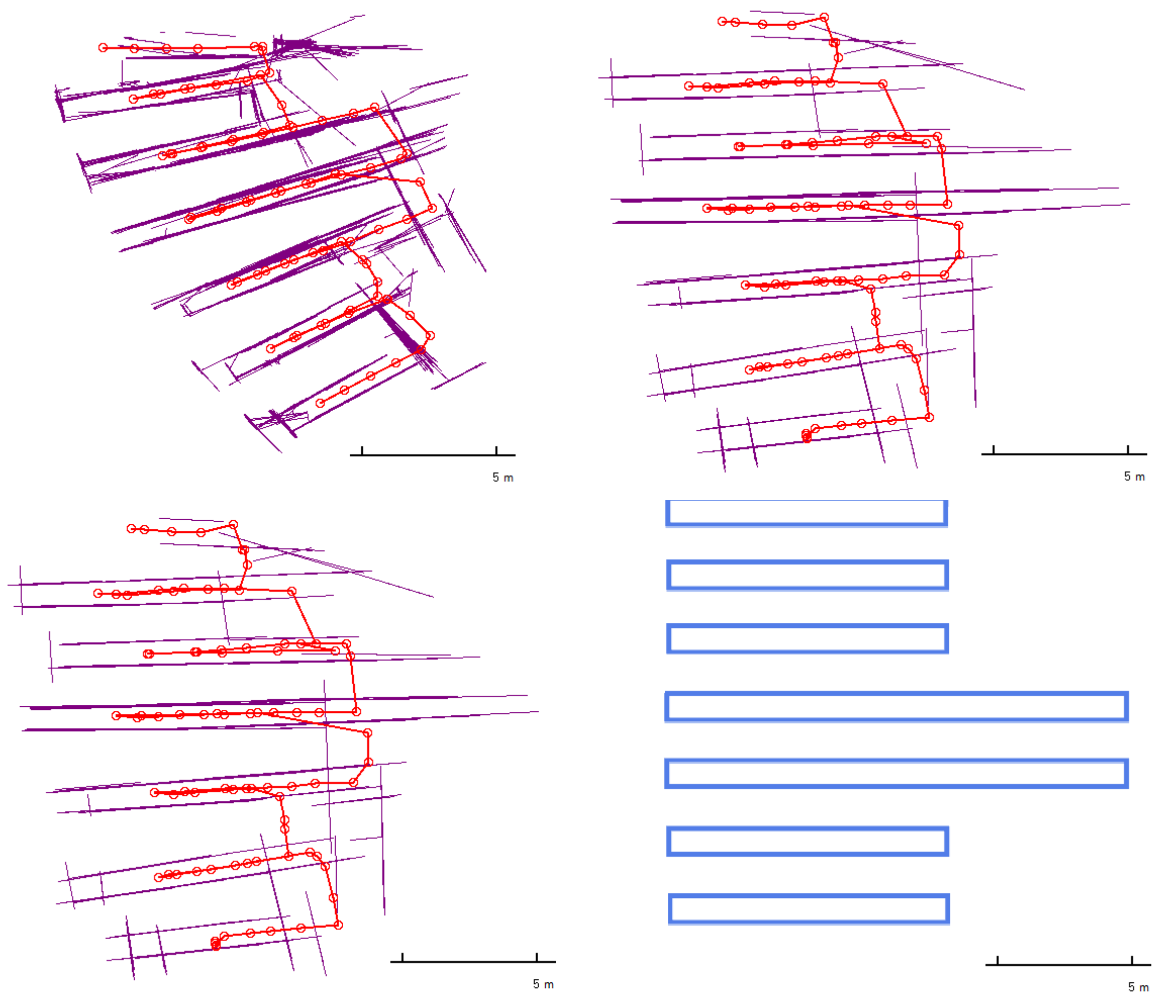


Fig. 2.19: **Library Data Set**: raw data map produced using odometry only (top-left); planar SLAM map (top-right); planar SLAM map [parallel-orthogonal constraint] (bottom-left); ground truth map (bottom-right)

B Kinect Calibration

B.1 Introduction

For this project, we placed 3-Kinect sensors on CoBot 4. See Figure 2.20. One Kinect is placed horizontally at waist height directly in the front center of the robot. Two Kinects are placed vertically on the sides of the base. They are angled at approximately 45° so that their fields of vision (FOV) cross. This leads to a wide field of vision.

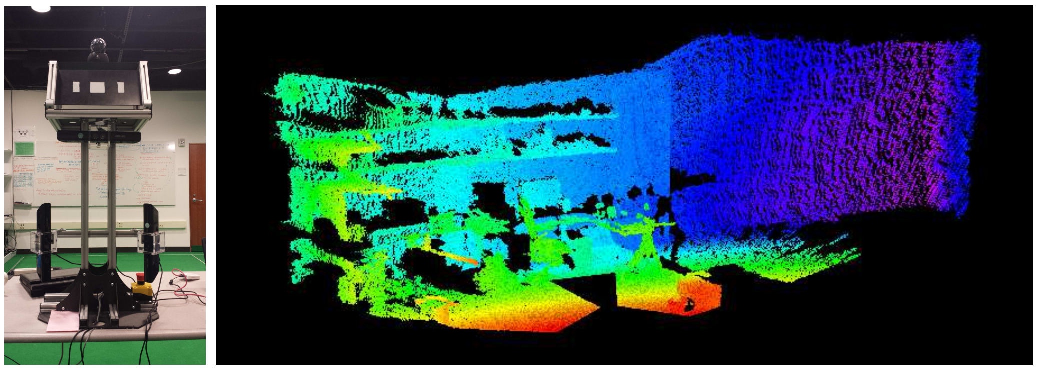


Fig. 2.20: CoBot4 base with 3 Kinect sensors (left); Wide FOV produced by using 3 Kinects

With multiple Kinects receiving data it becomes necessary to calibrate the relative poses to each other so that a combined point cloud can be generated with data from each sensor. Thus, without loss of generality, it is necessary to determine the 3D homogenous transform **H** from the frame of Kinect 2 to the frame of Kinect 1. Determining this transformation can be measured by hand or calibrated manually by using more advanced techniques and tools. However, these processes tend to be tedious and have to be repeated for each sensor anytime they are added or adjusted. We propose here a method for automatic calibration of the extrinsic poses of two or more Kinect-like RGBD sensors on a mobile robot. The objective is to give a robot the ability to automatically calibrate various sensors while operating in the environment (self-correcting sensor alignment). The work allows the robot to simply run in the environment and automatically compute the relative transforms between any two Kinect sensors without prior information on the poses of the sensors attached to the robot.

B.2 Approach

For simplicity, we assume there are just two Kinects, Kinect 1 and Kinect 2. The goal is to find the transform $\mathbf{H}$ which will take an observation in Kinect 2's frame and put it into Kinect 1's.

The approach is as follows

1. Drive the robot in the environment for an extended period of time.
2. At each timestep t, record the observed point cloud $\mathbf{C}_t^1$ from Kinect 1 and $\mathbf{C}_t^2$ from Kinect 2. The result is two sequences of point cloud observations: $\mathbf{C}_{0:N}^1$ and $\mathbf{C}_{0:N}^2$.
3. For each pair of consecutive point clouds $\left(\mathbf{C}_{i-1}^x, \mathbf{C}_i^x\right)$ use the ICP algorithm to determine the transform $\mathbf{H}_i^x$ between them. The result from this is two sequences of transforms $\mathbf{H}_{1:N}^1$ and $\mathbf{H}_{1:N}^2$ (i.e. the measured trajectories of each sensor).
4. Use a non-linear least squares optimization to determine the optimal transform $\mathbf{H}$ such that when applied to $\mathbf{H}_{1:N}^2$ minimizes the error between the two trajectories.

B.3 ICP

The approach above in Step 2 utilizes the Iterative Closest Point (ICP) algorithm for point cloud registration to incrementally register pairs of clouds in a sequence from each Kinect. The resulting sequences of transforms compose unique trajectories for each sensor in its own frame. The ICP algorithm is able to take two point clouds $\mathbf{C}_1$ and $\mathbf{C}_2$ and determine the homogenous transform $\mathbf{H}_2^1$ which puts the points in $\mathbf{C}_2$ into $\mathbf{C}_1$'s frame. See Figure 2.21 which shows two clouds being aligned. The ICP variant which is used is based off of a Point Cloud Library(PCL) implementation where surface normals are used when performing the point cloud matching between the two clouds.

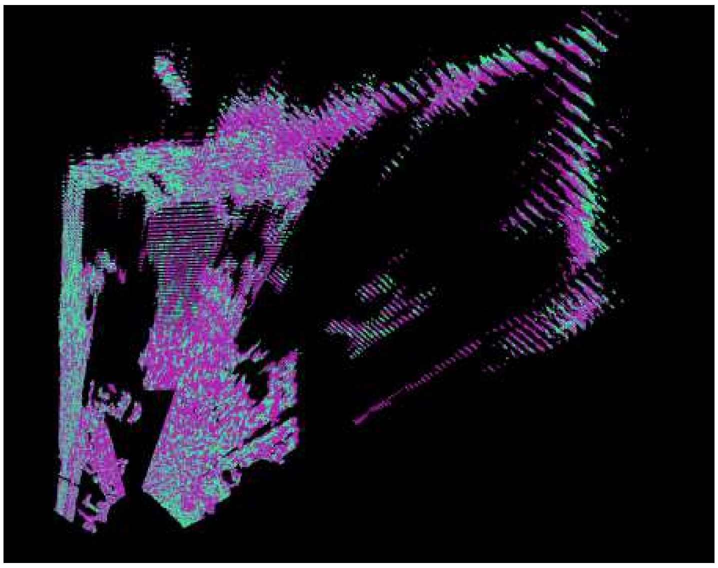


Fig. 2.21: Two points clouds being aligned with the ICP algorithm

B.4 Probabilistic Foundation

Our goal is to find the transform $\mathbf{H}$ from Kinect 2 to Kinect 1 and the true trajectory of Kinect 1 $\mathbf{H}^{1*}_{1:N}$ that are most likely given the data from each Kinect. Thus, we wish to optimize the following posterior.

$$P\left(\mathbf{H}, \mathbf{H}^{1*}_{1:N} | \mathbf{H}_{1:N}\right) \tag{2.13}$$

We solve for the MLE of the poster. We denote T to be the space of all transforms

$$\underset{\mathbf{H}\in T, \mathbf{H}^{1*}_{1:N}\in T^N}{\operatorname{argmax}} P\left(\mathbf{H}^{1}_{1:N}, \mathbf{H}^{2}_{1:N} | \mathbf{H}, \mathbf{H}^{1*}_{1:N}\right) \tag{2.14}$$

We can decompose the probability of Equation 2.14 like so.

$$\begin{aligned}
P\left(\mathbf{H}^1_{1:N}, \mathbf{H}^2_{1:N} | \mathbf{H}, \mathbf{H}^{1*}_{1:N}\right) &= \prod_{i=1}^{N} P\left(\mathbf{H}^1_i, \mathbf{H}^2_i | \mathbf{H}, \mathbf{H}^{1*}_i\right) \\
&= \prod_{i=1}^{N} P\left(\mathbf{H}^1_i | \mathbf{H}^{1*}_i\right) P\left(\mathbf{H}^2_i | \mathbf{H}, \mathbf{H}^{1*}_i\right) \\
&= \prod_{i=1}^{N} P\left(\mathbf{H}^1_i | \mathbf{H}^{1*}_i\right) P\left(\mathbf{H}\mathbf{H}^2_i\mathbf{H}^{-1} | \mathbf{H}^{1*}_i\right)
\end{aligned}$$

Here $\mathbf{H}\mathbf{H}^2_i\mathbf{H}^{-1}$ is the application of $\mathbf{H}$ to the trajectory observations of Kinect 2. See section B.5 for why. For notational convenience, we let $\bar{\mathbf{H}}^1_i = \mathbf{H}\mathbf{H}^2_i\mathbf{H}^{-1}$. If we make the assumption that the rotational and translational observations errors are independent, we can further decompose.

$$\begin{aligned}
P\left(\mathbf{H}^1_{1:N}, \mathbf{H}^2_{1:N} | \mathbf{H}, \mathbf{H}^{1*}_{1:N}\right) &= \prod_{i=1}^{N} P\left(\mathbf{H}^1_i | \mathbf{H}^{1*}_i\right) P\left(\bar{\mathbf{H}}^1_i | \mathbf{H}^{1*}_i\right) \\
&= \prod_{i=1}^{N} P\left(\mathbf{R}^1_i | \mathbf{R}^{1*}_i\right) P\left(\mathbf{T}^1_i | \mathbf{T}^{1*}_i\right) P\left(\bar{\mathbf{R}}^1_i | \mathbf{R}^{1*}_i\right) P\left(\bar{\mathbf{T}}^1_i | \mathbf{T}^{1*}_i\right)
\end{aligned}$$

Distribution Assumptions

Rotation

We measure the distance between two rotational transforms as the Euclidean distance between the two unit quaternions representing them. This metric is chosen for its spatial and computational efficiency[]. For convenience, define $\Phi\left(\mathbf{R}_a, \mathbf{R}_b\right) = \Phi\left(\mathbf{R}_b, \mathbf{R}_a\right)$ to be this distance.

We assume that the quaternion distance between a true rotation $\mathbf{R}^{1*}_i$ and an observed rotation $\mathbf{R}^1_i$ is distributed as folded normal distribution of some normal distribution $\mathcal{N}(0, \sigma_q)$. Thus,

$$P\left(\mathbf{R}^1_i | \mathbf{R}^{1*}_i\right) = P\left(\Phi\left(\mathbf{R}^1_i, \mathbf{R}^{1*}_i\right)\right) = \frac{2}{\sigma_q\sqrt{2\pi}} \exp\left\{-\frac{\left(\Phi\left(\mathbf{R}^1_i, \mathbf{R}^{1*}_i\right)\right)^2}{2\sigma_q^2}\right\}$$

and

$$P\left(\bar{\mathbf{R}}_i^1|\mathbf{R}_i^{1*}\right) = P\left(\Phi\left(\bar{\mathbf{R}}_i^1, \mathbf{R}_i^{1*}\right)\right) = \frac{2}{\sigma_q\sqrt{2\pi}}\exp\left\{-\frac{\left(\Phi\left(\bar{\mathbf{R}}_i^1, \mathbf{R}_i^{1*}\right)\right)^2}{2\sigma_q^2}\right\}$$

It is clear that $P\left(\mathbf{R}_i^1|\mathbf{R}_i^{1*}\right)P\left(\bar{\mathbf{R}}_i^1|\mathbf{R}_i^{1*}\right)$ is maximized when $\mathbf{R}_i^{1*}$ becomes the midpoint between $\mathbf{R}_i^1$ and $\bar{\mathbf{R}}_i^1$. Since we now know $\mathbf{R}_i^{1*}$, it can be removed from the optimization

$$\begin{aligned}
&P\left(\mathbf{R}_i^1|\mathbf{R}_i^{1*}\right)P\left(\bar{\mathbf{R}}_i^1|\mathbf{R}_i^{1*}\right)\\
&\quad= \frac{2}{\sigma_q\sqrt{2\pi}}\exp\left\{-\frac{\left(\frac{1}{2}\Phi\left(\mathbf{R}_i^1, \bar{\mathbf{R}}_i^1\right)\right)^2}{2\sigma_q^2}\right\}\frac{2}{\sigma_q\sqrt{2\pi}}\exp\left\{-\frac{\left(\frac{1}{2}\Phi\left(\mathbf{R}_i^1, \bar{\mathbf{R}}_i^1\right)\right)^2}{2\sigma_q^2}\right\}\\
&\quad= \frac{4}{\sigma_q^2 2\pi}\exp\left\{-\frac{\left(\Phi\left(\mathbf{R}_i^1, \bar{\mathbf{R}}_i^1\right)\right)^2}{4\sigma_q^2}\right\}\\
&\quad= \eta_1\exp\left\{-\frac{\left(\Phi\left(\mathbf{R}_i^1, \bar{\mathbf{R}}_i^1\right)\right)^2}{4\sigma_q^2}\right\} \qquad (2.15)
\end{aligned}$$

Translation

Similar to rotation, we make a Gaussian noise assumption on the observed translations. Assuming that

$P\left(\mathbf{T}_i^1|\mathbf{T}_i^{1*}\right)$ and $P\left(\bar{\mathbf{T}}_i^1|\mathbf{T}_i^{1*}\right) \sim \mathcal{N}(\mathbf{T}_i^{1*}, \Sigma)$ where the covariance matrix $\Sigma = \begin{pmatrix}\sigma_e^2 & 0 & 0\\ 0 & \sigma_e^2 & 0\\ 0 & 0 & \sigma_e^2\end{pmatrix}$

Once again $P\left(\mathbf{T}_i^1|\mathbf{T}_i^{1*}\right)P\left(\bar{\mathbf{T}}_i^1|\mathbf{T}_i^{1*}\right)$ is maximized by making $\mathbf{T}_i^{1*}$ the midpoint of $\mathbf{T}_i^1$ and $\bar{\mathbf{T}}_i^1$.

$$\begin{aligned}
P\left(\mathbf{T}_i^1|\mathbf{T}_i^{1*}\right) P\left(\bar{\mathbf{T}}_i^1|\mathbf{T}_i^{1*}\right) &= \eta_2 \exp\left\{-\frac{||\mathbf{T}_i^1 - \mathbf{T}_i^{1*}||^2}{2\sigma_e^2}\right\} \eta_2 \exp\left\{-\frac{||\bar{\mathbf{T}}_i^1 - \mathbf{T}_i^{1*}||^2}{2\sigma_e^2}\right\} \\
&= \eta_2 \exp\left\{-\frac{\frac{1}{4}||\mathbf{T}_i^1 - \bar{\mathbf{T}}_i^1||^2}{2\sigma_e^2}\right\} \eta_2 \exp\left\{-\frac{\frac{1}{4}||\bar{\mathbf{T}}_i^1 - \mathbf{T}_i^1||^2}{2\sigma_e^2}\right\} \\
&= \eta_2^2 \exp\left\{-\frac{||\mathbf{T}_i^1 - \bar{\mathbf{T}}_i^1||^2}{4\sigma_e^2}\right\} \qquad (2.16)
\end{aligned}$$

Optimization

Substituting using Equations 2.15 and 2.16 gives us

$$\begin{aligned}
\underset{\mathbf{H}\in T}{\operatorname{argmax}} P\left(\mathbf{H}_{1:N}^1, \mathbf{H}_{1:N}^2|\mathbf{H}\right) &= \underset{\mathbf{H}\in T}{\operatorname{argmax}} \prod_{i=1}^{N} \eta_1 \exp\left\{-\frac{\left(\Phi\left(\mathbf{R}_i^1, \bar{\mathbf{R}}_i^1\right)\right)^2}{4\sigma_q^2}\right\} \eta_2^2 \exp\left\{-\frac{||\mathbf{T}_i^1 - \bar{\mathbf{T}}_i^1||^2}{4\sigma_e^2}\right\} \\
&= \underset{\mathbf{H}\in T}{\operatorname{argmax}} \sum_{i=1}^{N} \ln\left[\eta_1 \exp\left\{-\frac{\left(\Phi\left(\mathbf{R}_i^1, \bar{\mathbf{R}}_i^1\right)\right)^2}{4\sigma_q^2}\right\} \eta_2^2 \exp\left\{-\frac{||\mathbf{T}_i^1 - \bar{\mathbf{T}}_i^1||^2}{4\sigma_e^2}\right\}\right] \\
&= \underset{\mathbf{H}\in T}{\operatorname{argmax}} \sum_{i=1}^{N} -\frac{\left(\Phi\left(\mathbf{R}_i^1, \bar{\mathbf{R}}_i^1\right)\right)^2}{4\sigma_q^2} - \frac{||\mathbf{T}_i^1 - \bar{\mathbf{T}}_i^1||^2}{4\sigma_e^2} \\
&= \underset{\mathbf{H}\in T}{\operatorname{argmax}} \sum_{i=1}^{N} \frac{\left(\Phi\left(\mathbf{R}_i^1, \bar{\mathbf{R}}_i^1\right)\right)^2}{4\sigma_q^2} + \frac{||\mathbf{T}_i^1 - \bar{\mathbf{T}}_i^1||^2}{4\sigma_e^2} \qquad (2.17)
\end{aligned}$$

Equation 2.17 is the nonlinear least squares problem to be solved. The resulting solution is the transform between Kinect 2 and Kinect 1.

B.5 Transform Application Proof

To see why $\mathbf{H}\mathbf{H}_i^2\mathbf{H}^{-1}$ is the application of $\mathbf{H}$ to the trajectory observations of Kinect 2, consider the following two equations.

1) $\mathbf{H}_i^2 p_2 = p_2'$

2) $\mathbf{H} p_2 = p_1$

The objective is to solve for transform $\mathbf{X}$ in terms of $\mathbf{H}_i^2$ and $\mathbf{H}$ such that

$$\mathbf{X}p_1 = p_1'$$

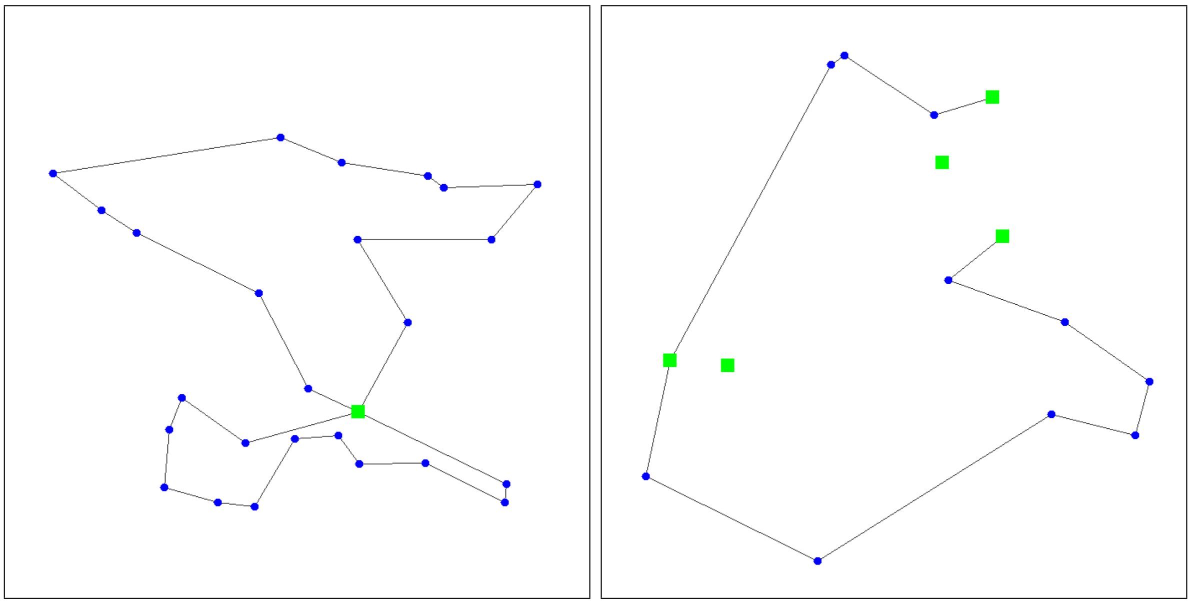
Inverting (2) and substituting it into (1) yields

$$\mathbf{H}\mathbf{H}^{-1}p_1 = p_2'$$

Applying $\mathbf{H}$ to both sides:

$$\begin{aligned}\mathbf{H}\mathbf{H}_i^2\mathbf{H}^{-1}p_1 &= \mathbf{H}p_2' \\ \mathbf{H}\mathbf{H}_i^2\mathbf{H}^{-1}p_1 &= p_1'\end{aligned}$$

C DCVRP Tour Partitioning Examples



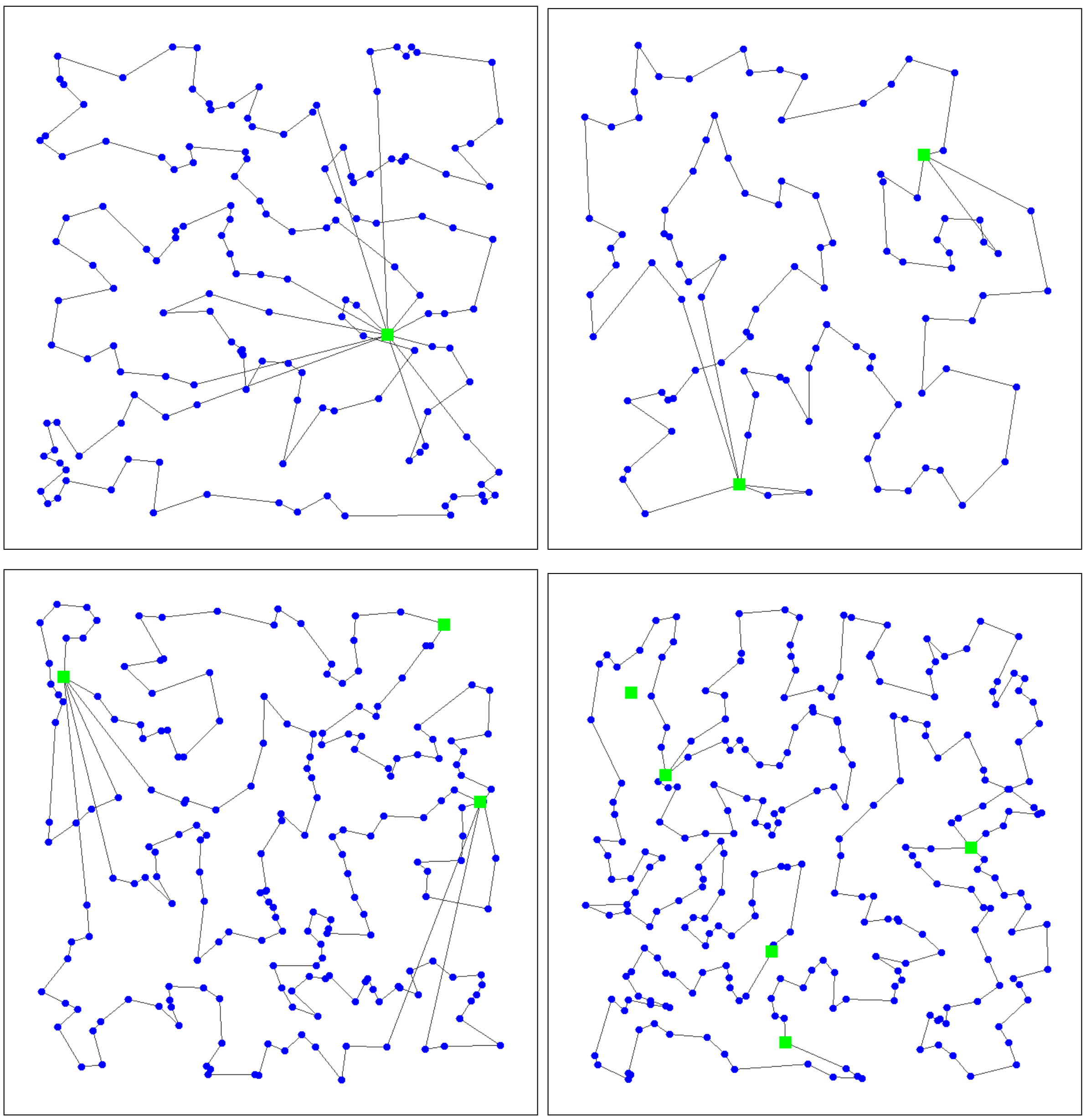


Fig. 2.22: Tour partitioning solutions to example Euclidean instances of DCVRPs

D BC Sweep Coverage Experiments for Grocery Store Test Map

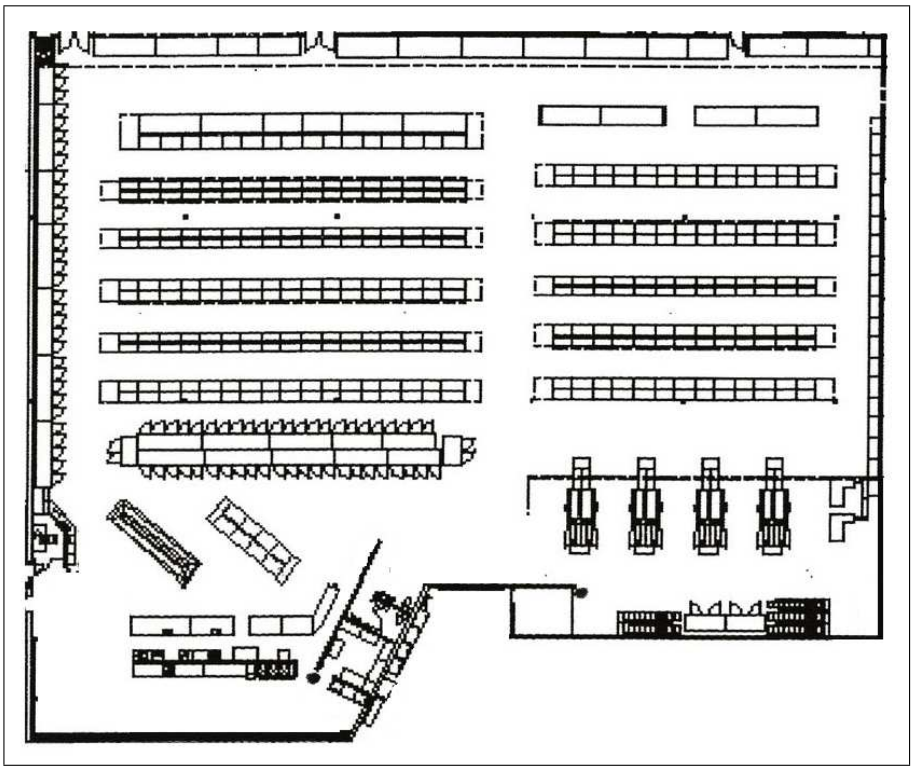


Fig. 2.23: An example convenience store layout.

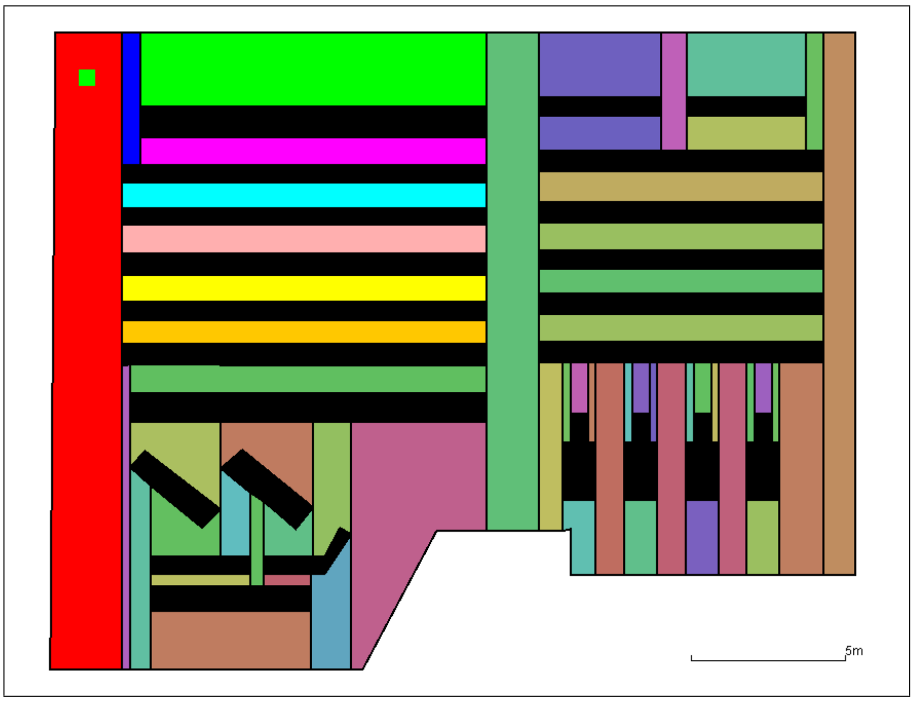


Fig. 2.24: Convenience store decomposition.

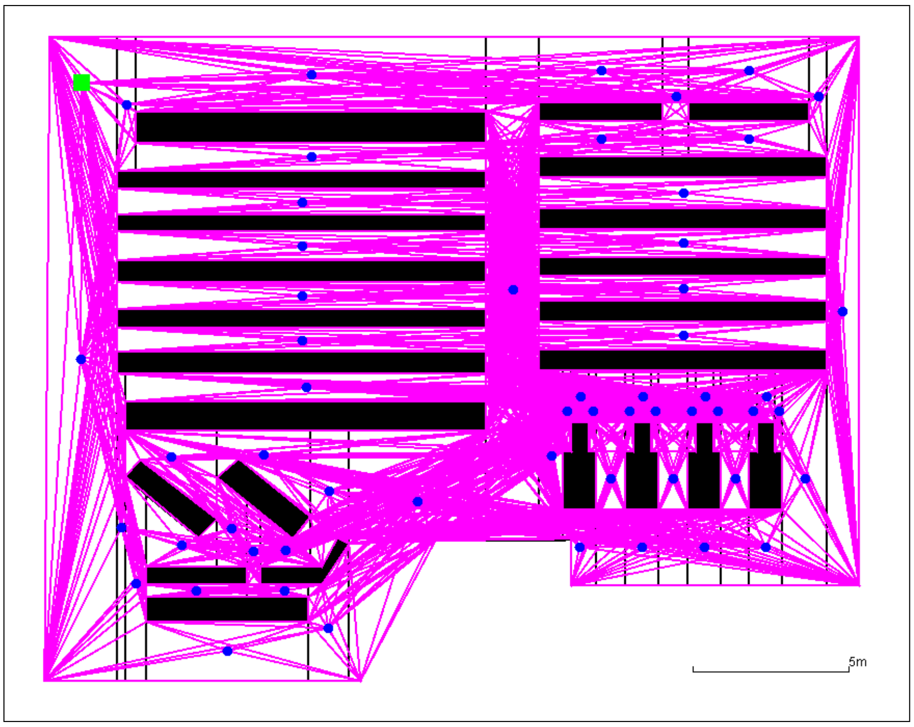


Fig. 2.25: Visibility graph for inter-cell travel.

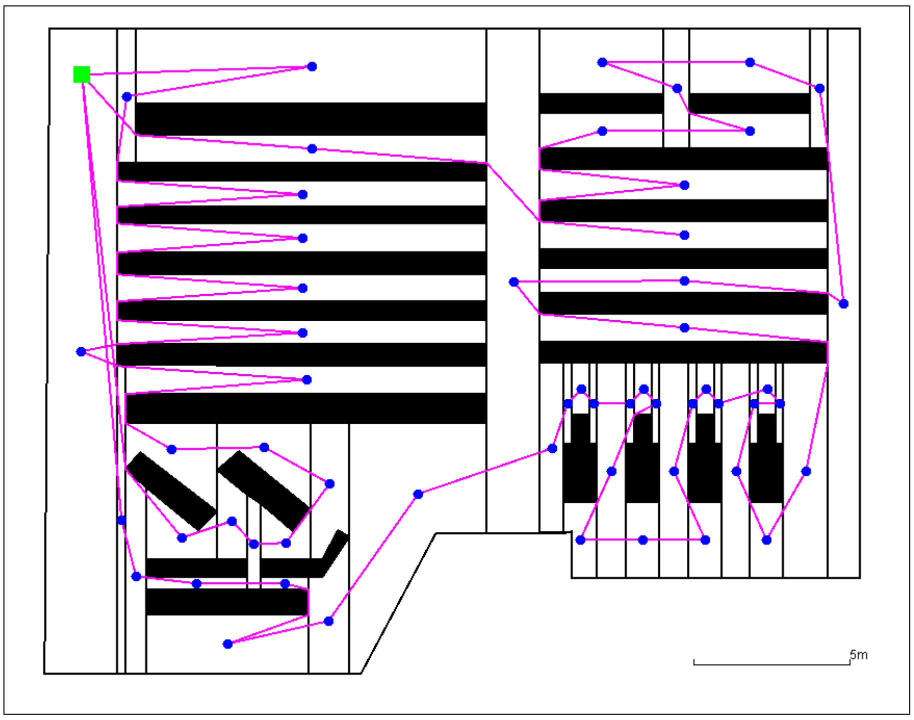


Fig. 2.26: TSP tour through cells.

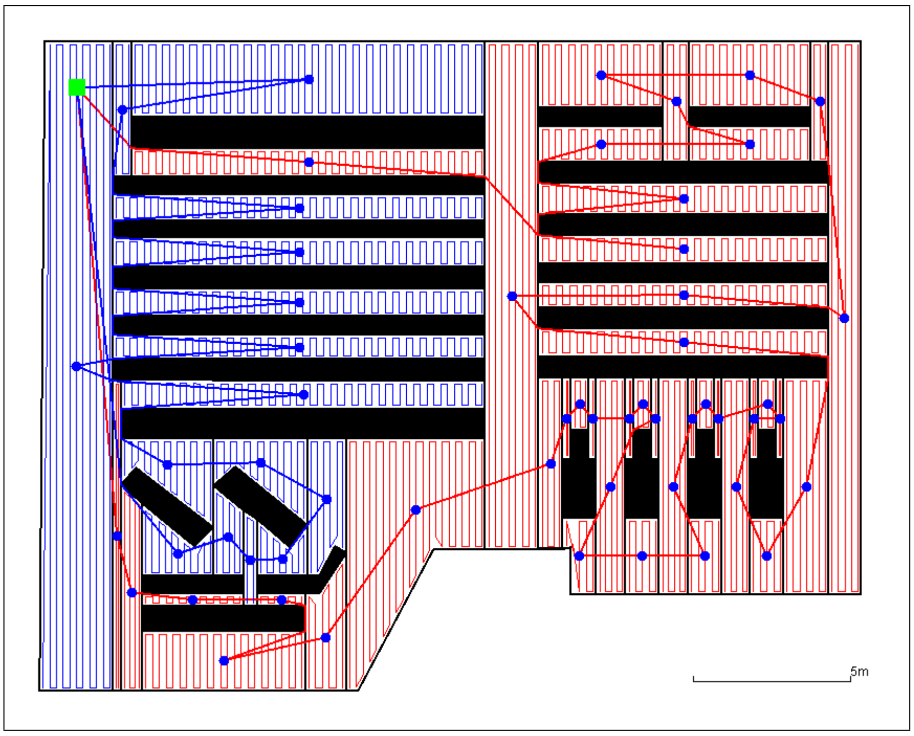


Fig. 2.27: Single depot, $\lambda = 5d_m$.

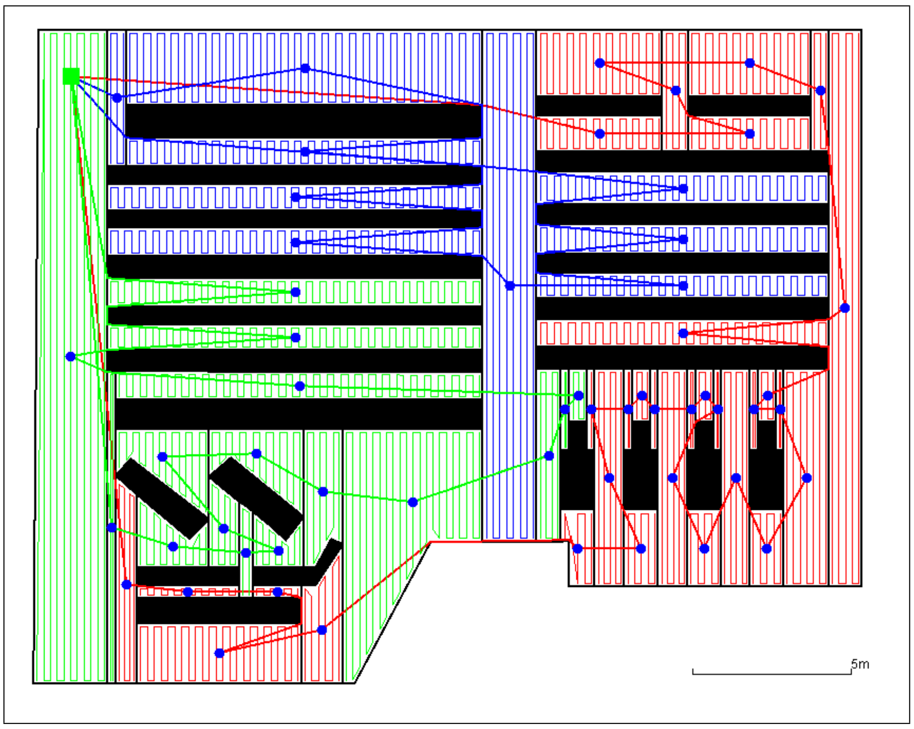


Fig. 2.28: Single depot, $\lambda = 3d_m$.

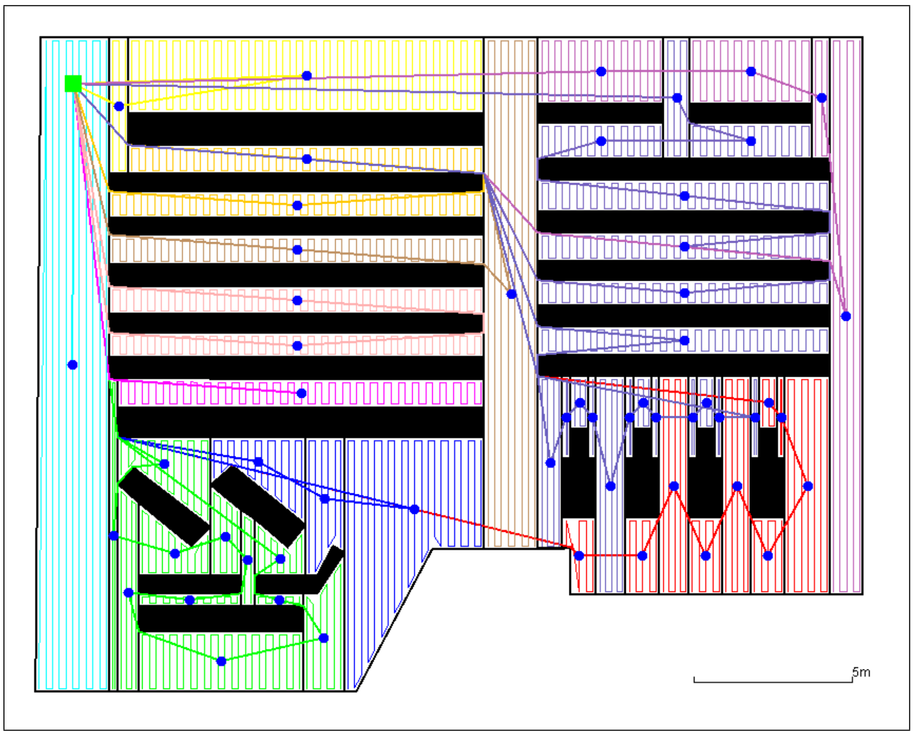


Fig. 2.29: Single depot, $\lambda = d_m$.

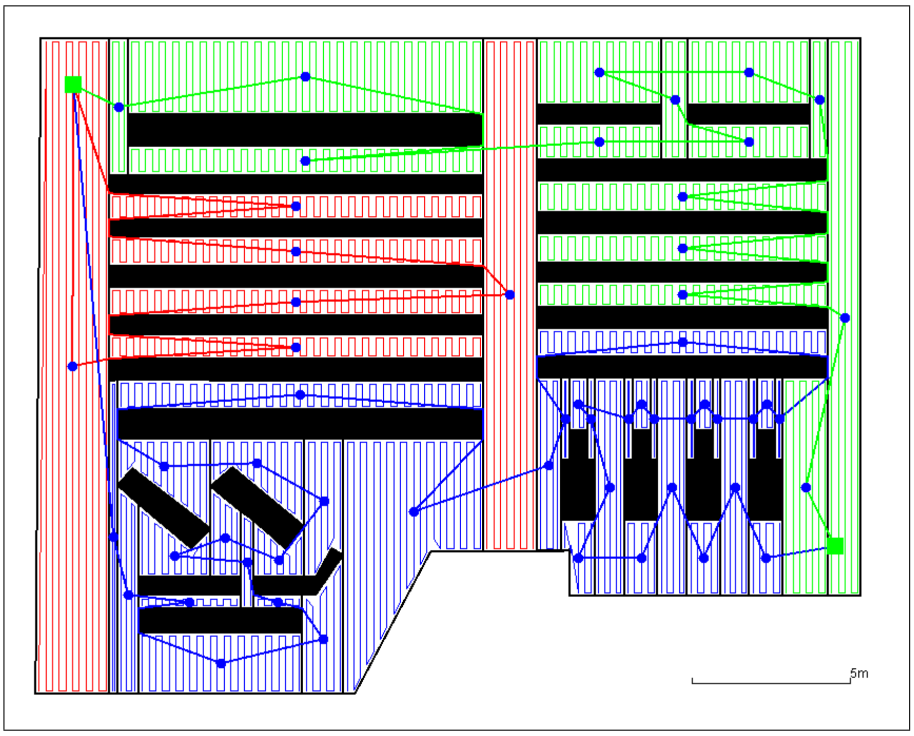


Fig. 2.30: Two depots, $\lambda = 3d_m$.

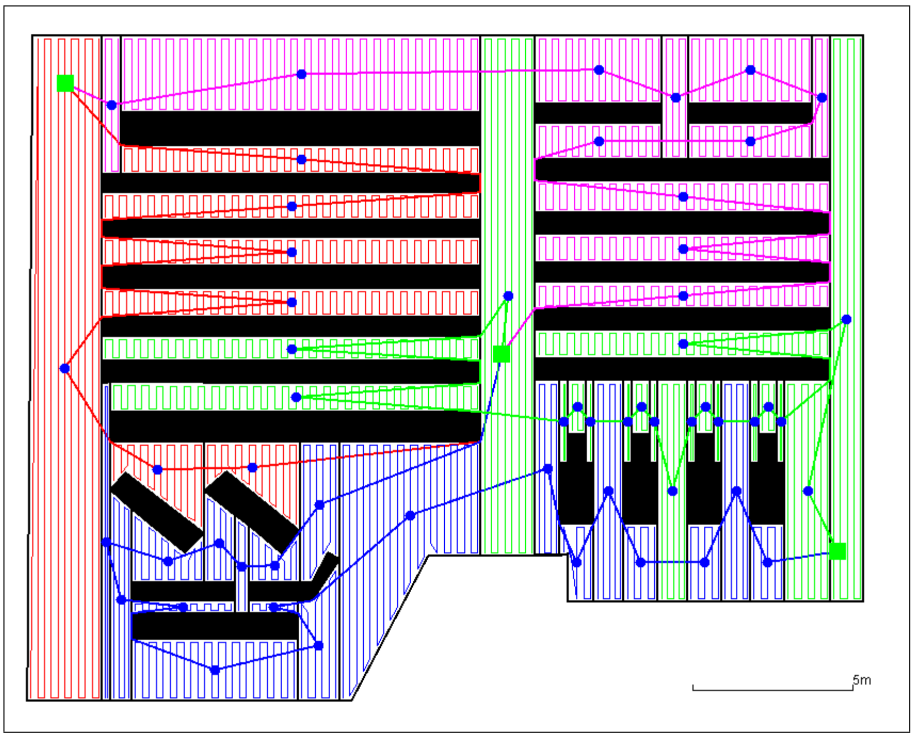


Fig. 2.31: Three depots, $\lambda = 2d_m$.

2.5 BC Sweep Timed Experiments (Full)

We present here our full results for timed experiments using the tour partitioning heuristic for the single depot case on the VRP problems given in [ABB+]. We ran the heuristic with a fuel capacity of $2d_m$, $4d_m$, $6d_m$ and $10d_m$. We compared the total fuel used on tours generated by the *BC Sweep* with a lower bound costs generated using the integer programming formulation presented in this work. Both *BC Sweep* and the integer programs were executed with a 2.40GHz Intel quad-core Core 2 Quad Q6600 processor. The integer programs were run with ILOG CPLEX 8.1 [ILO] with a time limit of 12000 CPU seconds.

Problem	BC Sweep Heuristic Value	CPU Time	Optimal/Lower Bound	Ratio
A-32-5	1341.8521	0.1220	881.6369	1.5220
A-33-5	1231.1320	0.1171	915.3952	1.3449
A-33-6	1316.6664	0.1175	1032.9360	1.2747
A-34-5	1366.7923	0.1216	952.1842	1.4354
A-36-5	1568.3419	0.1391	958.3373	1.6365
A-37-5	1146.6036	0.1385	950.1409	1.2068
A-37-6	1819.7120	0.1340	1127.3387	1.6142
A-38-5	1263.0735	0.1797	967.1293	1.3060
A-39-5	1623.7513	0.1563	1046.4135	1.5517
A-39-6	1479.4874	0.1643	1083.7777	1.3651
A-44-7	1790.3942	0.1533	1158.3592	1.5456
A-45-6	1579.7607	0.1492	1170.5393	1.3496
A-45-7	2549.1018	0.1510	1168.8896	2.1808
A-46-7	1701.9274	0.1454	1174.2022	1.4494
A-48-7	2057.2713	0.1712	1196.9806	1.7187
A-53-7	1833.5260	0.2048	1237.6864	1.4814
A-54-7	2158.0950	0.1989	1253.3963	1.7218
A-55-9	1892.4983	0.2033	1392.6237	1.3589
A-60-9	2551.8017	0.2483	1415.5110	1.8027
A-61-9	2011.9458	0.2354	1439.4879	1.3977
A-62-8	2356.2706	0.2490	1344.0424	1.6369
A-63-10	2331.9485	0.2593	1544.3866	1.5100
A-64-9	2711.8107	0.2703	1458.4742	1.8593
A-65-9	2167.9079	0.2802	1477.5247	1.4673
A-69-9	2078.0104	0.3106	1537.1212	1.3519
A-80-10	3115.0281	0.4895	1681.6676	1.8523

Table 2.2: *BC Sweep* timed experiments for benchmark problems with a fuel capacity $\lambda = 2d_m$.

Problem	BC Sweep Heuristic Value	CPU Time	Optimal/Lower Bound	Ratio
A-32-5	976.7994	0.1200	888.6141	1.0992
A-33-5	965.5320	0.1202	927.1183	1.0414
A-33-6	1098.7139	0.1187	1037.4614	1.0590
A-34-5	1050.2624	0.1234	948.9164	1.1068
A-36-5	1085.5080	0.1233	949.9573	1.1427
A-37-5	1001.8634	0.1282	951.1124	1.0534
A-37-6	1225.2856	0.1394	1098.8713	1.1150
A-38-5	1024.1601	0.1561	974.3002	1.0512
A-39-5	1071.5750	0.1562	1054.3126	1.0164
A-39-6	1238.3060	0.1612	1075.6960	1.1512
A-44-7	1327.7768	0.1511	1160.4964	1.1441
A-45-6	1241.2875	0.1792	1170.1926	1.0607
A-45-7	1416.5897	0.1612	1156.4692	1.2249
A-46-7	1360.7620	0.1765	1170.5404	1.1625
A-48-7	1412.1144	0.1617	1196.5563	1.1801
A-53-7	1395.2306	0.2284	1238.0001	1.1270
A-54-7	1525.9796	0.2026	1250.6669	1.2201
A-55-9	1520.7633	0.1999	1396.6109	1.0889
A-60-9	1743.0462	0.2426	1415.3698	1.2315
A-61-9	1619.2133	0.2554	1438.0602	1.1260
A-62-8	1644.5083	0.2532	1347.0455	1.2208
A-63-10	1797.7477	0.2648	1539.8955	1.1674
A-64-9	1811.8566	0.2593	1452.7961	1.2472
A-65-9	1689.8584	0.2781	1475.4044	1.1454
A-69-9	1711.2556	0.3321	1539.1846	1.1118
A-80-10	2044.0104	0.4428	1684.3730	1.2135

Table 2.3: *BC Sweep* timed experiments for benchmark problems with a fuel capacity $\lambda = 4d_m$.

Problem	BC Sweep Heuristic Value	CPU Time	Optimal/Lower Bound	Ratio
A-32-5	886.5527	0.1249	877.0180	1.0107
A-33-5	904.7263	0.1317	896.6229	1.0090
A-33-6	1063.0649	0.1271	1041.1782	1.0210
A-34-5	990.1600	0.1224	946.2012	1.0465
A-36-5	990.7470	0.1291	945.7633	1.0476
A-37-5	956.6080	0.1400	936.9275	1.0210
A-37-6	1149.6745	0.1329	1103.0719	1.0422
A-38-5	988.4790	0.1517	967.9090	1.0213
A-39-5	1062.6250	0.1685	1041.5212	1.0203
A-39-6	1132.6809	0.1543	1090.7617	1.0384
A-44-7	1228.4387	0.1465	1207.3985	1.0174
A-45-6	1198.8246	0.1515	1174.2733	1.0209
A-45-7	1280.1845	0.1746	1154.2068	1.1091
A-46-7	1276.9880	0.1544	1174.2646	1.0875
A-48-7	1265.8339	0.1781	1197.1468	1.0574
A-53-7	1328.1577	0.1853	1240.5438	1.0706
A-54-7	1383.8515	0.1973	1255.6921	1.1021
A-55-9	1455.8510	0.1965	1405.7774	1.0356
A-60-9	1591.5958	0.2619	1411.4460	1.1276
A-61-9	1559.8909	0.2727	1438.9646	1.0840
A-62-8	1523.3206	0.2826	1345.4170	1.1322
A-63-10	1680.4346	0.2900	1538.9563	1.0919
A-64-9	1616.7432	0.2941	1451.7449	1.1137
A-65-9	1629.1332	0.2940	1474.7291	1.1047
A-69-9	1673.7504	0.3472	1540.5185	1.0864
A-80-10	1876.0994	0.4672	1681.0542	1.1160

Table 2.4: *BC Sweep* timed experiments for benchmark problems with a fuel capacity $\lambda = 6d_m$.

Problem	BC Sweep Heuristic Value	CPU Time	Optimal/Lower Bound	Ratio
A-32-5	886.5527	0.1249	877.1055	1.0108
A-33-5	890.7367	0.1174	882.3824	1.0095
A-33-6	1038.1970	0.1211	1017.0348	1.0208
A-34-5	959.5369	0.1191	950.1764	1.0099
A-36-5	963.8611	0.1355	922.6063	1.0447
A-37-5	932.5413	0.1239	924.1408	1.0091
A-37-6	1081.2638	0.1313	1080.3570	1.0008
A-38-5	962.4897	0.1294	950.4388	1.0127
A-39-5	1043.5868	0.1347	1017.0440	1.0261
A-39-6	1084.2549	0.1484	1074.1657	1.0094
A-44-7	1179.8530	0.1478	1163.8716	1.0137
A-45-6	1177.4215	0.1594	1167.3235	1.0087
A-45-7	1191.0457	0.1437	1160.1634	1.0266
A-46-7	1209.0902	0.1600	1168.8796	1.0344
A-48-7	1250.0388	0.1753	1196.1606	1.0450
A-53-7	1259.7351	0.2013	1235.8571	1.0193
A-54-7	1296.9474	0.1889	1250.0115	1.0375
A-55-9	1447.4495	0.2014	1401.0568	1.0331
A-60-9	1457.8535	0.2295	1423.8379	1.0239
A-61-9	1498.3937	0.2624	1441.8770	1.0392
A-62-8	1420.3737	0.2358	1343.7366	1.0570
A-63-10	1624.6901	0.2430	1523.1985	1.0666
A-64-9	1600.8198	0.2615	1578.1061	1.0144
A-65-9	1541.0332	0.2550	1476.6809	1.0436
A-69-9	1592.7001	0.3074	1538.5492	1.0352
A-80-10	1851.8863	0.4595	1684.2879	1.0995

Table 2.5: ILP timed experiments for benchmark problems with a fuel capacity $\lambda = 10d_m$.

2.6 ILP Timed Experiments (Full)

We present here our full results for the timed experiments of the integer linear programs of the benchmark problems of [ABB+]. We ran experiments on all of the problems with fuel capacities $2d_m$, $4d_m$, $6d_m$ and $10d_m$. The integer programs were executed with a 2.40GHz Intel quad-core Core 2 Quad Q6600 processor and were run with ILOG CPLEX 8.1 [ILO] with a time limit of 12000 CPU seconds.

Problem	IP Best Value	CPU Time
A-32-5	1177.4083	12000.00
A-33-5	1095.5075	12000.00
A-33-6	1246.8588	12000.00
A-34-5	1172.6370	12000.00
A-36-5	1377.0328	12000.00
A-37-5	1064.0884	12000.00
A-37-6	1606.1097	12000.00
A-38-5	1176.1386	12000.00
A-39-5	1433.1643	12000.00
A-39-6	1362.2655	12000.00
A-44-7	1575.6450	12000.00
A-45-6	1476.4414	12000.00
A-45-7	2179.2248	12000.00
A-46-7	1536.1831	12000.00
A-48-7	1689.1814	12000.00
A-53-7	1607.0092	12000.00
A-54-7	1891.8696	12000.00
A-55-9	1775.2919	12000.00
A-60-9	2211.8483	12000.00
A-61-9	1811.2861	12000.00
A-62-8	2133.1589	12000.00
A-63-10	2185.4414	12000.00
A-64-9	2023.1842	12000.00
A-65-9	1929.8371	12000.00
A-69-9	1948.1469	12000.00
A-80-10	2858.1325	12000.00

Table 2.6: ILP timed experiments for benchmark problems with a fuel capacity $\lambda = 2d_m$.

Problem	IP Best Value	CPU Time
A-32-5	927.5231	12000.00
A-33-5	927.1183*	2790.95
A-33-6	1054.3566	12000.00
A-34-5	1005.5227	12000.00
A-36-5	1049.8718	12000.00
A-37-5	951.1124*	950.36
A-37-6	1163.3761	12000.00
A-38-5	990.2259	12000.00
A-39-5	1054.3126*	7082.17
A-39-6	1137.9278	12000.00
A-44-7	1262.1916	12000.00
A-45-6	1240.8878	12000.00
A-45-7	1365.3183	12000.00
A-46-7	1220.2775	12000.00
A-48-7	1312.2947	12000.00
A-53-7	1325.4526	12000.00
A-54-7	1360.6019	12000.00
A-55-9	1472.7442	12000.00
A-60-9	1575.1471	12000.00
A-61-9	1556.5062	12000.00
A-62-8	1509.1816	12000.00
A-63-10	1716.3710	12000.00
A-64-9	1623.0203	12000.00
A-65-9	1596.7408	12000.00
A-69-9	1636.9545	12000.00
A-80-10	1966.2320	12000.00

Table 2.7: ILP timed experiments for benchmark problems with a fuel capacity $\lambda = 4d_m$.

Problem	IP Best Value	CPU Time
A-32-5	877.0180*	103.83
A-33-5	896.6229*	165.58
A-33-6	1041.1782*	9420.03
A-34-5	979.1952	12000.00
A-36-5	974.6411	12000.00
A-37-5	936.9275*	52.13
A-37-6	1123.7067	12000.00
A-38-5	967.9090*	399.46
A-39-5	1050.3523	12000.00
A-39-6	1090.7617*	3657.27
A-44-7	1159.5174	12000.00
A-45-6	1193.4752	12000.00
A-45-7	1230.3585	12000.00
A-46-7	1193.8324	12000.00
A-48-7	1235.4481	12000.00
A-53-7	1265.8045	12000.00
A-54-7	1287.6646	12000.00
A-55-9	1426.2345	12000.00
A-60-9	1471.5767	12000.00
A-61-9	1474.5887	12000.00
A-62-8	1395.5335	12000.00
A-63-10	1603.5497	12000.00
A-64-9	1518.8033	12000.00
A-65-9	1521.0789	12000.00
A-69-9	1580.6932	12000.00
A-80-10	1807.0996	12000.00

Table 2.8: ILP timed experiments for benchmark problems with a fuel capacity $\lambda = 6d_m$.

Problem	IP Best Value	CPU Time
A-32-5	877.1055*	109.11
A-33-5	882.3823*	4.39
A-33-6	1017.0348*	37.52
A-34-5	950.1764*	950.00
A-36-5	922.6063*	0.94
A-37-5	924.1407*	12.19
A-37-6	1080.3570*	17.91
A-38-5	950.4388*	25.39
A-39-5	1017.0440*	9.17
A-39-6	1074.1657*	67.20
A-44-7	1163.8716*	728.16
A-45-6	1167.3235*	104.00
A-45-7	1160.1634*	1835.26
A-46-7	1168.8796*	61.50
A-48-7	1196.1606*	101.08
A-53-7	1235.8571*	78.45
A-54-7	1250.0115*	78.58
A-55-9	1401.0567*	736.38
A-60-9	1423.8379*	11075.20
A-61-9	11456.8850	12000.00
A-62-8	1343.7370*	70.34
A-63-10	1572.6670	12000.00
A-64-9	1504.4854	12000.00
A-65-9	1504.0432	12000.00
A-69-9	1555.57591	12000.00
A-80-10	1743.7459	12000.00

Table 2.9: ILP timed experiments for benchmark problems with a fuel capacity $\lambda = 10d_m$.

Bibliography

[] Vehicle Routing Data Sets.

[ABB$^+$] P. Augerat, J.M. Belenguer, E. Benavent, A. Corberán, D. Naddef, and G. Rinaldi. Computational Results with a Branch and Cut Code for the Capacitated Vehicle Routing Problem, Research Report 949-M. *Universite Joseph Fourier, Grenoble, France*. 36, 45, 63, 68

[AC91] N R Achuthan and L Caccetta. Integer linear programming formulation for a vehicle routing problem. *European Journal of Operational Research*, 52:86–89, 1991. 37

[ACZS03] Ercan U Acar, Howie Choset, Yangang Zhang, and Mark Schervish. Path planning for robotic demining: Robust sensor-based coverage of unstructured environments and probabilistic methods. *The International journal of robotics research*, 22(7), 2003. 17

[AH94] Esther M. + Arkin and Refael Hassin. Approximation algorithms for the Geometric Salesman Problem *. *Discrete Applied Mathematics*, 55(93):197–218, 1994. 18

[AM00] Esther M Arkin and Joseph S B Mitchell. Approximation Algorithms for Lawn Mowing and Milling. *Computational Geometry*, 17(1-2):25–50, 2000. 17

[AMO] Sameer Agarwal, Keir Mierle, and Others. Ceres Solver. 3, 12

[Bea83] Je Beasley. Route first - Cluster second methods for vehicle routing. *OMEGA*, 11(4):403–408, January 1983. 24

[BRH99] Z.J. Butler, A.a. Rizzi, and R.L. Hollis. Contact sensor-based coverage of rectilinear environments. *Proceedings of the 1999 IEEE International Symposium on Intelligent Control Intelligent Systems and Semiotics*, pages 266–271, 1999.

[BV12] Joydeep Biswas and Manuela Veloso. Depth camera based indoor mobile robot localization and navigation. *2012 IEEE International Conference on Robotics and Automation*, pages 1697–1702, May 2012. 2, 5

[BV13] Joydeep Biswas and Manuela M Veloso. Localization and Navigation of the CoBots Over Long-term Deployments. *Intelligent Robots and Systems*, 32, no. 14:1679–1694, 2013. 17

[BV14] Joydeep Biswas and Manuela Veloso. Episodic Non-Markov Localization: Reasoning About Short-Term and Long-Term Features. *IEEE International Conference on Robotics and Automation*, 2014. 2

[CAD11] Gerardo Carrera, Adrien Angeli, and Andrew J. Davison. SLAM-based automatic extrinsic calibration of a multi-camera rig. *2011 IEEE International Conference on Robotics and Automation*, pages 2652–2659, May 2011.

[CARL00] Howie Choset, Ercan Acar, Alfred A Rizzi, and Jonathan Luntz. Exact Cellular Decompositions in Terms of Critical Points of Morse Functions. *IEEE International Conference on Robotics and Automation San Francisco, CA*, 3:2270 – 2277, 2000. 17, 20

[Cho01] Howie Choset. Coverage for robotics , A survey of recent results. *Annals of Mathematics and Artificial Intelligence*, 31:113–126, 2001. 17

[Chr76] Nicos Christofides. Worst-Case Analysis of a New Heuristic for the Travelling Salesman Problem. *Report 388, Graduate School of Industrial Administration, CMU*, (February), 1976. 23

[Com11] Andrew I Comport. Real-time dense appearance-based SLAM for RGB-D sensors. 2011.

[CP97] Howie Choset and Philippe Pignon. Coverage Path Planning : The Boustrophedon Cellular Decomposition. *Proceedings of the International Conference on Field and Service Robotics, Canberra, Australia*, 1997. 17, 20

[DH93] Keith L Doty and Reid R Harrison. Sweep Strategies for a Sensory-Driven, Behavior-based Vacuum Cleaning Agent. *AAAI Fall Symposium*, pages 42–47, 1993. 17

[FB81] Martin A Fischler and Robert C Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981. 5

[FC04] John Folkesson and Henrik Christensen. Graphical SLAM - A Self-Correcting Map. *IEEE International Conference on Robotics and Automation*, 1(April):383–390, 2004. 4

[FSTC01] Skbastien Fabret, Philippe Soukrest, Michel Tai, and Lionel Cordesses. Farmwork Pathplanning for Field Coverage with Minimum Overlapping. *8th International Conference on Emerging Technologies and Factory Automation*, 2, 2001. 17

[Gag93] Douglas W Gage. Randomized Search Strategies with Imperfect Sensors. *SPIE Mobile Robots VIII*, 2058(September):270–279, 1993. 17

[GGR$^+$05] Andrea Garulli, Antonio Giannitrapani, Andrea Rossi, Antonio Vicino, and Via Roma. Mobile robot SLAM for line-based environment representation. *44th IEEE Conference on Decision and Control, European Control Conference. CDC-ECC '05.*, 2005. 3, 4

[HL06] Y.C. Hu and C.S.G. Lee. Deployment of mobile robots with energy and timing constraints. *IEEE Transactions on Robotics*, 22(3):507–522, June 2006. 18

[Hub01] Daniel F Huber. Fully Automatic Registration of Multiple 3D Data Sets. *Image and Vision Computing*, 21:637–650, 2001.

[HY01] Wesley H Huang and New York. Optimal Line-sweep-based Decompositions for Coverage Algorithms. *IEEE International Conference on Robotics and Automation*, 2, 2001. 18

[ILO] ILOG, Inc. ILOG CPLEX: High-performance software for mathematical programming and optimization. 36, 46, 63, 68

[Kar11] Imdat Kara. Arc based integer programming formulations for the Distance Constrained Vehicle Routing problem. *3rd IEEE International Symposium on Logistics and Industrial Informatics*, pages 33–38, August 2011. 37

[KB85] R.V. Kulkarni and P.R. Bhave. Integer programming formulations of vehicle routing problems. *European Journal of Operational Research*, 20(September 1983):58–67, 1985. 37

[Lat91] Jean-Claude Latombe. Robot Motion Planning: Edition en anglais. *Kluwer Academic, Boston, MA*, 1991. 17

[LDN84] Gilbert Laporte, Martin Desrochers, and Yves Nobert. Two exact algorithms for the distance-constrained vehicle routing problem. *Networks*, 14(1):161–172, 1984. 37

[LHD07] Cindy Leung, Shoudong Huang, and Gamini Dissanayake. Active SLAM in Structured Environments. *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2007. 3

[LpW79] Tomfis Lozano-prez and Michael A Wesley. An Algorithm for Planning Collision-Free Paths Among Polyhedral Obstacles. *Communications of the ACM*, 22(10), 1979. 20

[LSL90] CL Li and D Simchi-Levi. Worst-case analysis of heuristics for multidepot capacitated vehicle routing problems. *ORSA Journal on Computing*, 2(1), 1990. 24

[LSLD92] Chung-Lun Li, David Simchi-Levi, and Martin Desrochers. On the Distance Constrained Vehicle Routing Problem. *Operations Research*, 40(4):790–799, 1992. 24, 26, 30

[MCaPL13] Rizwan Macknojia, Alberto Chávez-aragón, Pierre Payeur, and Robert Laganière. Calibration of a Network of Kinect Sensors for Robotic Inspection over a Large Workspace University of Ottawa. pages 184–190, 2013.

[MT03] M. Montemerlo and S. Thrun. Simultaneous localization and mapping with unknown data association using FastSLAM. *IEEE International Conference on Robotics and Automation (Cat. No.03CH37422)*, pages 1985–1991, 2003. 3

[MTZ60] C. E. Miller, A. W. Tucker, and R. A. Zemlin. Integer Programming Formulation of Traveling Salesman Problems *. *Journal of the ACM (JACM)*, 7(4):326–329, 1960. 38

[NHS07] Viet Nguyen, Ahad Harati, and Roland Siegwart. A lightweight SLAM algorithm using Orthogonal planes for indoor mobile robotics. *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 658–663, October 2007. 4

[NR] Viswanath Nagarajan and R. Ravi. Approximation algorithms for distance constrained vehicle routing problems. *Networks*, 59(2):209–214. 24

[OLT06] E. Olson, J. Leonard, and S. Teller. Fast iterative alignment of pose graphs with poor initial estimates. *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.*, (May):2262–2269, 2006. 4

[PJTN08] L M Paz, P Jensfelt, J D Tard, and J Neira. EKF SLAM updates in O (n) with Divide and Conquer SLAM. *IEEE Transactions on Robotics*, pages 1107 – 1120, 2008. 3

[Pul99] K. Pulli. Multiview registration for large data sets. *Second International Conference on 3-D Digital Imaging and Modeling (Cat. No.PR00062)*, 1:160–168, 1999.

[PVP$^+$09] Kaustubh Pathak, Narunas Vaskevicius, Jann Poppinga, Max Pfingsthorn, Soren Schwertfeger, and Andreas Birk. Fast 3D mapping by matching planes extracted from range sensor point-clouds. *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1150–1155, October 2009. 4

[RL01] S. Rusinkiewicz and M. Levoy. Efficient variants of the ICP algorithm. *Proceedings Third International Conference on 3-D Digital Imaging and Modeling*, pages 145–152, 2001.

[RVvdH09] T Rabbani, G Vosselman, and F.A. van den Heuvel. Segmentation of point clouds using smoothness constraint. *ISPRS Commission V Symposium 'Image Engineering and Vision Metrology'*, 38:1–6, 2009. 6

[SBS] Lazar Sumar and Andrew Bainbridge-Smith. Feasabliity of Fast Image Processing Using Multiple Kinect Cameras on a Portable Platform.

[SEGL05] Robert Sim, Pantelis Elinas, Matt Griffin, and JJ Little. Vision-based SLAM using the Rao-Blackwellised particle filter. *IJCAI Workshop on Reasoning with Uncertainty in Robotics*, 2005. 3

[SKPY10] Aydin Sipahioglu, Gokhan Kirlik, Osman Parlaktuna, and Ahmet Yazici. Energy constrained multi-robot sensor-based coverage path planning using capacitated arc routing approach. *Robotics and Autonomous Systems*, 58(5):529–538, May 2010. 18

[SM] Aaron Staranowicz and Gian-Luca Mariottinini. A Comparative Study of Calibration Methods for Kinect-style cameras.

[SP90] Wesley Snyder and A Pirzadeh. A unified solution to coverage and search in explored and unexplored terrains using indirect control. 3:2113–2119, 1990.

[SRD06] P. Smith, I. Reid, and a. J. Davison. Real-Time Monocular SLAM with Straight Lines. *Procedings of the British Machine Vision Conference 2006*, pages 3.1–3.10, 2006. 4

[SXS+00a] Weihua Sheng, Ning Xi, Mumin Song, Yifan Chen, and Perry Macneille. Automated CAD-Guided Robot Path Planning for Spray Painting of Compound Surfaces Ford Motor Company. *Intelligent Robots and Systems*, pages 1918–1923, 2000. 17

[SXS+00b] Weihua Sheng, Ning Xi, Mumin Song, Yifan Chen, and Perry Macneille. Automated CAD-Guided Robot Path Planning for Spray Painting of Compound Surfaces Ford Motor Company. *Intelligent Robots and Systems*, pages 1918–1923, 2000.

[TBF05] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. Probabilistic Robotics. In *Probabilistic Robotics*, chapter The GraphS, pages 337–384. The MIT Press, Cambridge, 2005. 4, 7, 9, 10

[Thr03] S. Thrun. Results for outdoor-SLAM using sparse extended information filters. *IEEE International Conference on Robotics and Automation (Cat. No.03CH37422)*, 1:1227–1233, 2003.

[Thr06] S. Thrun. The Graph SLAM Algorithm with Applications to Large-Scale Mapping of Urban Structures. *The International Journal of Robotics Research*, 25(5-6):403–429, May 2006. 4

[TIC12] Alexander J B Trevor, John G Rogers Iii, and Henrik I Christensen. Planar Surface SLAM with 3D and 2D Sensors. *IEEE International Conference on Robotics and Automation*, pages 2–9, 2012. 4

[TT] Alex Teichman and Sebastian Thrun. Unsupervised intrinsic calibration of depth sensors via SLAM.

[Wat88] CDJ Waters. Expanding the scope of linear programming solutions for vehicle scheduling problems. *Omega*, 16(6):577–583, January 1988. 37

[WS05] J. Weingarten and R. Siegwart. EKF-based 3D SLAM for structured environment reconstruction. *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3834–3839, 2005. 3, 4

[YKPS14] Ahmet Yazici, Gokhan Kirlik, Osman Parlaktuna, and Aydin Sipahioglu. A Dynamic Path Planning Approach for Multirobot Sensor-Based Coverage Considering. 44(3):305–314, 2014. 18