WPCA: THE WREATH PRODUCT COGNITIVE ARCHITECTURE

by

Anshul Joshi

A dissertation submitted to the faculty of
The University of Utah
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

in

Computer Science

School of Computing

The University of Utah

December 2016

The University of Utah Graduate School

STATEMENT OF DISSERTATION APPROVAL

The dissertation of **Anshul Joshi**

has been approved by the following supervisory committee members:

Thomas C. Henderson,	Chair(s)	**03 Aug 2016** Date Approved
Charles D. Hansen,	Member	**03 Aug 2016** Date Approved
Elaine Cohen,	Member	**03 Aug 2016** Date Approved
Preston Thomas Fletcher,	Member	**03 Aug 2016** Date Approved
Tamim Asfour,	Member	**15 Aug 2016** Date Approved

by **Ross T. Whitaker**, Chair/Dean of

the Department/College/School of **School of Computing**

and by **David B. Kieda**, Dean of The Graduate School.

ABSTRACT

We propose to examine a representation which features combined action and perception signals, i.e., instead of having a purely geometric representation of the perceptual data, we include the motor actions, e.g., aiming a camera at an object, which are also actions that generate the particular shape. This generative perception-action representation uses Leyton's cognitive representation based on wreath products. The wreath product is a special kind of group which captures information through symmetries on the sensorimotor data. The key insight is the bundling of actuation and perception data together in order to capture the cognitive structure of interactions with the world. This involves developing algorithms and methods: (1) to perform symmetry detection and parsing, (2) to represent and characterize uncertainties in the data and representations, and (3) to provide an overall cognitive architecture for a robot agent. We demonstrate these functions in 2D text classification, as well as on 3D data, on a real robot operating according to a well-defined experimental protocol for benchmarking indoor navigation, along with capabilities for multirobot communication and knowledge sharing. A cognitive architecture called the *Wreath Product Cognitive Architecture* is developed to support this approach.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I would like to sincerely thank my advisor Prof. Thomas C. Henderson, whose unwavering patience and support throughout my program has been the greatest motivation for me and helped me finish this dissertation. Our technical discussions and learning from the ground up approach has helped me learn a lot more about my field, and also helped me with generating new ideas. We have explored the unexplored and his experience, knowledge, and insights have driven this work forward, and always will.

I would also like to express gratitude towards my committee members: Prof. Tamim Asfour, Prof. Elaine Cohen, Prof. Thomas P. Fletcher, and Prof. Charles (Chuck) Hansen, for their support, guidance, and their feedback which has helped shape this work into something better.

I owe a debt of gratitude to all my colleagues in the Omega-Infinity group, Wenyi Wang, Narong Boonsirisumpun, Dr. Linda DuHadway, and Nishith Tirpankar, who, during this work, have helped me solve many problems when they seemed unsolvable.

Last but not the least, we would like to acknowledge the support, in part, by the Air Force Office of Scientific Research (AFOSR) grant FA9550-12-1-0291 (2012-2015), and the Graduate Research Fellowship Award, Graduate School, University of Utah (2015-2016) which has funded this, and other work.

CHAPTER 1

INTRODUCTION AND BACKGROUND

We have proposed innate theories of symmetry as the cognitive basis for embodied robot agents [1–3] and more recently, a specific cognitive architecture based on *Bayesian Symmetry Networks* [4, 5]. This representation builds on the framework layed out by Leyton [6, 7] wherein he proposes that the wreath product captures the notion of a specific concept which is a representation of what something is or how it works; this may capture either a specific instance of an existing thing or an abstract description of a class of related objects. For Leyton, the wreath product provides the basis for concept representation, where a wreath product is a group formed by a splitting extension of the direct product of the fiber group which is acted on by a control group (usually a cyclic group – see details below) and is derived from related perception and actuation. The distinctive feature of this representation is that it is based on how the set of features comprising the object to be represented is generated – it is a generative theory of shape. Thus, the actuation control sequences are part of the description of an object and determine the control group hierarchy. This is important because objects are expressed in terms of the specific embodiment of the robot agent perceiving them.

In recent years robotics researchers have understood the importance of developing cognitive abilities of robots, rather than explicitly programming the robots with the knowledge and algorithms to process that knowledge for achieving results. This is evident from the amount of research that has been devoted to developing the most generalized and robust cognitive framework and internal representation that works for every possible context the robot operates in (see [8–12]). In the process of building cognitive systems, researchers have defined and used various paradigms of cognition to build intelligent agents, with each paradigm having its own advantages and disadvantages (see Vernon et al. [12] for an excellent overview of cognitive architectures). At a high level, cognitive systems can be classified into cognitivist and emergent systems paradigms. Cognitivist approaches rely on explicit symbolic knowledge built into the system by the designer, which may then learn and add to that knowledge and exhibit behaviors which are operations on this knowledge. Embodiment is not a requirement in cognitivist approaches. Emergent systems, on the other hand, are embodied and rely on self-organization where the intelligent agent constructs its world as a result of operations in that world. Most of today's practically used robots are built on cognitivist approaches, and although theoretically more powerful than cognitivist approaches, emergent systems have limited and incomplete practical implementations compared to cognitivist systems which have better capabilities at present. Vernon, however, claims that "cognitivist systems are brittle," the main reason being that their cognitive abilities are fixed and determined by human beings who design the system, and the knowledge and processes at the outset are not their own, but the system designer's. This proves to be a problem when essential capabilities such as generalization, creativity, anticipation, and learning are desired in or required of the agent.

An argument for actuation being an inseparable part of perception is made by Noë [13], who proposes that perception is not something that happens to us, but it is something we do. The effects of movement on sensory stimulation are necessary to understand perception. Noë suggests that perception is not just a process *in the brain* that constructs an *internal representation* of the world - which we also propose as part of this framework - but a "skillful activity on the part of the animal as a whole." In support he gives examples of "experiential blindness" where the mere presence of visual stimulus does not result in "seeing" things; seeing requires understanding the movement and thought relevant to a particular visual stimulus. Noë also provides case studies (e.g., experiential blindess) and test subject testimonies where it is evident that sensorimotor skills are necessary for forming valid perceptual patterns in response to environmental changes. Noë [13] provides a philosophical and psychological argument for the primary role of actuation in perception. He states (p. 102):

> The sensorimotor dependencies that govern the *seeing* of a cube certainly differ from those that govern the *touching* of one, that is, the ways cube appearances change as a function of movement is decidedly different for these two modalities. At an appropriate level of abstraction, however, these sensorimotor dependencies are isomorphic to each other, and it is *this* fact – rather than any fact about the quality of sensations, or their correlation – that explains

> how sight and touch can share a common spatial content. When you learn to represent spatial properties in touch, you come to learn the transmodal sensorimotor profiles of those spatial properties. Perceptual experience acquires spatial content thanks to the establishment of links between movement and sensory stimulation. At an appropriate level of abstraction, these are the same across the modalities. We can illustrate this by means of a simple example. If something looks square, then one would need to move one's head in characteristic ways to look at the corners. One would have to move one's hands *the same way* at the appropriate level of abstraction to feel each corner.

Note that Noë's discussion may involve more of the human actuation system (e.g., neck, torso, etc.) than we exploit in the character recognition problem.

Asada [8] stresses the idea of "physical embodiment" of an agent which specifies the constraints of interaction between an agent and its environment that are responsible for generating the rich contents of its processes and consequences. In the various stages of human development (embryo, fetus, and so on), interactions with the physical environment determines information structuring (sensorimotor representation, etc.). Furthermore, Asada asserts that self-exploration plays an important role in cognitive developments in infancy, in which a systematic exploration of perceptual consequences resulting from self-produced actions provides a "sense of the bodily self."

Bajcsy [14] defines *active perception* as a study of modeling and control strategies for perception. In her argument she states that perception is not passive but active. Perception is the result of active exploration, probing and searching; percepts do not simply fall onto sensors. The pupils adjusting to the illumination, lens adjusting focus depending on distance, eyes converging or diverging, moving the head to get a better view, are all examples of active perception. An example of active sensing using a passive sensor by actively changing its state parameters is depth from focus (or defocus) where the focal length is adjusted, and relative depths of pixels are approximated based on how pixels go in and out of focus. However, Bajcsy differs from Aloimonos [15], regarding active perception as a scientific paradigm, in that she emphasizes it more as a study of *modeling* and *control strategies*. For example, active perception is crucial to the modeling of sensors, objects, environment, and the interaction between them for various processes like manipulation, mobility, and recognition.

Mishra, Aloimonos et al. [15, 16] have used the active vision paradigm in robotics for image segmentation, where they use object fixation by an active observer (e.g., a moving robot who fixates on one point on an object) for segmenting that particular object from the scene. In their work, they discover depth boundaries of objects by using various monocular cues (e.g., color, texture) and binocular cues (stereo disparities and optical flow) and fixating on a point. This fixation point lies inside a particular region of interest which might be the object, and segmentation of this object is the problem they solve. The problem of segmentation is modeled as finding the enclosing (closed) contour around the fixation point. This is achieved by first converting the image from Cartesian to log-polar space (the fixation point is the pole) where the probability of a pixel being near the depth edge defines its brightness, and the log-polar image is used to normalize different contours within the same object around the fixation point. This boundary edge map is then improved, first using a graph cut-based approach, and then using binocular cues: a disparity stereo map and optical flow field at multiple times steps.

Another example of active vision in robotics is given by Allen et al. [17] where real-time servoing of a robot arm is performed. Input is provided using multiple calibrated, unregistered cameras, which robustly calculate the 3D position of a moving object using optical flow, and this position is used by a kinematic control algorithm to move the robotic arm, thus tracking the moving object. Tarabanis et al., in [18], give a survey of sensor planning ("active perception" by Bajcsy's definition) in computer vision. Tarabanis defines sensor planning as developing strategies for automatically determining sensor parameter values that achieve a certain vision task (e.g., object recognition, object manipulation, etc.), given information about the environment, to a certain degree of satisfaction. He argues that sensing strategies are especially needed for dynamic situations (where the environment, the robot, or the sensor settings might change over time), as well as for sensor systems that perform object segmentation, since a single sensor configuration might not be sufficiently informative.

An approach similar in spirit to ours is the Object-Action Complex (OAC) methodology described in [19] which is the basis for symbolic representations of sensory-motor experience and behaviors. The description of physical interactions of the robot with the environment are encoded as a linkage between perceptual aspects and actuation actions. OAC's provide a framework for representing interactions, both low-level reactive behavior and high-level deliberative planning. Following a certain set of design ideas, OAC's attempt to represent models of interactions with the world, e.g., those that might be mediated through a low level control program, thus providing a framework for formalizing actions and their effects in artificial cognitive systems.

Leyton [7] proposes a cognitive framework that incorporates symmetry-based processes that, by acting in the environment, are able to generate concepts from the relation between actuation and perception. For us, these concepts form the knowledge base of the robot, and it builds affordances (goal-directed interactions with the environment) in terms of this knowledge. Leyton proposes that,"The human perceptual system is structured as an n-fold wreath product $G_1 \wr G_2 \ldots \wr G_n$," where G_i is a specific type of group, and $\wr$ is the wreath product symbol. Our aim is to develop a capable and robust cognitive system based on Leyton's cognitive representation.

We propose to advance robot cognition by developing an operational wreath product based cognitive architecture. In addition, the wreath product should not only provide an abstraction mechanism for a robot to see similarity in structures, but will also allow robot knowledge sharing: the wreath product provides the abstraction of a concept, and annotations provide the mapping to specific actuator systems.

The thesis of this work is that:

> Wreath product group representations enable concept formation in an embodied agent, and together with Bayesian methods to characterize uncertainty, provide the basis for a robust robot cognitive architecture.

In order to demonstrate this, we make the following contributions:

1. The development of a combined sensorimotor representation for 2D and 3D shapes based on an implementation of the wreath product, and algorithms for their discovery and manipulation in real world data.

2. The development of a Bayesian characterization of the uncertainty in the wreath product representations extracted from sensorimotor data, including algorithms for the propagation of uncertainty from sensors and actuators through to the shape representations, as well as the exploitation of uncertainty for planning purposes.

3. The development of a wreath product based cognitive architecture in which plans may be encoded as wreath products, affordances are captured by these, and broader

issues relating to robot intelligence can be explored, including the provision of a universal language for 2D and 3D shape representation and robot motion planning.

4. The validation of the above theory in terms of 2D text classification and a 3D environment mapping and navigation benchmark application.

This dissertation is divided into chapters as follows. Chapter 2 discussed the idea of wreath product both in the context of Leyton's cognitive concept as well as an entity within the context of a mathematical group, followed by detailed examples of how this translates into usable concepts that exploit actuation and perception as a bundle. Thereafter, we present an approach using context-free grammar to represent wreath products for the purpose of knowledge sharing, followed by high-level mechanisms to discover wreath products from data. We conclude Chapter 2 by a detailed analysis of characterizing and exploiting uncertainty associated with real data.

Chapter 3 delves into more details of the design of our cognitive architecture - Wreath Product Cognitive Architecture (WPCA) - by a high-level discussion of its various working blocks, and how each of them fits into our broader idea of creating a practical cognitive architecture. Each block is described later in the chapter in significant detail, along with a characterization of the kind of 3D depth data we deal with, and the methodologies used to process them. Various algorithms implemented as parts of different blocks of the framework are also presented and discussed in detail, along with the 3D geometric analysis of the data. Plan library that contains detailed high-level plans for achieving high-level goals are also described in detail.

Chapter 4 presents results obtained while applying our wreath product analysis methods on 2D data, specifically, the problem of character classification in engineering drawing. A novel system called the Enhanced Non-Deterministic Analysis System (ENDAS) was developed as a high-level agent-based system that is used to classify characters in engineering drawing. Wreath Product Constraint Sets (WPCS) are developed for representing various characters using wreath products. The concept of exploiting the actuation information in a WP is also explored in this chapter. We show with experimental results that we achieve high character classification accuracy using this approach.

Chapter 5 deals specifically with 3D depth data analysis on a robot operating in a real indoor environment. We have demonstrated good localization performance using WPCA and this performance is discussed along with the benchmarks compared against. High-level plans and algorithms used to achieve this performance is described in detail. Multirobot communication scenarios using WP representation are presented which extends WPCA to multiple robots operating in the same environment.

Chapter 6 concludes the dissertation with a brief discussion of goals achieved so far, as well as of a strong potential for expanding the work in the future in multiple areas.

CHAPTER 2

WREATH PRODUCT REPRESENTATIONS, RECOVERY FROM DATA AND UNCERTAINTY CHARACTERIZATION

This chapter elaborates on the high-level concept of a wreath product and its application to shape analysis and generative structure. The high-level mechanism for converting a wreath product to a linear representation of strings using context-free grammar is also discussed. Mechanisms for wreath product discovery and uncertainty characterization are also explained.

2.1 Wreath Product Representation

Leyton's main idea is that wreath products provide a generative representation of shape by describing how the point set defining a shape is generated by taking a single point and moving it to all other points in the shape. The movements are characterized by group actions on sets, and thus, a major role is played by symmetry. We propose to provide an operational implementation of this theory in which the group operators correspond to actuation processes and the point sets to perception data (e.g., 2D or 3D point sets from camera images or range finders).

Weyl [20] defines symmetry as (pp. 44–45):

> Given a spatial configuration, $\mathcal{F}$, those automorphisms of space which leave $\mathcal{F}$ unchanged form a group, τ, and this group describes exactly the symmetry possessed by $\mathcal{F}$.

Much work has been done on finding symmetry in data (see [21]). The important point for us is that shape is described not only in terms of the set of points (or their automorphisms) comprising the shape, but also in terms of a sequence of actions which generates the shape. For example, for 2D and 3D shapes, this includes translation, rotation and reflection. Wreath products allow multiple distinct representations of the same shape (set of points) generated in different ways. This fits well with our previous work on representations for robot knowledge sharing (see [22]) in that wreath products provide an abstract robot control scheme description which can be mapped to diverse actuation systems (either within the same robot or across distinct robot platforms). Thus, the symmetries exploited here are not so much those invariants of a shape (i.e., automorphisms), but those which describe possible actuator motions for the robot.

Here we give the wreath product's high-level significance for shape generation. A wreath product, G, is a group formed by the semidirect product of two subgroups, F (or more precisely a direct product of copies of F) and C. C is the *control* group whose action is to map copies of F, the *fiber* group, to each other.

If $\prod_1^n F$ is a set of copies (at different spatial locations) of some set of points, and C defines a permutation between these sets ($|C| = n$), then $\prod_1^n F \rtimes C$ is the wreath product, G, of F acted on by C; i.e., $G = F \wr C$. To obtain a wreath product group, G, the direct product of n copies of F serves as the normal subgroup to G. Consider as an example the wreath product representation of the outline of a square. For example, $e \wr \mathcal{Z}_2 \wr \Re \wr \mathcal{Z}_4$ indicates that:

1. e is a point (the meaning is a specific point in a given coordinate frame with the group consisting of just the identity element acting on the point; see Figure 2.1).

2. $\Re$ specifies a (continuous Lie group) translation of the point along a specific line in space. In practice, it is represented by a finite set of translations for a specific set of edge points (see Figure 2.2).

3. $\mathcal{Z}_4$ indicates the set of 90-degree rotations about the center of the square and which act on the line segments to move them onto each other. Note that without additional annotations in the implementation, the meaning assigned to this group is abstract; annotations apply to a specific instance of the shape (see Figures 2.3(a)–2.3(c)).

4. $\mathcal{Z}_2$ acts as the characteristic function defined in such a way as to select points in the shape, and reject points not in the shape (see Figure 2.3(d)).

Each new group to the right side of the wreath symbol defines a control action on the group to the left, and thus provides a description of how to generate the shape. For example, to draw a square, start at the specified point (details of the x,y values of the point, etc., are left to annotations in the representation), translate some length along a straight line, then rotate 90 degrees and draw the next side, and repeat this two more times. Figure 2.4 depicts the process mentioned in the previous paragraph, of how control flows from the rotation group, $\mathcal{Z}_4$, down to copies of the translation group, $\Re$.

Although this may work well in theory, major difficulties arise when attempting to implement a working system that can use sensor and actuator data to describe arbitrary objects in a scene. Figure 2.5 shows an example result of producing WP representations for the various frequently occuring objects in an indoor setting (planes, lines, and points). Discovery of cylinder and sphere is discussed elsewhere [4]. Results on the extraction of wreath product representations in 2D and 3D are given below.

In order to understand a wreath product we briefly explain the concept of a *semidirect product* in group theory, that underlies the concept of a wreath product. Consider a homomorphism ϕ given by $\phi_h(n) = hnh^{-1}$ for all $n \in N$ and $h \in H$, where H and N are groups, and H is a group that acts on N by conjugation. For each $h \in H$, conjugation by h is an element of $Aut(N)$ (automorphism group of N).

Given two groups N and H, and a group homomorphism ϕ from H into $Aut(N)$, $N \rtimes_\phi H$ denotes the *semidirect* product of N and H with respect to ϕ and satisfies the following:

1. $N \rtimes_\phi H$ contains elements from $N \times H$

2. Group operation $\star$ of $N \rtimes_\phi H$ is defined as: $(n_1, h_1) \star (n_2, h_2) = (n_1 \phi_{h_1}(n_2), h_1 h_2)$, where $n_1, n_2 \in N$ and $h_1, h_2 \in H$.

Now consider a group L where L consists of the direct product of $k = \mid H \mid$ copies of N, i.e., $L = N_1 \times N_2 \times ...N_k$. **The *wreath product*, $\mathbf{G} = \mathbf{N} \wr \mathbf{H}$, is formed by the *semidirect product* of L and H. Thus $\mathbf{G} = \mathbf{N} \wr \mathbf{H} \equiv \mathbf{G} = \mathbf{L} \rtimes \mathbf{H}$.**

The creation, manipulation and sharing of wreath products requires a clear and unambiguous representation. To this end, we provide a context-free grammar, G_{wp}, and a (nondeterministic) pushdown automaton, P_{wp}:

$$G_{wp} = (V, \Sigma, R, S)$$

where

$$V = \{W, F, C, B, N, D\}$$

$$\Sigma = \{\wr, \Re, \mathcal{O}, \mathcal{Z}, \Re^2, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$$

$$S \equiv W$$

$$R = \{W \to FC, F \to B\wr, B \to C, B \to FC, C \to e,$$

$$C \to \Re, C \to \mathcal{O}, C \to \mathcal{Z}N, N \to ND,$$

$$C \to \Re^2, W \to e,$$

$$N \to 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9,$$

$$D \to 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9\}$$

G_{wp} produces strings of the form $G_1 \wr G_2 \wr \ldots \wr G_n$, where G_i is one of the groups $\{e, \Re, \Re^2, \mathcal{Z}_n, \mathcal{O}\}$; e is the identity group, $\Re$ is the 1D translation group, $\Re^2$ is the 2D translation group, $\mathcal{Z}_n$ is the cyclic group of order n, and $\mathcal{O}$ is the continuous 2D rotation group. P_{wp}, the PDA corresponding to G_{wp}, is given in Figure 2.6. Examples of wreath products and their corresponding point sets are given in Table 2.1. G_{wp} and P_{wp} provide the minimal basis for sharing wreath product representations.

Unfortunately, this syntactic description does not completely capture the meaning of the wreath product. For example, e as the identity group represents a specific point in some coordinate frame, and this information must be added to the syntactic representation. This is done using an attribute grammar (see [23] for details of attribute grammars). Each rewrite rule has an associated set of functions describing what values are to be assigned to the nonterminal symbols (it is assumed that the terminal symbols are assigned a value during the scanning process). Table 2.2 gives the value functions associated with G_{wp}.

This representation is well-suited to efficient parsing, storage and sharing. However, for other purposes, namely during data analysis and for uncertainty specification, an alternative form of the wreath product tree proves more effective. The tree form is semantically equivalent to the attribute grammar, G_{wp}, but makes the normal subgroup direct product explicit. It is straightforward to convert between the string and tree representations.

2.1.1 Restrictions to the Representation

The continuous Lie groups, $\Re$ and $\mathcal{O}$, are only used to depict the ideal line and circle. Any specific line segment or circle recovered from sensory data is constructed from a finite set of points in the data. To express this, we use T to represent a finite version of translation, and $\Box$ for $\mathcal{O}$. For example, if $P = \{(11,10),(12,10),\ldots(20,10)\}$ is a set of edge pixels in an image, then T is the group of translations formed as follows:

1. Given n points, $\bar{e}_i$, $i = 1 : n$, they are ordered so that $\bar{e}_i \times \bar{e}_j$ all point in the same direction.

2. $T = \{\bar{e}_i - \bar{e}_1\}, i = 1 : n$

3. The $\mathcal{Z}_2$ group defines in the fixed coordinate frame of T, the points that are in the line segment.

Discrete circles (i.e., those arising from a finite number of samples) are handled in a similar manner.

2.2 Wreath Product Discovery

Figure 2.7 describes the basic mechanisms involved in mapping data to a specific wreath product representation. A standard symmetry analysis on a set of points (in a 2D or 3D image) results in the detection of translational, rotational and reflective symmetries which can be structured via, e.g., a dihedral group, D_4 (see our work on this in [1–3]). However, this approach is agnostic as to how the shape was produced. The WPCA requires that the symmetries result from a sequence of actions. In the figure, the corners (e_1,e_2,e_3,e_4) are detected, then the edges of the square are found as translations of these points (e.g., $e_1 \rightarrow e_2 \equiv R_1$, $e_2 \rightarrow e_3 \equiv R_2$, etc.), and finally, the complete set of points in the square shape is found to result from the action of the four rotations (r_0,r_{90},r_{180},r_{270}) acting on the edge segments. To determine the associated actuation sequence needed to produce the shape, the motor sequence of some set of actuators on the robot need to be aligned with this specific shape; e.g., the motor controls necessary to move the center pixel of the robot's camera to each corner would suffice.

As another example, consider a 3D cube which also has multiple wreath product representations corresponding to distinct shape generation sequences. We have been working with a Kinect sensor to obtain RGB-D data and Figure 2.8 shows both a 1-face and a 3-face view of the cube (in this case, a footstool found in the office). Figure 2.9 shows the geometry and wreath product tree for the generative sequence we use in analyzing cubes in indoor scenes. The $\mathcal{Z}_2$ group describes the reflective symmetry between parallel faces; the $\mathcal{Z}_3$ group describes the rotational symmetry between the pairs of faces. The resulting wreath product is then $e \wr \Re^2 \wr \mathcal{Z}_2 \wr \mathcal{Z}_3$ (we only describe down to the planar face level in the figure).

Next, we describe the results of applying wreath product discovery algorithms to 2D and 3D shapes.

2.3 Uncertainty Characterization in Wreath Product Trees

Although wreath products are typically described as a sequence of strings, it is convenient to interpret the wreath product as a tree structured Directed Acyclic Graph (DAG), and this allows a straightforward association of a Bayesian network. To define a tree for a wreath product, $F \wr C$, take C as the root, and its children are the n copies of F in the normal subgroup of G.

Figure 2.10[1] shows how a wreath product can be developed into a *wreath product tree* which is defined inductively as follows:

- The control group on the right end of the wreath product is the root node, and its children are the components of the direct product normal (fiber) subgroup.
- Root of each child node is one of the G(F)s, and its children are formed recursively.

We propose a methodology to construct the BN associated with the wreath product, as well as a method for initializing the Conditional Probability Tables (CPT's). For example, consider the analysis of Kinect data for a planar surface, and how uncertainty can be ascribed to a planar hypothesis. Figure 2.11 (a) shows the plane fitted to a set of planar points that defines its reference normal (in red) and (b) shows the distribution of angular deviation of normal belonging to planes fit to local neighborhoods, with respect to the reference normal. This data allows the specification of the probability that a local normal is aligned with the correct normal.

Given an abstract representation (e.g., for a cube) as shown in Figure 2.9 (b), we attach a Bayesian framework to characterize the uncertainty in each element of the wreath product

[1]This network was constructed using the AgenaRisk software which accompanies [24].

(e.g., as shown in Figure 2.12). In order to populate the conditional probability tables (CPT's) which define the complete joint probability distribution, it is necessary to initialize certain tables based on sensorimotor data, and to propagate that uncertainty in a correct manner through the tree. For example, the $\mathcal{Z}_3$ group has a prior 5% probability since this is statistically the frequency of cube corners in the indoor environment. Similarly, gathering statistics from the environment leads to values for the other CPT's: if the $\mathcal{Z}_3$ symmetry is false, then there is a 30% conditional probability of the existence of a $\mathcal{Z}_2$ symmetry, otherwise an 80% conditional probability. The conditional probability for flat faces is 70% with no known $\mathcal{Z}_2$ symmetry, otherwise, 95%. Figure 2.12 shows the probabilities with no evidence asserted. On the other hand, if three planar faces (i.e., $\Re^2$) are found that have a $\mathcal{Z}_3$ symmetry, then Figure 2.13 shows the change in probabilities. The probability for the unseen planar faces rises to 91%. The Bayesian Wreath Product Tree thus offers a clear cognitive benefit to an agent in terms of understanding and predicting the detailed geometric structure of the environment. This type of information may not be readily available to a robot without this cognitive structure.

In practice, this works reasonably well; the left side of Figure 2.14 shows Kinect data for a 3-face view of a cube shape, and the right hand side shows the three sets of similar $\Re^2$ planar data found (i.e., the faces are the result of the 2D planar translation relation between local planar fits). Each fit carries its own likelihood (e.g., the least squares error). This data is then used to compute a representative normal for each planar face, and then the existence of the $\mathcal{Z}_3$ symmetry can be characterized by the error found in the planar face fits when rotated through $\frac{2\pi}{3}$ about the mean vector of the three planar face vectors. A similarity measure can be computed and works fairly well; see Figure 2.15 where (a) shows the similarity measure (best match of three original normals with rotated versions of themselves), and (b) shows the trajectories of the three normal endpoints under the rotation.

More detailed information on wreath product representation and recovery is given in Chapters 4 and 5 which describe the 2D and 3D applications.

Table 2.1. Shapes and corresponding wreath product strings.

Point Set	*Wreath Product String*
circle	$\mathcal{O}$
square	$e \wr \mathcal{Z}_2 \wr \Re \wr \mathcal{Z}_4$
triangle	$e \wr \mathcal{Z}_2 \wr \Re \wr \mathcal{Z}_3$
cube	$e \wr \mathcal{Z}2 \wr \Re^2 \wr \mathcal{Z}_2 \wr \mathcal{Z}_3$
plane	$e \wr \Re^2$

Table 2.2. Attribute grammar semantic functions for G_{wp}.

Rewrite Rule	*Value Functions*
$W \rightarrow FC$	$v(W) := \prod_1^n F \rtimes C$
$F \rightarrow B$	$v(F) := \prod_{i=1}^n F$; n= $\mid C \mid$
$B \rightarrow C$	$v(B) := v(C)$, a specific group
$B \rightarrow FC$	$v(B) := \prod_1^n F \rtimes C$
$C \rightarrow e \mid \Re \mid \mathcal{Z}N \mid \mathcal{O}$	$v(C) :=$ value of specific group
$N \rightarrow ND$	$v(N) := 10N + D$
$N \rightarrow 1 \mid 2 \mid \ldots \mid 9$	$v(N) :=$ value of selected digit
$D \rightarrow 0 \mid 1 \mid \ldots \mid 9$	$v(D) :=$ value of selected digit

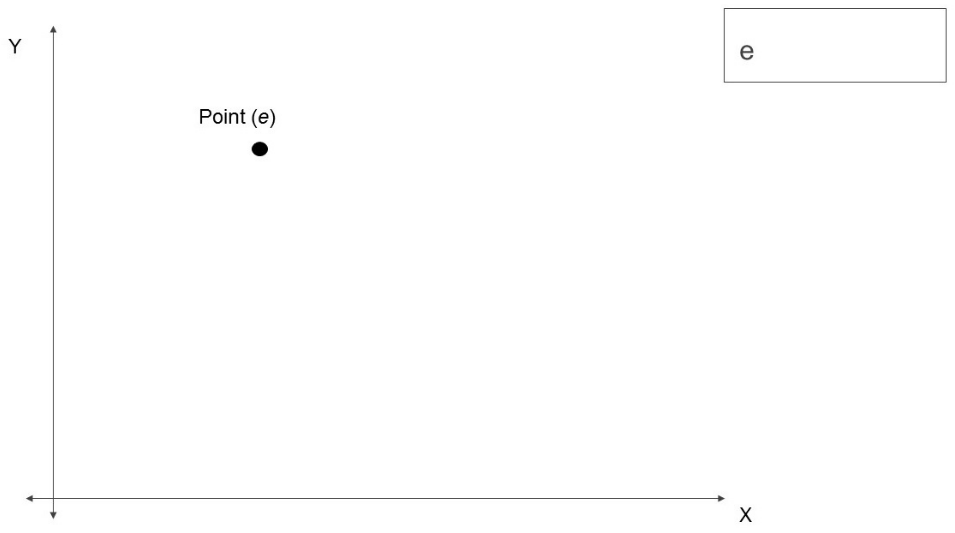


Figure 2.1. Point *e*.

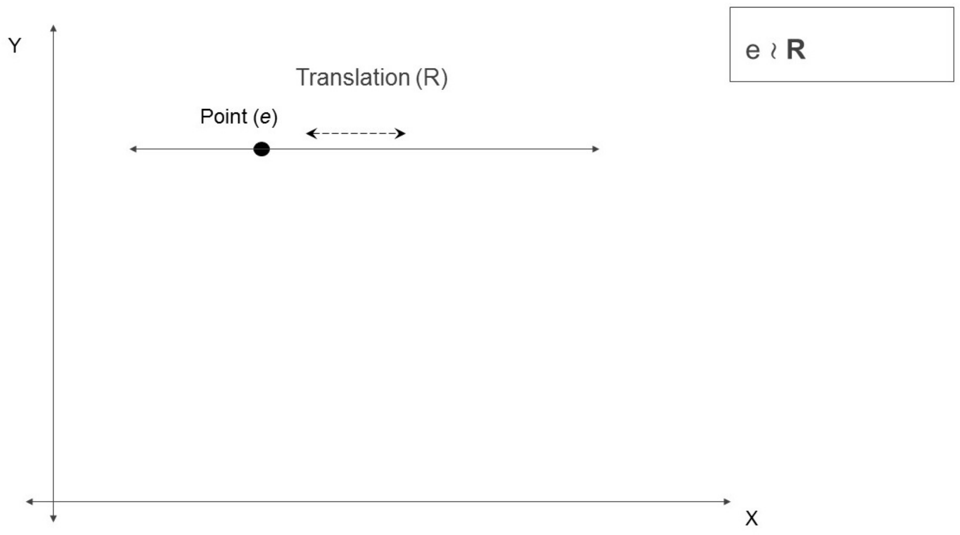
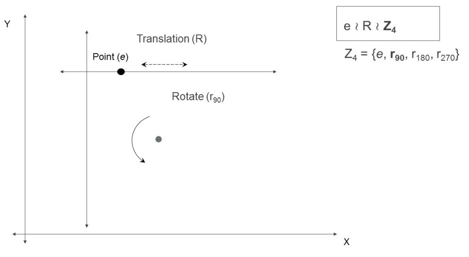
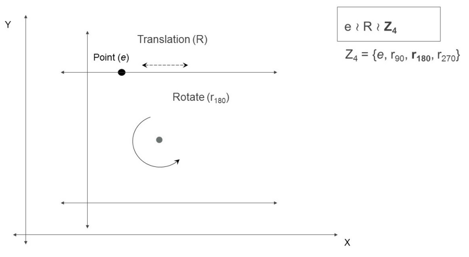


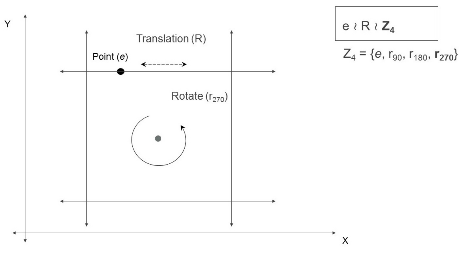
Figure 2.2. Translation of point with group $\Re$.


Y
X
Translation (R)
Point (e)
Rotate (r_{90})
$e \wr R \wr \mathbf{Z_4}$
$Z_4 = \{e, \mathbf{r_{90}}, r_{180}, r_{270}\}$

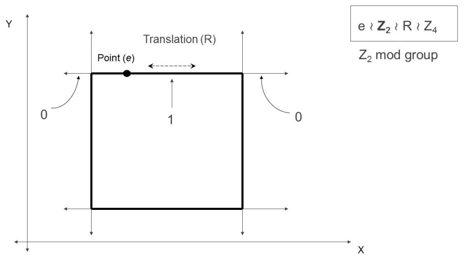
(a) Rotation by r_{90}.


Y
X
Translation (R)
Point (e)
Rotate (r_{180})
$e \wr R \wr \mathbf{Z_4}$
$Z_4 = \{e, r_{90}, \mathbf{r_{180}}, r_{270}\}$

(b) Rotation by r_{180}.


Y
X
Translation (R)
Point (e)
Rotate (r_{270})
$e \wr R \wr \mathbf{Z_4}$
$Z_4 = \{e, r_{90}, r_{180}, \mathbf{r_{270}}\}$

(c) Rotation by r_{270}.



(d) Characteristic function $\mathcal{Z}_2$ applied to e's to turn them on or off.

Figure 2.3. Square generation process.

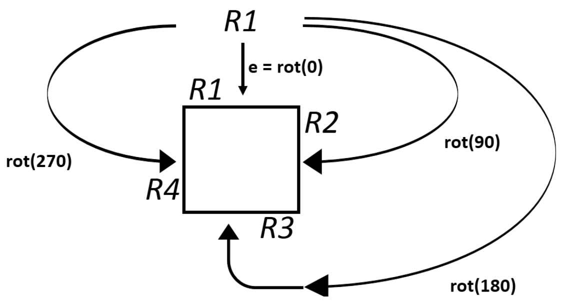


Figure 2.4. Control flow in a wreath product gives an explicit definition in terms of actuation of how to generate the shape; rot(0) (rotate by 0 degrees), the group identity, acts on the first copy, $\Re_1$ to obtain the top of the square; rot(90) acts on $\Re_1$ by rotating it 90 degrees about the center of the square to obtain the right side, $\Re_2$; rot(180) acts on $\Re_1$ by rotating it 180 degrees to obtain $\Re_3$, and finally, rot(270) acts on $\Re_1$ rotating it 270 degrees to obtain $\Re_4$. The rotations, R1, R2, R3, and R4, denote the group action (of $\mathcal{Z}_4$) on the sides of the square.

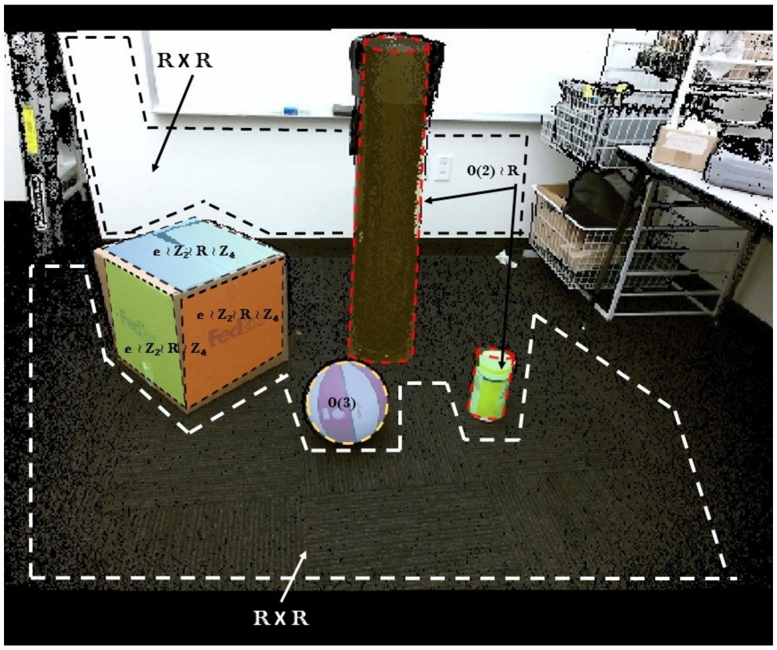


Figure 2.5. Wreath product discovery in a scene.

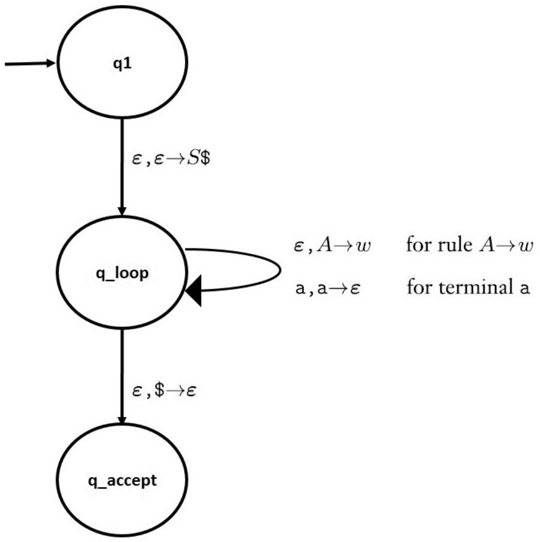


Figure 2.6. PDA for wreath product language.

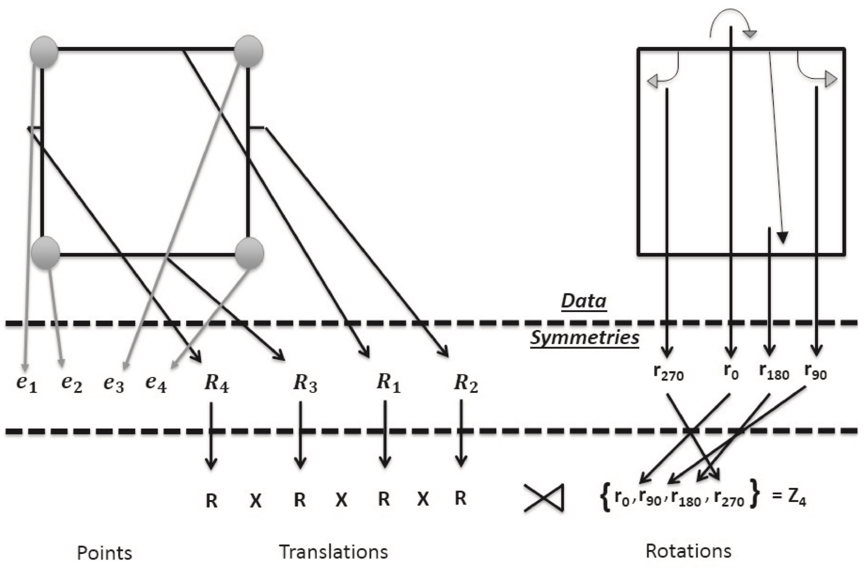

Data
Symmetries
e_1 e_2 e_3 e_4 R_4 R_3 R_1 R_2
r_{270} r_0 r_{180} r_{90}
R X R X R X R
$\{r_0, r_{90}, r_{180}, r_{270}\} = Z_4$
Points
Translations
Rotations

Figure 2.7. Mechanisms for the detection of symmetries in the square.

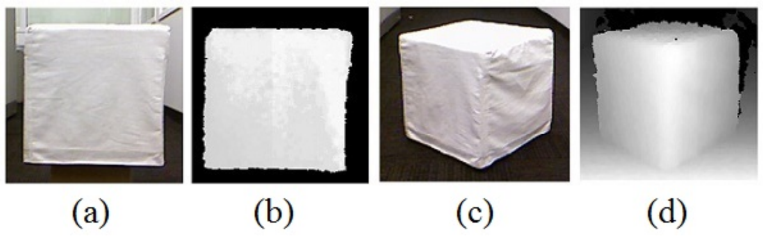


Figure 2.8. Illustration of different views of a cube and corresponding depth images. (a) RGB view of one face of cube (b) Depth map of one face (c) RGB view of three visible faces of cube (d) Depth map of three face view.

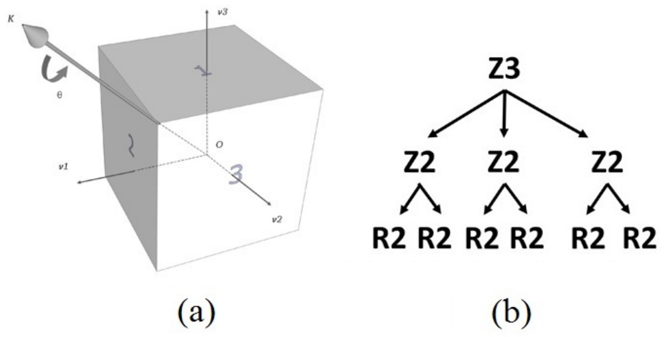


Figure 2.9. Cube and corresponding (3D) wreath product representation. (a) Geometry of $\mathcal{Z}_3$ representation of cube (b) The wreath product tree for $\Re^2 \wr \mathcal{Z}_2 \wr \mathcal{Z}_3$.

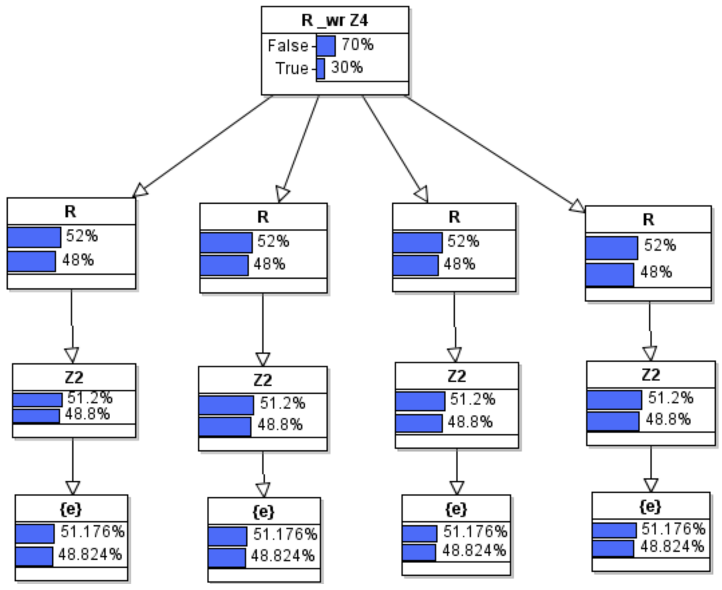


Figure 2.10. The wreath product tree for a square shape, displayed as a Bayesian network.

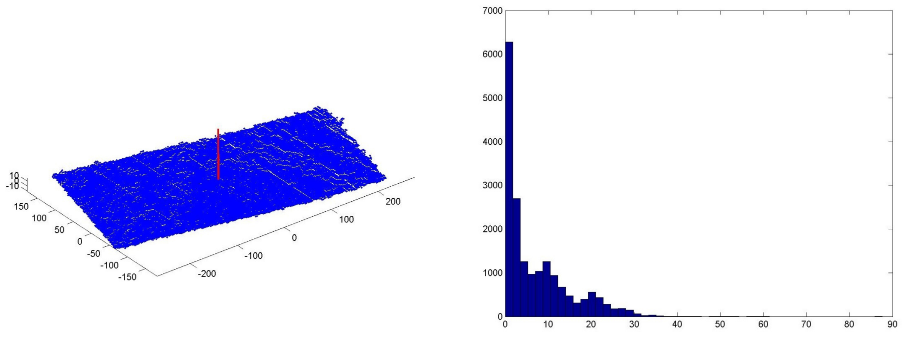


Figure 2.11. Kinect depth data noise characteristics. (a) Reference normal (in red) derived from plane-fitting (b) Histogram of angles between normals of a neighborhood and the reference normal.

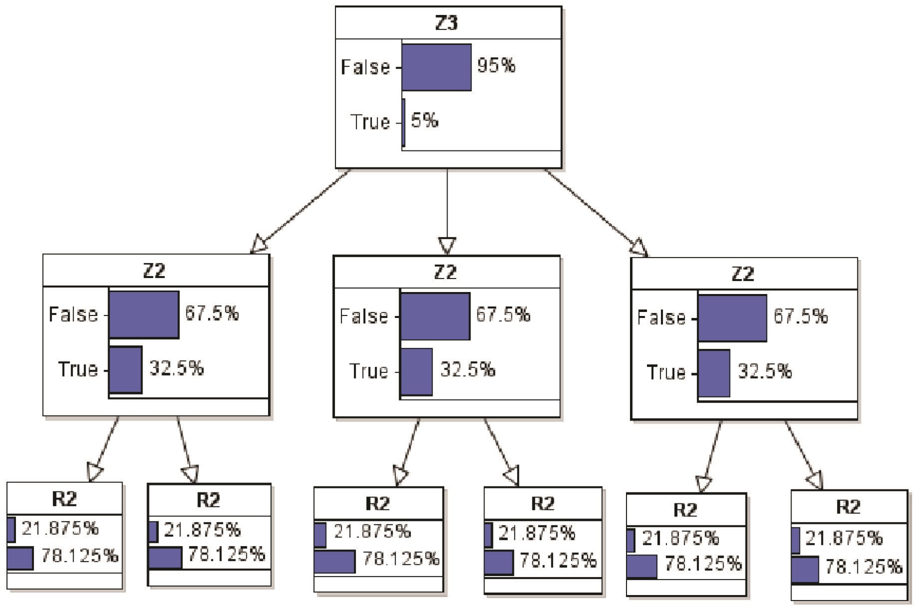


Figure 2.12. A Bayesian wreath product tree for cube.

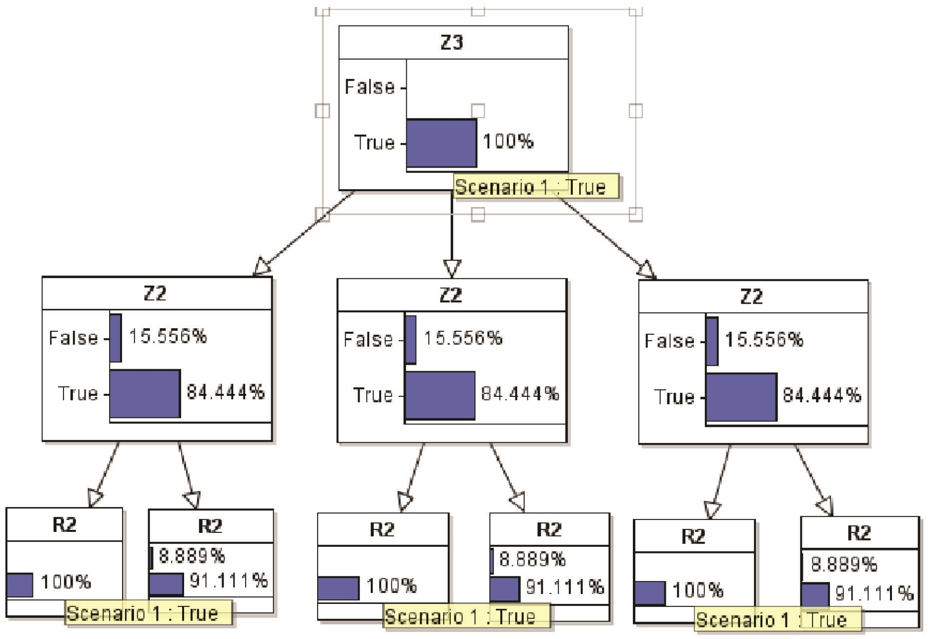


Figure 2.13. Bayesian wreath product tree for cube with observation probabilities.

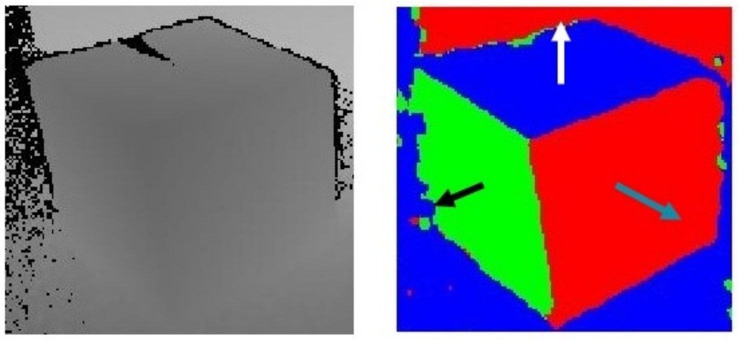


Figure 2.14. Kinect depth data for cube (left) and recovered $\Re^2$ symmetries (right).

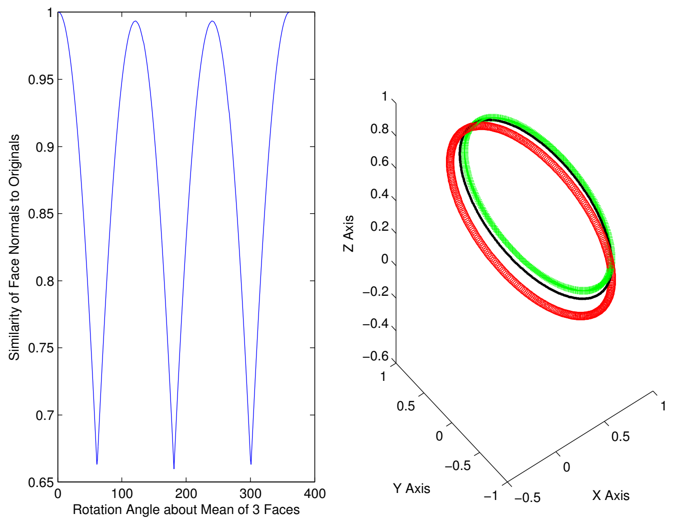


Figure 2.15. Cube (3D) symmetry detection on real data. (a) Similarity of 3 surface normals under rotation about their Mean Vector (b) Trajectories of normal endpoints under rotation.

CHAPTER 3

WPCA: THE WREATH PRODUCT COGNITIVE ARCHITECTURE

The *Wreath Product Cognitive Architecture* (WPCA) is our implementation of a wreath product approach for a cognitive framework. This chapter describes, in-depth, the design principles and various modules that form the WPCA. The beliefs, processes, and modules that form the WPCA are explained in detail.

3.1 WPCA Overview

The layout of the WPCA – a cognitive architecture which exploits the wreath product representation – is shown in Figure 3.1. This architecture allows object representations to be constructed from groups discovered in sensorimotor data, and combined to form wreath products.

As can be seen, the WPCA is a variation of a Beliefs, Desires and Intentions (BDI) architecture (see [25, 26]) consisting of:

- a set of repositories:
  - *belief store*: maintains all beliefs about the world.
  - *goal store*: asserts the goals of the robot agent as well as their priorities.
  - *intention hierarchy*: keeps a small number of currently selected most important specific goals.
  - *plan library*: consists of processes designed to achieve specific goals.
- a number of short-term memories:
  - *percepts*: information derived from sensorimotor data.

  - *options*: a set of possible intentions which are then filtered to get a small set of intentions.
  - *action*: selected agent action (e.g., take data, move, communicate).
  - *reactive action*: emergency control of robot (e.g., obstacle avoidance).

- and a set of processes:
  - *perceive*: acquire and convert raw data into percepts (the first step toward wreat hproduct construction).
  - *belief revision*: updates beliefs, including: add new beliefs, revise old beliefs and update uncertainties.
  - *analyzer*: considers beliefs, goals and current intentions and produces a new set of intentions.
  - *filter*: selects a small set of intentions from the options.
  - *plan selection*: given the current intentions, find appropriate plans to achieve them.
  - *reactive control*: interrupt normal cognitive processing in emergency situations.
  - *execute action*: robot platform level control.

This system provides the mapping from percepts to action for the robot system, but is called from a higher-level module that provides the percept and receives the action. In our framework, this higher-level module drives the *perceive*, and *execute* action blocks as depicted in the architecture diagram. These blocks are part of the code which interfaces the WPCA to the hardware, i.e., the robot, and the Kinect, and thus handles the image acquisition module (which collects structural and RGB data from the Kinect), execution of actions (robot motion in our case), other housekeeping activities, such as termination of execution upon completion, and logging all these processes for debug purposes. The blocks *belief revision*, *analyzer*, *options* generation, *filter*, *plan selection*, *action generation* are part of the robot's "brain" which processes perceptual data (executes reactive behavior if deemed necessary), revises its beliefs based on this data, analyzes the current intentions, goals, and beliefs to generation options (actions that can be taken), and then selects the plan that lays out the sequence of steps (actions) needed to achieve a particular goal. For example, one of the goals in our framework is to "discover a world frame" and the plan will perceive its environment checking for a world frame (a corner in a wall, e.g., which it can reliably find), move around in its environment until it finds the world frame. Every plan in our framework is a state machine, with each state assessing current beliefs and recommending a particular action to take. Action can be taking data, translating (forward or backward), and rotating in place by a given angle (positive or negative rotation). The following sections give a high-level overview of various aspects of this architecture. Detailed workings of the various building blocks of the architecture are given later.

3.2 Beliefs

Beliefs express robot knowledge, and generally have a specification for their uncertainty. For example, when a geometric object such as a planar surface is discovered in the robot's range data, the belief may be created that asserts the existence of a plane. Note that before such a belief is added to the *beliefs store*, it is checked against existing plane beliefs and merged if it matches one of those. In addition, beliefs may serve to express goals by requiring the WPCA to prioritize reducing their uncertainty. In particular, to set up the 3D mapping and navigation benchmark, six goal beliefs are included in the initial beliefs of the robot and prioritized as follows:

1. Floor Plane
2. Gravity Vector
3. World Frame
4. Find All Wreath Products
5. Combine Wreath Products
6. Terminate Execution.

3.2.1 Floor Plane

The floor plane is very special, and it must be discovered before any other goals can be addressed. The initial uncertainty of the floor plane is set to infinity, and until this is reduced to an acceptable level, the robot takes data and tries to identify the floor. In order to identify the floor plane, the robot takes advantage of innate knowledge describing what the floor's normal vector should be in the camera frame.

3.2.2 Gravity Vector

We believe the gravity vector is a strong perceptual cue, and provide innate knowledge to the robot so that the gravity vector is simply the negative of the floor normal (see Leyton [7] for more details on the special role of the gravity direction). Note that the assumption here is that the robot starts off in a position where the floor normal is

$$[0, 0, 1]^T.$$

Mechanisms would have to be put in place to accommodate for other motion planes (ramps, etc.).

3.2.3 World Frame

The world frame is a coordinate transform which describes the robot's coordinate frame in world frame coordinates. This frame has an uncertainty which is initialized to infinity. Once the floor plane (and thus the gravity vector) is known, finding the world frame becomes the robot's top goal. To find this frame, three mutually perpendicular planes which meet in a corner visible in the range data are sought. In this respect, the world frame ($\{e\} \wr R \times R \wr \mathcal{Z}_3$) resembles the wreath product of the cube ($\{e\} \wr R \times R \wr \mathcal{Z}_2 \wr \mathcal{Z}_3$ as described in Chapter 2 section 2.2, except there are no reflected planes. The uncertainty of the frame is determined from these three planes. All other mapping and navigation is performed in the world frame. Percepts are expressed in world frame coordinates. All data acquired in the previous time steps before the world frame is found are deleted, and data found in the current time step are represented in this world frame.

3.2.4 Find Wreath Products

After the world frame is found, the robot's highest priority goal is to discover new wreath product primitives; this includes planes, lines, and points. Both depth and rgb images are analyzed to find these.

3.2.5 Combine Wreath Products

Once no more primitive wreath products can be found, the next goal is to use innate knowledge to discover more complex semantic categories. For example, a pair of parallel planes results in $\Re \times \Re \wr \mathcal{Z}_2$ where $\mathcal{Z}_2$ is either a reflection or a rotation. A cube would be an even more complex wreath product comprised of three $\Re \times \Re \wr \mathcal{Z}_2$ pairs.

3.2.6 Terminate Execution

Once no more wreath product combinations can be found, the robot's goal is to wrap up its activities. This includes saving the beliefs and shutting down the physical robot.

3.3 Innate Beliefs

The robot is provided with a set of innate (initial) beliefs (some corresponding to goals as described above). Some of these beliefs provide a mechanism for the robot to keep track of where it is in the execution of a plan. Some important initial beliefs not described above include the following:

1. Camera Pose

   This is a transformation matrix (4x4) which describes the pose of the camera coordinate frame in the robot coordinate frame.

2. Ever Localized

   If the robot has already determined a world frame, but after some time, the uncertainty in its pose grows too large, this belief is used to let the robot know that it needs to simply reacquire the existing world frame rather than reinvent the world frame.

3. Floor Plane

   This is the assertion that the floor has been found. Other information associated with the floor includes the index into beliefs that points to the WP plane that corresponds to the floor.

3.4 Perception

The principal perception device of the robot is the Kinect V2 (Kinect for Windows v2). Kinect is a time-of-flight sensor which emits an infrared (IR) laser, which reflects off different objects in the scene which is then recorded by an IR camera. The time between laser emission and capture gives the distance to the object. Time-of-flight is recorded per pixel (if the corresponding world point is within a certain range), and thus the depth images are considerably accurate and less noisy than other IR pattern projection structural sensors such as the Kinect v1 or the Asus Xtion. If the depth cannot be determined at a certain pixel, it is assigned either a NaN (Not a Number) value, which is cleaned up by the perception module.

The Kinect v2 is a reasonably priced consumer-grade sensor, has been studied in the field of robotics and computer vision since its launch in 2014, and is widely considered to be one of the best performing time-of-flight sensors for the price, for both gaming and computer vision/robotics applications [27–30] (see [31] for an excellent characterization of the Kinect v2 performance). It has an IR camera, an IR projector, and an RGB camera and outputs the following data:

1) RGB image (1920×1080 px).

2) Infrared image (512×424 px).

3) Depth image (512×424 px).

In addition, an RGB image registered with the IR camera can be generated as the intrinsic and distortion parameters of the device are known. Since the intrinsic parameters and the depth value at each pixel are known, the point cloud can also be generated by determining x and y values per pixel (assuming depth z is known for the pixel), using Equation 3.1, where i and j are the pixel coordinates, and f_x, f_y, c_x, c_y are intrinsic parameters of the IR camera.

$$x = \frac{(i - c_x) \cdot z}{f_x}, \; y = \frac{(j - c_y) \cdot z}{f_y} \tag{3.1}$$

The depth (IR) camera has a field of view of 70.6° (horizontal) ×60.0° (vertical), whereas the color camera has a field of view of 84.1° (horizontal) ×53.8° (vertical). It outputs depth images at a frame rate of 30 frames per second. The frame rate is fixed since it is closely coupled with the time-of-flight mechanism. The operating range of the Kinect is stated to be between 0.5 m and 4.5 m (however, it was observed in our experiments that the minimum distance should be 0.6 m to avoid NaN-pixels and noisy data. The Kinect v2 was calibrated in our experiments using OpenCV calibration technique, and the intrinsic parameters were determined to be as follows: $f_x = 362.3099,\ f_y = 362.5217,\ c_x = 260.1303,\ c_y = 205.9577$ whereas the distortion coefficients were found to be $k_1 = 0.0973,\ k_2 = -0.2822,\ p_1 = 0.0003,\ p_2 = 0.0009,\ k_3 = 0.1059$. We use the libfreenect2 library (driver) by Echtler et al. [32], which outputs the registered RGB image (registered to the depth image), as well as the X,Y,Z points per pixel, along with the depth map. We have modified the libfreenect2 driver source code to suit our needs and output the data in our preferred format so that it can be accessed directly.

The Kinect sensor is mounted on the robot, at a height of 0.82 m above the ground and 0.2032 m in front of the robot center, and it is tilted downwards at 27.86° with respect to the ground (see Figure 3.2). The transformation that represents camera frame points (data acquired from the Kinect), in robot frame can be given as follows: in order to align Y_C with Y_R, we perform a rotation of $27.86° + 90°$, about the X-axis of the camera frame. Then we translate the points by:

$$\bar{t} = \begin{bmatrix} 0 \\ 0.2032 \\ 0.820 \end{bmatrix}$$

(Note that the translation vector $\bar{t}$ is in meters).
If ${}^{R}P$ represents points in the robot (R) frame, and ${}^{C}P$ represents points in the Kinect frame, then the transform from points in the Kinect frame to the robot frame, is given by

$${}^{R}T_C = \begin{bmatrix} \mathcal{R} & t \\ \mathbf{0}^T & 1 \end{bmatrix}$$

where $\mathcal{R}$ is the rotation matrix to align the axes of the frame C with R, and t is the translation vector between the origins of these two frames. Thus, ${}^{R}P_C = {}^{C}P\ \mathcal{R} + t$, will transform points from the camera frame to the robot frame. Based on physical measurements, we have determined $\mathcal{R}$ and t to be the following:

$$\mathcal{R} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -0.4674 & 0.8841 \\ 0 & -0.8841 & -0.4674 \end{bmatrix}$$

and

$$t = \begin{bmatrix} 0 \\ 0.2032 \\ 0.820 \end{bmatrix}$$

3.5 Plans

In operation, the WPCA supports the beliefs, desires and intentions in the form of finite state machines (FSM). Wreath products can also be used to represent plans in the WPCA. Consider the simple sentence and desire "Go to point B from point A in a straight line." Since points A and B are points in space,"...point B from point A in a straight line" can be represented as a wreath product for a straight line segment ($e \wr \mathcal{Z}_2 \wr \Re$). Thus, a wreath product with the semantics (i.e., geometry) required for this specific motion must be instantiated. The following discussion gives a high-level overview of the wreath product interpretation process. A wreath product directly describes a way to generate a set of points by a specific action sequence. That sequence is abstracted to a specific set of groups which express Euclidean transforms like translation, rotation, etc.

In the example posed above, the agent uses the wreath product abstraction $e \wr \mathcal{Z}_2 \wr \Re$ annotated with the specifics of the motion (i.e., e is set as a specific start point in a particular coordinate frame; $\mathcal{Z}_2$ is set to a specific characteristic function; $\Re$ is the specific line between start and end points of the task. What is key is that the *execution* of the abstract wreath product actuation groups can be mapped to multiple actuation systems simultaneously; for example, while driving the wheels to move along the path, it can also drive the camera to scan along the path to watch for obstacles. Alternatively, the vision system can be used to monitor that the motion is indeed linear – for example by watching for specific symmetries in the visual data (there should be a focus of expansion in a regular camera image, or there should only be columnar motion in a polar image expanded about the focus of expansion – Figure 3.3). In Figure 3.4 : (a) and (b) are the log polar transforms of perceptual data (images) as the robot moves forward, while (c) shows the symmetry in the direction of the optical flow vectors (the majority of them are in the $\frac{\pi}{2}$ direction as indicated by the histogram).

3.6 Detailed BDI Component of WPCA

The following subsections briefly explain, at an algorithmic level, how all the significant blocks of the architecture are designed and executed, and the workings of various algorithms used in the perceptual data processing, action generation, and plan library.

3.6.1 Environment Interaction Module

The environment interaction module works similar to a simulator, where, at each step, it perceives the environment, and takes the action returned by the "brain" of the robot. Algorithm 1 illustrates, at a very high level, how the environment module works: The program starts with basic housekeeping activities such as clearing persistent variables from the previous run, setting the configurations for communication with robot hardware, and logging. The terminating condition for the program is received from the brain of the robot, the WP_BOT, when it has finished executing its plans. The program alternates between capturing perceptual data and executing action as determined by the WP_BOT. If the action specifies to capture data, the KINECT drivers will be called and data acquired and cleaned (more details are given in section 3.4). If the action specifies to move (either translate, or rotate), the robot's drivers will be called to move the robot.

Algorithm 1: WP_RUN_BDI algorithm

```
Data:
    – Clear persistent variables
    – Set Robot Hardware Connection Constants
    – Initialize Logger
while Terminate condition not set do
    if action = TAKE_DATA then
        Call KINECT drivers;
        TAKE_DATA();
        Clean acquired data;
    else
        empty data/use previous data;
    end
    Bundle acquired data as percept;
    //Call robot brain to process current percept;
    action = WP_bot(percept, del_t, action);
    Log action;
    if action = MOVE or ROTATE then
        Initiate robot controller;
        MOVE_ROBOT();
    end
end
Close logger;
```

3.6.2 Robot Brain (WP_BOT)

WP_BOT is the brain of the robot and utilizes the BDI paradigm. It is comprised of the WP discovery, localization, belief revision, analyzer, filter, and plan selection and action generation functions. At the start, beliefs are initialized with the innate knowledge available to the robot, and the variables that will persist throughout the execution of this program. Variables needed for localization of robot are also initialized. The percepts (discussed earlier in sub-section 3.4), are passed on as arguments to the DATA_TO_WP function, which processes these percepts to create wreath product sets (WP_SETS) that contain (possibly newly) discovered wreath products. The robot starts its life with little innate knowledge. As part of the innate knowledge the transform that takes the camera frame to the robot frame (base of the robot) is supplied. Based on this, the robot will first discover the floor vector (i.e., the vector of the plane that is pointing up). Once the floor vector is discovered, the normal of the floor plane that best matches the floor vector, is assigned as the floor plan normal, and the corresponding floor plane index is updated in the beliefs. Once the floor plane vector is discovered, the robot will try to discover the world frame. The world frame is any corner in the surroundings that has 3 orthogonal walls - one of which needs to be the floor - that meet at a single point, since this point and the wall plane normals can uniquely identify a world frame (e.g., if the robot started off inside a cylindrical room, it cannot find a unique world frame). The plane normals of these 3 orthogonal planes, originating at the corner point form the X,Y,Z-axes of the world frame. Note that the floor plane is always the Z-axis pointing opposite the gravity vector, and the X and Y axes follow the right-hand rule. The structure of WP_BOT is shown in Algorithm 2. The function DATA_TO_WP (Algorithm 2 line 1) is described in sub-section 3.6.3.

3.6.3 Data To WPs: Wreath Product Construction Cycle

Algorithm 3 transforms range data from the depth sensor into wreath products. The process starts by building planes from 3D data points in the camera frame that are segmented using the RANSAC (RANdom Sample And Consensus) algorithm ([33, 34]). For example, in Figure 3.5 (a), the three orthogonal planes found during the search for world frame and segmented using RANSAC are shown in different colors, along with the axes

Algorithm 2: WP BOT.

```
Data: Initialize persistent variables, Set Beliefs to innate knowledge, Initialize
      localization variables
1  [WP_SET] ← DATA_TO_WP(percepts);
2  if World frame found then
3      [$\mu, \Sigma$] ← Update_Robot_Pose();
4      [$L, c$] ← Detect_landmark_Correspondence();
5      [$\mu, \Sigma$] ← Update_Robot_Pose_Using_LM_Correspondence();
6      [$^{W}T_{R}$] ← Update_Robot_Pose($\mu$);
7      Update_Pose_In_Beliefs($^{W}T_{R}$)
8  end
9  BELIEF_REVISION(WP_SET);
10 [OPTIONS] ← WP_ANALYZER(INTENTIONS);
11 [INTENTIONS] ← FILTER(OPTIONS);
12 [PLAN] ← PLAN_SELECTION(INTENTIONS);
13 [ACTION] ← PLAN_TO_ACTION(PLAN);
14 if current_plan_terminated then
15     [INTENTIONS] ← UPDATE_INTENTIONS();
16 end
17 if terminate_signal then
18     UPDATE_BELIEFS();
19     TERMINATE();
20 end
```

found (which are the normals to the three planes located at the origin).

Even though these normals might be very close to, but not exactly orthonormal, they can be converted into three orthonormal vectors (M_{ortho}) using equation 3.2, where $\mathcal{R}$ is the rotation matrix in the transform $^{W}T_{R} = \begin{bmatrix} \mathcal{R} & t \\ \mathbf{0}^{T} & 1 \end{bmatrix}$.

$$\mathcal{R}_{ortho} = \mathcal{R}(\mathcal{R}^{T}\mathcal{R})^{-\frac{1}{2}} \tag{3.2}$$

Each plane point is then transformed from the camera frame to the robot frame (using the innate knowledge of the transform $^{R}T_{C}$ - camera frame to robot frame). For each plane, the plane parameters are found, namely, the plane normal n_{pi} using singular value decomposition (SVD), distance to the plane from origin d_{pi}, and error, ϵ_{pi}, of the plane points fit to the plane (lines 3 - 10). Line 12 removes duplicate planes by merging planes that have similar surface normal and similar distance from the origin. Lines 15 and 18 will find lines (R) and points (E) based on intersection of two (i, j), or three $[R \times R]$ planes (i, j, k), respectively. More details on the E, R, and $[R \times R]$ notation can be found in [4]. Figure 3.5 (b) shows the WPs discovered by this process.

The yellow star has been added which signifies the location of the world frame origin. The plane points (in black) are sparsely plotted to show the 3 planes (denoted by $[R \times R]$ and green dotted lines added for demarcation). Each plane's normal is shown as a blue arrow. Lines (R's) are intersections of pairs of planes and are shown in red, whereas points (E's) are intersections of three planes and are shown as blue dots. This particular image was generated from a set of beliefs that contained 12 innate beliefs (robot pose, camera to robot transform, gravity vector, and other information), and an additional 11 beliefs were discovered (four planes, five lines, and two points), for a total of 23 beliefs. Since an indoor office environment is mostly comprised of planar surfaces, this algorithm effectively finds most of the planes, lines, and points, and their parametric information, if any. All these planes, points, and lines are added to the WP_set. Note that the superscript R - not to be confused with the wreath product R which is a line - of a wreath product signifies that all these WPs have been transformed from the camera reference frame to the robot reference frame; they will be transformed into the world reference frame when the world frame is discovered, in the MERGE function.

3.6.4 Belief Revision

This function merges newly discovered beliefs (as wreath products) with existing beliefs (see Algorithm 4). Note that a distinction is made in our algorithms between points that comprise a plane, or a line (${}^{R}p_i$ for example) which will be used for frame transformations etc., and wreath product points (denoted by E). Lines 3 – 7 transform all newly discovered WPs to the world reference frame - denoted by superscript W - where they are compared against the existing planes (lines 9 – 14), lines (lines 16 – 21) and points (lines 23 – 28) in beliefs to check if they are duplicates, and to merge them if so. The functions $\Psi_{[R\times R]}$, Ψ_R, and Ψ_E - for planes, lines and points, respectively - check the parameters to determine if two WPs match each other spatially, and $\Gamma_{[R\times R]}$, Γ_R, and Γ_E merge those two WPs for planes, lines, and points, respectively. The beliefs are then updated accordingly.

Algorithm 3: DEPTH TO WP.

```
[P] ← RANSAC(percepts);
$^{R}T_{C}$ ← get_transform(Beliefs);
for all plane points $^{C}p_i$ in P do
    $^{R}p_i \leftarrow transform(^{R}T_C, ^{C}p_i)$;
    $n_{pi} \leftarrow svd(^{R}p_i)$;
    $d_{pi} \leftarrow distance(^{R}O, ^{R}p_i, n_{pi})$;
    $\epsilon_{pi} \leftarrow plane\_error(^{R}p_i, n_{pi}, d_{pi})$;
    $^{R}[R \times R]_i \leftarrow [^{R}p_i, n_{pi}, d_{pi}]$
    $WP\_set(w) \leftarrow ^{R}[R \times R]_i$;
end
for all planes $^{C}[R \times R]_i$ in WP_set do
    Remove_Duplicates();
end
for all plane pairs $^{R}[R \times R]_i$ and $^{R}[R \times R]_j$ in WP_set do
    $R_{i,j} \leftarrow intersection(^{R}[R \times R]_i, ^{R}[R \times R]_j)$;
end
for all plane triples $^{R}[R \times R]_i$, $^{R}[R \times R]_j$, and $^{R}[R \times R]_k$ in WP_set do
    $E_{i,j,k} \leftarrow intersection(^{R}[R \times R]_i, ^{R}[R \times R]_j, ^{R}[R \times R]_k)$;
end
$WP\_set(w) \leftarrow ^{R}R_{i,j}$;
$WP\_set(w) \leftarrow ^{R}E_{i,j,k}$;
return WP_set
```

3.6.5 Belief Store

Beliefs are stored on disk as an array of structures, where each index corresponds to a belief. For an example of the structure of this array, see Appendix A, section A.1. These innate beliefs are described; note that each also has a type and an associated uncertainty.

- **belief (1) [type: POSE]**: robot pose (a 4×4 rigid transform matrix initially set to the identity before it is discovered).
- **belief (2) [type: VECTOR]**: gravity vector which is known to be $[0, 0, -1]^T$ (pointing downwards).
- **belief (3) [type: ACTION]**: retrieve all WPs in the belief store (innate beliefs as well).
- **belief (4) [type: ACTION]**: extend the WPs discovered so far.
- **belief (5) [type: ACTION]**: combine WPs discovered so far. These WPs that are combined do not qualify for merging, as they are not the same WPs, but they satisfy

Algorithm 4: MERGE BELIEFS.

```
1  ${}^{W}T_R \leftarrow$ get_transform(Beliefs);
2  for all new planes ${}^{R}[R \times R]_i$ do
3      ${}^{R}p_i \leftarrow get\_points({}^{R}[R \times R]_i)$;
4      ${}^{W}p_i \leftarrow transform({}^{W}T_R, {}^{R}p_i)$;
5      $n_{pi} \leftarrow svd({}^{W}p_i)$;
6      $d_{pi} \leftarrow distance({}^{W}O, {}^{W}p_i, n_{pi})$;
7      $\epsilon_{pi} \leftarrow plane\_error({}^{W}p_i, n_{pi}, d_{pi})$;
8      ${}^{W}[R \times R]_i \leftarrow [{}^{W}p_i, n_{pi}, d_{pi}]$;
9      for all existing planes ${}^{W}[R \times R]_j$ in Beliefs do
10         $[{}^{W}p_j, n_{pj}, d_{pj}] \leftarrow {}^{W}[R \times R]_j$;
11         if $\Psi_{[R \times R]}({}^{W}[R \times R]_j, {}^{W}[R \times R]_i) < \tau$ then
12             ${}^{W}[R \times R]_j \leftarrow \Gamma_{[R \times R]}({}^{W}[R \times R]_j, {}^{W}[R \times R]_i)$;
13             Beliefs $\leftarrow {}^{W}[R \times R]_j$;
14         end
15     end
16     for all existing lines ${}^{W}R_j$ in Beliefs do
17         $[{}^{W}l_j, n_{lj}, d_{lj}] \leftarrow {}^{W}R_j$;
18         if $\Psi_R({}^{W}R_j, {}^{W}R_i) < \tau$ then
19             ${}^{W}R_j \leftarrow \Gamma_R({}^{W}R_j, {}^{W}R_i)$;
20             Beliefs $\leftarrow {}^{W}R_j$;
21         end
22     end
23     for all existing points ${}^{W}E_j$ in Beliefs do
24         $[{}^{W}e_j] \leftarrow {}^{W}E_j$;
25         if $\Psi_E({}^{W}E_j, {}^{W}E_i) < \tau$ then
26             ${}^{W}E_j \leftarrow \Gamma_E({}^{W}E_j, {}^{W}E_i)$;
27             Beliefs $\leftarrow {}^{W}E_j$;
28         end
29     end
30 end
```

certain symmetry constraints. For example, opposite walls of a room that satisfy a reflective symmetry will be combined to a form a $[R \times R] \wr \mathcal{Z}_2$ WP.

- **belief (6) [type: POSE]**: 4×4 rigid transform matrix from camera frame to robot frame. This transformation has been explained in detail in section 3.4.

- **belief (7) [type:ACTION]**: Terminate robot run. A run of the robot may execute several plans, and any one of the plans can set the terminating condition. Once this condition is set, the robot will save all the beliefs in the store (disk), and stop execution.

- **belief (8) [type: FACT]**: This is a type of "FACT" which says whether the robot was ever localized, i.e., has it ever found the world frame. Ideally, the robot will find the world frame once and all beliefs will be transformed into this world frame. But occasionally the robot can get lost where the uncertainty of its world frame increases beyond a certain threshold. In this case the robot will relocalize itself and transform the subsequently discovered beliefs in the new frame.

- **belief (9) [type: FACT]**: The "floor" belief is a special kind of plane in our architecture, because the robot moves on the floor (we are assuming it would not go on ramps, or change floors, etc.). The floor plane WP is discovered during the "Find Floor Plane" plan detailed later, but to summarize, the floor plan is usually found during the very first data acquisition cycle when planes are segmented using RANSAC, and the plane whose normal is opposite to the gravity vector is selected as the floor plane. The "id" (index) of this plane in the beliefs is stored in the "belief(9).info," which points to the floor plane WP.

- **belief (10) [type: FACT]**: WPs have been combined to form higher level WPs.

- **belief (11) [type: IMAGE2]**: The "world frame image" is simply the RGB image (registered with the depth data), or in other words, the snapshot of the world frame. This RGB image is used during multi-robot navigation and localization algorithm, to match the world frame originally discovered by one robot, to the world frame newly discovered by another robot.

- **belief (12) [type: FACT]**: "Plan Done?" simply means whether the current plan in execution is done or otherwise. This notifies the analyzer to select the next plan based on current priorities.

3.6.6 Analyzer

The *Analyzer* module considers current options and beliefs, and returns viable options. The *Analyzer* has access to the plan library as well as current beliefs. Whenever the *Analyzer* is called, it first checks whether any reactive behavior needs to be executed; for example, it will query "Is there any obstacle present?" If there is, the reactive behavior will be called, else normal execution resumes. During normal execution, the *Analyzer* will consider all options (plans that can be executed, including reactive behavior) based on priority, and whether the plan has already been executed, or uncertainties are too high / too low for the plan to be executed. For example, highest priority will be obstacle avoidance - the robot cannot take normal action if obstacles are present. Then it will try to find the floor plane, if the uncertainty associated with the floor plane is too high. Once the floor plane is found, the world frame discovery will be the next highest priority task if the uncertainty associated with the world frame is too high. Once the floor plane uncertainty is lowered, the floor mapping plan can be executed, and so on.

3.6.7 Plan Library

In the WPCA, plans are finite state machines (FSMs), each one achieving a high-level goal (for example, "detect world frame"). After each state, the control is returned to the WP-BOT that either takes an action, or acquires data depending on the action suggested by the plan. Figure 3.6 illustrates one of the plans: "Find World Frame." The start and final states are shown with dark edges, the blue arrows indicate transitions to next states, and green arrows indicate that the world frame was found, and leads to the final state.

- **Start State**: *The execution of the plan starts here.* The start state of every plan will direct the robot to acquire Kinect data (since the robot may have moved before plan execution started). When the action is set to "Take Data," control returns back to the WP BOT where the action is executed, and the control comes back to the plan. The next state is set as state 1.

- **State 1**: *Check if world frame exists in current view.* This state will first check if there exist 3 orthogonal planes (one of which is the floor). Algorithm 5 first checks, for all triples of planes ($[R \times R]_i$, $[R \times R]_j$, $[R \times R]_k$), if any of them are parallel to each other (lines 2,3). If not, this candidate triple is checked by Algorithm 6 for a corner point - *pt_intersection* which is the intersection of the three planes - and how close the corner is to all three planes (*score*, determined by how many plane points fall within a sphere of a given radius R). The maximum score from all triples and corresponding intersection point is chosen for determining the world frame. Algorithm 7 determines the world frame by taking cross product of pairs of surface normals of all three planes (cross product of two vectors is orthogonal to both those vectors). The cross product vector which is in the direction of the floor plane normal is chosen as the Z-axis (else the vectors are swapped and the cross product is determined again to satisfy this condition). The two vectors whose cross product is a valid z-axis are chosen in the order in which they satisfy the right-hand rule ($\bar{z} \leftarrow \bar{x} \times \bar{y}$). These three vectors corresponding to the world frame axes are arranged in a rotation matrix $\mathcal{R}$ as shown on line 11. The orthonormality of R is improved by line 12 as discussed before. The origin of this world frame is the *pt_intersection*. $\mathcal{R}$ and *pt_intersection* is combined on line 14 to yield transform $\mathcal{T}$ from world frame to robot frame. The inverse of $\mathcal{T}$ yield the robot frame to world frame transform. Uncertainty of this world frame is determined by uncertainty of the three planes forming the world frame. If a corner is found, the next state is set to 14, else it is set to 2. States 3, 5, 7, 9, 11 are similar to 1, except for the action they take when world frame is not found in that state. If such a frame is not found in the current location, the robot finds the largest plane that it can see, apart from the floor, and it sets the action to rotate towards that plane by an angle such that its Y-axis will be perpendicular to that plane. Once this action is set, control returns to WP BOT for action execution.

- **States 2, 4, 6, 8, 10, 12:** *Acquire kinect data and move to their respective next states.* WPs that are not the floor plane are deleted from current beliefs (since only the floor plane remains fixed with respect to the robot's motion). For example, in state 2, Action is set to "Take Data" and next state is set to 3.

- **State 3**: *Check if world frame exists in current view.* This state will first check if it can see a corner, as in state 1. If a corner is found, the next state is set to 14, else it is set to 4. If such a frame is not found in the current location, the robot finds the distance to the largest plane that it can see, and it sets the action to translate by the determined distance towards that plane. Once this action is set control returns to WP BOT for action execution.

- **State 5**: *Check if world frame exists in current view.* This state first checks if it can see a corner, as in state 1. If a corner is found, the next state is set to 14, else it is set to 6. If such a frame is not found in the current location, the robot sets the action to rotate 45° counter-clockwise. Once this action is set control returns to WP BOT for action execution.

- **State 7**: *Check if world frame exists in current view.* This state first checks if it can see a corner, as in state 1. If a corner is found, the next state is set to 14, else it is set to 8. If such a frame is not found in the current location, the robot sets the action to rotate 90° clockwise (to be oriented at 45° with respect to the wall). Once this action is set control returns to WP BOT for action execution.

- **State 9**: *Check if world frame exists in current view.* This state first checks if it can see a corner, as in state 1. If a corner is found, the next state is set to 14, else it is set to 10. If such a frame is not found in the current location, the robot sets the action to rotate 45° clockwise (to be oriented parallel to the wall). Once this action is set control returns to WP BOT for action execution.

- **State 11**: *Check if world frame exists in current view.* This state first checks if it can see a corner, as in state 1. If a corner is found, the next state is set to 14, else it is set to 12. If such a frame is not found in the current location, the robot finds the distance to the wall in front - after rotating 90° to the previous wall - that it can see (if the wall is out of range for the Kinect sensor, the robot will move by a fixed 1 m increments till it sees the wall). The action is set to move forwards by the distance discovered to the wall it is now looking at. Once this action is set control returns to WP BOT for action execution.

- **State 13:** *Check if corner is found, else repeat from state 5.* If a corner is found, the next state is set to 14, else it is set to 5 and action is set to "Do Nothing." Thus, control goes to state 5 (after returning from WP BOT), and the process is repeated at the current wall it is facing, from state 5 onwards.

- **State 14:** *Final state.* This state is only reached if a world frame is found. At the start of this state, the uncertainty of the world frame in beliefs(1) is set to the mean of the uncertainties of the three orthonormal planes forming the world frame (initially, by default, the uncertainty is a large number). Here, each plane's uncertainty is determined by the RMS error of the distance of points of the plane to the regression plane as determined by the eigenvectors. Thereafter, the robot frame to world frame transform (${}^{W}T_{R}$) is determined based on the distance to the corner point and the three orthonormal vectors. The camera to robot transform (${}^{R}T_{C}$) is already part of the innate beliefs. Using these transforms, the WP beliefs determined in the current cycle are represented in the world frame. Hereafter, any new WPs generated in any subsequent plans are represented in the world reference frame. The plan is set to "Done" and execution of this plan will be halted once control goes back to WP BOT.

Algorithm 5: MULTI-ROBOT WORLD FRAME MATCHING

```
   input  : $LM_2$: All observed WPs in robot 2's frame.
            $LM_1$: All WP in beliefs from robot 1
   output: $E$: Error (plane parameter) between matches planes in $LM_1$ and $LM_2$.
 1 for all plane triples $[R \times R]_i,\ [R \times R]_j,\ [R \times R]_k$ in $LM_2$ do
 2     if $[R \times R]_i \cdot [R \times R]_j \approx 1$ or $[R \times R]_j \cdot [R \times R]_k \approx 1$ or $[R \times R]_i \cdot [R \times R]_k \approx 1$ then
 3         continue with next triple;
 4     else
 5         [scores, $pt\_intersections$] $\leftarrow$
           CHECK_CORNER_LOCALE($[R \times R]_i,\ [R \times R]_j,\ [R \times R]_k$);
 6     end
 7 end
 8 $[score, index] \leftarrow max(scores)$;
 9 if $score > \tau$ then
10     [world_frame, world_frame_uncertainty] $\leftarrow$
       GET_WORLD_FRAME($[R \times R]_i,\ [R \times R]_j,\ [R \times R]_k, pt\_intersection_{index}$);
11 end
12 return $E$
```

Algorithm 6: CHECK CORNER LOCALE

```
   input  : Plane triples $[R \times R]_i,\ [R \times R]_j,\ [R \times R]_k, R$
   output: score, $pt\_intersection$
 1 $pt\_intersection \leftarrow find\_intersection\_point()$;
 2 $score \leftarrow$
   $get\_num\_points\_within\_sphere([R \times R]_i,\ [R \times R]_j,\ [R \times R]_k, pt\_intersection, R)$;
 3 return $score$, $pt\_intersection$
```

Algorithm 7: GET WORLD FRAME

```
input  : plane triple $[R \times R]_1,\ [R \times R]_2,\ [R \times R]_3$ in $LM_2$, floor plane normal
         $\bar{n}_{floor}, pt\_intersection$
output: world_frame, world_frame_uncertainty.
1  $\bar{n_1} \leftarrow normal([R \times R]_1),\ \bar{n_2} \leftarrow normal([R \times R]_2),\ \bar{n_3} \leftarrow normal([R \times R]_3);$
2  $c_{1,2} \leftarrow \bar{n_1} \times \bar{n_2},\ c_{2,3} \leftarrow \bar{n_2} \times \bar{n_3},\ c_{3,2} \leftarrow \bar{n_3} \times \bar{n_1};$
3  $[v,\ max_{dp}] \leftarrow max(c_{1,2} \cdot \bar{n}_{floor}, c_{2,3} \cdot \bar{n}_{floor}, c_{3,1} \cdot \bar{n}_{floor});$
4  if $max_{dp} < 0$ then
5      $\bar{z} \leftarrow -\bar{v};$
6      $[\bar{x}, \bar{y}] \leftarrow swap(c_{1,2}, c_{2,3}, c_{3,1})$
7  else
8      $\bar{z} \leftarrow \bar{v};$
9      $[\bar{x}, \bar{y}] \leftarrow select(c_{1,2}, c_{2,3}, c_{3,1})$
10 end
11 $\mathcal{R} = \begin{bmatrix} \bar{x}_1 & \bar{y}_1 & \bar{z}_1 \\ \bar{x}_2 & \bar{y}_2 & \bar{z}_2 \\ \bar{x}_3 & \bar{y}_3 & \bar{z}_3 \end{bmatrix};$
12 $\mathcal{R}_{ortho} = \mathcal{R}(\mathcal{R}^T\mathcal{R})^{-\frac{1}{2}};$
13 $\bar{t} \leftarrow pt\_intersection;$
14 $\mathcal{T} = \begin{bmatrix} \mathcal{R}_{ortho} & t \\ 0 & 1 \end{bmatrix};$
15 $world\_frame \leftarrow \mathcal{T}^{-1};$
16 $world\_frame\_uncertainty \leftarrow [u_i, u_j, u_k];$
17 return $world\_frame, world\_frame\_uncertainty$
```

3.6.8 Action Execution

In WPCA, at a high-level, two kinds of actions are possible: Motion and Perception.

3.6.8.1 Motion

The robot platform is an iRobot *Create* differential-drive platform. Thus, there are two discrete possible movements for the robot: rotate and translate. The hardware driver for this robot [35] does not allow for complicated combined rotation and translation and leads to a simple motion model. Robot hardware and wheel slip mean that the motion execution of the robot is not perfect and this has been quantified. There is a certain motion bias (usually negative) introduced when the robot moves. For example, if the robot is commanded to rotate 30° clockwise, the robot might rotate only 25° clockwise in reality. To understand this motion bias, tests were performed for rotation as well as translation motions.

The approach used to determine the bias is very simple: the robot is placed on a circular chart with a hole large enough for the circular robot to rotate in. The circumference is marked with angles $[0, 360]^\circ$ with 1° increments (see Figure 3.7).

The robot is then commanded to rotate 10 times for 5 positive angles ($10^\circ, 30^\circ, 60^\circ, 90^\circ, 180^\circ$) and their negative counterparts, for a total of 100 rotations. For every command, the error between the amount of rotation commanded, and the amount the robot actually rotated is recorded. This gives us the rotation error as a function of amount of rotation commanded, and a mapping of commanded rotation to rotation error (see Table 3.1 for the error data). Based on our experiments, it was observed that the robot tends to rotate less than what is commanded, in a majority of occasions (the signs of errors in Table 3.1 reflect this fact). Note that the error increases as the rotation command angle value increases. In operation, large rotation values such as 180° were not used, hence only moderate error was encountered. To ensure actual rotation stays true to the command, positive and negative rotation commands are compensated with appropriate values from the table, and for other rotation command values, compensation is determined by interpolation.

Forward motion translation error of the robot was quantified by translating it a positive distance (forward translation) 10 times over a meter, and over two meters, and recording the error along the X and Y axes of the robot. The translation error in Y axis is determined to be 0.0294 m., with variance 0.00004, and is a function of the translation distance (thus, for 2 m, the error will be 2×0.0294). The distance moved can be given by Equation 3.3

$$d_{moved} = 0.9706 \times d_{commanded} \tag{3.3}$$

It should be noted that unlike the rotational error, the robot always moves less by the amount stated above. This error is compensated for, by adding it to the command to translate. The translation error in X axis is determined to be 0.0176 m. with variance 0.0010. and the robot tends to move towards the positive X axis. Action commands are 3-tuple; for rotation and translation they are represented as $[0, \pm\delta\theta, 0]$ and $[0, 0, \pm norm(\delta x, \delta y)]$, respectively, where $\delta\theta$ is the rotation angle, and $\delta x, \delta y$ specify movement in x, y direction, and the amount of translation is given by $norm(\delta x, \delta y)$.

3.6.8.2 Perception

At code-level, the action $[1, 0, 0]$ specifies to take data. When WP BOT gets this action command, the Kinect v2 driver code is called to acquired the depth image, (x, y, z) data, and the registered RGB image. Details of the perception mechanism have already been discussed previously in subsection 3.4.

3.7 Goals (Prioritized)

Goals are prioritized tasks based on the order in which they need to be completed, for subsequent goals to be achieved. For example, the gravity vector and the floor plane must be found first, before the world frame can be discovered. The floor mapping goal needs to be completed, before robot belief-sharing goal can be achieved.

- **Priority 1:**
  - **Gravity Vector:** Find out which way gravity points
  - **World Frame:** Localize robot in the world frame for the first time.
  - **World Frame Rediscovery (if applicable):** Relocalize robot in the world frame, if lost.
- **Priority 2:**
  - **Floor Mapping:** Find direction of gravity vector
  - **Robot Sharing:** Localize robot in the world frame.

The following subsections briefly discuss the gravity vector and world frame detection plans. World frame rediscovery, floor mapping, and robot sharing plans are discussed in chapter 5

3.7.1 World Frame Discovery: Corner Detection

The corner detection algorithm uses RANSAC as mentioned in subsection 3.6.3. When the agent starts off in a new world, it can either be in a cluttered scene or an uncluttered scene. Figure 3.8 shows the depth image of a cluttered scene whereas Figures 3.9 and 3.10 show two different views of the same cluttered scene, with planes segmented using

RANSAC shown in different colors (green represents the floor, red represents the side wall, and blue represents the back wall that is partially hidden by the legs of a table, seen as vertical gaps in Figure 3.9).

Gaps in the floor (green) also result from occluded views of the boxes under the table. Note that these 3D points are in the robot frame of reference, thus the robot is at $x = 0, y = 0$ in its own frame of reference, i.e., at the origin, and it is looking towards the wall depicted in blue. Couple of things should be noted in this particular processed data set example:

- Of the total number of points - 217088 - only a subsampled set of 72,363 points is used. Out of these, RANSAC determines only 49,471 points as belonging to any plane (inliers); remaining 22,892 points are determined to be outliers. Thus, we have a fairly large percentage - 31% - of points that do not belong to a plane, because RANSAC filters them out. This percentage varies depending on the scene, however, it has been observed that RANSAC is effective at discarding points that do not belong to planes.

- The three distinct planes determined using RANSAC are fairly far apart from each other. Ideally, at a corner, the three planes should have a larger percentage of points close to one another, and at the corner, points from all three planes should be very close to one another.

These observations help us determine whether the current set of points arise from where three orthonormal planes - two orthogonal walls and the floor meet to form a corner. The first such corner found when the agent starts exploring, is determined to be its initial world frame.

Compared to the cluttered scene, an uncluttered scene is shown in Figure 3.11 that contains only 3 (or more) planes orthogonal to each other. The depth image of the uncluttered scene is shown in Figure 3.12 and red lines are added to demarcate where the 3 planes intersect each other.

The number of points in the original point cloud is 108,544, of which 96,571 points are retained after RANSAC segmentation (a retention rate of 88.97%). For these 3 orthogonal planes, their point of intersection determines the corner point (origin of the world frame) we are interested in. In order to find the corner point, we solve the linear system of equations of the planes. The general equation of a plane is given by

$$ax + by + bz + d = 0$$

For this plane, the representative normal is given by $\hat{\mathbf{n}} = (a, b, c)$. Let $\mathbf{A}$ represent the matrix of coefficients of the three planar equations, i.e., the coefficients a_i, b_i, c_i, for the ith plane, let $\mathbf{x}$ represent the unknowns, i.e., $(x, y, z)^T$, and let $\mathbf{b}$ represent the right-hand side of the equations, i.e., $-d_i$, for the ith plane. This gives us the system of linear equations of the form $\mathbf{Ax} = \mathbf{b}$. This can be solved using $\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$ – since the intersection point cannot be at the robot origin – giving us the point where the three planes intersect. For this particular data in Figure 3.13, this point was determined to be $(-0.16, 2.165, -0.01)$, and is shown as the thick black point (for emphasis), in Figure 3.13. Since all the points are in the robot frame and the robot Y-axis is pointed at the origin, the X and Z values will be small whereas the Y-value will be large.

Table 3.1. Rotation error compensation.

Rotation Command (°)	Error Mean (°)	Error Variance
10	-3.4	1.82
30	-2.75	1.125
60	-1.65	1.225
90	-1.25	0.4583
180	-9.35	6.5583
-10	1.8	0.4556
-30	3.3	4.1778
-60	2.1	0.71
-90	2.95	0.8583
-180	9.9	23.3778

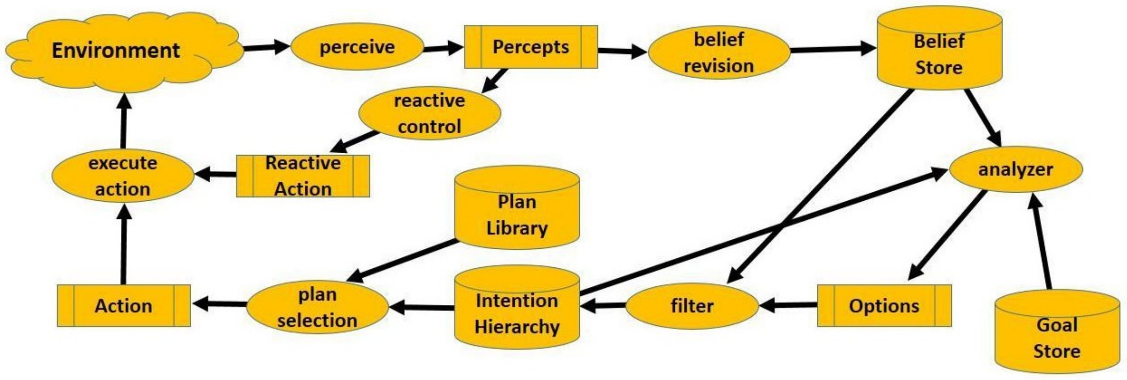


Figure 3.1. BDI architecture.

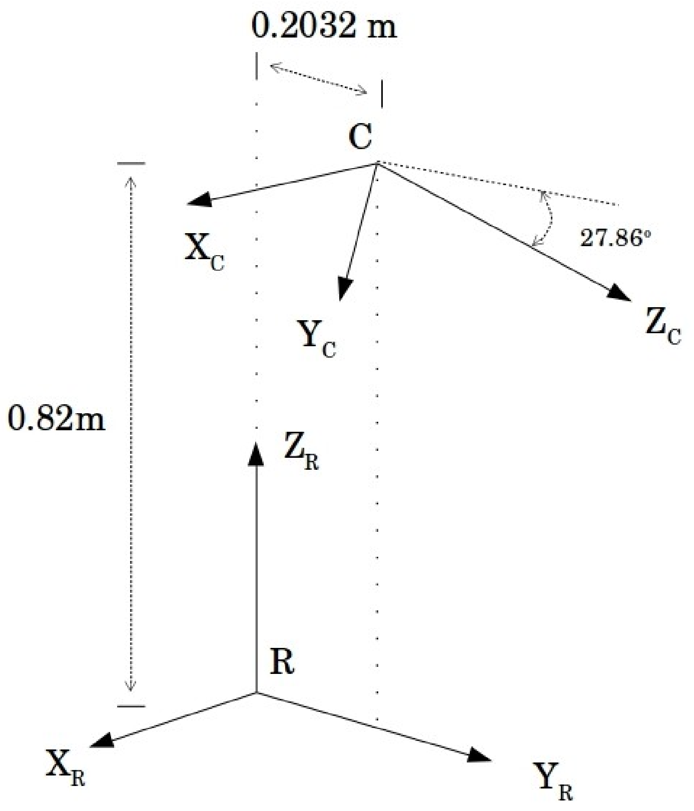


Figure 3.2. Illustration of the transform between Kinect frame (C) and robot frame (R). Z_C-axis and Y_R-axis lie in the same (vertical) plane, Z_C axis is rotated clockwise about the X_C axis by 27.86°, and X_C and X_R axes point in the same direction. Camera frame origin C is translated by 0.82 m above, and 0.2032 m in front of the robot origin R.

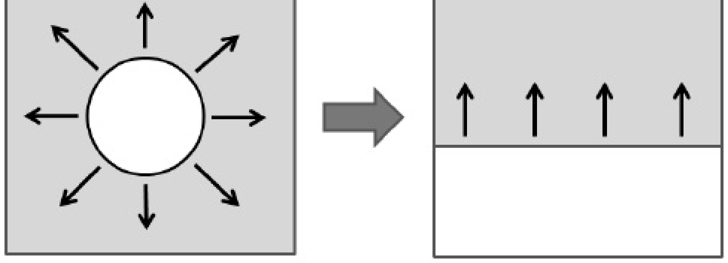


Figure 3.3. Polar image optical flow method to detect pure translation.

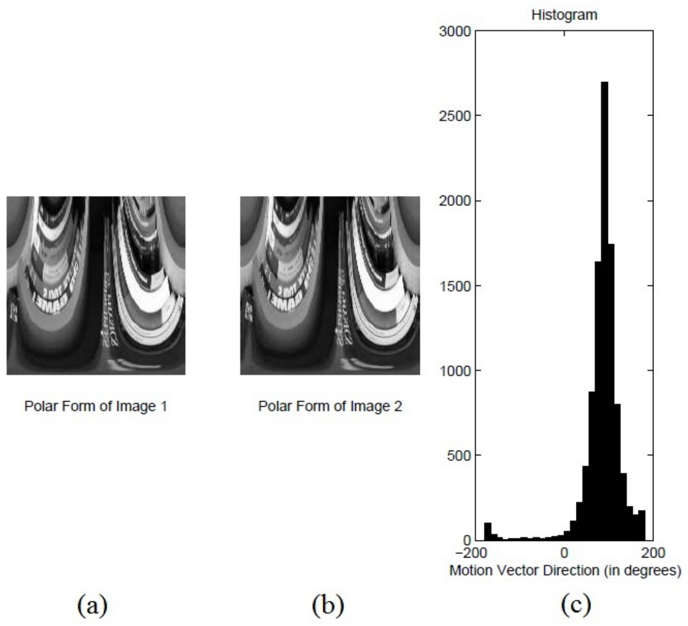


Figure 3.4. Log polar transforms of consecutive images upon Translation, shown by (a) and (b), and (c) is the histogram of direction of optical flow motion vectors.

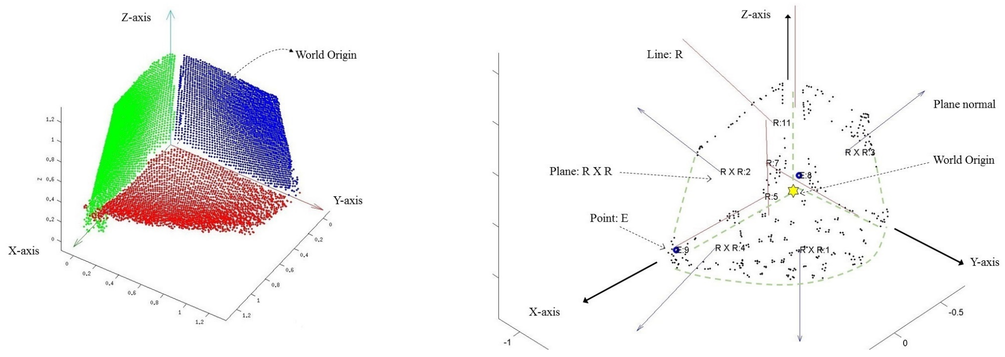


Figure 3.5. WP discovery from data: (left) shows the different planes segmented using RANSAC and (right) shows the detailed WPs discovered.

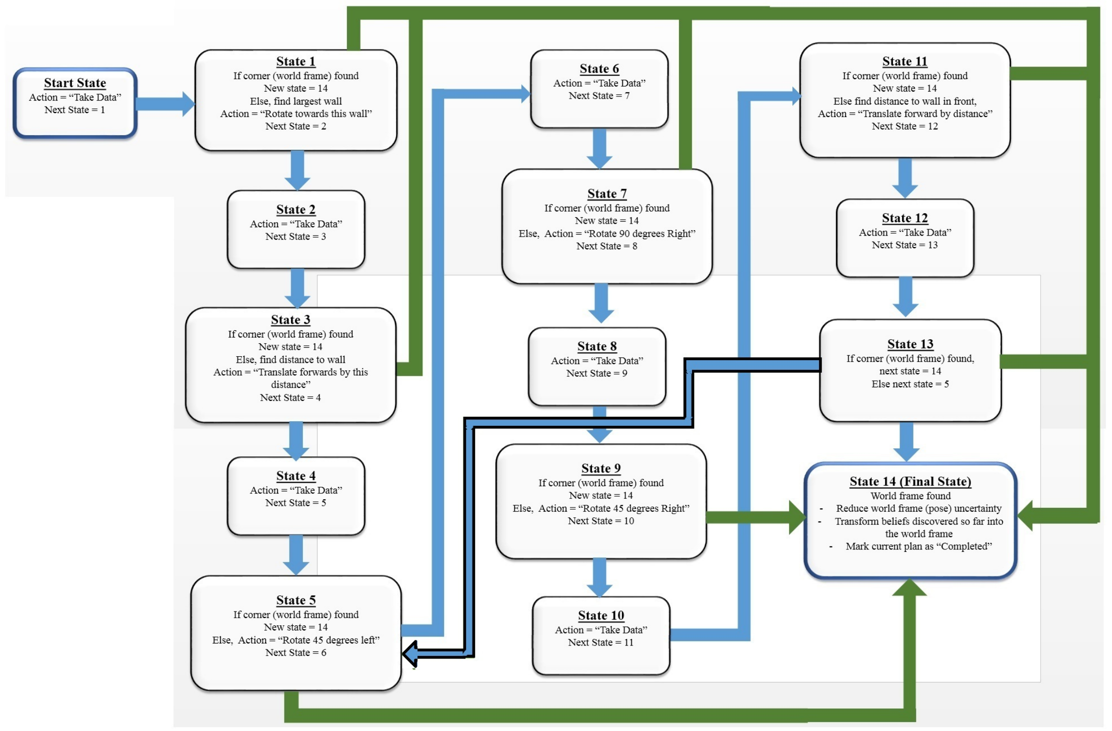

Start State
Action = "Take Data"
Next State = 1
State 1
If corner (world frame) found
New state = 14
Else, find largest wall
Action = "Rotate towards this wall"
Next State = 2
State 2
Action = "Take Data"
Next State = 3
State 3
If corner (world frame) found
New state = 14
Else, find distance to wall
Action = "Translate forwards by this distance"
Next State = 4
State 4
Action = "Take Data"
Next State = 5
State 5
If corner (world frame) found
New state = 14
Else, Action = "Rotate 45 degrees left"
Next State = 6
State 6
Action = "Take Data"
Next State = 7
State 7
If corner (world frame) found
New state = 14
Else, Action = "Rotate 90 degrees Right"
Next State = 8
State 8
Action = "Take Data"
Next State = 9
State 9
If corner (world frame) found
New state = 14
Else, Action = "Rotate 45 degrees Right"
Next State = 10
State 10
Action = "Take Data"
Next State = 11
State 11
If corner (world frame) found
New state = 14
Else find distance to wall in front,
Action = "Translate forward by distance"
Next State = 12
State 12
Action = "Take Data"
Next State = 13
State 13
If corner (world frame) found,
next state = 14
Else next state = 5
State 14 (Final State)
World frame found
- Reduce world frame (pose) uncertainty
- Transform beliefs discovered so far into the world frame
- Mark current plan as "Completed"

Figure 3.6. FSM for the plan "Find World Frame."

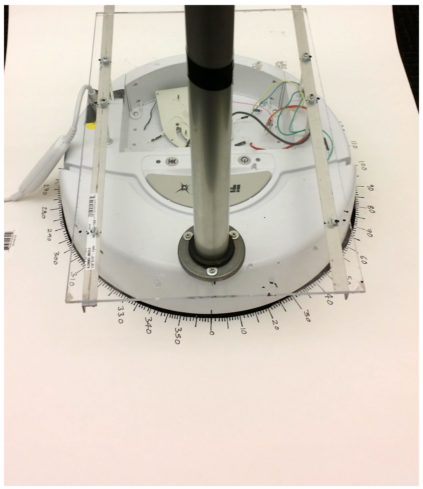


Figure 3.7. Determining rotation motion bias for the robot.

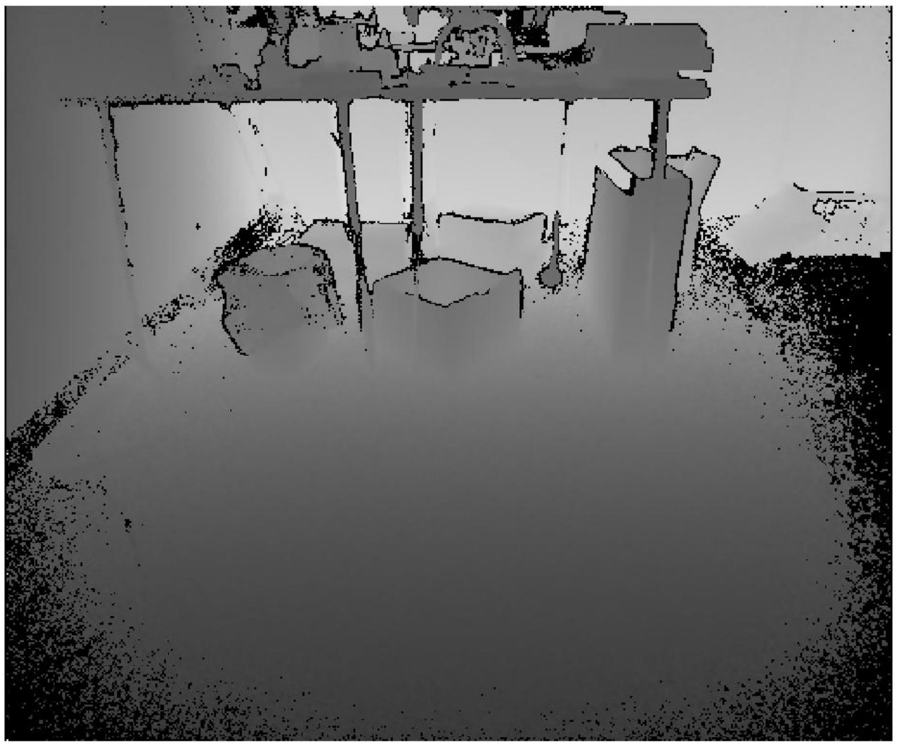


Figure 3.8. Depth image of the cluttered scene.

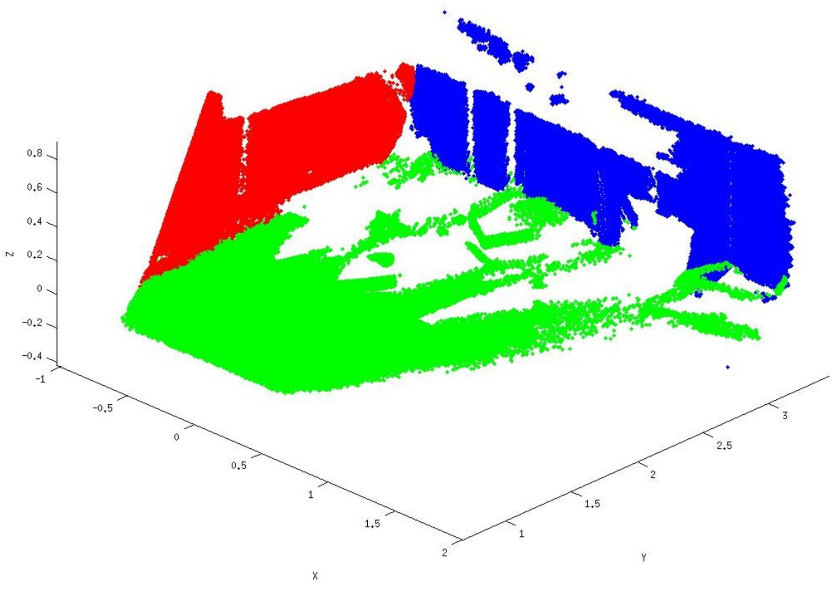


Figure 3.9. One view of the 3 planes detected in a cluttered scene, shown in different colors based on point labels.

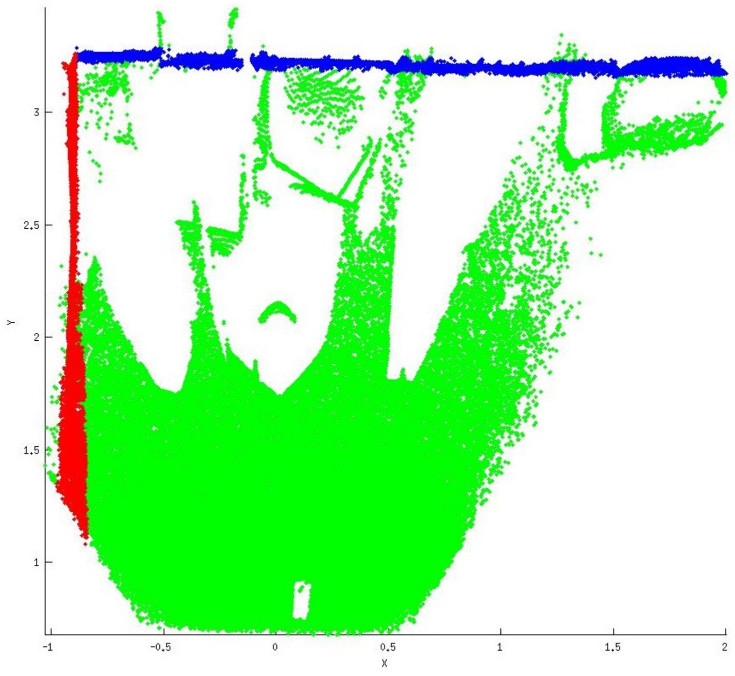


Figure 3.10. X-Y (top) view of the 3 planes detected in a cluttered scene, shown in different colors based on point labels.

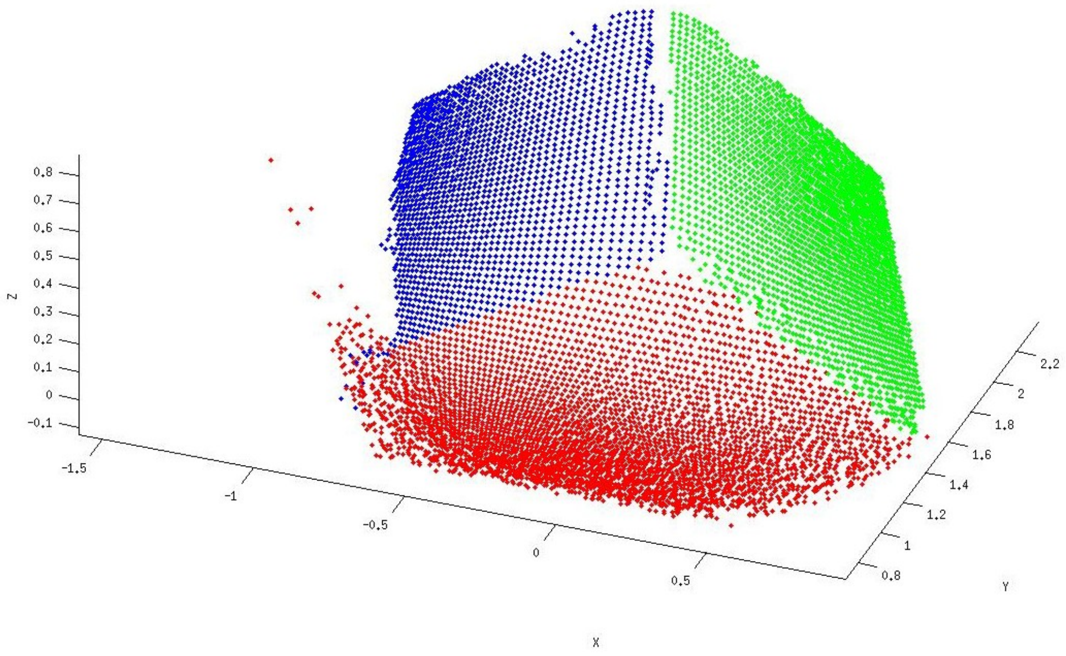


Figure 3.11. One view of the uncluttered scene with orthogonal planes marked in different colors, segmented by RANSAC.

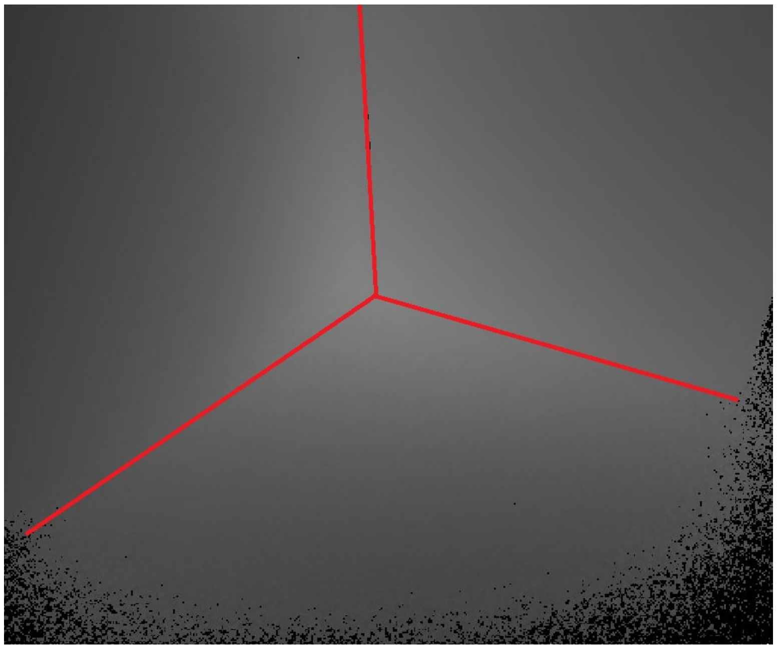


Figure 3.12. Depth image of uncluttered corner.

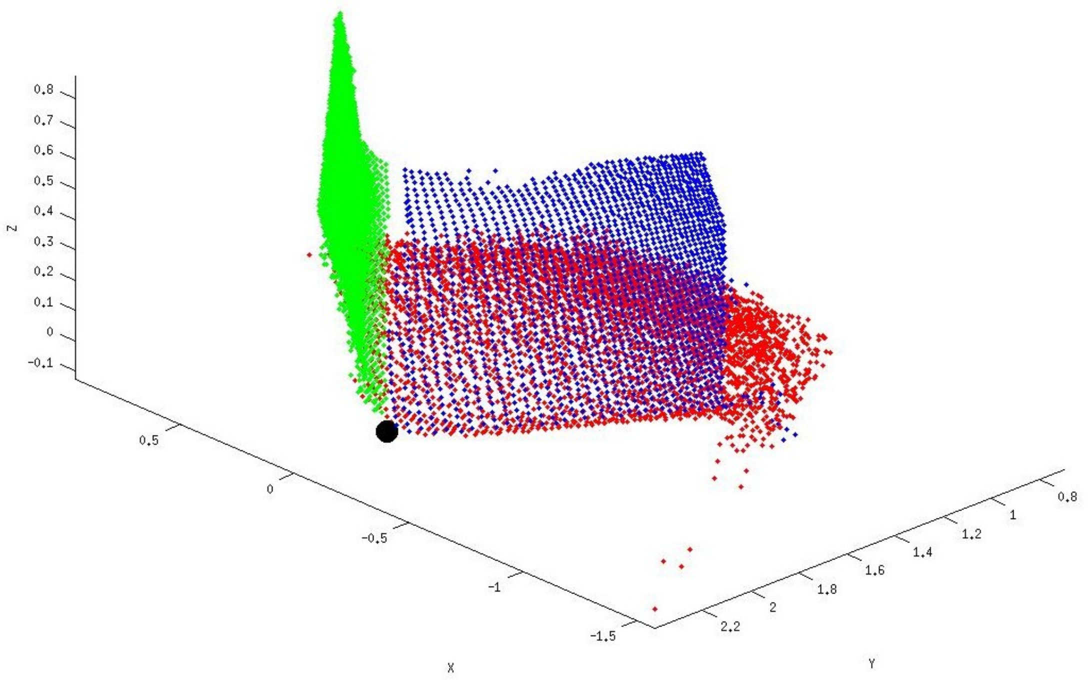


Figure 3.13. Rotated view of the planes, with the corner marked as a thick black point.

CHAPTER 4

WREATH PRODUCT BASED 2D SHAPE ANAYLSIS IN ENDAS

Although the original intent was to explore the application of wreath products to 3D data analysis, in the course of the work we obtained some interesting results in the analysis of 2D shapes, and, in particular, in the classification of text in engineering drawings.[1] We show that this representation offers several advantages with respect to robust and effective semantic analysis of Computer-Aided Design (CAD) drawings in terms of classification rates. Document analysis methods have been studied for several decades and much progress has been made; see [36, 37] for an overview. However, there are many classes of document images which still pose serious problems for effective semantic analysis. Of particular interest here are CAD drawings, and more specifically sets of scanned drawings for which either the electronic CAD no longer exists, or which were produced by hand. We demonstrate results on a set of CAD-generated drawings for automotive parts.

One major issue with the wreath product approach is that the symmetry groups (operators) are defined in some specific Cartesian coordinate frame; i.e., a translation is along a line defined in that frame. One of our goals in this work is to find a representation in terms of the natural motor actuation signals of the observing robot agent. This leads to the hypothesis that shape representation based on the generative actuation process of the observing agent can be effective and efficient. We describe here a shape recognition method based on encoding the shape in terms of the actuation signal needed to generate (or observe) the shape. The method is effective in that it robustly characterizes shape, and it is effective in that it has low computational complexity.

In a more general setting, autonomous robots are embedded in some static or dynamic environment, and are expected to represent and carry out tasks in this environment in the

[1]This work was done in collaboration with Narong Boonsirisumpun and we appreciate his contributions.

presence of sensing and actuation uncertainty. This requires the agent to have a robust representation of its environment, as well as of the plans and actions that it can execute in that environment [21]. Our work here deals with creating such a representation that is robust by (1) being abstract in nature and grounded in (symmetry) theory so that it is more general, and (2) being mathematically well-structured so that it is practical. This framework is exploited to classify 2D characters.

Here we extend our previous work [5] to include direct incorporation and exploitation of actuation data in the analysis of shape. Our main result here is the development of a novel shape analysis method and the demonstration of its effectiveness in the text analysis of engineering CAD documents. Figure 4.1 shows the overall scheme for both 2D and 3D datasets. The 2D data of interest here consists of scanned engineering drawings like those shown in Figure 4.2. The image analysis consists of the extraction of basic shape symmetries (represented as wreath products), followed by symmetry parsing (given as Wreath Product Constraint Sets), finally passing through a classification component where hypotheses are formed as described in the figure. We provide a formal grammar for this parsing in which the lowest level terminal symbols are simple symmetries and nonterminal symbols correspond to more complex shapes. The hypotheses produced by the system are ranked according to a Bayesian analysis based on the wreath product directed acyclic graph as well as the parse tree. Much work has been done in engineering document analysis (see [36] for a detailed survey), but to our knowledge, there are few implemented systems in which shape is represented in terms of actuation primitives. One example of such work is that of Plamondon [38–41]), but that approach has a very different basis rooted in the kinematics of human rapid movement. Other recent studies of more global properties of document analysis, e.g., using deep convolutional networks [42, 43], are more conventional in that they are still based on the geometric properties of the points comprising the shape, rather than exploiting how the shape is synthesized. For another survey of document analysis and recognition, see [44].

For the basic description of the original work on the wreath product sensorimotor approach, see [37]. Here we go beyond their results by developing a coherent approach to the semantic analysis of large sets of CAD drawing images. Figure 4.2 shows examples of the types of images we analyze; on the left is a text file that accompanies an engineering drawing to explain how to paint the structure; on the right is a hand-drawn design of a nuclear waste storage facility.

The left image is a text drawing that provides information about the drawing and the image on the right is a hand drawn plan for one of the double-shell nuclear waste storage tanks at Hanford, WA. The semantic information in such drawings is needed to develop electronic CAD for automotive parts and for nondestructive examinations, respectively. The overall goal is to find the basic character strokes (defined as Wreath Product Primitives), followed by character classification (using Wreath Product Constraint Sets) and finally word recognition (by dictionary lookup) from those. Figure 4.3 shows the Enhanced Non-Deterministic Analysis System (NDAS) which achieves this analysis; ENDAS uses agents to achieve a *parse* of the image. The levels of NDAS correspond to preprocessing, terminal symbol hypotheses, and nonterminal symbol hypotheses. Every *start symbol* represents a complete parse of the image (e.g., a *Text Image*).

We apply Leyton's idea of a generative model of shape directly to low-level image analysis of drawings. Some examples of the types of symmetry include:

- *the translation symmetry group denoted by* $\Re$ $(1D)$: the invariance of pixel sets under translation defines a straight line segment.

- *the rotation symmetry group denoted by* $\mathcal{O}(2)$ $(2D)$: the invariance of pixel sets under rotation defines a circle.

- *the reflection symmetry group denoted by* $\mathcal{Z}_2$ $(2D)$: the invariance of a set of pixels under reflection about a line in the plane describes bilateral symmetry in 2D.

From these symmetry features, we apply this idea to generate the **Wreath Product Constraint Set** (WPCS) to improve the segmentation of low-level geometric primitives in engineering drawings. For example, the lowercase letter set ("b," "d," "p," "q") all look similar in shape. But using symmetry analysis, each character shows that the important symmetry structure in their shape is only one circle ($\mathcal{O}(2)$) and one straight line ($\Re$).

So, we can write a WPCS for each letter ("b," "d," "p," "q") to organize the detection of their features ($\mathcal{O}(2)$ and $\Re$) in the desired position and differentiate between these four characters. We then create agents to search for such WPCS's.

4.1 Structural Model

In this section we introduce a structural model of technical drawings that allows an agent-based organization of the ENDAS system. We define the layout of the technical drawings in terms of structural grammar. $G = (V, \Sigma, R, S)$ where V is a set of non-terminals, Σ is a set of terminals, R is a set of rewrite rules, and S is the start symbol.

4.1.1 Terminal Structure Set

- $a|b|...|z|A|B|...|Z|0|1|...|9|\%|\$|...|\#|.|,|-|'|(|)|...|?$
- $Space \equiv ''\quad''$ (image segment which is a white space)
- $HSpace \equiv Space$ with a nearby left and right segment
- $VSpace \equiv Space$ with a nearby up and down segment
- $Line \equiv$ image segment which is a straight solid line.
- $Arc \equiv$ image segment which is an arc.
- $Circle \equiv$ image segment which is a circle.

4.1.2 Nonterminal Structure Set

- $Letter := a|b|...|z|A|B|...|Z$
- $Digit := 0|1|...|9$
- $SpecialChar := \%|\$|...|\#$
- $Punctuation := .|,|-|'|(|)|...|?$
- $Char := Letter \mid Digit \mid SpecialChar \mid Punctuation$
- $Word := Char \mid Char\ Word$
- $LineOfText := Word\ HSpace\ Word \mid Word\ HSpace\ LineOfText$
- $PageOfText := LineOfText\ VSpace\ LineOfText \mid LineOfText\ VSpace\ PageOfText$
- $Text := Word \mid LineOfText \mid PageOfText$

- $ArrowHead : Line + Line$

  $|Line + Line + Line$

- $PointerRay := Line + ArrowHead$

- $PointerLine := ArrowHead + Line + ArrowHead$

- $PointerArcRay := Arc + ArrowHead$

- $PointerArcLine := ArrowHead + Arc + ArrowHead$

- $Box := Line + Line + Line + Line$

- $PointerPair := (PointerRay + PointerRay)|(PointerLine + PointerLine)$

- $PointerArcPair := (PointerArcRay + PointerArcRay)|(PointerArcLine + PointerArcLine)$

- $Dimension := (PointerPair|PointerArcPair) + Text$

- $Graphic := Line|Box|Circle|ArrowHead$
  $|PointerRay|PointerLine|PointerArcRay$
  $|PointerArcLine|Dimension$

- $TextinBox := Text + Box$

- $Table := TextinBox\ TextinBox \mid TextinBox\ Table$

- $GraphicDrawing := Graphic \mid Graphic\ Table \mid Graphic\ Text$

- $TextDrawing := PageOfText \mid PageOfText\ Text \mid Table \mid Table\ Text$

- $Drawing := TextDrawing|GraphicDrawing$

4.2 Wreath Product Primitives

Define a *wreath product primitive* (WPP) as either a $e \wr \mathcal{Z}_2 \wr \Re$ group or a $e \wr \mathcal{Z}_2 \wr O(2)$ group. As a first step, a set of WPP's is fit to the pixels in each connected component. Given a connected component and a WPP set for that component, a *minimal WPP cover set* is a combination of WPP's that cover the connected component skeleton, and if any WPP is removed, the component is no longer covered.

From each WPP set, the complete set of minimal WPP cover sets is found, and they provide the initial characterization of what defines a particular shape. For example, Figure 4.4 shows some examples of WPP minimal cover sets.

Leyton described wreath products abstractly as symbol sequences and every $e \wr \mathcal{Z}_2 \wr \Re$ wreath product is equivalent to every other. We, however, are faced with unique, existing instances, and thus, associate a coordinate frame (generally, the rectangle containing the symbol) with each as well as descriptions of the $\mathcal{Z}_2$ mod group which is used to produce finite length sets (i.e., end points for line segments and angular limits for circles).

The WPP minimal cover sets shown in Figure 4.4 are then used to produce a WPCS which will characterize the shape. The additional information in the WPCS over the minimal cover set includes any symmetries between WPP's in the set. For example, the two WPP's in the lowercase letters "a" and "e" have both vertical and horizontal reflection symmetries; the letter "A" has a vertical reflection symmetry between the two side arms; the letter "M" has vertical reflection symmetries on the two side arms and the two inner arms; the digits "0" and "2" do not have higher level symmetries.

4.3 Wreath Product Constraint Sets

We propose *Wreath Product Constraint Sets* (WPCS) as a mechanism to represent shape (here: lower and upper case English letters and the digits 0 to 9). A *wreath product constraint set* (WPCS) is a set of WPPs as well as any higher level symmetries (e.g., reflection symmetries which are described in this same coordinate frame as the WPPs):

1. Uses $\Re$ and $O(2)$ wreath product groups as the basic shape constituents.

2. Enumerates further wreath products that hold between constituents (even singletons).

3. Adds specific (geometric) constraints between shape constituents (e.g., set operations).

Note that we use $\Re$ to represent the 1D translation symmetry group, $O(2)$ for the 2D rotation symmetry group (both of these are continuous), and Z_n to represent the cyclic group of order n (e.g., discrete set of rotations). As an example, consider the representation of the upper case letter "A" shown in Figure 4.5.

The left side of the figure shows the basic constituents of the letter "A" – in this case five $\Re$ groups; the right hand side of the figure shows the two constraints in the WPCS: (1) C_1 describes the triangle in the letter, and (2) C_2 describes the reflection symmetry between the left and right side line segments. Note that Z_2 is the cyclic group of order 2 and models several geometric symmetries. We denote reflection by adding the annotation : Ref, and rotation by : Rot. Each of these groups has its own specific coordinate axes (e.g., the $z-$axis for rotation, and a specific line in the plane for the reflections. For the top-level Z_2 : Ref group in C_2 this axis is the $y-$axis, while for the lower level reflections, it will be the line bisecting the respective side at the points indicated in the drawing (on the left of the figure).

R_i in the figure is the i^{th} straight line segment. Note that there will be additional information added to the representation to describe the actuation processes which give rise to these constituents (see below). Compare this to the WPCS representation of the capital letter "H" shown in Figure 4.6. It can be seen that there is only one constraint in the set (the triangle is not found), and the highest level reflection symmetry describes the horizontal reflection of the entire "H" figure. Thus, the WPCS representation exposes both the similarities (e.g., the common subgraph) between the two letters, as well as the differences. Also, note that there are multiple WPCS representations for a set. For example, the letter "A" can also be represented as the two sides and the cross bar in the middle (i.e., 3 $\Re$ groups, instead of 5).

The basic WPs for letter representation are $\Re$ (straight line segments) and $O(2)$ (circles). Therefore, we have developed special analysis algorithms to produce $\Re$ and $O(2)$ hypotheses.

4.3.1 $\Re$ Hypotheses

In order to discover $\Re$ hypotheses, we use the connected component image and its skeleton image (i.e., medial axis transform). Figure 4.7 shows a skeleton overlaid on the original image. The basic logic of the $\Re$ hypothesis approach is:

```
for every pixel, p, in the skeleton
  V := skeleton pixels visible from p
  R hyp := pixel sets with p as endpt
```

Figure 4.8 shows the $\Re$ hypotheses found in a sample capital letter "A" skeleton image.

V is formed by checking visibility in terms of the a straight line of pixels in the original image connecting p and the visible skeleton pixel. p is an endpoint if it is one of the two most distant points of the projection of the points in V onto the best fit line to the points in V.

4.3.2 $O(2)$ Hypotheses

Circular sections (or parts thereof) are more difficult to find. Figure 4.9 shows the letter "C" and its skeleton. The basic logic of the $O(2)$ hypothesis approach is:

```
for every pixel,p, in the skeleton
  flow := distance of pixel from p
  T := pixels within distance 15 of p
  O(2)_hyp := best fit circles in T
```

Figure 4.10 shows the $O(2)$ hypotheses found for the sample letter "C" image. As another example, Figure 4.11 shows the lower case letter "a" image. Figure 4.12 shows the $O(2)$ hypotheses found for it. Distance here is in terms of chain code distance (8-neighbor steps).

4.4 Character Templates and Segment Classification

Given $\Re$ and $O(2)$ basic constituents, it is possible to define WPCS templates for all the character shapes; e.g., those for the letters "A" and "H" shown above. It is also possible to learn the templates by taking a set of training samples, extracting the constituents and finding the constraints between the constituents. This would involve either setting a hard threshold to produce predicates for the constraints (e.g., for when a reflection holds between two point sets), or adding a probabilistic framework. In the course of this study, we discovered that the actuation signals which encode the basic constituents provide an effective and efficient shape representation for the WPCS; this is described in the next section.

4.5 Actuation Signals as Representation

Since we do not have an embodied agent in this application, we resort here to *virtual actuators*, and in particular, a *virtual camera* for image acquisition and a *virtual hand* for shape generation. We show that a character can be represented as a wreath product constraint set which provides an abstract representation of the shape and allows for

1. recognition and classification of text characters, and

2. structured knowledge transfer from the camera actuation system to the robot hand actuation system in order to achieve shape synthesis.

4.5.1 Virtual Camera

Since we are working with images that have already been scanned, we have developed *virtual actuators* and corresponding actuation command streams for the given data acquired from the image. This works as follows:

- Each individual connected component is set in a circumscribing rectangle (Figure 4.11 shows the subwindow for the lower case letter "a").

- A virtual camera is positioned in the middle of the image and above the image by one-half the length of the longer rectangular side.

- The skeleton of the connected component is found next (see Figure 4.7).

- The camera is aimed at each of its constituent pixels in turn, and the pan and tilt angles are recorded. For example, Figure 4.13 shows the pan and tilt angles for the lower case letter "a."

4.5.2 Virtual Hand

The specific *virtual hand* considered here is an RR robot (i.e., two revolute joints). The base of the arm is located at the center of the character subwindow, and the lengths of the links are equal and set to one-fourth the length of the greater diagonal of the subwindow (see Figure 4.14). This allows the virtual hand to place its endpoint anywhere within the subwindow.

Given that a shape will need to be generated in a standard Cartesian coordinate frame, it will be necessary, in general, to learn the transform from (pan,tilt) space to $(\theta1, \theta2)$ space so as to obtain the same (x, y) point. However, in this specific case, the transform is given as:

$$c_2 = D = \frac{(dtan(\theta_{pan}))^2 + (dtan(\theta_{tilt}))^2 - 2a^2}{2a^2}$$

$$s_2 = \sqrt{1 - D^2}$$

$$\theta_2 = atan2(s_2, c_2)$$

$$\theta_1 = atan2(dtan(\theta_{tilt}), dtan(\theta_{pan})) - atan2(s_2, 1 - c_2))$$

For example, the (θ_1, θ_2) signal representation for lower case "a" is shown in Figure 4.15. In the experiments described below, we use the (pan,tilt) representation of the WPCS constituents.

4.6 Classification Based on Translation Symmetry

It is possible to represent shapes in terms of just the local translation symmetries. This means that for every pixel in the shape, the maximal translation direction is determined; this is done by finding the maximal set of pixels forming a line through the given pixel and that stay inside the shape. It is not necessary to recover the $\Re$ or $O(2)$ constituents for this approach. Statistics of the pixel-based translation symmetries provide enough information for shape classification. For experiments we use segments from the image shown in Figure 4.16. There are 1174 legitimate characters (although seven of these do not satisfy the classification assumptions – e.g., wrong Euler number, etc.); of these 1161 are correctly classified in the top five hypotheses (we allow multiple hypotheses and then reduce them when words are found).

Consider a character such as that shown in Figure 4.11. At each pixel, the direction and extent of the 1D translation is determined. Figure 4.17 shows the results for this character. Our first classification approach exploits the first order statistic of orientation. The directions are aligned with the closest of 0, 45, 90, and 135 degrees.

In order to constrain the orientations somewhat according to their 2D distribution in space, we construct a histogram for each of four sub-regions of the image: (1) upper portion, (2) lower portion, (3) left portion, and (4) right portion (in this study the portion is 2/3's). The method also uses Euler number and horizontal and vertical symmetry measures.

Figure 4.18 shows the four histograms catenated, and for comparison, the histogram from another letter "a" from a test image. In practice the four histograms from a test character, and the norm of the resulting 4-tuple is used as the distance measure (in this case, the vector was $p = [0.0425, 0.0409, 0.0459, 0.0069]$). Each unknown character (connected component) in a test image is compared to each character template, and the top five matches are kept. Using this approach, we obtained a 95% classification rate; when only the top 1, 2, 3, and 4 hypotheses are kept, the classification rate are 80%, 90%, 95%, and 98%, respectively. Figure 4.19 shows the precision and recall plot for this method.

4.7 Classification Based on Pan-Tilt Actuation Signals

It is also possible to use explicit actuation data to classify unknown characters. Given a set of pixels along the skeleton of a character, the pan and tilt angle for the *virtual camera* are found as described earlier. For the template character "a" the pan and tilt angles are shown in Figure 4.13. A distance measure between the shapes is then based on the difference of the two (pan,tilt) signals for the different characters. We use the following simple measure:

$$\mu((p_1,t_1),(p_2,t_2)) = \sum_{i\in\mathcal{I}_1} argmin_{j\in\mathcal{I}_2}(\mid (p_i,t_i) - (p_j,t_j) \mid)$$

where $\mathcal{I}_1$ and $\mathcal{I}_2$ are the index sets for points in shape 1 and 2, respectively. This is the sum of the distances to closest (pan,tilt) pair in the other set. Using this method, we obtained a 99% classification rate for the top five hypotheses; when only the top 1, 2, 3, and 4 hypotheses are kept, the classification rate are 88%, 96%, 97%, and 98%, respectively. Figure 4.20 shows the precision and recall plot for this method.

4.8 Classification Based on RR Robot Actuation Signals

We also explored the use of explicit 2-revolute joint robot actuation data to classify unknown characters. Given a set of pixels along the skeleton of a character, the two joint angles for the *virtual robot hand* are found as described earlier. For the template character "a" the pan and tilt angles are shown in Figure 4.13. A distance measure between the shapes is then based on the difference of the two (θ_1, θ_2) signals for the different characters. We use the following simple measure:

$$\mu((p_1,t_1),(p_2,t_2)) = \sum_{i\in\mathcal{I}_1} argmin_{j\in\mathcal{I}_2}(\mid (\theta 11_i, \theta 12_i) - (\theta 21_j, \theta 22_j) \mid)$$

where $\mathcal{I}_1$ and $\mathcal{I}_2$ are the index sets for points in shape 1 and 2, respectively. This is the sum of the distances to closest (θ_1, θ_2) pair in the other set. Using this method, we obtained a 98% classification rate for the top five hypotheses; when only the top 1, 2, 3, and 4 hypotheses are kept, the classification rate are 79%, 93%, 97%, and 98%, respectively. Figure 4.21 shows the precision and recall plot for this method.

We have developed a WPCS representation which is simply a list of the R WPP's, followed by the $O2$ WPP's, and then followed by the higher level wreath product symmetries found in the shape. For example, the WPCS's for the shapes in Figure 4.4 are characterized as:

- **"a"**: "O;O;Z2O;"
- **"A"**: "R;R;R;Z2R;"
- **"O"**: "O;Z2O;"
- **"e"**: "O;O;Z2O;"
- **"M"**: "R;R;R;R;Z2R;Z2R;Z2R;"
- **"2"**: "R;O;"

The next step in the process is to associate a specific generative mechanism with the shape. Here we use the virtual sensors (pan-tilt camera) and actuators which were proposed in [37]; the pan-tilt control angles for a camera trace for each of the characters are shown in Figure 4.22. Character classification for an unknown shape is started by producing the WPCS's for the shape (note there may be several). Next, these are compared at the abstract level to the character template WPCS's, and where a match is found, then the pan-tilt actuation generative data are compared. Any match at this level that is above threshold produces a character hypothesis. The final step uses the character hypotheses to produce legal word hypotheses (using a dictionary).

4.9 Experiments

The tests were run of the image shown in Figure 4.23 which resulted in 1,111 connected components to classify. The 62 templates for lower and uppercase letters and the ten digits resulted in 184 minimal cover WPCS's for the 62 characters. A total of 3,333 minimal cover

WPCS's were generated for the 1,111 connected components. The abstract wreath product filter eliminated 67% of the unknown hypotheses; note that this check only requires comparison of their wreath product string representations. The remaining hypotheses were matched to templates by a 2D pointwise comparison of their pan-tilt function values. An unknown is considered a match if the correct character is in the top 5 best pan-tilt distance matches. The classification rate is very good with this approach ($\sim$ 99%) when using the top five hypotheses.

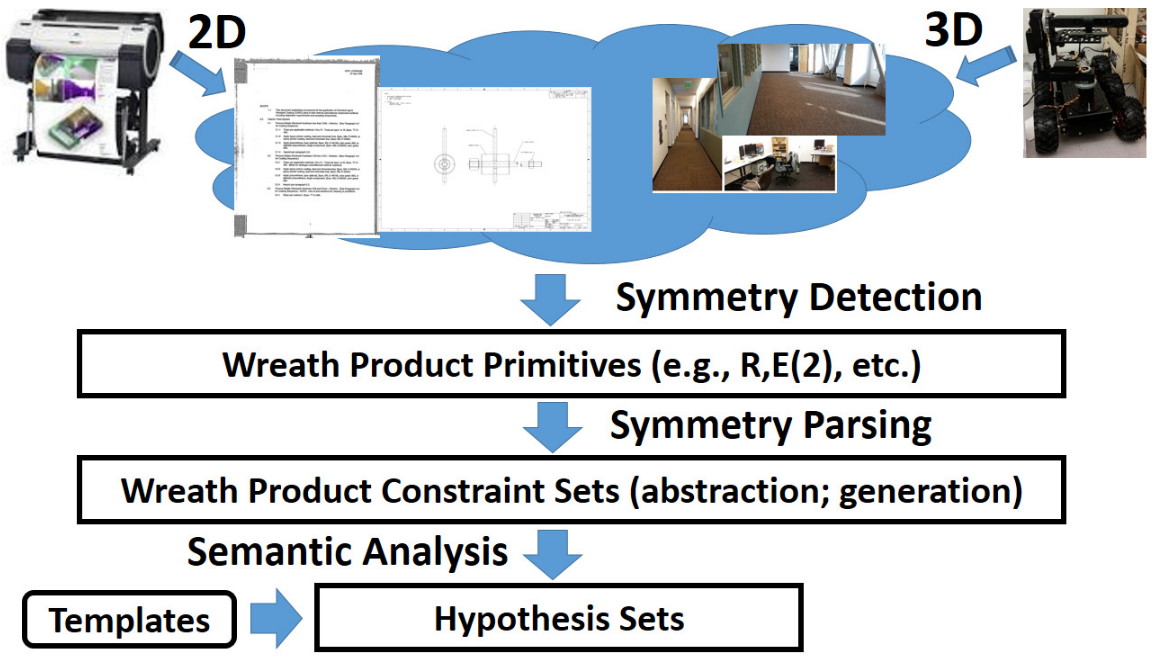
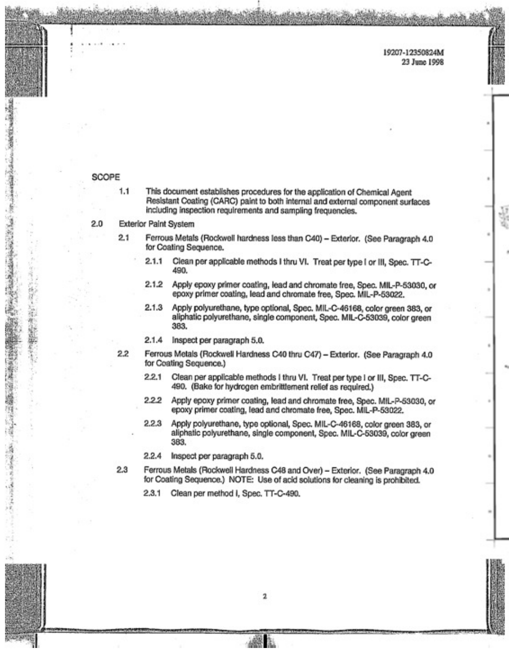

2D
3D
Symmetry Detection
Wreath Product Primitives (e.g., R,E(2), etc.)
Symmetry Parsing
Wreath Product Constraint Sets (abstraction; generation)
Semantic Analysis
Templates
Hypothesis Sets

Figure 4.1. Overall symmetry analysis flow.



19207-12350824M
23 June 1998

SCOPE

1.1 This document establishes procedures for the application of Chemical Agent Resistant Coating (CARC) paint to both internal and external component surfaces including inspection requirements and sampling frequencies.

2.0 Exterior Paint System

2.1 Ferrous Metals (Rockwell hardness less than C40) – Exterior. (See Paragraph 4.0 for Coating Sequence.

2.1.1 Clean per applicable methods I thru VI. Treat per type I or III, Spec. TT-C-490.

2.1.2 Apply epoxy primer coating, lead and chromate free, Spec. MIL-P-53030, or epoxy primer coating, lead and chromate free, Spec. MIL-P-53022.

2.1.3 Apply polyurethane, type optional, Spec. MIL-C-46168, color green 383, or aliphatic polyurethane, single component, Spec. MIL-C-53039, color green 383.

2.1.4 Inspect per paragraph 5.0.

2.2 Ferrous Metals (Rockwell Hardness C40 thru C47) – Exterior. (See Paragraph 4.0 for Coating Sequence.)

2.2.1 Clean per applicable methods I thru VI. Treat per type I or III, Spec. TT-C-490. (Bake for hydrogen embrittlement relief as required.)

2.2.2 Apply epoxy primer coating, lead and chromate free, Spec. MIL-P-53030, or epoxy primer coating, lead and chromate free, Spec. MIL-P-53022.

2.2.3 Apply polyurethane, type optional, Spec. MIL-C-46168, color green 383, or aliphatic polyurethane, single component, Spec. MIL-C-53039, color green 383.

2.2.4 Inspect per paragraph 5.0.

2.3 Ferrous Metals (Rockwell Hardness C48 and Over) – Exterior. (See Paragraph 4.0 for Coating Sequence.) NOTE: Use of acid solutions for cleaning is prohibited.

2.3.1 Clean per method I, Spec. TT-C-490.

2

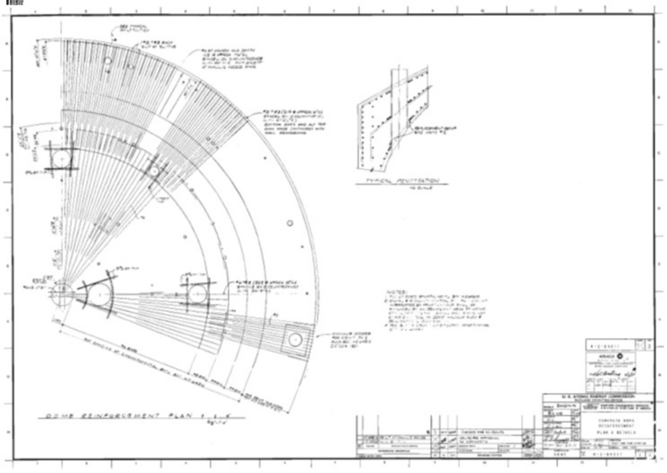


Figure 4.2. Two CAD drawings: (left) text image that is included with CAD to explain how to paint the structure, and (right) a hand-drawn design of a nuclear storage facility.

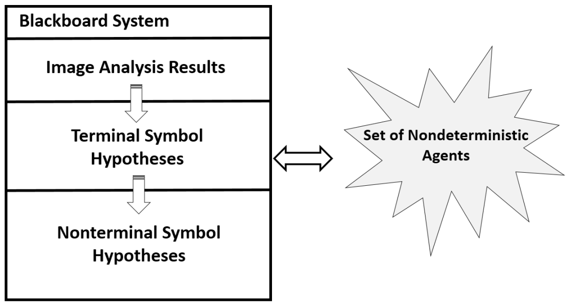

Blackboard System
Image Analysis Results
Terminal Symbol Hypotheses
Nonterminal Symbol Hypotheses
Set of Nondeterministic Agents

Figure 4.3. The ENDAS system.

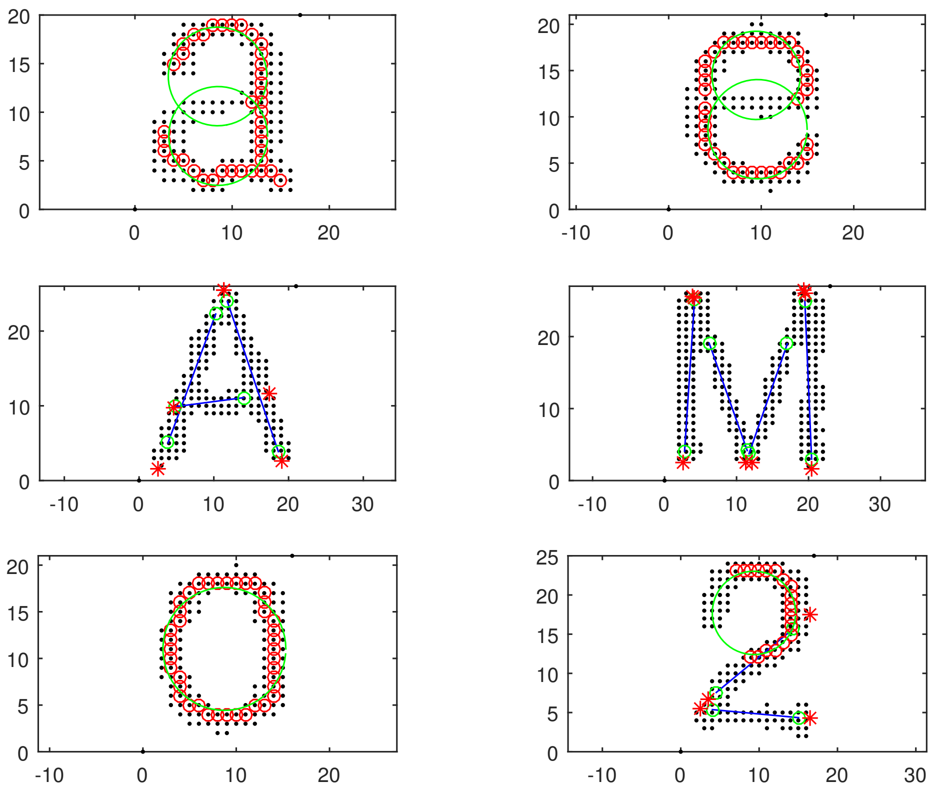


Figure 4.4. Example WPP minimal cover sets.

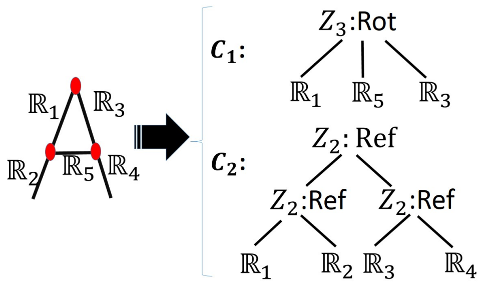


Figure 4.5. Wreath Product Constraint Set for letter "A."

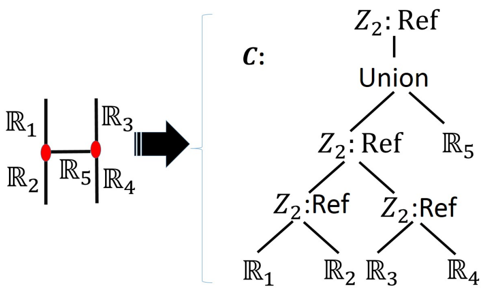


Figure 4.6. Wreath Product Constraint Set for letter "H."

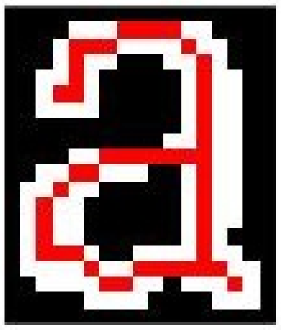


Figure 4.7. Skeleton of lower case letter "a" overlaid on original image.

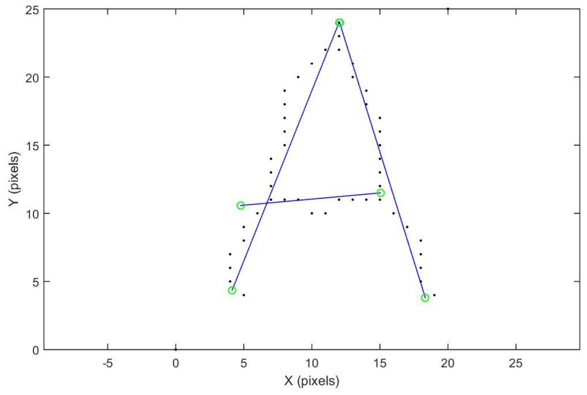


Figure 4.8. $\Re$ hypotheses for letter "A."

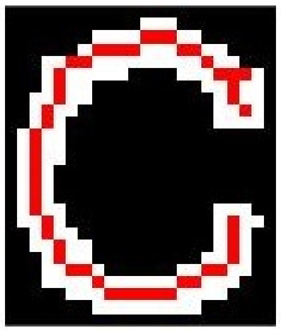


Figure 4.9. Skeleton overlaid on original image for letter "C."

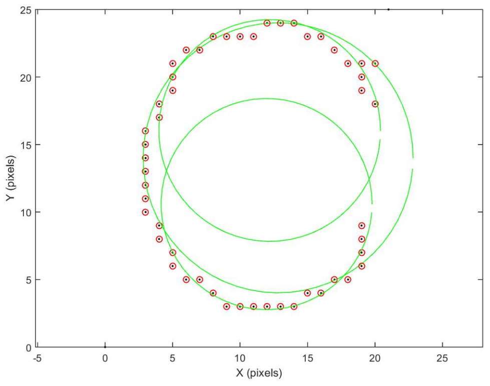


Figure 4.10. $O(2)$ hypotheses for letter "C."

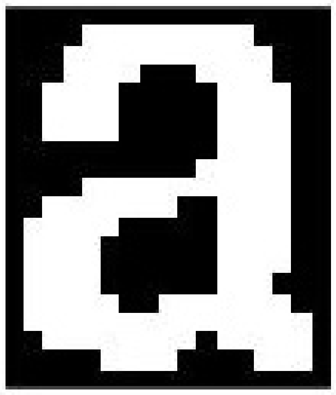


Figure 4.11. Image of lower case letter "a."

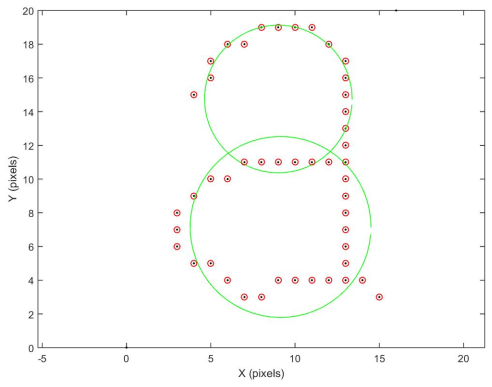


Figure 4.12. $O(2)$ hypotheses for letter "a."

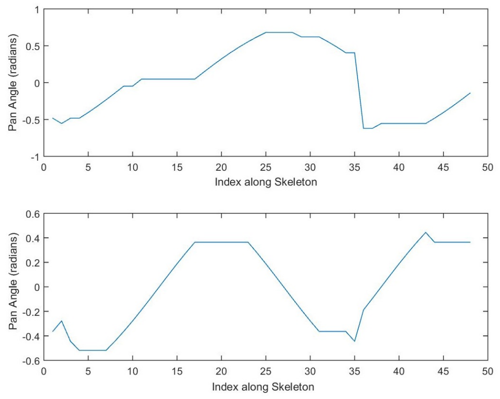


Figure 4.13. The *pan* and *tilt* signals for lower case "a."

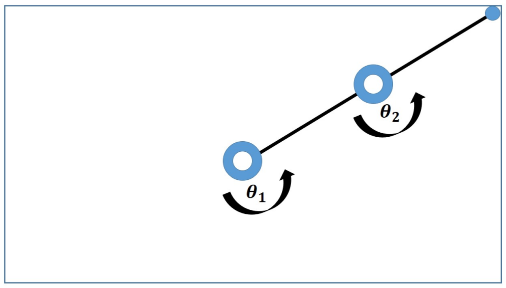


Figure 4.14. A simple 2-revolute joint (RR) robot hand.

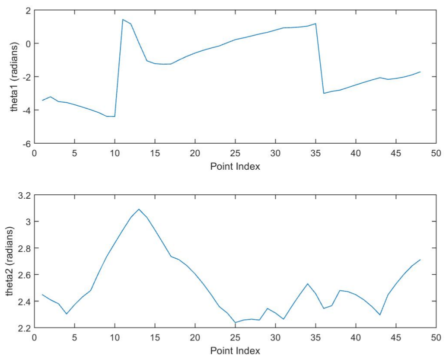
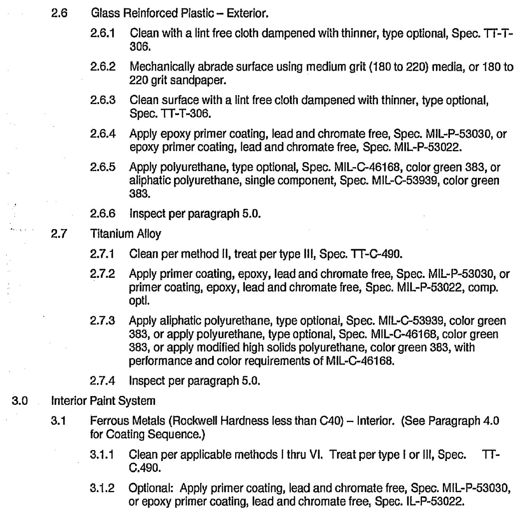


Figure 4.15. Lower case letter "a" representation by (θ_1, θ_2).



2.6 Glass Reinforced Plastic – Exterior.

2.6.1 Clean with a lint free cloth dampened with thinner, type optional, Spec. TT-T-306.

2.6.2 Mechanically abrade surface using medium grit (180 to 220) media, or 180 to 220 grit sandpaper.

2.6.3 Clean surface with a lint free cloth dampened with thinner, type optional, Spec. TT-T-306.

2.6.4 Apply epoxy primer coating, lead and chromate free, Spec. MIL-P-53030, or epoxy primer coating, lead and chromate free, Spec. MIL-P-53022.

2.6.5 Apply polyurethane, type optional, Spec. MIL-C-46168, color green 383, or aliphatic polyurethane, single component, Spec. MIL-C-53939, color green 383.

2.6.6 Inspect per paragraph 5.0.

2.7 Titanium Alloy

2.7.1 Clean per method II, treat per type III, Spec. TT-C-490.

2.7.2 Apply primer coating, epoxy, lead and chromate free, Spec. MIL-P-53030, or primer coating, epoxy, lead and chromate free, Spec. MIL-P-53022, comp. optl.

2.7.3 Apply aliphatic polyurethane, type optional, Spec. MIL-C-53939, color green 383, or apply polyurethane, type optional, Spec. MIL-C-46168, color green 383, or apply modified high solids polyurethane, color green 383, with performance and color requirements of MIL-C-46168.

2.7.4 Inspect per paragraph 5.0.

3.0 Interior Paint System

3.1 Ferrous Metals (Rockwell Hardness less than C40) – Interior. (See Paragraph 4.0 for Coating Sequence.)

3.1.1 Clean per applicable methods I thru VI. Treat per type I or III, Spec. TT-C.490.

3.1.2 Optional: Apply primer coating, lead and chromate free, Spec. MIL-P-53030, or epoxy primer coating, lead and chromate free, Spec. IL-P-53022.

Figure 4.16. Image used in experiments.

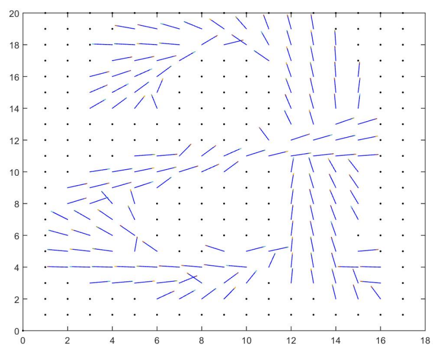


Figure 4.17. Translation symmetry directions for lower case letter "a."

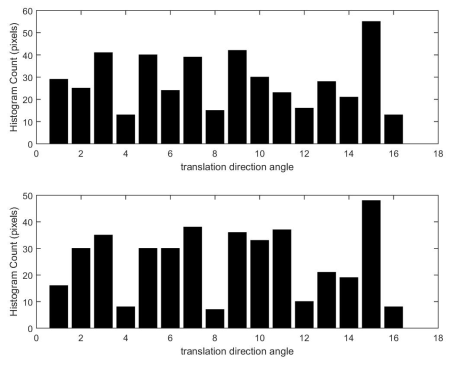


Figure 4.18. Comparison of combined symmetry translation direction histograms from four subwindows in two different "a" segments.

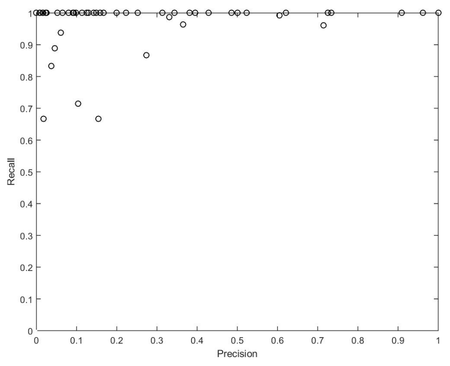


Figure 4.19. Precision and recall plot for symmetry translation classification.

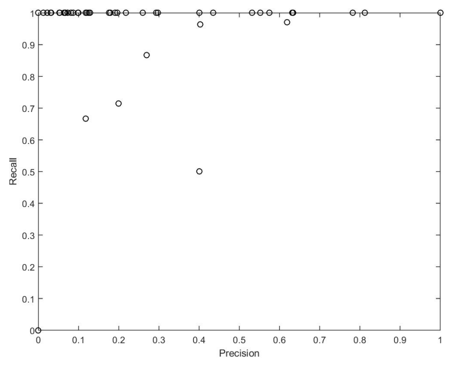


Figure 4.20. Precision and recall plot for pan-tilt actuation data classification.

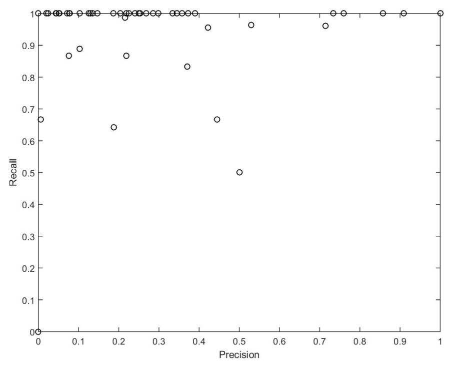


Figure 4.21. Precision and recall plot for RR robot actuation data classification.

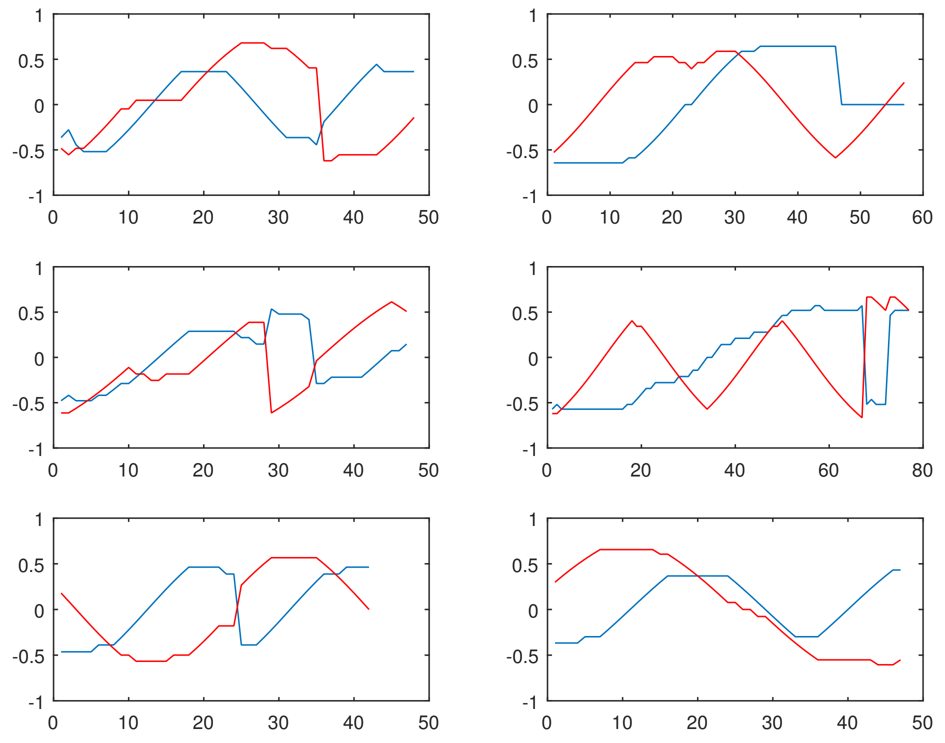
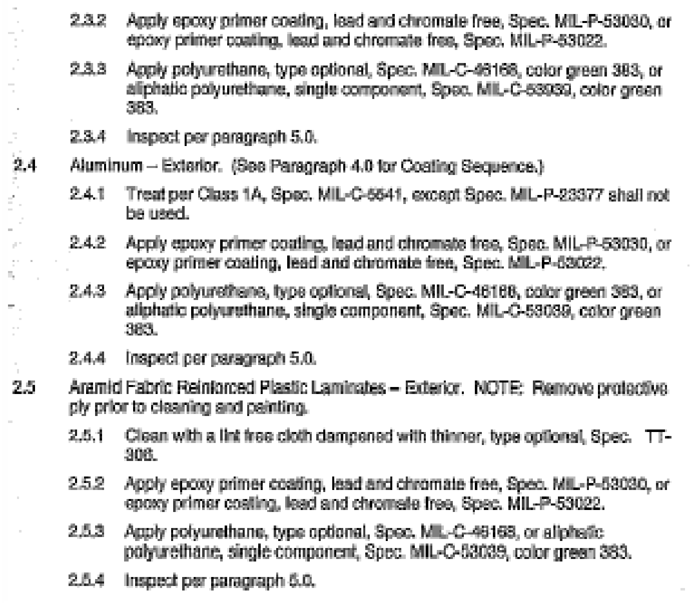


Figure 4.22. Pan tilt control angles for the characters in Figure 4.4.



2.3.2 Apply epoxy primer coating, lead and chromate free, Spec. MIL-P-53030, or epoxy primer coating, lead and chromate free, Spec. MIL-P-53022.

2.3.3 Apply polyurethane, type optional, Spec. MIL-C-46168, color green 383, or aliphatic polyurethane, single component, Spec. MIL-C-53039, color green 383.

2.3.4 Inspect per paragraph 5.0.

2.4 Aluminum – Exterior. (See Paragraph 4.0 for Coating Sequence.)

2.4.1 Treat per Class 1A, Spec. MIL-C-5541, except Spec. MIL-P-23377 shall not be used.

2.4.2 Apply epoxy primer coating, lead and chromate free, Spec. MIL-P-53030, or epoxy primer coating, lead and chromate free, Spec. MIL-P-53022.

2.4.3 Apply polyurethane, type optional, Spec. MIL-C-46168, color green 383, or aliphatic polyurethane, single component, Spec. MIL-C-53039, color green 383.

2.4.4 Inspect per paragraph 5.0.

2.5 Aramid Fabric Reinforced Plastic Laminates – Exterior. NOTE: Remove protective ply prior to cleaning and painting.

2.5.1 Clean with a lint free cloth dampened with thinner, type optional, Spec. TT-306.

2.5.2 Apply epoxy primer coating, lead and chromate free, Spec. MIL-P-53030, or epoxy primer coating, lead and chromate free, Spec. MIL-P-53022.

2.5.3 Apply polyurethane, type optional, Spec. MIL-C-46168, or aliphatic polyurethane, single component, Spec. MIL-C-53039, color green 383.

2.5.4 Inspect per paragraph 5.0.

Figure 4.23. Test image.

CHAPTER 5

WREATH PRODUCT 3D DATA ANALYSIS: LOCALIZATION AND MAPPING, ROBOT KNOWLEDGE SHARING AND BENCHMARK RESULTS

We explored wreath product application to 3D range data by implementing WPCA on a differential-drive robot. This chapter covers the localization and mapping, multirobot communication, and performance of our architecture compared against similar systems.

5.1 Navigation Benchmarking

In order to judge the effectiveness and robustness of operation of the WPCA representational framework in the real world, we demonstrate its performance in terms of a benchmarking protocol. Benchmarking protocols for a robot operating in a real dynamic environment are more difficult to engineer and implement, compared to those in machine learning or computer vision benchmarks, that evaluate algorithms operating on datasets. In order to solve this problem Sprunk et al. [45] defined, for the first time, a detailed benchmarking protocol for evaluation of robot indoor navigation algorithms. However, their benchmarking protocol is specifically designed for evaluating the entire navigation software of a robot, which also includes localization. However, navigation is not the primary focus of this work. Hence we use only the localization performance of their experiments as one of the benchmarks for determining the performance of our navigation approach using our framework and data representation. In order to better evaluate our benchmarking performance, we also compare against three other systems: (1) The Fast Sampling Plane Filtering (FSPF) algorithm by Biswas et al. [46] on a Kinect-based robot system, (2) Microsoft Research's P1 robot platform which utilizes Sprunk's framework, and (3) Endres' hand-held Kinect-based SLAM implementation on a dataset. They are briefly explained as follows:

1. **Sprunk's protocol:** In their benchmarking protocol, Sprunk et al. [45] attempt to standardize the methodology used to assess the robustness of navigation algorithms. To this end, they use a Pioneer P3-DX differential drive robot as a *reference robot*, to which all other robots and their navigation algorithms are compared. Their protocol uses a Pioneer P3-DX differential-drive robot with a SICK LiDAR sensor. To model the protocol for all possible users, they standardize most of the entities and processes involved in testing a navigation system, for example, the environment, the ground truth determination, environment dynamics, testing periods (simulated *months*, *years*, etc.), performance statistics, etc. As mentioned earlier we compare their localization performance to the WPCA system, and we try to model the environment as they suggest, namely testing the localization in an office and an atrium (discussed below).

2. **FSPF:** The system that resembles our framework the closest in terms of sensor, and methodologies and algorithms used, is the Fast Sampling Plane Filtering (FSPS) by Biswas et al. from CMU [46–49]. Unlike our system, their system's main purpose is navigation and obstacle avoidance, and it does not implement a higher-level representational framework or behaviors. They predominantly use lines which are down-projected planes discovered in the data, for localization. Our system uses a combination of plane, line and point wreath products for localization. However, they use an older version of the Kinect v2 sensor (Kinect v1) and work on point clouds, similar to our system. They also process point clouds in a way similar to us, by segmenting them using RANSAC, and determining their parameters. The major difference between the two systems is that they compare down-projected planes (lines) to an existing 2D vector map of the floor plan, determine which lines are visible, and localize based on that, whereas we generate the floor plan from scratch by exploring the environment.

3. **MSR:** Microsoft Research's experiments employ Sprunk's experimental benchmarking protocol, although in a different environment and on a different robot - the Microsoft Robotics Prototype 1 (MSR-P1).

4. **Endres' system:** This system tracks a hand-held Kinect v1 sensor and generates a

dense 3D model of the environment. Their approach extracts and matches visual features (SURF, SIFT, ORB) from RGB data of consecutive frames, and evaluates the depth images for these frames to determine 3D correspondences between them. They use a published benchmark dataset [50] which contains the RGB-D images from Kinect with time-synchronized ground truth poses.

5.1.1 Robot

For our experiments, we use the SYMMBOT (shown in Figure 5.1). SYMMBOT is a differential-drive robot based on an iRobot Create platform (the Matlab toolbox for iRobot Create [35] was used to issue commands to the Create). It has a Kinect V2 (time-of-flight RGB-D depth sensor). The architecture is implemented on an IBM Thinkpad, as shown in Figure 5.1. It runs Linux (Ubuntu 14.04, 64-bit), has an 2.50GHz Intel Core i5, and 8 GB RAM. Symmbot is a mobile robot; it needs an external power supply cable, as shown in the figure, to power the Kinect V2, as the robot's battery is insufficient for this purpose. The laptop has its own battery, and the robot battery is used to drive the robot.

5.1.2 Environment

The environment is divided into two types of *areas*:

- **Atrium**: This is a predominantly open space (at least 90 % of its surface is without furniture). An open area in Warnock Engineering Building (WEB), as shown in Figure 5.2 top, serves as the atrium in the experiments described here.

- **Office:** This is an area densely occupied by desks, office chairs, and shelves. The Virtual Reality (VR) lab in Warnock Engineering Building (WEB), as shown in Figure 5.2 bottom, serves as the office space.

5.1.3 Ground Truth

To evaluate the performance of a robot this protocol requires ground truth. In Sprunk's and MSR's method (both employ the same protocol), this ground truth is obtained by placing markers above the robot that identity the position of way-points. The robot has an upward-facing camera that locates these markers (which are checkerboard patterns at a height of 2.45m above ground). The experimenter initially manually drives to each way point and records its position within the reference software. In Biswas' FSPF method, their system is compared against three different Collective Gradient Refinement (CGR)-based laser rangefinder localization algorithms which simulate laser range scans [51]. Endres' system used RGB-D datasets where each frame is time-synchronized to the ground truth pose which is obtained using a high-accuracy motion-capture system. To generate our benchmark tests, the experimenter manually marks the position and orientation of the robot, with respect to the world frame origin, each time the robot stops, and measures it using a tape measure. The robot localization results are compared to this hand-measured ground truth.

5.2 Benchmark Results

The statistics used to evaluate the navigation performance of the WPCA are the location and orientation error of the robot (see [52]). Tables 5.1 and 5.2 show the performance measures (in terms of localization error in x, y-location in meters and angular error (orientation in degrees), respectively, for the mean, median, minimum, and maximum error encountered in the *lab* and *Atrium* environment when we use the RANSAC method for WP discovery, compared to other systems. Similarly, Tables 5.3 and 5.4 show the performance measures (in terms of localization error in x, y-location in meters and angular error (orientation in degrees), respectively, for the mean, median, minimum, and maximum error encountered in the *lab* and *Atrium* environment when we use the pixel-based approach method for WP discovery. Statistics that were not available for some of these systems has been marked with an "X." We observe that WPCA system performs better than the Pioneer, MSR and FSPF systems, and is comparable to the Endres'12 system, with respect to the localization error. Our system does not perform so well for the orientation error, compared to the systems that provided orientation error data. However, it must be noted that our algorithms have very high tolerances for landmark correspondence since landmark rediscovery is more important to us than filtering out landmarks that are fairly close to one another, but which might not meet tight tolerances, which tends to impact the orientation measures significantly.

5.3 Floor Mapping

Chapter 3, section 3.7 describes the various goals (materialized as plans) that the robot can achieve. One of these goals is mapping the environment. We describe the plan used to achieve this in detail, and demonstrate the workings of our motion and landmark localization algorithms that are used to localize the robot and account for motion and perception uncertainties. The finite state machine for the floor mapping plan is shown in Figure 5.3. This automaton works as follows:

5.3.1 Start State

This state starts off by acquiring data and then moving to state 1.

5.3.2 State 1

This state incorporates the following functions that segment the floor plane and identify its boundary, as well as identify the next point the robot should move to.

5.3.2.1 Clean floor plane

RANSAC is an iterative method that selects points randomly and assign them to planes that fit the plane model with least error. In WPCA, points can belong to multiple planes, given their plane parameters. For example, in Figure 5.4, the green plane illustrates the floor whereas the blue plane illustrates a vertical wall. The red points on the floor should belong to the floor plane, but got assigned to the blue plane since they also satisfy the plane equation of the wall (since it is an infinite plane mathematically, it extends beyond the points that are assigned to it, as depicted by the dotted line).

To avoid this ambiguity, all points that lie below a certain z-threshold, are checked against the plane equation of the floor. For example,

$$ax + by + cz = d \tag{5.1}$$

is the equation of a 3D plane, where a, b, c are components of the vector normal to the plane, d is the distance of the plane from the origin, and $[x, y, z]^T$ is any point that satisfies this equation. For the floor plane, $a = 0, b = 0$, and $d = 0$, hence we only retain points that have $|z| < threshold$.

5.3.2.2 Get floor points

Floor points detected using RANSAC might be a subset of all the floor points that can be seen by the camera. Thus, we use an additional step to get as many valid points into the floor plane as possible. Our DEPTH_TO_WP algorithm in subsection 3.6.3 gives us the parameters $[a, b, c, d]^T$ of an $[R \times R]$ plane and a subset of points that lie in the floor plane. We first transform points in the depth image ${}^R[x_i, y_i, z_i], i = 1 : N$, from the robot frame to the world frame, using the ${}^W T_R$ transform. We then check for points ${}^W[x_i, y_i, z_i], i = 1 : N$, if $|z| < threshold$; if so, the point gets assigned to the floor plane. This expands the currently known floor points and gives a more complete set of points that belong to the floor. Figure 5.5 illustrates this in practice with one example. Figure 5.5(a) shows the RGB image registered with the depth image, Figure 5.5(b) shows the corresponding x, y, z point cloud run through the distance function as mentioned above, and the resulting segmented floor as white pixels, and Figure 5.5(c) shows Canny edge detector applied to the segmented floor pixels to yield the boundary of the floor in this image. Figure 5.6 demonstrates similar process for another location.

5.3.2.3 Get boundary points

The boundary points of the floor can be found using the intersection of a plane with the floor. Algorithm 8 below illustrates the process to determine floor boundary. This algorithm gets all the floor points from the belief that contains the floor information. It also retrieves all the beliefs in the belief set that are planes (lines 1 and 2, respectively). Thereafter, for every point in the current nonfloor plane $[R \times R]_i$, the line intersection of this plane with the floor plane $[R \times R]_f$ is found as R_{if} (line 5). For every point p_i on the floor plane $[R \times R]_f$, we check if it satisfies the equation of the line R_{if}, using the function Ξ, and those that satisfy are added to ι. Γ contains these points for every plane $[R \times R]_i$ that intersect with the floor. To further ensure the completeness of the floor plane boundary, we run all points p_i in the point cloud data, through the function Ξ (line 11), which ultimately gets added to Γ, giving us a more complete floor plane boundary. Note that we have the data not only in the point cloud format, but also the correspondence of each point to a pixel location, as well as its RGB value. Thus, we have the information $[x, y, z, i, j, R, G, B]$, where i and j are the row and column, per pixel at our disposal, which is how Figures 5.5

Algorithm 8: FIND FLOOR BOUNDARY.

```
Data: $\iota \leftarrow \{\}$, $\Gamma \leftarrow \{\}$
1  $F \leftarrow$ get_all_floor_points();
2  $[LMP]$ = get_all_Plane_landmarks(Beliefs);
3  for all planes $[R \times R]_i$ in LMP do
4      for floor plane $[R \times R]_f$ do
5          $R_{if} \leftarrow I([R \times R]_f, [R \times R]_i)$;
6          for points $p_i$ in $[R \times R]_f$ do
7              $\iota \leftarrow \Xi(p_i, R_{if})$;
8          end
9          $\Gamma \leftarrow \iota$;
10         for all points $p_i$ in $[x_i, y_i, z_i]^T \in$ Point Cloud do
11             $\iota \leftarrow \Xi(p_i, R_{if})$;
12         end
13         $\Gamma \leftarrow \iota$;
14     end
15 end
16 return $\Gamma$
```

and 5.6 are generated.

5.3.2.4 Identify next point

In order to follow a path on the floor we utilize the concept of the wall following algorithm for maze solving [53–56]. The gist of this method is that, given a maze that has a solution, if a robot follows the left wall (turning as the wall turns), eventually it will reach the exit; i.e., the solution of the maze. If we consider the floor as a maze, and start at the world frame location and keep following the left wall, eventually we reach the starting location (the world frame location) again, and this completes the floor mapping. Since the robot does not have a range sensor on the side, it relies on its front-facing depth sensor. Thus, at every step, it turns towards the wall and determines the next point to move to. The current step in the floor mapping plan generates - given the boundary pixels generated by the previous step - the next point that the robot must move to, which will be the right-most visible point of the connected floor boundary, since we are following the left wall. The right-most visible point can either be the right-most edge of the visible image as the floor boundary disappears towards the right, it can be a jump edge point, or it can be a break in the floor boundary generated by the previous step as a result of noisy data.

Figure 5.7 depicts the process of the wall following approach along with finding the next point to move to.

The robot is at location 1 initially, looking at the world frame, with origin O and X, Y axes depicted in the top left of the image. The solid black line is the floor boundary, as seen from above. Once the world frame is determined and control is passed to the floor mapping algorithm, the $find\_next\_point()$ function gets called. This function finds the right-most point of the floor boundary visible in the currently acquired Kinect data. This floor boundary is depicted as the set of dark (0) pixels in Figure 5.5(c), which is the edge traced from left to right to find the next point (x, y, z) associated with a pixel (i, j) that satisfies the conditions stated above. Initially the starting pixel on the floor boundary is chosen as the pixel among the edge pixels in the current view, whose associated point is closest to the world origin O. Starting from this pixel, we move to any edge pixels that neighbor the current pixel (i, j) on its right side. In the next cycle this neighboring pixel is the current pixel, and the process is repeated until:

1. A pixel does not exist on the right-side neighborhood of the current pixel, or,
2. The difference between the depth (z) value of the point associated with the next pixel is significantly different than the one associated with the current pixel, indicating a jump edge (based on empirical evidence this difference threshold is chosen to be 0.15 m.).

Thus, the edge is traced until one of the conditions above is satisfied, and the (x, y, z) point associated with the current pixel (i, j) is chosen as the next point. The *next point* is depicted as the yellow point 1 on the floor boundary. When control reaches this function again in the future, point 1 will be considered as the starting point and the process repeated for all the other *next points*.

5.3.2.5 Get action set

After a point, say the yellow point 1 depicted by p shown in Figure 5.8, is selected as the *next point* to be visited, an action set must be generated that contains a set of commands that the robot must execute so that it moves in such a way that the robot is directly looking at the point 1, and its orientation is perpendicular to the floor boundary around this point.

The idea behind this approach is that the robot needs to expand its view around the area of the *next point*, so that it can continue the process of finding the boundary further ahead. Figure 5.8 illustrates this process.

After the *next point* 1 is found, the robot must find a point to move to (depicted by the red dot p_2), such that it is looking at the point 1 directly. Consider the short line segment, determined by the range scan function discussed in Chapter 4, depicted by $p_1 p$. Given the vector $\bar{p_1 p}$, the x, y components, in the floor plane, of the vector orthogonal to these two vectors, given by $\bar{n_1}$ can be found as

$$\begin{bmatrix} \bar{n_1}(x) \\ \bar{n_1}(y) \end{bmatrix} = \begin{bmatrix} \cos(\frac{\pi}{2}) & \sin(\frac{\pi}{2}) \\ -\sin(\frac{\pi}{2}) & \cos(\frac{\pi}{2}) \end{bmatrix} \begin{bmatrix} p_2(x) - p_1(x) \\ p_2(y) - p_1(y) \end{bmatrix} \tag{5.2}$$

Once $\bar{n_1}$ is known, the point to which the robot must move is simply this vector (direction), scaled by the distance d. Based on empirical evidence, this distance d is set to 1.5 m which is neither too close for depth data loss, nor too far for ambiguous depth data acquisition and noise.

Once this point on the floor (red dot p_2) is known, the robot actions to rotate clockwise θ_1, translate t, and rotate counter-clockwise θ_2 can be found as follows:

$$\theta_1 = \cos^{-1}\left(\frac{\bar{v_1} \cdot \bar{v_2}}{(|\bar{v_1}| \cdot |\bar{v_2}|)}\right) \tag{5.3}$$

$$t = |\bar{v_2}| \tag{5.4}$$

$$\theta_2 = \pi - \cos^{-1}\left(\frac{\bar{n_1} \cdot \bar{v_2}}{(|\bar{n_1}| \cdot |\bar{v_2}|)}\right) \tag{5.5}$$

where $\bar{v_1}$ is the current robot heading, and

$$\bar{v_2} = p_2 - R$$

is the vector from current robot location R to point p_2. These three actions are returned as action sets for the robot to execute. Note that these action sets do not necessarily orient or translate the robot exactly normal to $\bar{n_1}$, owing to noisy data and erroneous robot movements, but the Kinect's 70° horizontal field of view is wide enough that the subsequent portions and points (e.g., the yellow dot 2) are seen after these motions are executed. It should be noted that since the data is noisy, especially at the boundary where the floor meets the wall, depending on the features around the robot some points may be discarded due to distance to the sensor or lighting effects, resulting in breaks in the boundary.

Figure 5.9 shows the action generation process on real data for the starting location of the robot. The initial pose is ($[x, y, \theta]^T$, where θ is in radians):

```
initial_pose =

    0.9102
    1.4062
    2.5119
```

and is shown as a red dot representing location and a red arrow representing orientation. The blue points are floor points, and their density decreases as distance from the sensors increases; i.e., points farther away are less dense hence we see the pattern at each location as the robot moves. The X and Y axes are shown as red and green arrows, respectively. Points p and p_1 correspond to those shown in Figure 5.8, whereas *Next Location* corresponds to point p_2 in Figure 5.8. The black points near the corner are the floor boundary points determined so far, and they appear as a thick edge because the margin for selecting floor points is set to 0.1 m. The Kinect data is especially noisy where the floor meets the wall (it tends to shift upwards as we get closer to the wall). This can result in breaks in the contiguous boundary, and a larger margin ensures maximum contiguous boundary. For this particular location, the action set generated is:

```
action_set =

         0  -81.8805         0
         0         0    1.5362
         0  121.6262         0
```

which indicates

(1) rotate clockwise by $-81.8805°$, to face *Next Location*.

(2) translate 1.5362 m to arrive at *Next Location*.

(3) rotate counter-clockwise by $121.6262°$ degrees, to face point p.

5.3.2.6 Set start point to the next point

Once the robot is in position and is looking at the *next point*, this point is set as the start point when this algorithm is called again by state 8. The next state is set to 3, for executing the action sets generated by the method stated above.

5.3.3 State 3

The robot executes the first rotation by θ_1 and sets next state to 4.

5.3.4 State 4

The robot acquires Kinect data and sets next state to 5.

5.3.5 State 5

The robot executes the translation by t m, and sets next state to 5.

5.3.6 State 6

The robot acquires Kinect data and sets next state to 7.

5.3.7 State 7

The robot executes the second rotation by θ_2 and sets next state to 8.

5.3.8 State 8

This state is similar to state 2, where it generates the next action set. However, this state also checks if the *next point* is sufficiently close to the *start point*, which implies that the robot mapped the entire space and reached the world frame origin again. Of course, this check does not get called in the first iterations of this FSM, so that the plan does not complete prematurely.

5.3.9 State 9

If it is detected that the robot has completed the task by coming back to the origin, the boundary points are updated with the data from the last iteration, and the plan is marked as "complete."

The final floor boundary generated is shown in Figures 5.10(a) (which shows the boundary as generated) and 5.10(b) (which demarcates the boundary to elaborate on the breaks in the boundary generated due to noise).

In Figure 5.10(b), the world frame axes are shown as red and green arrows respectively. The red dotted lines at the top mark the start of the corridor, and serve as *virtual walls* at the software level that prevent the robot from further exploring the corridors. The blue boxes near the glass windows of the atrium indicate noisy data generated due to intense sunlight, thus, no data is generated for those patches on the floor, resulting in breaks or noisy edges. Finally, the red boxes marked with red cross are areas on the floor behind the earthquake resistant metallic pillars that cannot be accessed, which prevents the robot from generating the floor boundary at that area. Note that robot is blocked in software at restricted or inaccessible spaces.

5.4 Mapping the Environment

Environment mapping functionality follows the same plan as the floor mapping, although when we map the environment we consider not just the floor, but all the WPs that are generated along the way. The environment mapped is shown in Figure 5.2. This environment is on the second floor of the Warnock Engineering Building at the University of Utah. Dynamic environment and moving obstacle detection and avoidance is out of the scope of this work, and thus, this environment is ideal for our operations, as it is a low-traffic region. We have mapped both the cluttered environment of the lab as well as the uncluttered environment of the atrium. However, the lab is a small room and due to the large field of view of the Kinect is mapped in a few steps. Therefore, most of our experiments are performed in the large atrium ($\approx 12 \times 5\ m^2$).

5.4.1 Motion Localization

Our localization algorithm (based on [57, 58]) works in two steps as indicated in Algorithm 2 (line 3 corresponds to state update using current pose and control data, also known as motion update, and line 5 corresponds to state update using landmark correspondences, if any). These two steps are elaborated on, in the following subsections.

Algorithm 9 describes the procedure to update the robot pose based on control commands sent to the robot. The algorithm accepts the current state μ, covariance Σ, and control u_t as the arguments. Based on whether a translation motion occurred ($v_t \neq 0$) or a rotational motion occurred ($\omega_t \neq 0$), the state is updated accordingly. This algorithm is a standard motion model update. Since the robot has only two discrete motions - translation and rotation - and not a combination of both at the same time, this model is simple.

Algorithm 9: UPDATE ROBOT POSE.

```
1  $[v_t, \omega_t]^T \leftarrow u_t$;
2  $F_x \leftarrow I_{3,3}$;
3  if $\omega_t = 0$ then
4      $T \leftarrow \begin{bmatrix} \delta_t v_t \cos(\mu_{t-1,\theta} + \frac{\pi}{2}) \\ \delta_t v_t \sin(\mu_{t-1,\theta} + \frac{\pi}{2}) \\ 0 \end{bmatrix}$;
5  else
6      $T \leftarrow \begin{bmatrix} -\frac{\omega_t}{v_t} \sin \mu_{t-1,\theta} + \frac{\omega_t}{v_t} \sin(\mu_{t-1,\theta} + \omega_t \delta_t) \\ -\frac{\omega_t}{v_t} \cos \mu_{t-1,\theta} - \frac{\omega_t}{v_t} \cos(\mu_{t-1,\theta} + \omega_t \delta_t) \\ \omega_t \delta_t \end{bmatrix}$;
7  end
8  $\bar{\mu}_t = \mu_{t-1} + F_x \times T$;
9  $G_t = I + F_x^T \begin{bmatrix} 0 & 0 & -\frac{v_t}{\omega_t} \cos \mu_{t-1,\theta} + \frac{v_t}{\omega_t} \cos(\mu_{t-1,\theta} + \omega_t \delta t) \\ 0 & 0 & -\frac{v_t}{\omega_t} \sin \mu_{t-1,\theta} + \frac{v_t}{\omega_t} \cos(\mu_{t-1,\theta} + \omega_t \delta t) \\ 0 & 0 & 0 \end{bmatrix}$;
10 $\bar{\Sigma}_t = G_t \Sigma_{t-1} G_t^T + F_x^T R_t F_x$;
11 $\mu_t = \bar{\mu}_t$;
12 $\Sigma_t = \bar{\Sigma}_t$;
13 return $\mu_t, \Sigma_t$
```

5.4.2 Wreath Product (WP Landmark) Localization

WP landmark localization (shown in Algorithm 10) matches – using function Ψ, and threshold τ – newly discovered landmarks, Z_t (transformed from robot frame into the world frame), to existing WP landmarks in beliefs (LMs), and adds the indexes to correspondence C (lines 1 – 3) if they are close enough (Algorithm 11 elaborates the process of landmark correspondence detection). We localize the robot based on the geometric constraints that must be satisfied if the type of WP landmark correspondence is known. For example, lines (parallel to the floor) and (nonfloor) planes intersecting with the floor plane form line constraints (lines 15, 22), i.e., the robot has to be on a line that is at a certain distance from, and parallel to the observed line. If multiple constraint lines are present, pairs of these lines will intersect to give location hypotheses, whereas for single lines the closest point on the constraint line is our possible location (added to the hypotheses).

Algorithm 10: LANDMARK LOCALIZATION.

```
Data: $C \leftarrow \{\}$, $\mathcal{H}_\mu \leftarrow [\mu_{t,x,y}, \sigma_\mu]$, $\mathcal{H}_\theta \leftarrow [\mu_{t,\theta}, \sigma_\theta]$ Observations
input: $Z_t \leftarrow$ Observations
output: $\mu_t, \Sigma_t$
1  $[LM]$ = get_all_WP_landmarks(Beliefs);
2  ${}^W T_R$=get_transform(Beliefs);
3  $C \leftarrow$ LANDMARK_CORRESPONDENCES($Z_t, LM$);
4  for all point observations $E^i_{z_t}$ and LMs $E^j_{lm}$ in C do
5      $r \leftarrow norm([0,0],[x^i_{z_t}, y^i_{z_t}])$;
6      circles $\leftarrow [x_{E^j_{lm}}, y_{E^j_{lm}}, r]$;
7  end
8  for all line observations $R^i_{z_t}$ and LMs $R^j_{lm}$ in C do
9      if $R^j_{lm} \perp floor$ then
10         $ip \leftarrow I(R^i_{z_t}, floor)$;
11         $r \leftarrow norm(ip)$;
12         circles $\leftarrow [x_{R^j_{lm}}, y_{R^j_{lm}}, r]$;
13     else
14         $[a^j, b^j] \leftarrow \Phi(R^j_{lm})$, $c^j \leftarrow \eta((R^i_{z_t}, R^j_{lm})$;
15         $lines \leftarrow [a^j, b^j, c^j]$;
16     end
17 end
18 for all plane observations $[R \times R]^i_{z_t}$ and LMs $[R \times R]^j_{lm}$ in C do
19     if $[R \times R]^j_{lm}$ not parallel to floor then
20         $R_{int} \leftarrow I([R \times R]^i_{z_t}, floor)$;
21         $[a^j, b^j] \leftarrow \Phi(R_{int})$, $c^j \leftarrow \eta(R_{int}, R'_{int})$;
22         $lines \leftarrow [a^j, b^j, c^j]$;
23     end
24 end
25 $\mathcal{H}_{\mu(x,y)}, \mathcal{H}_\theta \leftarrow \Upsilon(circles)$, $\mathcal{H}_{\mu(x,y)}, \mathcal{H}_\theta \leftarrow \Lambda(lines)$;
   /* Each $\mathcal{H}_{\mu(x,y)} \leftarrow [x, y, \sigma^2_{(x,y)}]$, $\mathcal{H}_\theta \leftarrow [\theta, \sigma^2_\theta]$ */
26 $\mu_t, \Sigma_t \leftarrow$ COMBINE_HYPOTHESES($\mathcal{H}_{\mu(x,y)}, \mathcal{H}_\theta$)
```

Lines intersecting the floor planes (for example, vertical lines) give point constraints, and coupled with already known point landmarks signify that the robot is at a certain distance from these points (lines 6, 12), i.e., the robot has to be on the circle(s) with these point(s) as center(s). For a single circle constraint, the closest point on this circle to the current robot location is an additional hypothesis. If multiple circles are present, intersection of pairs of these circles give additional hypotheses. An orientation estimate θ hypothesis is simply the difference between existing landmark orientation and newly observed landmark ori-

Algorithm 11: LANDMARK CORRESPONDENCES

```
input  : $Z_t$: All observed landmarks in robot frame. $LM$: All landmarks in beliefs
output: $C$: correspondences between $Z_t$ and $LM$.
1 for all observations ${}^{R}z_t^i$ in $Z_t$ do
2     for all landmarks $lm^j$ in $LM$ do
3         ${}^{W}z_t^i = transform({}^{W}T_R, {}^{R}z_t^i)$;
4         if $\Psi({}^{R}z_t^i, lm^j) < \tau$ then
5             $C \leftarrow \{i, j\}$;
6         end
7     end
8 end
9 return $C$
```

entation, added to the current orientation hypotheses. Function I is used to determine the intersection of a line or plane with the floor, function Φ calculates line parameters (a, b), and η determines the constraint line's distance (parameter c in the line equation) from the origin and the side of the line the robot is located on. Multiple hypotheses ($\mathcal{H}_\mu$ for location and $\mathcal{H}_\theta$ for orientation) might be generated depending on how many constraint circles and constraint lines are discovered as explained above, and determined by functions Υ and Λ, respectively (line 25). These hypotheses are combined based on the weights K, that are determined using the standard applied optimal estimate [57] (see Algorithm 12), and are based on the the noise variances associated with the hypotheses (lines 6 – 7).

5.5 Multirobot Knowledge Sharing

Multirobot knowledge sharing helps two robots share their acquired knowledge, in the form of WP beliefs. We demonstrate this capability in a simple fashion by sharing one robot's knowledge base with another robot which can then avoid mapping the environment. We demonstrate the multirobot knowledge sharing with two scenarios, both of which are likely to occur in day-to-day operation of multiple robots. First scenario deals with world frame image matching, whereas second scenario deals with WP data matching.

5.5.1 Scenario 1: Exchange of Knowledge by World Frame Correspondence.

In this scenario we assume that a robot B has not been in robot A's environment before, but has the world frame (RGB) image of A's environment (we also assume some form

Algorithm 12: COMBINE HYPOTHESES

```
input : $\mathcal{H}_{\mu(x,y)}, \mathcal{H}_\theta$
output: $\mu_t, \Sigma_t$
1
2 for all hypotheses pairs $\{h^i_\mu, h^j_\mu\} \in \mathcal{H}_\mu$, and $\{h^p_\theta, h^q_\theta\} \in \mathcal{H}_\theta$, do
      $$V_\mu \leftarrow \begin{bmatrix} \sigma^2_{\mu,1}/\sigma^2_{\mu,1} & \sigma^2_{\mu,1}/\sigma^2_{\mu,2} & \dots & \sigma^2_{\mu,1}/\sigma^2_{\mu,n} \\ \vdots & \vdots & \ddots & \vdots \\ \sigma^2_{\mu,n}/\sigma^2_{\mu,1} & \sigma^2_{\mu,n}/\sigma^2_{\mu,2} & \dots & \sigma^2_{\mu,n}/\sigma^2_{\mu,n} \end{bmatrix}$$
      $$V_\theta \leftarrow \begin{bmatrix} \sigma^2_{\theta,1}/\sigma^2_{\theta,1} & \sigma^2_{\theta,1}/\sigma^2_{\theta,2} & \dots & \sigma^2_{\theta,1}/\sigma^2_{\theta,m} \\ \vdots & \vdots & \ddots & \vdots \\ \sigma^2_{\theta,m}/\sigma^2_{\theta,1} & \sigma^2_{\theta,}/\sigma^2_{\theta,2} & \dots & \sigma^2_{\theta,m}/\sigma^2_{\theta,m} \end{bmatrix}$$
3 end
4
      $$K_{x,y,(1\dots N)} = [1/\sum_{j=1}^{N} V_\mu(1,j), 1/\sum_{j=1}^{N} V_\mu(2,j), \dots, 1/\sum_{j=1}^{N} V_\mu(N,j)]^T$$
      $$K_{\theta,(1\dots M)} = [1/\sum_{j=1}^{M} V_\theta(1,j), 1/\sum_{j=1}^{M} V_\theta(2,j), \dots, 1/\sum_{j=1}^{M} V_\theta(M,j)]^T$$
  $\hat{\mu}_{t(x,y)} \leftarrow \sum_{i=1}^{N} K_i \mathcal{H}_{\mu(x,y),i}$ ,   $\hat{\theta}_t \leftarrow \sum_{i=1}^{M} K_i \mathcal{H}_{\theta,i}$
5 $\sigma^2_{x,y} = \sqrt{\sum_{i=1}^{N} K_{x,y,i}\sigma_{x,y,i}}$
6 $\sigma^2_\theta = \sqrt{\sum_{i=1}^{M} K_{\theta,i}\sigma_{\theta,i}}$
7 $\mu_t \leftarrow [\hat{\mu}_t(x,y), \hat{\theta}_t]^T$
  $$\Sigma_t \leftarrow \begin{bmatrix} \sigma^2_{x,y} & 0 & 0 \\ 0 & \sigma^2_{x,y} & 0 \\ 0 & 0 & \sigma^2_\theta \end{bmatrix}$$
  return $\mu_t, \Sigma_t$
```

of communication interface between the robots has been established for image exchange such robot B can acquire this image from robot A at the start). When B finds this world frame it simply needs to localize itself in A's frame and acquire the WP knowledge base from A. The key to achieving this functionality is world frame matching. Figure 5.11(a) shows the world frame image for robot A which starts at this particular section on the floor, and builds WPs with respect to this frame, whereas Figure 5.11(b) shows the image of the world frame during the robot B's run, encountered midway during execution.

Similarly Figure 5.12(a) shows the world frame image for robot A, whereas Figure

5.12(b) shows the image of the world frame during the robot B's run, encountered after loop closure.

B does not know beforehand, which particular corner in the current environment corresponds to A's world frame, but it can determine this using RGB data of the candidate corners. We do not consider the original RGB image in our calculations since the size of the images are different and they are shifted along the baseline and the RGB pixels are not associated directly with x, y, z, but use the RGB image registered with the depth camera (the dark pixels near the metal grate in Figure 5.11(a) and 5.11(b) are artifacts of inverse mapping occurring during this registration process).

A simple 2D correlation using Equation 5.6 on the two images in Figure 5.11 yields a score of 0.8506.

$$c = \frac{\sum_i \sum_j (A_{ij} - \bar{A})(B_{ij} - \bar{B})}{\sqrt{(\sum_i \sum_j (A_{ij} - \bar{A})^2)(\sum_i \sum_j (B_{ij} - \bar{B})^2)}} \tag{5.6}$$

where $\bar{A}$ is mean(A), and $\bar{B}$ is mean(B). The 2D correlation using the two images in Figure 5.12 yields a score of 0.9043.

Figures 5.13 - 5.15 illustrate the matching of world frame image of robot A to corners encountered during robot B's run that do not match the original image (invalid corners), and their respective 2D correlation scores.

The basic idea and assumption behind this approach is that corners will be sufficiently different in terms of their color and other features that will help identify similar corners (two corners in a building that are not the same but look exactly alike would have no solution with the 2D approach; this ambiguity can be solved for by using 3D structure of the environment (see scenario 2 discussed in subsection 5.5.2).

Once a high correlation location is found, robot B acquires range data and builds a coordinate frame and localizes itself in this frame, and it acquires the belief set from robot A and it can proceed with its other plans since it is now localized in this frame. Thus, it can avoid repeating the WP building process if it has access to the RGB image of the world frame of another robot.

5.5.2 Scenario 2: Exchange of Knowledge by Environment Shape and WP Matching.

In this scenario, two robots A and B are exploring the same environment, although both have started their run from different world frames in the same environment (this scenario is also useful to test for world frame similarity, if ambiguity exists, as discussed in scenario 1. After their respective runs are complete, they can exchange their WP knowledge - for example, to discover flaws in their individual beliefs or to merge the two sets. For example, consider Figure 5.16 which shows the floor map, obtained from the floor mapping plan, of the atrium environment, and Figure 5.17 shows the same floor map, demarcated with lines for convenience.

The black points are all the points on the floor plane. The two (unexplored) corridors are indicated at the north side of Figure 5.17. The left-most (green dotted lined) pillar is used as the initial world frame of robot A and is shown by the red (X-axis) and blue (Y-axis) arrows. Robot A moves in a clockwise direction starting from this position, and covers the entire area before closing the loop at the starting position.

Figure 5.18 shows a bird's-eye view of the atrium environment mapped, with various planes shown in different colors.

On the other hand, robot B starts at a different world frame location as shown in Figure 5.19.

The world frame location is on the right (East) side of the floor and marked with X and Y axes. After both robots have finished mapping the floor we use range scan algorithm to determine the shape of the mapped floor using the upright plane WPs (walls, pillars, doors, etc.). The algorithm proceeds as follows: both robots find the mean points P_i (P_A and P_B, for the two robots) of their respective floor maps which are in different world frames. If the environment is the same in both cases, these points should be close to each other spatially in the real world. We then start scanning, with the mean point P_i as the center, from $0°$ to $359°$ with respect to each frame's X-axis, and step in the direction of this angle in small increments (similar to how a radar would scan the world around it). During these small increments, if a WP plane is encountered within 0.05 m, the increments for the current angle are added to get the range to this WP plane from P_i. We then start the same process from the next angle. This is similar to a polar coordinate representation. Since lines and points are less likely to be encountered owing to discrete increments of angles and distance increments, we only consider planes for this process. Two robots that have scanned the same environment get similar range scans for their WP planes, albeit shifted (circularly) by a certain angle depending on the orientation between their respective world frames.

Figures 5.20 shows the floor map for robot A (in blue) and WP planes shown in various colors (note that the colors are repeated, hence, there is not necessarily a correspondence between planes of the same color). The mean point P_A is shown as a big red dot at the center of the floor. The concentric green "x" circles indicate discrete increments from P_A till a WP plane is encountered, for all angles from $[0^\circ, 360^\circ]$. Figure 5.21 shows the range scan for robot B. Notice that the mean points P_A and P_B are not exactly at the same location, but are close to each other. Also note that the axes are switched for the two scans.

These ranges for robots A and B are shown in Figures 5.22 and 5.23, respectively. Notice that they have the same structure but are circularly shifted by 270° (or 90° depending on the direction of rotation). By keeping ranges for one robot constant, the ranges for other robot are shifted one angle at a time (anti-clockwise rotation $\implies$ 270° shift and clockwise rotation $\implies$ 90° shift.), and the score at any particular shift value s is calculated as $Score_s = \sum_{i=0^\circ}^{359^\circ} abs(d_{i,1} - d_{i+s,2})$, where $d_{i,1}$ is the range value for first robot for angle i and $d_{i,2}$ is the range value for second robot for angle i, and $s \in [0^\circ, 359^\circ]$. The maximum score corresponded to a shift of 270° in our experiments for the range values. Figure 5.24 (range scan of robot B shifted by $s = 270^\circ$) matches Figure 5.22 (range scan of robot A) very well.

The WP planes from the two datasets were also matched with one another, using distance function d, for these shifts and the plane parameters were checked for the two sets and the maximum number of matches - corresponding to minimum distance, implying maximal plane alignment - were found to be at a shift of 267°. We used the distance function d, shown in Equation 5.7, to measure how close the two sets of planes align with one another when one set is circularly shifted by a certain shift value, where P_1 and P_2 are the two planes from two sets, $\bar{n}_1$ and $\bar{n}_2$ are their respective plane normals, d_1 and d_2 are their respective distances from their respective origins, and these two measures (dot product $\bar{n}_1 \cdot \bar{n}_2$ and absolute difference of their distance from their respective origin $|d_1 - d_2|$) are weighted by a. In our experiments both are weighted equally, thus $a = 0.5$.

$$d(P_1, P_2) = \frac{a\frac{(1-\bar{n}_1\cdot\bar{n}_2)}{2} + (1-a)(1-e^{-|d_1-d_2|})}{2} \tag{5.7}$$

This shift is applied to the sequence of WP occurrence (in other words, indexes of WP planes) in robot B's dataset, to align the indexes with that of WP dataset of robot A. Once the correct correspondence is known, a rotation corresponding to the shift angle (267° in our case) is applied to one of the world frame to align the axes of the two world frames, and since the correspondence between WP planes indexes is now known between the two datasets, the translation is found by finding the translation between any pair of matching WP planes in the two aligned datasets, to give the complete transform between the two world frames. In the example discussed above, this transform was determined to be

$$^{B}T_{A} = \begin{bmatrix} -0.0523 & 0.9986 & 0 & -0.0253 \\ -0.9986 & -0.0523 & 0 & 10.5500 \\ 0 & 0 & 1.0000 & 0 \\ 0 & 0 & 0 & 1.0000 \end{bmatrix}$$

Note that the respective world frames for A and B are translated with respect to each other only in positive X direction (in A's frame), and minimally in A-frame's Y-axis, and they are rotated with respect to each other by 270°. Thus the transform given above matches the data. Once this transform is determined, it can be applied to the first dataset (or the inverse of this transform applied to the second dataset) to align the WP datasets with one another. The alignment performed using this transform is shown in Figure 5.25 where the planes from dataset A are shown in blue and those from dataset B are shown in red. Note that these planes are represented in the world frame for robot B which is why the X-axis is aligned breadth-wise instead of length-wise, like in the case of robot A's world frame.

Table 5.1. Location error (in meters) comparison between systems using RANSAC.

System→	**WPCA** (*lab*)	**WPCA** (*Atrium*)	Pioneer	MSR	FSPF	Endres'12
Mean	0.0989	0.2065	0.22	0.23	0.7	0.097
Median	0.0948	0.1939	-	-	1.08	-
Min	0.0511	0.0454	0.12	0.03	0.17	0.034
Max	0.2574	0.4180	0.32	0.43	3.47	0.16

Table 5.2. Angular (orientation) error (in degrees) comparison between systems using RANSAC.

System→	**WPCA** (*lab*)	**WPCA** (*Atrium*)	Pioneer	MSR	FSPF	Endres'12
Mean	10.4221	5.9360	-	0.5	-	3.39
Median	8.9267	5.2658	-	-	-	-
Min	2.3491	0.2658	-	-	-	1.84
Max	29.1521	13.7671	-	2.5	-	4.94

Table 5.3. Location error (in meters) comparison between systems with planar segmentation using pixel-based approach.

System→	**WPCA** (*lab*)	**WPCA** (*Atrium*)	Pioneer	MSR	FSPF	Endres'12
Mean	0.0733	0.3624	0.22	0.23	0.7	0.097
Median	0.1029	0.2832	-	-	1.08	-
Min	0.0363	0.0593	0.12	0.03	0.17	0.034
Max	0.1029	0.6984	0.32	0.43	3.47	0.16

Table 5.4. Angular (orientation) error (in degrees) comparison between systems with planar segmentation using pixel-based approach.

System→	**WPCA** (*lab*)	**WPCA** (*Atrium*)	Pioneer	MSR	FSPF	Endres'12
Mean	3.8647	3.6306	-	0.5	-	3.39
Median	3.5648	3.6279	-	-	-	-
Min	0.4344	0.3719	-	-	-	1.84
Max	12.4380	7.6306	-	2.5	-	4.94

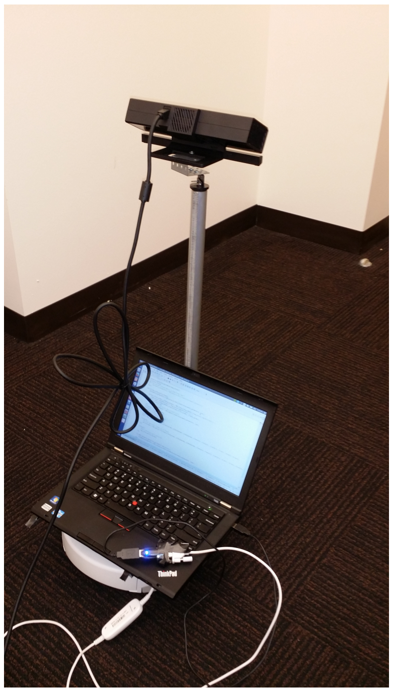


Figure 5.1. Symmbot.

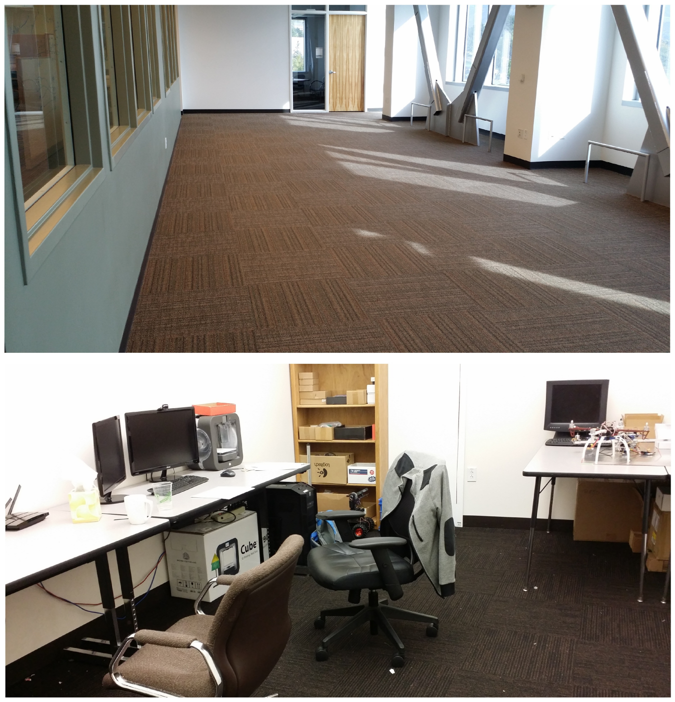


Figure 5.2. Example of (top) a hallway, and (bottom) an office in the Warnock Engineering Building.

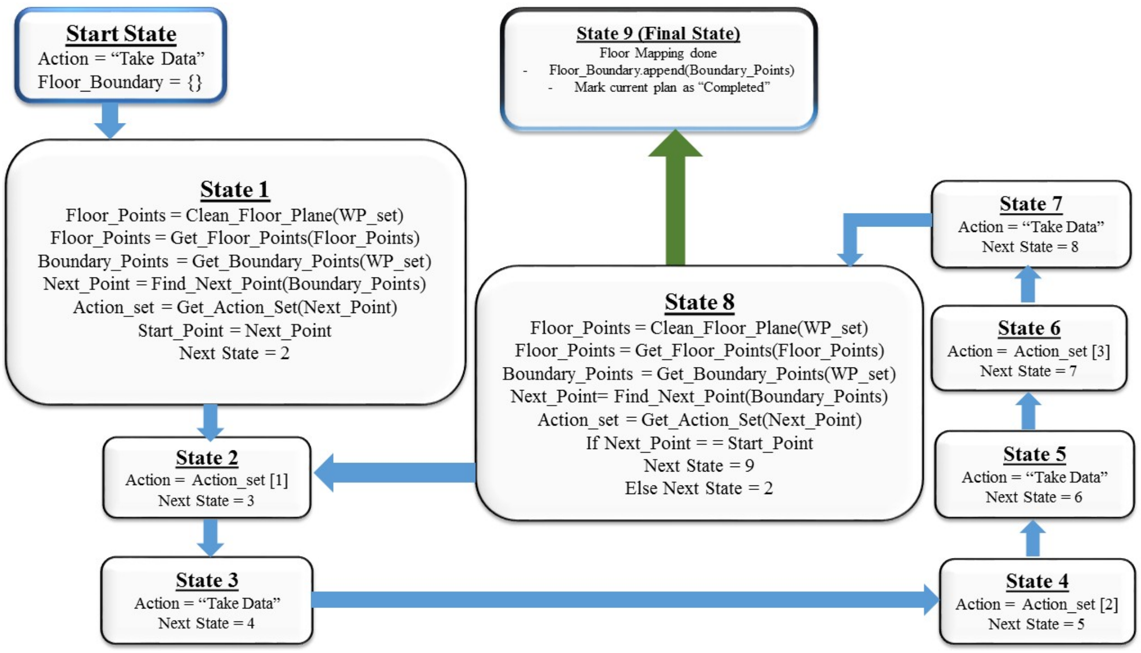

Start State
Action = “Take Data”
Floor_Boundary = {}
State 9 (Final State)
Floor Mapping done
- Floor_Boundary.append(Boundary_Points)
- Mark current plan as “Completed”
State 1
Floor_Points = Clean_Floor_Plane(WP_set)
Floor_Points = Get_Floor_Points(Floor_Points)
Boundary_Points = Get_Boundary_Points(WP_set)
Next_Point = Find_Next_Point(Boundary_Points)
Action_set = Get_Action_Set(Next_Point)
Start_Point = Next_Point
Next State = 2
State 7
Action = “Take Data”
Next State = 8
State 8
Floor_Points = Clean_Floor_Plane(WP_set)
Floor_Points = Get_Floor_Points(Floor_Points)
Boundary_Points = Get_Boundary_Points(WP_set)
Next_Point= Find_Next_Point(Boundary_Points)
Action_set = Get_Action_Set(Next_Point)
If Next_Point = = Start_Point
Next State = 9
Else Next State = 2
State 6
Action = Action_set [3]
Next State = 7
State 2
Action = Action_set [1]
Next State = 3
State 5
Action = “Take Data”
Next State = 6
State 3
Action = “Take Data”
Next State = 4
State 4
Action = Action_set [2]
Next State = 5

Figure 5.3. Finite state machine of the floor mapping plan.

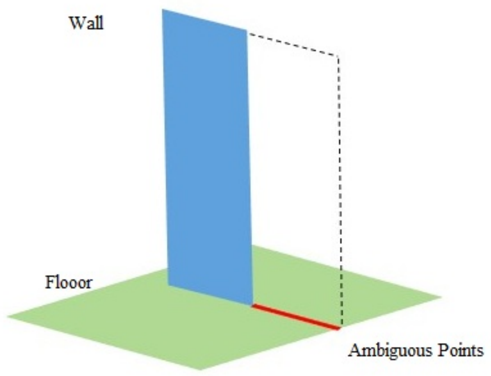
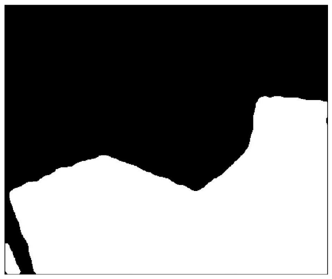


Figure 5.4. Planes can be identified with any points that solve the plane equation.

(a) Registered RGB image.


(b) Floor segmented from the entire x,y,z point cloud based on distance function.

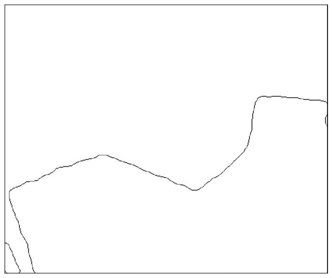
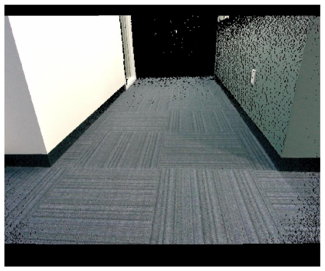
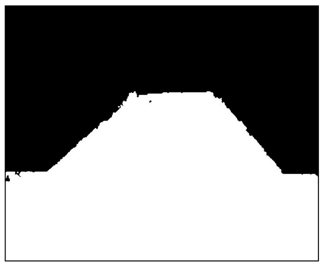

(c) The boundary of the floor based on plane intersections with the floor, with gaps filled with BW morph functions.

Figure 5.5. Floor segmentation view 1.


(a) Registered RGB image.


(b) Floor segmented from the entire x,y,z point cloud based on distance function.

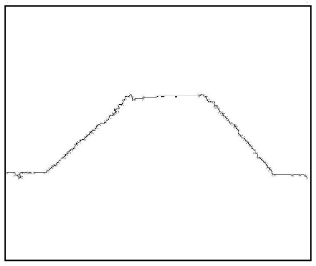

(c) The boundary of the floor based on plane intersections with the floor, with gaps filled with BW morph functions.

Figure 5.6. Floor segmentation view 2.

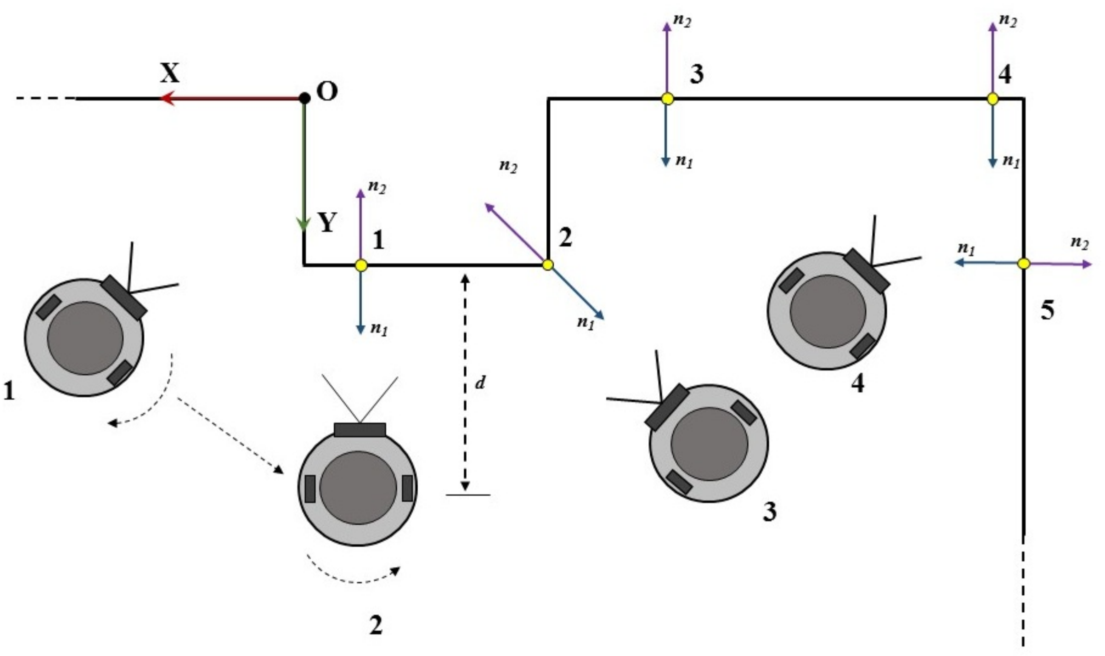


Figure 5.7. Depiction of the wall following algorithm in top view.

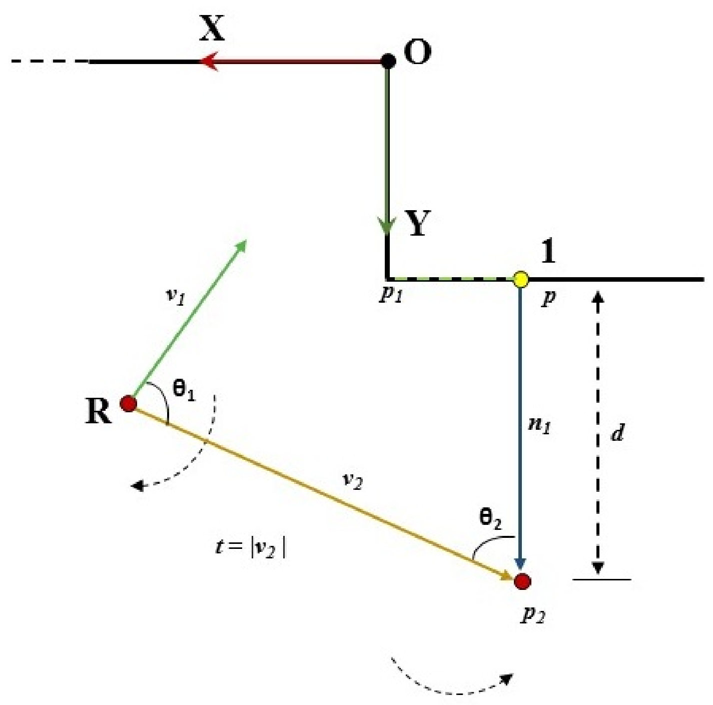


Figure 5.8. Action set generation.

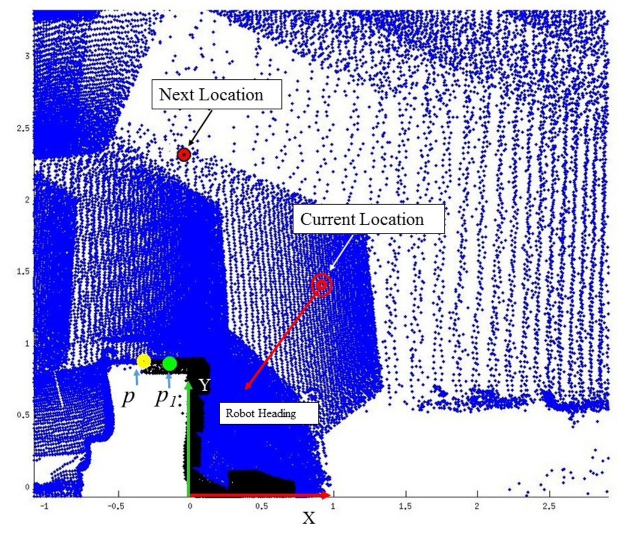
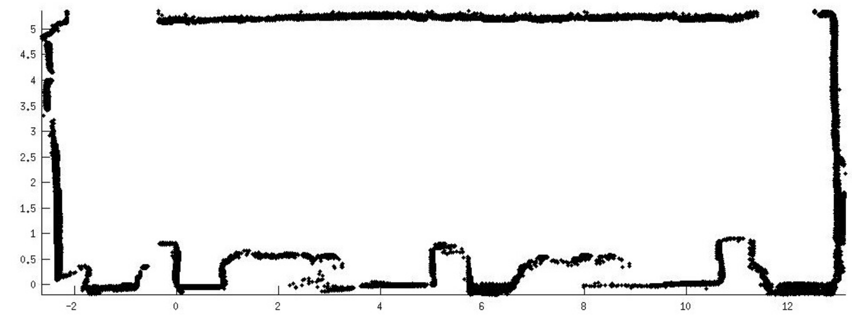
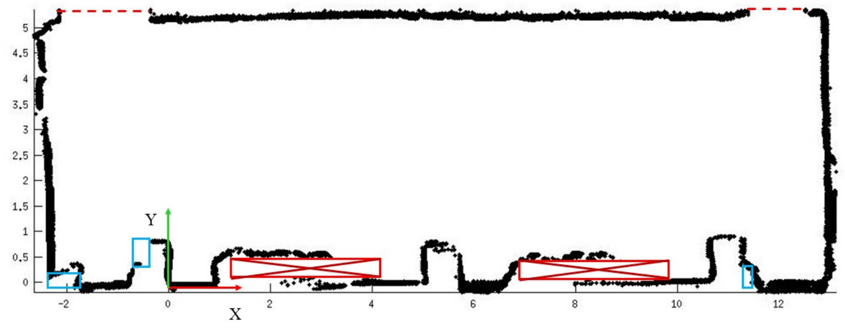


Figure 5.9. A zoomed in image of the action set generation process on real data. Floor points are blue dots. The X and Y axes are showing as red and green arrows respectively. Current robot location and robot heading are shown as red dot and red arrow. Points p and p_1 correspond to those shown in Figure 5.8, whereas *Next Location* corresponds to point p_2 in Figure 5.8.

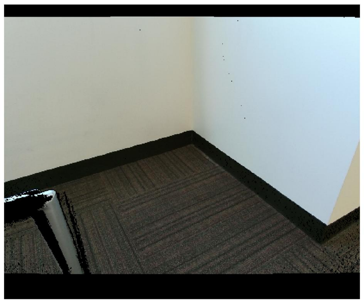


(a) Floor boundary generated using wall following technique.



(b) Analysis of floor boundary generated using wall following technique.

Figure 5.10. Floor boundary generated using the wall following technique.


(a) World frame corner image for first world frame detection.

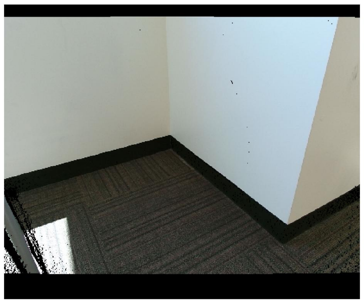
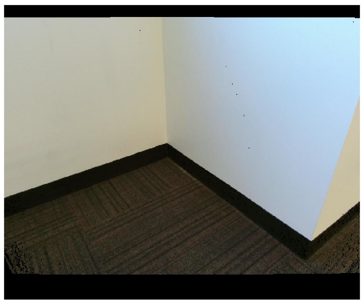

(b) World frame corner image for second world frame detection, in another run (loop closure).

Figure 5.11. Comparison between original world frame and candidate world frame in a different sequence of actions. 2D correlation score = 0.8506.


(a) World frame corner image for first world frame detection.

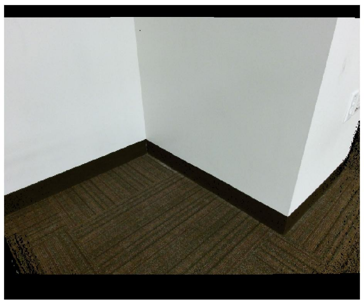
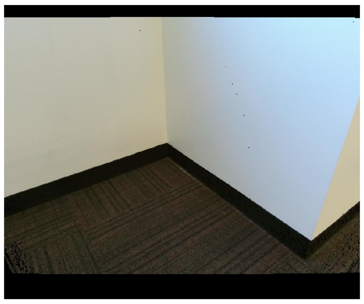

(b) World frame corner image for second world frame detection in the same run (loop closure).

Figure 5.12. Comparison between original world frame and candidate world frame. 2D correlation score = 0.9043.


(a) World frame corner image for first world frame detection.

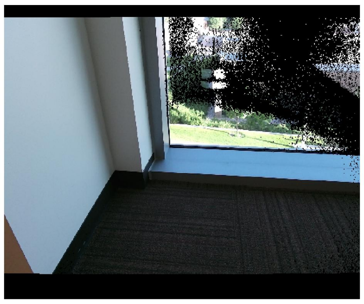
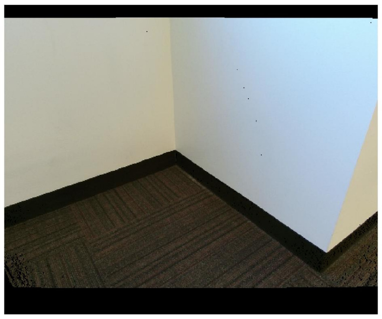
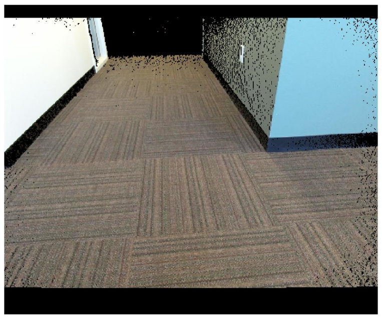

(b) Candidate World frame corner image for second world frame detection.

Figure 5.13. Candidate world frame 2. 2D correlation score = 0.3627.

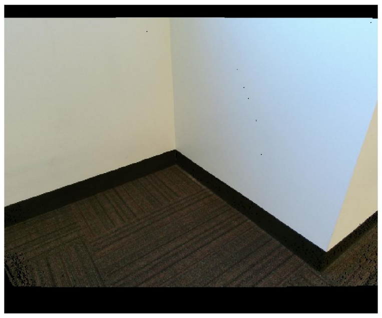


(a) World frame corner image for first world frame detection.

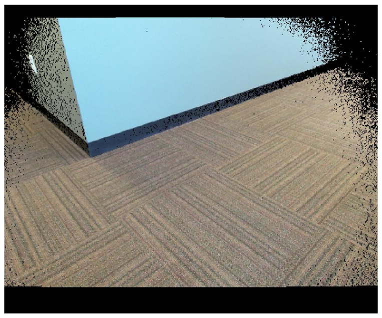


(b) Candidate World frame corner image for second world frame detection

Figure 5.14. Candidate world frame 3. 2D correlation score = 0.4599.



(a) World frame corner image for first world frame detection.



(b) Candidate World frame corner image for second world frame detection

Figure 5.15. Candidate world frame 4. 2D correlation score = 0.4007.

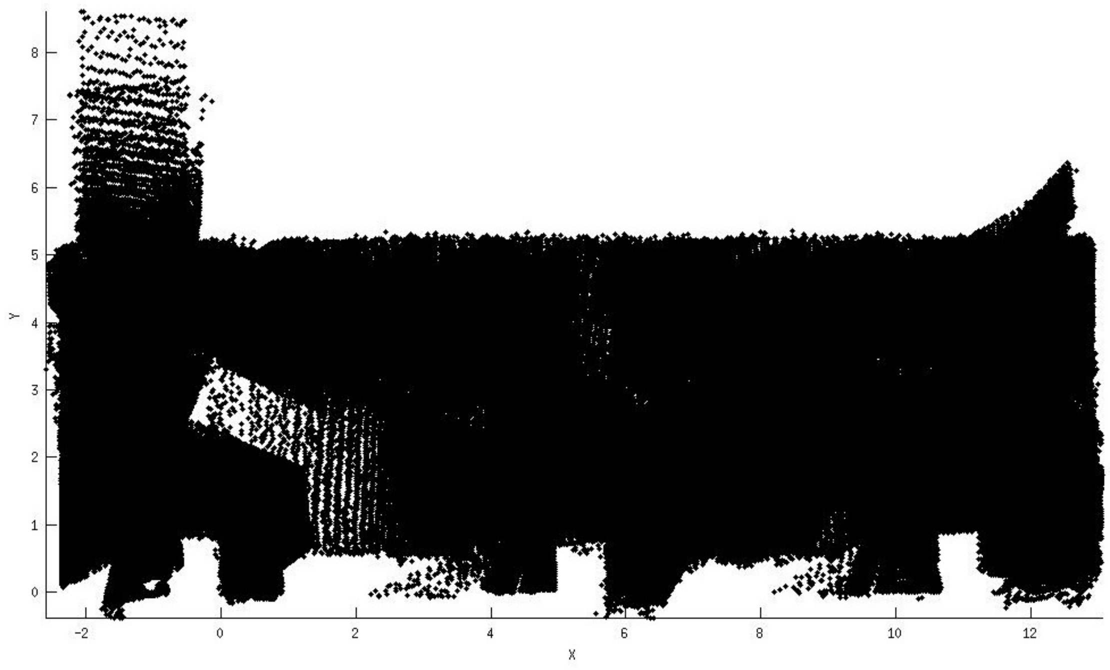


Figure 5.16. Points on the floor shown in the X-Y view.

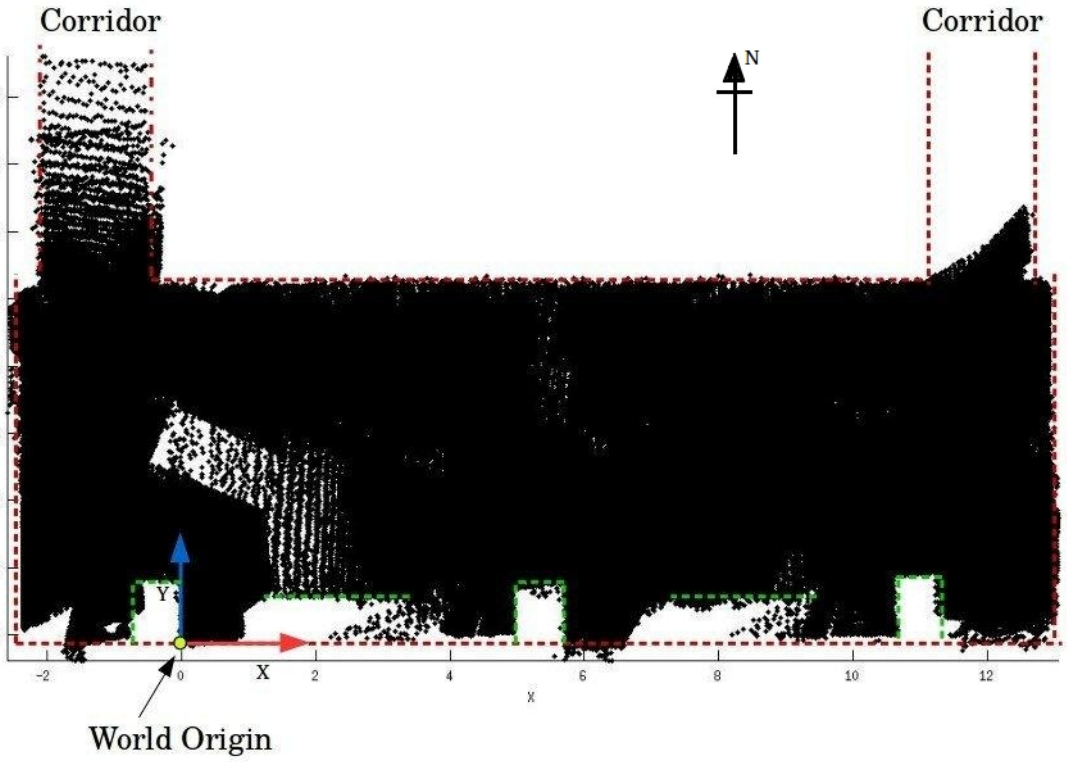


Figure 5.17. Points on the floor shown in the X-Y view with lines added for demarcation. The red dashed lines shows the rectangular bounding rectangles of the floor that matches the data closely. The green dotted lines indicate the occluding pillars and other structures obstructing view of the south-side wall. The green dot indicates the world frame origin and the red and blue arrows indicate X and Y axes, respectively.

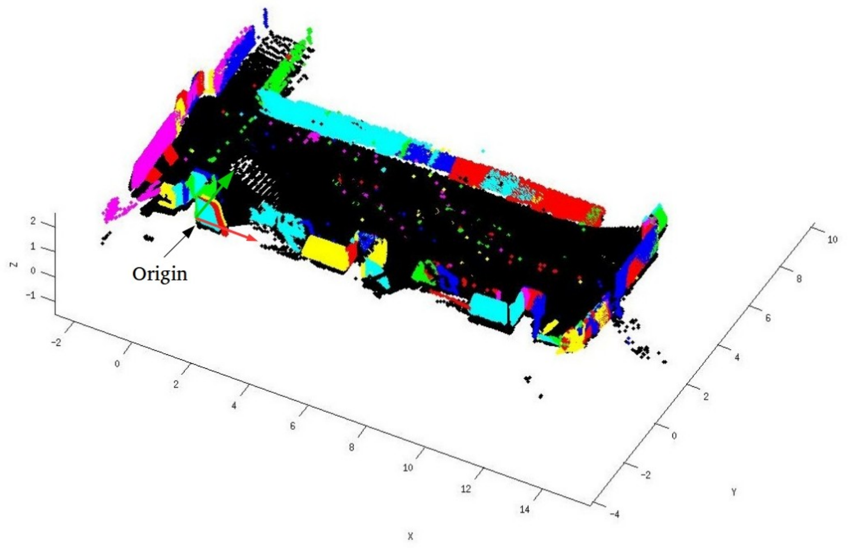


Figure 5.18. All the WP planes shown in different colors in a bird's-eye view (note that different colors does not necessarily mean different planes and same color does not necessarily mean same plane. The limited options of colors in the plotting function means colors have to be shared by various planes).

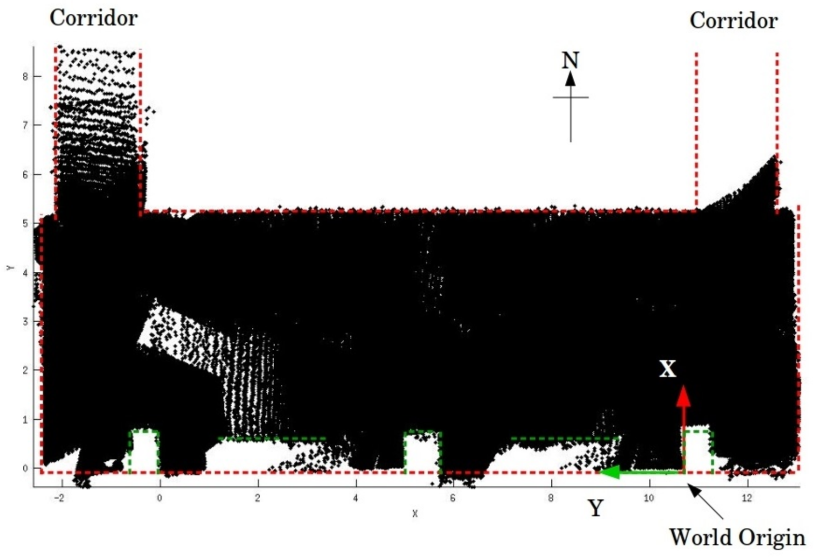


Figure 5.19. Points on the floor shown in the X-Y view with lines added for demarcation. The red dashed lines shows the rectangular bounding rectangles of the floor that matches the data closely. The green dotted lines indicate the occluding pillars and other structures obstructing view of the south-side wall. The green dot indicates the world frame origin and the red and blue arrows indicate X and Y axes respectively. Note that robot B starts at a different world origin on the right (east) side of the floor.

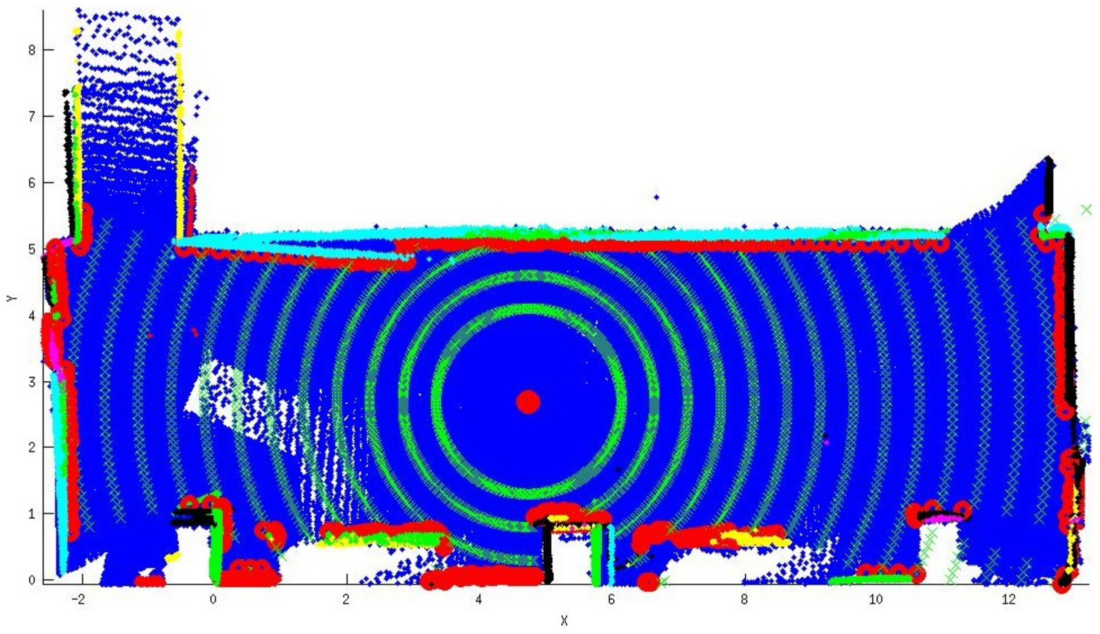


Figure 5.20. Range scan of map for robot A.

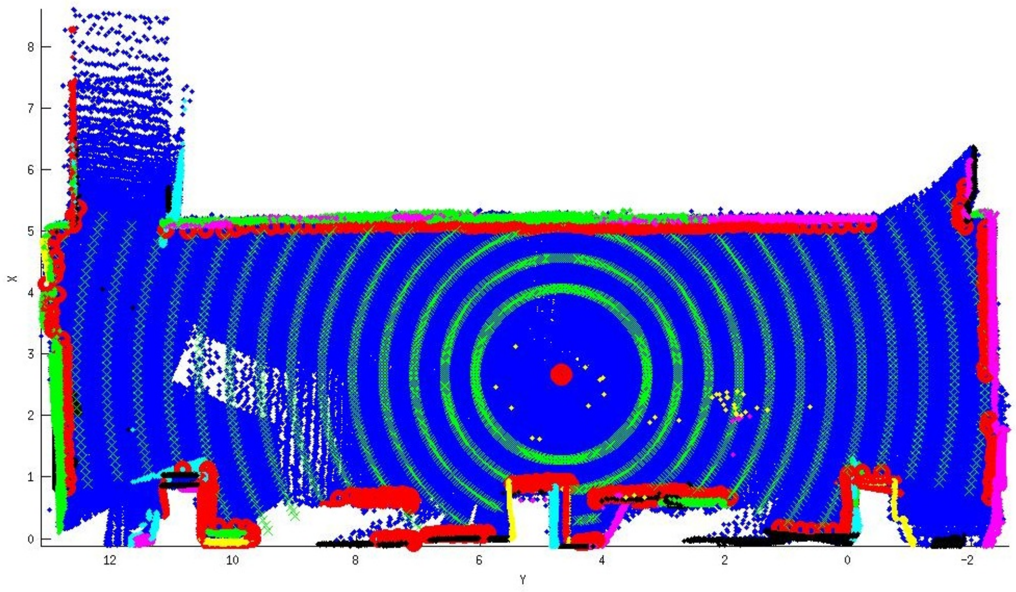


Figure 5.21. Range scan of map for robot B.

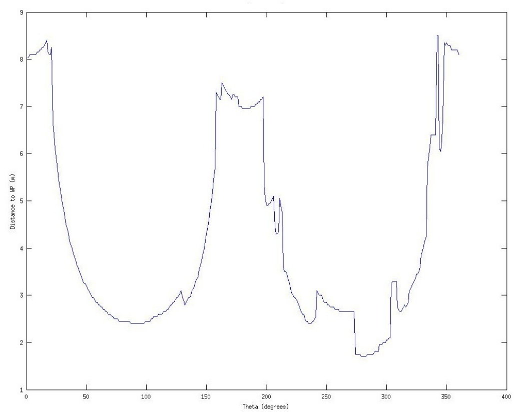


Figure 5.22. Range scan distances of map for robot A.

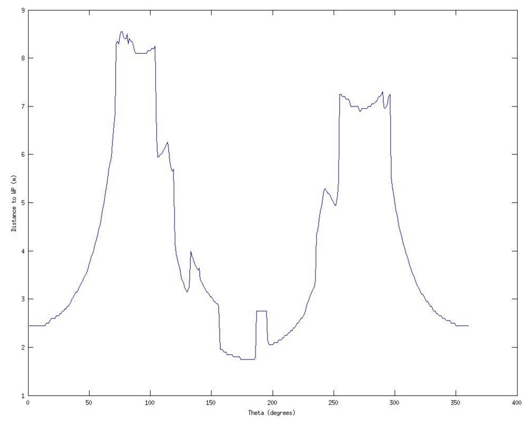


Figure 5.23. Range scan distances of map for robot B.

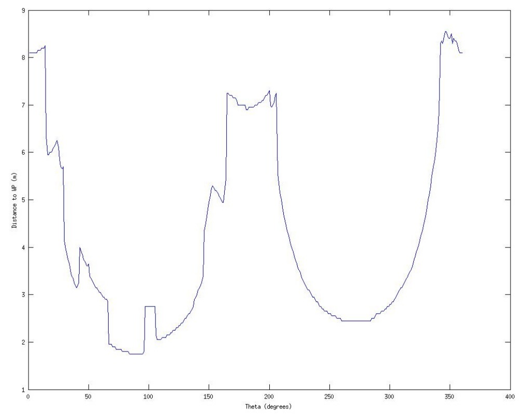


Figure 5.24. Range scan distances of map for robot B, shifted by 270°.

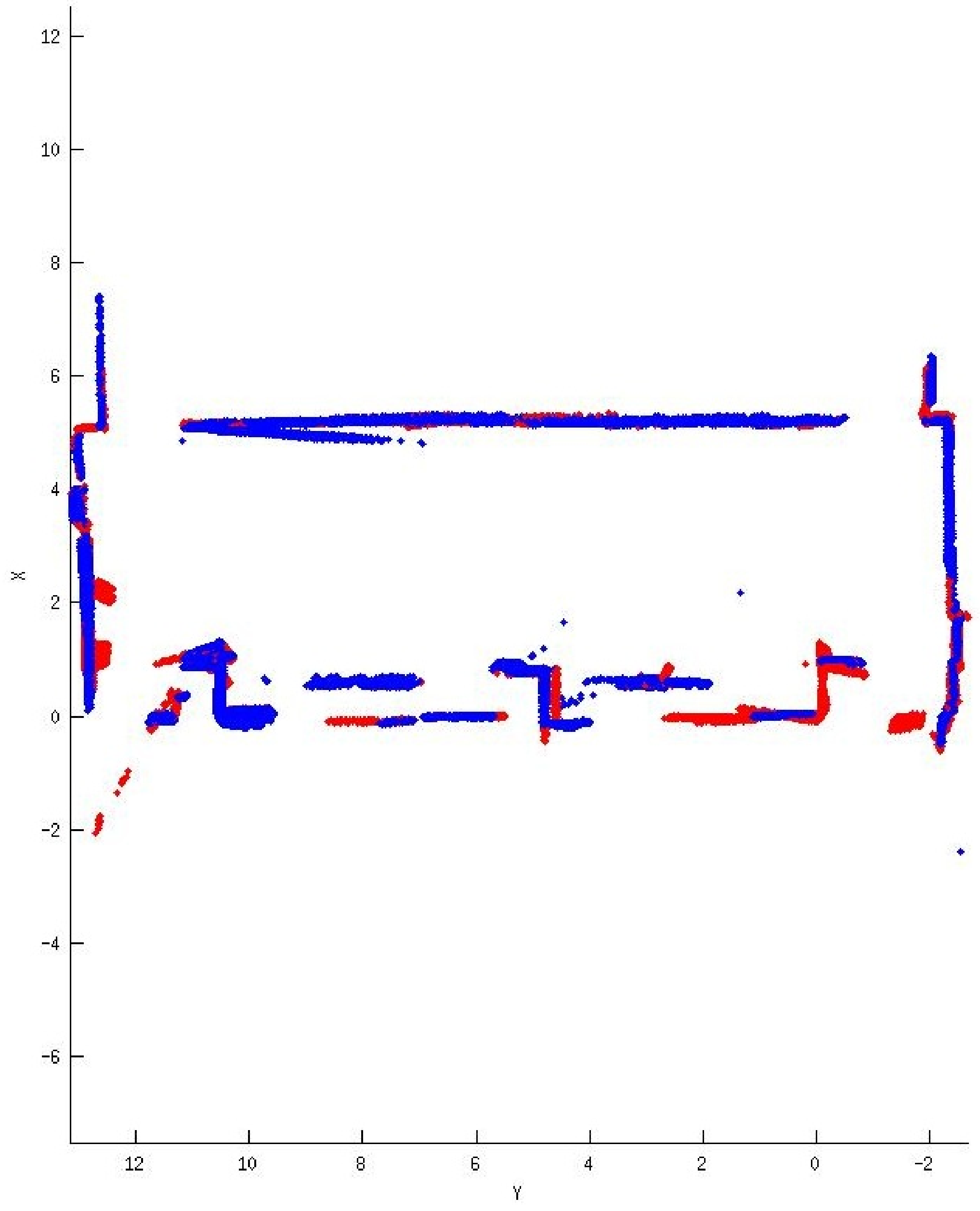


Figure 5.25. WP planes from two different environment overlayed.

CHAPTER 6

CONCLUSIONS AND FUTURE WORK

We conclude this dissertation with a summary of contributions of our work and potential for future work and enhancements in the WPCA. Our contributions include:

1. The first implementation of Wreath Product representation of Leyton's ideas, to the best of our knowledge.
2. The Enhanced Non-Deterministic Agent System (ENDAS) for 2D character classification uses WP and symmetry concepts, as well as binds perception and actuation to create an effective and efficient representation of 2D data.
3. A BDI architecture implementation which manages WPCA in operation.
4. A demonstration of the performance of WPCA in the context of mapping and localization using WPs as landmarks.
5. Incorporation of actuation as a generative mechanism in the WPCA, and its application to 2D character classification.
6. Multirobot communication and knowledge sharing using WP representation.

We have demonstrated a novel practical implementation of a cognitive architecture using the wreath product representation for indoor environmental data and a Belief Desire Intention (BDI) framework that gives structure for realizing these representations. To the best of our knowledge, this is the only existing practical implementation of Leyton's wreath product cognitive model. This architecture is also another example of a working BDI framework module that is modeled on the human practical reasoning process. The amalgamation of the wreath product representation and BDI framework brings together two powerful cognitive architectures concepts: effective representations that tie together perception and actuation, and a framework based on practical reasoning to use these representations to achieve high-level goals in the form of plans. We have addressed not only 3D depth data for the purposes of robot operation in a real environment, but have also found interesting results working with 2D data and applying wreath product principles to character classification in engineering drawings. We have demonstrated that our implementation works very well in the context of robot localization, one of the most essential functionalities of a mobile robot, and a precursor to other intelligent capabilities. The performance of our localization algorithm using wreath products as landmarks compares fairly well to localization algorithms on other systems similar to ours; it performs better than other systems in terms of localization error, but does not perform well in terms of orientation error which can be attributed to trade-off between landmark correspondence discovery, and matching thresholds. Future work in this regard would involve improving orientation accuracy without negatively affecting location accuracy or landmark discovery. Experimentation with sensors using other depth measurement technique (e.g., structured light or LIDAR sensors) is also possible. Our representations are structured in a way that allows for multirobot knowledge sharing in scenarios where the collective knowledge of a set of robots will be larger than their individual knowledge bases, vastly reducing the time and effort required overall. We also provided a context-free grammar and pushdown automaton for the representation of wreath products. Finally, we have shown how WPs can be interpreted as motion plans.

6.1 Future Work

The WPCA provides a base upon which to build additional cognitive capabilities for an autonomous agent. In that respect, some avenues to explore are as follows:

6.1.1 Innate Knowledge

In living beings, some innate knowledge comes from evolutionary experience. For example, the fight-or-flight response, the feeling of stress or hunger, etc., are some of the built-in biochemical mechanisms, acquired through evolution, that drive the responses of living beings. Some of this innate knowledge gives rise to the intrinsic motivation, while other motivations are acquired by existence in the environment. In future, analyses will be done to identify additional innate beliefs required for improving the robot's cognitive capabilities, or to identify which beliefs should be acquired initially to improve effectiveness.

6.1.2 Learning

6.1.2.1 Reinforcement learning

Although not in the scope of this work, learning mechanisms can be added to the plan library, in order to promote some actions using reward functions, while discouraging others that can impede progress. These learning mechanisms can be used not only in the high-level WP plans that generate actions, but also in the low-level robot control functions, given the robot motion model, to reduce error.

6.1.2.2 Own structure

Discovery of relations between perception and actuation systems is an important process for an autonomous agent; in most practical applications this information is built into the system by the designer. We have explored some aspects of this in the form of *sensorimotor recontruction*, and *symmetry bundles* in [1, 3, 59]. The discovery of own structure can be explored further by adding more sensors and actuators, and mechanisms to determine the relations formed during operation. This capability can be tested on more complicated robot systems where the robot will be able to identify the relationship between its own perception and actuation systems upon ″waking up″ in the world with minimal knowledge. For example, the iRobot Create has simple *translate* and *rotate* actuation mechanisms. On more complicated Unmanned Ground Vehicle (UGV) platforms, other motions - e.g., driving on the circumference of a circle - can be tested. In addition, pan-and-tilt camera mounts will provide additional capabilities to trace an object visually, akin to our work on *virtual camera*, in Chapter 4.

6.1.2.3 Deep learning

We have recently demonstrated the applicability of recurrent neural networks to form simple concepts of shapes using the perception and actuation signals, where we describe how wreath products can be represented as recurrent neural networks where the actuations are represented one one side of the network and the perceived symmetries are represented on the other side (see [60]). For complex relationships between these, the advancements in deep learning [61] can be exploited, where the relations between actuation, and resulting symmetries is complicated and spans many layers.

6.1.3 Abstraction

6.1.3.1 Generalization

A variety of objects or structures can be generalized. For example, a number of man-made objects in the environment can be generalized as rectangular parallelepipeds, cylinders, etc. A robot with generalization mechanism may look at a hallway and a storage box and will generalize generalize them both as parallelepipeds, or it may look at a cylindrical pillar and a water bottle and generalize them both as cylinders, etc., even though these objects may have protrusions, recesses, or imperfections. This mechanism can be informed by the WP shape representations.

6.1.3.2 CAD architecture mapping

Since WPCA works on indoor 3D data, it is ideally suited for mapping the architecture of the environment for CAD purposes to measure the build quality, either by incorporating CAD models in software, or by analyzing scanned images of CAD drawings (see our work on CAD drawing analysis in [62]).

6.1.3.3 Summarization

Producing summarization sentences (e.g., ”Square has symmetries $(e \wr \mathcal{Z}_2 \wr \mathfrak{R} \wr \mathcal{Z}_4)$ and $(e \wr \mathcal{Z}_4 \wr \mathfrak{R})$”), and/or semantic labels (e.g., ”room,” ”box,” ”pillar,” etc.) can be a useful cognitive capability, where the performance of the system is tested by a human tester, on how accurate these sentences are.

6.1.3.4 Natural objects

Autonomous capabilities need to be tested outdoors as well. For that purpose, abstract representations of non-man-made objects are necessary; e.g., trees, rocks, uneven surfaces. These objects may or may not have some form of symmetry. For example, wreath product can be used to represent crumpled paper, formed by applying symmetry-breaking operations to a sphere. Asymmetrical objects (such as a tree) might possess symmetry in parts. For example, the stem, branches, and leaves, may possess their own symmetry.

6.1.4 Further 3D Applications and Experiments

6.1.4.1 Multiple depth sensors

An additional Kinect (or a single sensor that can be aimed) will acquire data from the floor and lower walls, as well as upper walls and ceiling in the same run, providing a larger picture of the environment. This capability will be essential for the CAD architecture mapping, and other scenarios discussed above, and will be useful for platforms that are smaller than our present iRobot Create to acquire data effectively.

6.1.4.2 As-built vs. as-designed

An extension of the CAD architecture mapping described above, the As-build vs. As-designed method will match CAD design of an object/indoor environment, to the respective structure perceived by the robot, in order to determine the build quality.

6.1.4.3 Outdoor mapping and localization

A completely autonomous robot must be able to operate outdoors as well. New challenges arise in an outdoor environment; e.g., lighting, different surfaces, and other environmental conditions affect the sensing capabilities, as well as motion of the robot, since a UGV has to adjust to varying surface conditions, and an Unmanned Aerial Vehicle (UAV) has to deal with turbulent air, etc. In an indoor environment, most structures around the robot remain relatively uniform (e.g., most surfaces are planar) whereas in outdoor environment, the surfaces can vary depending on the landscape and the operational environment. Accessibility for the robot is another issue in an outdoor environment. These challenges imply more work (error correction, adaptive capability, additional planning) for both perception and actuation systems, and this will be studied in more detail in the future.

APPENDIX

BELIEF STORE DATA

A.1 Innate Beliefs

```
beliefs(1).name = 'Robot Pose';
beliefs(1).id = 1;
beliefs(1).type = POSE;
beliefs(1).info = eye(4,4);
beliefs(1).uncertainty = Inf*eye(3,3);
beliefs(1).children = 0;
beliefs(1).parent = 0;

beliefs(2).name = 'Gravity Vector';
beliefs(2).id = 2;
beliefs(2).type = VECTOR;
beliefs(2).info = [0;0;-1];
beliefs(2).uncertainty = [Inf,Inf,Inf];
beliefs(2).children = 0;
beliefs(2).parent = 0;

beliefs(3).name = 'Find WP';
beliefs(3).id = 3;
beliefs(3).type = ACTION;
beliefs(3).info = 0;    % 0: not active, 1: active
beliefs(3).uncertainty = 0;
beliefs(3).children = 0;
beliefs(3).parent = 0;

beliefs(4).name = 'Extend WP';
beliefs(4).id = 4;
beliefs(4).type = ACTION;
beliefs(4).info = 0;    % 0: not active, 1: active
beliefs(4).uncertainty = 0;
beliefs(4).children = 0;
beliefs(4).parent = 0;

beliefs(5).name = 'Combine WPs';
beliefs(5).id = 5;
beliefs(5).type = ACTION;
beliefs(5).info = 0;    % 0: not active, 1: active
beliefs(5).uncertainty = 0;
beliefs(5).children = 0;
beliefs(5).parent = 0;

beliefs(6).name = 'Camera Pose';
beliefs(6).id = 6;
beliefs(6).type = POSE;
R = CS5320_gen_R([1;0;0],-(27.864+90)*pi/180);
T = eye(4,4);
T(1:3,1:3) = R;
T(2,4) = 0.2032; %Translation from the center of the robot
to Kinect base
T(3,4) = 0.82; %Translation from the location of the Kinect
to its base
beliefs(6).info = T;    % pose of camera wrt robot
beliefs(6).uncertainty = 0.0001*eye(3,3);
beliefs(6).children = 0;
beliefs(6).parent = 0;

beliefs(7).name = 'Terminate';
beliefs(7).id = 7;
beliefs(7).type = ACTION;
beliefs(7).info = 0;    % 0: not active, 1: active
beliefs(7).uncertainty = 0;
beliefs(7).children = 0;
beliefs(7).parent = 0;

beliefs(8).name = 'Ever Localized?';
beliefs(8).id = 8;
beliefs(8).type = FACT;
beliefs(8).info = 0;    % 0: not active, 1: active
beliefs(8).uncertainty = 0;
beliefs(8).children = 0;
beliefs(8).parent = 0;

beliefs(9).name = 'Floor';
beliefs(9).id = 9;
beliefs(9).type = FACT;
beliefs(9).info = 0;    % index of WP for floor
beliefs(9).uncertainty = inf;
beliefs(9).children = 0;
beliefs(9).parent = 0;

beliefs(10).name = 'WPs combined';
beliefs(10).id = 10;
beliefs(10).type = FACT;
beliefs(10).info = 0;    % 0 if not combined, else 1
beliefs(10).uncertainty = 0;
beliefs(10).children = 0;
beliefs(10).parent = 0;

beliefs(11).name = 'World Frame Image';
beliefs(11).id = 11;
beliefs(11).type = IMAGE2;
beliefs(11).info = [rgb_im];    % will be occupied by
registered rgb image
beliefs(11).uncertainty = 0;
beliefs(11).children = 0;
beliefs(11).parent = 0;

beliefs(12).name = 'Plan Done?';
beliefs(12).id = 12;
beliefs(12).type = FACT;
beliefs(12).info = 0;    % 0: not done, 1: done
beliefs(12).uncertainty = 0;
beliefs(12).children = 0;
beliefs(12).parent = 0;
```

REFERENCES

[1] T. C. Henderson, E. Cohen, E. Grant, M. Draelos, N. Deshpande, and A. Joshi, "Symmetry as a Basis for Perceptual Fusion," in *Proc. IEEE Conf. Multisensor Fusion and Integration for Intelligent Systems*, (Hamburg, Germany), pp. 101–107, September 2012.

[2] T. Henderson, A. Joshi, and E. Grant, "From Sensorimotor Data to Concepts: The Role of Symmetry," Tech. Rep. UUCS-12-005, University of Utah, School of Computing, Salt Lake City, UT, USA, October 2012.

[3] T. Henderson, H. Peng, C. Sikorski, N. Deshpande, and E. Grant, "Symmetry: A Basis for Sensorimotor Reconstruction," Tech. Rep. UUCS-11-011, University of Utah, School of Computing, Salt Lake City, UT, May 2011.

[4] A. Joshi, T. C. Henderson, and W. Wang, "Generative Cognitive Representation for Embodied Agents," in *Proc. IEEE Conf. Multisensor Fusion and Integration for Intelligent Systems*, (Beijing, China), pp. 1–7, September 2014.

[5] A. Joshi, T. Henderson, and W. Wang, "Robot Cognition using Bayeisan Symmetry Networks," in *Proc. Int. Conf. Agents and Artificial Intelligence*, (Angers, France), pp. 696–702, March 2014.

[6] M. Leyton, *Symmetry, Causality, Mind*. Cambridge, MA: MIT Press, 1992.

[7] M. Leyton, *A Generative Theory of Shape*. Berlin: Springer, 2001.

[8] M. Asada, K. Hosoda, Y. Kuniyoshi, H. Ishiguro, T. Inui, Y. Yoshikawa, M. Ogino, and C. Yoshida, "Cognitive Developmental Robotics: A Survey," *IEEE Trans. Autonomous Mental Development*, vol. 1, no. 1, pp. 12–34, 2009.

[9] J.-C. Buisson, "A Computer Model of Interactivism and Piagetian Assimilation Applied to Visual Perception," in *Proc. 31st Annu. Symp. Jean Piaget Society*, (Berkeley, CA), June 2001.

[10] G. H. Granlund, "Organization of Architectures for Cognitive Vision Systems," in *Cognitive Vision Systems, Lecture Notes in Computer Science*, pp. 37–55, Berlin: Springer, 2006.

[11] F. Guerin, "Learning Like a Baby: A Survey of Artificial Intelligence Approaches," *The Knowledge Engineering Review*, vol. 26, no. 2, pp. 209–236, 2011.

[12] D. Vernon, G. Metta, and G. Sandini, "A Survey of Artificial Cognitive Systems: Implications for the Autonomous Development of Mental Capabilities in Computational Agents," *IEEE Trans. Evolutionary Computation*, vol. 11, no. 2, pp. 151–180, 2007.

[13] A. Noë, *Action in Perception*. Cambridge, MA: MIT Press, 2004.

[14] R. Bajcsy, "Active Perception," *Proc. IEEE*, vol. 76, pp. 966–1005, March 1988.

[15] J. Aloimonos and A. Badyopadhyay, "Active Vision," in *Proc. IEEE 1st Int. Conf. Computer Vision*, (London, UK), pp. 333–356, June 1987.

[16] A. K. Mishra, C. Fermuller, and Y. Aloimonos, "Active Segmentation for Robots," in *IEEE/RSJ Int. Conf. Intelligent Robots and Systems*, (St. Louis, MO, USA), pp. 3133–3139, October 2009.

[17] P. K. Allen, B. Yoshimi, and A. Timcenko, "Real-Time Visual Servoing," in *Proc. IEEE Conf. Robotics and Automation*, (Sacramento, CA USA), pp. 851–856, April 1991.

[18] K. Tarabanis, P. K. Allen, and R. Y. Tsai, "A Survey of Sensor Planning in Computer Vision," *IEEE Trans. Robotcis and Automation*, vol. 11, no. 1, pp. 86–104, 1995.

[19] N. Krueger, C. Geib, J. Piater, R. Petrick, M. Steedman, F. Woergoetter, A. Ude, T. Asfour, D. Kraft, D. Omrcen, A. Agostini, and R. Dillmann, "Object-Action Complexes: Grounded Abstractions of Sensory-Motor Processes," *Robotics and Autonomous Systems*, vol. 59, no. 1, pp. 23–40, 2011.

[20] H. Weyl, *Symmetry*. Princeton, NJ: Princeton University Press, 1952.

[21] Y. Liu, H. Hel-Or, C. Kaplan, and L. V. Gool, *Computational Symmetry in Computer Vision and Computer Graphics*. Hanover, MA: Now Publishers, Inc., 2010.

[22] T. Henderson and Y. Fan, "RobotShare: A Google for Robots," *Special Issue on Cognitive Humanoid Robots of the Int. J. Humanoid Robotics*, vol. 5, pp. 311–329, June 2008.

[23] S. Reghizzi, *Formal Languages and Compilation*. Berlin, Germany: Springer Verlag, 2009.

[24] N. Fenton and M. Neil, *Risk Assessment and Decision Analysis with Bayesian Networks*. Boca Raton, FL: CRC Press, November 2012.

[25] G. Weiss, *Multiagent Systems*. Cambridge, MA: MIT Press, 2013.

[26] M. J. Wooldridge, *Reasoning about Rational Agents*. Intelligent Robots and Autonomous Agents, Cambridge, MA, USA: The MIT Press, 2000.

[27] P. Fankhauser, M. Bloesch, D. Rodriguez, R. Kaestner, M. Hutter, and R. Siegwart, "Kinect v2 for Mobile Robot Navigation: Evaluation and Modeling," in *Proc. IEEE Int. Conf. Advanced Robotics*, (Istanbul, Turkey), pp. 388–394, July 2015.

[28] T. Palys and W. Zorski, "Enhanced Movement Tracking with Kinect Supported by High-Precision Sensors," in *Federated Conf. Computer Science and Information Systems*, (Lódz, Poland), pp. 883–888, September 2015.

[29] L. Yang, L. Zhang, H. Dong, A. Alelaiwi, and A. E. Saddik, "Evaluating and Improving the Depth Accuracy of Kinect for Windows v2," *IEEE Sensors J.*, vol. 15, no. 8, pp. 4275–4285, 2015.

[30] S. Zennaro, M. Munaro, S. Milani, P. Zanuttigh, A. Bernardi, S. Ghidoni, and E. Menegatti, "Performance Evaluation of the 1st and 2nd Generation Kinect For Multimedia Applications," in *Int. Conf. Multimedia and Expo*, (Torino, Italy), pp. 1–6, June 2015.

[31] D. Pagliari and L. Pinto, "Calibration of Kinect for Xbox One and Comparison between the Two Generations of Microsoft Sensors," in *Sensors J.*, (Busan, South Korea), pp. 69–89, November 2015.

[32] F. Echtler, C. Kerl, L. Xiang, T. Wiedemeyer, Lars, R. Gordon, hanyazou, laborer2008, R. Wareham, M. Goldhoorn, F. Facioni, gaborpapp, alberth, S. Fuchs, Federico, jmtatsch, J. Blake, H. Jungkurth, Y. Mingze, vinouz, D. Coleman, R. Rawat, P. Reynolds, P. Viau, Ludique, Alistair, and J. Billingham, "libfreenect2: Release 0.1.1." [web page] https://github.com/OpenKinect/libfreenect2, January 2016.

[33] M. A. Fischler and R. C. Bolles, "Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography," *Commun. ACM*, vol. 24, pp. 381–395, 1981.

[34] M. Zuliani, "RANSAC Toolbox for Matlab." [web page] http://www.mathworks.com/matlabcentral/fileexchange/18555, November 2008.

[35] J. M. Esposito, B. Owen, J. Koehler, and D. Lim, "Matlab Toolbox for the Create Robot," August 2011.

[36] T. Henderson, *Analysis of Engineering Drawings and Raster Map Images*. New York, NY: Springer Verlag, 2014.

[37] T. C. Henderson, N. Boonsirisumpun, and A. Joshi, "Actuation in Perception: Character Classification in Engineering Drawings," in *Proc. IEEE Conf. Multisensor Fusion and Integration for Intelligent Systems*, (San Diego, CA, USA), pp. 145–151, September 2015.

[38] R. Plamondon, "A Kinematic Theory of Rapid Human Movements. I: Movement Representation and Generation," *Bio. Cybernetics*, vol. 72, no. 4, pp. 295–307, 1995.

[39] R. Plamondon, "A Kinematic Theory of Rapid Human Movements. II: Movement Time and Control," *Bio. Cybernetics*, vol. 72, no. 4, pp. 309–320, 1995.

[40] R. Plamondon, "A Kinematic Theory of Rapid Human Movements. III: Kinematic Outcomes," *Bio. Cybernetics*, vol. 78, no. 2, pp. 133–145, 1998.

[41] R. Plamondon, C. O'Reilly, J. Galbslly, A. Almaksour, and E. Anquetil, "Recent Developments in the Study of Rapid Human Movements with the Kinematic Theory: Applications to Handwriting and Signature Synthesis," *Pattern Recognition Lett.*, vol. 35, no. 1, pp. 225–235, 2014.

[42] A. Harley, A. Ufkes, and K. G. Derpanis, "Evaluation of Deep Convolutional Nets for Document Image Classification and Retrieval," in *Int. Conf. Document Analysis and Recognition*, (Nancy, France), pp. 991–995, August 2015.

[43] L. Kang, P. Ye, Y. Li, and D.Doermann, "A Deep Learning Approach to Document Image Quality Assessment," in *Proc. IEEE Int. Conf. Image Processing*, (Paris, France), pp. 2570–2574, October 2014.

[44] S. Marinai, "Introduction to Document Analysis and Recognition," in *Stud. Computational Intelligence*, (Berlin, Germany), pp. 1–20, Springer, 2008.

[45] C. Sprunk, G. Parent, L. Spinello, G. D. Tipaldi, W. Burgard, and M. Jalobeanu, "An Experimental Protocol for Benchmarking Robotic Indoor Navigation," in *Proc. Int. Symp. Experimental Robotics*, (Marrakech, Morocco), pp. 487–504, June 2014.

[46] J. Biswas and M. Veloso, "Depth Camera-based Indoor Mobile Robot Localization and Navigation," in *Proc. IEEE Int. Conf. Robotics and Automation*, (St. Paul, MN, USA), pp. 1697–1702, May 2012.

[47] J. Biswas, *Vector Map-Based, Non-Markov Localization for Long-Term Deployment of Autonomous Mobile Robots*. Pittsburg, PA USA: PhD Thesis 2014, The Robotics Institute, Carnegie Mellon University.

[48] J. Biswas and M. Veloso, "Wifi Localization and Navigation for Autonomous Indoor Mobile Robots," in *Proc. IEEE Int. Conf. Robotics and Automation*, (Anchorage, AK, USA), pp. 4379–4384, May 2010.

[49] J. Biswas and M. Veloso, "Multi-sensor Mobile Robot Localization for Diverse Environments," in *RoboCup 2013: Robot World Cup XVII: 468-479*, (Eindhoven, Netherlands), pp. 468–479, Springer Berlin, June 2013.

[50] J. Sturm, S. Magnena, N. Engelhard, F. Pomerleau, F. Colas, D. Cremers, R. Siegwart, and W. W. Burgard, "Towards a Benchmark for RGB-D SLAM Evaluation," in *RGB-D Workshop on Advanced Reasoning with Depth Cameras at Robotics: Science and Systems Conf.*, (Los Angeles, CA, USA), June 2011.

[51] J. Biswas, B. Coltin, and M. Veloso, "Corrective Gradient Refinement for Mobile Robot Localization," in *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems*, (San Francisco, CA, USA), pp. 73–78, September 2011.

[52] A. Joshi and T. C. Henderson, "Wreath Product Cognitive Architecture (WPCA)," in *Proc. IEEE Conf. Multisensor Fusion and Integration for Intelligent Systems*, (Baden-Baden, Germany), September 2016. (To Appear).

[53] J. Cai, X. Wan, M. Huo, and J. Wu, "An Algorithm of Micromouse Maze Solving," in *Proc. IEEE Int. Conf. Computer and Information Technology*, (Bradford, UK), pp. 1995–2000, June 2010.

[54] J. Cai, J. Wu, M. Huo, and J. Huang, "A Micromouse Maze Solving Simulator," in *Proc. IEEE Int. Conf. Future Computer and Communication*, vol. 3, (Wuhan, China), pp. 686–689, May 2010.

[55] S. Mishra and P. Bande, "Maze Solving Algorithms for Micro Mouse," in *Proc. IEEE Int. Conf. Signal Image Technology and Internet Based Systems*, (Bali, Indonesia), pp. 86–93, November 2008.

[56] A. B. S. Saman and I. Abdramane, "Solving a Reconfigurable Maze using Hybrid Wall Follower Algorithm," *Int. J. Computer Applications*, vol. 82, pp. 22–26, November 2013.

[57] A. Gelb, J. F. Kasper, R. A. Nash, C. F. Price, and A. A. Sutherland, eds., *Applied Optimal Estimation*. Cambridge, MA: MIT Press, 1974.

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

[59] T. Henderson, A. Joshi, and W. Wang, “The Cognitive Symmetry Engine,” Tech. Rep. UUCS-13-004, The University of Utah, School of Computing, Salt Lake City, UT, USA, September 2013.

[60] T. Henderson and T. Beall, “A Sensorimotor Approach to Concept Formation using Neural Networks,” in *Proc. IEEE Conf. Multisensor Fusion and Integration for Intelligent Systems*, (Baden-Baden, Germany), September 2016. (To Appear).

[61] J. Schmidhuber, “Deep Learning in Neural Networks: An Overview,” *Neural Networks*, vol. 61, pp. 85–117, 2015.

[62] T. Henderson, N. Boonsirisumpun, and A. Joshi, “Symmetry Based Semantic Analysis of Engineering Drawings,” in *Proc. IEEE Conf. Multisensor Fusion and Integration for Intelligent Systems*, (Beijing, China), pp. 1–6, September 2014.