ABSTRACT

Back-Projective Priming: Toward Efficient 3d Model-based Object Recognition via Preemptive Top-down Constraints

by Ryan Dellana

July, 2016

Director of Thesis: Dr. Ronnie W. Smith

Major Department: Computer Science

This thesis introduces back-projective priming, a computer vision technique that synergistically fuses object recognition and pose estimation by augmenting 3D models with geometric constraints. It also enables the use of image features too indistinct for use by other model fitting algorithms such as geometric hashing. To efficiently accommodate features that do not provide a scale attribute, we've developed a "match pair" finding heuristic called second-order similarity that reduces model fitting time complexity from a worst case of $O(N^2)$ to $O(N*Log(N))$.

An object recognition problem that is simple, practical, and well explored by other researchers is the problem of locating electrical outlets from the vantage point of a mobile robot. To demonstrate the relative merits of back-projective priming, we use it to build a system capable of locating generic electrical outlets in unmapped environments. Compared to our baseline algorithm, back-projective priming is shown to provide superior sensitivity when dealing with the challenges of low contrast, perspective distortion, partial occlusion, and decoys.

Back-Projective Priming: Toward Efficient 3d Model-based Object Recognition via Preemptive Top-down Constraints

A Thesis

Presented to the Faculty of the Department of Computer Science

East Carolina University

In Partial Fulfillment of the Requirements for the Degree

Master of Science in Computer Science

by

Ryan Dellana

July, 2016

Back-Projective Priming: Toward Efficient 3d Model-based Object Recognition via Preemptive Top-down Constraints

by

Ryan Dellana

APPROVED BY:

DIRECTOR OF THESIS:_____
Ronnie W. Smith, PhD

COMMITTEE MEMBER:_____
M. H. Nassehzadeh Tabrizi, PhD

COMMITTEE MEMBER:_____
Qin Ding, PhD

CHAIR OF THE DEPARTMENT OF COMPUTER SCIENCE:

_____
Venkat N. Gudivada, PhD

DEAN OF THE GRADUATE SCHOOL:

_____
Paul J. Gemperline, PhD

ACKNOWLEDGEMENTS

I'd like to thank my advisor, Dr. Ronnie Smith, the other members of my thesis committee, and my parents for their support throughout this process.

TABLE OF CONTENTS

LIST OF TABLES …. vii

LIST OF FIGURES …. viii

CHAPTER 1: INTRODUCTION …. 1

CHAPTER 2: THE PROBLEM …. 3

2.1 Electrical Outlet Discovery …. 3

2.2 Other Self-Feeding Robots …. 4

CHAPTER 3: TOOLS AND FORMALISMS …. 8

3.1 Coordinate Systems …. 8

3.2 Kinematic Tree …. 10

3.3 Mapping Between 3D Space and 2D Images …. 11

3.4 Oriented Edges …. 12

CHAPTER 4: BACK-PROJECTIVE PRIMING …. 13

4.1 Background …. 13

4.2 Prime …. 14

4.3 Sense …. 17

4.3.1 Feature Extraction …. 19

4.3.2 Feature Matching …. 20

4.3.3 Finding Match Pairs …. 22

4.3.4 Generating Fits …. 25

4.3.5 Evaluating Potential Fits …. 25

4.3.6 Fit Clustering …. 26

4.3.7 Optional Post-Validation …. 26

CHAPTER 5: EXPERIMENTAL DESIGN …. 27

5.1 Introduction …. 27

5.2 The Robot …. 27

5.3 Test Configurations …. 28

5.4 Data Collection …. 30

5.5 Metrics …. 31

5.6 The Baseline Algorithm …. 32

CHAPTER 6: RESULTS …. 34

CHAPTER 7: CONCLUSIONS AND FUTURE WORK …. 39

REFERENCES …. 41

APPENDIX A: DRAWING FROM COGNITIVE SCIENCE …. 43

LIST OF TABLES

1. Table 6.1: True-Positive Rate by Angle and Distance …..... 34

2. Table 6.2: False-Positive Rate by Angle and Distance …..... 35

3. Table 6.3: True-Positive Rate by Scenario and Angle …..... 35

4. Table 6.4: False-Positive Rate by Scenario and Angle …..... 36

5. Table 6.5: Translation Error by Scenario and Angle …..... 37

6. Table 6.6: Rotation Error by Scenario and Angle …..... 37

7. Table 6.7: Accuracy of Back-Projective Priming …..... 38

LIST OF FIGURES

1. Figure 3.1: Translation (a) and Rotation (b) …. 9

2. Figure 3.2: Euler Angles …. 9

3. Figure 3.3: Kinematic Tree …. 10

4. Figure 3.4: solvePNP and projectPoints …. 11

5. Figure 3.5: Extracting Oriented Edges …. 12

6. Figure 4.1: Pose-Space Samples (right) and their Back-Projections (left) …. 15

7. Figure 4.2: The Steps of Sense …. 17

8. Figure 4.3: Feature Lists …. 22

9. Figure 5.1: Mongo Robot (a), Range of Motion (b), Arm Camera View (c) …. 28

10. Figure 5.2: Testing Environment …. 29

11. Figure 5.3: Outlet Configurations …. 30

12. Figure 5.4: Test Angles …. 31

13. Figure 5.5: Baseline Algorithm …. 33

14. Figure A.1: Kanizsa's Triangle (a) and Feedback Pathways (b) …. 45

CHAPTER 1: INTRODUCTION

Robust object recognition is a central goal in the discipline of computer vision. Yet, as the blind man illustrates, knowing what and where things are is a problem that goes beyond vision. While the meta-problem of perception spans all the areas that comprise cognitive science, most computer vision research is confined to computer science, addressing narrowly-defined problems. Consistent with the software engineering best practice of creating modules with high cohesion and low coupling, solutions to these problems typically focus on the intrinsic features of objects, and rarely take advantage of context. In real-world scenarios, objects can appear ambiguous due to the effects of lighting, perspective, and occlusion. In such situations, context provides the only means of resolving this ambiguity. Taking general inspiration from gestalt psychology, neuroanatomical findings, and the success of hierarchical pattern recognition approaches, a novel framework has been developed for context-based object-recognition/pose-estimation. High-level geometric constraints are used to guide fitting of a 3d model to a 2d image through a process termed "back-projective priming" (bp-priming) [1].

BP-Priming enables the synergistic fusion of object recognition and pose estimation, plus the use of image features too indistinct for use by other model fitting algorithms such as geometric hashing. To efficiently accommodate features that do not provide a scale attribute, we've developed a “match pair” finding heuristic called second-order similarity that reduces model fitting time complexity from a worst case of $O(N^2)$ to $O(N*Log(N))$.

As a specific, well-studied, problem within mobile robotics, "electrical outlet discovery" is used for benchmarking performance. An experimental setup, consisting of a mobile robot with a "plug arm" and a collection of interchangeable prop-walls and outlets has been constructed for validation. The results show that, in object recognition scenarios with known geometric constraints, bp-priming offers improved sensitivity, efficiency, and robustness to visual obfuscators compared with a baseline system using only bottom-up processing.

Chapter 2 details the constrained version of the electrical outlet discovery problem and examines robots developed by other researchers that solve similar problems. Tools and formalisms foundational to bp-priming and our baseline algorithm are introduced in chapter 3, with bp-priming explained in chapter 4. Chapters 5 and 6 deal with the experimental design and results, with the baseline algorithm that we use for comparison purposes described in section 5.6. In the final chapter, we discuss the strengths and limitations of the current system and possible strategies for improving and expanding it. For a broader discussion of the cognitive science concepts that motivate this work, see Appendix A.

CHAPTER 2: THE PROBLEM

2.1 Electrical Outlet Discovery

Suppose we have a mobile robot that needs to recharge itself. If it only operates in a single building, a good solution might be to build a custom charging station with some kind of unique visual marker to make it easy to find. However, a more adaptive robot should be able to use standard outlets without being told where to find them or what exactly they look like, the problem of *electrical outlet discovery*. Additionally, in order to actually plug in, the robot needs to know the position and orientation of the outlet, the problem of *pose estimation*.

As we will see in the next section, specialized sensors like laser scanners or stereo-vision enable simple foreground extraction, which makes object recognition much easier to solve. Yet, these "shortcuts" distract from the broader perceptual problems, largely freeing the robot from having to "understand" what it's looking at. In contrast, when tele-operating the robot with a view from a single camera, a human can easily find an outlet and plug the robot into it. So, in an effort to keep the solution "cognitive", we prohibit the use of depth sensors in our version of the problem, allowing the use of only a single camera. However, we do allow the robot to use information about the position and movement of its various degrees of freedom (i.e. proprioception).

The average human has thousands of hours of visual experience with indoor environments, and so has a vast store of "common sense" to draw upon. In lieu of this, we'd like to supply the robot with the subset of North American building code that outlines the valid structure and placement of walls, electrical outlets, light switches, phone jacks, windows, doors, etc. Since codes across the country vary slightly and are not always adhered to strictly, we can assume outlet placement will be consistent with

that of local residential structures. This would provide a general set of constraints without a detailed map of the building or template of the outlet. To keep the scope of this thesis manageable, we will focus only on the outlet cover, which is modeled as a 3" by 4.75" rectangle. Besides its simplicity, another advantage of using the outlet cover is that its contours remain visible at extreme angles. Future work will attempt to incorporate other building features and use a more detailed outlet model that allows discrimination between outlets and cable jacks.

2.2 Other Self-Feeding Robots

Several notable electrical-outlet-seeking robots have been developed since 2000, the two most recent by former technology incubator Willow Garage. They initially developed ROS, an open source "Robot Operating System", used by many different robots including ours. In addition to providing a common development and runtime framework to simplify collaboration between researchers, ROS offers a large collection of useful modules for mapping, navigation, motion planning, computer vision, visualization, and simulation, as well as drivers for common sensors and robotic hardware. Alongside this software framework, Willow Garage developed the PR2 robotics platform, and by 2009 had achieved continuous indoor operation using autonomous door opening and plugging in. Their initial system is described in [2] (System 1), and a significant improvement in outlet recognition/pose-estimation is explained in [3] (System 2). Both use a 2D occupancy grid map pre-annotated with outlet positions in conjunction with the standard ROS navigation stack to initially get within the general region (about 3m x 3m) of an outlet. While this means that neither is technically performing outlet discovery, the map is 2D and only annotated approximately, so they still have to estimate outlet pose in the image.

Both systems use the PR2 equipped with a six degree of freedom robotic arm featuring accurate position control and force sensing. The pan-and-tilt head contains two pairs of cameras with each pair providing stereo-vision for depth perception. There are also two laser scanners ("lidars"), one at ankle height to provide obstacle detection around the base, and the other mounted on a tilting platform at shoulder-height to give depth information within the robot's work envelope. Unlike cameras, which are subject to the effects of lighting, laser scanners work by measuring the time-of-flight of laser pulses reflected off a rapidly rotating mirror, giving robust depth information within a planar region. In the head, System 1 contains an additional high-resolution 5 megapixel camera, while System 2 instead has a wrist-camera to provide a view of what's directly in front of the gripper.

Both systems perform outlet recognition in two phases, far-field and near-field. In far-field, a depth map obtained from stereo-vision is used to identify outlet candidates on the texture-less white wall via an unspecified method. Candidates are removed if they are non-planar, or outside expected bounds on size/height. Next, frontal views of the remaining candidates are generated through perspective rectification using the wall pose obtained from the lidar in the base. The frontal views are then compared against an outlet template to eliminate more false-positives. The robot drives to within half a meter of the closest remaining candidate, and performs near-field recognition to get a more accurate pose-estimation.

The near-field algorithm of System 1 requires an outlet to have a cluster of four sockets darker than their surroundings, and it's also implied these sockets need to be orange. The centers of the "socket-blobs" provide the four non-collinear points required to solve for outlet pose using the standard PnP-solve (Perspective-n-Point) algorithm [4]. To insert the plug, visual differencing is used, a technique that involves tracking the plug, and gradually moving it visually closer to the socket. Even

with visual-differencing, System 1 is only accurate to within one centimeter, which usually prevents it from correctly lining up the plug on the first try. To deal with this, they use a trial-and-error approach where the robot makes multiple plugging attempts in a spiraling pattern using force feedback to detect success.

The near-field detection algorithm of System 2 is accurate to within less than a millimeter, eliminating the need for visual differencing. It uses the wrist-camera for a closer view so it can reliably detect sockets using just the holes, which are found by looking for small regions that are darker than their surroundings. This use of socket holes enables it to detect most standard outlets rather than being limited to four-socket outlets like System 1.

Despite its impressive accuracy, System 2 is still limited in terms of generalizability and performance. It doesn't work when outlet holes are not visible such as for dark sockets or viewing angles greater than 60 degrees. It also requires the wall pose obtained from the lidar to accurately calculate outlet pose, while many robots are not equipped with laser scanners. Because there are a large number of potential socket holes detected at different scales, the algorithm spends a lot of time trying to find the best combination using geometric hashing [5].

Two older systems described in [6] and [7] are also worth noting, as both wander around without a map and so perform actual outlet discovery. Mechanically [6] consists of a custom 2 degree-of-freedom plug arm mounted atop a standard omni-directional mobile base, while [7] is unfinished. [7] scans along walls with the aid of a lidar that is used in conjunction with a zoom camera to maintain a consistent field-of-view. This enables it to use a single fixed-size socket template for matching. While a unique approach, the use of a depth sensor and specialized camera places it outside of our problem

scenario, and the use of a template limits its robustness to perspective distortion and occlusion.

All things considered, the problem solved by [6] is the most similar to our own, in that it does outlet discovery using only a single monocular camera. This is accomplished using a Viola Jones cascade detector trained on a manually-labeled dataset of 846 positive and 1400 negative instances. Cascade classifiers are an excellent example of boosting and have historically been popular for their efficiency. Their disadvantages include the large amount of work required to create training sets, and their inability to handle significant occlusion, variation in viewing angle, or deviation from the training-set. They also don't provide any pose information, which makes it unclear how [6] accomplishes this.

All the systems we've looked at are custom tailored to detect standard outlets and have low tolerance for occlusion and perspective distortion. They tend to use highly modular, strictly bottom-up perceptual hierarchies, with pattern recognizers at each level set to high sensitivity producing many false-positives which are filtered out after-the-fact at higher levels using context. The detection and elimination of these false-positives is computationally costly, and even with sensitivity set fairly high, many true-positives are missed due to noise and missing data. To overcome these limitations, our approach is to use context proactively (a.k.a. priming) to selectively boost sensitivity to probable patterns while reducing sensitivity to improbable ones. This gives higher true-positive rates while avoiding the cost of filtering out false-positives.

CHAPTER 3: TOOLS AND FORMALISMS

3.1 Coordinate Systems

To reason about the 3-dimensional world, a system called tf [8] is used. Tf is part of ROS and can stand for either Transform, or the name of it's creator, Tully Foote. Tf treats the world as a 3-dimensional Euclidean space represented using Cartesian coordinates. Various Cartesian frames of reference are defined, usually bound to objects of interest, with the fixed frame being defined relative to the task at hand. For example, during robot navigation, an odometry frame is typically defined as fixed to the spot on the floor where the robot started. The location of the robot is thus given by the position of the robot's base frame (centered between the drive wheels) relative to the odometry frame. In other situations, such as when attempting to grasp an object, we are interested in the relative positions of the gripper, camera, and target object frames. Tf is used both to spawn frames and query for their relative positions across time.

The relative position between two frames is referred to as the transform between them. A transform from a frame P to another frame Q, describes how you would translate and rotate P in order to bring it to the position of Q. These translation and rotation vectors, taken together, define the pose of Q relative to P. In tf, translation is measured in meters and represented using a three-tuple consisting of (X, Y, Z), representing forward, left, and upward displacement (Figure 3.1a).

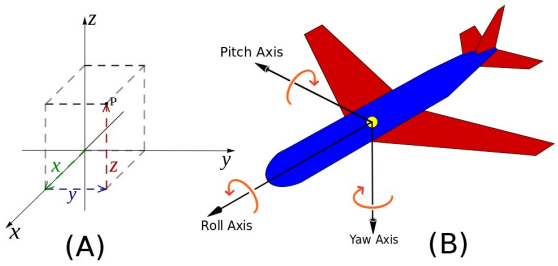

Figure 3.1 Translation (a) and Rotation (b)

There are many different methods of representing rotation in a Euclidean space including 3x3 matrix, quaternion, Euler angles, and axis-angle (a.k.a. Rodrigues). Tf uses quaternions natively, but includes functions to convert to and from Euler. The computer vision framework, Opencv, uses the axis-angle representation. We use Euler angles for their more intuitive nature, and handle conversions in a custom Pose class. Euler rotation vectors are still more difficult to work with than translation vectors, because, unlike the components of the translation vector, the order in which the components of rotation are applied affects the outcome of the rotation. Euler angles have traditionally been used in aviation which is why their three components are often referred to as yaw, pitch, and roll (Figure 3.1b). More specifically, a Euler angle vector is a 3-tuple which describes a sequence of three "intrinsic rotations" (in radians) that carry the coordinate frame with them. Because the coordinate frame rotates, each intrinsic rotation changes the meaning of each subsequent rotation (Figure 3.2).

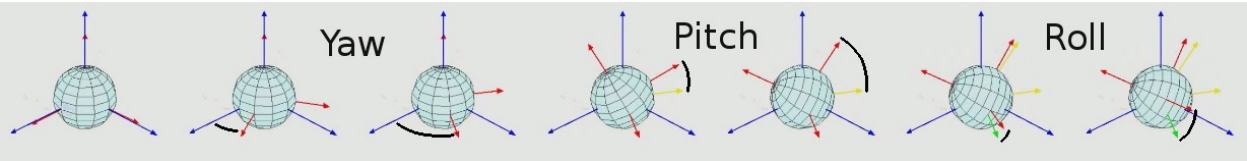

Figure 3.2 Euler Angles

3.2 Kinematic Tree

To model the form and motion of a robot, tf frames are arranged into a kinematic tree/chain (Figure 3.3), where each child frame has one or more degrees of freedom relative to its parent frame. Figure 3.3 shows the kinematic tree of our test robot, which will be described in detail in Section 5.2. The “elevator” frame is the child of “base_link”, and has a single linear degree of freedom, thus allowing it to raise/lower relative to base_link. The “arm” frame, in-turn, is the child of elevator, and has a single rotational degree of freedom relative to it. The “camera” frame is fixed relative to the arm frame. To represent the 3d position of a detected external object, a tf frame for the object is spawned relative to the camera frame. Tf can then be queried for the pose of said object relative to other important frames such as base_link. It can also be used to spawn "hypothetical frames" and subsequently get their poses relative to the camera for use in back-projecting hypotheses (this will be explained later).

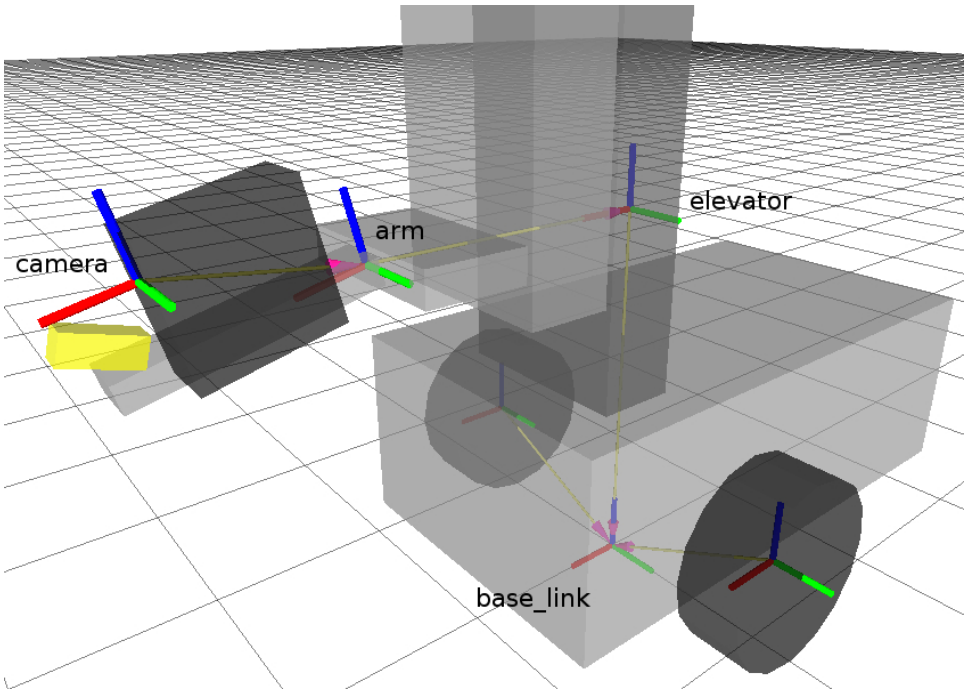

Figure 3.3 Kinematic Tree

3.3 Mapping Between 3D Space and 2D Images

Projecting a model into an image at a particular pose (a.k.a. rendering) is the "back-projection" in back-projective priming. When working in OpenCV, the model of an object consists of a set of 3D points defined relative to the object center. For example, in the model of a standard-sized outlet cover, these could correspond to the four corners. Model points are typically chosen to be easy to locate in a 2D-image using features such as texture points, corners, edges, contours, or color "blobs." Once three or more correspondences have been found, the pose of the model relative to the camera can be calculated. In OpenCV (cv2 library), this may be accomplished using the function cv2.solvePNP(model_points, image_points, ...), which returns the rotational and translational components of the pose as two separate vectors. When given the model and these vectors, the reverse of solvePNP can be performed using cv2.projectPoints(model_points, rotation_vector, translation_vector, ...), which gives the 2D image coordinates of the 3D model points when projected into the image. Figure 3.4 was generated using color tracking to locate four corners of a bright green outlet cover, followed by cv2.solvePNP to get the pose vectors, and then cv2.projectPoints to draw both the outlet model and the axis.

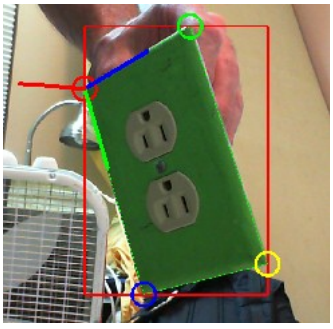

Figure 3.4 solvePNP and projectPoints

3.4 Oriented Edges

Oriented edges (Figure 3.5) are also important to bp-priming. Extraction begins with Canny edge detection (cv2.Canny) and dilation (cv2.dilate) with a 2x2 kernel to seal small edge gaps. The resulting binary image undergoes contour extraction (cv2.findContours), and then polygon approximation of the contours (cv2.approxPolyDP). These simplified contours are then broken up into individual edges which are represented using midpoint, length, and orientation angle in radians. By explicitly providing length and orientation, this representation makes it easier to meaningfully compare edges.

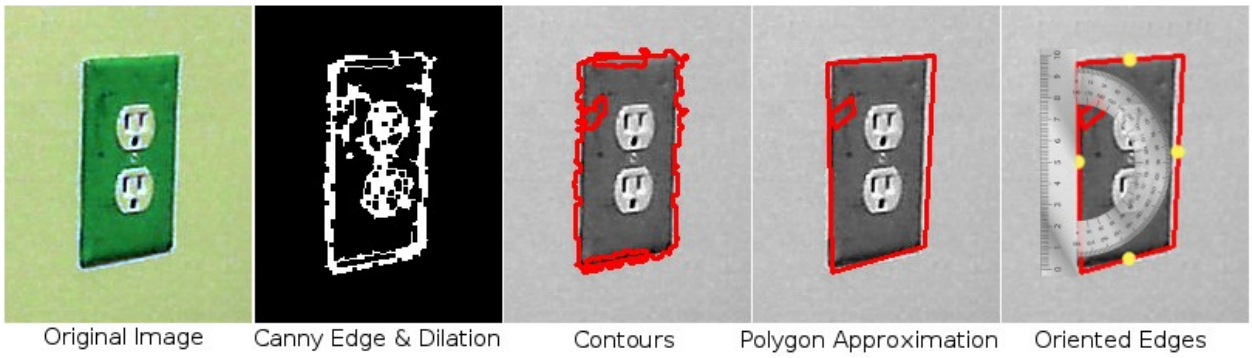

Figure 3.5 Extracting Oriented Edges

CHAPTER 4: BACK-PROJECTIVE PRIMING

4.1 Background

When asked to find an electrical outlet in a room, you probably start by locating a wall and then scanning across the section of it about a foot above the floor. In general, by imagining what an object might look like in the current context, you can prime your vision system. This is the basic idea behind back-projective priming. While the mechanism for this in the human brain likely has more in-common with neural net and stochastic approaches than with symbolic/rule-based ones, the use of explicit 3D-models and constraints offers the benefits of transparency, and is feasible when looking at the problem in its simple form.

It can be useful to view a building as a hierarchy of 3d models. At the top of the hierarchy is the building as a whole, which can be decomposed into the floor, ceiling, and walls. Walls, in turn, may contain other models such as doors, windows, baseboards, light switches, phone jacks, and electrical outlets, which themselves can be broken down further. Some models, such as the outlet cover, are easily defined using a static CAD model. Others, like walls, have some invariant attributes (ex: planar, rectangular, span floor to ceiling), but do not have a fixed 3d structure, instead being defined by a set of structural constraints yielding the space of possible 3d configurations. Perhaps this implies that concepts like "wall" should be discarded in favor of a set of primitive static models. These sorts of ontological problems are what you'd expect to encounter when using an explicit approach, and the impetus for future work developing a more fluid non-symbolic one. For now we will look at an easily defined subset of building features to demonstrate the general concept.

Given knowledge about the relative locations of some models, we can constrain the space of possibilities in the search for others. Take, for instance, the constraint that any wall should have a pitch angle exactly 90 degrees greater than that of the floor, and an outlet, in-turn, will have the exact same rotational vector as the wall that it's in. Since a wall will always have a fixed roll value of 0, both the pitch and roll values of any potential outlet are known a-priori. The z coordinate of the outlet is expected to be 12 inches above the floor plane, so that, overall, there are only three variable components of pose for any outlet, x, y, and yaw. If, however, we've already found a wall, then, relative to the wall's coordinate frame, the outlet can only vary in terms of the y component of translation. Finding a wall dramatically shrinks the space of possible outlet poses, and, contrariwise, finding an outlet would automatically indicate the presence/pose of a wall. In this way, as models are identified, they provide context (i.e. geometric constraints) which shrinks the pose-space of related models yet to be detected.

To keep the problem tractable while still demonstrating the benefits of priming, we've simplified the scenario to one of searching for the outlet cover with only the pose of the floor and camera known. For now, the constraints between the floor and outlet cover pose are hard-coded. In the future, a constraint programming framework may be used to incorporate other features. Our outlet model is a simple rectangle consisting of four points and four edges. Initially, no wall poses are known, which produces a space of possible outlet poses based on different combinations of the unbound variables x, y, and yaw.

4.2 Prime

Bp-priming assumes that we have a 3d model of a specific object that we wish to detect and localize, as well as geometric constraints on the pose of the model relative to some reference frame. In

this case, our model is a rectangle representing an outlet cover, and our fixed frame of reference is the floor in front of the camera, for which we use base_link (Figure 3.3). The two high-level operations are priming and sensing, where prime() takes the geometric constraints and propagates them down into the image plane so that sense(input_image) can use them to guide model fitting. The two operations are somewhat disjoint in that a call to prime() needn't be followed by sense(...), and sense(...) can be run repeatedly after only a single call to prime(). Let us take a look at the basic steps these functions perform.

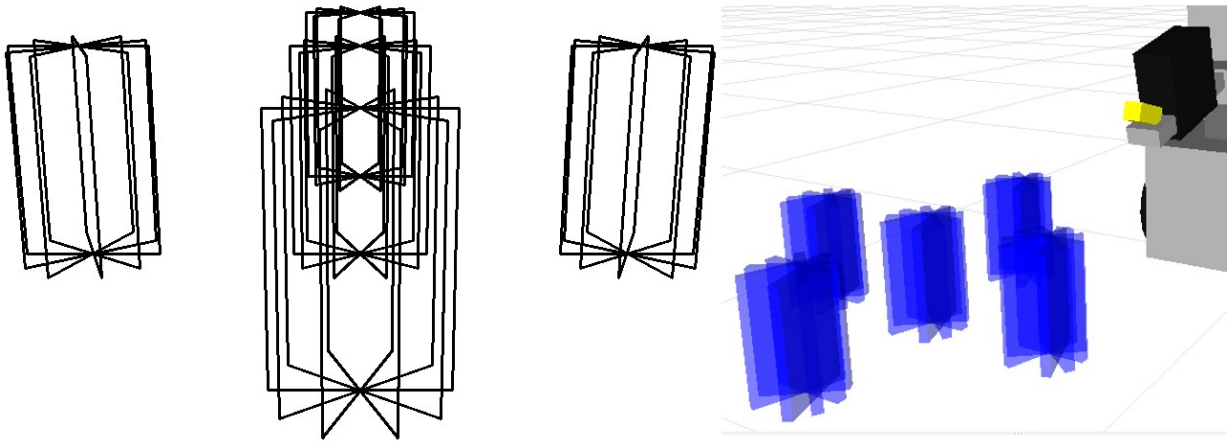

Figure 4.1 Pose-Space Samples (right) and their Back-Projections (left)

The steps of prime():

1. Generate a list of sample poses that fall within the pose-space defined by our constraints. This currently uses a hard-coded function that samples in uniform (x,y) increments in a circular area surrounding a reference frame approximately three feet in front of the camera. The cross shaped sample distribution shown in figure 4.1 is actually a circle. At each of these (x,y) positions, a fixed number of yaw angles are sampled (five in figure 4.1).
2. Spawn tf frames for these poses relative to the base_link frame.
3. Query for the poses of the frames relative to the camera frame.
4. For each pose relative to the camera, render (back-project) the model.
5. Extract features from each render, binding them to their corresponding back-projection. Note that "back-projection" refers to the combination of a model, pose, and the features extracted from the render of said model at said pose.

The end product of prime() is a collection of feature objects each bound to a particular back-projection. Figure 4.1 shows the sampled 3d poses in tf (right) and a composite image of their corresponding 2d renders (left). Note that the number of samples in this example is kept small for illustration purposes.

For this artificially narrow problem, the pose-space is simply the set of all possible outlet poses where the pitch and roll are 0 degrees and height/z is 12 inches. In the current implementation, the set of samples taken from this space consisted of 11 rotations at 10 different (x,y) positions for a total of 110 poses. As in Figure 4.1, the (x,y) sample region is a circle surrounding a reference frame about four feet in front of the robot. This is, of course, just an engineering expedience. Pose-space in a fully-developed implementation would be the space of object poses that do not violate the geometric

constraints currently at play. If, for example, the robot perceived a wall in front of it, the initially circular pose-space would collapse to a line running along the wall at 12 inches above the floor. A long-term goal of potential future research is to develop better ways to represent and optimally sample the pose-space while constraints change in real-time.

4.3 Sense

Once the system is primed, we are ready to start capturing images from the camera to be processed by sense(...). Let's get an overview of the steps involved, before going into further detail for each. Figure 4.2 (a) through (d) illustrate what happens in a call to sense() after having primed with the poses of Figure 4.1.

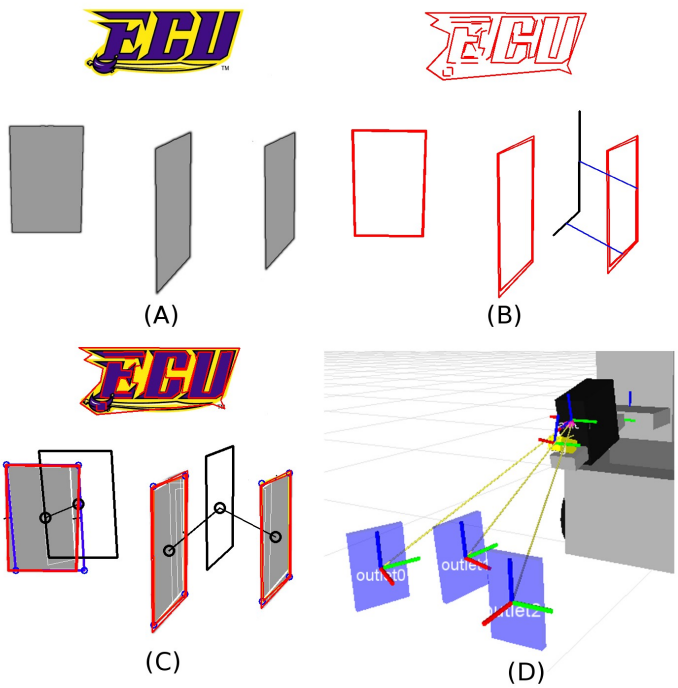

Figure 4.2 The Steps of Sense

The steps of sense(input_image):

1. Extract features from the image; in this case, oriented edges (Figure 4.2b).
2. Find matches between the image features and back-projection features (bp-features). At the conclusion of this step, each bp-feature will have a list of potential image-feature matches in descending order of similarity.
3. For each back-projection, attempt to find a list of good "match pairs" that can be used as a basis for fitting the model to the image. A match-pair is a set of two matches between image and bp-features (Figure 4.2b).
4. For each back-projection, use each match pair to translate and scale the model 2-dimensionally to generate a potential fit (Figure 4.2c). Note that each back-projection can generate multiple fits (Figure 4.2c).
5. Evaluate each potential fit in a second wave of feature matching with tighter tolerances than the first. If a fit is successful (i.e. is a correct guess), many features other than those in the match-pair will also find close matches in the image.
6. It is typical for multiple fits to be generated that share common features in the image. These fit-clusters must be collapsed to a single fit. Once the clusters are identified, choose the best fit in each cluster and ignore the rest.
7. As an optional final step, the pose of each fit can be post-validated using tf to ensure that the scaling and translation operations didn't put them outside of the constrained pose-space (Figure 4.2d).

Figure 4.2 is also deliberately simplified for clarity. The ECU logo was added to demonstrate how the system ignores features dissimilar from those that are primed. In addition to the image features, subfigure (b) shows one of the evaluated match-pairs used in generating the right-most fit in

subfigure (c). Notice in (c) how a single back-projection can generate more than one fit, in addition to how the 2d fitting transformations change both the position and scale of the back-projection.

To reiterate, a feature is either a bp-feature or image feature, with the former extracted from a back-projection and the latter from an image. A match is a tuple consisting of one bp-feature and one image-feature that are similar to each other. A match pair is a set of two matches, which provide two pairs of points to estimate relative scale during model fitting. A fit is an attempt to superimpose a back-projection over a particular part of the image. Now we will go into more detail about each step involved in sense(...), starting with a more in-depth look at features and feature matching.

4.3.1 Feature Extraction (Step 1)

Features in bp-priming are patterns extracted from the raw image or the render of a back-projection. They allow us to find similarities between different parts of the image (or between images). Features are composed of attributes, and a feature of a given class will share the same attributes as other members of that class. These attributes can be used to calculate the similarity or distance between features during matching. All feature classes share some common attributes including (x,y) position in the image, orientation, and scale. The attributes of an oriented edge feature include midpoint (position), angle (orientation), and length (scale). In most applications, these three basic attributes are not used in matching, but are rather viewed as a nuisance. Indeed, most features commonly used in computer vision are designed to be invariant to position, rotation, and scale so that matching can work despite changes in the pose of the target pattern (object) relative to the camera. However, in practice this is hard to achieve, as rotation of a complex 3d object can drastically change it's appearance. Most standard features like SIFT (Scale Invariant Feature Transform) work very poorly on objects that either lack a distinguishing visual "texture" or possess significant intra-class texture variation. Examples of

this include empty walls, transparent/translucent objects, and electrical outlets. In such cases, the only way to reliably identify objects is via a combination of context and contour features, and in the case of polyhedra, oriented edges. For an explanation of how oriented edges are extracted, refer to figure 3.5.

4.3.2 Feature Matching (Steps 2 & 5)

The similarity between two oriented-edges is one minus their average normalized attribute distance, which is the sum of the normalized distances between each attribute divided by the number of attributes. More formally, given two features of a class with n attributes, f1 and f2, their similarity is given by:

$$similarity(f1, f2) = 1.0 - \frac{1}{n} \sum_{attr=1}^{n} normDist_{attr}(f1_{attr}, f2_{attr})$$

The function $normDist_{attr}(...)$ gives the normalized distance between two values of a particular attribute, *attr*. In the case of (x,y) position, this is the euclidean distance divided by the width of the image. For angle, it's the percentage of the maximum possible difference in angle which maps (0° -> 90°) to (0.0 -> 1.0). For length, a relative difference of 100% is considered maximal, and anything above that is capped at 1.0. So, suppose we have an 800 pixel wide image in which oriented edge p is 20 pixels away from q, 50% longer than q, and 36 degrees different in angle, then the normalized distances for each of these attributes would be 0.025, 0.5, and 0.4 respectively. The average normalized distance would be 0.31, so the similarity would be 0.69.

In practice, the normalized distances are usually calculated relative to a tolerance which is set for each attribute on a case-per-case basis. If, for example, we were to set the tolerance for position to 0.1, the matcher would throw out any potential matches where position differs by more than 10% of the image width. In such a case, if we were to measure their position similarity relative to the full image width, we would never see similarity values less than 90% since all others would have been rejected

prior to comparison. This effectively reduces the relative influence of the position attribute in calculating similarity. By instead using the tolerance value as the denominator, we once again have similarity values ranging from 0% to 100%, thus increasing the relative weight of position. In general, the tighter the tolerance on a particular attribute, the higher the sensitivity of the matcher in calculating said attributes similarity, and thus the more small differences in said attribute influence the overall similarity value. In the first wave of matching (step 2) the tolerance values for position, length, and angle are set to 20%, 50%, and 10 degrees respectively. In this step we want to search a wide area for plausible fits at various scales, so we can only afford to be picky about angle, hence its relatively tight tolerance. In wave 2 (step 5), we want to make sure the features of a fit line up properly and are roughly the same scale as their corresponding image features. To accomplish this we tighten the tolerances for position and length to 0.8% and 8%.

Note that, in the actual implementation, the image width is considered 1.0, and all image coordinates are normalized relative to this. So in our example with an 800 pixel image, a feature at an x coordinate of 400 would be at position 0.5. The advantage of using normalized image coordinates is that they're independent of image resolution. In our previous similarity example, if (position, length, angle) tolerances were set at (0.1, 0.8, 0.5), the normalized distances would come out as (0.25, 0.625, 0.8) giving a similarity of 0.44. To account for the tolerances, the similarity expression can be modified as follows:

$$similarity(f1, f2) = 1.0 - \frac{1}{n} \sum_{attr=1}^{n} \frac{normDist_{attr}(f1_{attr}, f2_{attr})}{tolerance_{attr}}$$

Unfortunately, Python is relatively slow, so we want to minimize the number of matches evaluated. To help with this, OpenCV offers a heavily optimized matcher that can take two sets of normalized feature vectors and for each feature in the first set, find the k-nearest-neighbors in the

second set according to the euclidean distance. It would be ideal to rely completely on this matcher function, but there's one major limitation to it. The angle attribute of an oriented edge is modular (wraps back around) and there's no way to tell the matcher to treat it as such, making it unable to compare angles correctly. So, as a compromise, we use the matcher only on the position and length attributes. This narrows down the set of possible matches to k candidates, which can then be evaluated using the python-based similarity metric described above.

4.3.3 Finding Match Pairs (Step 3)

At the end of step 2, a given back-projection will have a list of matches for each of its features (Figure 4.3). Now the challenge is to figure out which pairs of matches are "good pairs." The first problem is that, as the length of the match list for each feature grows, the number of possible pairs grows by roughly n-squared where n is the number of items in each list. Specifically, if you have L lists each with N items, the number of possible pairs of items between the lists is given by:

$$f(L, N) = \binom{L}{2} \cdot N \cdot N$$

So, if a given back-projection had 10 matches for each of its 4 features, the number of pairs would be 600. If this is average for all back-projections, then multiply this figure by the number of back-projections. Supposing there were 30, this would result in the algorithm evaluating 18,000 pairs. To cut down on this number, we use a heuristic dubbed "second-order similarity." The basic observation underlying this is that a cluster of image features that will produce a good fit, will all tend to have very similar similarity values when paired with the features of the correct back-projection. The second-order similarity algorithm for finding "good pairs" works as follows:

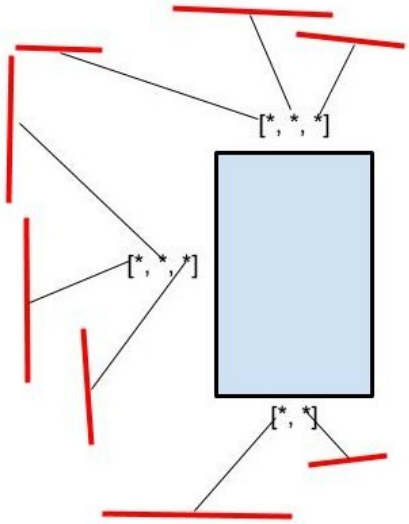

Figure 4.3 Feature Lists

```
# Concatenate the match lists for all features in a given back-projection
all_matches = [], good_pairs = []
for each feature in back-projection:
        for each match in feature:
                all_matches.append(match)
# Sort the concatenated list in descending order of similarity.
all_matches.sort()
# Iterate over the list and examine each adjacent pair of matches.
# If a given pair come from different parent lists, then add them to the “good pairs” list.
for i = 0 to all_matches.length - 2:
        if all_matches[i].parent_list != all_matches[i+1].parent_list:
                good_pairs.append((all_matches[i], all_matches[i+1]))
return good_pairs
```

When this finishes, we will have a list of match pairs with maximal second-order similarity. So how many pairs does this produce relative to the number that are possible? The length of the concatenated list is L*N, so the number of match pairs we examine is equal to L*N - 1, each of these either producing a "good pair" or not. For a given model, L is a constant, and sorting is only O((L*N)*log(L*N)), so the overall time complexity is O(N*log(N)) (much better than $O(N^2)$). However, to examine each "good pair" produced involves a relatively large time constant. As a result, the algorithm initially appears to scale approximately O(N), or time linearly related to the number of pairs examined. Going back to our example, this heuristic would yield up to 39 pairs per back-projection giving a total of 1,170 possible pairs (approximately 15 times fewer). So, in scenarios where the number of pairs is less than roughly 1000, approximately linear scaling means it's about 15 times faster.

Of course, not all match pairs with high second-order similarity necessarily produce good fits. All the metric tells us is if two matches are a similar distance from the back-projection, but they could be a similar distance in opposite directions. So, before a given match pair is used to generate a candidate fit, there are two additional tests that can be used to weed it out, angle similarity and scale similarity. Suppose we have a given match pair (bp1, im1) and (bp2, im2), where bp1 is a back-projection feature matched to image-feature im1. We simply compare the angle of the line between the position of bp1 and im1 to the line between bp2 and im2, and if they're beyond a certain tolerance (e.g. 10°) we discard the pair. Scale similarity works in much the same way, except we're comparing the ratio of the scale of bp1 and im1 to the ratio of the scale of bp2 to im2 (in the case of oriented edges, scale = length). The remaining match pairs are used to generate fits.

Unfortunately, it's also possible for a good match pair to be neglected by this heuristic. If a back-projection differs significantly in scale from the correct fit, it causes the relative translation vectors of the correct matches to be non-parallel. Non-parallel translation vectors result in differing second-order similarity of the position attribute, lowering the chances that they'll appear close to each other in the sorted match list. In practice, this can be somewhat mitigated by increasing the density of the pose-space sample, but at the cost of performance due to extra back-projections and match evaluations. A modified version of the heuristic that adjusts for differences in scale might also be possible if a custom feature matcher were to be developed.

4.3.4 Generating Fits (Step 4)

For each match pair of a given back-projection, we generate a potential fit by scaling and translating the model points. What this does is superimpose the back-projection onto the image so that the matching bp-features overlay their corresponding image-features (see Figure 4.2c).

4.3.5 Evaluating Potential Fits (Step 5)

The tolerances of the first matching wave were deliberately lax with the intent of generating lots of potential fits. These fits are subsequently evaluated using a second wave of feature matching, with feature similarity tolerances much tighter. Position and scale are expected to match almost perfectly, while angle is a little more lenient. Many fits fail to match a minimum of two features and are eliminated. For those that remain, strength is found using the average similarity of the top match for each feature, which is given by:

$$strength(fit) = \frac{1}{n} \sum_{feature=1}^{n} similarity(fit_{feature}, topMatch(fit_{feature}))$$

4.3.6 Fit Clustering (Step 6)

It is often the case that several different back-projections will generate fits that bind to the same image feature. If two fits are bound to at least one common image feature, then they are considered to be in the same fit cluster. The relationship is also transitive, so that two fits, A and C, can share zero features but still be in the same fit cluster because they both share a feature with another fit, B. Only the strongest fit in each cluster is retained.

4.3.7 Optional Post-Validation (Step 7)

As an optional final step, the pose of each fit can be post-validated using tf to ensure that the scaling and translation operations didn't put them outside of the constrained pose-space. In the case of electrical outlets, roll and pitch must be 0 degrees plus or minus 5 degrees, and height above the ground should fall within a range of 11 to 13 inches. This process is illustrated nicely by the baseline algorithm (see section 5.6).

The more back-projections that are generated in priming, the more densely packed their features are in the image plane, and so the tolerances during the first matching wave can be made tighter. This limits the amount a given back-projection can be scaled and translated, which reduces the chances of generating a fit that falls outside the constrained pose space. In such a scenario, post-validation may not be necessary. On the other hand, this produces more potential matches, which take time to evaluate. Future work will attempt to find the optimal balance between the costs of back-projection, wave one matching, and post-validation.

CHAPTER 5: EXPERIMENTAL DESIGN

5.1 Introduction

The goal of our experimental design is to measure the performance of bp-priming on the electrical outlet detection/pose-estimation example problem. A good design provides realism, variation, ground-truth, repeatability, and a baseline system for comparison. Data collected from an actual robot in a room provides the realism, with a reconfigurable prop wall for variation, and tracking points for ground truth. Static datasets gathered from this test scenario enable repeatable offline testing so that the effects of different algorithms and parameters can be fairly assessed. As we were unable to obtain implementations of algorithms used by other researchers, we crafted our own baseline system that works strictly bottom-up without any priming. We will now describe these elements in more detail.

5.2 The Robot

The robot (Mongo, Figure 5.1) uses a differential drive platform for mobility, elevator to adjust the height of the plug, and pivoting arm to control the pitch angle of the plug. When eventually completed, the arm will include a gripper assembly and provide general pick-and-place capabilities. This is why it features the pitch control instead of just having the plug fixed horizontally. Its senses include monocular vision and basic proprioception provided by a collection of encoders, limit switches, and a potentiometer. The plug is directly mounted to the end of the arm, in view of the single arm-mounted camera (Figure 5.1c). There are also plans to install a stereo vision pan-and-tilt "head" on top of the elevator. Mongo has been successfully plugging in under remote control, demonstrating the feasibility of the design.

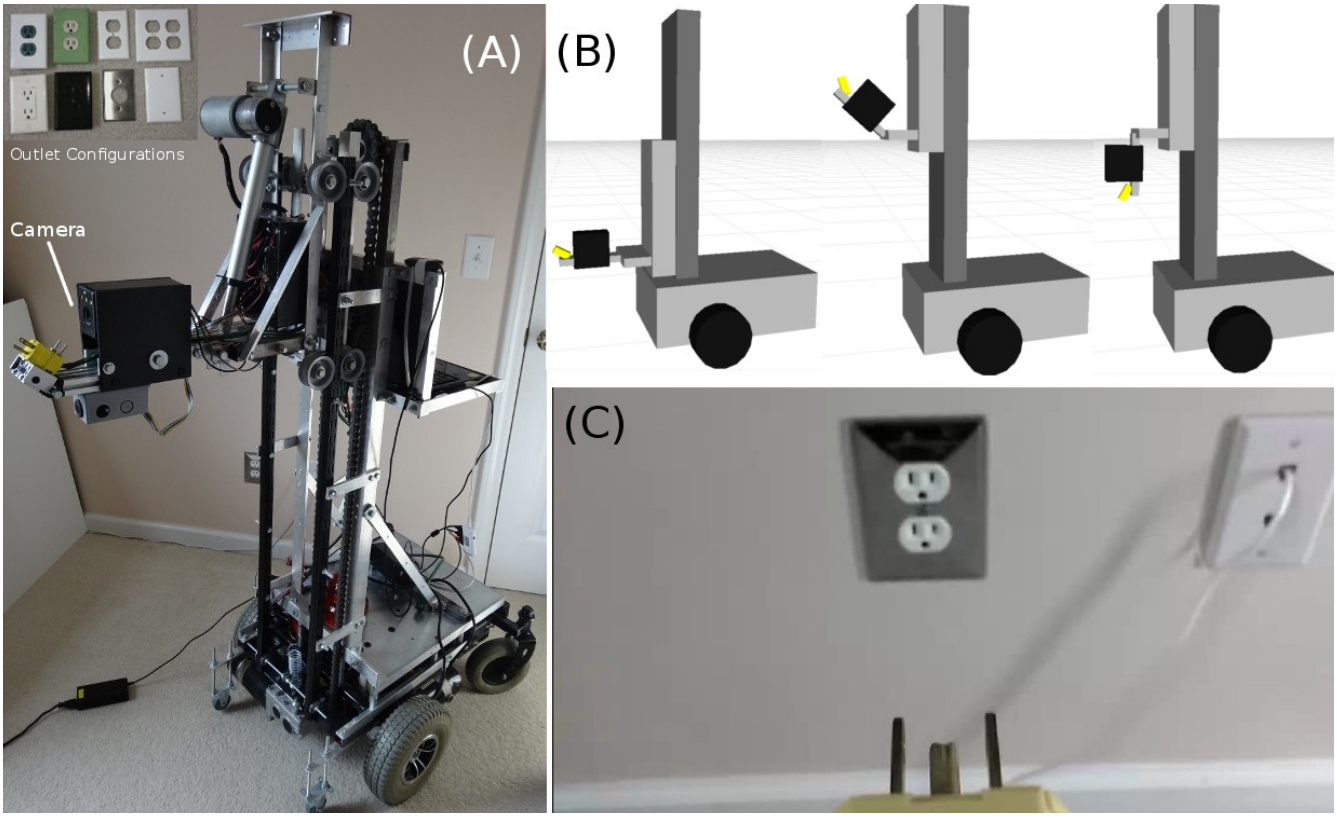

Figure 5.1 Mongo Robot (a), Range of Motion (b), Arm Camera View (c)

5.3 Test Configurations

In the test setup, the robot is situated in an open room facing a small section of fake wall with an outlet at its center (Figure 5.2). The wall is flat white with no discernible visual texture, and includes a white base-board for realism. Illumination is provided by three overhead compact florescent bulbs and is kept constant across different test scenarios. Affixed to the wall are four red ping-pong balls that serve as tracking-points to provide ground-truth for outlet position. Each tracking-point is offset from the center of the outlet by 0.20 meters in the Y and Z directions. To make a scenario more challenging, the prop wall can be altered in three ways. First, the standard green outlet cover can be replaced with a white cover that minimizes contrast with the wall, making it harder to detect edges. Second, the outlet can be partially occluded by placing a large piece of white tape across it diagonally. Lastly, decoy

outlets (index cards with an outlet photo) can be placed at incorrect positions on the wall to encourage false positives. These factors are combined to produce five distinct test configurations (Figure 5.3), one with no alterations, three each with just one alteration, and one with all three combined.

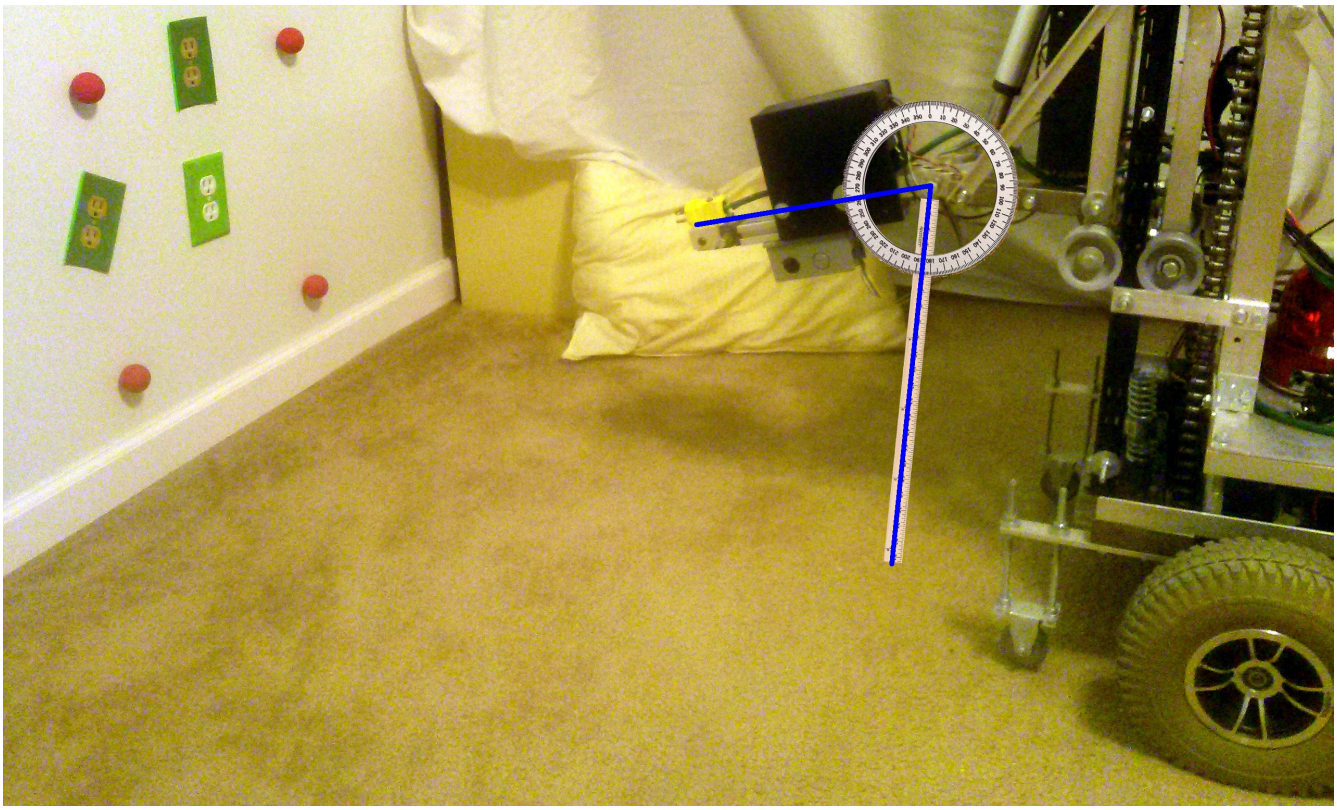

Figure 5.2 Testing Environment

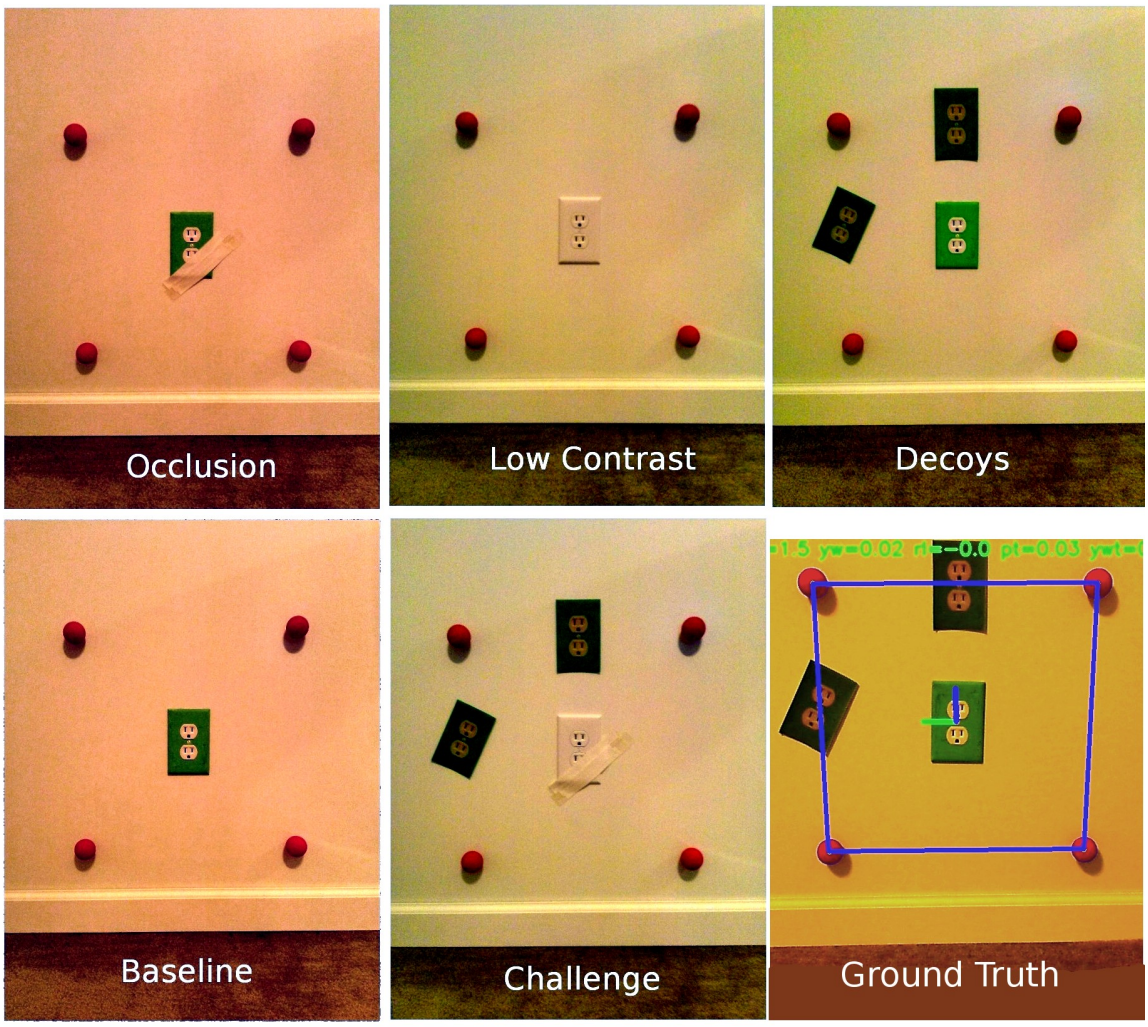

Figure 5.3 Outlet Configurations

5.4 Data Collection

To provide an ideal vantage point for gathering test data, the camera is kept fixed at 0.5 meters above the ground with a downward tilt angle of 15 degrees. Note that, in practice, information from the encoders can be used to handle variation in camera position. Once a wall configuration is set, the robot uses the tracker-points to precisely position itself in a predetermined sequence. For each of three viewing angles, 0, 45, and 85 degrees (Figure 5.4), the robot moves in a straight line towards the outlet in increments of 0.25 meters, starting at 2.25 meters away and finishing at 1.5 meters for a total of 12

angle-distance combinations. Ten images are captured at each position, along with the corresponding camera pose and ground-truth outlet-pose. The resulting 120 labeled images form a data-set used for off-line evaluation of both the baseline algorithm and bp-priming algorithm. During this evaluation, the algorithm under testing is re-initialized on each image, simulating how it would perform if the current image had been the first captured.

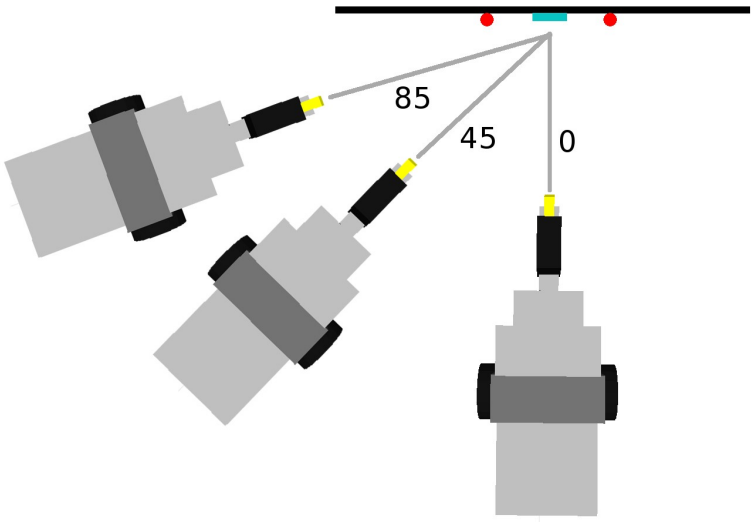

Figure 5.4 Test Angles

5.5 Metrics

A given algorithm outputs a list of detections with pose data for each. Although the detection aspect can be viewed as simple classification, the ability to detect multiple instances of the target model within the same image makes traditional metrics (e.g. precision/recall) less useful in gauging accuracy. Instead, we look at each detection separately, evaluating it as either a true-positive or false-positive. To be labeled as a true-positive, the translational component of a detection must fall within an "acceptance region" surrounding the ground truth frame. A detection counts as a true-positive if the 3d euclidean

distance between the detection and ground truth frames is less than 0.3 meters. If more than one detection falls within this region, then the one closest to ground truth is kept while the others are labeled as false-positives. For true-positives, we find pose estimation error by querying tf for the transform between the detection frame and ground-truth frame. The translational and rotational components of this transform constitute error values for the translational and rotational components of the detected pose.

5.6 The Baseline Algorithm

With a bottom-up detector, the earlier processing stages are not primed using constraints from later stages. With no pose constraints during feature matching, finding a potential outlet cover amounts to finding a rectangle that may have undergone perspective distortion. For such a simple shape, texture-based features like SIFT are of no use, leaving only basic geometric features like corners and oriented edges. Since we'd also like our solution to be robust to partial occlusion, as few as two edges should be sufficient to identify an outlet candidate. Unfortunately, this requirement makes a bottom-up detector infeasible, as almost any two connected edges could be interpreted as an outlet cover in one of several possible orientations. However, a less capable algorithm can be created with the concession that all four edges must be visible (i.e. no occlusion). Contour analysis allows us to find quadrilaterals which are each treated as candidate outlet covers. The four vertices of each quadrilateral are paired with the points of the model in a left-to-right, top-down order. These correspondences are then fed into solvePnP to get the pose of the candidate outlet cover. The last step is post-validation, in which each candidate is either accepted or rejected based on pose constraints. To be accepted, roll and pitch must be 0 degrees plus or minus 5 degrees, and height above the ground should fall within a range of 11 to 13 inches.

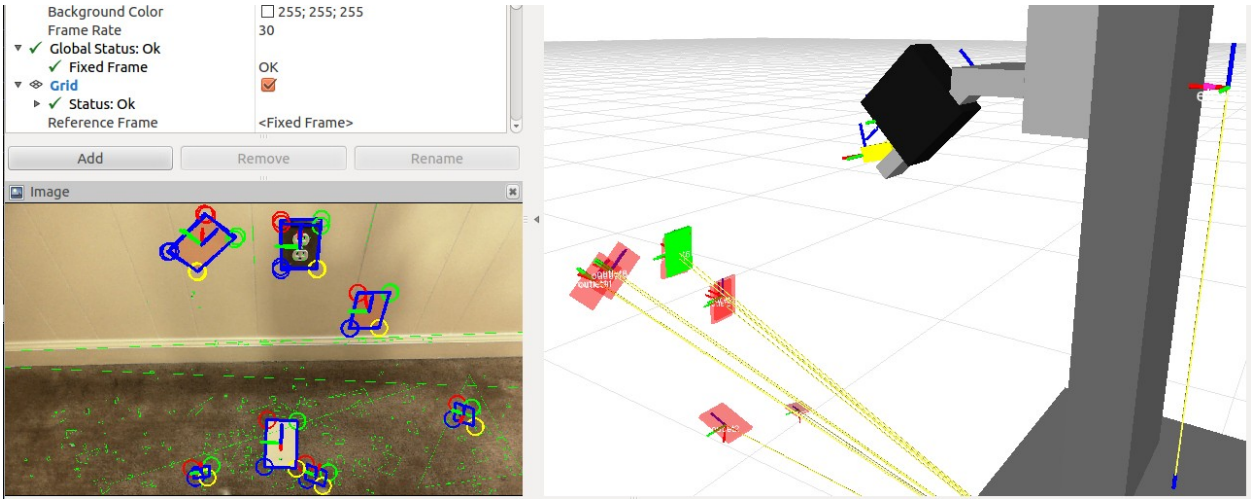

Figure 5.5 Baseline Algorithm

Figure 5.5 illustrates a case where we have one actual outlet and several index cards on the wall and floor to attempt to confuse the detector. Candidate poses are overlaid on the original image (left) and added as tf frames for validation (right). After first publishing the transforms relative to the camera frame, they are queried relative to the base-link (floor) frame and evaluated based on the geometric constraints. Those that passed post-validation are shown as green, while those that failed are red.

CHAPTER 6: RESULTS

True-Positive/False-positive rate is the proportion of images for which at least one outlet was detected within/outside-of the valid region. Translation and rotation error are calculated using euclidean distance and sum of pitch/roll/yaw difference in meters and radians respectively. The results are summarized in the tables below. For each combination of scenario, distance, and angle, the results for ten images are averaged. So, the total number of images = (scenarios * distances * angles * 10) = (5 * 4 * 3 * 10) = 600.

Table 6.1 True-Positive Rate by Angle and Distance

True-Positive Rate by Angle and Distance					
		0°	45°	85°	All
1.5m	Baseline	40%	44%	48%	44%
	BP-Priming	78%	92%	54%	75%
1.75m	Baseline	40%	52%	40%	44%
	BP-Priming	70%	82%	32%	61%
2.0m	Baseline	36%	40%	50%	42%
	BP-Priming	58%	74%	10%	47%
2.25m	Baseline	34%	44%	54%	44%
	BP-Priming	58%	42%	4%	35%
All	Baseline	38%	45%	48%	44%
	BP-Priming	66%	73%	25%	55%

Table 6.2 False-Positive Rate by Angle and Distance

False-Positive Rate by Angle and Distance					
		0°	45°	85°	All
1.5m	Baseline	0.0%	0.0%	10.0%	3.3%
	BP-Priming	2.0%	2.0%	0.0%	1.3%
1.75m	Baseline	0.0%	0.0%	6.0%	2.0%
	BP-Priming	4.0%	0.0%	0.0%	1.3%
2.0m	Baseline	0.0%	0.0%	0.0%	0.0%
	BP-Priming	2.0%	2.0%	2.0%	2.0%
2.25m	Baseline	0.0%	0.0%	0.0%	0.0%
	BP-Priming	2.0%	8.0%	2.0%	4.0%
All	Baseline	0.0%	0.0%	4.0%	1.3%
	BP-Priming	2.5%	3.0%	1.0%	2.2%

It can be seen here that BP-Priming does better than the Baseline for all but the greatest distance and largest angle, with its relative performance being particularly poor at 85°. False-Positive rates do not vary significantly, and at greater distances are typically the result of translation error placing the outlet frame just outside the acceptance region.

Table 6.3 True-Positive Rate by Scenario and Angle

True-Positive Rate by Scenario and Angle					
	easy	decoy	low-contrast	occlusion	challenge
Baseline 0°	97.5%	82.5%	0.0%	7.5%	0.0%
Baseline 45°	100.0%	100.0%	0.0%	25.0%	0.0%
Baseline 85°	92.5%	90.0%	0.0%	55.0%	2.5%
BP-Priming 0°	100.0%	92.5%	55.0%	82.5%	0.0%
BP-Priming 45°	97.5%	90.0%	50.0%	85.0%	40.0%
BP-Priming 85°	45.0%	52.0%	12.5%	12.5%	2.5%

Table 6.4 False-Positive Rate by Scenario and Angle

False-Positive Rate by Scenario and Angle					
	easy	decoy	low-contrast	occlusion	challenge
Baseline 0°	0.0%	0.0%	0.0%	0.0%	0.0%
Baseline 45°	0.0%	0.0%	0.0%	0.0%	0.0%
Baseline 85°	2.5%	7.5%	7.5%	2.5%	0.0%
BP-Priming 0°	0.0%	5.0%	5.0%	2.5%	0.0%
BP-Priming 45°	2.5%	0.0%	2.5%	7.5%	2.5%
BP-Priming 85°	0.0%	0.0%	0.0%	2.5%	2.5%

As expected, the ability of BP-Priming to use as few as two oriented edges allows it to significantly outperform the Baseline on the low-contrast, occlusion, and challenge scenarios. Once again we see that the detection accuracy of BP-Priming is poor for 85 degrees, particularly for the three scenarios where only two outlet edges are visible. This is likely due to the high selectivity of the second wave fitting algorithm with regard to angle. The top and bottom oriented edges are so short at 85-degrees that slight perturbations put the angle beyond tolerance, thus preventing a fit.

Surprisingly, the breakdown reveals that Priming was significantly outperformed by the Baseline on the occlusion scenario at 85 degrees. This was unexpected since the Baseline requires all four edges to be visible. However, a look at the annotated images shows that the steep angle nullifies the effect of the occlusion tape, allowing the contour to frequently form a quadrilateral.

Another oddity is how Priming achieved a 40% detection rate for the challenge scenario, but only at 45-degrees. This is apparently due to shadowing at that angle allowing the two unoccluded edges to be reliably detected despite low contrast. Note that this performance is only 10% worse than low-contrast at the same angle.

Table 6.5 Translation Error by Scenario and Angle

Average Translation Error by Scenario and Angle					
	easy	decoy	low-contrast	occlusion	challenge
Baseline 0°	0.034m	0.040m	NA	0.057m	NA
Baseline 45°	0.023m	0.016m	NA	0.062m	NA
Baseline 85°	0.020m	0.032m	NA	0.021m	0.051m
BP-Priming 0°	0.026m	0.028m	0.081m	0.056m	NA
BP-Priming 45°	0.038m	0.037m	0.064m	0.070m	0.079m
BP-Priming 85°	0.030m	0.030m	0.052m	0.010m	0.018m

Table 6.6 Rotation Error by Scenario and Angle

Average Rotation Error by Scenario and Angle					
	easy	decoy	low-contrast	occlusion	challenge
Baseline 0°	10.83°	10.31°	NA	22.75°	NA
Baseline 45°	5.67°	2.46°	NA	13.92°	NA
Baseline 85°	5.16°	5.04°	NA	3.15°	15.99°
BP-Priming 0°	7.05°	5.73°	7.56°	10.87°	NA
BP-Priming 45°	6.88°	7.56°	2.92°	6.59°	15.18°
BP-Priming 85°	3.21°	2.81°	1.49°	3.15°	13.64°

Both translation and rotation error statistics are difficult to interpret since scenario-angle combinations with a poor false-positive rate use fewer samples to calculate their averages, thus making it easier for outliers to skew results. However, it does seem clear that translation error is higher for the low-contrast, occlusion, and challenge scenarios. This is likely the result of having fewer visible edges to base the calculation on.

Rotation error for bp-priming is further influenced by the density of back-projections. For example, if rotation angles were sampled at an interval of 10 degrees at each translational position, then the rotation error would average 5 degrees due simply to the coarseness of sampling. The unusually

high rotation error rates for the baseline at 0 degrees can be attributed to its use of all visible points in a quadrilateral contour. Bp-priming is immune to this problem because it ignores features that don't fit the model.

Table 6.7 Accuracy of Back-Projective Priming

TP and FP Rates by Scenario and Angle for BP-Priming					
	easy	decoy	low-contrast	occlusion	challenge
0° TP	100.0%	92.5%	55.0%	82.5%	0.0%
0° FP	0.0%	5.0%	5.0%	2.5%	0.0%
45° TP	97.5%	90.0%	50.0%	85.0%	40.0%
45° FP	2.5%	0.0%	2.5%	7.5%	2.5%
85° TP	45.0%	52.5%	12.5%	12.5%	2.5%
85° FP	0.0%	0.0%	0.0%	2.5%	2.5%

A side-by-side comparison of True and False Positives rates for each angle and scenario shows perfect detection accuracy when viewing a high-contrast unoccluded outlet directly from the front, which is fortunately also the most realistic real-world scenario.

CHAPTER 7: CONCLUSIONS AND FUTURE WORK

This thesis demonstrates how constraints can be used to prime lower-level pattern matching modules to enhance model-based object recognition, with the specific application of enabling a robot to detect electrical outlets despite various forms of visual obfuscation that impede a purely bottom-up approach. Our specific approach, Back-Projective Priming, enables the use of features traditionally considered unsuitable for object recognition such as oriented edges. It is also capable of detecting and estimating the pose of an electrical outlet with as few as two feature points, compared to the four required by most other systems.

One limitation of the current system is the unacceptably high false-positive rate. It is true that most of these false-positives only occur in one of several consecutive frames and so could be ignored by the system on the basis of their frequency alone. Two bottom-up methods of reducing false-positives include the incorporation of additional feature types that allow modeling of the socket holes, or a neural network to post-validate the region of interest around a candidate detection. The use of additional feature types may require a more advanced rendering engine such as OpenGL to produce more realistic back-projections. The top-down fix would be to acquire more constraints by detecting walls and other important landmarks, thus shrinking the pose-space sufficiently to likely eliminate all false-positives encountered so far. To this end, constraint programming may be useful in generating the pose-space samples.

Another direction to go would be to improve the oriented-edge detector as it’s limitations underlie the poor detection rate on the low-contrast and challenge scenarios. Indeed, the illusory contours of Kanizsa's Triangle (Figure A.1) suggest that the line-detector itself should be able to take top-down constraints to improve its sensitivity to the lines we’re looking for. To achieve sufficient

cohesion and learning ability will likely require moving towards a connectionist architecture, possibly deep convolutional neural networks [9] with feedback.

A more feasible next step would be to see how well the system performs at detecting light switches, as they have the same model but different constraints. To improve performance, rewriting key sections of the algorithm in C++ could result in an order of magnitude speed-up. Multi-processing and GPU acceleration are also possibilities.

REFERENCES

[1] Ryan Dellana. Back-Projective Priming: Toward Efficient 3d Model-based Object Recognition via Preemptive Top-down Constraints. In *MAICS*, pages 91-96, 2015.

[2] Wim Meeussen, Melonee Wise, Stuart Glaser, Sachin Chitta, Conor McGann, Patrick Mihelich, Eitan Marder-Eppstein et al. Autonomous door opening and plugging in with a personal robot. In IEEE International Conference on Robotics and Automation (ICRA), pages 729-736, 2010.

[3] Victor Eruhimov, and Wim Meeussen. Outlet detection and pose estimation for robot continuous operation. In IEEE/RSJ International Conference on Intelligent Robots and Systems, pages 2941-2946, 2011.

[4] Long Quan, and Zhongdan Lan. Linear n-point camera pose determination. IEEE Transactions on pattern analysis and machine intelligence 21, no. 8, pages 774-780, 1999.

[5] Yehezkel Lamdan, and Haim J. Wolfson. Geometric Hashing: A General and Efficient Model-Based Recognition Scheme. In Proceedings from Second International Conference on Computer Vision. Pages 238-249, 1988.

[6] Eduardo R. Torres-Jara, A self-feeding robot. PhD diss., Massachusetts Institute of Technology, 2002.

[7] Luis Bustamante, and Jason Gu. Localization of electrical outlet for a mobile robot using visual servoing. In Canadian Conference on Electrical and Computer Engineering, pages 1211-1214, 2007.

[8] Foote, Tully. tf: The transform library. In IEEE International Conference on Technologies for Practical Robot Applications (TePRA), pages 1-6, 2013.

[9] Charles F. Cadieu, Ha Hong, Daniel LK Yamins, Nicolas Pinto, Diego Ardila, Ethan A. Solomon, Najib J. Majaj, and James J. DiCarlo. Deep neural networks rival the representation of primate IT cortex for core visual object recognition. In PLoS Computational Biology 10, no. 12, 2014.

[10] Micah M. Murray, Glenn R. Wylie, Beth A. Higgins, Daniel C. Javitt, Charles E. Schroeder, and John J. Foxe. The spatiotemporal dynamics of illusory contour processing: combined high-density electrical mapping, source analysis, and functional magnetic resonance imaging. In The Journal of Neuroscience 22, no. 12, pages 5055-5073, 2002.

[11] Charles D. Gilbert, and Wu Li. Top-down influences on visual processing. In Nature Reviews Neuroscience 14, no. 5, pages 350-363, 2013.

[12] Mark E. Auckland, Kyle R. Cave, and Nick Donnelly. Nontarget objects can influence perceptual processes during object recognition. In Psychonomic bulletin & review 14.2, pages 332-337, 2007.

[13] Jeff Hawkins, and Sandra Blakeslee. On Intelligence. Macmillan, 2007.

APPENDIX A: DRAWING FROM COGNITIVE SCIENCE

Notions of "pattern" in Artificial Intelligence are closely related to the concept of "schema" (plural schemata) from psychology. Schemata are described as patterns of thought or behavior that organize categories of information and the relationships among them, used both in perception and the consolidation of memory. For example, a person may have a schema for "chair", so that when they encode a memory of an office, rather than storing a mental "photograph" of each chair, they store a bunch of "pointers" to a single chair schema. A particular memory of being in said office, an episodic memory trace, could use a network of nested schema, with "office" having references to "cubicle", which may use "chair", and so on.

Perception takes limited sensory data and uses schemata to help remove noise and fill in the blanks with what's probable given the context. We can describe this more formally as hierarchical pattern recognition, a system where simple patterns are chunked into higher-level patterns in the structure of a directed acyclic graph. In a hierarchy of pattern recognition modules, "bottom-up" is where simple patterns in the raw data are found first and subsequently passed to higher-level modules in an upward cascade, where the output from the modules in each layer becomes input for the modules in the next. "Top-down" is a cascade proceeding through the same modules in the opposite direction, where a pattern is generated rather than detected. Top-down has been implicated in psychological phenomena including hallucination, imagination, recall, attention, and priming. One of the broader ideas explored in this thesis is that perception can be achieved via the interaction of bottom-up and top-down processes.

This concept is certainly not new, being especially prominent in the unified "whole is greater than sum of parts" view of perception put forth in Gestalt psychology. Take for example illusory contours such as in Kanizsa's Triangle (Figure A.1a). Since most would consider line detection to necessarily precede triangle detection, illusory contours suggest that higher level pattern recognizers exert top-down influence on lower level modules. [10] believe this effect is due to feedback modulation of areas V2 and V1 from "higher-tier lateral-occipital areas, where illusory contour sensitivity first occurs" (Figure A.1b). Indeed, there is a growing body of evidence and general consensus among neuroscientists for the importance of top-down feedback connections in the human visual system [11]. There is also experimental evidence showing that people recognize objects with greater speed and accuracy when they occur within the expected context [12]. There have even been machine learning algorithms directly inspired by the wiring diagram of the cerebral cortex, for instance, Jeff Hawkins Hierarchical Temporal Memory [13].

This thesis specifically concerns "priming", which can be understood as the use of constraints from higher modules to bias lower modules *before* bottom-up processing takes place. These "constraints" are the top-down "expectations" or "context." Depending on how our modules represent information, when lower modules are biased by higher modules, either their sensitivity is altered or their search space is explicitly constrained. In general, priming results in greater sensitivity to the patterns primed for and lower sensitivity to non-primed or negatively-primed patterns. The benefits of this can include resistance to noise, improved efficiency, and the ability to detect positively-primed patterns despite missing data. However, priming can be counter-productive if higher-level modules are wrong, resulting in false-positives for positively-primed patterns ("confirmation-bias/hallucinations") and false-negatives for negatively-primed patterns ("distraction"). In the most general sense, this thesis states that priming is useful in scenarios where context is plentiful and robust. In terms of object

recognition, if we know the structure of what we're looking for, and where it's likely to be given the current context, we can make a reasonably accurate guess about the state of the world that produced the image, even if the image is of low quality.

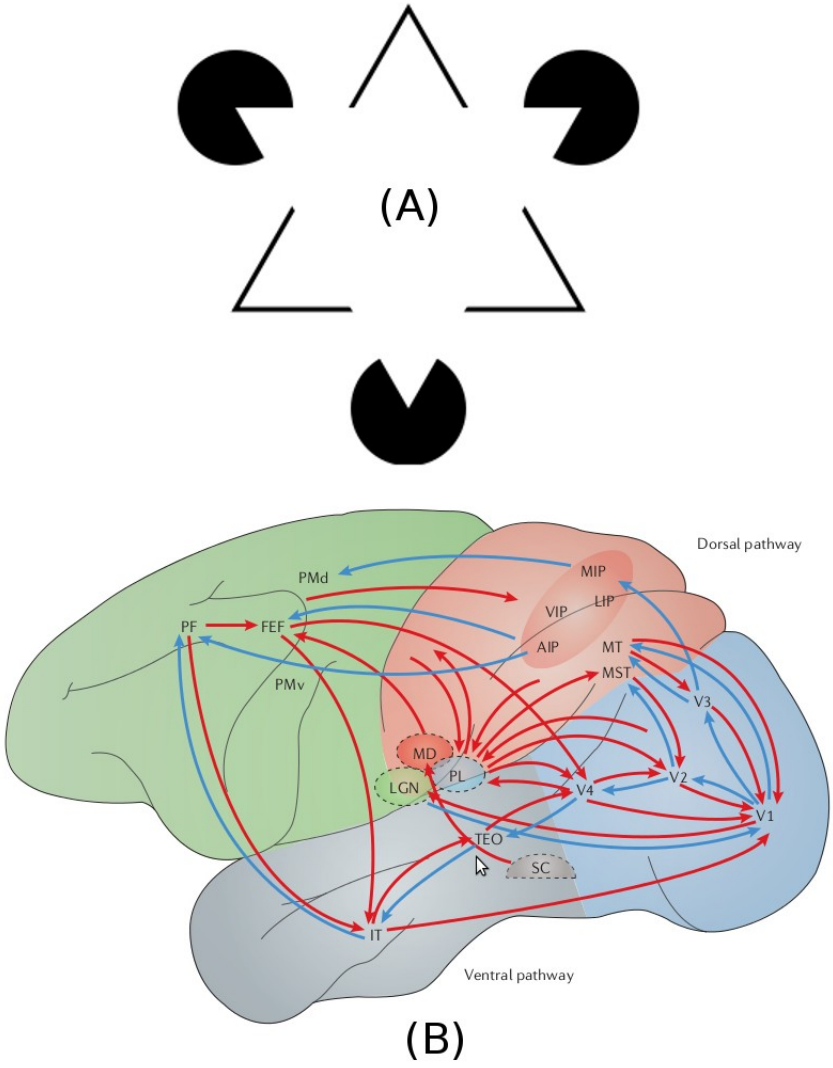

Figure A.1 Kanizsa's Triangle (a) and Feedback Pathways (b)