Intelligent Monitoring of Assembly Operations

Peter Anderson-Sprecher

CMU-RI-TR-02-03

Submitted in partial fulfillment of the requirements for the degree of Masters in Robotics

The Robotics Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

June 2011

Thesis Committee: Dr. Reid Simmons (Chair), Dr. Paul Rybski, Brian Becker

Contents

Abstract

To ensure safety, state-of-the-art robotic assembly environments typically require separation of assembly robots from human assembly workers. This separation has the disadvantage of requiring significant space and cost to install, and more importantly, it forces humans and robots to work in isolation, leaving little opportunity for cooperation.

The Intelligent Monitoring of Assembly Operations (IMAO) project aims to develop an accurate and precise safety monitoring system for industrial assembly workcells, one which will allow closer interaction of human and robotic workers than is currently possible while still maintaining a safe environment. By enabling humans and robots to work together, we can open a whole new array of cooperative assembly strategies that use the strengths of both human and robotic workers to maximum effect.

In this work, we first present the IMAO system architecture and key engineering accomplishments. Our proposed safety system uses multiple 3D range cameras to detect people, compares their locations against robot locations and possible future trajectories, and uses the resulting data to predict and prevent any possible collisions.

Secondly, we present in detail several research challenges that have been addressed during the course of this work. One of these is a new method in accessibility-based background subtraction for evidence grids, and the other is a fast means of computing reachability bounds within manipulator workspaces based on joint-level motion constraints.

Finally, we present results of a preliminary system evaluation using a small test workcell equipped with four sensors and a seven degree-of-freedom manipulator. Initial tests indicate that IMAO is effective for detecting people and preventing collisions in assembly environments, even when people are in close proximity to robotic manipulators. Further testing and evaluation is still ongoing to improve reliability and establish robustness for operation in complex real-world environments.

Chapter 1

Introduction

Safety is an important challenge in robotic industrial assembly. Assembly robots are large, move quickly, carry heavy or hazardous parts, and can be fatal if they collide with an unaware worker.

State-of-the art industrial assembly environments typically rely on the physical separation of assembly robots and human workers to ensure safety. Figure 1.1 is an example of such an environment, which uses physical barriers to keep people out of the robot's work area during operation. Other environments may use light curtains and other safety sensors to stop all robotic operations in the event that someone inadvertently enters an active assembly area. While these measures do ensure safety, they require a large amount of space to set up properly and limit the kinds of assembly tasks that can be performed.

The goal of the Intelligent Monitoring of Assembly Operations project (IMAO) is to use 3D sensing and robot motion modeling to improve upon the precision offered by current safety systems, thereby enabling human and robotic assembly workers to work in close proximity without sacrificing safety. Once people can work together with robots rather than separately, a wide range of new assembly techniques will become possible that allow both to do the tasks they are best at. People can rely on robots to do the heavy lifting while they inspect for quality and connect small parts, for example a robot could pick up and move a heavy vehicle chassis while a worker fastens wires and small bolts. Additionally, such a combined setup will allow workcells to be constructed in spaces much smaller than currently possible, and at lower cost.

There are several existing active research areas aimed at improving safety of industrial robots. Some of this is in force-compliant and soft manipulators, trying to minimize injury in the event that there is a collision (e.g., [19, 14]). There are also efforts within industry to improve robot controllers so that they can take advantage of more fine-grained safety regions and allow closer human-robot cooperation [1, 6]. We believe that our approach will work well in conjunction with these methods to create highly safe, flexible, and efficient assembly systems.

1.1 System architecture

The foremost goal of our safety system is to prevent robots from colliding with people or any other unexpected objects in the environment. As such, we need to know two things: first, the location of any people or other foreign objects, and second, the location of all robots. Also, we must detect potential collisions far enough in advance to prevent them, so in addition to gathering position data we must model what future motion is possible for both people and robots. To capture this information, IMAO operates by detecting and modeling two particular regions in space:

1. *Safety zones* encompass and provide a safety margin around people and other unexpected objects in the

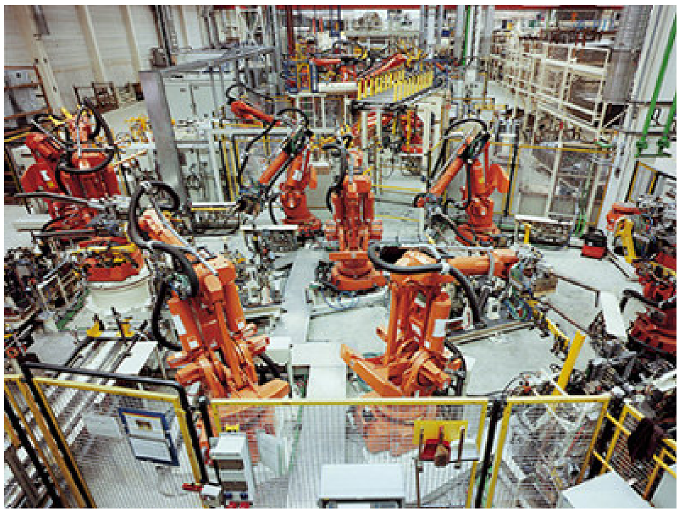


Figure 1.1: State-of-the-art industrial assembly workcell. Robots are fenced off from human workers to ensure safety. Image source: [1].

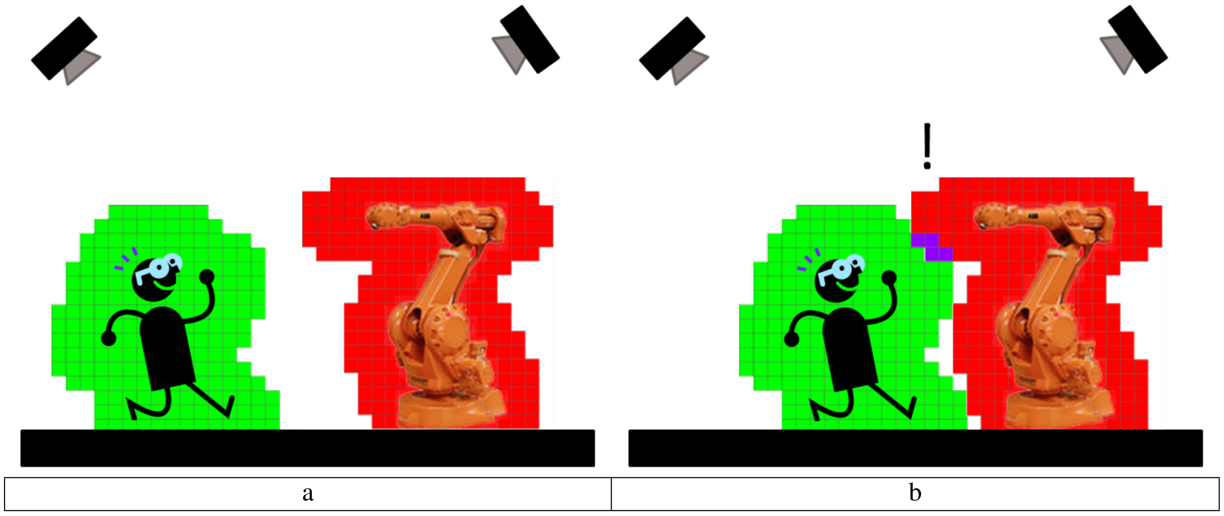


Figure 1.2: Safety and danger zones. As long as the zones are disjoint (a) the system is safe any may operate normally. Once they intersect (b) the system is no longer safe and robots must be halted to prevent a possible collision.

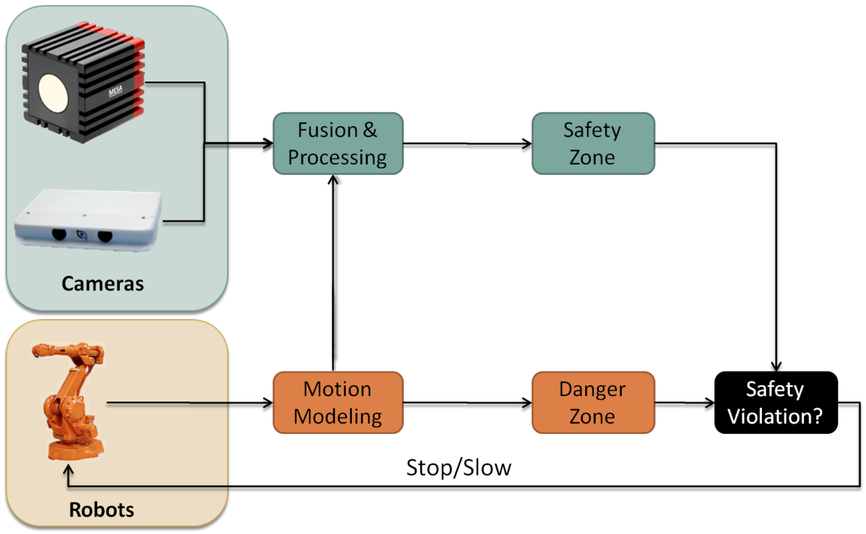


Figure 1.3: IMAO system architecture. Cameras shown are Swissranger SR400 flash lidar (top) and TYZX G3 EVS stereo camera (bottom).

environment. Detection incorporates readings from multiple 3D cameras in an evidence grid representation of the workcell to determine occupancy.

2. *Danger zones* encompass and provide a safety margin around robot manipulators and their payloads. These may be generated by known robot models and controller information.

Accordingly, IMAO is divided into two pipelines, one that calculates danger zones based on robot controller state information, and one that calculates safety zones based on 3D range sensor data. Safety zones do depend on robot state for purposes of background subtraction, but the required information is available early in the danger zone pipeline and does not constitute a performance bottleneck. The software architecture is shown in Figure 1.3.

Figure 1.2 illustrates safety and danger zones in a sample workcell. As long as all safety and danger zones are disjoint, the system is deemed to be safe and operation can continue as normal. Once they intersect, the system is no longer safe and all robots must halt. In practice, bordering the danger zone there is also a *warning zone*, which is wider and can allow the robot to slow or take corrective action before it is necessary to issue an emergency stop.

1.2 Timing and safety stops

How large these safety and danger zones need to be depends on the overall latency of the detection system as well as the time required to complete a safety stop. To reflect this, during zone generation we use a time horizon t_{max} to designate how far in the future we need to model motion of robots and people. The only requirement for this time parameter is that it be sufficiently large to allow a safety violation to be detected and then a subsequent safety stop to be fully executed before a person and a manipulator can collide. However, we do not want t_{max} to be larger than it needs to be, as larger values directly affect the size of safety and danger zones and normal operation may be impeded due to unnecessary safety stops.

The minimum value of the time parameter t_{max} needed to ensure safe operation can be calculated from the performance characteristics as follows:

$$t_{max} \geq \max(t_{safety}, t_{danger}) + t_{check} + t_{stop} + t_{frame} \qquad (1.1)$$

where

- t_{safety} is the time required to produce a safety zone, starting from the time the sensors capture a frame of data.
- t_{danger} is the time required to produce a danger zone, starting from the time the robot state information is read.
- t_{check} is the time required to compare safety and danger zones and issue a safety stop if needed.
- t_{stop} is the time required for the robot to fully stop in the event of a violation.
- t_{frame} is the time between successive data frames, equivalent to 1/framerate.

In practice, t_{safety} is usually larger than t_{danger}, due principally to the length of the camera frame capturing process and evidence grid creation. In our test workcell, we have values of approximately $t_{safety} = 180ms$, $t_{danger} = 90ms$, $t_{check} = 10ms$, $t_{stop} = 20ms$, and $t_{frame} = 100ms$, giving a $t_{max}$of about $300ms$. Currently, processing must be distributed across several machines to achieve this level of performance. Further investments in hardware and/or code optimizations could decrease this time further.

1.3 Test workcell

All tests and images described in this work were gathered from the IMAO test workcell. The test workcell is a 4m x 4m x 2m cage with two Swissranger SR4000 flash LIDAR time-of-flight cameras and two Tyzx G3 EVS stereo cameras, positioned at the four corners of the cage. Our test manipulator is the 7 degree-of-freedom arm attached to Bullwinkle, from the TRESTLE project [12]. See Figure 1.4 for the setup of the test workcell. All robot and sensor simulation, as well as visualization of safety and danger zones, uses the OpenRAVE simulator [4]. All evidence grids, safety zones, and danger zones are represented using a voxel grid with a voxel size of 5cm.

Our choice to use a combination of flash LIDAR and stereo cameras was deliberate. No sensor is perfect, and the different sensors have different failure modes: flash LIDAR cameras have difficulty seeing specular and non-reflective objects, and stereo cameras have difficulty detecting uniform surfaces and repeating textures. If most parts of the workcell are observed by at least one sensor of each type, detection should be more robust to unfavorable circumstances.

In the rest of this work we describe the creation of safety zones and danger zones in detail, along with several research challenges that arose as part of it. We then conclude with a description of how these zones can be integrated to produce a full safety system, and show some preliminary evaluation results within our test workcell.

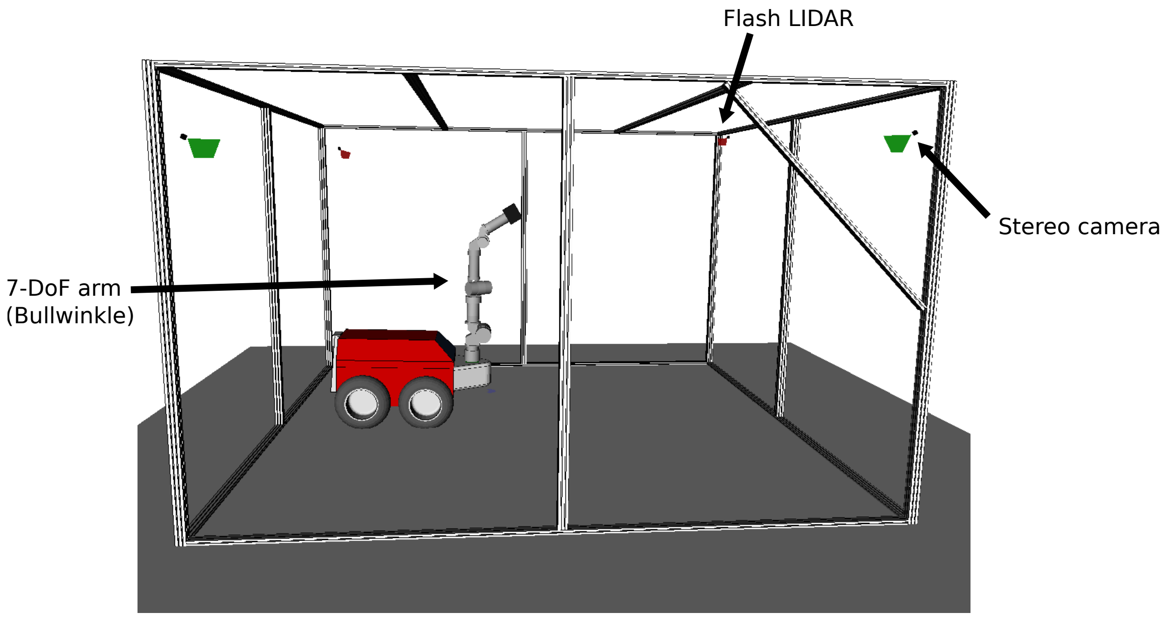


Figure 1.4: IMAO test workcell. Sensors are Tyzx G3 EVS stereo cameras (front, green) and Swissranger SR4000 flash LIDAR (back, red). Manipulator is a 7 degree-of-freedom arm attached to Bullwinkle (center).

Chapter 2

Safety Zones

A *safety zone* is defined as the region around a detected person or other foreground object into which it might move within some time horizon t_{max}. Safety zones are primarily concerned with detecting people, but a detected foreground object might also be something else brought in by people that is not part of the expected workcell background, such as tool chests or boxes. The *background* is comprised of two parts: first, unmoving objects that were present prior to system initialization, and second, active assembly robots and associated manipulator payloads. Additionally, safety zones must encompass any regions that cannot be sensed by the cameras, either because they are outside their field of view or because there are objects occluding them from view. This is essential, as we cannot guarantee that cameras will always be able to see all areas in a workcell, and any unobserved regions may contain people.

Objects are detected by constructing an evidence (occupancy) grid based on point clouds obtained from all cameras in the workcell. The evidence grid is a 3D voxel grid containing occupancy probabilities for each cell in the environment, as first introduced by Elfes and Moravec [5, 16].

Accordingly, the following steps are used to generate safety zones:

1. Gather point clouds from all sensors.
2. Create a single-sensor evidence grid from each point cloud.
3. Create a fused evidence grid by combining probabilities from each single-sensor grid.
4. Obtain foreground probabilities by subtracting background objects.
5. Threshold foreground probabilities and dilate to give safety zones.

The rest of this chapter describes our implementation of these steps in detail.

2.1 Evidence grid creation

The evidence grid creation process is illustrated in figure 2.1. Evidence grids are formed from point clouds using a probabilistic evidence model as described in Elfes [5]. All evidence grids start with a prior occupancy probability of 50%, then, for each sensor ray formed from the camera origin to an observed point, probabilities are adjusted by applying a range-specific evidence model. This model has the effect of raising occupancy probabilities near the sensed point, lowering them near the camera origin, and leaving them unchanged beyond the point. We did not use the theoretical evidence model described in [5], but rather the empirically derived model described in [17].

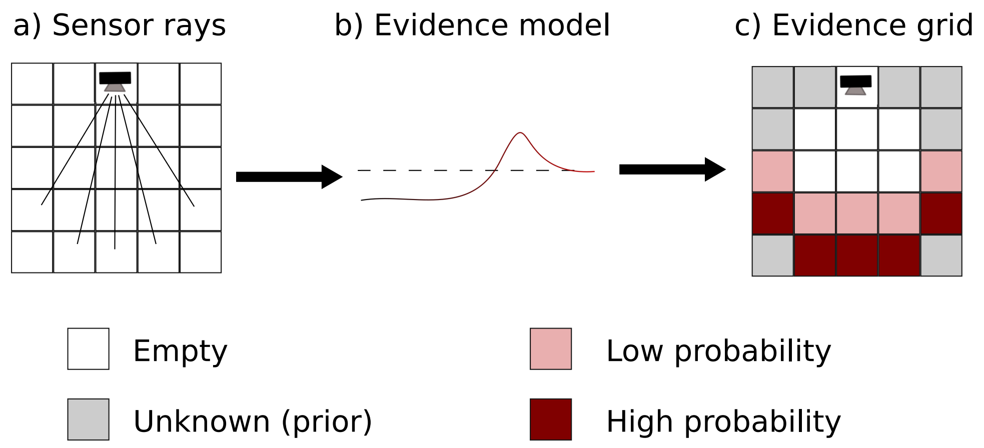


Figure 2.1: Evidence grid creation. a) Sensor rays are formed from the camera origin to observed points. b) Probabilistic evidence model used to assign occupancy probability based on proximity to observed point. c) Lines are rasterized into grid using the evidence model.

Because of the large number of points in the point clouds (For our tests, about 25000 points per frame per sensor in the test environment), the evidence grid rasterization process is the most computationally intensive part of safety zone creation. Therefore, it is critical to use a fast method for converting the point clouds into evidence grids.

To maximize speed, we pre-compute evidence models for every possible range reading, each of which takes the form of an evidence value vector ranging from the sensor origin to where the probability goes back to the prior following the peak. Vectors are calculated at a resolution equal to the voxel size (5cm). This allows us to use a modified version of Bresenham's line algorithm with a fourth dimension to track how far the ray has traveled along evidence profile. The slope along the evidence vector is the norm of the physical dimensions; if x is our traversal dimension we would increment the evidence vector iterator at each step by

$$\sqrt{1+(\frac{dy}{dx})^2+(\frac{dz}{dx})^2}. \tag{2.1}$$

Using this we can rasterize about 10^6 sensor rays per second in our test environment. However, at this time evidence grid creation is single-threaded and done in software, so speed could be significantly improved by computing in parallel, or better yet, offloading the rasterization to a GPU.

In order to make use of the sensor data, all evidence grids must be fused to give a composite occupancy probability for the entire workcell. To accomplish this we use the log-odds representation of probabilities as described by Elfes [5]. Then, if we assume sensor readings are independent from each other (while this is not generally true, in practice it is an acceptable assumption), fusion is reduced to addition of the evidence values from each input sensor. Evidence grid fusion is illustrated in Figure 2.2.

Note that when detecting moving objects, in order for the fused grid to be a good representation of reality data from all input grids must be gathered at exactly the same time. If frames are temporally offset, arms and other thin objects may vanish during fusion, as positive evidence from one sensor will be canceled out by negative evidence from another. This is obviously unacceptable from a safety perspective, so great care must be taken to ensure that all input frames are precisely synchronized.

The construction of accurate fused grids is also dependent on having a good extrinsic camera calibration. If cameras are not properly calibrated, they will produce conflicting readings which can cause false positives,

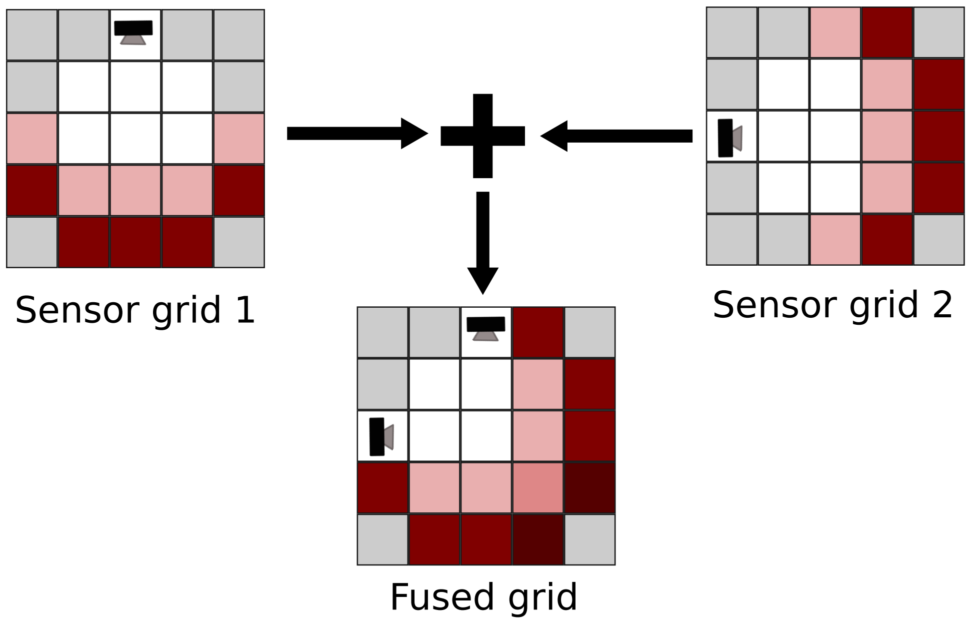


Figure 2.2: Evidence grid fusion. Evidence values from individual sensor grids (top) are added together to create a grid with combined probabilities (bottom). Because probabilities are stored in log-odds format, fusion is an addition operation.

or worse, cause true foreground regions to disappear.

2.2 Background subtraction

Next, we generate a foreground evidence grid by removing background objects. The goal is to have each cell contain the probability that a voxel is occupied by a foreground object, i.e. something that is not a) part of the static background, or b) a manipulator or manipulator payload.

Removal of manipulators and payloads must be performed separately from static background removal, as they violate the assumption that background is unmoving. Fortunately, the robot motion modeling used during danger zone generation already generates voxel representations of all manipulators and associated payloads. The only difference is that danger zones need motion projected out to some time in the future, whereas background subtraction only requires a robot's present location, which can be derived from the reachability grid by thresholding at $t = 0$. This does mean that background subtraction cannot start until robot motion modeling is complete, but in practice the evidence grid creation is significantly slower so it will not affect the overall system latency.

In practice, there are several complications that can cause problems when removing robots from a foreground grid. Robots may not be calibrated perfectly with respect to the cameras, which can result in parts of the robot remaining after subtraction. Also, cameras are inherently noisy and will detect parts of a robot that are not present in the predicted model. Even if robots are modeled and detected perfectly, they can cause occlusions nearby, which may be turned into safety zones. To account for these errors it is often necessary to expand the size of the subtracted robot model by some margin. In this case, to ensure safe operation it is then also necessary to add a similar margin to the robot's danger zone, as otherwise foreground objects could

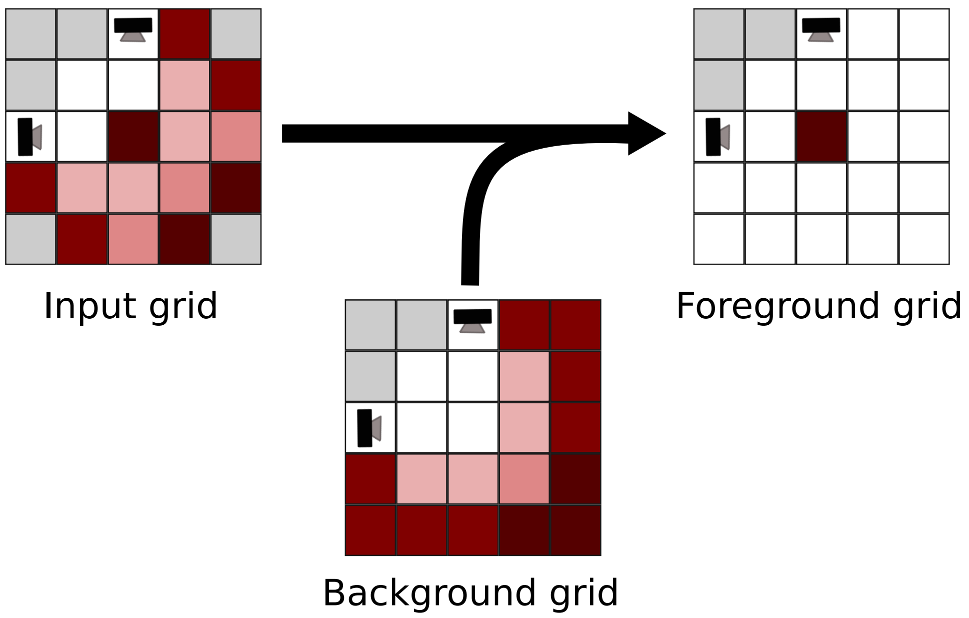


Figure 2.3: Evidence grid background subtraction. Background grid is used to remove background regions from fused occupancy grid, giving a grid containing foreground occupancy probabilities. Construction of background grid is non-trivial (see Chapter 3).

potentially hide within the padded region.

The other half of our background subtraction challenge, removing unmoving background objects, is similar to a standard background subtraction problem in computer vision. We choose to use a static background model, meaning that an operator is required to make sure that every person and anything that may move in the environment (except manipulators, as they are modeled separately) is out of the workcell prior to start-up. At that time the background grid is gathered, which is used by all future frames to remove background objects from the grid during run-time. Figure 2.3 shows an example background subtraction operation.

Building the background grid based on the observed data during initialization is non-trivial, however. While the straightforward approach would be to just use the fused background occupancy grid gathered at start up, there is a particular problem when dealing with large, solid objects such as boxes and work-benches. Since the sensed portion of the object is by necessity just the exterior, using this straightforward approach would subtract the exterior at run-time but leave the interior unchanged, at the prior probability. Since we have to treat unobserved regions as potentially occupied, this would cause many erroneous safety zones.

To fully remove such background objects, we developed and tested a novel approach that uses motion planning-inspired accessibility analysis to fill in closed-off regions. This algorithm, as well as experimental results demonstrating its efficacy, are detailed in Chapter 3.

2.3 Safety zone creation

The final safety zones can be built based on the foreground evidence grid. Safety zones are not probabilistic (a voxel is either in a safety zone or not), so the first step is to threshold the foreground grid at some probability to decide which regions must be given safety zones. This probability must be at or below the prior (i.e.,

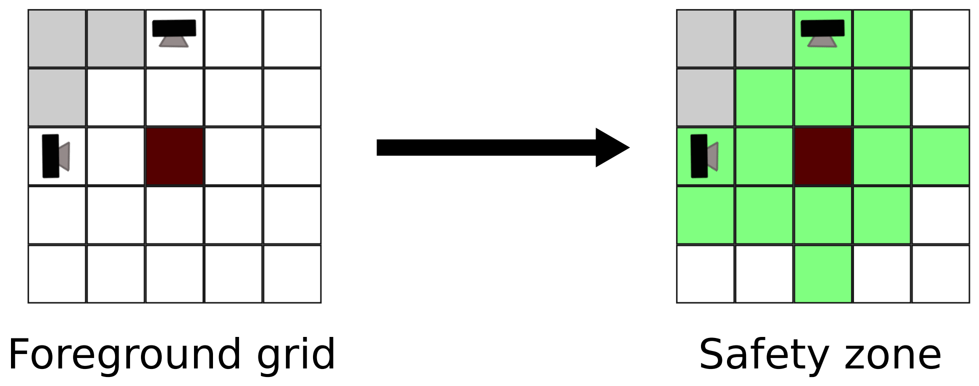


Figure 2.4: Safety zone creation. Foreground regions are dilated by a safety margin to produce safety zones.

$\leq 50\%$) to ensure that unobserved and occluded regions are included in safety zones. Also, as an added step we throw out any very small, isolated foreground regions using a connected components analysis, as these are most likely caused by sensor noise and are too small to contain a person.

The main challenge in safety zone calculation is how to model a zone such that it encompasses not only current occupancy, but all regions that a detected person might move into in a given amount of time. We chose to not track or predict movement, and make no assumptions other than that people will not move faster than some maximum speed. Thus, to create the danger zones we use morphological operations to dilate detected foreground regions by a radius $r \geq s_{max}t_{max} + l_v$, where s_{max} is the assumed maximum movement speed, t_{max} is the safety zone time horizon described in Section 1.2, and l_v is the length of a voxel, added to account for resolution errors. s_{max} is a human factors measure often used for safety systems; a common value is about $2m/s$. An example two-voxel safety zone dilation is shown in Figure 2.4.

Chapter 3

Background subtraction and accessibility analysis

Previous attempts at background subtraction within evidence grids either do so prior to sensor fusion or do so naively, simply ignoring any cells with a high background occupancy probability. A key weakness of these approaches is that they cannot reason about interiors of objects or other unobserved regions. Recognizing and removing solid object interiors is important for any application that must be able to differentiate between occupied and unknown space after background subtraction. Figure 3.1 shows observed foreground after background objects have been removed. It is important to make sure that objects such as the solid boxes shown are fully removed so they do not spawn spurious safety zones.

Most previous approaches to multi-sensor background subtraction have done so directly on sensor images using 2D vision techniques [3, 7, 10]. Background is subtracted from each individual sensor, then remaining foreground data are combined to determine occupancy of foreground objects.

Performing background subtraction directly at the evidence grid level confers several advantages. For instance, by simply discarding an individual reading when it fits a background model, we lose the ability to differentiate the region in-between the sensor and the sensed object, which is known to be empty, from the region beyond the reading, which is unobserved. Distinguishing between unknown and unoccupied regions is important for many applications such as ours, where the potential cost of an object occupying an unobserved region is high. Since evidence grids keep track of whether regions are occupied, unoccupied, or simply unknown, subtraction at the grid level can remove occupied areas while leaving empty regions alone.

In mobile robotics, Simultaneous Localization and Mapping (SLAM) is another application for which evidence grid-based background subtraction is advantageous. SLAM approaches implicitly generate a background map either through a static environment assumption (e.g., [18]) or through moving object detection (e.g., [25]). Background subtraction in these applications can give important information about unexplored areas, as well as highlighting newly observed objects.

One simple approach to background subtraction within evidence grids is to simply ignore any cells that are occupied, according to some background map. However, the regions subtracted by this method are limited to the exterior surface of objects, as object interiors cannot be directly observed. For large objects, interior regions will always have prior (unobserved) occupancy probabilities, which will remain unchanged during background subtraction. Again, this poses a problem for applications that must conservatively interpret unknown regions as potentially occupied.

To address the problem of interior region removal, we introduce the notion of *accessibility analysis,* a method of measuring and removing regions that may be assumed to be empty due to surrounding background objects. We define a region to be *accessible* if and only if there exists a path to it from some known open

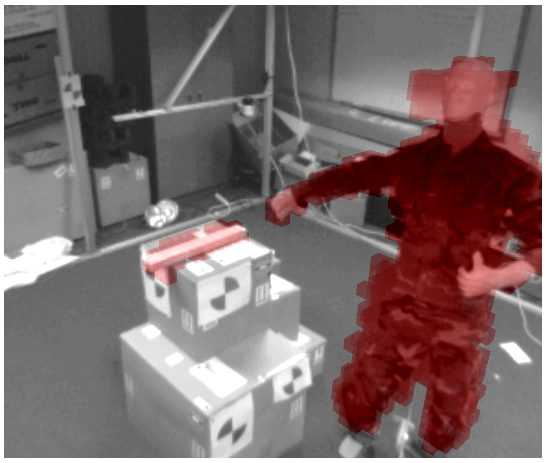


Figure 3.1: Background subtraction visualization. Red indicates sensed foreground regions.

region that does not traverse through any occupied region. However, note that, because of sensor error and gaps in coverage, there will often exist small cracks or occlusions in what is an otherwise solid object, leading to an overly generous estimate of accessibility. For this reason, we allow for an adjustable degree of tolerance to gaps by measuring accessibility with respect to an object of some given radius.

With this concept in mind, determining accessibility is similar to a standard robotic motion planning operation. This gives a simple algorithm for measuring accessibility:

1. Derive a configuration space by erosion of empty regions.
2. Find all locations reachable from some known accessible region within configuration space.
3. Revert to physical space by dilation of accessible (empty) areas.

Figure 3.2 illustrates these steps on a sample background map.

In this chapter, we present two methods for performing background subtraction and accessibility analysis within a fused evidence grid. The first is a natural extension of the simple threshold-based approach to include accessibility analysis. Second, we present a novel probabilistic algorithm that computes accessibility probabilities for all cells based on a min-cost path analysis. Finally, we test and compare both methods using simple background subtraction scenarios in our 4-sensor test workcell. Our results demonstrate that accessibility analysis is an effective method for interior region removal, and that probabilistic subtraction is particularly well-suited for detecting small foreground objects.

3.1 Prior Work

To date, there has been little research in background subtraction within the evidence grid framework. Work on background subtraction in multi-sensor systems has historically performed subtraction on single sensors prior to data fusion. For cases using monocular cameras with silhouette-based 3D projection, background subtraction has been performed using standard computer vision techniques [3, 7]. Other work has used statistical analysis at the point cloud level, such as in [10].

Background subtraction on fused data is implicit in approaches to Simultaneous Localization and Mapping (SLAM) as well as Structure from Motion (SfM) in non-static environments [15, 25, 26]. There has

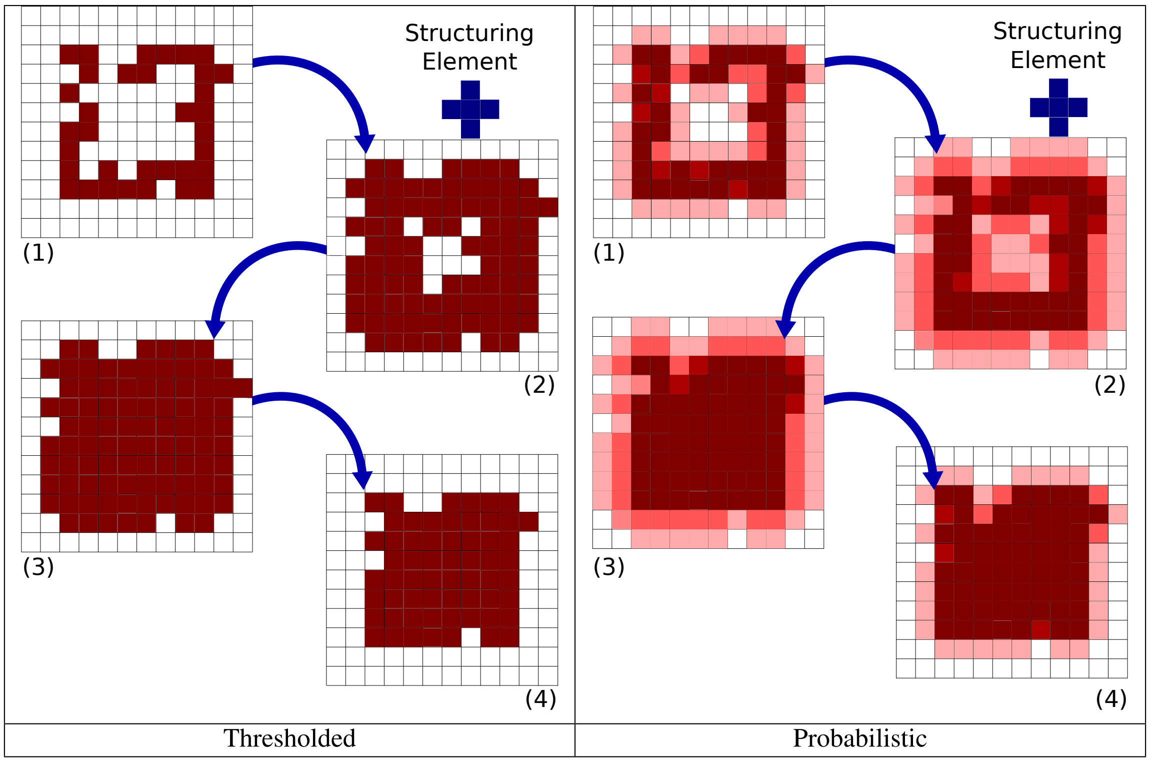


Figure 3.2: Accessibility analysis example, for thresholded (left) and probabilistic (right) methods ($r = 1$). (1) The initial background map; (2) Configuration space generated by erosion of empty regions; (3) Configuration space with inaccessible regions filled; (4) Final background map in metric space after dilation of empty cells.

also been research in object segmentation for fused data. Standard data clustering techniques may be used on either point clouds or evidence grids [13]. There are also studies particular to segmentation in point clouds [9] and in evidence grids [24]. However, while useful for finding certain foreground features, these methods do not directly address the problem of background subtraction and accessibility.

The next two sections describe our thresholded and probabilistic approaches to background subtraction.

3.2 Thresholded background subtraction

In the thresholded approach, we perform accessibility analysis based on a binary thresholded background map. Thresholding simplifies the problem, but has the disadvantage that once a cell is classified as background, it will always be removed even if the occupancy probability for the cell later increases. This method is illustrated in Figure 3.2 (thresholded).

The algorithm starts with a background evidence grid G_B, which contains for each cell the probability of occupancy by a background object. This grid can be gathered by building a standard fused evidence grid at a time when it is known that all foreground objects are out of the environment.

A cell c in the grid can have one of two states, occupied or empty. The probability of occupancy for a cell c according to an evidence grid G_B is denoted $P(c = occ; G_B)$.

Next, we threshold G_B according to a background probability threshold t_B, which produces the set of empty cells E.

$$E = \{c | P(c = occ; G_B) < t_B\} \tag{3.1}$$

In practice, this threshold can be set to any value above the prior probability; we chose $t_B = 0.51$.

To perform accessibility analysis, we use standard morphological operations with a structuring element R, which is defined as an object of uniform radius r. Choosing r is application dependent; it should be approximately the size of the smallest foreground object that we want to detect. Higher values will make the system more robust to sensor noise and gaps in coverage, but also may introduce false negatives.

To derive the configuration space E_R of R in E, we erode E:

$$E_R = E \ominus R \tag{3.2}$$

Now, we need to determine which regions in E_R are accessible. An accessible cell in configuration space is defined as any cell that can be reached starting from a cell that is known a priori to be accessible.

To compute accessibility, first a connected components algorithm is used to group the cells in E_R. Then, given a set of known accessible cells A_0, a connected group E_i is accessible if and only if it intersects with A_0, meaning that there is an unobstructed path from A_0 to E_i. This gives a set A_R of cells that are accessible in configuration space:

$$A_R = \{E_i | E_i \cap A_0 \neq \varnothing\}. \tag{3.3}$$

The method of selecting A_0 is application dependent. In our implementation, we have a grid with a bottom boundary on the floor, a top boundary on the ceiling, and open sides, so A_0 is defined to be all cells on the four sides of the grid. This choice is valid since any foreground entities will need to travel through one of those boundaries to enter the grid.

Finally, we obtain the set of all accessible cells A by dilation of A_R, reversing (3.2):

$$A = A_R \oplus R. \tag{3.4}$$

Now, we can use this accessibility map to filter probabilities at run-time by introducing a prior foreground occupancy probability $P(fore)$. This can be derived from the accessibility probability $P(acc)$:

$$P(fore) = \begin{array}{c} P(acc)P(fore|acc) + \\ P(\neg acc)P(fore|\neg acc). \end{array} \tag{3.5}$$

Naturally $P(fore|\neg acc) = 0$, and for our purposes we use $P(fore|acc) = 0.5$. This gives our final foreground prior

$$P(fore) = \frac{P(acc)}{2}. \tag{3.6}$$

Since we are dealing with thresholded cells, $c \in A \iff c = acc$, and we have

$$P(c = fore) = \begin{cases} c \in A & 0.5 \\ c \notin A & 0 \end{cases}. \tag{3.7}$$

Application of this prior will have the effect of reducing foreground occupancy probabilities to zero for inaccessible cells, and leaving accessible cells unchanged.

3.3 Probabilistic background subtraction

The probabilistic version is conceptually similar to the thresholded version, but instead of thresholding and producing a set of accessible cells A, we estimate for each cell its *probability* of being accessible, $P(c = acc; G)$. Thus, our goal is to create a new evidence grid G_A that contains the accessibility probability for each cell

$$G_A(c) = P(c = acc; G). \tag{3.8}$$

The meaning of a cell being accessible in this case is the same as before: a cell is accessible if and only if in the background grid there exists a path to it from some known accessible cell by an object of radius r. Our approach follows the same motion planning-based steps as in the thresholded method, but is generalized to the continuous domain.

This method is illustrated in Figure 3.2 (probabilistic).

3.3.1 Configuration space probabilities

First, we perform an analog of the erosion operator to generate configuration space probabilities. A cell c is in the configuration space E_R of our structuring element R if and only if the set of cells R_c it occupies when centered at c are empty:

$$R_c = emp \iff c \in E_R, \tag{3.9}$$

where $P(c = emp)$ is the probability that a cell is empty, and $P(c = emp) = 1 - P(c = occ)$.

We define R_c for an object of radius r as

$$R_c = \{c' \mid ||c' - c||_1 \leq r\}. \tag{3.10}$$

Now, we can write the configuration space probabilities:

$$P(R_c = emp) = P(\bigcap_{c' \in R_c} c' = emp). \tag{3.11}$$

Note that (3.11) is the probabilistic analog to (3.2). As is typical in occupancy grids, we assume that cell probabilities are independent, so this simplifies to

$$P(R_c = emp) = \prod_{c' \in R_c} (1 - P(c' = occ)). \tag{3.12}$$

For practical purposes we do not actually want to use the straight background probabilities in computing the configuration space. For large sizes of r, and for long accessibility paths, even relatively low values of $P(c = occ; G)$ will cause accessibility to approach zero rapidly. The problem is even more pronounced with unobserved cells, as they have a prior probability of 0.5. This is mainly a result of the cell independence assumption, and to mitigate it we set unobserved cells and cells with an occupancy probability lower than a fixed threshold t_E to 0, using a thresholding function $f(c, G)$.

$$f(c, G) = \begin{cases} P(c = occ; G) < t_E & 0 \\ c \text{ is unobserved} & 0 \\ \text{else} & P(c = occ; G) \end{cases} \tag{3.13}$$

Now we can define our configuration space grid G_{E_R} based on the background probability grid G_B, using (3.12). Also, from here on we choose to work in the negative log space for computational stability.

$$G_{E_R}(c) \quad = \quad -\sum_{c' \in R_c} \log(1 - f(c', G_B)) \tag{3.14}$$

3.3.2 Configuration space accessibility

Next, we must compute the probability that cells are accessible in configuration space, $P(R_c = acc)$. This is analogous to (3.3):

$$R_c = acc \iff c \in A_R.$$

A cell is accessible in configuration space if and only if there exists some path P of adjacent cells traveling from a known accessible configuration $R_0 \subset A_0$ to a configuration R_d such that $\forall c \in P, R_c \in E_R$. However, because of the infeasibility of combining all possible paths, we approximate this with the most probable path:

$$P(R_c = acc) \approx \max_{\{P | c \in P\}} P(\bigcap_{c' \in P} R_{c'} = emp). \tag{3.15}$$

In practice, this is not an unreasonable approximation. Most areas will consist of unoccupied space, where the path taken is irrelevant. Areas that do require entry into occupied space will typically have a single point of entry that gives maximum accessibility probability, with all less probable paths being insignificant in comparison.

The log probability of accessibility via a path P is then the conjunction of all probabilities along that path:

$$\begin{array}{rcl} \log P(R_d = acc; P) & = & \log P(R_{P_0}, \ldots, R_d) \\ & = & \sum_i \log P(R_{P_i} | R_{P_0}, \ldots, R_{P_{i-1}}) \end{array}. \tag{3.16}$$

We can now derive the cost of moving from a cell c_i to an adjacent cell c_j from the conditional probabilities if we assume $P(R_{P_i})$ is independent of $P(R_{P_0}), \ldots, P(R_{P_{i-2}})$ given $P(R_{P_{i-1}})$.

$$\begin{array}{rcl} \text{cost}(c_i, c_j) & = & -\log P(R_{c_j} = emp | R_{c_i} = emp) \\ & = & \sum_{c \in R_{c_j}, c \notin R_{c_i}} G_{E_R}(c) \end{array} \tag{3.17}$$

The conditional independence assumption here does introduce inaccuracy, in that any time there is a turn in the path there will be cells that are in $R_{P_{i-2}}$ but not in $R_{P_{i-1}}$. This has the effect of penalizing turns in the path; for an object radius r, there are r cells that are double counted in any turn. However, as R incorporates $\approx 2r^2$ cells (depending on grid dimensionality), this is usually a minor problem.

With these costs, we can use Dijkstra's algorithm to compute the most probable path from some R_0 to each R_c. This yields a configuration space accessibility grid G_{A_R} with values

$$G_{A_R}(c) = \min_P \sum_{i=1}^{n} \text{cost}(P_{i-1}, P_i). \tag{3.18}$$

3.3.3 Individual cell accessibility

Finally, we need to extract the individual accessibility probabilities $P(c = acc)$, analogous to (3.4) in that

$$c = acc \iff c \in A.$$

Since $R_c = acc \rightarrow \forall c' \in R_c, c' = acc$, we can write

$$P(c = acc) = P(\bigcup_{c' \in R_c} R_{c'} = acc). \tag{3.19}$$

Now for any two cells $c_i, c_j \in R_c$ where $P(R_{c_i} = acc) > P(R_{c_j} = acc)$, we assume that the path to R_{c_i} travels through R_{c_j}. This is a reasonable approximation, at least for relatively small values of r. Then $P(R_{c_i}|R_{c_j}) = 1$ and $P(R_{c_i} \cup R_{c_j})$ can be simplified to

$$\begin{aligned} P(R_{c_i} \cup R_{c_j}) &= P(R_{c_i}) + P(R_{c_j}) - P(R_{c_j})P(R_{c_i}|R_{c_j}) \\ &= P(R_{c_i}) \end{aligned} \tag{3.20}$$

Therefore, we can take the maximum probability, giving

$$\begin{aligned} P(c = acc) &= \max_{c' \in R_c}(P(R_{c'} = acc)) \\ &= \exp(-\min_{c \in R_c} G_{A_R}(c)). \end{aligned} \tag{3.21}$$

3.3.4 Foreground filtering

During run-time, foreground extraction uses accessibility probabilities in computing the prior $P(fore)$, as shown in (3.6). This prior may be used during fusion in accordance with standard practice. In our experiments we use the independent opinion pool method with probabilities stored in log-odds format as defined in [5], so foreground filtering consists of adding the prior to the current fused evidence grid.

3.4 Experimental methods

Our test environment consists of a 4m x 4m x 2m frame, with one 3D sensor mounted in each upper corner near the ceiling (Figure 3.3). Two sensors are Swissranger SR4000 flash LIDARs and the other two are TYZX G3 EVS stereo cameras. All senors are oriented towards the center of the environment.

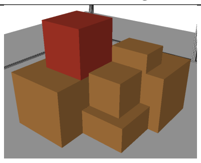
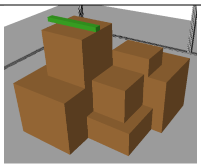
Sensor evidence models are generated using a parametrized sensor model similar to the one used by Moravec et al in [17]. Evidence models for each point are one-dimensional, as the sensor resolutions are dense enough to give sufficient coverage without conical evidence profiles, and single-line profiles significantly speed up processing. The model used for all tests has a peak occupancy probability of 0.56 at the reading and a trough at the sensor origin of 0.44. Rasterization into the evidence grid is performed using Bresenham's line algorithm.

All tests were performed using arrangements of boxes with known ground truth. Ground truth evidence grids were generated within the OpenRAVE simulator [4] using simulated sensors. All simulated sensors had the same calibration, field of view, and resolution as the real sensors. We used an ideal 0-1 sensor model for generating evidence grids in simulation, where the voxel containing the sensor reading is marked as occupied with $P(occ) = 1$ and all voxels leading up to it are marked as unoccupied with $P(occ) = 0$.

Tests used a 3D evidence grid containing an area with dimensions 1.7m x 1.5m x 1.5m in the center of the environment. The voxel size for all experiments was 5 cm. For accessibility analysis, all tests used an accessibility object radius $r = 10\,\text{cm}$. The thresholded method used a background threshold $t_b = 0.51$, and the probabilistic method an empty threshold $t_e = 0.1$.

To measure the performance of background subtraction, we compared the observed evidence grid with the simulated ideal grid using a standard precision-recall curve. We used a foreground threshold t_f to classify voxels in the observed grid as occupied or empty, then checked the classification against the ideal grid. Any voxels that were unknown in the ideal grid, e.g., voxels in the interiors of foreground objects, were ignored. Precision-recall curves were generated by measuring error rates across different values of t_f.

Table 3.1: Test Setups

Scenario	Foreground	CAD image
Large	One additional box	
Small	One 2x4 placed on top	

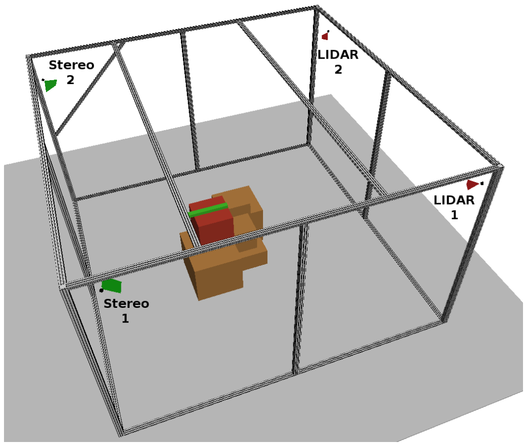
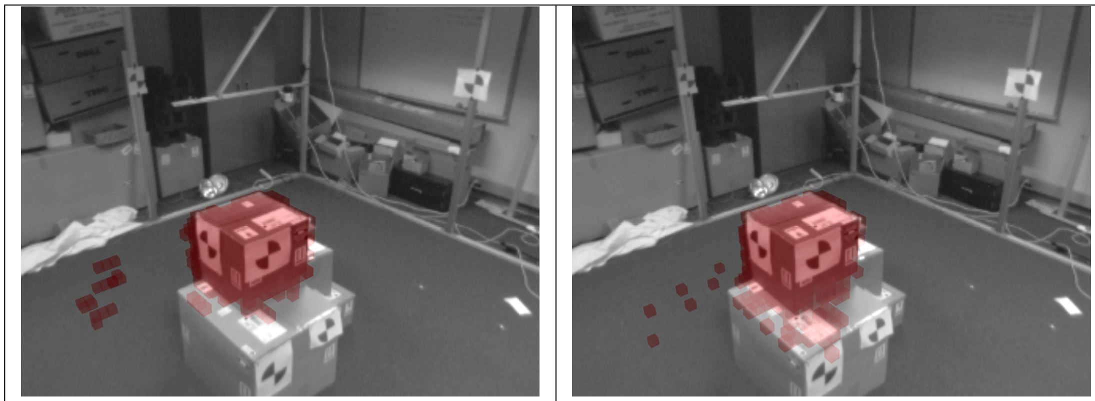


Figure 3.3: The test environment



a) Thresholded | b) Probabilistic

Figure 3.4: Foreground occupancy for large object test, using $t_f = 0.73$.

In addition to the probabilistic and thresholded background subtraction techniques presented in this work, we also tested baseline versions of these methods with no accessibility analysis. For the thresholded algorithm, we used the set of all empty cells E instead of the accessible cells A when computing the prior probability $P(fore)$ in (3.7). For the probabilistic algorithm, we substituted the probability that a cell is empty in the background grid $1 - P(occ)$ in place of the accessibility probability $P(acc)$ in (3.6).

All measurements were gathered in two test scenarios, with background and foreground objects as indicated in Table 3.1. One scenario tested detection of large objects, and the other detection of small objects.

3.5 Experimental results

Error rates for the large object test are shown in Figure 3.5 and for the small object test in Figure 3.6. Images showing foreground voxels, taken from the perspective of sensor Stereo 1 (see Figure 3.3), are shown in Figure 3.4 (large object), and in Figure 3.7 (small object).

In the large object test, results are good for both the thresholded and probabilistic methods. In the versions without accessibility analysis, results are significantly worse. This is to be expected, as the baseline methods will naturally classify interior regions of the background boxes as occupied for any threshold $t_f < 0.5$. In the small object test, performance is dramatically better with the probabilistic method, particularly under high recall.

Accessibility analysis does negatively affect performance under some circumstances, such as those seen in the low-recall portion of Figure 3.6. When detecting a flat surface such as the top of the background box, sensor noise will naturally produce scattered false positive voxels, making it uneven. During the accessibility computation, the morphological operations in effect perform a closing operation, which will exacerbate the problem by marking many of the gaps in-between the noise-generated voxels as inaccessible. However, our results indicate that this is not a major problem in practice.

Differences between the thresholded and probabilistic methods are most visible in boundary areas at the edges of background objects, where there is a moderate accessibility probability. In the thresholded method, these regions will likely be part of the background map and always removed, but in the probabilistic version the regions are still sensitive to changes in evidence. Since they are at a boundary between empty and occupied space, the probabilistic accessibility path does not need to go through occupied regions to reach them, and base subtracted foreground probability will be only slightly below 0.5.

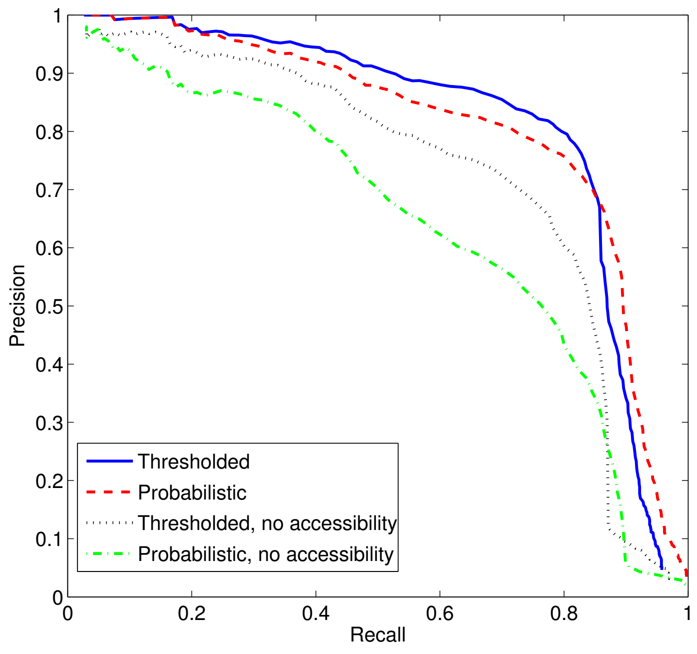


Figure 3.5: Precision-recall for large object test

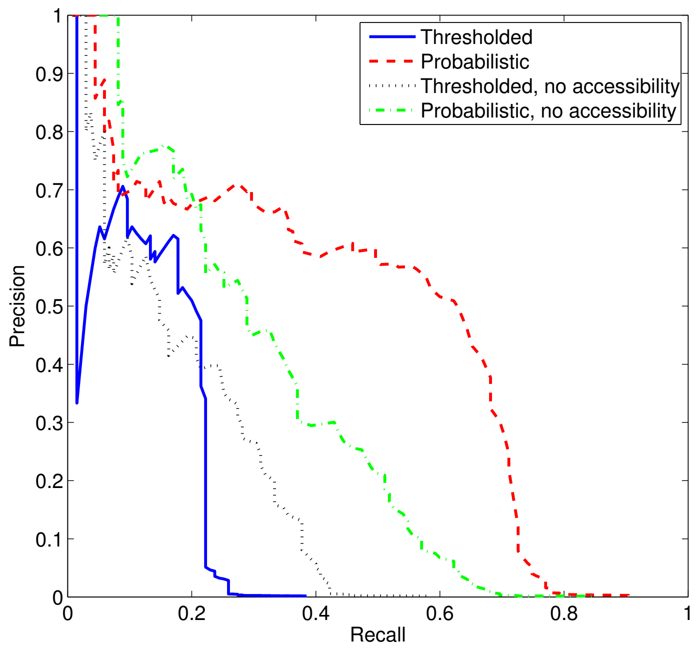


Figure 3.6: Precision-recall for small object test

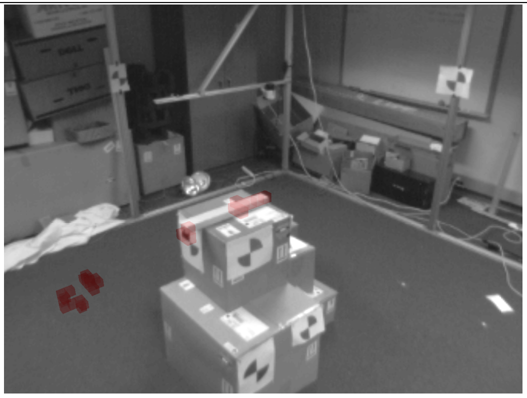
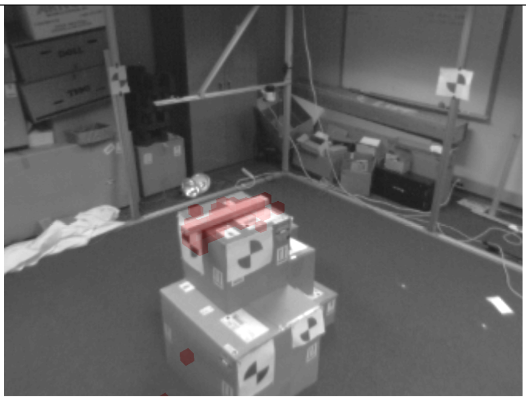
	
a) Thresholded	b) Probabilistic

Figure 3.7: Foreground occupancy for small object test, using $t_f = 0.73$.

In many situations this sensitivity is helpful. For small object detection, the increase in evidence may be a result of the addition of a thin foreground object touching a background object. This means that small objects may be detected even though they occupy voxels with high background occupancy probability (Figure 3.7a), something that is not possible using the thresholded version (Figure 3.7b). This accounts for the drastic difference in recall between thresholded and probabilistic subtraction in Figure 3.6.

However, this feature can also introduce unwanted noise, as there are circumstances where occupancy probabilities may increase even though no foreground object is added. In Figure 3.4b we see one example of this type of false positive, which results from occlusion of an area from one or more sensors by a foreground object. While intuitively objects should have higher occupancy probability if there are more sensors reading it, in reality this is often not the case, especially near object boundaries. Since the evidence ascribed to a voxel by a given sensor reading is a function of the distance between them, for voxels near a surface a sensor reading normal to the surface will give higher evidence than one at an oblique angle. The phantom voxels seen on top of the lower box in Figure 3.4b are a result of the occlusion of low-angle sensor readings. Note that the thresholded results in Figure 3.4a do not exhibit this behavior.

Adding the background prior during run-time to perform subtraction is fast; our 30,600 voxel grid took under 1 ms on a standard desktop computer. However, computing the background prior is more lengthy: the thresholded method requires approximately 20 ms, and the probabilistic version nearly 400 ms. However, because we use a static background, this computation only needs to be performed once at start-up and is not a significant consideration.

The video clip accompanying this work demonstrates the IMAO system in action with several people moving about the environment. The entire sensing, fusion, and visualization system runs at 10 hz, distributed across two computers.

3.6 Conclusions and future work

Our results indicate that background subtraction at the occupancy grid level is highly effective. Foreground filtering is reliable using either method presented, accurately detects objects down to single-voxel resolution, can be done in real-time, and is easily parallelizable.

For most applications that require coarse-grained detection of large objects, the thresholded version is sufficient. When greater sensitivity and detection of small objects is required, the probabilistic method performs significantly better, but is also more vulnerable to false positives.

In all cases, accessibility analysis is shown to be effective under the conditions of this study. Even with a small gap tolerance, interior regions are removed completely, allowing for much greater precision than would otherwise be possible. However, we expect the success of the interior region removal to be highly dependent on sensor coverage. Our test environments were designed such that all portions of the objects were visible to at least one sensor, but with less dense sensor coverage determining accessibility would be impossible, leaving interiors of background objects unknown.

While current results are successful within the parameters of this study, there are several areas where the approach could be extended to improve results and broaden the potential applications of this work.

First, this work was limited to the study of static backgrounds. Testing this approach with a dynamically changing background map would increase its versatility and allow it to be used in less controlled environments. The methods developed in this work are compatible with any standard background modeling technique, as accessibility analysis is performed after the background is gathered. However, using a dynamic background would require recomputing accessibility every time it changes, which will slow performance.

Also, the use of an evidence grid with a cell independence assumption requires many approximations, particularly in the probabilistic algorithm, which leads to decreased accuracy. The evidence grid independence assumption could be relaxed to a first-order Markov assumption, which may allow more accurate accessibility estimates, but likely at significant computational cost.

Chapter 4

Danger zones and robot motion modeling

A *danger zone* is a robot's analog to safety zones; they designate regions into which a robot may enter within some given period of time. However, there are few similarities in how they are generated. Unlike for people, where we have no information *a priori* about their location and must rely on sensors to detect them, for robots we know from the workcell setup exactly where they are, and from the controller exactly what their current position is. Similarly, while we can make no assumptions about the movement of people, robots have well-defined movement capabilities defined by robot geometry and how fast the controller may move joints.

Figure 4.1 shows our danger zone generation architecture. The process starts with joint state information provided by the controller, which is passed into a motion model that a) measures the region currently occupied by the robot, and b) measures how long it will take to reach adjacent areas. All regions reachable within the time horizon t_{max} are then classified as a danger zone, and a larger time threshold can be applied to produce a warning zone.

Choosing what motion model to use is non-trivial, and is the principal challenge in producing a danger zone. Figure 4.2 shows three possible motion models, based on varying degrees of knowledge about the robot.

In an ideal world, we would know the exact planned trajectory of a robot. In this case, making a danger zone is simply a matter of sweeping the robot along its planned trajectory for the next t seconds, as shown in Figure 4.2a. If the safety zone does not intersect with the planned trajectory, then the system is safe, and if it does, we can slow or alter course to avoid it. However, this is not always possible due to real-time adjustments in trajectories, or, more commonly, because an off-the-shelf controller may not provide a way to access full

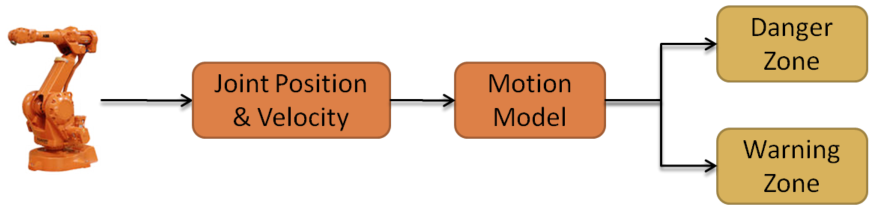


Figure 4.1: Danger zone generation. Controllers provide robot joint state information, which is passed through a motion model to produce danger and warning zones.

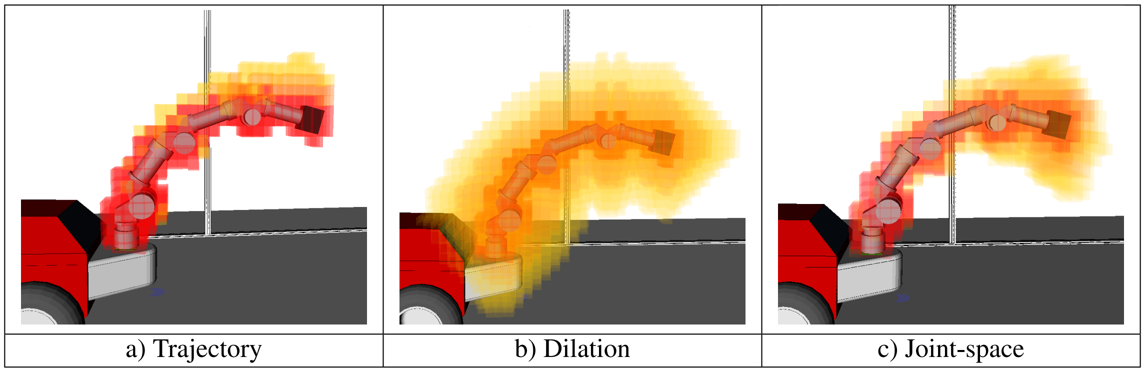


Figure 4.2: Possible robot motion models. a) Full trajectory model, b) Naive dilation model, c) Joint-space model.

trajectory information.

The other extreme would be to make no assumptions about what motion a robot may take and model it as we would a person, dilating by a radius dictated by some maximum speed, as in Figure 4.2b. However, robot controllers typically operate in terms of joint angles and do not provide a straightforward means to guarantee some maximum physical speed, particularly once you attach payloads. There are exceptions to this, such as ABB's SafeMove [6]. Even if one can determine some upper bound it will be very generous, as any maximum speed must correspond to a worst-case bound for the far end of the manipulator under full extension.

The ideal general-purpose model would be to specify our limits with respect to the joints of the robot manipulator. Controllers can easily constrain motion based on joint position, speed, and/or acceleration, yielding bounds that are specific to the geometry, pose, and velocity of any manipulator as in Figure 4.2c. However, this poses a problem in that modeling motion in joint space requires exploring a large number of possible poses that is exponential in the degrees of freedom of the robot.

To solve this problem, we propose a novel method for creating a *reachability grid*, a voxel-based representation of the minimum time needed for a manipulator to reach any physical location within its workspace. We use up to second-degree constraints on joint motion to model motion limits for each joint independently, followed by successive voxel approximations to map these limits on to the robot's physical workspace. This reachability information is general enough that we believe it has significant applications outside of the IMAO system. The remainder of this chapter explains this method in detail.

4.1 Manipulator motion bounding

Modeling the motion of robotic manipulators is an open problem in robotics. Globally, manipulators are constrained to their workspace, the set of all reachable regions, and within a workspace, some regions may be reached quickly, whereas others require a longer time. Knowing which regions are reachable in a short amount of time, which are more distant, and which are wholly unreachable is useful both to the robot, which must plan its movements according to its limitations, and to outside agents, which may need to take possible robot motions into account when planning their own movements. However, finding these regions is made difficult by the high-dimensional configuration space of most manipulators.

Our application, manipulator safety monitoring in unconstrained environments, requires a conservative estimation of regions into which a manipulator may enter in the near future in order to re-adjust or stop motion whenever humans might enter the robot's workspace. Other possible applications include trajectory

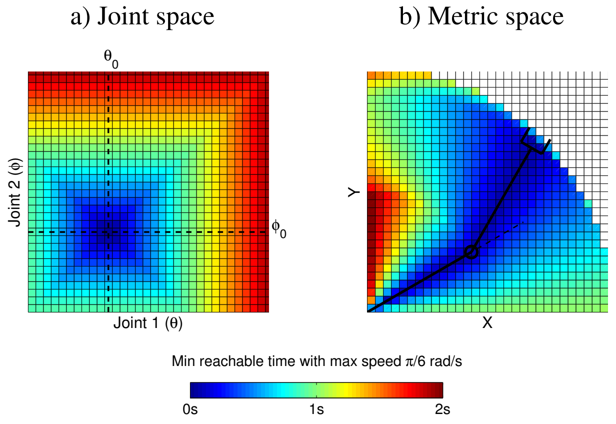


Figure 4.3: Reachability grid for 2 degree-of-freedom planar arm. a) Reachability in joint space. b) Reachability in metric space. Arm is drawn in current pose (θ_0, ϕ_0).

and task planning, as manipulators can use motion bounds to prioritize closer objectives over farther ones, or use the estimated time as a distance heuristic in motion planning.

How much time is needed for a robot to reach a given position depends on a set of constraints on its joint motion, usually enforced by the controller based on mechanical and power limitations. Constraints usually take the form of limited joint positions, velocities, and/or accelerations, though higher-order terms such as jerk limits are sometimes used. These constraints allow reasoning about how much time is required to reach a given joint pose given an initial position and velocity. Figure 4.3a shows an example map of reachability times in joint space, using constant joint velocity limits.

However, targets and obstacles are represented in physical (metric) space rather than joint space. This means that to have useful reachability information, we need to compute the robot's workspace and map minimum reachable times for corresponding joint values into it. Figure 4.3b shows the example robot's workspace with associated minimum reachable times. While this is straightforward for a simple robot such as this, real robots often have joint spaces that are too high-dimensional to exhaustively search all possible poses. For example, Figure 4.4 shows a reachability grid for a 7 degree-of-freedom (DoF) arm. While the physical space swept out by the arm is small, the corresponding 7-dimensional joint space is too large to work with directly.

We solve this problem by collapsing the swept volume of each link to an intermediate voxel approximation before considering it in the computation of previous links, which allows us to avoid searching the entire high-dimensional joint space and instead calculate bounds for each joint independently. The drawback is that it may produce optimistic times in some cases, as it cannot reason about self-collisions and obstacles. However, the general solution is intractable for high-DoF manipulators, and given that approximations are required, optimism is desirable for many applications, such as robot safety monitoring and A* planning heuristics.

Depending on the application, we may care about measuring regions that are reachable by any part of the manipulator (e.g., safety), or regions reachable by just the end effector (e.g., planning). Our experiments focus on the safety application, measuring reachability with respect to the entire manipulator, but results would apply equally to modeling the position of the end effector.

Our approach consists of two parts: first, we produce an invertible joint motion model that uses second-order constraints to determine time bounds for each individual joint. Second, we map these models into a robot's workspace using a recursive voxel-based workspace approximation. Since this is an approximate algorithm, we empirically measure the accuracy and computational speed of the approach on a simulated 4

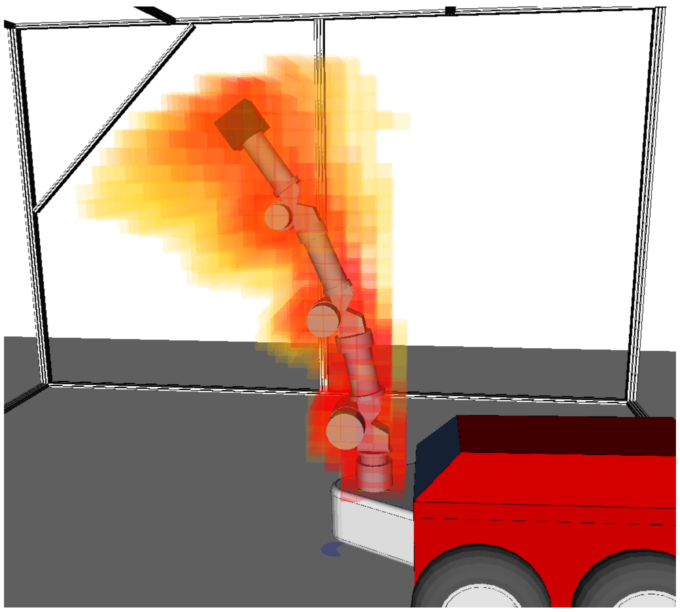


Figure 4.4: Reachability grid for a 7 degree-of-freedom manipulator. Yellow and orange regions require more time to reach than red regions.

DoF manipulator. Our results indicate that accurate reachability grids can be computed in real-time, even for robots with many degrees of freedom, and that approximation errors are minor and almost exclusively biased towards optimistic reachability estimates.

4.1.1 Prior work

Zacharias et al [22, 27] have used reachability grids to model possible robot motion and object manipulability. Their approach has the advantage of considering end effector orientation as well as position, but it cannot operate in real time, and assumes an analytical solution for the inverse kinematics of the manipulator. Others have used voxelization for estimation of swept volume, similar to how we compute the volume swept by a robot link [2].

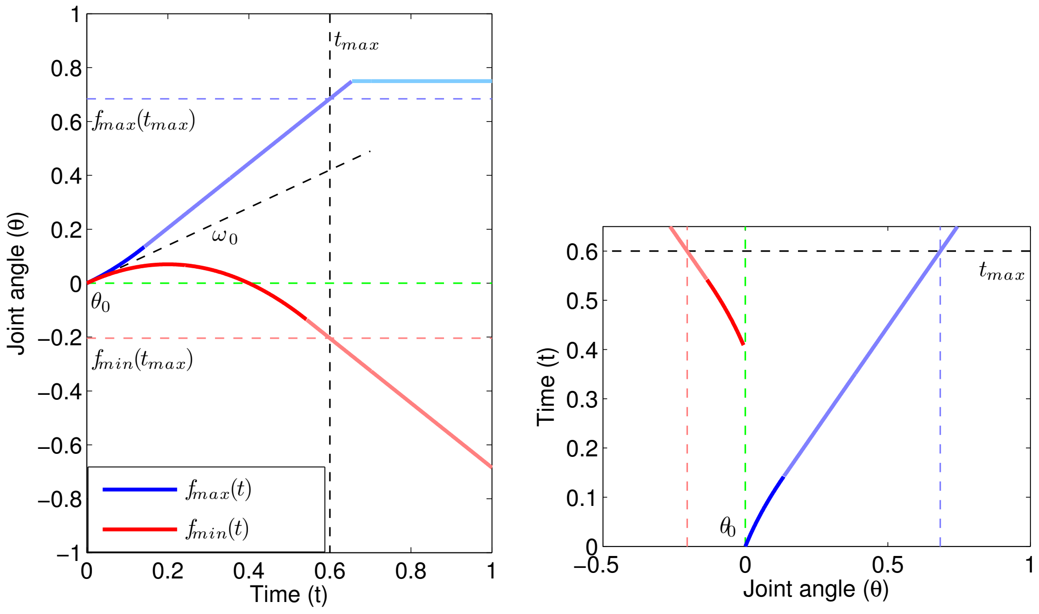
There has also been significant work in robot workspace estimation, a key component of our work. Several works deal with workspace solutions tailored to specific manipulators [8, 21]. Others have explored more general analytical approaches [11, 23]. Rastegar and Deravi studied the effect of joint motion constraints on workspaces [20].

However, none of these approaches operate in real time, and application of any workspace-based method would have to be run many times on varying sets of joint constraints to give a full reachability grid.

4.2 Modeling joint motion

In order to bound the time needed to reach any given joint position, we first require a set of *a priori* motion constraints for each joint. These constraints may consist of:

1. Constrained joint angle $\theta \in [\theta_{min}, \theta_{max}]$



a) Motion bounds	b) Inverse bounds ($f^{-1}(\theta)$)

Figure 4.5: a) Maximum motion bound $f_{max}(t)$ and minimum motion bound $f_{min}(t)$ (Eq 4.3) for a single manipulator joint. Joint has initial position θ_0 and initial angular velocity ω_0. Bounds increase quadratically until velocity limit is reached, at which point bounds increase linearly until the final position limit. b) Inverse motion bound function $f^{-1}(\theta)$. Gives minimum time t needed to reach any given angle θ.

2. Constrained joint velocity $\omega \in [\omega_{min}, \omega_{max}]$

3. Constrained joint acceleration $\alpha \in [\alpha_{min}, \alpha_{max}]$

Any subset of these constraints may be used; absent constraints are assumed to be infinite. In principle, we could also use other constraints, such as jerk limits, but these are not analytically invertible so it is preferable to avoid them, if possible.

Using the above constraints and a given initial joint position θ_0 and velocity ω_0, we can use piecewise functions to place bounds $[f_{min}(t),\ f_{max}(t)]$ on the joint motion as a function of t. Upper and lower trajectory bounds for a sample joint are shown in Figure 4.5a.

As long as the joint velocity ω is below the maximum ω_{max}, we assume maximum acceleration and the upper bound is governed by the trajectory quadratic

$$\theta_0 + \omega_0 t + \frac{1}{2}\alpha_{max} t^2. \tag{4.1}$$

This holds for all $t \leq t_1$ where t_1 is the time at which $\omega = \omega_{max}$:

$$t_1 = \frac{\omega_{max} - \omega_0}{\alpha_{max}} \tag{4.2}$$

For $t > t_1$, θ rises linearly with slope ω_{max} until θ_{max} is reached at time t_2 (unless θ_{max} is reached prior to t_1, in which case this linear segment does not exist). This gives the piecewise maximum bound function $f_{max}(t)$:

$$f_{max}(t) = \begin{cases} \theta_0 + \omega_0 t + \frac{1}{2}\alpha_{max} t^2 & \text{if } t \leq \min(t_1, t_2) \\ f_{max}(t_1) + \omega_{max}(t - t_1) & \text{if } t_1 < t \leq t_2 \\ \theta_{max} & \text{otherwise} \end{cases} . \tag{4.3}$$

The minimum bound $f_{min}(t)$ is derived similarly using $(\theta_{min}, \omega_{min}, \alpha_{min})$.

These bounds provide, for some given maximum time bound t_{max}, a corresponding joint range $[f_{min}(t_{max}), f_{max}(t_{max})]$ over which to compute reachability. In practice, t_{max} is set to an application-specific value corresponding to the maximum useful time horizon. To compute the entire workspace we can let $t_{max} \to \infty$, in which case the range is $[\theta_{min}, \theta_{max}]$.

Finally, in order to compute the minimum reachable time for some joint configuration, we also need to compute the inverse $t = f^{-1}(\theta)$, i.e., given a joint value, find the minimum time needed to reach it. This inverse bound is illustrated in figure 4.5b.

As long as we use at most second-order constraints, the bounds are analytically invertible. In case of multiple solutions, we take the lowest non-negative time:

$$f_{max}^{-1}(\theta) \quad = \quad \{\min t \,|\, t \geq 0,\, f_{max}(t) = \theta\} \tag{4.4}$$

The true minimum reachable time for a joint angle is then given by the lesser of the min or the max bound:

$$f^{-1}(\theta) = \min(f_{max}^{-1}(\theta), f_{min}^{-1}(\theta)) \tag{4.5}$$

Since we assume constraints for different joints are independent, any robot pose now has a minimum reachable time equal to the maximum value of $f^{-1}(\theta)$ across all joints. In the next section, we present a workspace estimation algorithm that uses these bounds to produce a reachability grid in metric space.

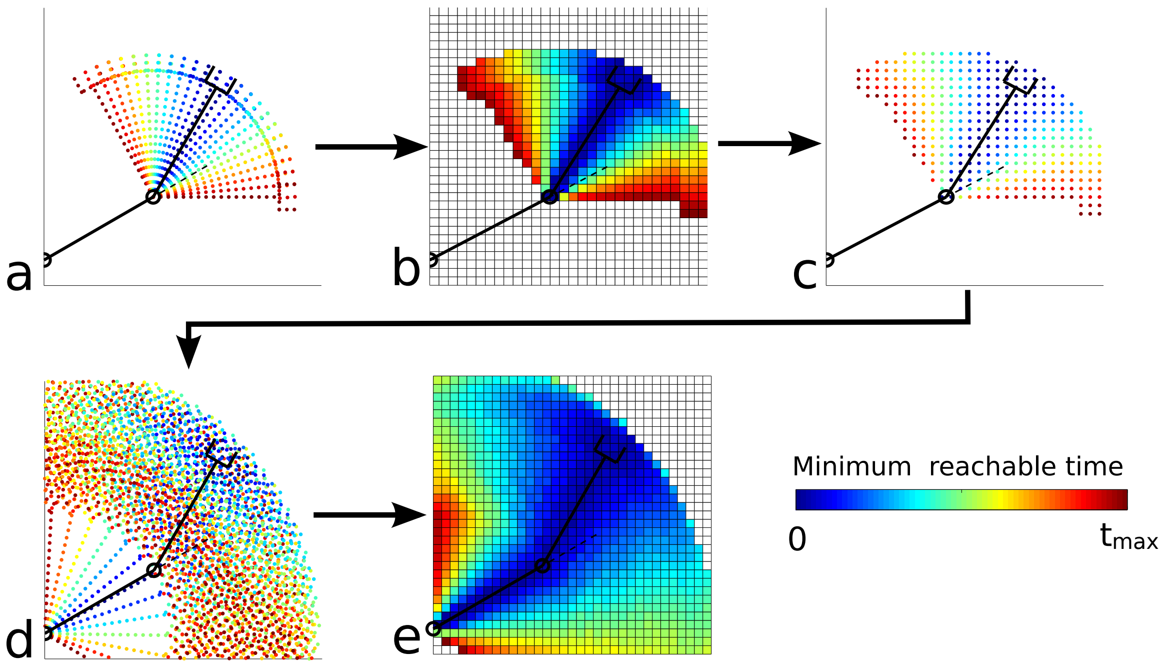


Figure 4.6: Workspace reachability grid computation. a) Point-cloud representation of last link is swept through its range of motion. b) Swept cloud is voxelized. c) Reduced cloud extracted from voxel grid. d) First link is added to reduced cloud and swept through its range of motion. e) Final reachability grid is obtained by voxelizing point cloud with minimum time in each voxel.

4.3 Workspace estimation

The straightforward brute-force approach to mapping joint poses into a workspace would be to sample the entire joint space at regular intervals and save the minimum time value found for each voxel. However, as the size of the joint space is exponential in the number of joints, this method is intractable for high-DoF manipulators.

We instead present an algorithm that uses successive voxel approximations to sweep each joint independently across its range of motion $[f_{min}(t_{max}), f_{max}(t_{max})]$. Our approach is based on the insight that, once we have the workspace for one link calculated in its reference frame, we can calculate the workspace of the previous link recursively by treating the last link's workspace as an attached rigid body. Accordingly, we can use roughly the following steps to compute a reachability grid:

1. Sweep point-cloud representation of last link (end effector) through the range of motion of the last joint
2. Reduce swept cloud by collapsing to intermediate voxel grid
3. Attach reduced point cloud as rigid body to end of link $n-1$ and repeat
4. After all links are swept, voxel grid for base link is final reachability grid

Figure 4.6 illustrates these steps, and pseudocode is shown in Algorithm 4.1. All intermediate point clouds and voxel grids must store the estimated minimum reachable time t associated with each point p and voxel v, respectively.

Note that the algorithm assumes known point cloud representations of all manipulator links. In practice, a suitable cloud can be produced from a triangle mesh model by sampling the enclosed volume at a density equal to, or greater than, that of the output reachability grid. If reachability is being calculated for only the end effector instead of the entire manipulator body, then no initial point cloud representation is needed and a single point at the end effector will suffice.

The point cloud reduction at each joint allows this to run efficiently, even on high DoF manipulators for which an exhaustive search would be impossible. If all points were kept at each stage, the number of points in the cloud would increase exponentially with each manipulator link, making the computation intractable beyond a few degrees of freedom. Given a robot with n degrees of freedom, each of which has ψ angular range of motion, our grid reduction algorithm runs in linear time, $O(n\psi)$, whereas a brute-force solution requires exponential time, $O(\psi^n)$.

4.3.1 Approximation errors

The approximation speedup does come at the cost of induced error, as every time we collapse a point cloud into an intermediate voxel grid, every point will move to the nearest voxel center. To minimize this effect, we typically want to use a smaller grid for the intermediate grid (i.e., *subGrid* in Algorithm 4.1) than the final reachability grid ($finalGrid$ in Algorithm 4.1). As we shrink the intermediate voxel size, we can approach zero error, but in doing so point clouds reduce less and performance approaches that of brute-force.

An additional source of error is in sweeping through the possible angles for each joint. We have to choose how finely to sample these angles; coarser sampling will mean faster computation and smaller clouds, but at the cost of possible skipped regions.

To control these factors, we introduce two parameters, as illustrated in Figure 4.7, that we can use to trade off between accuracy and computational speed. Both parameters are normalized to the final reachability grid cell size.

1. The **subvoxelization ratio** is the ratio of the voxel size of the intermediate grids to the voxel size of the final grid.

Algorithm 4.1 Reachability grid workspace computation

```
$cloud \leftarrow \emptyset$
$finalGrid \leftarrow$ fill($\infty$)
**for** link = n **to** 1 **do**
    $cloud$.addPoints(cloud($link$),0) {$t = 0$ at $\theta = \theta_0$}
    $subGrid \leftarrow$ fill($\infty$)
    **for** $\theta = f_{min}(t_{max})$ **to** $f_{max}(t_{max})$ **step** $\Delta\theta$ **do** {See eq. 4.3,4.6}
        **for all** $(p,t)$ **in** $cloud$ **do**
            $p' \leftarrow$ rotate($p,\theta$)
            $t' \leftarrow \max(t, f^{-1}(\theta))$ {See eq. 4.5}
            **if** link > 1 **then**
                $v \leftarrow subGrid$.getVoxelByPoint($p'$)
            **else**
                $v \leftarrow finalGrid$.getVoxelByPoint($p'$)
            **end if**
            $v.t \leftarrow \min(v.t, t')$
        **end for**
    **end for**
    **if** link > 1 **then**
        $cloud \leftarrow$ voxelCenters($subGrid$)
    **end if**
**end for**
**return** $finalGrid$
```

2. The **step factor** bounds the step size used when sweeping the point cloud through a joint's range of motion.

The step parameter $\Delta\theta$ is set based on the step factor s such as to limit motion of all points p around the joint axis a to some fraction of the voxel size v:

$$\Delta\theta = \frac{sv}{\max(a \times p)} \tag{4.6}$$

We expect the subvoxelization ratio to have a significant effect on error, much more so than the step factor. Voxelization can shift points at most half the diagonal length of a voxel for each link, so we get a maximum displacement error bound ε for an n-DoF manipulator and a subvoxel length l of $\varepsilon \leq \frac{\sqrt{3}}{2}nl$. However, in practice the error will be much less than this. In the next section, we examine the error rate empirically, and test the effect of our approximation parameters on accuracy and computation speed.

4.4 Experimental methods

All tests used a simulated 4-DoF manipulator with two pitch and two roll joints, as shown in Figure 4.8. This arm was chosen because computing ground truth for a higher-DoF arm is prohibitively slow. The arm is 1m long at full extension, and the calculated reachability grid has a voxel width of 5cm. All computation was single-threaded and executed on an Intel Core 2 Quad desktop computer with 2GB of RAM. For robot simulation and visualization, we used the OpenRAVE simulator [4].

Reachability grids were gathered at 10 randomly selected poses. Joint constraints consisted of a maximum speed of $|\omega| \leq 1$rad/s. There were no position or acceleration constraints. Reachability was computed out to

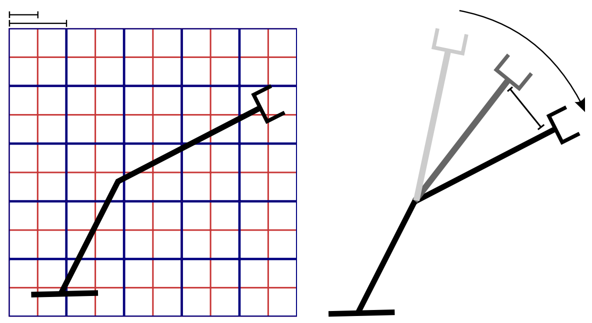

a) Subvoxelization ratio b) Step factor

Figure 4.7: Parameters used in workspace estimation. a) Subvoxel ratio is size ratio of voxels used in cloud reduction (red) to final reachability grid (blue). Ratio shown is 1/2. b) Step factor is worst-case motion between steps of θ during sweep, as a fraction of final voxel size. Factor shown is ≈ 1.

$t_{max} = 0.5s$, giving a total freedom of 1 radian in each joint.

For each test pose, a ground truth reachability grid was computed using a uniformly sampled brute-force search of the joint space. The step factor when calculating ground truth grids was 0.4, ensuring a worst-case distance between sweep steps of 2cm.

Comparison grids were generated with subvoxelization ratios ranging from 0.1 to 1.0, and step factors ranging from 0.1 to 1.5, both at 0.025 increments. False positives and negatives were measured by comparing the full measured workspace against the ground truth grid, i.e., we compared the set of all voxels with minimum reachable time $t \leq 0.5s$.

4.5 Experimental results

Results indicate that our approach does produce an accurate workspace estimate, particularly at low subvoxelization ratios. Figure 4.8 shows an example of errors observed for our 4-DoF test arm. Figure 4.9 shows precision, recall, and processing time as a function of the subvoxelization ratio and step factor parameters, averaged over all 10 test poses.

The primary effect of the approximation errors appears to be a slight dilation of the robot workspace, particularly near the end of the manipulator chain. As shown in Figure 4.8, false positives are more common at the end, while false negatives are uncommon and present only near the base. This is unsurprising, as the large number of points approximated at each step will naturally cause a diffusion effect, which is pronounced with more successive approximations. This means that areas near the end effector will see the most growth due to being moved at every step in the kinematic chain. For most subvoxelization ratios, false positives were no farther than 1 voxel (5cm) from their true locations, but for ratios above about 0.8 some were as distant as 2 voxels. In the reachability grid, the net effect of this dilation is a consistent underestimation of reachability times, the magnitude of which is dependent on the size of the dilation.

Figure 4.9c shows worst case error distance under different parameter values. Most parameter values ensure that false positives are within 1 voxel of their true locations, and even for the most aggressive approximations errors are within 2 voxels (10cm). Note that for more complex robots, we would expect this displacement error to increase linearly with the number of degrees of freedom. However, this is difficult to verify empirically due to the intractability of computing ground truth reachability for high-DoF robots.

As expected, when varying parameters, the subvoxelization ratio has the most significant effect on overall

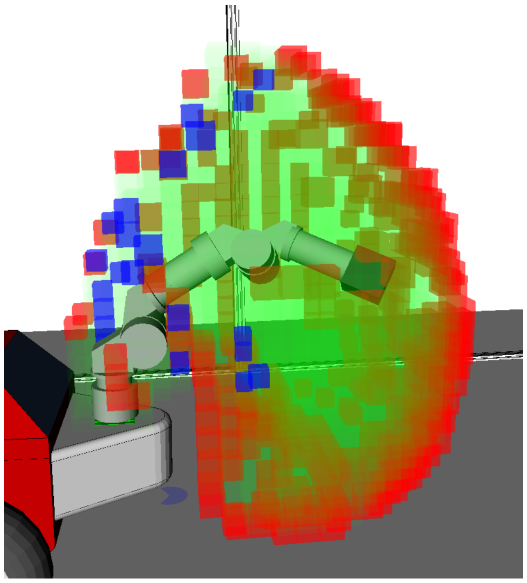


Figure 4.8: Test 4-DoF manipulator with observed errors. Red voxels are false positives (present only in estimated grid), blue voxels are false negatives (present only in ground truth), and the green voxels are correct (present in both estimated and ground truth). Parameters are 0.5 for both subvoxelization ratio and step factor.

performance. Errors are markedly lower for small ratios, but at the cost of dramatically longer processing times, as the number of subvoxels in the workspace is proportional to the cube of their size. In practice, we found that using a ratio of about a half voxel gave a good trade-off between performance and accuracy.

As is shown in Figure 4.9(a,b) certain structured error appears at subvoxelization ratios close to, but under, 1, most noticeably at 0.975. This is due to the relationship between the voxel positions in the intermediate grid and the final grid. When there is a small difference in size between the intermediate and final voxels, the relative positions of voxel centers will be consistently biased over large regions of space, meaning that, depending on the position of the manipulator workspace, boundaries may grow or shrink considerably. In practice, this means that subvoxelization ratios close, but not equal, to 1 should be avoided, if possible.

Varying the step factor does not have a clear-cut effect on overall accuracy, but lower values appear to produce a more generous workspace estimate, whereas higher values produce a more conservative estimate. This is because lower ratios produce larger point clouds, which will naturally diffuse to create larger workspaces. Precision-recall curves generated by varying the step factor are shown in Figure 4.10. Step factors significantly above 1 run the risk of skipping over regions and may cause gaps in the swept volume. However, due to the cloud diffusion effect and the fact that the step factor is a worst-case distance, we did not observe this below factors of about 1.25. The step factor does not have a major effect on speed unless very small values are used (computation time should vary with its reciprocal). We found that a step factor of about 1 gave a good trade-off between precision and recall.

For robots with higher degrees of freedom than our test robot, we would expect the maximum displacement error, and correspondingly the false positive rate, to increase linearly with the number of degrees of freedom. However, this is difficult to verify empirically due to the intractability of computing ground truth reachability for high-DoF robots. That said, we have used our approach on a 7-DoF arm (as shown in Figure 4.4) and results appear consistent with those seen at the 4-DoF level.

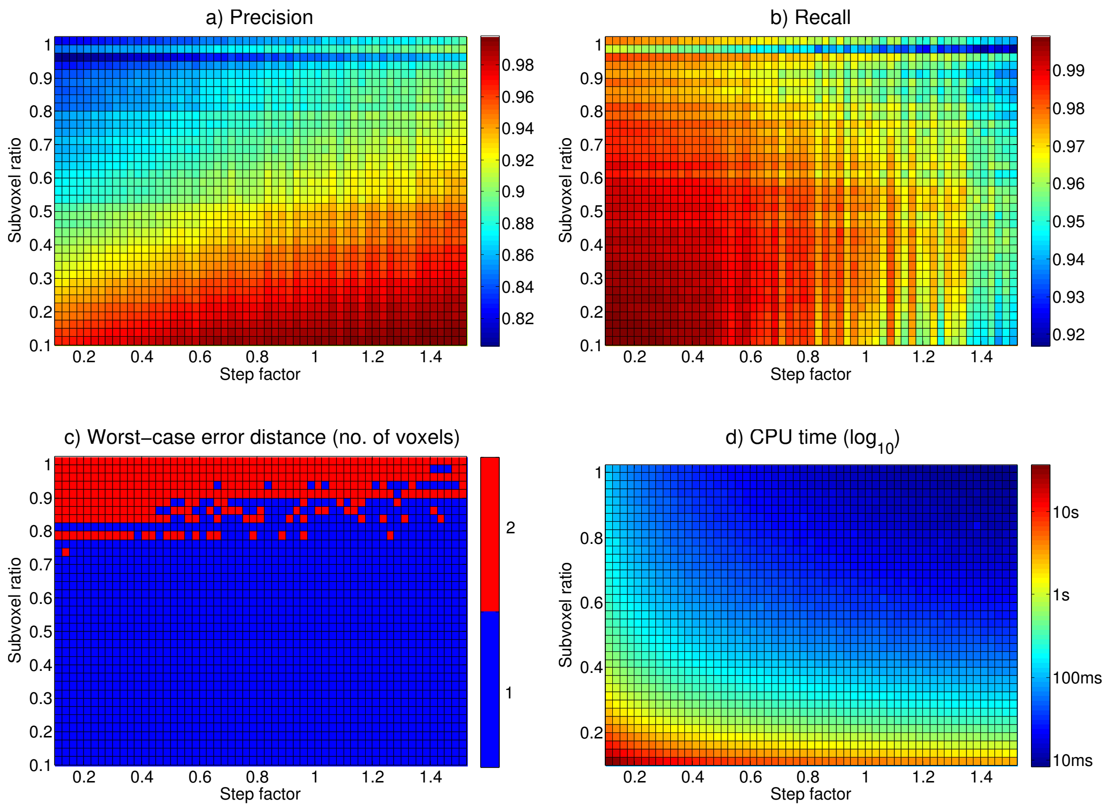


Figure 4.9: Experimental results for reachability grid computation. a) Precision, b) Recall, c) Worst-cast distance error, and d) CPU processing time, as a function of subvoxelization ratio and step factor parameters (see Figure 4.7).

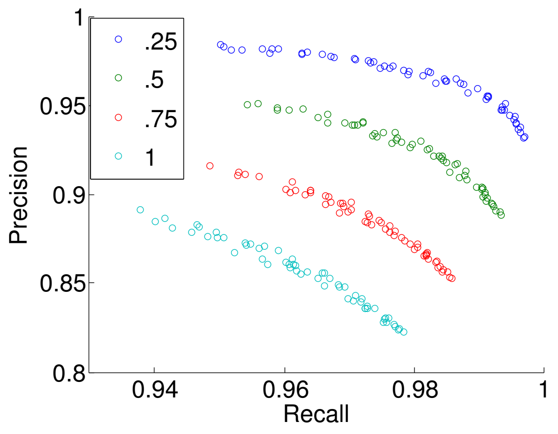


Figure 4.10: P-R curves at selected subvoxelization ratios from .25 to 1. Curves are obtained through varying values of step factor along a given subvoxelization ratio.

4.6 Conclusions and future work

Our results indicate that *reachability grids* are a fast and accurate way to compute motion bounds, even for high degree-of-freedom manipulators. Errors typically result in a displacement of no more than a few voxels away from true locations. The algorithm can run in real-time for even high-dimensional manipulators, is easily parallelizable, and many of the operations could be implemented on graphics hardware for further speedups.

The fact that our approach does not take collisions into account, combined with approximations in the workspace estimation, often results in underestimates of reachability times but rarely or never results in overestimates. This makes it particularly well-suited to applications that require an optimistic estimate of reachability, such as robot safety monitoring and A* path planning heuristics.

While positional bounding as we have demonstrated here is sufficient for some applications, many manipulation tasks need to take end effector orientation into account as well. In principle our approach should work in a higher-dimensional space, but performance will be reduced significantly, so further study is needed to demonstrate its suitability.

Another key limitation of our approach is that it is unable to take any collisions or obstacles into account. Opportunities for improvement here may be limited, as fully exploring the joint space is inherently intractable for high-DoF manipulators. However, with further research it may be possible to reason about collisions in a limited fashion by using the existing method to carve out large regions of obstacle-free joint space while spending extra time in areas near obstacles.

Chapter 5

System evaluation

Full testing and evaluation of the IMAO system is an ongoing effort as of this writing. However, to date we have run several preliminary tests using our four-sensor test environment with Bullwinkle's 7-DoF arm (see Figure 1.4).

All tests involved a simple sweep motion of the robotic arm, which has a maximum speed of about 16cm/s at the end effector. Both safety and danger zones were generated using a dilation of 15cm. Note that danger zones shown did not use joint-based motion modeling as described in in Chapter 4 since these tests were conducted prior to the formulation of that approach.

Tests were comprised of three trials:

1. The arm sweeping into a stationary dummy (Figure 5.1a). This verified basic functionality and safety.

2. The arm sweeping under a person's outstretched arm (Figure 5.1b) and into their torso. This demonstrated that in addition to stopping safely, the manipulator can move nearby a sensed object (the outstretched arm) without stopping needlessly.

3. The arm sweeping while a person walks through its workspace (Figure 5.2). This demonstrates safety with moving objects.

To verify that our safety system performs correctly, we measured the distance between the manipulator and the closest part of the person or dummy standing in its way after the arm came to a stop. Because the safety and danger zones are each 15cm wide, the nominal stopping distance is 30cm, minus however far the arm can move in the time it takes to sense and stop. This time should correspond approximately to our zone generation parameter t_{max}, defined in eq. 1.1, but may be up to t_{frame} less depending on exactly when the frame producing the violation is gathered. Additionally, the distance may be shorter if there is a deceleration period for the arm, though with a slow arm, such as Bullwinkle's, this is negligible. Using our value of $t_{max} = 300ms$ and a robot speed of 16cm/s, we expect the arm to travel about 5cm after zones intersect, giving an expected stopping distance of 25cm $\pm$ 5cm for voxel rounding.

In the first test, the arm stopped successfully with a remaining distance of 34cm. This is slightly farther than projected, probably due to some false positive voxels appearing on the surface of the dummy, which are relatively common in practice due to sensor noise. In the the second test, the arm also stopped successfully with a final distance of 25cm, almost exactly our projected distance. The robot did not stop until reaching the subject's torso despite moving near his arm most of the way, verifying that foreground objects could be near robots without causing unnecessary safety stops. The final test with a moving person did not yield a stopping distance because the subject's motion at the time of the stop precluded measurement, but the robot did stop successfully without colliding.

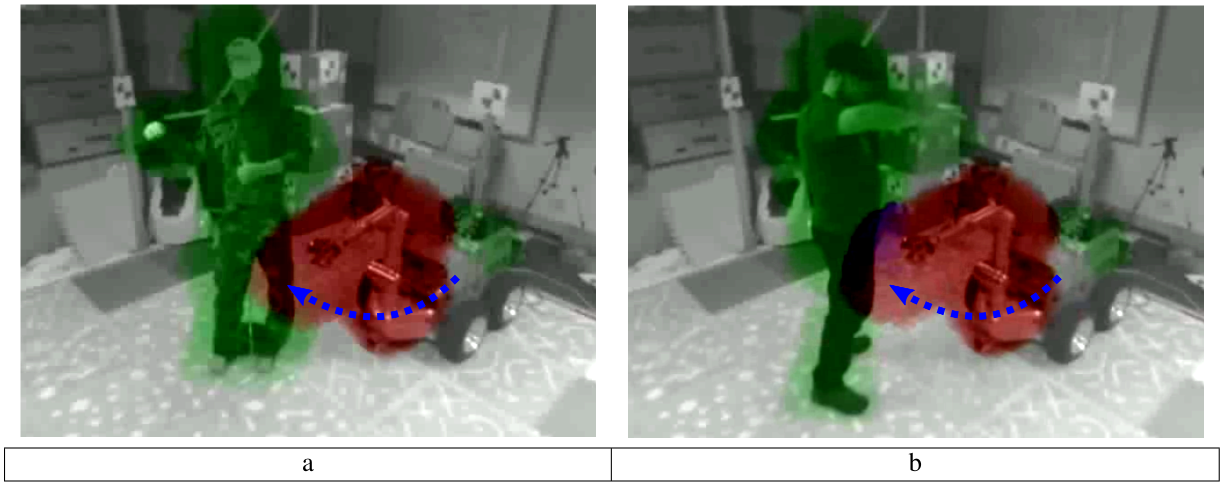


Figure 5.1: Collision tests with a stationary foreground and moving manipulator. Green region is the safety zone, red region is the danger zone, blue is intersection between the two. a) Arm swings into stationary dummy. b) Arm swings into person with outstretched hand

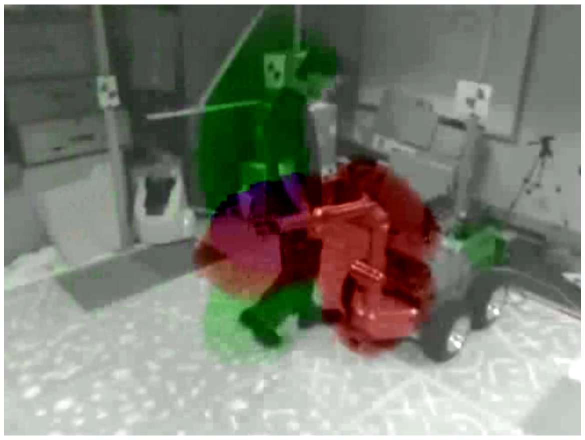


Figure 5.2: Collision test with moving foreground and moving manipulator. Green region is the safety zone, red region is the danger zone, blue is intersection between the two.

These tests are only a first attempt and further work is still needed to fully evaluate the performance of the IMAO system. The safety zones we used are much smaller than the theoretically recommended values based on maximum movement speed: with $t_{max} = 300ms$ and an assumed maximum movement speed of 2m/s, we should theoretically be required to use safety zones of about 60cm, or about 4 times what is shown in our tests. Further testing is needed to determine if these larger safety zones are acceptable to enable people to perform useful work in the vicinity of the robots, or if we need to lower t_{max} through optimization and increased computing power.

Additionally, our test workcell is a very open environment, with almost nothing in it other than the robot and the foreground. The one background object that is in the workcell during the tests, the robot base, generated a false positive safety zone during robot movement as shown in Figure 5.1. This was caused by variations in sensing due to occlusions. If there are more background objects, and especially if the safety zone size is increased significantly, false positives like these may very easily impede normal operation. Further testing is needed to find what level of background clutter is tolerated, and what if anything can be done to reduce the incidence of false positives without jeopardizing true detection.

5.1 Conclusions

The Intelligent Monitoring of Assembly Operations project as currently designed and tested promises to be a fast and accurate safety monitoring system for use in industrial assembly environments. We have constructed a robust architecture that supports a wide variety of assembly scenarios, and could serve as a framework for future development. Additionally, during the course of this work we have addressed several fundamental research challenges that have other applications beyond the specific goals of this project. The accessibility-based background analysis method proposed here would be well-suited for any safety-conscious sensing application, and our fast motion bounding technique could form the basis of a good heuristic for manipulator path planning.

Further optimization and testing is still required to ensure that IMAO works well in real-world environments. Tests conducted thus far used very simple environments, and did not involve manipulation of complex payloads. Preliminary results indicate that a major challenge in making the system useful for real-world environments is the prevention of spurious safety zones: in particular while robots are moving, varying degrees of occlusions have the potential to cause false safety zones near the robots, resulting in unnecessary safety stops. Another related challenge is to make zone generation run quickly enough so that the zones, in particular safety zones, can be as small as possible.

Also, in order to take full advantage of the safety methods presented here, further work is needed in constructing frameworks and procedures for cooperative human-robot assembly. Cooperative assembly tasks should be planned such that assembly runs smoothly with no delays, while keeping enough distance between robots and humans during normal operation such that robots can operate quickly and efficiently without unneeded safety stops.

Ease of use for operators is also key: in a real-world setting workers will need to be able to use the system safely and effectively without a large amount of training. In particular, tasks such as background gathering, workcell setup, and camera calibration and validation would need to be as automated as possible.

5.2 Acknowledgments

Special thanks to Reid Simmons, Daniel Huber, Paul Rybski, Chris Niessl, Thomas Koletschka, Kartik Babu, and Seth Koterba for their work on the Intelligent Monitoring of Assembly Operations project. This work was supported by the Office of Naval Research under contract #N00014-09-D-0584.

Bibliography

[1] ABB. Safemove white paper, 2008. Doc no. ROP 6904, http://www.abb.com/.

[2] K. Abdel-Malek, D. Blackmore, and J. Yang. Swept volumes: Foundations, perspectives, and applications. *International Journal of Shape Modeling*, 12:87–127, 2006.

[3] G.K.M. Cheung, T. Kanade, J. Bouguet, and M. Holler. A real time system for robust 3d voxel reconstruction of human motions. In *2000 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 2, page 2714, 2000.

[4] R. Diankov. *Automated Construction of Robotic Manipulation Programs*. PhD thesis, Carnegie Mellon University, Robotics Institute, August 2010.

[5] A. Elfes. *Occupancy Grids: A Probabilistic Framework for Robot Perception and Navigation*. PhD thesis, Carnegie Mellon University, 1989.

[6] S. Kock et al. Taming the robot: better safety without higher fences, April 2006. Doc no. 9AKK105152A2830, http://www.abb.com.

[7] J. Franco and E. Boyer. Fusion of multiview silhouette cues using a space occupancy grid. In *2005 IEEE International Conference on Computer Vision*, volume 2, pages 1747–1753, 2005.

[8] F. Freudenstein and E.J. Primrose. On the analysis and synthesis of the workspace of a three-link, turning-pair connected robot arm. *Journal of Mechanisms, Transmissions, and Automation in Design*, 106:365–370, 1984.

[9] A. Golovinskiy and T. Funkhouser. Min-cut based segmentation of point clouds. In *IEEE Workshop on Search in 3D and Video (S3DV) at 2009 International Conference on Computer Vision*, 2009.

[10] D. Hahnel, R. Triebel, W. Burgard, and S. Thrun. Map building with mobile robots in dynamic environments. In *2003 IEEE International Conference on Robotics and Automation*, volume 2, pages 1557–1563, September 2003.

[11] E. J. Haug, Chi-Mei Luh, F. A. Adkins, and Jia-Yi Wang. Numerical algorithms for mapping boundaries of manipulator workspaces. *Journal of Mechanical Design*, 118(2):228–234, 1996.

[12] Carnegie Mellon University Robotics Institute. Trestle: Autonomous assembly by teams of coordinated robots. http://www.frc.ri.cmu.edu/projects/trestle/robots.htm.

[13] A. Jain, M. Murty, and P. Flynn. Data clustering: A review. *ACM Computing Surveys*, 31:265–322, September 1999.

[14] S. Jeong, T. Takahashi, and E. Nakano. A safety service manipulator system: The reduction of harmful force by a controllable torque limiter. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 162–167, September 2004.

[15] A. Kundu, M. Krishna, and C.V. Jawahar. Realtime motion segmentation based multibody visual slam. In *Seventh Indian Conference on Computer Vision, Graphics and Image Processing*, 2010.

[16] M. C. Martin and H. P. Moravec. Robotic evidence grids. Technical Report CMU-RI-TR-96-06, Robotics Institute, Carnegie Mellon University, 1996.

[17] H. Moravec and M. Blackwell. Learning sensor models for evidence grids. *CMU Robotics Institute 1991 Annual Research Review*, 1:116–121, 1991.

[18] H. Moravec and A. Elfes. High resolution maps from wide angle sonar. In *1985 IEEE International Conference on Robotics and Automation*, volume 2, pages 116–121, 1985.

[19] T. Morita and S. Sugano. Double safety measure for human symbiotic manipulator. In *1997 IEEE/ASME International Conference on Advanced Intelligent Mechatronics*, pages 16–20, Tokyo, Japan, 1997.

[20] J. Rastegar and P. Deravi. The effect of joint motion constraints on the workspace and number of configurations of manipulators. *Mechanism and Machine Theory*, 22(5):401 – 409, 1987.

[21] J. Spanos and D. Kohli. Workspace analysis of regional structures of manipulators. *J. Mech. Transm. Autom. Des.*, 107(2):216–222, Jun 1985.

[22] F. Stulp, A. Fedrizzi, F. Zacharias, M. Tenorth, J. Bandouch, and M. Beetz. Combining analysis, imitation, and experience-based learning to acquire a concept of reachability. In *9th IEEE-RAS International Conference on Humanoid Robots*, pages 161–167, 2009.

[23] Y. C. Tsai and A. H. Soni. An algorithm for the workspace of a general n-r robot. *ASME Journal of Mechanisms, Transmissions, and Automation in Design*, 105:52–57, 1983.

[24] D. Vasquez, F. Romanelli, T. Fraichard, and C. Laugier. Fast object extraction from bayesian occupancy grids using self organizing networks. In *9th International Conference on Control, Automation, Robotics and Vision*, pages 1–6, 2006.

[25] C. Wang and C. Thorpe. Simultaneous localization and mapping with detection and tracking of moving objects. In *2002 IEEE International Conference on Robotics and Automation*, volume 3, pages 2918–2924, 2002.

[26] S. Wangsiripitak and D.W. Murray. Avoiding moving outliers in visual slam by tracking moving objects. In *2009 IEEE International Conference on Robotics and Automation*, pages 375–380, 2009.

[27] F. Zacharias, C. Borst, and G. Hirzinger. Capturing robot workspace structure: representing robot capabilities. In *IROS*, pages 3229–3236, 2007.