Active Object Segmentation for Mobile Robots

Evan Herbst * Dieter Fox *

May 28, 2014

Abstract

The goal of active vision is to change intrinsic or extrinsic properties of the sensor in order to get new and improved information. In the case of 3-D object modeling from vision, this can mean moving the camera to view the scene from a new angle or to get a close-up view of an object that has been localized and is being modeled. We discuss using active vision to improve the speed and utility of map completion and object segmentation. Importantly, in order to be able to process untextured surfaces, we avoid relying on the existence of distinctive visual point features.

1 Introduction

One of the most rapidly expanding subfields of commercial robotics is that of service robotics for indoor environments populated by people. One particularly important entity in these environments is the *object*, defined by physical contiguity and independent movement. Some of the major types of vision algorithm meant for reasoning about objects are segmentation, recognition and affordance analysis. We have developed multiple algorithms for segmenting objects in indoor environments using different kinds of motion cues: change detection [16], dense feature matching [18] and dense nonrigid motion estimation [19]. One characteristic these three methods have in common is that they are *passive* segmentation techniques: each is given a fixed group of measurements from which to extract a segmentation. We should be able to improve the speed and accuracy of segmentation if we have access to a physical agent with the ability to rearrange the scene to give itself more information, because future measurements can be directly related to a notion of what information is contained in past measurements. If the

*University of Washington Computer Science Department

robot's camera is movable, moving it can provide views that would never be obtained by a passive agent. Humans who might otherwise move the camera probably won't know the robot's intentions or current knowledge, and it is very unlikely that humans would feel the need to rearrange objects for the viewing benefit of a camera with a fixed position in the environment, as they would be unaware that this would be helpful to vision algorithms. Additionally, some objects are moved by humans, but only very seldom, so that a robot might not observe movement of all objects in its environment for months or years after entering service.

These arguments motivate *active object segmentation*, in which the robot literally takes matters into its own hands and causes the camera or objects to move. (The more general concept of *active perception* was introduced by Bajcsy [3].) Having the ability to choose what data to process means a robot can segment objects when it wants and can choose how much of a scene to understand before moving on. Importantly for algorithms using formal methods in robotics (e.g. [26]), it is also possible to control the amount of certainty we have about object boundaries: the robot can manipulate an object whose configuration it is unsure of until its model has the desired amount of uncertainty. In this report we present a method for choosing motions of a camera mounted on a robot to ensure that the robot continually sees new parts of the scene.

2 Background

This work builds on recent work in online robot mapping and passive object segmentation.

2.1 Online SLAM

The problem of static SLAM, simultaneous scene mapping and localization of a camera moving through an unchanging scene, has been the subject of a great amount of work in robotics since the mid-1980s and [11, 31]. With the introduction of high-frame-rate RGB-D cameras such as the Microsoft Kinect (designed by PrimeSense Inc. [7]) and of GPGPU techniques have come online GPU-based SLAM techniques based on the recursive-filtered volumetric reconstruction of Curless and Levoy [8]. The geometric map is represented with a truncated signed distance function (TSDF) over voxels. This representation permits extraction of a surface at any time, and can be extended to store surface color. The work described in this report makes

heavy use of the Patch Volumes online SLAM system [14], a recent GPU-based volumetric mapping system that can align and add RGB-D frames to a volumetric map at about ten frames per second.

2.2 Online Object Segmentation

Most of today's online SLAM systems are meant for static scenes, in which the camera is assumed to be the only moving thing. In [15] we extend the Patch Volumes static SLAM system to segment and model *multiple* rigid objects. This is the only work we know of that performs dense segmentation and modeling of multiple scene elements in an online fashion. We use change detection between the current video and a previously built map to identify moved objects. We model each of these objects separately, using static SLAM, and create a *background map* containing none of the objects seen to have moved. This system that performs online passive mapping and object segmentation can be naturally extended to use active vision to control the movement of the camera. We illustrate this idea in fig. 1, which explains how online object segmentation with change detection works.

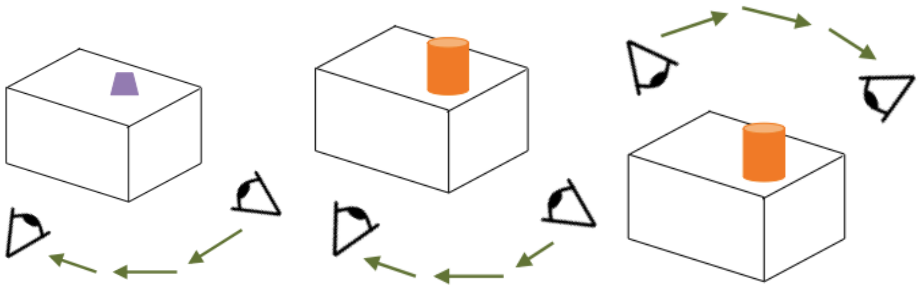


Figure 1: the advantage of active over passive mapping techniques, illustrated in the context of change detection. First the camera traverses the path of green arrows in (a), observing the purple pyramid on the table. After some time passes, the camera returns to the scene and traverses a similar path in (b), observing the orange cylinder. It is not possible to decide from only these views whether the purple pyramid is still present, as the space it would occupy is occluded from all the camera's viewpoints. However, if the camera is held by a robot that can control the camera's movement, it is possible to view behind the orange cylinder (c) and observe whether the purple pyramid is present. (In this example, the robot is able to reason that the pyramid might still be present in the same location. In general there are other algorithms motivating the choice of viewpoints.)

2.3 Active Object Segmentation

The goal of this work is to aid 3-D segmentation, given 2-D segmentation tools such as those mentioned in sec. 1, by viewing previously unseen parts of the map. The *map completion* problem is defined as moving the camera to see all surfaces in the map, and the (less well-defined) *active segmentation* problem is to move the camera to improve the speed and/or accuracy with which we can segment objects.

For our purposes, map completion is a proxy problem for object model completion. In the online mapping and modeling system just discussed, if an occluder is removed and we observe an object behind it, change detection alone cannot segment the object completely because there is no change detection evidence for the section of the unoccluded object that was behind the occluder. However, since we know the boundaries of the occluded region (which includes the object as well as some other surfaces), we can still make a model that contains the object by taking advantage of having a camera on the robot. The object and background in the resulting model cannot be separated perfectly until we see the same region of background without the object. Therefore we wish to model at least some interesting portions of each scene we see as completely as possible in order to give us maximal information for processing future scenes. Another reason to use active vision for object modeling is that the more complete an object model, the more useful it is for object discovery.

The most popular paradigms for map completion (sometimes called "exploration") are frontier-based and next-best-view exploration. Given a map representation that divides space into free, surface, and unseen areas, *frontiers* are defined as the parts of the map that border both free and unseen space [35]. Frontier-based approaches to map completion iteratively select frontiers to view with the sensor (usually a camera). Next-best-view approaches, which are a type of generate-and-test algorithm, select a set of possible future viewpoints to score according to some notion of usefulness of a view, which depends on the current contents of the map. Notions of usefulness on a volumetric map are generally computationally expensive to calculate.

We introduce a frontier-based framework for solving map completion and related problems such as active segmentation. We do this with a general representation of preferences on what areas are viewed next. We can incorporate a wide range of preferences for selecting viewing locations so as, for example, to focus on completing our models of regions we know to be objects (e.g., from change detection) but that are partly unseen, or to

prioritize viewing regions of space that are blocking the movement of our mobile manipulator due to being unseen, but that are likely to be free of surfaces. We use a computationally inexpensive method for selecting views given preferences, allowing us to make decisions at a higher rate than any other 3-D next-best-view approach of which we are aware.

3 Related Work

Every next-best-view view selection algorithm proposes a set of possible sensor poses for the next sensor reading, selects the best by some metric (usually based on information gain), moves the sensor to the selected pose, and integrates the resulting reading into a representation of the scene. Information gain is very hard to define for applications in which the goal is to measure variables that are currently completely unknown, so most work uses rough approximations for view scoring. Work on selecting next best views by using projections of unseen space into potential camera poses goes back at least to 1985 [6], and there has been a large amount of work recently using similar algorithms in 3-D, e.g. [25, 33, 29]. Thrun et al. [32] discuss robot exploration during 2-D mapping by selecting next views to maximize an approximation of information gain. We compare to such an approach in 3-D.

The most common variant is *myopic* next-best-view planning, in which only a single future view is considered. For example, Triebel et al. [33] select a next pose for a laser range finder to maximize an approximation to information gain given by assuming a point distribution over readings at each single scan (in particular, they assume the first voxel along a ray with occupancy probability over a threshold will be hit by the beam). Their pose proposal step constrains next views to be reachable by the manipulator on which their sensor is mounted. Gonzalez-Baños and Latombe [13] select a next view using a combination of traversal cost and a novelty score given by the total volume of the currently unseen area that would be seen by the new view. Potthast et al. [29] use the penetration depth of each hypothetical pixel ray into an unseen region as an estimate for information gain, reasoning that the more unseen space could be seen through, the more likely we are to find out where the surface actually is.

By contrast to these myopic methods, Atanasov et al. [1] select camera poses for tabletop object classification by planning a sequence of two or more viewpoints for each pre-segmented object. They maintain a discrete distribution over object models in a database to estimate which known object is

currently being observed. They include traversal cost in planning. Similarly, Hollinger et al. [20] first select a set of overlapping views of a ship hull and plan a path that can use any subset of these views to cover the entire hull for visual search purposes. They use this method only after building a complete object model; they use view selection to reduce uncertainty in that model. In both these projects, view selection and path planning are done off line.

Blodow et al. [5] select next-view poses for a mobile robot with a laser scanner by ray-tracing in 2-D (similar to a 2-D laser scanner such as a Hokuyo) at each proposed pose until each ray hits a non-free voxel. Their score function rewards poses from which the numbers of frontier voxels (representing new information) and surface voxels (in order to perform alignment to the existing map) thus rendered are equal. This is an example of a technique that fits in both the next-best-view and frontier-based categories.

Holz et al. [21] quantitatively compare frontier-based and non-frontier-based view selection methods and find that the two frameworks yield similar results, despite the fact that next-best-view approaches consider more information in view scoring than the relatively simple frontier-based methods they test. Their main research contribution is a quantitative study of early stopping in map completion, in which replanning happens as soon as the current target map region (part of the frontier) is visible rather than after the current target pose is reached.

Shen et al. [30] use a gas diffusion model to identify frontiers in a low-memory, low-computation setting and show that they can produce a map of similar quality to one produced by a more conventional frontier-based algorithm operating on a non-embedded platform.

The theory of submodular optimization (an analogue of convexity for discrete optimization) allows for efficient optimization of objectives in which acquiring the same or similar information loses value over time as the variables involved are modeled better and better by the growing set of previous views. Golovin and Krause [12] define *adaptive submodularity* for problems in which action selection should depend on recently acquired information. They present a greedy algorithm for optimization of adaptive submodular objectives, which is applicable, for example, to the map completion problem. Javdani et al. [24] note that information gain is not adaptive submodular, and so is theoretically not usable as an objective for view selection. They suggest instead using objectives based on hypothesis pruning, in which actions are selected to reduce the set of plausible hypotheses by the maximum number of elements at each timestep. They use an objective based on hypothesis pruning with a tactile sensor to do object pose estimation.

One issue in designing map completion-like algorithms is selecting a ter-

mination condition. Object model completion using probabilistic measures of completeness as a function of viewing direction is demonstrated by Zillich et al. [36]. They use point feature matches to a model as the basic unit of information about an object, and update the model after each view. One of the nicest features of their work is a condition for when to stop acquiring new views of an object based on the distribution of point features in its model. Krainin et al. [25] use a similar measure of model completeness defined at all vertices of an object mesh, which is assumed to be complete and watertight before view planning.

Our setup is different from those in most of the references above (excepting Zillich et al. [36]) in that they all use sensors mounted in fixed positions on a robot base, whereas we attach a camera to the end of a robot arm. Our scenario is more general than that of Zillich et al. in that they are interested in modeling a single object with a known background, whereas we are interested in modeling the entire scene, starting with no knowledge of any surfaces in the scene. Therefore we integrate camera motion with a SLAM system. We use an online SLAM system [14] and we desire to map the scene as quickly as possible, without spending valuable camera time sitting in place during slow planning. Our motivation for this decision is the intuition that the more of the time the camera is in motion, the more quickly we will acquire new information. Since our system is designed for use on a robot, we want the entire system to run as close to in real time as possible. This motivates the use of a novel frontier-based approach in which we select views using a simple scoring method and expect to move only partway to the selected view before selecting another.

Our ultimate interest is in using view selection for object segmentation. Possibly the most similar existing work to this goal is that of Holz et al. [22], who use view selection for object detection in a bin picking task. They use a pipeline of geometric primitive fitting followed by object detection. Each voxel in a volumetric map is given a value for view selection purposes; values are based on the outputs of the two pipeline stages at each voxel, which can change over time as view selection and the detection pipeline are run alternatingly. Important differences from our work are that Holz et al. detect a known object rather than novel ones and that they assume their object's shape is composed of a small set of geometric primitives.

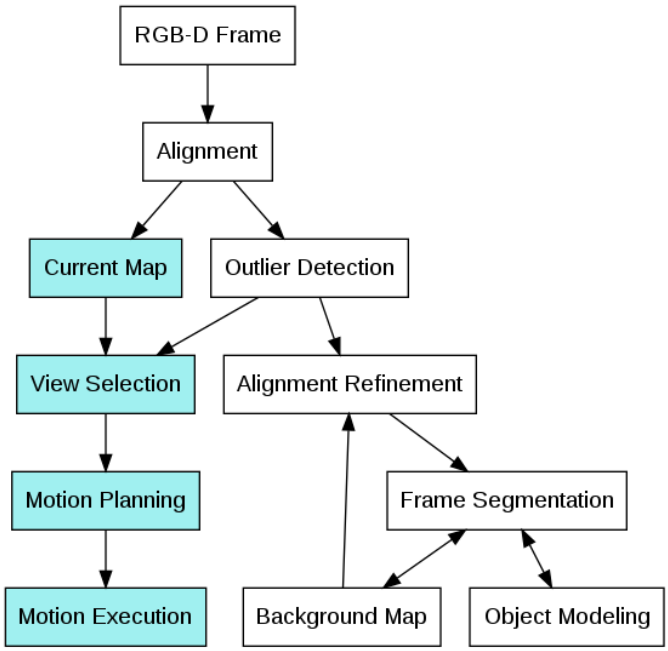


Figure 2: a diagram of our active online mapping and object segmentation system. Blue nodes are not present in the system of [15]. This diagram should be interpreted as showing information flow rather than execution flow, as various parts of the system run in parallel.

4 Framework Overview

We have implemented a view selection and motion planning framework as an extension to the online 3-D reconstruction and object segmentation system introduced in [15]. We show a flow chart of modules in the entire system in fig. 2. We run mapping in one thread, object modeling (introduced in [15]) in another, view selection and motion planning in a third, and motion execution also in its own thread. View selection uses the output of mapping, and produces a control input for a (simulated or real) robot with a movable camera. For example, the camera can be attached to a kinematic-chain manipulator or can be free-flying (this is one approach to simulating something similar to a quadrucopter). Automated view selection could also be used as a hint to a human carrying the camera, similarly to [10], in which the user is shown what areas of a map being built are still incomplete.

Active mapping involves 1) view selection, which suggests future camera

Algorithm 1 pseudocode for our view selection and motion planning algorithm. This code runs iteratively starting as soon as the map is initialized and ending once a termination criterion is reached.

Input: map $\mathcal{V}$

if $ViewTarget$ defined **then**
  $Success \leftarrow (\neg$TargetInFreeSpace$(ViewTarget, \mathcal{V})$
  $\wedge \neg$TargetUnmeasurable$(ViewTarget))$
  if $Success$ **then**
    $(Plan, Success) \leftarrow$ TryPlanToPose$(p, \mathcal{V})$
  if $Success$ **then**
    execute $Plan$
  else
    $ViewTarget \leftarrow$ undefined
if $ViewTarget$ not defined **then**
  $\mathcal{M} \leftarrow$ ExtractMesh$(\mathcal{V})$ ▷ Marching Cubes
  $\mathcal{F} \leftarrow$ ExtractFrontier$(\mathcal{M})$
  $\mathcal{V} : \mathcal{F} \mapsto \mathbb{R} \leftarrow$ ValuateFrontierTris$(\mathcal{F})$
  sort $\mathcal{F}$ by $\mathcal{V}$ ▷ index into $\mathcal{F}$
  $i \leftarrow 0$
  $Success \leftarrow$ false
  while $\neg Success$ and $i < |\mathcal{F}|$ **do**
    $(T_{next}, i) \leftarrow$ NextTriByValue$(\mathcal{F}, \mathcal{V}, i)$
    $\mathcal{P} \leftarrow$ SuggestViewingPoses(T_{next})
    $\mathcal{P} \leftarrow$ FilterByStaticReachability$(T_{next}, \mathcal{P}, \mathcal{V})$
    $\mathcal{P} \leftarrow$ FilterByCollisionChecking$(T_{next}, \mathcal{P}, \mathcal{V})$
    $\mathcal{P} \leftarrow$ FilterByNonvisibility$(T_{next}, \mathcal{P}, \mathcal{V})$
    $\mathcal{P} \leftarrow$ FilterByNonmeasurability$(T_{next}, \mathcal{P}, \mathcal{V})$
    for all $p \in \mathcal{P}$ **do**
      $(Plan, Success) \leftarrow$ TryPlanToPose$(p, \mathcal{V})$
      if $Success$ **then**
        execute $Plan$
        end forall
end the experiment if we exhausted the pose suggester (there's nothing unseen that we can move the camera to)

poses, and 2) motion planning given a target camera pose. Alg. 1 outlines our active mapping update algorithm. After initialization of active mapping, this update is run repeatedly until an externally defined completion condition is met.

At any given time there can be a view target (a location on a known surface or on the 3-D frontier) defined. (Initially there isn't.) We plan to move toward a configuration viewing the current target if there is a target.

We choose a new target when the target becomes seen (i.e. moves off the frontier into known-free space) or is found to be unmeasurable. (In real scenes, some surfaces will never or rarely return readings to an infrared sensor. We model this phenomenon by quantizing view targets, remembering which quantized targets were viewed but did not return valid depths, and avoiding targeting these views again.)

Target selection starts with ExtractMesh, which runs the Marching Cubes [28] mesher for volumetric maps. Our version extracts both the zero level set of the signed distance function and the boundaries between empty and unseen space (frontiers), and we use ExtractFrontier to enumerate only the frontier triangles in this triangulated mesh. ValuateFrontierTris assigns a viewing utility to each triangle; we will use these values to rank views later. We run through triangles in decreasing order of utility, suggesting one or more poses from which to view each in turn and attempting to plan to each of these before giving up on a triangle. Viewing poses are suggested using only information about the single triangle in question, so we need to filter them (lines 19–22 of alg. 1) based on reachability given physical robot constraints, collision checking with the environment, visibility of the target triangle from the suggested camera pose (again, there can be obstacles), and information about unmeasurable surfaces (in real scenes, some surfaces don't return readings). When planning (line 24) is successful, we set the triangle's centroid as the new view target.

Having a long list of potential view targets (a room-sized map in progress with a 2-cm voxel resolution contains between 100,000 and 1,000,000 frontier triangles) gives us robustness to failure of planning to a single target. At any point, if there is *any* frontier region we are able to plan toward, we can continue mapping.

5 Simple and Inexpensive Planning

In most previous work, view selection has been a computationally expensive operation run once each time the robot reaches a new target. We make view selection/planning inexpensive enough to be run continually while the robot is moving. This means that we might plan the robot's motion for, for example, time interval $t \in [1, 2)$ when $t = 0$ and we don't yet have all the map information we will at $t = 1$. However, because our planning is fast and we plan short motions, we have *almost* the most recent information, and we have the advantage that the robot is continually improving the map rather than sitting idle during planning. One consequence of this approach

is that Holz et al. [21]'s "rapid rechecking" isn't so important for us since we don't move the camera very far at a time. Holz et al. check at each processing iteration whether the current view target has already moved into free space, and if so cancel the rest of the movement toward it; in our case, this cancelation consists of setting the view target to "none". The rechecking strategy is, of course, still useful: on the map completion problem in a simulated environment that we will discuss shortly, rechecking saves us 15% of travel distance and 35% of runtime. All the results we report in this section use rechecking.

To help ensure that the camera will move continuously, we keep a queue of up to $k = 2$ planned future motions pending at all times. The queue size limit helps avoid planning with too much stale information.

Another motivation for this continual-replanning approach is that the faster our replanning is, the sooner it can make use of new information in the environment; in the limit, if planning can run once per incoming frame, it becomes possible to replan based on objects that move during planning. This will allow the camera to follow objects that are moving as they are being observed, in order to complete object models *during* the movement of those objects, or to track how nonrigid objects' shapes change as they move.

One benefit of continual replanning is the ability to use simple and fast planners. Common options for motion planning of general-case motions are rapidly exploring random trees [27] and iterative path smoothers such as CHOMP [37]. A smoother must be initialized with an approximate path (which can be very simple, such as a straight-line path in configuration space). We have experimented with both methods. RRTs as implemented in OpenRAVE [9] run for two to four seconds on our problems (with a large amount of occlusion in the environment), and CHOMP usually runs for .5 to 1 second. Due to our continual-replanning approach, however, we have another option: configuration interpolation. After selecting an end pose for the path, for kinematic chains we use linear interpolation in configuration space, and for a free-flying camera, whose configuration is its pose, we use linear interpolation of translation and spherical linear interpolation of rotation. This planner runs in an average of .03 seconds, with a standard deviation of .02, and takes longer than .1 seconds less than 1% of the time. Linear interpolation is a reasonable choice because we keep paths short: in order to avoid having the execution of a path take much longer than planning the next path, we execute only the beginning of each planned path. Thus each view target takes one or more executed plans to reach. In return for the advantage of being able to use very simple planning, we have the potential disadvantage of not being able to move between two very separated areas

of the environment that are both interesting; that in general requires more sophisticated planners. As the emphasis of this work is on simplicity and on planning only to nearby locations, we leave exploration of this tradeoff for future work.

6 View Selection

The mesh extraction algorithm in Patch Volumes is a version of marching cubes modified to provide meshes for both surfaces and the frontier. We use the centroid and normal of each frontier triangle as a potential view target location and orientation. We limit the size of this set of triangles by subsampling with a required minimum distance between samples.

Function ValuateFrontierTris in alg. 1 assigns a value to each frontier triangle according to some idea of how useful it would be for the camera to view that triangle next. Where the camera might view the triangle from depends on SuggestViewingPoses, but almost certainly there will be more than one triangle in the view, so the value of a triangle is allowed to take information about other parts of the scene into account. Our framework for view selection is general and the particular algorithm can be anywhere on the efficiency spectrum, from a simple approach that values all frontiers the same to a sophisticated algorithm that uses globally defined information about surfaces in the scene. For example, one of the most common approaches to scoring views is to render the scene into each view and process the set of scenels seen by each view. We evaluate such an approach later in this section.

The simplest goal for a view selection algorithm is map completion: selection of new poses can cease when there is no unseen space left in the reconstructed map. Some value functions defined to achieve map completion are given in sec. 7.1. We define a framework for selecting views rather than a single algorithm because we are also interested in more complex goals. The effects we can achieve with our per-scenel valuation include:

- Sometimes (e.g., with the use of change detection) we have a partial segmentation of an object and can perform the rest of the segmentation if we choose views to see the edge of the surface region known to be part of the object. By assigning high value to frontier triangles near interesting surfaces, we can force the system to view unseen parts of known objects before moving to areas not already known to contain objects.

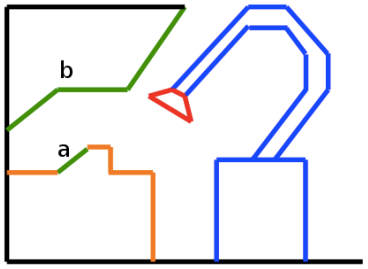


Figure 3: an illustration of potential motions being blocked by unseen space. The robot (blue) is holding the camera (red). Black shows some of the edges of the map volume. The camera has already seen a table with an object on it (orange). Green shows frontiers: edges between free and unseen space. The robot would like to move the camera to view the left side of the object (a), which is currently on the frontier, but due to the physical extent of the robot, it would have to penetrate the unseen space above the table (b) in order to do so. This problem suggests having a way to prioritize looking at unseen space far from known objects. (Any algorithm in our framework would still look at the blocking space in this figure eventually, because all suggested view targets are considered in order; the issue is one of efficiency rather than completeness.)

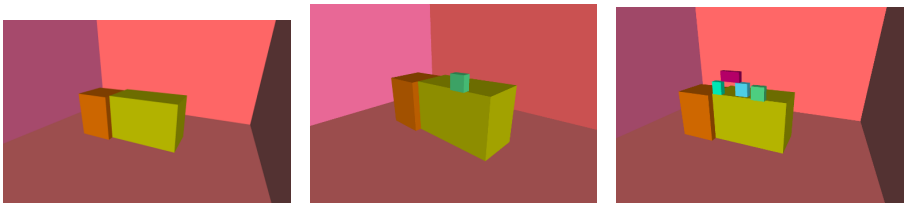
- One possibly desirable behavior when modeling objects is to view all sides of an object before moving to another. This objective function is very non-submodular and so cannot be achieved by submodular optimization methods, which have nice theoretical properties [24]. By defining a value function that rewards both proximity to the current camera position and proximity to partially modeled objects, we can achieve this kind of behavior.

- Especially when the robot is a kinematic chain, it is possible for there to be large regions of unseen space blocking the robot's path to an area it needs to reach. See fig. 3 for an illustration. In our framework it is possible to raise the value of frontiers in such regions over time so that seeing through them is given higher priority than trying to view unseen parts of the intended target region.

To guarantee that a view selection system will find *all* objects in a scene, map completion must be incorporated into the value function, because any unseen region of space can contain an object. However, prioritizing space by using partial information about known objects or about regions in which

objects are likely to appear (as provided, e.g., by [2]) allows us to make the best use of our time if it is known that there is a time budget but the budget is unknown.

7 Experiments



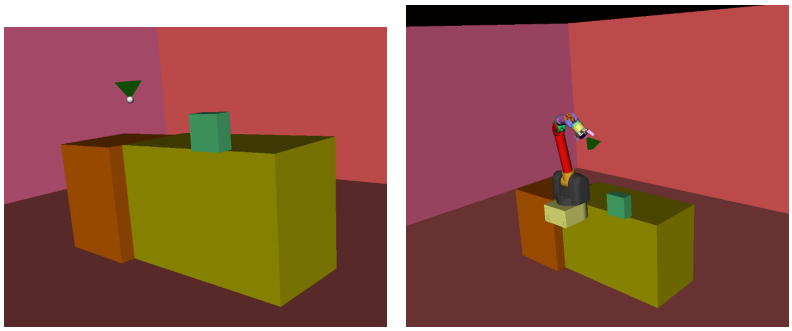
(a) SIMENV0 (b) SIMENV1 (c) SIMENV2

Figure 4: the three simulated environments SIMENV0/1/2 used in our simulation experiments. They vary only in the objects on the table.

We show three experiments: one to compare view selection algorithms, one to demonstrate the use of our framework for object completion, and one to explore differences between flying and arm-mounted cameras. Due to limited hardware availability, the experiments we present here are all in simulation. We use three similar simulated scenes, each containing a yellow and an orange table inside a room with walls, floor and ceiling that are varying shades of pink. One scene has no objects on the table, one has one object, and one has four. These scenes are shown in fig. 4. Each scene is bounded by walls so that it cannot take infinite time to complete a map. Each scene is the same size, about 4 m^2 horizontally and 2 m vertically. In two experiments we model a camera attached to a free-flying small spherical robot (an approximation to a quadrucopter). In the other, we model a camera mounted on a kinematic chain: a WAM arm [4] attached to a stationary base. Both models are shown in fig. 5.

All space in the map is initially unseen. To allow the camera to move at all, we carve out some "known" free space around the robot before starting mapping. In the case of a free-flying camera we provide a .4-meter cube of initial free space; in the case of the WAM assembly, we provide space including the entire robot and some extra space for the arm's elbow to be able to bend in its first few movements.

When running with simulated environments, we provide Patch Volumes with ground-truth alignments rather than using its alignment functionality,



(a) (b)

Figure 5: the two simulated robot models we use in our experiments. (a) a free-flying camera (technically mounted to a small white sphere); (b) a camera mounted on the palm of a WAM arm with a Barrett hand. We do not allow the base of the WAM to move. In each image the camera viewing direction is indicated with a green frustum.

because its alignment is unstable in environments with zero color or depth texture. We have verified empirically that adding a small amount of color noise to otherwise single-color-valued surfaces is enough to avoid this instability, but that tactic is unavailable in our current software framework, so for ease of implementation we avoid running alignment in simulation.

There are at least three types of termination condition we can use in map completion-related applications: we are done mapping when there are no frontiers left in the map, when we are no longer able to plan to any remaining frontier (due to limitations of a kinematic-chain robot or due to hidden surfaces in the map caused by inaccuracy in alignment), or when there are no high-value scenels left in our list of potential targets. We end an experiment when any of these conditions occurs.

In our map completion experiments, when online change detection functionality is not needed, we disable online change detection and object modeling to better evaluate view selection. The object completion experiment requires change detection to identify surfaces of interest. Therefore runtime numbers from these different experiments are not comparable.

7.1 Map Completion with a Free-Flying Camera

As a first step toward active segmentation, we evaluate our system on the simpler problem of map completion in a simulated environment (SIMENV1).

This will help us choose a view selection algorithm to modify for the purpose of active segmentation. We use a free-flying camera, whose robot configuration consists of its pose, represented as a quaternion and 3-D translation. We compare the performance of four view selection algorithms on this problem:

- RANDOM-FRONTIER assigns a random score in $[0, 1]$ to each frontier triangle. Holz et al. [21] refer to this algorithm as "random frontier".

- DISTANCE-TO-CAM assigns a score U based on the distance between the camera pose at the start of planning and the suggested camera pose X viewing each frontier triangle:

$$U_d(X) \quad = e^{-D_p(X)}, \tag{1}$$

  where $D_p(X)$ gives the pose distance between the camera pose at the start of planning and camera pose X, calculated as the Euclidean distance between the translations in meters plus 0.5 times the angular distance between the rotations in radians. We refer to this pseudo-unit as "meter-equivalents".

  We could equally well get the distance numbers into the $[0, 1]$ range by dividing by the maximum distance observed rather than by exponentiating. Either would result in the same ranking of scores.

- INFO-GAIN assigns a score based on an approximation to the expected information gain of the suggested next view over variables representing the state of each voxel that would be seen. The current scene, including frontiers, is rendered from each suggested view X and the number of pixels $N(X)$ in the hypothetical image that would see currently unseen space is counted. The score is then

$$U_{ig}(X) \quad = \frac{N(X)}{\max_{X \in \mathcal{X}} N(X)}, \tag{2}$$

  where $\mathcal{X}$ is the set of camera poses being evaluated. This is similar to the evaluation function of Blodow et al. [5], with the difference that since we update the map at each frame, we have no need to include a measure of map overlap in the evaluation function. The other main difference between these two algorithms is that we don't sample our suggested camera poses randomly.

- TRAVERSAL-COST measures the approximate traversal cost for the camera. The score of suggested pose X is

$$U_{tr}(X) \quad = e^{-D_t(X)}, \tag{3}$$

where $D_t(X)$ is the approximate traversal cost from the current camera pose to camera pose X. Distance is calculated as for DISTANCE-TO-CAM. We approximate the traversal cost similarly to how Wettach et al. [34] do: we create a graph whose nodes are the centers of a voxel grid overlaid on the map (for speed, we use a grid with larger resolution than that used for mapping) as well as the current and all suggested future camera positions. We then compute shortest paths between all pairs of nodes. The distance function we use between nodes takes both translational and rotational distance into account. (We arbitrarily assign the same rotation as that of the current camera pose to each node created from an arbitrary voxel center.) For view selection, we only need to use the distances between the current and each proposed future camera pose. Wettach et al., who want to move the robot along the resulting path, additionally use a continuous optimization to produce a smooth path; we compute paths to use in a separate step, so do not need to smooth paths here.

Due to the exponential dependence of the number of graph nodes in this approach on the configuration space dimension, we will probably need to explore an approach based on a probabilistic roadmap (PRM) once we move to higher dimensions (e.g. for our 7-dof WAM arm). In three dimensions we find this approach faster than a PRM.

- INFO-TRAVERSAL combines the information gain and traversal cost objectives additively:

$$U_{it}(X) \quad = \alpha U_{ig}(X) + (1 - \alpha) U_{tr}(X). \tag{4}$$

  We use $\alpha = .4$, an educated guess.

Table 1 shows numerical measures of the performance of map completion using each of these view selection algorithms. Each algorithm was run three times to completion. The shortest path length traversed by the camera during the whole run should intuitively be achieved by traversal-cost-based algorithms, and here INFO-TRAVERSAL and TRAVERSAL-COST in fact do achieve the shortest paths. DISTANCE-TO-CAM achieves the smallest runtime to map completion. Multiple algorithms have very high rates of planning success, meaning they successfully avoid selecting next views that are hard to reach from the current one, which would be a negative quality in our continual-replanning framework. The quickest algorithm to run each iteration of view selection and planning is DISTANCE-TO-CAM, which is probably why it completes the map soonest. RANDOM-FRONTIER has a quicker

	Algorithm	Runtime (s)	Distance traveled (m)	# View targets	Planning success	Avg update time (s)
run 1	RANDOM-FRONTIER	6508	663.3	2685	45%	1.4
	DISTANCE-TO-CAM	795	89.9	353	98%	1.4
	INFO-GAIN	19379	120.5	487	90%	25.6
	TRAVERSAL-COST	4127	89.3	296	98%	13.7
	INFO-TRAVERSAL	7581	61.3	224	4%	33.6
run 2	RANDOM-FRONTIER	3997	426.7	1670	51%	1.0
	DISTANCE-TO-CAM	950	112.0	422	95%	1.0
	INFO-GAIN	50894	260.5	1049	3%	38.8
	TRAVERSAL-COST	3200	74.5	240	76%	12.6
	INFO-TRAVERSAL	11792	78.6	288	5%	40.7
run 3	RANDOM-FRONTIER	5703	592.3	2368	62%	1.0
	DISTANCE-TO-CAM	998	108.6	418	12%	1.2
	INFO-GAIN	30740	193.5	762	4%	30.0
	TRAVERSAL-COST	3641	83.7	272	27%	13.2
	INFO-TRAVERSAL	8999	53.2	181	2%	48.3

Table 1: comparison of view selection algorithms on the map completion problem on SIMENV1. Runtime is in many cases the most important statistic. Distance traveled may also be important depending on hardware. Planning failure is almost always due to collisions with obstacles, and occasionally due to failure to find a configuration reaching the initial camera pose for the plan. (In the case of a kinematic-chain robot, this second case is inverse kinematics failure.) "Update time" refers to time spent per run of view selection and motion planning. This includes runs in which there is a current view target and no view selection is necessary; such runs can be very fast. Each successive view target is reached after planning one or more times.
Each run of each algorithm represents a separate process.

Algorithm	Runtime (s)	Distance traveled (m)	# View targets	Planning success	Avg update time (s)
RANDOM-FRONTIER	6138	638.5	2550	38%	1.1
DISTANCE-TO-CAM	994	118.4	432	38%	1.1
INFO-GAIN	42523	246.1	975	16%	27.6
TRAVERSAL-COST	2807	61.3	209	64%	13.2
INFO-TRAVERSAL	31852	176.9	684	1%	46.4

Table 2: comparison of view selection algorithms on the map completion problem on SIMENV2. See tbl. 1 for explanation of metrics.

view selection step than DISTANCE-TO-CAM, but has a higher percentage of failed plans, meaning it must replan more times for each view target update.

Table 2 shows numerical results for map completion on SIMENV2, which has qualitatively different topology from SIMENV1 (there is an object resting on two others, creating a hole in the surface). Each algorithm was run once to completion. The runtime numbers are larger than for the less complicated scene SIMENV1, and planning success rates are lower. On this scene

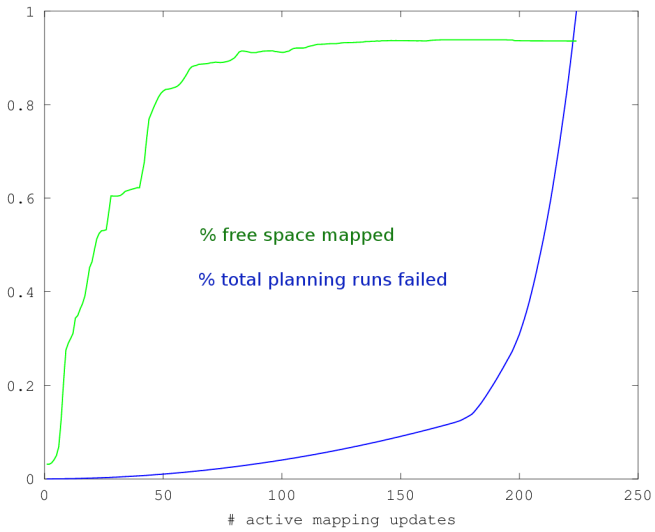


Figure 6: map completion (in terms of observed volume) and total planning success rate over time for INFO-TRAVERSAL. As shown in tbl. 1, the planning failure rate is 96% at the end of the run; here it appears to be 100% because these numbers include successes as failures. (Each update ends with one planning success.) Qualitative conclusions from this graph are still valid.

TRAVERSAL-COST has the shortest path to completion. It is still the case that DISTANCE-TO-CAM has a shorter update time than any of the other non-random algorithms.

INFO-TRAVERSAL achieves a very short path to view the whole scene, but has a very low planning success rate. Fig. 6 shows the completeness of the map produced for SIMENV1 (taken from fig. 7 graphed with the (approximate) percentage of failed planning runs after each active mapping update using INFO-TRAVERSAL. Completeness levels off while planning failure becomes more and more frequent over time. One plausible explanation for this behavior is that when most of the volume has been observed, when (we see empirically) there are usually many small unobserved areas left in far-flung regions of the scene, the information gain objective conflicts with the traversal cost objective and faraway areas of the scene containing relatively large unseen regions dominate the top of the list of potential next views. It is possible that the tradeoff parameter α in the value function would be best changed over time during mapping; this would be an interesting future

experiment. Another possibility is that running collision checking before we limit the length of a path has more negative an effect on view selection methods that suggest views anywhere in the scene than it does on those that prefer to suggest views near the current view. To test this hypothesis, we also ran map completion on SIMENV2 using INFO-TRAVERSAL with collision checking run only after path length limiting, to reduce checking of collisions with parts of the map that are not actually traversed until later. This run had a similar planning success rate of 3%, suggesting that overzealous collision checking is not a major cause of planning failures. (This discussion is likely to also apply to INFO-GAIN, which also has a low planning success rate on most runs.)

We can also look at how quickly each algorithm reduces the size of the remaining free space. All these algorithms eventually view all or almost all surfaces in the map, but they do so at different rates. In fig. 7 we show how quickly each algorithm reduces the free space in the scene. We show completion as a function of iteration count rather than of time because each of these algorithms could be tweaked to be faster or slower; it's not at all clear that our implementation of each is the best/most efficient way to implement it, and we want to offset the table above in which we emphasize runtime. For example, both INFO-GAIN and TRAVERSAL-COST could be made faster by greatly limiting the number of options for next viewpoint, as most previous work using those algorithms has. In particular, most of the time in the information-gain-based update is spent in rendering the scene from many viewpoints. However, algorithms like DISTANCE-TO-CAM that use only information local to a mesh triangle to score that triangle, as opposed to spatially dispersed information, allow for considering many view options quickly. This makes the algorithm fit our framework, which is designed for use with efficient view selection and planning, well. Therefore we choose to build on DISTANCE-TO-CAM as a view selection method for active segmentation purposes.

Fig. 8 shows a view of a map of SIMENV1 made with DISTANCE-TO-CAM running on a free-flying camera. There are no unobserved surfaces in this map.

7.2 Map Completion with a Robot Arm

For comparison, we run some view selection algorithms on the map completion problem on the SIMENV1 and SIMENV2 environments, as above, but using a kinematic-chain robot model. This robot's configuration consists of the angles of seven rotational joints. We use the OpenRAVE inverse

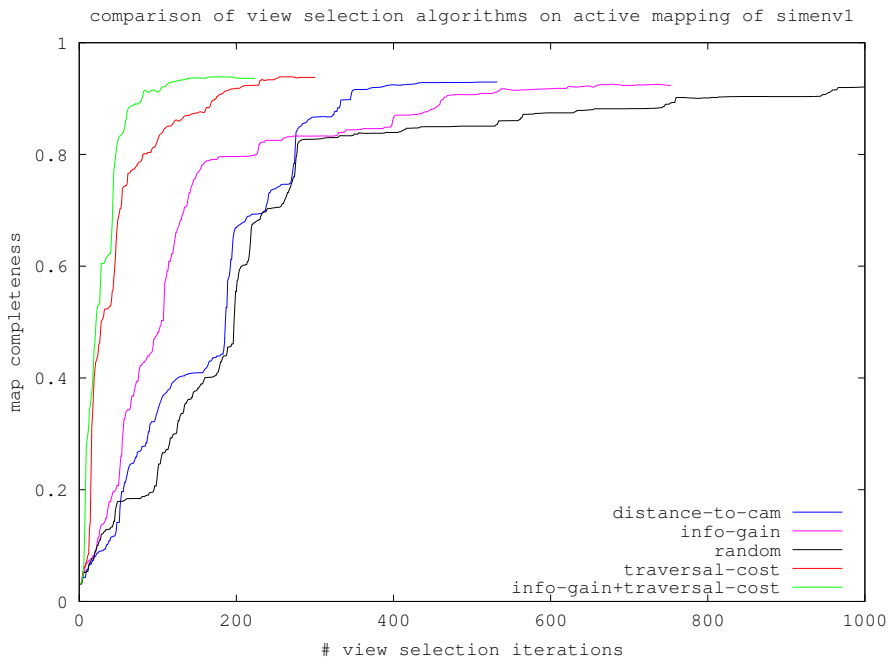
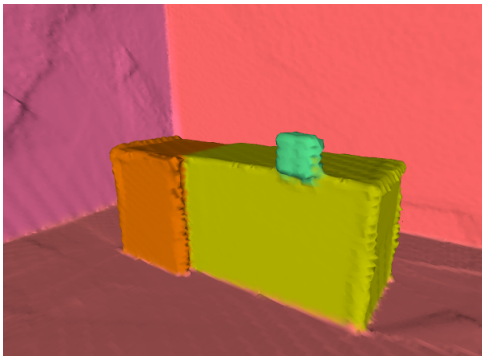


Figure 7: completeness of the map as a function of number of active mapping updates executed on SIMENV1. Completeness is measured by free space in the scene that is known free in the map. The x axis is update iterations, which unit is not a linear function of time. The 100% mark (the top of the graph) is, for computational reasons, calculated in a way that slightly overshoots what any algorithm can actually achieve, so although the graph shows each algorithm achieving only about 95% completeness in terms of volume, each algorithm does in fact view all the surfaces in the scene.

kinematics module, `ikfast`, to implement reachability and collision testing. `ikfast` gives us correct reachability results well over 99% of the time (but not 100%).

We show results for the DISTANCE-TO-CAM and INFO-GAIN view selection algorithms in tbls. 3 and 4. We selected these two algorithms in order to compare a method using only local to one using global scene information on this robot model. As in the earlier experiment, the local method has better runtime, which is the single statistic we are most interested in in this case, as runtime is a limiting factor due to the mapping speed limitations discussed above.



(a)

Figure 8: a map of SIMENV1 made with the DISTANCE-TO-CAM view selection algorithm running on a simulated free-flying camera.

	Algorithm	Runtime (s)	Distance traveled (m)	# View targets	Planning success	Avg update time (s)
run 1	DISTANCE-TO-CAM	1025	89.5	270	17%	3.7
	INFO-GAIN	22628	175.9	562	12%	39.4
run 2	DISTANCE-TO-CAM	1307	119.4	340	20%	3.8
	INFO-GAIN	28464	192.3	565	11%	50.3
run 3	DISTANCE-TO-CAM	1160	99.8	304	32%	3.7
	INFO-GAIN	31820	228.2	722	9%	44.0

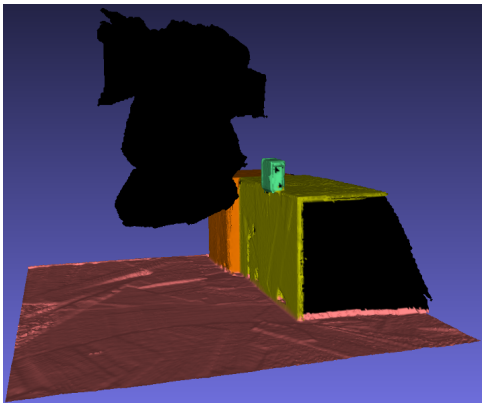
Table 3: comparison of the closest-frontier and information-gain view selection strategies on the map completion problem on SIMENV1 with a simulated kinematic-chain robot. Each view selection algorithm was run three times to completion. (Each run of each algorithm represents a separate process.)

Algorithm	Runtime (s)	Distance traveled (m)	# View targets	Planning success	Avg update time (s)
DISTANCE-TO-CAM	1002	92.6	267	25%	3.6
INFO-GAIN	16062	138.3	435	12%	36.8

Table 4: comparison of the closest-frontier and information-gain view selection strategies on the map completion problem on SIMENV2 with a simulated kinematic-chain robot. Each view selection algorithm was run once to completion.

There are a few differences between these results and those for the free-flying camera model. Runtimes are greater for the kinematic chain, probably because reachability and collision checking are more complicated than for a non-articulated robot. Each algorithm has much lower planning success with the kinematic chain than with the free-flying model; this may be due to the limitations of linear interpolation of joint angles as a planning method.

We have used linear interpolation in previous experiments (not described in this document), and it has worked well, but there we have mainly used it for motions through space that is known to be free or assumed to be easily navigable.



(a)

Figure 9: a cutout of a map of SIMENV1 made by active mapping running on a simulated WAM arm (with the DISTANCE-TO-CAM view selection algorithm). Unseen regions of space are shown in black. The large irregularly shaped unseen area is the region in which the robot is located (its base does not move) and which the camera is unable to see through. The end of the table is also unseen because the arm is unable to move the camera to look at that surface.

Fig. 9 shows part of a map of SIMENV1 made with DISTANCE-TO-CAM running on the simulated WAM. There are large areas the camera is unable to see because of robot arm movement limitations.

7.3 Object-Centric Map Completion

Now we consider the problem of object model completion, one way to do active segmentation. In this experiment we wish to model known objects in the scene as fully as possible without manipulating them, and without mapping more of the scene than is necessary. Our view selection framework allows us to design an algorithm for this problem using previously acquired partial information about object boundaries in the scene. To demonstrate the usefulness of using partial segmentation information to inform view selection for the object model completion problem, we compare two view selection algorithms. We again model a camera mounted on a free-flying robot. We

first make a map of SIMENV0 using map completion, and use it as the background map against which SIMENV2 was compared via change detection to identify surfaces that have changed. As a baseline, we ran map completion using the DISTANCE-TO-CAM view selection algorithm. To improve results, we propose a change-detection-based view selection algorithm, denoted COMBINED-CHDET-CAMDIST, which computes view scores for unseen triangles as a combination of distance to camera pose and distance to likely-changed surfaces (defined by having $p(m) > .6$, although the exact threshold is not critical):

$$U(X) \quad = e^{-\min(D_m(t(X)),10)-.4D_p(X)}, \tag{5}$$

where $t(X)$ denotes the triangle that pose X is computed in order to view, and $D_m(t)$ gives the geodesic distance on the scene mesh between triangle t and the nearest likely-changed triangle. (This metric fails when there are unseen regions of space disconnected from the surfaces in the map. This is not a problem for this experiment because disconnected unseen regions are likely to be far from any surfaces, as the camera has to have seen between the two, and here we are concerned with unseen regions close to particular surfaces.) The maximum distance (here 10) is used only to avoid underflow in the result of the exponentiation.

Change detection information is aggregated during mapping in the form of a voxel grid representing the current scene. At each frame processed by mapping, the result of change detection between the current frame and the background map is projected into this grid. We only project information for pixels that have valid depth values in the frame. For each such pixel p, whose depth in the frame is z with standard deviation σ_z, we copy per-pixel change detection results to each voxel that projects to p with a depth $d \in [z - 2\sigma_z, z + 2\sigma_z]$. Because we use a voxel grid for this operation rather than a set of frames, as we do in [17], the aggregation operation is constant-time rather than becoming slower as the video being recorded becomes longer.

As a score measure, we use the count of frontier mesh triangles that are within 3 cm geodesic distance of any likely-changed surface in the map being built of the current scene (here SIMENV2). The more frontier triangles are very close to likely-changed triangles, the more interesting surfaces in the scene have not been thoroughly explored yet. The number of such frontier triangles may fluctuate during a run but should eventually be reduced to a very small number (zero, up to map irregularities that occur in real

environments). The more quickly this happens, the better the view selection algorithm is at prioritizing completion of partially seen objects. Once the last such triangle is observed, the robot is no longer mapping partially known objects, and until more objects are discovered, mapping proceeds as in vanilla map completion, or, if the goal of mapping is specifically to complete known objects, mapping can stop.

We ran each algorithm once, as they are both deterministic modulo concurrency effects. The initial camera pose observed parts of all the moved objects in the scene, so that all surface regions of interest initially bordered frontiers. Both algorithms did eventually remove all frontiers near the moved objects. DISTANCE-TO-CAM observed the last frontier triangle near a moved surface at the 3950th mapping iteration, 3788 seconds into execution. The number of frontier triangles near moved surfaces was 213 before the first camera movement, reached a peak of 4074, and did not dip below 213 for the final time until 4351 seconds into execution. COMBINED-CHDET-CAMDIST performed much better. It observed the last frontier triangle near a moved surface at the 796th mapping iteration, 775 seconds into execution. The number of frontier triangles near moved surfaces was 335 before the first camera movement, and stayed under 300 for the rest of the run. (As mentioned earlier, these runtime numbers are larger than those in previous experiments because this experiment required enabling the online change detection functionality in the online mapping system, which slows mapping and thus decreases the speed at which the camera can move.)

8 Practical Considerations

We now discuss some practical points in implementing an active mapping system and in using it on a real robot.

Some surfaces (especially transparent surfaces and shiny black surfaces) don't return readings to an infrared depth camera such as we use. The map used during active mapping must include a representation of regions that are unseen due to not returning readings rather than due to not having been viewed yet. We store the number of times each quantized view (quantized view target position and orientation) has been chosen as a target, and avoid selecting targets in a single quantization bin more than $n = 2$ times.

A camera on a robot arm will point in various directions while it is being moved between targets, due to noncontinuousness of joints in the arm. The mapping system needs to be robust enough to work with long intervals of camera views of ceilings, walls and other relatively untextured

surfaces. This may require either disabling mapping during long movements or integration of arm odometry into the mapping system. Patch Volumes currently only uses visual information. Luckily, unlike in next-best-view planning (e.g. [5]), due to our continuous map aggregation we don't also have the related problem of making sure that each new view sufficiently overlaps existing ones. However, figuring out how to accurately map with general camera motions is still future work.

The simple planners we use in our continual-replanning framework are not guaranteed to be able to produce paths. If we have an almost complete map and there are only a few widely scattered unseen areas left, we may need a sophisticated planner that is deterministically or probabilistically complete, such as an RRT, to guarantee that we can continue to find paths to remaining unseen regions. In general we have a long list of possible view targets at any given time, and we rely on being able to plan to one of the first few targets suggested in order for view selection to be fast, so to integrate a fallback planner with our framework without making it unusably inefficient, we might have to reduce the set of potential view targets. This has not been a major problem in practice so far: most of our experimental runs so far have completed the maps in question despite only having a simple planner available.

In order to communicate effectively with robotic hardware, our framework uses information about the expected runtime of both an iteration of mapping and an iteration of view selection plus planning. In particular, if we expect view selection and planning to last T_{plan} seconds and each iteration of mapping to last T_{map} seconds, and we want to avoid moving the camera more than D_{map} meter-equivalents (see 7 for a definition of this unit) per mapping iteration, we ensure that no plan (which is intended to be run while the next iteration of view selection and planning happens) is of length more than

$$D_{planmax} \quad = T_{plan} \frac{D_{map}}{T_{map}}. \tag{6}$$

At present all these numbers are hardcoded. We estimate $T_{plan} = 100$ seconds for INFO-GAIN and 1 second for all other algorithms, as INFO-GAIN is so slow that the simulation idles most of the time when a small estimate is used, and we use $T_{map} = 1$ second from empirical experience. However, any of these quantities could be continually reestimated at runtime. Regardless of the source of this information, it generally won't be perfect, although the more deterministic the planner is, the better our runtime estimate for it will

be. It may also be possible to make the system more sophisticated, so that the motion execution module continually checks whether planning is not complete and, if not, sends only a very small part of the current trajectory to be executed, rather than sending the entire output of planning at once.

9 Summary and Future Work

One of the major benefits of using mobile robots for vision is that a robot can choose how to explore its environment, which a passive vision algorithm cannot. In this report we have presented a framework for continual view selection and motion planning in an online active mapping framework, in which the robot can move the camera. We assign a value to each scene element that represents a frontier area in the 3-D map, and prioritize planning to scenels with high value. We use a simple and fast planner in order to keep the camera in motion as much of the time as possible. We use this framework for the map completion problem as well as for a variant emphasizing completing models of previously partly seen objects, which is useful for active object segmentation.

Our planning so far has encompassed kinematic-chain robots, such as robot arms, and free-flying robots (an approximation to autonomous helicopters), but has not included robots with both wheels and arms. Planning for these is very computationally inefficient [23]. It may be that an online-updated probabilistic roadmap is the most useful avenue of attack for bringing motion planning for such a robot into our framework, which assumes that basic actions such as planning are relatively efficient and can be run often.

The operation of removing occupied volume from the background map that is used in [15] might be better done volumetrically, as opposed to the current rendering-based approach, which throws away some information about free space during the surface removal operation. We have just introduced another operation that is best done in a volumetric map: aggregating change detection results over time, which we use to help define some value functions, is more efficiently done in a fixed-size voxel grid than in a set of images. Each image is much smaller than the voxel grids we use for mapping, but that representation increases in size as new frames are processed. For now we implement change detection aggregation in our own volumetric representation on the CPU, due to Patch Volumes being closed-source, but ideally both these operations, and probably others, would be implemented as part of a GPU volumetric mapping API.

References

[1] N. Atanasov, B. Sankaran, J. Ny, T. Koletschka, G. Pappas, and K. Daniilidis. A hypothesis testing framework for active object detection. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2013.

[2] A. Aydemir and P. Jensfelt. Exploiting and modeling local 3-d structure for predicting object locations. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2012.

[3] R. Bajcsy. Active perception. Technical report, Penn, 1988.

[4] Barrett, Inc. Wam^{TM} arm. `http://www.barrett.com/robot/products-arm.htm`.

[5] N. Blodow, L. Goron, Z. Marton, D. Pangercic, T. Ruehr, M. Tenorth, and M. Beetz. Autonomous semantic mapping for robots performing everyday manipulation tasks in kitchen environments. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2011.

[6] C. Connolly. The determination of next best views. In *IEEE International Conference on Robotics and Automation (ICRA)*, 1985.

[7] PrimeSense corporation. Primesense. `http://www.primesense.com/`.

[8] B. Curless and M. Levoy. A volumetric method for building complex models from range images. In *ACM Transactions on Graphics (Proc. of SIGGRAPH)*, 1996.

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

[10] H. Du, P. Henry, X. Ren, D. Fox, D. Goldman, and S. Seitz. Interactive 3d modeling of indoor environments with a consumer depth camera. In *International Conference on Ubiquitous Computing (UbiComp)*, 2011.

[11] H. Durrant-Whyte. Consistent integration and propagation of disparate sensor observations. In *IEEE International Conference on Robotics and Automation (ICRA)*, 1986.

[12] D. Golovin and A. Krause. Adaptive submodularity: Theory and applications in active learning and stochastic optimization. *Journal of Artificial Intelligence Research (JAIR)*, 2011.

[13] H. Gonzalez-Baños and J. Latombe. Navigation strategies for exploring indoor environments. *International Journal of Robotics Research (IJRR)*, 2002.

[14] P. Henry, D. Fox, A. Bhowmik, and R. Mongia. Patch volumes: Segmentation-based consistent mapping with rgb-d cameras. In *International Conference on 3-D Vision (3DV)*, 2013.

[15] E. Herbst, P. Henry, and D. Fox. Toward online 3-d object segmentation and mapping. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2014.

[16] E. Herbst, P. Henry, X. Ren, and D. Fox. Toward object discovery and modeling via 3-d scene comparison. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2011.

[17] E. Herbst, X. Ren, and D. Fox. Rgb-d object discovery via multi-scene analysis. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2011.

[18] E. Herbst, X. Ren, and D. Fox. Object segmentation from motion with dense feature matching. In *ICRA Workshop on Semantic Perception, Mapping and Exploration*, 2012.

[19] E. Herbst, X. Ren, and D. Fox. Rgb-d flow: Dense 3-d motion estimation using color and depth. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2013.

[20] G. Hollinger, B. Englot, F. Hover, U. Mitra, and G. Sukhatme. Uncertainty-driven view planning for underwater inspection. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2012.

[21] D. Holz, N. Basilico, F. Amigoni, and S. Behnke. Evaluating the efficiency of frontier-based exploration strategies. In *41st International Symposium on Robotics and 6th German Conference on Robotics (ISR/ROBOTIK)*, 2010.

[22] D. Holz, M. Nieuwenhuisen, D. Droeschel, J. Stckler, A. Berner, J. Li, R. Klein, and S. Behnke. Active recognition and manipulation for mobile robot bin picking. In *Gearing up and accelerating cross-fertilization between academic and industrial robotics research in Europe - Technology transfer experiments from the ECHORD project (v94 of Springer Tracts in Advanced Robotics)*. 2014.

[23] A. Hornung, M. Phillips, E. Jones, M. Bennewitz, M. Likhachev, and S. Chitta. Navigation in three-dimensional cluttered environments for mobile manipulation. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2012.

[24] S. Javdani, M. Klingensmith, J. Bagnell, N. Pollard, and S. Srinivasa. Efficient touch-based localization through submodularity. Technical Report CMU-RI-TR-12-25, CMU Robotics Institute, 2012.

[25] M. Krainin, B. Curless, and D. Fox. Autonomous generation of complete 3-d object models using next-best-view manipulation planning. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2011.

[26] H. Kress-Gazit. Robot challenges: Toward development of verification and synthesis techniques. *Robotics and Automation Magazine*, 2011.

[27] S. LaValle. Rapidly exploring random trees: a new tool for path planning. Technical Report TR 98-11, Iowa State, 1998.

[28] W. Lorenson and H. Cline. Marching cubes: A high-resolution 3-d surface construction algorithm. In *ACM Transactions on Graphics (Proc. of SIGGRAPH)*, 1987.

[29] C. Potthast and G. Sukhatme. A probabilistic framework for next best view estimation in a cluttered environment. *Journal of Visual Communication and Image Representation (JVCI)*, 2013.

[30] S. Shen, N. Michael, and V. Kumar. Autonomous indoor 3-d exploration with a micro-aerial vehicle. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2012.

[31] R. Smith and P. Cheeseman. On the representation and estimation of spatial uncertainty. *International Journal of Robotics Research (IJRR)*, 1987.

[32] S. Thrun, W. Burgard, and D. Fox. *Probabilistic Robotics*. MIT Press, 2005.

[33] R. Triebel, B. Frank, J. Meyer, and W. Burgard. First steps toward a robotic system for flexible volumetric mapping of indoor environments. In *IFAC Symposium on Intelligent Autonomous Vehicles*, 2004.

[34] J. Wettach and K. Berns. Dynamic frontier-based exploration with a mobile indoor robot. In *41st International Symposium on Robotics and 6th German Conference on Robotics (ISR/ROBOTIK)*, 2010.

[35] B. Yamauchi. A frontier-based approach for autonomous exploration. In *IEEE International Symposium on Computational Intelligence, Robotics and Automation*, 1997.

[36] M. Zillich, J. Prankl, T. Moerwald, and M. Vincze. Knowing your limits: Self-evaluation and prediction in object recognition. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2011.

[37] M. Zucker, N. Ratliff, A. Dragan, M. Pivtoraiko, M. Klingensmith, C. Dellin, J. Bagnell, and S. Srinivasa. CHOMP: Covariant hamiltonian optimization for motion planning. *International Journal of Robotics Research (IJRR)*, 2013.