# Robotic Object Detection: Learning to Improve the Classifiers Using Sparse Graphs for Path Planning

**Zhaoyin Jia**
School of Electrical and Computer Engineering
Cornell University
zj32@cornell.edu

**Ashutosh Saxena**
Department of Computer Science
Cornell University
asaxena@cs.cornell.edu

**Tsuhan Chen**
School of Electrical and Computer Engineering
Cornell University
tsuhan@ece.cornell.edu

## Abstract

Object detection is a basic skill for a robot to perform tasks in human environments. In order to build a good object classifier, a large training set of labeled images is required; this is typically collected and labeled (often painstakingly) by a human. This method is not scalable and therefore limits the robot's detection performance.

We propose an algorithm for a robot to collect more data in the environment during its training phase so that in the future it could detect objects more reliably. The first step is to plan a path for collecting additional training images, which is hard because a previously visited location affects the decision for the future locations. One key component of our work is path planning by building a sparse graph that captures these dependencies. The other key component is our learning algorithm that weighs the errors made in robot's data collection process while updating the classifier. In our experiments, we show that our algorithms enable the robot to improve its object classifiers significantly.

## 1 Introduction

Object detection (e.g., [Meger *et al.*, 2008; Forssén *et al.*, 2008; Leibe *et al.*, 2008; Felzenszwalb *et al.*, 2008; Sapp *et al.*, 2008]) is an essential component for performing several robotic tasks, including object fetching [Kemp *et al.*, 2008; Li *et al.*, 2011] and door opening [Meeussen *et al.*, 2010; Sturm *et al.*, 2010; Klingbeil *et al.*, 2008]. More generally, reliable object detection would enable any robot with a camera to be useful in a variety of other scenarios. In this paper, we propose an algorithm that allows a robot to actively collect more images for improving its detectors during its training phase. Specifically, it plans and executes a path that maximizes its utility of collecting images from more views.

In the area of computer vision and robotics, a lot of research has been done on active vision [Laporte and Arbel, 2006; Laporte *et al.*, 2004; Denzler and Brown, 2002; Jia *et al.*, 2010], where a robot actively selects the next best view, and uses pre-trained classifiers to perform multi-view recognition. However, the goal of this work is different. In our work, the robot's task is to **actively build** the object classifiers during the **training** time. Our work is motivated by active learning, where the algorithm selects the next data-point to be labeled and uses that for training. Our proposed algorithm also selects a path where the robot takes the new views, and then uses these views for training the classifiers.

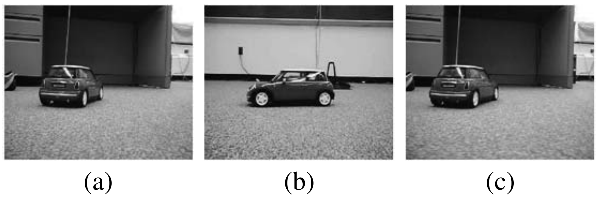


Figure 1: Our algorithm enables a robot to collect more images to re-train the object detector. If the robot has already seen the object in view (a), then collecting the image from view (b) is a better plan than collecting it from (c). That is, the view (a) and (c) are dependent on each other.

For a robot to perform object detection in common home and office environments, existing online datasets are not good enough because of two reasons. First, they may not have enough data-points for that object type or that specific object. Second, online images may be available for some canonical views, but in practice a robot may see the object from any arbitrary view. Objects can appear drastically different from different viewing angles. For example, a car can become completely different between the side view and the front view. Therefore, we believe that by actively collecting more data in its environment and learning how to use it, a robot can significantly improve the detection performance.

Two questions arise while actively building the object classifier: **where** to take the images and **how** to update the object model. For the first question, we note that not all additional views are equally useful. New training images are dependent on each other, and observing from one view may make future observations from other views redundant (see Fig. 1). To address this, we build a sparse graph that represents these dependencies and present an efficient algorithm to plan a path. Also we consider additional challenges such as: the robot may be able to observe multiple objects in the same view, some obstacles may prohibit the robot from moving into certain areas, and other obstacles may block the view.

The second question of "how" to update the classifier is hard, because in order to use the collected data, a robot first needs to separate the object from the background. To label the object in the new training images, we use a Hidden Markov Model (HMM) to combine the evidences from tracking subsequent views with the beliefs from the existing classifiers. These labels may still be noisy, and therefore they are used with a grain of salt—in the re-training phase these labels are weighted to represent how good the algorithm thinks the new data-points are.

In our experiments for eight objects in six different obstacle scenarios, we show that the robot improved its classifiers significantly by using our algorithm.

## 2 Related Work

In the field of robotic active recognition, a robot is asked to recognize the object in as few actions as possible, or attain a high recognition accuracy in early steps (e.g., [Laporte and Arbel, 2006; Laporte *et al.*, 2004; Denzler and Brown, 2002; Meger *et al.*, 2010]). It is usually achieved by selecting the next view based on measurement related to entropy. In our work, however, *the robot's goal is not to detect the current object at hand, but rather to use that object in order to improve its model for* **future** *classifications*. This is a quite different goal from previous literature and requires a different algorithm. In our problem, we would like the image set to cover all the faces of the object, whereas for active recognition the robot's future action is usually evaluated based on the current ambiguity.

In the sensor placement problem [Krause *et al.*, 2008a; 2008b], the goal is to place sensors so that the errors in future measurements are minimized. Given a map, they measure the dependencies between any two locations in a Gaussian Process and calculate the near-optimal sensor placement. At first blush, our problem may seem to resemble the sensor placement problem. However, there are important distinctions. First, multiple views become important in getting the new training images. We aim to maximally cover the different views of the object, and this is a quite different objective. Second, we prefer capturing multiple objects in one image to improve the efficiency. Third, we need to output a (preferably short) path that a robot can follow without hitting obstacles, and this makes the problem more involved.

There is a large body of work in path-planning, where the goal is to find a path with minimum cost that avoids the obstacles. [LaValle, 2006] presents an overview of path planning. In our work, since the cost of a path is dependent on the previous locations where the robot took the images, the standard path planning algorithms do not apply.

Our ultimate goal is to improve object detection for robotics. There are a lot of researches that develop object detection algorithms for different robotic applications, such as [Meger *et al.*, 2008; Forssén *et al.*, 2008; Klingbeil *et al.*, 2010; Ekvall *et al.*, 2007]. In particular, [Meger *et al.*, 2008; Forssén *et al.*, 2008] build an attentive search robot that actively recognizes multiple objects in the environment. [Klingbeil *et al.*, 2010] focuses on a specific scenario and the robot can identify various buttons in the elevator. [Ekvall *et al.*, 2007] combines the object recognition task with the 3D mapping problem. Most of these approaches follow a conventional process: as the first step a human-labeled training set is used to train a classifier for the respective application. The classifiers are fixed once learned. In contrast, our method updates the classifier by making the robot actively collect new training data. There is also some work in the area of online learning, where a robot keeps gathering more data in order to improve its performance. For example [Sofman *et al.*, 2006] proposes the online learning algorithms for robot navigation. In our case, however, we are specifically considering the properties of the image data and modeling the inaccuracies in collecting the data for object detection.

## 3 Overview

Our approach consists of the following three steps:

1. Model the importance of different views by a "utility" function. We want to take training images from locations that efficiently cover different views of the object.
2. Plan a path for the robot to take new training images. New images are dependent on each other for re-training the classifier. We model this dependency through a sparse graph. We then plan a path for the robot that maximizes the total utility gain while keeping the total length of path small and also avoiding obstacles.
3. Use the collected images to update the classifier. We first generate the labels using an HMM that combines tracking and detection beliefs. We then use this labeled data for training our soft-margin classifier, which takes this noise in the labeling into account.

## 4 Utility function

In order to plan a path for collecting new training images, we first define a "utility" function that measures the importance of observing the object from different views.

One property that we need to model is that the utility changes after each observation because of the dependencies between the views. I.e., if the robot has already observed the object from one view, then observations from other similar views should give less utility.

More formally, we define the utility function $u$ over the 2D map as follows. For an object $i$, let the remaining utility at time $t$ be $u_i^t$. An observation from the robot will provide some information of this object. We define this information as a utility gain, $f_i^t$. Considering all the objects, the total utility at time $t$ is $u^t = \sum_i u_i^t$, and the total gain is $f^t = \sum_i f_i^t$. At time $t+1$, the remaining utility of the object decreases because of the previous observation (i.e., there is no additional utility in observing it from the same viewpoint):

$$u_i^{t+1} = u_i^t - f_i^t$$

If the robot has observed the object from every possible viewpoint, then the remaining utility will become zero.

The initial utility, $u_i^0$, is an isotropic distribution in the angular coordinate with respect to the object center, i.e. we assume that each view is equally useful for future training. In the radius coordinate we set a peak indicating the ideal distance for observation. The utility will decrease when the robot moves away from this location. The utility map gets updated at each step. Fig. 2 shows an example of the utility distribution after one observation.

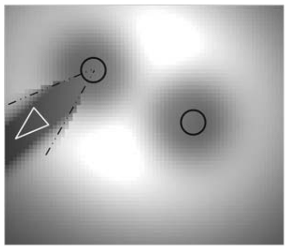


Figure 2: The utility distribution for two objects (plotted as circles) after camera observations (plotted as a triangle). The brightness indicate the utility value of the location, and the dotted lines indicate the neighborhood of the view. After the observation, the utility of the current view location is set to zero, and the utilities of the neighboring views also decrease in the remaining areas in the map.

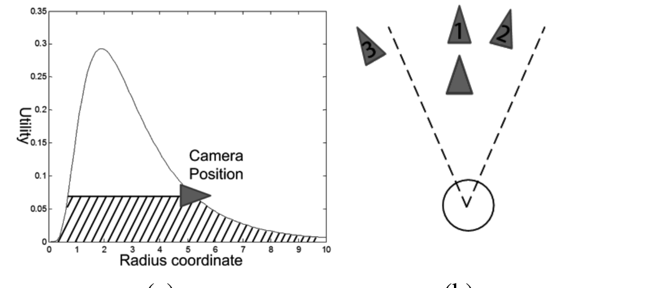
**Modeling the utility gain as a function of the viewing angle:** In the following sections we describe how to calculate the utility gain $f$ based on the utility distribution $u$. Common objects look similar when observed from similar viewing angles, and therefore there is only a small gain in utility by observing from a similar viewing angle. More formally, when the robot observes the object from view $\theta_c$, it also attains partial information of the neighboring viewpoints ($\theta_c$-$\theta_n$) to ($\theta_c$+$\theta_n$). The amount of information decreases as we deviate further from $\theta_c$, shown in Fig. 3b. We model this variation as a scaled Gaussian function, with $w(\theta) = \mathcal{N}(\theta|\mu = \theta_c, \sigma)/\mathcal{N}(0|\mu = 0, \sigma)$ as the weighting function. In this function we always have $w(\theta_c) = 1 \geq w(\theta)$, which ensures that the center view $\theta_c$ has the full utility gain, and there is no utility left at the exact same location. Therefore the utility gained $f^t$ at viewpoint $\theta_c$ is

$$f^t(\theta_c) = \int_{\theta_c-\theta_n}^{\theta_c+\theta_n} w(\theta)u^t(\theta)d\theta \tag{1}$$

**Modeling the utility gain as a function of the distance:** There is usually an optimal distance at which we get a full and detailed image of the object. If we come closer, some parts of the object may no longer be visible; if we step any farther, we may start to lose the details because the object appears smaller in the image. We model this by a log-normal function, which has a peak at the optimal value, falls off slowly as we go farther away, and goes to zero when we fail to capture the full view. (See Fig. 3 (a).) If we have observed from a particular view-point when the object is fully visible, another image farther away at this angle will gain no additional utility because it is exactly a sub-sampled version of the image closer. Therefore, our final equation is modified as:

$$f(r_c, \theta_c) = \int_{\theta_c-\theta_n}^{\theta_c+\theta_n} \int_{r=0}^{\infty} w(\theta) \min(u(r_c, \theta_c), u(r, \theta))drd\theta \tag{2}$$

Finally, we set the utility of a viewpoint to zero if the object is completely occluded by an obstacle.



(a) (b)

Figure 3: (a) Utility value for a camera position along radius coordinate. (b) Given one camera position at $\theta_c$ (in red triangular), exact the same view angle to the object (circle) should be suppressed (no utility for the camera '1'). There is less utility for similar view angle (camera '2'). Cameras (like '3') capturing different view angles should not be affected (positions outside $[\theta_c - \theta_n, \theta_c + \theta_n]$).

**Occlusion:** We also consider the obstacle in the environment in a simplified way. In this case, we draw a line from the camera location $(r_c, \theta_c)$ to the object center and check the collision along this line. If this line is collided the obstacle, then the camera is occluded and cannot contain any information about the object. We set the utility gain at this location to zero, i.e. $f(r_c, \theta_c) = 0$.

## 5 Path-planning using Sparse Graph

The goals of our path-planning algorithm are as follows.

1. The new views of the object are dependent on each other. The algorithm should collect images that maximize the total utility gained considering this dependency. We build a sparse graph to address this problem.
2. The total distance of the path should be minimized while avoiding obstacles.

Naively picking the locations with the largest utility will not solve the problem because the utility function evolves dynamically. Taking an image at one location affects optimization problems of subsequent steps. This makes an exhaustive evaluation intractable even for a relatively small problem (e.g., with a robot in just $100 \times 100$ grid).

Therefore, we build a "sparse graph" connecting points that affect each other's utility gain. We note that if the robot makes an observation at point $(x, y)$, then the utility function $u^t$ is only changed for points $(x', y')$ that are close to $(x, y)$. More formally, we sample the space randomly to obtain an initial location set $\{P_{i,i=1...N}\}$ without updating the utility map, having utility gain $f(P_i)$ for each location. $P_i$ and $P_j$ are neighbors if they are (a) facing the same object and (b) their view angle difference is within $[-\theta_n, \theta_n]$, i.e. they are in similar viewpoints of an object. This means that if the robot takes a picture at $P_i$ , then the utility gain for $P_j$, $f(P_j)$, would be changed. However,disjoint points will not affect each other.

Our next step is based on random sampling to achieve a path that is close to the global optimal. This is inspired by the Probabilistic Road Map algorithm [LaValle, 2006; Geraerts, 2006] that has been found to perform well in practice. From our sparse graph, we pick the top $M$ sample points in a way that they are not neighbors of each other. I.e., we start by picking the point with the maximum utility gain, and then ignore all its neighbors. We then pick a point with the highest utility in the remaining set and repeat the process. This ensures that the selected sample points are distributed around the map and not clustered. All of them are added to the "current" set of points $\{S_{i,i=1,\ldots,M}\}$ . These $M$ positions indicate the high utility regions in the map individually, but do not necessarily constitute a good set of locations jointly, because we have explicitly ignored the neighbors when choosing them.

In the final step, we refine our initial guess by incorporating the fact that utility gets updated after each observation. For each position $S_i$ that is selected before, we re-sample $W$ more points around it, calculate the utility associated with each point (i.e. the utility given that all the other points $S_{j,j\neq i}$ were visited and fixed), and find the best point among these $W$ points. We replace the original point $S_i$ with this new best point. Because we fix the other points, we can efficiently calculate the updated utility gain $f(P_n)$ when replacing $S_i$ with the new point [1]. We do this iteratively over all $M$ points until it converges. The algorithm is formally described in Algorithm 1.

**Algorithm 1** Path-planning Algorithm

```
Given utility $u(x, y)$, $M$, $N$
Randomly sample $N$ points, $P_{j,j=1,\ldots,N}$,. Independently calculate utility gain for each point $f(P_j)$.
**for** i=1 to M **do**
    $S_i = \max f(P_j)$
    Set $f(P_n) = 0$, $P_n$ are the neighbors to $S_i$.
**end for**
Link $S_i, i = 1\ to\ M$ to form a short path using heuristic optimization.
**repeat**
    **for** i=1 to M **do**
        sample $W$ neighbor points $P_n$ to $S_i$
        calculate distance change $D(P_n)$ and utility $f(P_n)$ given all the other $S_{j,j\neq i}$ fixed
        $V(P_n) = \lambda \times f(P_n) - D(P_n)$.
        $S_i = \arg\max_{P_n}(V(P_n))$
    **end for**
**until** converge
```

## 6 Updating the Model

After the robot finishes collecting the new training images from the computed path, it needs to figure out how to use these images for updating its object classifier.

These images are however not labeled, and automatically labeling is hard because of background clutter in real environments. We first estimate the labels for the images by combining two sources: (a) we already have a few training images that are manually labeled (such as from an online dataset). A classifier $I_{ori}$ can be built with these few training samples, and some views of the object can be labeled through $I_{ori}$. (b) Given these initial labels from $I_{ori}$, we can track the object across the collected multi-view images. Although individually each source is noisy, we propose an algorithm that robustly combines them.

### 6.1 Tracking the object using Implicit Shape Model

Different views of the same object vary significantly, and thus a simple SIFT [Lowe, 1999] matching would fail for tracking.

We follow the Implicit Shape Model (ISM) [Leibe *et al.*, 2008] to solve this problem, where we use a "feature set" to represent the target object, $O$. First we take the most likely detection in the newly acquired images using the pre-trained classifier $I_{ori}$. Within this detected region, we compute the SIFT descriptors, $C_0$ (used as the codebook), and their locations, $l_0$. They together form the initial feature set , and we save the voting of each feature to the object center, i.e the 2D-$(x, y)$-shift $s$ from the feature location $l_0$ to the object center $c$, $s = c - l_0$. We record this as a 2D-spatial distribution $d_0(O, c|C_0, l)$.

When we try to identify (i.e. track) the object in a new frame, we first compute the SIFT features along with their locations, and match them to the codebook. Matched features will vote for the new object center based on the previously 2D-spatial distribution, e.g. if the current feature $w_t^i$ at location $l_t^i$ is matched to the codebook $C^j$ with shift $s^j$, then feature $w_t^i$ will cast a vote at $l_t^i + s^j$ for the object center. We predict the new object center according to the vote, and update the feature set by including all the features contained in the predicted object region, forming the new codebook and voting distribution $d_t$.

Specifically, in image $t$ we compute the SIFT descriptors $w_t = \{w_t^i, i = 1, ..., N_t\}$ and their locations $l_t = \{l_t^i, i = 1, ..., N_t\}$ ($N_t$ is the total number of the features). The probability of identifying the object $O_t$ at location $c_t$ depends on two things: (a) distance between current features $w_t^i$ to the codebook $C^{t-1}$, modeled as $p(C^{t-1}|w_t^i)$; (b) features' voting of the object's location at $c_t$ given the current feature location $l_t^i$. It is calculated based on the previous 2D voting distribution, $d_{t-1}(O_t, c_t|C^{t-1}, l)$. The probability of the object center given the current features is:

$$p(O_t, c_t|w_t, l_t) = \sum_i d_{t-1}(O_t, c_t|C^{t-1}, l_t^i)p(C^{t-1}|w_t^i) \tag{3}$$

We choose the object center $\tilde{c_t}$ in the new image by selecting the location with the maximum probability, $\tilde{c_t} = \arg\max_{c_t} p(O_t, c_t|w_t, l_t)$. The resulting probability $P(O_t)$ represents our belief in finding the object based on tracking (we are not writing $c_t, w_t, l_t$ explicitly for brevity). Fig. 4a shows an example of a probability map, and Fig. 4b shows the object location with the maximum probability. To track the object across the views, we update the codebook $C$ and their shifts $s$ by adding new features and dropping old ones that no longer match for a few frames. We also simplify the model by assuming the $t^{th}$ tracking result only depends on

[1]Trying to balance the total distance change $D(P_n)$ and utility $f(P_n)$ after replacing $P_n$ with $S_i$, we select $P_n$ by maximizing $V(P_n) = \lambda \times f(P_n) - D(P_n)$ and replace it with $S_i$. $\lambda$ is a parameter to weigh these two factors.

the frame (t-1): $p(O_t|O_{t-1}) = p(O_t)$. (assuming the feature set is completely updated at each stage). This probability gives us a measure of the confidence in the label of the object based on tracking.

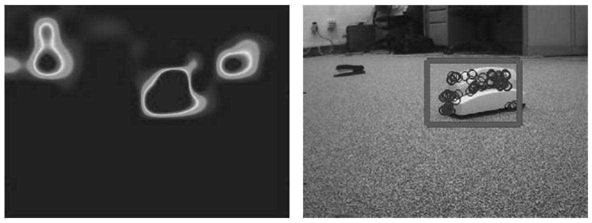
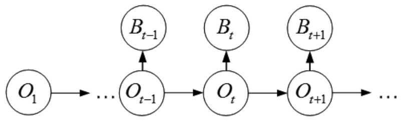


Figure 4: Use ISM to find the object. (a) the voting space after matching the feature. (b) the bounding box is generated by taking the extremes in x and y coordinates of the matched feature locations.



$$\begin{tikzcd} & B_{t-1} & B_t & B_{t+1} & \\ O_1 \arrow[r] & \cdots O_{t-1} \arrow[u] \arrow[r] & O_t \arrow[u] \arrow[r] & O_{t+1} \arrow[u] \arrow[r] & \cdots \end{tikzcd}$$

Figure 5: An HMM model to estimate the label noise.

### 6.2 Combining Detection Beliefs with Tracking Beliefs

Previously $p(O_t|O_{t-1})$ is a belief from tracking. We also get a belief $p(B_t|O_t)$ from the pre-trained detector, $I_{ori}$, where $B_t$ represents if the object is recognized in the image $t$.

The tracking is noisy and may miss the object because of occlusions—we cannot completely rely on tracking to propagate the labels. On the other hand, the original classifier $I_{ori}$ does not perform well on the new views. We combine these two beliefs by formulating this problem as a Hidden Markov Model (HMM), where $O_t$ represents the hidden state at time $t$, and $B_t$ are the observations made by running the original classifier $I_{ori}$, shown in Fig. 5. We use the Viterbi algorithm [Viterbi, 1967] to determine the probability of each state being the object, $P_{HMM}(O_t)$.

### 6.3 Re-training the model

We use Support Vector Machines (SVM) to train the detector, and incorporate the belief $P_{HMM}(O_t)$ for our new datapoints by modifying the SVM training process.

We weigh each training image by using $C_t$, which is computed using the log-odds of $P_{HMM}(O_t)$:

$$C_t = C_0 - \exp\left(-\alpha \log P_{HMM}(O_t)\right) \tag{4}$$

where $C_0$ is a set parameter for the maximum penalty on $\xi$. $\alpha$ is a parameter to ensure $C_t > 0$. A high value in $C_t$ means that we have a high believe in this training sample and only allow very small amount of error, otherwise it will result in a large penalty in minimization.

We update the final classifier in the SVM formulation as:

$$\begin{aligned} \min_{w,\xi_t,b} \quad & |w|^2 + \sum C_t \xi_t \\ \text{s.t.} \quad & y_t(wx_t + b) > 1 - \xi_t, \\ & \xi_t \geq 0 \end{aligned} \tag{5}$$

where $x_t$ is the feature for detection (we use Histogram of Oriented Gradient (HoG) [Dalal and Triggs, 2005][Felzenszwalb *et al.*, 2008] feature to do this), and $y_t$ is the training label (object(1) or non-object (-1)). We use CVX [Grant and Boyd, 2010] for this optimization.

## 7 Experiments

**Robot hardware.** We use the Rovio robot with a Vivotek pan-tilt camera mounted on it. The Rovio robot has a built-in sensor for self-localization. The pan-tilt camera produces images with a resolution of $640 \times 480$.

**Experimental setting.** We implement our algorithm on a ground robot that navigates an environment containing obstacles in different configurations. The robot collects new training images on eight objects in six different categories: two network cameras, a toy car, a digital camera, a stapler, two different telephones, and a mouse. For each object, we manually label around 10 training images to build the original pre-trained classifier $I_{ori}$. Our testing set is in a separate collection with a total of 1000 images from different views and scales, on which we compute precision-recall curves. Most of the testing images are from the novel view angle comparing to the labeled training images. A detection is a true positive if and only if the predicted bounding box has over 50% overlap with the ground truth. We use edge detection and homography (using the known height of the camera) to localize the objects on ground.

**Environment scenarios.** An environment with obstacles is a more realistic situation. Suppose a robot is going in an office to update the model of a stapler on the desk, then the wall and the desk become the obstacles in the way. We test on five settings with obstacles, and consider two types of obstacles, as shown in Fig. 6. The first one, shown in dashed lines, is a see-through obstacle (e.g., a table)—the robot can see through these obstacles, however it cannot get past or walk into them. The second one (e.g., walls) is shown as filled rectangles, and the robot can neither see through nor walk through them.

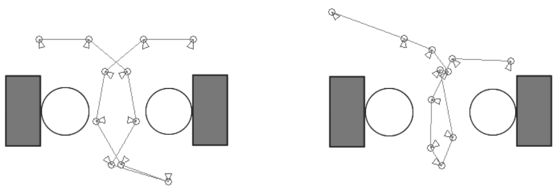


Figure 7: Example of path planning for two objects: (a) The baseline path that takes the images around the object. (b)The proposed path, which prefers viewpoints with both objects visible and thus improves efficiency.

**Algorithms compared.** We compare our algorithm "soft" with the following approaches:

**ori:** (original model) the original classifier $I_{ori}$ with solely the manually labeled training images.

**base:** (baseline path) the robot follows the baseline path to update the classifier, and directly uses the predicted objects

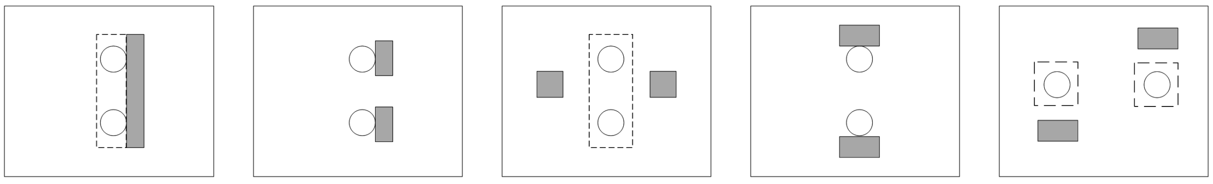


Figure 6: Five settings with obstacles in the environment. Two types of obstacles are considered.

| | cam-1 | cam-2 | car | di-cam | mou | ph-1 | ph-2 | stapler | ave |
|---|---|---|---|---|---|---|---|---|---|
| ori | 46.2 | 68.9 | 59.0 | 64.1 | 26.8 | 87.1 | 80.8 | 56.4 | 61.2 |
| base | 58.9 | 74.9 | 73.1 | 75.5 | 43.3 | 82.0 | 77.9 | 74.1 | 70.0 |
| semi | 51.2 | 70.0 | 63.2 | 39.3 | 12.4 | 88.2 | 74.5 | 63.1 | 57.7 |
| pp | 65.8 | 76.1 | **82.0** | 83.1 | 46.0 | 84.2 | 84.9 | 80.8 | 75.4 |
| soft | **69.4** | **80.4** | 80.6 | **84.5** | **51.8** | **88.5** | **86.7** | **83.1** | **78.1** |

Table 1: Average precision (in %) on 8 objects. The value is averaged over different environment settings for each object.

to re-train.[2]

**semi:** (semi-supervised Alg.) the robot follows the proposed path **pp**, and we implement an existing Semi-supervised algorithm [Leistner *et al.*, 2008] to re-train the classifier with the collected images.

**pp:** (proposed path) the same setting with **base**, but the robot follows the path computed using our approach.

**soft:** (soft-weighting Alg.) this is our full algorithm that has the same setting with **pp**, but uses our proposed algorithm to re-train the model.

**Results.** For path-planning, Fig. 7(a) shows an example of the baseline path, and Fig. 7(b) shows the path computed by our proposed method.

Table 1 shows the detection average precision of the different algorithms for each object. We detail the results on all five different algorithms in six different environments (one with no-obstacle and five different settings with obstacles) in Fig. 8. We see that compared to the baseline method, our method takes better images, more reliably uses them for re-training, and hence increases the performance of the detectors. It can choose efficient locations to observe multiple objects in one image, (Fig. 7 shows an example). Therefore, although we constrain the number of total collected images, the proposed method can acquire more images for each individual object. For **semi**, because the appearance of the object is very different when the view changes, directly measuring the similarity between features cannot give a boost. The proposed soft-margin algorithm performs the best among the five because it builds a more robust classifier that handles the errors in tracking.

We show some detection results of **base** and **soft** in Fig. 10. The proposed algorithm can localize the object in the testing images better, and has fewer false positives than the baseline. Two examples of the Precision-Recall curves are shown in Fig. 9.

Fig. 8 shows the performance of the algorithms in different obstacle scenarios (Fig. 6). We observe that the environment with no obstacles (**no-ob**) has the highest performance because in the free space with no occlusion, different object-views can be better captured and used to learn. Also, in setting **ob-3** the proposed path offers less improvement over the baseline because given the constraint of the obstacles, the baseline path and proposed path are very similar. They all travel around the two objects and form a circle around the obstacle (in dashed lines), and thus two methods have very similar performances in this situation. However in other settings, the proposed paths can clearly out-perform the baselines. After combining with our soft-weighting scheme (**soft**), the proposed algorithm becomes the best one.

[2]The baseline method samples the points around the object on the maximum utility circle.

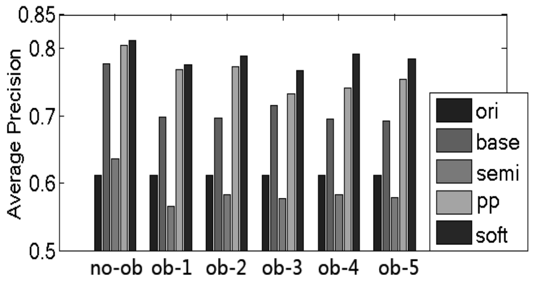


Figure 8: The average precision of five methods in six cases: no-ob: the settings with no obstacle in the environment. ob-1 to ob-5: five settings with obstacles. Y-axis is the mean of average precision for different methods.

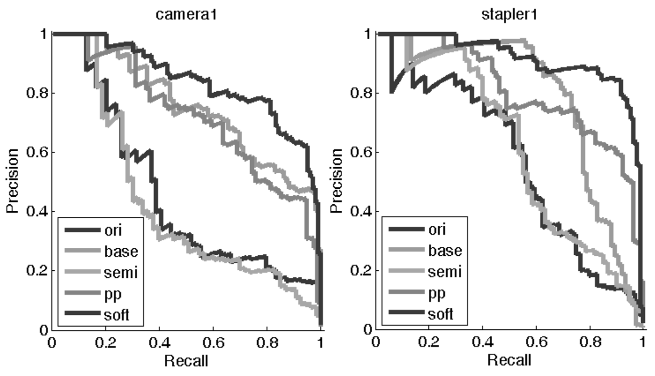


Figure 9: Two examples of the Precision-Recall curves ((a): camera 1; (b): stapler) using different algorithms.

## 8 Conclusion

We proposed an algorithm for enabling a robot to improve its classifiers by efficiently collecting and using more train-

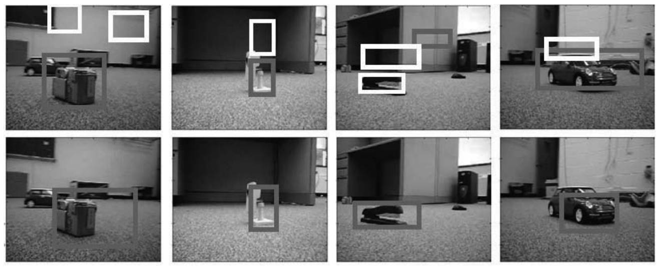


Figure 10: Improvement in object detection with the proposed algorithm. *The first row*: it shows the results using the baseline path to update the detector (**base**). *The second row*: it shows the results of the detector updated with the proposed path and soft-weighting (**soft**). Bounding boxes indicate the object detection and the red rectangles represent the one with the maximum beliefs.

ing data from its environment. While collecting additional training images from new views, a previously visited location affects the utility of collecting data from a future location and therefore the path planning problems becomes hard with views being dependent on each other. We address this issue by proposing a path planning algorithm that uses a sparse graph to model these dependencies. After the robot collected additional images by following this path, we generated labels for these images by combining beliefs from tracking and beliefs from existing detectors by using an HMM. We found that our approach handles the errors from tracking well, and the resulting method for updating the classifier is quite robust. We then performed robotic experiments in six different settings with obstacles, and our robot was able to improve its detectors significantly for eight objects from six categories.

## References

[Dalal and Triggs, 2005] N Dalal and B Triggs. Histograms of oriented gradients for human detection. In *CVPR*, 2005.

[Denzler and Brown, 2002] J Denzler and C M Brown. Information theoretic sensor data selection for active object recognition and state estimation. *PAMI*, 24(2):145–157, 2002.

[Ekvall *et al.*, 2007] S Ekvall, D Kragic, and P Jensfelt. Object detection and mapping for service robot tasks. *Robotica*, 25(2):175–187, 2007.

[Felzenszwalb *et al.*, 2008] P F Felzenszwalb, D McAllester, and D Ramanan. A discriminatively trained, multiscale, deformable part model. In *CVPR*, 2008.

[Forssén *et al.*, 2008] P Forssén, D Meger, K Lai, S Helmer, J J Little, and D G Lowe. Informed visual search: Combining attention and object recognition. In *ICRA*, 2008.

[Geraerts, 2006] R Geraerts. Sampling-based motion planning: Analysis and path quality, 2006.

[Grant and Boyd, 2010] M. Grant and S. Boyd. CVX: Matlab software for disciplined convex programming, version 1.21. http://cvxr.com/cvx, August 2010.

[Jia *et al.*, 2010] Z Jia, Y Chang, and T Chen. A general boosting-based framework for active object recognition. In *BMVC*, 2010.

[Kemp *et al.*, 2008] C C Kemp, C D Anderson, H Nguyen, A J Trevor, and Z Xu. A point-and-click interface for the real world: laser designation of objects for mobile manipulation. In *HRI*, 2008.

[Klingbeil *et al.*, 2008] E Klingbeil, A Saxena, and A Ng. Learning to open new doors. In *RSS workshop on Robot Manipulation*, 2008.

[Klingbeil *et al.*, 2010] E Klingbeil, B Carpenter, O Russakovsky, and A Y Ng. Autonomous operation of novel elevators for robot navigation. In *ICRA*, 2010.

[Krause *et al.*, 2008a] A Krause, J Leskovec, C Guestrin, JV Briesen, and C Faloutsos. Efficient sensor placement optimization for securing large water distribution networks. *JWRPM*, pages 134(6),516–526., 2008.

[Krause *et al.*, 2008b] A Krause, A P Singh, and C Guestrin. Near-optimal sensor placements in gaussian processes: Theory, efficient algorithms and empirical studies. *JMLR*, 9:235–284, 2008.

[Laporte and Arbel, 2006] C Laporte and T Arbel. Efficient discriminant viewpoint selection for active bayesian recognition. *IJCV*, 68(3):267–287, 2006.

[Laporte *et al.*, 2004] C Laporte, R Brooks, and T Arbel. A fast discriminant approach to active object recognition and pose estimation. In *ICPR*, 2004.

[LaValle, 2006] S M LaValle. *Planning Algorithms*. Cambridge University Press, 2006.

[Leibe *et al.*, 2008] B Leibe, A Leonardis, and B Schiele. Robust object detection with interleaved categorization and segmentation. volume 77, pages 259–289, 2008.

[Leistner *et al.*, 2008] C Leistner, H Grabner, and H Bischof. Semi-supervised boosting using visual similarity learning. In *CVPR*, 2008.

[Li *et al.*, 2011] C Li, TP Wong, N Xu, and A Saxena. Feccm for scene understanding: Helping the robot to learn multiple tasks. In *Video contribution in ICRA*, 2011.

[Lowe, 1999] D G Lowe. Object recognition from local scale-invariant features. In *ICCV*, 1999.

[Meeussen *et al.*, 2010] W Meeussen, M Wise, S Glaser, S Chitta, C McGann, P Mihelich, E Marder-Eppstein, M Muja, V Eruhimov, T Foote, and et al. Autonomous door opening and plugging in with a personal robot. In *ICRA*, 2010.

[Meger *et al.*, 2008] D Meger, P Forssén, K Lai, S Helmer, S McCann, T Southey, M A Baumann, J J Little, and D G Lowe. Curious george: An attentive semantic robot. *RAS*, 56(6):503–511, 2008.

[Meger *et al.*, 2010] D Meger, A Gupta, and J J Little. Viewpoint detection models for sequential embodied object category recognition. In *ICRA*, 2010.

[Sapp *et al.*, 2008] B Sapp, A Saxena, and A Ng. A fast data collection and augmentation procedure for object recognition. In *AAAI*, 2008.

[Sofman *et al.*, 2006] B Sofman, E Lin, J A Bagnell, J Cole, N Vandapel, and A Stentz. Improving robot navigation through self-supervised online learning. *JFR*, 23(11-12):1059–1075, 2006.

[Sturm *et al.*, 2010] J Sturm, K Konolige, C Stachniss, and W Burgard. Vision-based detection for learning articulation models of cabinet doors and drawers in household environments. In *ICRA*, 2010.

[Viterbi, 1967] A J Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *TIT*, IT-13:260–269, 1967.