Appears in the
Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94),
Seattle, Washington, July 1994.

Learning to Select Useful Landmarks

Russell Greiner
Siemens Corporate Research
Princeton, NJ 08540
greiner@learning.siemens.com

Ramana Isukapalli
Department of Computer Science
Rutgers University
ramana@cs.rutgers.edu

Abstract

To navigate effectively, an autonomous agent must be able to quickly and accurately determine its current location. Given an initial estimate of its position (perhaps based on dead-reckoning) and an image taken of a known environment, our agent first attempts to locate a set of landmarks (real-world objects at known locations), then uses their angular separation to obtain an improved estimate of its current position. Unfortunately, some landmarks may not be visible, or worse, may be confused with other landmarks, resulting in both time wasted in searching for invisible landmarks, and in further errors in the agent's estimate of its position. To address these problems, we propose a method that uses previous experiences to learn a selection function that, given the set of landmarks that might be visible, returns the subset which can reliably be found correctly, and so provide an accurate registration of the agent's position. We use statistical techniques to prove that the learned selection function is, with high probability, effectively at a local optimal in the space of such functions. This report also presents empirical evidence, using real-world data, that demonstrate the effectiveness of our approach.

1. Introduction

To navigate effectively, an autonomous agent R must be able to quickly and accurately determine its current location. R can obtain fairly accurate estimates of its position using dead-reckoning; unfortunately, the errors in these estimates accumulate over long distances, which can lead to unacceptable performance (read "bumping into walls" or "locating the wrong office"). An obvious way to reduce this problem is to observe the environment, and use the information in these observations to improve our estimate of R's position; *cf.*, the work using Kalman filters (Kosaka & Kak 1992; Cox & Wilfong 1990) and other techniques (Smith & Cheeseman 1987; Kuipers & Levitt 1988; Fennema *et al.* 1990; Engelson 1992) We will model the environment using only a set of "landmarks", each a (potentially visible) real-world object at a known location; these objects could be doors, corners and pictures when specifying the hallways within building, or major buildings, junctions and prominent signs when specifying the streets within a city.[1] Given an initial estimate of its position (perhaps based on dead-reckoning) and an image taken of a known environment, R can first attempt to locate a set of possibly visible landmarks, then use their angular separation to obtain an improved estimate of its current position.

Landmark-based position estimation is a popular technique in robot navigation (Case 1986; Sugihara 1988; 1987; Levitt & Lawton 1990). Many of these landmark-based methods assume that all landmarks can be found reliably. Unfortunately, some landmarks may not be visible; for example, certain corners may always be in shadow and so are difficult to see, or some hanging pictures may have been removed after the floor-plan was released. These can force R to waste time, searching in vain for invisible landmarks. Worse, some landmarks may be easily confused with others; *e.g.*, door A may be mistaken for door B, or some landmark A (say the convex corner of two walls) may be occluded by another object B (say the convex corner of filing cabinet) that looks sufficiently similar that R might think that B is A. As this can cause R to believe that A is located at B's position, these mis-identified objects can produce further errors in R's estimate of its position.[2]

It therefore makes sense to search for only the subset

[1] Notice this information is essentially the same as the information required for the navigation task itself, to specify the destination or some required intermediate points. *N.b.*, we assume that this set of all possible landmarks is known initially; this contrasts with other systems that also attempt to learn the set of landmarks from the observations; *cf.*, (Kuipers & Byun 1988) and others.

[2] Another possible complication is that R may identify a wide landmark correctly, but mistakenly refer to the wrong position within that landmark. Also, R uses a *set* of identified landmarks to locate its position; depending on the geometric positions of these landmarks, small errors in landmark location may lead to quite large errors in R's positional estimate. We of course prefer landmarks sets that provide position estimates that are relatively insensitive to errors in landmark identification.

of the potentially visible landmarks that can be found reliably, which are not confusable with others, etc. Unfortunately, it can be very difficult to determine this good subset *a priori*, as (1) the landmarks that are good for one set of R-positions can be bad for another; (2) the decision to seek a landmark can depend on many difficult-to-incorporate factors, such as lighting conditions and building shape; and (3) the reliability of a landmark can also depend on unpredictable events; *e.g.*, exactly where R happens to be when it observes its environment, how the building has changed after the floor-plan was finalized, and whether objects (perhaps people) are moving around the area where R is looking. These factors make it difficult, if not impossible, to designate the set of good landmarks ahead of time.

This report presents a way around this problem: Section 2 proposes a method that learns a good "selection function" that, given the set of landmarks that may be visible, returns the subset which can usually be found correctly. We also use statistical techniques to prove that this learned selection function is, with high probability, effectively at a local optimum in the space of such functions. Section 3 then presents empirical results that demonstrate that this algorithm can work effectively. We first close this section by presenting a more precise description of the performance task, showing how R estimates its position:

Specification of Performance Task: At each point, R will have an estimate $\hat{\mathtt{x}}$ of its current position x and a measure of the uncertainty (here the covariance matrix). R uses the `LMs( x )` algorithm to specify the subset of the landmarks that *may* be visible from each position `x`; we assume `LMs(` $\hat{\mathtt{x}}$ `)` is essentially the same as `LMs( x )`. (*I.e.*, we assume that R's estimate of its position is sufficient to specify a good approximation of the set of possibly appropriate landmarks.) R also uses an algorithm `Locate(` $\hat{\mathtt{x}}$, $\hat{\sigma}$, `img, lms )` that, given R's estimate of its position $\hat{\mathtt{x}}$ and uncertainty $\hat{\sigma}$, an image `img` taken at R's current position and a set of landmarks `lms`, returns a new estimated position (and uncertainty) for R.

To instantiate these processes: In the current RATBOT system (Hancock & Judd 1993), the `LMs` process uses a comprehensive "landmark-description" of the environment, which is a complete list of all of the objects in that environment that could be visible, together with their respective positions. This could be based on the floor-plan of a building, which specifies the positions of the building's doors, walls, wall-hangings, etc.; or in another context, it could be a map of the roads of a city, which specifies the locations of the significant buildings, signs, and so forth. The `Locate(` $\hat{\mathtt{x}}$, $\hat{\sigma}$, `img, lms )` process first attempts to find each landmark $l_i \in$ `lms` within the image `img`; here it uses $\hat{\mathtt{x}}$ and $\hat{\sigma}$ to specify where in the image to look for this l_i. It will find a subset of these landmarks, each at some angle (relative to a reference landmark). `Locate` then uses simple geometric reasoning to obtain a set of new estimates of R's position; perhaps one from each set of three found landmarks (Hancock & Judd 1994), or see (Gurvits & Betke 1994). After removing the obvious outliers, `Locate` returns the centroid of the remaining estimates as its positional-estimate for R, and the variance of these estimates as the measure of uncertainty; see (Hancock & Judd 1993).

As our goal is an *efficient* way of locating R's position, our implementation uses an inexpensive way of finding the set of landmarks based on simple tests on the visual image; *n.b.*, we are not using a general vision system, which would attempt to actually identify specific objects and specify particular qualities from the visual information.[3]

2. Function for Selecting Good Landmarks

While many navigation systems would attempt to locate *all* of the landmarks that might be visible in an image (*i.e.*, the full set returned by `LMs(` $\hat{\mathtt{x}}$ `)`), we argued above that it may be better to seek only a subset of these landmarks: By avoiding "problematic" landmarks (*e.g.*, ones that tend to be not visible, or confusable), R may be able to obtain an estimate of its location more quickly, and moreover, possibly obtain an estimate that is more accurate.

We therefore want to identify and ignore these bad landmarks. We motivate our approach by first presenting two false leads: One immediate suggestion is to simply exclude the bad landmarks from the catalogue of all landmarks that `LMs` uses, meaning `LMs(·)` will never return certain landmarks. One obvious complication is the complexity of determining which landmarks are bad, as this can depend on many factors, including the color of the landmark, the overall arrangement of the entire environment (which would specify which landmarks could be occluded), the lighting conditions, etc. A more serious problem is the fact that a landmark that is hard to see from one R position may be easy to see, and perhaps invaluable, from another; here, R should be able to use that landmark when registering its location from some positions, but not from others.

We therefore decided to use, instead, a selection function Sel that filters out the bad landmarks from the set of possibly visible landmarks, `lms = LMs(` $\hat{\mathtt{x}}$ `)`: Here, each selection function $\mathtt{Sel}_i$ returns a subset $\mathtt{Sel}_i$`( lms,` $\hat{\mathtt{x}}$, $\hat{\sigma}$ `) =` $\mathtt{lms}_i \subseteq$ `lms`; R then uses this subset to compute its location, returning `Locate(`$\hat{\mathtt{x}}$, $\hat{\sigma}$, img, $\mathtt{lms}_i$ `)`. We want to use a selection function Sel_i such that `Locate(` $\hat{\mathtt{x}}$, $\hat{\sigma}$, `img,` $\mathtt{lms}_i$ `)` is reliably close to R's true position, `x`. To make this

[3] Figure 3 shows, and describes, the actual "images" we use. Also, this articles does not provide pseudo-code for either `LMs` or `Locate`, as our learning algorithm regards these process as black-boxes.

more precise, let

$$\mathrm{Err}(\,\mathrm{Sel}_i,\, \langle \mathtt{x}, \hat{\mathtt{x}}, \hat{\sigma}, \mathrm{img}\rangle\,) \quad = \quad \|\, \mathtt{x} \;-\; \mathtt{Locate}(\,\hat{\mathtt{x}},\, \hat{\sigma},\, \mathrm{img},\; \mathrm{Sel}_i(\mathrm{LMs}(\hat{\mathtt{x}}),\; \hat{\mathtt{x}}, \hat{\sigma})\;)\, \|$$

be the error[4] for the selection function Sel_i and any "situation" $\langle \mathtt{x}, \hat{\mathtt{x}}, \hat{\sigma}, \mathrm{img}\rangle$, and let

$$\mathrm{AveErr}(\,\mathrm{Sel}_i\,) \quad = \quad E_{\langle \mathrm{x}, \hat{\mathrm{x}}, \hat{\sigma}, \mathrm{img}\rangle}[\,\mathrm{Err}(\,\mathrm{Sel}_i,\, \langle \mathtt{x}, \hat{\mathtt{x}}, \hat{\sigma}, \mathrm{img}\rangle\,)\,]$$

be the expected error, over the distribution of situations $\langle \mathtt{x}, \hat{\mathtt{x}}, \hat{\sigma}, \mathrm{img}\rangle$, where $E[\cdot]$ is the expectation operator. Our goal is a selection function Sel_{opt} that minimizes this expected value, over the set of possible selection functions.

The second false lead involves "engineering" this optimal selection function initially. One problem, as observed above, is the difficulty of determining "analytically" which landmarks are going to be problematic for any single situation. Worse, recall that our goal is to find the selection function that works best *over the distribution of situations*; which depends on the distribution of R's actual positions when the function is called, the actual intensity of light sources, what other objects have been moved where, etc. Unfortunately, this distribution of situations is not known *a priori*.

We are therefore following a third (successful) approach: of *learning* a good selection function. Here, we first specify a large (and we hope, comprehensive) class of possible selection functions $\mathcal{S} = \{\mathrm{Sel}_i\}$. Then, given "labeled samples" — each consisting of R's position and uncertainty estimates, the relevant landmark-set and image, and as the label, R's actual position — identify the selection function Sel_i which minimizes $\mathrm{AveErr}(\,\mathrm{Sel}_i\,)$.

Space of Selection Function: We define each selection function $\mathrm{Sel}_k \in \mathcal{S}$ as a conjunction of its particular set of "heuristics" or "filters"; $\mathrm{Filters}(\mathrm{Select}_k) = \{f_1, \ldots, f_m\}$, where each filter f_i is a predicate that accepts some landmarks and rejects others. Hence, the $\mathtt{Select}_k\mathtt{(\ lms,\ }\hat{\mathtt{x}}\mathtt{,\ }\hat{\sigma}\mathtt{\ )}$ procedure will examine each $\ell \in \mathtt{lms}$ individually, and reject it if *any* f_i filter rejects it; see Figure 1.

While we can define a large set of such filters, this report focuses on only two parameterized filters:

$$\mathrm{BadType}_{K_3}(\ell, \hat{\mathtt{x}}, \hat{\sigma}) : \quad \text{Reject } \ell \textit{ if } \mathrm{Type}(\,\ell\,) \notin K_3$$

$$\mathrm{TooSmall}_{k_1,k_2}(\ell, \hat{\mathtt{x}}, \hat{\sigma}) : \text{Reject } \ell \textit{ if } \|\mathrm{Posn}(\ell) - \hat{\mathtt{x}}\| > k_1 \text{ and } \mathrm{AngleWidth}(\ell,\ \hat{\mathtt{x}}) < k_2$$

[4] As we are also considering the efficiency of the overall process, we will actually use the slightly more complicated error function presented in Section 3 below. This is also why we did not address the landmark-selection task using robust analysis: Under that approach, our system would first spend time and resources seeking each landmark, and would then decide whether to use each possible correspondence. As our approach, instead, specifies which landmarks should be sought, we will gather less data, and so expend fewer resources.

```
Selj( lms: landmarks, x̂: pos'n, σ̂: var.): landmarks
  OK_LMs ← {}
  ForEach ℓ ∈ lms
    KeepLM ← T
    ForEach fi ∈ Filters( Selj )
      If [ fi(ℓ, x̂, σ̂) ≡ Ignore ]  Then KeepLM ← F
    End (inner) ForEach
    If [ KeepLM ≡ T ]   Then OK_LMs  ← OK_LMs + ℓ
  End (outer) ForEach
  Return( OK_LMs )
End Select
```

Figure 1: PseudoCode for Sel_j Selection Function

where $\mathrm{Type}(\ell)$ refers to the type of the landmark ℓ, which can be "Door", "BlackStrip", etc.[5] The parameter K_3 specifies the subset of landmark-types that should be used. Using "$\mathrm{Posn}(\ell)$" to refer to ℓ's real-world coördinates and "$\mathrm{AngleWidth}(\ell,\ \hat{\mathtt{x}})$" to refer to the angle subtended by the landmark ℓ, when viewed from $\hat{\mathtt{x}}$, $\mathrm{TooSmall}_{k_1,k_2}(\ell, \hat{\mathtt{x}}, \hat{\sigma})$ rejects the landmark ℓ if ℓ is both too far away (greater than k_1 meters) and also too small (subtends an angle less than k_2 degrees), from R's estimated position $\hat{\mathtt{x}}$.

Using these filters, $\mathcal{S} = \{\mathrm{Sel}_{k_1,k_2,K_3}\}$ is the set of all selection functions, over a combinatorial class of settings of these three parameters. As stated above, we want to find the best settings of these variables, which minimize the expected error $\mathrm{AveErr}[\,\mathrm{Sel}_{k_1,k_2,K_3}\,]$.

Hill-Climbing in Uncertain Space: There are two obvious complications with our task of finding this optimal setting: First, as noted above, the error function depends on the distribution of situations, which is not known initially. Secondly, even if we knew that information, it is still difficult to compute the optimal parameter setting, as the space of options is large and ill-structured (*e.g.*, K_3 is discrete, and there are subtle non-linear effects as we alter k_1 and k_2).

We use a standard hill-climbing approach to address the second problem, based on a set of operators $\mathcal{T} = \{\tau_k\}$ that each map one selection function to another; *i.e.*, for each $s \in \mathcal{S}$, $\tau_k(s) \in \mathcal{S}$ is another selection function. We use the obvious set of operators: τ_1^+ increments the value of k_1 and τ_1^- decrements k_1's value; hence $\tau_1^+(\,\mathrm{Sel}_{5,\,8,\,\{t1,t3,t7\}}\,) = \mathrm{Sel}_{6,\,8,\,\{t1,t3,t7\}}$ and $\tau_1^-(\,\mathrm{Sel}_{5,\,8,\,\{t1,t3,t7\}}\,) = \mathrm{Sel}_{4,\,8,\,\{t1,t3,t7\}}$. Similarly, τ_2^+ and τ_2^- respectively increment and decrement the k_2 value. There are 9 different τ_3^i operator, each of which "flips" the i^{th} bit of K_3; hence $\tau_3^{t1}(\,\mathrm{Sel}_{5,\,8,\,\{t1,t3,t7\}}\,) = \mathrm{Sel}_{5,\,8,\,\{t3,t7\}}$ and $\tau_3^{t8}(\,\mathrm{Sel}_{5,\,8,\,\{t1,t3,t7\}}\,) = \mathrm{Sel}_{5,\,8,\,\{t1,t3,t7,t8\}}$.

To address the first problem — *viz.*, that the distribution is unknown — we use a set of observed examples

[5] The current system contains nine different types: Miscellaneous, Black_Strip, Concave_Corner, Convex_Corner, Dark_Door, Light_Door, Picture, FireExtinguisher and Support_between_Windows.

$\textsc{LearnSF}(\,\mathrm{Sel}_1 : \texttt{select\_fn},\ \epsilon : \Re^+,\ \delta : \Re^+\,) : \texttt{select\_fn}$

For $\quad j \leftarrow 1..\infty \quad$ do

$\quad \mathcal{T}[\,\mathrm{Sel}_j\,] \ \leftarrow\ \{\,\tau_k(\mathrm{Sel}_j)\,\}_k$

$\quad$ Take $\quad n \leftarrow m(\,\frac{\epsilon}{2},\ \frac{\delta\, 6\, |\mathcal{T}[\mathrm{Select}_j]|}{j^2\,\pi^2}\,)\quad$ samples,

$\quad\quad \mathcal{U} \ \leftarrow\ \{\,u_1,\,\ldots,\,u_n\,\}\qquad$ *[Each $u_i = \langle \mathbf{x}_i, \hat{\mathbf{x}}_i, \hat{\sigma}_i, img_i\rangle$]*

$\quad$ **If** $\quad \exists\, \mathrm{Sel}' \in \mathcal{T}[\,\mathrm{Sel}_j\,]\quad$ such that

$\qquad\qquad \hat{E}^{(\mathcal{U})}[\,\mathrm{Sel}_j\,] - \hat{E}^{(\mathcal{U})}[\,\mathrm{Sel}'\,]\ \geq\ \frac{\epsilon}{2}$

$\quad$ **then** Let $\quad \mathrm{Sel}_{j+1} \leftarrow \mathrm{Sel}'$

$\quad$ **else** $\left[\,\text{Here, } \forall\,\mathrm{Sel}',\ \hat{E}^{(\mathcal{U})}[\,\mathrm{Sel}_j\,] - \hat{E}^{(\mathcal{U})}[\,\mathrm{Sel}'\,]\ <\ \frac{\epsilon}{2}\,\right]$

$\qquad$ Return Sel_j

End For

End LearnSF

Figure 2: (Simplified) PseudoCode for LearnSF

to estimate the relevant information: Let

$$\begin{aligned}\hat{E}^{(\mathcal{U})}_{k_1,k_2,K_3} &= \hat{E}^{(\mathcal{U})}[\,\mathrm{Err}(\,\mathrm{Sel}_{k_1,k_2,K_3}\,)\,] \\ &= \tfrac{1}{|\mathcal{U}|}\textstyle\sum_{u_i\in\mathcal{U}} \mathrm{Err}(\,\mathrm{Sel}_{k_1,k_2,K_3},\ u_i\,)\end{aligned}$$

be the empirical average error of the selection function $\mathrm{Sel}_{k_1,k_2,K_3}$ over the set of training samples $\mathcal{U} = \{\,\langle \mathbf{x}_i, \hat{\mathbf{x}}_i, \hat{\sigma}_i, \mathrm{img}_i\rangle\,\}_i$, which we assume to be independent and identically distributed. We then use some statistical measure to relate the number of samples seen, to our confidence that $\hat{E}^{(\mathcal{U})}_i$ will be close to the real mean $\mu_i = E_{u_j}[\,\mathrm{Err}(\,\mathrm{Sel}_i,\ u_j\,)\,] = \mathrm{AveErr}(\,\mathrm{Sel}_i\,)$ value. In particular, we need a function $m(\cdots)$ such that, after $m(\epsilon,\delta)$ samples, we can be at least $1-\delta$ confident that the empirical average $\hat{E}^{(\mathcal{U})}$ will be within ϵ of the population mean μ; *i.e.*, $|\mathcal{U}|\ \geq\ m(\epsilon,\delta)\ \Rightarrow\ Pr[\,|\hat{E}^{(\mathcal{U})} - \mu|\ >\ \epsilon\,]\ \leq\ \delta$. If we can assume that the underlying distribution of error values is close to a normal distribution, then we can use

$$m_{Norm}(\epsilon,\delta) = \left(\frac{\lambda}{\epsilon} z^{-1}(1-\frac{\delta}{2})\right)^2$$

where the $z(p) = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{p} e^{-\frac{x^2}{2}}\,dx$ function computes the p^{th} quantile of the standard normal distribution $\mathcal{N}(0,1)$ (Bickel & Doksum 1977).

The LearnSF algorithm, sketched in Figure 2,[6] combines the ideas of hill-climbing with statistical sampling: Given an initial selection function $\mathrm{Sel}_1 = \mathrm{Sel}_{k_1,k_2,K_3} \in \mathcal{S}$, and the parameters ϵ and δ, LearnSF will use a sequence of example situations $\{u_i\}$ to climb from the initial Sel_1 through successive neighboring selection functions $(\mathrm{Sel}_1, \mathrm{Sel}_2, \mathrm{Sel}_3, \ldots)$ until reaching, and returning, a final Sel_m. With high probability, this Sel_m is essentially a local optimum. Moreover, LearnSF requires relatively few samples for each climb. To state this more precisely:

[6] We actually use much more efficient, but more complex, algorithm that, for example, decides whether to climb to a new $\mathrm{Sel}_{j+1} \leftarrow \mathrm{Sel}'$ after observing each image, (rather than a batch of n images); see (Greiner & Isukapalli 1994; Greiner 1994).

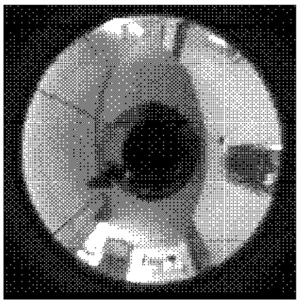


Figure 3: RatBOT's view (looking up at tree ornament), and "strip", corresponding to annulus in image

Theorem 1 (from (Greiner & Isukapalli 1994)) *The LearnSF(Sel_1, ϵ, δ) process incrementally produces a series of selection functions $Sel_1, Sel_2, \ldots, Sel_m$, such that each $Sel_{j+1} = \tau_j(Sel_j)$ for some $\tau_j \in \mathcal{T}$ and, with probability at least $1-\delta$,*

1. *the expected error of each selection function is strictly better than its predecessors i.e.,* $\forall\, 2 \leq j \leq m:\ AveErr(Sel_{j-1})\ <\ AveErr(Sel_j);$ *and*
2. *the final selection function (returned by LearnSF), Sel_m, is an "ϵ-local optimum" — i.e.,* $\neg\exists\,\tau\in\mathcal{T}:\ AveErr(\,\tau(Sel_m)\,)\ <\ AveErr(\,Sel_m\,) - \epsilon$.

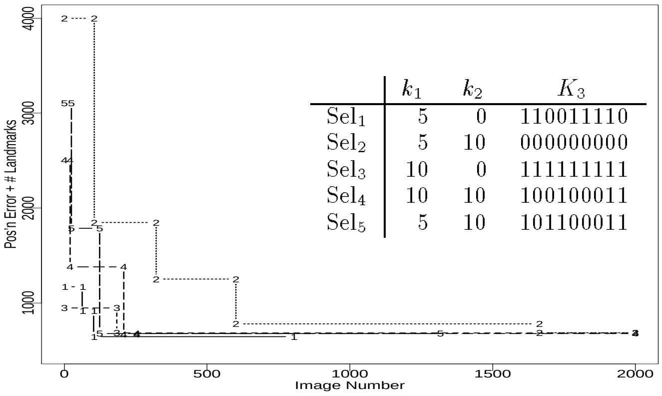
given the statistical assumption that the underlying distribution is essentially normal. Moreover, LearnSF will terminate with probability 1, and will stay at any Sel_j (before either terminating or climbing to a new Sel_{j+1}) for a number of samples that is polynomial in $\frac{1}{\epsilon}$, $\frac{1}{\delta}$, $|\mathcal{T}|$ and $\lambda = \max_{\tau\in\mathcal{T},\ Sel\in\mathcal{S},u} |Err(Sel,u)\ -\ Err(\tau(Sel),u)|$, which is the largest difference in error between a pair of neighboring selection functions for any sample. □

3. Empirical Results

To test the theoretical claims that a good selection function can help an autonomous agent to register its position efficiently and accurately, and also that LearnSF can help find such good selection functions, we implemented various selection functions and the LearnSF learning algorithm, and incorporated them within the implemented autonomous agent, RatBOT, described in (Hancock & Judd 1993). This section describes our empirical results.

We first took a set of 270 "pictures" at known locations within three halls of our building. Each of these pictures is simply an array of 360 intensity values, each corresponding to the intensity at a particular angle, in a plane parallel to the floor; these are shown on right-hand side of Figure 3.[7] We have also identified 157 different landmarks in these regions, each represented as

[7] These were obtained using a "Nomad 200" robot with a CCD camera mounted on top, pointing up at a spherical mirror (which is actually a christmas tree ornament); see left-hand side of Figure 3. We then extract from this image a 1-pixel annulus, which corresponds to the light intensity at a certain height; see right-hand side of Figure 3.



	k_1	k_2	K_3
Sel_1	5	0	110011110
Sel_2	5	10	000000000
Sel_3	10	0	111111111
Sel_4	10	10	100100011
Sel_5	5	10	101100011

Figure 4: LEARNSF's Hill-Climbs, for different initial Selection Functions

simply an object of a specified type (one of the nine categories), located between a pair of coördinates $\langle x_1, y_1\rangle$ and $\langle x_2, y_2\rangle$; where, once again, this $\langle x, y\rangle$ plane is parallel to the floor and goes through the center of the bulb.

Each experiment used a particular initial selection function, error function, values for ϵ and δ, way of estimating R's position, and statistical assumption. We first describe one experiment in detail, then discuss a battery of other experiments that systematically vary the experimental parameters.

Experiment#1 Specification: LEARNSF began with the Sel_1 selection function shown in Figure 4. This function rejects a landmark if either it is more than 5 meters away from our estimated position and also subtends an angle less than 0 degrees,[8] or if the landmark's type is one of Concave_Corner, Convex_Corner, or Support_between_Windows (these are the second, third and ninth types, corresponding to the bits that are 0 in the Sel_1 row of Figure 4). We used $\delta = 0.05$, meaning that we would be willing to accept roughly 1 mistake in 20 runs. The $\epsilon = 0.1$ setting means that we do not care if the average error of two selection functions differs by less than 0.1m; as we allowed errors as large as 4m, this corresponds to an allowable tolerance of only 2.5%.

As our goal is to minimize both positional error and computational time, we use an error function that is the weighted sum of the positional error (which is the difference between the obtained estimated position and the real position) and "#landmarks–to–pos'n-error ratio" times the number of landmarks that were selected. Here, we set the ratio to 0.01, to mean, in effect, that each additional landmark "costs" 0.01m.

Finally, while *we* know that image img_i is taken at location $\mathbf{x}_i$, it unrealistic to assume that RATBOT will know that information; in general, we assume that RATBOT will instead see an approximate $\hat{\mathbf{x}}_i$. We

[8] As nothing can subtend an angle strictly less than 0^o, this first clause is a no-op — *i.e.*, it will not reject *any* landmark.

Sample #	Selection Function	E[LM-Err]
0	$\text{Sel}_1 = \langle\langle 5,\ 0\rangle;\ [110011110]\rangle$	1.171
62	$\text{Sel}_{1:1} = \langle\langle 5,\ 0\rangle;\ [110111110]\rangle$	0.916
102	$\text{Sel}_{1:2} = \langle\langle 5,\ 2\rangle;\ [110111110]\rangle$	0.647

Table 1: Data for LEARNSF's Climbs from Sel_1

model this by setting $\hat{\mathbf{x}}_i = x_i + \nu_i^{(\sigma)}$, where each $\nu_i^{(\sigma)}$ is a normally-distributed random value with mean zero and variance σ. Here, we used $\sigma = 0.3$m. Recall also that the `Locate` function needs a value for $\hat{\sigma}$ to constrain its landmark-location process; we also set $\hat{\sigma}$ to be σ.

Experiment#1 Results: Given these settings, LEARNSF observed 62 labeled samples before climbing to the new selection function $\text{Sel}_{1:1} = \langle\langle 5,\ 0\rangle;\ [110111110]\rangle$, which differs from Sel_1 only by *not* rejecting all `Convex_Corners`.[9] It continued using this selection function for 40 additional samples, before climbing to the $\text{Sel}_{1:2} = \langle\langle 5,\ 2\rangle;\ [110111110]\rangle$ selection function, which rejects landmarks that are both more than 5m from R's estimated position and also less than 2^o. It continued using this $\text{Sel}_{1:2}$ function for another 700 samples before LEARNSF terminated, declaring this selection function to be a "0.1-local optimum" — *i.e.*, none of $\text{Sel}_{1:2}$'s neighbors has a utility score that is more than $\epsilon = 0.1$ better than $\text{Sel}_{1:2}$. (We found, in fact, that $\text{Sel}_{1:2}$ is actually a *bona fide* local optimum, in that none of its neighbors is even as good as it is.)

The solid line (labeled "1") in Figure 4 shows LEARNSF's performance here. Each horizontal line-segment corresponds to a particular selection function, where the line's y-value indicates the "average test error" of its selection function, which was computed by running this selection function through all 270 images.[10] These horizontal lines are connected by vertical lines whose x-value specify the sample number when LEARNSF climbed. Table 1 presents a more detailed break-down of this data.

Other Variants: Our choice of $\text{Sel}_1 = \langle\langle 5,\ 0\rangle;\ [110111110]\rangle$ was fairly arbitrary; we also considered the four other reasonable starting selection functions shown in Figure 4. Notice that $\text{Sel}_2 = \langle\langle 5,\ 10\rangle;\ [000000000]\rangle$ rejects *every* landmark; and $\text{Sel}_3 = \langle\langle 5,\ 0\rangle;\ [111111111]\rangle$ accepts *every* landmark. Figure 4 also graphs the performance of these functions. Notice that LEARNSF finds improvements in all five cases.

We also systematically varied the other parameters: trying values of ϵ = 1.0, 0.2, 0.1, 0.05, 0.02, 0.01, 0.005; δ = 0.005, 0.01, 0.05, 0.1; σ =

[9] $\text{Sel}_{i:j}$ refers to the selection function reached after j climbs, when starting from Sel_i. Hence, $\text{Sel}_{i:0} \equiv \text{Sel}_i$.

[10] To avoid testing on the training data, we computed this value using a *new* set of randomly-generated positional estimates, $\{\hat{\mathbf{x}}'_i = x_i + \nu_i^{(\sigma)\prime}\}$, where again $\nu_i^{(\sigma)\prime}$ is a random variable drawn from a 0-mean σ-variance distribution.

0, 0.3, 0.5, 1.0; and the "#landmark–to–pos'n-error ratio" of 0, 0.02, 0.05, 0.1, 0.2. (The 0 setting tells LearnSF to consider only the accuracy of a landmark set, and not the cost of finding those landmarks.) We also used LearnSF_{HI}, a variant of LearnSF that replaces the $m_{Norm}(\cdot)$ function with the weaker

$$m_{HI}(\epsilon, \delta) \quad = \quad \frac{1}{2}\left(\frac{\lambda}{\epsilon}\right)^2 \ln \frac{2}{\delta}$$

function, which is based on Hoeffding's inequality (Hoeffding 1963; Chernoff 1952), and so does *not* require the assumption that the error values are normally distributed. All of these results are reported, in detail, in (Greiner & Isukapalli 1994).

Summary of Empirical Results: The first obvious conclusion is that selection functions are useful; notice in particular that the landmarks they returned enabled R to obtain fairly good positional estimates — within a few tenths of a meter. Notice also that the obvious degenerate selection function, Sel_3 which accepted all landmarks, was *not* optimal; *i.e.*, there were functions that worked more effectively. Secondly, this LearnSF function works effectively, as it was able to climb to successively better selection functions, in a wide variety of situations. Not surprisingly, we found that the most critical parameter was the initial selection function; the values of ϵ, δ, σ and even the "#landmark–to–pos'n-error ratio" had relatively little effect. We also found that this LearnSF_{Norm} system seemed to work more effectively than the version that did not require the normality assumption, LearnSF_{HI}: in almost all instances, both systems climbed through essentially the same selection functions, but LearnSF_{Norm} required many fewer samples — by a factor of between 10 and 100! (In the numerous different runs that used $\delta = 0.05$, LearnSF_{Norm} climbed a total of 84 times and terminated 24 times, and so had $84 + 24 = 108$ opportunities to make a mistake; it made a total of only 3 mistakes, all very minor.) Finally, LearnSF's behavior was also (surprisingly) insensitive to the accuracy of R's estimated position, over a wide range of errors; *e.g.*, even for non-trivial values of $|x - \hat{\mathbf{x}}|$.

4. Conclusion

While there are many techniques that use observed landmarks to identify an agent's position, they all depend on being able to effectively find an appropriate set of landmarks, and will produce degraded or unacceptable information if the landmarks are not found, or mis-identified. We can avoid this problem by using only the subset of "good" landmarks. As it can be very difficult to determine this subset *a priori*, we present an algorithm, LearnSF, that uses a set of training samples to *learn* a function that selects the appropriate subset of the landmarks, which can then be used robustly to determine our agent's position. We then prove that this algorithm works effectively — both theoretically and empirically, based on real data obtained using an implemented robot.

Acknowledgments

We gratefully acknowledge the help we received from Thomas Hancock, Stephen Judd, Long-Ji Lin, Leonid Gurvits and the other members of the RatBOT team.

References

Bickel, P. J., and Doksum, K. A. 1977. *Mathematical Statistics: Basic Ideas and Selected Topics.* Oakland: Holden-Day, Inc.

Case, M. 1986. Single landmark navigation by mobile robots. In *SPIE*, volume 727, 231–38.

Chernoff, H. 1952. A measure of asymptotic efficiency for tests of a hypothesis based on the sums of observations. *Annals of Mathematical Statistics* 23:493–507.

Cox, I., and Wilfong, G., eds. 1990. *Autonomous Robot Vehicles.* Springer-Verlag.

Engelson, S. P. 1992. Active place recognition using image signatures. In *SPIE Symposium on Intelligent Robotic Systems, Sensor Fusion V*, 393–404.

Fennema, C.; Hanson, A.; Riseman, E.; Beveridge, J.; and Kumar, R. 1990. Model-directed mobile robot navigation. *IEEE Transactions on Systems, Man and Cybernetics* 20(6):1352–69.

Greiner, R., and Isukapalli, R. 1994. Learning to select useful landmarks. Technical Report SCR-LS94-473.

Greiner, R. 1994. Probabilistic hill-climbing: Theory and applications. Technical report, SCR.

Gurvits, L., and Betke, M. 1994. Robot navigation using landmarks. Technical Report SCR-94-TR-474, SCR/MIT.

Hancock, T., and Judd, S. 1993. Ratbot: Robot navigation using simple visual algorithms. In *1993 IEEE Regional Conference on Control Systems.*

Hancock, T., and Judd, S. 1994. Hallway navigation using simple visual correspondence algorithms. Technical Report SCR-94-TR-479, SCR.

Hoeffding, W. 1963. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association* 58(301):13–30.

Kosaka, A., and Kak, A. C. 1992. Fast vision-guided mobile robot navigation using model-based reasoning and prediction of uncertainties. *Computer Vision, Graphics, and Image Processing* 56(3):271–329.

Kuipers, B. J., and Byun, Y.-T. 1988. A robust, qualitative method for robot spatial learning. In *AAAI-88*, 774–79.

Kuipers, B. J., and Levitt, T. S. 1988. Navigation and mapping in large-scale space. *AI Magazine* 9(2):25–43.

Levitt, T. S., and Lawton, D. T. 1990. Qualitative navigation for mobile robots. *Artificial Intelligence* 44:305–60.

Smith, R., and Cheeseman, P. 1987. On the representation and estimation of spatial uncertainty. *International Journal of Robotics Research* 5(4):56–68.

Sugihara, K. 1987. Location of a robot using sparse visual information. *Robotics Research: The Fourth International Symposium*, 319–26. MIT Press.

Sugihara, K. 1988. Some location problems for robot navigation using a single camera. *Computer Vision, Graphics and Image Processing* 42(1):112–29.