Ideas for SfM-Inspired Path Planning

David Fridovich-Keil
dfk@eecs.berkeley.edu

February 1, 2016

1 Project description

The aim of this project is to design a robust path planner for a mobile outdoor robot. Since the primary motivation for exploration is the collection of sufficient data to create a highly detailed 3D environment map in post-processing, the design of the robot planner is predicated on that goal.

In particular, the robot in this project is a quadrotor platform with several pairs of low-resolution stereo grayscale images, and a single high-resolution color camera on a three-axis stabilized gimbal. Note that there are *no laser scanners*, and *no Kinect-style depth sensors*. In practice, this means that all depth estimation is done using stereo disparity maps and, more generally, **Structure from Motion**, or SfM.

In light of this, we will design the path planning algorithm around our knowledge of how SfM works. That is, SfM relies on matching corresponding **keypoints** between images, and from these correspondences triangulating 3D **landmarks** in the environment. A keypoint is simply a location in an image that is in some way special – for example, the most common type of feature is a corner of some sort, since corners are very distinctive. There are a whole host of feature detectors out there; at the end of the day, most of the popular ones are more or less equivalent in terms of performance, so we will not go into any further detail here. Of course, as soon as two keypoints are determined to match, it is possible to triangulate an approximate 3D position of the corresponding landmark in the environment. Doing this for a large number of landmarks allows the joint optimization of camera pose and landmark position; this is called **bundle adjustment**.

2 Big idea

Since our only sensors are cameras, the final model generation step will be one gigantic bundle adjustment problem. Of course, it is not hard to see that having more landmarks over a larger volume will improve reconstruction quality. The big contribution of this project to the great body of literature on path planning is a novel formulation of this intuitive idea within the existing framework of information-theoretic approaches to exploration.

3 Background

This section provides a (very) brief overview of the state of the art in path planning. For a more in-depth description of these topics, consult the appropriate references. For the most part, this is a summary of the corresponding section(s) in [1].

3.1 What is Exploration?

Exploration is defined as the process by which a robot constructs a map of its environment by moving and acquiring new sensor measurements. It is comprised of two distinct subtasks:

1. Identifying and scoring potential actions the robot can take in order to learn more about the environment

2. Autonomously choosing one of these candidate actions, updating the robot's position on the map, and receiving new sensor measurements and registering them to the map

There are two broad classes of algorithms to solve the exploration problem – geometric and information-theoretic methods – which are described in the following subsections.

3.2 Geometric Exploration Methods

Begin by considering a partially explored environment consisting of regions which are either occupied, unoccupied, or unknown. Formally, each point x in the environment is assigned a label, $L(x) \in \{\text{occupied, unoccupied, unknown}\}$. Perhaps the most obvious case for illustrative purposes is a two dimensional environment made up of discrete squares. That is, each point in the environment is a tuple of integers – i.e. $x = (i, j) : i, j \in \mathbb{Z}$. In this case, it is a simple matter to find the boundary between the explored and unexplored regions (e.g. a simple floodfill will do the job).

One approach that has been incredibly influential is to find these so-called *frontiers* and then, using some heuristic, choose one and plan a path towards it. This is based on the seminal work by Yamauchi, [2].

Of course, there are quite a lot of reasonable heuristics one might consider using. The simplest is probably just to move toward the nearest frontier, but that is not always the best choice. At the end of the day, this method is highly sensitive to the choice of hueristic, which makes it somewhat less general than we might like.

Moreover, it completely breaks down in three dimensions, where it is not such an easy task to even *find* the frontiers, let alone find a *good* frontier to go to. There are two reasons for this. First, in three dimensions algorithms like floodfill take exponentially longer to compute – this can be overcome by clever optimizations like caching results. Second, and worse, 3D environments tend to be more complex than 2D, with more varied shapes and occlusions, leading to far more frontiers, not all of which are worth visiting, yet any one of which might be chosen by an imperfect heuristic.

For these reasons, the community of late has started to gravitate toward a dramatically different formulation based on information theory.

3.3 Information-Theoretic Exploration Methods

Information-theoretic methods may seem dramatically different from purely geometric methods, but in essence they are simply a generalization with some added mathematical formalism.

3.3.1 What is Information Theory?

First off, information theory itself bears some explanation. Of course, this is an entire field, so the summary provided here is merely meant as a very rough guide. For a more in-depth background consult any standard textbook.

Information theory is the mathematical study of information – that is, information theory provides a formal way to understand and analyze what we mean by *information*. It is very closely linked to probability theory, in the sense that information is usually a property of a random variable, rather than a quantity of raw data.

Suppose that a discrete random variable X has a particular distribution, $\mathbb{P}\{X = x\} = p(x)$. The *information content* of X, also called **entropy**, can be calculated from the distribution $p(x)$. The most common form of entropy is called the Shannon entropy, and it is calculated as follows:

$$H_S = -\sum_x p(x) \log_2 p(x) \tag{1}$$

There are many other ways to calculate entropy however; we leave a more in-depth discussion for later sections.

3.3.2 Occupancy Grid

Now let us consider the same simple 2D voxel-grid we described above, in which each point $x = (i, j) : i, j \in \mathbb{Z}$ is assigned a label $L(x) \in \{\text{occupied, unoccupied, unknown}\}$. Of course, this last label, "unknown" is not a *true* label, in the sense that once the robot has explored the entire map, no voxel will be labelled "unknown."

This leads us to an elegant formalism: consider the map to be a collection of random variables X_{ij}, each of which independently takes a label $X_{ij} \in \{\text{occupied, unoccupied}\}$ with some probability. Of course, a priori those probabilities are equal, since we know nothing about the environment.

Elementary probability theory lets us write the joint distribution of a collection of voxels $m = \{X_{ij}\}_{(i,j)\in\mathcal{M}}$ for some set $\mathcal{M}$ as follows:

$$p(m) = \prod_{(i,j)\in\mathcal{M}} p(X_{ij}) \tag{2}$$

What we have just described is called an **occupancy grid**, and it is an especially popular way of representing a map. Of course, there are downsides: most obviously, that the world is not actually a voxel grid, and by reducing it to one introduces sampling artifacts.

3.3.3 Application to Exploration

Now that we have expressed the environment as a collection of binary (Bernoulli) random variables, and using our knowledge of entropy, we can devise a very elegant solution to the exploration problem. That is, for some entropy metric $H(m)$ and paths $r \in \mathcal{R}$, we can write down the optimization problem:

$$r^* = \arg\min_{r \in \mathcal{R}} H(m|r) \tag{3}$$

Intuitively we wish to minimize entropy (which is like the randomness left in the map) given that we take path r. Of course, this is in principle a very difficult problem – after all, how can we know for sure what the entropy will be after taking a particular path?

But the game is not over yet. Consider the following problem, which is only slightly different:

$$r^* = \arg\min_{r \in \mathcal{R}} \mathbb{E}[H(m|r)] \tag{4}$$

Here, we have taken the expectation of the entropy metric. This is usually something we can actually calculate, assuming that we have a good idea of how our robot senses the environment. Such a **sensor model** enables us to anticipate which voxels we will observe over a particular path. Note that the actual observations themselves – i.e. whether or not a particular voxel is observed to be occupied or not – is irrelevant to the calculation of entropy. Entropy only measures randomness, which is obviously decreased when we observe more voxels.

Before proceeding, let us clarify the notation used above, and make it slightly more precise. $H(m|r)$ is intended to mean "the entropy of map m after taking path r." This is silghtly confusing, of course, because the vertical bar is usually used for condiitioning a probability distribution. Fortunately, that is actually what we are doing here. Recall that m is actually a collection of independent random variables, and that a path and a sensor model define a set of possible observations. That is, each path r actually corresponds to a random variable z corresponding to the observations acquired by the robot. These observations follow a distribution which we can explicitly calculate from the sensor model and the existing map. Thus, it makes sense to talk about evaluating the expectation of the entropy of the map conditioned on these observations z, which are actually a function (albeit a random one) of the deterministic path r.

In the literature, this optimization problem is usually expressed slightly differently. Consider a reward function $\mathcal{J}(m, z_\tau(x_\tau))$ (think of this as how much of the map is known to the robot), for the map m and observations $z_\tau(x_\tau) = (z_{t+1}(x_{t+1}), \ldots, z_{t+\tau}(x_{t+\tau}))$ for robot configurations $x_\tau = (x_{t+1}, \ldots, x_{t+\tau})$. This set of robot configurations is equivalent to the path r used above. Using this notation, the problem becomes:

$$x_\tau^* = \arg\max_{x_\tau \in \mathcal{R}} \mathcal{J}(m, z_\tau(x_\tau)) \tag{5}$$

Typically the reward function $\mathcal{J}$ is chosen to be the **divergence** of the conditional distribution of the map m given observations z_τ. Divergence is a distance metric between

probability distributions – in this case, it is essentially the difference in entropy between the prior distribution of m and the posterior. Using the Shannon entropy, the associated Kullback-Leibler divergence of two distributions $(p(x), q(x))$ for a random variable X is:

$$D_{KL}(p||q) = \sum_x p(x) \log_2 \left(\frac{p(x)}{q(x)} \right) \tag{6}$$

As with entropy, there are many other definitions of divergence.

4 Adaptation to SfM

Occupancy grids are very common for general SLAM (**s**imultaneous **l**ocalization **a**nd **m**apping) applications because they are easy to implement and, with sufficient memory and the proper data structures, scale sufficiently well to large environments. However, structure from motion is a very special case of SLAM – for SfM to work, it needs to model the landmarks as *points*, not as voxels. Of course, it is easy to convert from a point cloud into an occupancy grid; unfortunately, the other direction is not really possible without significant quantization error.

At the end of the day, though, the occupancy grid is not the fundamental reason why we must devise a novel exploration scheme. Rather, the problem is that, as is, the information-theoretic approach is *volume*-based in the sense that it's figure of merit is essentially a measure of how much volume (or area in 2D) has been expored. In our project, we would like to be able to build a high-resolution 3D model of a building, which means that we require *lots of points*, not a large volume.

Thus, we must devise a way to reformulate the information-theoretic approach to exploration in terms of a point cloud, rather than an occupancy grid.

4.1 Modeling

There are two other obvious shortcomings in the information-theoretic approach:

1. Typically, occupancy grids model only whether or not each voxel is occupied.
2. Occupancy grids discretize (sample) the environment into a set of voxels.

Since we are interested in reconstructing models with *lots of points*, we would like our model to capture something about the number of points in each region of space. In a discrete formulation, this would mean keeping a count of points that fall into each voxel, rather than just an estimate of the probability that the voxel is occupied. Note that this is almost a trivial extension of the traditional occupancy grid formulation. Instead of each voxel being modeled by a Bernoulli random variable, it would be modeled by another random variable that takes non-negative integer values and which models discrete "arrivals." That best choice seems to be a Poisson random variable (see the subsection below for details). Once we have a known probability distribution (even with an unknown parameter – the "rate" in this case,

but recall that before the Bernoulli variable had an unknown parameter too) we can go ahead and calculate entropy and KL divergence as before.

More fundamentally, occupancy grids are unsatisfactory because they discretize space. The world is not discrete! Modeling the environment as a *continuous* space would seem to solve this problem; interestingly, it also solves the first problem at the same time. Notice that if landmarks are modeled by points in a continuum, then it is absolutely impossible for more than a single landmark to be at the exact same point. Thus, one possible representation of the environment would be a simple list of the locations of observed landmarks. Of course, a simple list is not likely to be efficient for any sort of geometric computation we may require; nevertheless, it is sufficient to store all the information in the model.

Going to a continuous representation begs the question of how to model the presence of landmarks in a probabilistic sense. In essence, we propose to apply the Poisson model we intuited in the discrete case. Specifically, we will use a **non-homogeneous, multi-dimensional Poisson process** to model the "arrival" of landmarks in the scene. Typically, Poisson process are used to model the arrivals of, say, customers in a queue or photons at a photodetector. In this case, the landmarks are being observed, rather than *arriving*, but the principle is the same.

As described below, Poisson processes have very nice mathematical properties that, among other things, allow us to write down very simple closed-form expressions for quantities of interest such as expectation and variance, and approximate values for entropy.

4.1.1 Aside: Poisson Processes

Poisson processes are one of the most elegant stochastic processes, and they have some very nice properties. Typically, Poisson processes are conceived as arrivals in time; i.e. if $\{X_t\}$ is a Poisson process with rate λ, then

$$X_s \sim \text{Pois}(\lambda s) \tag{7}$$

$$\mathbb{P}\{X_s = k\} = e^{-\lambda s}\frac{(\lambda s)^k}{k!} \tag{8}$$

A non-homogeneous Poisson process is one with a non-constant rate. For example, if $\lambda = \lambda(t)$, then

$$X_s \sim \text{Pois}\left(\int_{\mu}^{s+\mu} \lambda(t)dt\right) \tag{9}$$

Finally, an n-dimensional Poisson process is one that has multiple "time" dimensions (of course, these need not actually be time and in our case they represent space). In this case, for rate $\lambda(x, y)$ in two dimensions (for simplicity), we have

$$X_{(x,y)} \sim \text{Pois}\left(\int_{\mu}^{x+\mu}\int_{\nu}^{y+\nu} \lambda(s,t)dsdt\right) \tag{10}$$

Think of this last version as modeling fish in the ocean. Over any given x-by-y rectangle of ocean, the probability of observing k fish is $\mathbb{P}\{X_{(x,y)} = k\}$ if fish are distributed as a 2D Poisson process with rate function $\lambda(x, y)$. Of course, it is a simple matter to replace fish in the ocean with landmarks in an arbitrary environment.

4.1.2 Toward a Formal Model Definition

Now that we are somewhat familiar with Poisson processes, it is natural to model the scene as a Poisson process defined over the continuous scene-space $\mathcal{S}$. In other words, the ground-truth locations of landmarks in the environment may be approximated (or predicted) using the assumption that they are Poisson-distributed according to some rate function $\lambda(\cdot)$ defined on $\mathcal{S}$.

Formally then, we can define the **random variable** $N_s, s \subseteq \mathcal{S}$ to be the number of landmarks in some region s. According to the Poisson model, the distribution of N_s is simply:

$$N_s \sim \text{Pois}\left(\int_{\theta \in s} \lambda(\theta) d\theta\right) \tag{11}$$

Note that the rate function entirely determines the distributions of all the $\{N_s\}$, so we must choose it carefully in order to make the model at all meaningful. To get some insight as to exactly what this $\lambda(\cdot)$ represents, consider what the map will look like once we have explored everything and observed all landmarks. In this case, $\lambda(\cdot)$ will tell us the expected concentration of landmarks at each point – in other words, it is almost like a probability density function, except that we do not require it to integrate to one.

Given this intuition, we can make some intelligent modeling assumptions. For instance, it would make sense for $\lambda_0(\cdot) = \rho$ a priori. That is, the **prior** distribution of landmarks ought to be uniform with some constant rate parameter $\rho > 0$. Over time, if our exploration strategy is successful, we should observe $\lambda_t(\cdot) \to \Lambda(\cdot)$ where we use capital Λ to represent the ground-truth. This convergence is exactly what is proposed and explained in the original paper on information-theoretic exploration [3], except that it has been adapted to a continuous environment representation.

The challenge now becomes: *how to do we update the rate function?* That is, how can we merge new observations with existing ones so that over time $\lambda_t(\cdot) \to \Lambda(\cdot)$. Let us assume that we can identify landmarks with sufficiently accurate and consistent descriptors such that we can tell with a high degree of certainty when we have observed a *new* landmark and when we are simply re-observing one we have already seen before. Under this assumption (which is not too far-fetched, given the mechanisms of typical structure from motion pipelines), one natural approach is to place an n-dimensional Gaussian distribution (with some covariance that may be derived from the triangulation process) at each new landmark's coordinates. Over time, the rate function will look like a mixture of Gaussians, for which things like path planning and smoothing are well-studied.

4.2 Model Updates

There is one very big caveat to this rather simplistic model. How do we update regions of $\mathcal{S}$ efficiently? In occupancy grids, it is fairly simple to ray-trace through the grid so that observing a landmark at some location actually also implies that the voxels on the ray to that landmark are all unoccupied. We really need to do something like this in continuous space to make the exploration more efficient.

4.2.1 Bayesian Update

One way to solve this problem would be to trace out some cylinder from the current robot location to the observed landmark, find all enclosed landmarks, and – for example – increase the covariance to reflect increased uncertainty regarding their true location. In general, what we are really interested in is the conditional distribution of N_{cyl}, the number of obstacles in the cylinder. To a good approximation, the prior distribution is $N_{cyl} \sim \text{Pois}(n_{obs} + \rho V_{cyl})$ – that is, the expected number of landmarks is the number of observed landmarks plus the raw free-space density times the cylinder's volume. However, once we observe a landmark at the far end of the cylinder, we must adjust the probability of finding landmarks inside the cylinder. That is, we must find some way of expressing the following probability distribution for the location of a landmark X in the cylinder conditioned on the observation of another landmark Z located at the end of the cylinder:

$$\mathbb{P}\{X \in dx | Z = \mu_z\} = \frac{\mathbb{P}\{Z \in dz | X = \mu_x\}\mathbb{P}\{X \in dx\}}{\mathbb{P}\{Z \in dz\}} \tag{12}$$

In general, this corresponds to the following problem in one dimension:

$$\mathbb{P}\{Y \in dy | Y > \alpha\} = \frac{\mathbb{P}\{Y > \alpha | Y = y\}\mathbb{P}\{Y \in dy\}}{\mathbb{P}\{Y > \alpha\}} \tag{13}$$

$$= \begin{cases} \frac{1 \cdot p(y)}{\int_\alpha^\infty p(y)dy} & y > \alpha \\ 0 & \text{otherwise} \end{cases} \tag{14}$$

Here, we are taking variable Y in one dimension and deriving the distribution conditioned on some extrinsic measurement that says $Y > \alpha$. In our case, we might have X outside the cylinder. Regardless, the effect is to zero out part of the distribution of Y and rescale the rest so that it integrates to 1. Unfortunately, this is very, very difficult to represent and calculate efficiently in a computer program, so we must come up with some reasonable approximation.

4.2.2 A Simple Approximation

Before approximating, we should note that it is not completely infeasible to represent these sorts of conditional distributions efficiently. One approach may be to have each landmark store a mean, covariance, constraints of some form, and the corresponding scale factor which

can be numerically approximated to make the distribution integrate to unity. Each time a new landmark is observed, the program could query a kd-tree for landmarks with means in some region of space, then apply a new constraint, and calculate a new mean and scale factor (numerically). This is, of course, somewhat involved and inefficient due to its reliance on numerical integration at every update, for (in principle) a large number of obstacles.

One further note: we have not, as yet, accounted for any change in ρ, the background (a priori) rate. Ideally, each new observation should reduce ρ in some cylinder between the sensor and the landmark. However, this is again somewhat difficult to represent computationaly, so we will have to come up with a feasible alternative.

Now that we have established what *ideal* model updates might look like, we must come up with some reasonable, efficient, and realistic approximations.

At first glance, there are two obvious options:

1. *Simply remove all landmarks that lie sufficiently close to the line of sight to a newly observed landmark.* This has the benefit of simplicity (it is very easy to identify all such landmarks and remove them). However, it will probably not be very robust to errors in position/orientation: if the robot's position and orientation are not sufficiently precise, this approach would end up deleting potentially valid landmarks (and not deleting the offending ones).

2. *Move all landmarks that lie close to the line of sight to the new landmark, and also adjust their covariance to reflect that change.* In other words, approximate the true Bayesian update in the previous subsection with a Gaussian.

I believe that the second option is by far the better one, as it will be more noise-robust and because it is a truer approximation to the Bayesian update.

Now the question becomes – *what Gaussian should we use in the update?* Our objective is really just that the mean and covariance of the new distribution should be a good approximation to the true distribution was derived earlier. Of course, there are many different metrics we might use, but regardless what seems to be most important is that the mean should move away from the line of sight and that the covariance should increase along the direction of movement. For the moment, we will leave a derivation of precisely the right magnitude adjustments for future work, and outline the general algorithm in Algorithm 1, using arbitrary scalar constants α, β for the size of these adjustments.

How do we add directional variance to a covariance? In principle, covariance implies a quadratic form (i.e. an ellipsoid), so all we need to do is compute a new covariance matrix that only models variance along that direction and add it to the existing covariance. This is very simple. Suppose that the new unit direction is v, then vv^T is a dyadic covariance matrix with unit variance in the desired direction. The update is thus:

$$\Sigma \leftarrow \Sigma + \beta v v^T \tag{15}$$

Algorithm 1 Simple Model Update Algorithm

```
while True do
  pose ← GetRobotPose()
  obs ← GetNewLandmark()
  cyl ← GetLineOfSight()
  for x in cyl do
    dir = GetDirectionToLineOfSight(pose, obs, x)
    x.mean = x.mean − α · dir
    x.cov.AddDirectionalVariance(dir, β)
  end for
end while
```

4.2.3 Reformulation: Linear Filtering

This approach may work in practice; however, it is not so interesting from a theoretical perspective. A better approach may be to reconsider the cylindrical observation occlusion model. Recall that the Bayesian update from this model was what introduced non-Gaussian effects in the first place.

Suppose instead that we model observational occlusions more probabalistically, by specifying first and second order statistics. Let X and Y be existing landmarks with means μ_x, μ_y and covariances $\Sigma_x, Sigma_y$ respectively. Suppose that the robot moves around the environment, and at some point is very nearly collinear with both X and Y (without loss of generality, we assume that X is the landmark closer to the robot). If the robot observes Y to be in some location y, we can write down the following about the conditional expectation and variance of X:

$$\mathbb{E}[X|Y = y] = \mu_x + \Sigma_{XY}\Sigma_Y^{-1}(y - \mu_y) \tag{16}$$

$$\Sigma_{X|Y=y} = \Sigma_X - \Sigma_{XY}\Sigma_Y^{-1}\Sigma_{XY}^T \tag{17}$$

These formulae describe how to update the first and second order statistics of the random variable X given the new observation of Y, which is exactly what we need to do linear filtering (i.e. Kalman filtering). All that remains is to determine a reasonable way of computing cross-covariance matrix Σ_{XY} so that it captures the essense of the occlusion problem. One obvious choice is to take cross covariance to be roughly Gaussian in the orthogonal complement to the line connecting the robot (at location r) and y. We denote this line withe unit direction ℓ, and we say that the plane is spanned by the orthogonal, unit-length columns of the matrix Q. Formally, we write:

$$h(\mathbb{E}[XY^T]) \equiv e^{-\gamma\|\mu_x - \ell\ell^T\mu_x\|_2^2} \tag{18}$$

Of course, $\mathbb{E}[XY^T]$ is a matrix, so we have written it as the argument to some function $h(\cdot)$ which returns the dominant eigenvalue. Actually, we want to set Σ_{XY} equal to this value along each of the columns of Q – i.e. we want a planar cross-covariance. This is achieved as

follows:

$$\mathbb{E}[XY^T] = e^{-\gamma\|\mu_x - \ell\ell^T\mu_x\|_2^2}(I - \ell\ell^T) \tag{19}$$

This approach can be extended to a continuously operating filter, adjusting landmark distributions over time and more and more sensor measurements accumulate and as the robot moves about the environment. Algorithm 2 is a sketch of how it might be implemented.

Algorithm 2 Linear Filtering Model Update Algorithm

```
map ← EmptyMatrix()
while True do
  pose ← GetRobotPose()
  obs ← SenseLandmarks()
  for x in obs do
    if x ∉ map then
      map.AddLandmark(x)
    else
      map.UpdateLandmark(x)
    end if
    occ ← map.GetOccluders(pose, x)
    if occ is not NULL then
      submap ← map.SubMap(occ)
      submap.CalculateCrossCovariance(pose, x)
      submap.UpdateMean()
      submap.UpdateCovariance()
      map.ReincorporateSubmap(submap)
    end if
  end for
end while
```

4.3 A New Objective

The next order of business is to use the model explained and revised above to improve the original objective function $\mathcal{J}$. Since we are explicitly modeling landmark locations as a Poisson process, it makes sense to consider an objective function whose optimization converges on complete knowledge of the process, i.e. so that we know the exact distributions of all the $\{N_s : s \subseteq \mathcal{S}\}$.

Recall that each $N_s \sim \text{Pois}\left(\int_{\theta \in s} \lambda(\theta) d\theta\right)$, which implies that we will have full knowledge of the distributions of the $\{N_s\}$ if we fully know the rate function $\lambda(\cdot)$ – that is, if $\lambda(\cdot) \to \Lambda(\cdot)$ (as was suggested earlier).

Following a naive application of the information-theoretic framework from (for example) [3], we might try to maximize the expected mutual information gain (i.e. the expected difference in entropy after incorporating measurements) for all the random variables $\{N_s\}$.

Of course this is absurd, so one may consider simply maximizing the same objective but for $N_{\mathcal{S}}$, the total number of landmarks in the environment.

This seems reasonable at first. However, there is a huge problem with this approach – as we observe more landmarks and populate the environment model (the desired outcome), the entropy of $N_{\mathcal{S}}$ actually *increases*. This is easy to see, because the variance of the Poisson distribution is equal to its rate (and its expectation); i.e. as we observe more points, the uncertainty in the true number of points actually increases, which means minimizing that uncertainty would force the robot to remain idle.

Let us return to the idea of using $\lambda(\cdot)$ as a proxy for information gain. For any given (current) estimate $\lambda(\cdot)$, we can compute its distance from the true rate function using some arbitrary function norm:

$$\epsilon = \|\lambda(\cdot) - \Lambda(\cdot)\| \tag{20}$$

Of course, we will need to specify exactly what this function norm is and how to calculate it efficiently, but the principle is certainly valid. Of course, we do not really know $\Lambda(\cdot)$, so we can't even evaluate the norm in the first place. Regardless, we write the desired optimization problem as follows (where $\mathcal{X} \subset \mathcal{S}$ is some feasible set of nearby points which the robot might wish to visit):

$$p^* = \arg\min_{p \in \mathcal{X} \subset \mathcal{S}} \mathbb{E}[\|\lambda(\cdot|z(p)) - \Lambda(\cdot)\|] \tag{21}$$

This approach is simply a heuristic which chooses the point in some feasible set $\mathcal{X}$ which in expectation moves the global rate estimator $\lambda(\cdot)$ the farthest toward $\Lambda(\cdot)$.

Now we must address the concern noted above: i.e., we do not know $\Lambda(\cdot)$ a priori, and we are already doing our best to estimate it (that's $\lambda(\cdot)$!), so how do we avoid using it in the first place while still achieving the same sort of behavior? One simple approach is to simply use $N_{\mathcal{S}}$, the total number of points, as a proxy for $\Lambda(\cdot)$, which makes intuitive sense because, if $N_{\mathcal{S}}$ converges to the true value N, then up to spatial error, $\lambda(\cdot) \to \Lambda(\cdot)$. This essentially disregards all spatial information, yet it is trivial to calculate and may be a **sufficient statistic** of sorts (this requires a formal definition and proof of course). Still, the general idea would be to maximize the expected number of points in the map after moving toward the next position x. Formally, we write this as:

$$x^* = \arg\max_{x \in \mathcal{X}} \mathbb{E}[N_{\mathcal{S}}|z(x)] \tag{22}$$

4.4 Regularization

Often, we want the solution to optimization problems to have some particular property. For example, in regression analysis, we often want the solution vector to be sparse. To achieve this, we use some sort of **regularization**. Regularization is the addition of some other term(s) to the objective function that penalize (or reward) poor (or good) solutions. In the regression example, one common approach is to penalize the L_1 norm of the solution vector.

In path planning, it is quite common to penalize some particular measure of *cost*, like expected energy usage. That is a fine approach for our problem, however we can use regularization more immediately to ensure efficient exploration behavior.

4.4.1 Promoting a Uniform Distribution

Consider a toy example, in which the entire environment consists of two highly detailed object in space. As is, our objective function will drive the robot to explore one of these obejcts very carefully, but it will almost certainly not do a good job of exploring the space around the object. More importantly, it will almost certainly never even *find* the other object. In order to incentivize the planner to find the other object, we can reward it for finding landmarks that are far apart. One way to do this is to add on a term that is the sum of squared distances between all landmarks x in the map m, as follows:

$$\text{ssd}(m) = \sum_{i \neq j} \|x_i - x_j\|_2^2 \tag{23}$$

Note that the distribution of landmarks which minimizes this sum of squares penalty is the uniform distribution, which is of course what we are after.

Unfortunately, there is really no simple way to calculate this expectation of this sum of squares efficiently, particularly as the map grows larger and larger. This leads us to consider an alternative approach.

4.4.2 Downweighting Old Observations

Another way to ensure we don't get stuck in such local optima (as described in the toy example above) is to downweight old observations. That is, scale original objective by a regularizer which penalizes the possibility of observing old points. For example, consider the following regularizer $g(t, t_0)$, where t is the current time and t_0 is the original time of observation of a particular landmark (below, we reparameterize g as $g(t, x)$ for a landmark x with initial observation time t_0):

$$g(t, t_0) = e^{-\zeta(t-t_0)^2} \tag{24}$$

Since the expected locations of new landmarks are (in expectation) at the same locations as the existing landmarks, we can write down the full, regularized optimization problem as follows (where $\mathcal{O}$ is the set of expected observations):

$$p^* = \arg\max_{p \in \mathcal{X}} \sum_{x \in \mathcal{O}} g(t, x) \tag{25}$$

References

[1] Erik Nelson, "Environment Model Adaptation for Autonomous Exploration," *MS Thesis: Carnegie Mellon Robotics Institute*. Pittsburgh, PA: May 2015.

[2] Brian Yamauchi. "A frontier-based approach for autonomous exploration." In *Computational Intelligence in Robotics and Automation, 1997. CIRA '97, Proceedings, 1997 IEEE International Symposium on*, pp. 146-151. IEEE, 1997.

[3] Frederic Bourgault, Alexei A. Makarenko, Stefan B. Williams, Ben Grocholsky, and Hugh F. Durrant-Whyte. "Information Based Adaptive Robotic Exploration." In *Intelligent Robots and Systems, 2002 IEEE/RSJ International Conference on*, Vol. 1.