1 Dense 3D Reconstruction using Robot Hand-Mounted Cameras

This is a general sketch of an idea for reconstructing a dense 3D scene using a hand-mounted depth camera.

1.1 Problem

Consider a robot that has a configuration space $C \subseteq \mathbf{R}^N$. Mounted on one of the links of the robot is a depth sensor. The task is, given a sequence of noisy configurations at each time step $\{q_1, q_2, \ldots, q_T\}$, and a sequence of noisy sensor readings (point clouds) captured simultaneously $\{Z_1, Z_2, \ldots, Z_T\}$, where $Z_i = \{\mathbf{x}_1, \ldots, x_N\} \subseteq \mathbf{R}^3$, reconstruct the true dense 3D geometry of the scene.

1.2 World Representation

We will represent the world as a truncated signed distance function (TSDF). It is defined as $D(\mathbf{x}) : \mathbf{R}^3 \rightarrow \mathbf{R}$, and represents the (signed) distance to the nearest obstacle in meters, up to a truncation distance τ.

The gradient of the distance field $\nabla D : \mathbf{R}^3 \rightarrow \mathbf{R}^3$, is clearly defined and easy to compute.

1.3 General Approach

We will iteratively build up the TSDF. D_t is calculated by first estimating a configuration of the robot $q_t^* = q_t + \epsilon_t^*$ which minimizes the squared error between the sensor measurements Z_t and the previous TSDF model D_{t-1}. Then, the sensor measurements are projected back into the world given q_t^*, and the usual TSDF update is applied.

1.4 Noise Model

The robot's configuration at time step t is given by joint encoder readings q_t, which is a random value drawn from the distribution:

$$q_t = q_t^{\text{true}} + \epsilon_t$$

Where q_t^{true} are the true joint angles of the robot, and ϵ_t is a random offset drawn from some unknown random distribution Q.

$$\epsilon_t \sim Q(q_t^{\text{true}})$$

In practice, Q is highly nonlinear, and depends on the dynamics of the system (for example, the direction of gravity, or other factors that depend on the robot's joint angles). Therefore, finding an *a. priori* represention of Q may be infeasible. The task is to compute ϵ_t^*, which is an estimate of the current noisy offset.

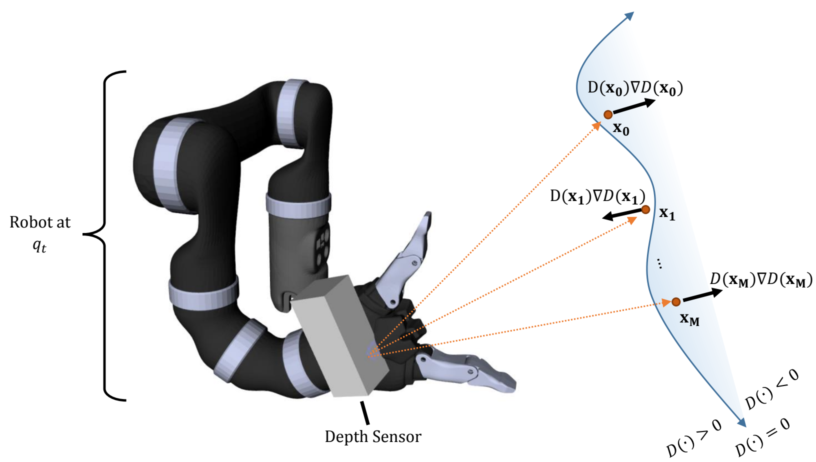

Figure 1: A diagram of the robot sensing a surface with a depth sensor. Rays from the depth sensor are shown as orange dotted lines. The point cloud $\{\mathbf{x}_1, \ldots, \mathbf{x}_M\}$ is used to determine the gradient of the distance field ∇D at each point.

1.5 Computing ϵ_t^*

We can compute ϵ_t^* by performing **gradient descent** in the configuration space of the robot. Define the **forward kinematics function** $F(q_t, Z_t) : \mathbf{R}^3 \to \mathbf{R}^3$, which merely transforms the point cloud Z_t into the world frame, given configuration $q_t \in C$. Call

$$F_t = F(q_t + \epsilon_t, z_t)$$

Then, the objective function $h(\epsilon_t, D_{t-1}) : C \to \mathbf{R}$ is given by the sum squared distances of all of the projected points in F_t:

$$h(\epsilon_t, D_{t-1}) = \sum_{x \in F_t} (D_{t-1}(x))^2$$

1.5.1 Computing the Gradient

Now, we want to find the partial differential of h with respect to ϵ_t:

$$\begin{aligned}
\frac{\partial h}{\partial \epsilon_t} &= \frac{\partial}{\partial \epsilon_t} \sum_{\mathbf{x}_t} (D_{t-1}(\mathbf{x}))^2 & (1)\\
&= \sum_{\mathbf{x}_t} \frac{\partial}{\partial \epsilon_t} (D_{t-1}(\mathbf{x}))^2 & (2)\\
&= \text{Making use of the chain rule? Since xis a function of } q_t \ldots & (3)\\
&= 2 \sum_{\mathbf{x} \in F_t} D_{t-1}(\mathbf{x}) \frac{\partial \mathbf{x}}{\partial \epsilon_t} \nabla D_{t-1}(\mathbf{x}) & (4)
\end{aligned}$$

And, since $\frac{\partial \mathbf{x}}{\partial \epsilon_t}$ is the change in $\mathbf{x}$, a projected sensor point, with respect to ϵ_t, the configuration of the robot at time t, we have (with some handwaving):

$$\frac{\partial \mathbf{x}}{\partial \epsilon_t} = J_{\mathbf{x}} \in \mathbf{R}^{3 \times N}$$

Where $J_{\mathbf{x}}$ is the serial manipulator Jacobian computed for the point $\mathbf{x}$, as though it were rigidly attached to the manipulator by the ray connecting $\mathbf{x}$ to the sensor. J has the form:

$$J_{\mathbf{x}} = \begin{bmatrix} | & \cdots & | \\ \frac{\partial \mathbf{x}}{\partial \epsilon_t(1)} & \cdots & \frac{\partial \mathbf{x}}{\partial \epsilon_t(N)} \\ | & \cdots & | \end{bmatrix} \quad (5)$$

And so:

$$\frac{\partial h}{\partial \epsilon_t} = 2 \sum_{x \in F_t} D_{t-1}(x) \mathbf{J}_x{}^{\mathrm{T}} \nabla D_{t-1}(x)$$

We will call this quantity $\nabla h(\epsilon_t)$. It has a nice physical interpretation: imagine all the points in the point cloud are attached rigidly to the robot manipulator on rods. At then end of each rod x, apply a force $D_{t-1}(x)\nabla D_{t-1}(x)$. The resulting torque on the robot's joints is proportional to $\nabla h(\epsilon_t)$ by a factor of 2 (Fig. 1).

1.5.2 Gradient Descent

Now, we just follow the update rule, setting ${\epsilon_t}^{(0)} = \epsilon_{t-1}$:

$${\epsilon_t}^{(i+1)} = {\epsilon_t}^{(i)} - \lambda \nabla h(\epsilon_t^{(i)})$$

Where λ is a learning rate. We follow the gradient until convergence, yielding ϵ_t^*, treating the joint limits of the robot as a constraint.