Automatic Reconstruction of Textured 3D Models

Ramona Stefanescu

October 18, 2018

The problem

- How can we scan the environment's surface geometry using laser range-finders and color cameras
- How can we accurately express all the data in a single coordinate system
- How can we convert separate scans into a single surface description
- How can we combine the color information from the different views to reconstruct a high quality texture

System Calibration

- A camera with lens distortions can be described by the following parameters:
  - focal length
  - principal point
  - skew coefficient
  - image distortion coefficients
- A basic 2D laser range finder or LiDAR consists of a 1D laser range finder rotating mirror deflecting the beam to different directions.
- The 3D coordinates of a scanned point p corresponding to a measured range r in the coordinate frame of the 3D scanner:

$$p = \begin{cases} \cos(\theta)(\cos(\omega) * r + dx) \\ \sin(\theta)(\cos(\omega) * r + dx) \\ \sin(\omega) * r + dz \end{cases} \tag{1}$$

Extrinsic Calibration

- We seek to find the rotation R and the translation t between the camera and the laser range finder.
- In photogrammetry, the function to minimize is usually the error of reprojecting a 3D point onto the image plane.
- This would require knowing the exact correspondence of a 3D lidar measurement and a 2D image point.
- Instead, use a planar checkerboard pattern exhibiting features visible in both sensors. The shape of the same calibration pattern can also be observed by the laser range finder and one can robustly establish a correspondence between camera and lidar data.

- For the region in the range image corrsponding to the calibration patern, a robust total least squares estimator is used to fit a plane to the set of points in 3D.
- The equation of a plane using a normal vector n and a scalar d from the origin is:

$$n \cdot x - d = 0 \tag{2}$$

- where x denotes a 3D point in the plane.
- The least square estimator results in an estimate of n_l and d_l in the laser coordinate frame and of n_c and d_c in camera coordinate frame
- Estimating the rigid transformation between the laser and camera frame can now be achieved by finding the transform that minimizes the difference in observations of the calibration plane.

Transformation between camera and lidar

- While a distance metric for point features can be defined easily, it is not so obvious for planes.
- We separate the problem by estimating translation and rotation independently.
- The translation t can be estimated by minimizing the difference in distance from the camera origin to the planes, represented in the camera coordinate system and the laser coordinate system.
- The rotation R can be estimated by minimizing the difference in the angular difference between the normals of the corresponding planes.

Transformations - cont

Iterative Closest Point (ICP)

- Given two independently acquired sets of 3D points $\mathcal{P} = \{p_i\}$ with M points and $Q = \{q_i\}$ with N points, we want to find the rigid body transformation T consisting of a rotation R and a translation t which minimizes the error metric H.

$$H(R,t) = sum_{(p_i,q_i)\in\mathcal{J}}||Rp_i + t - q_i||^2 \tag{3}$$

Normal Distributions Transform (NDT)

- NDT is one of the scan matching methods
- In this algorithm, the scanned space is divided into cells, and the input points in each cell are converted into normal distribution which characterizes the distribution of points.

Least square methods

- When fitting a line we have $y = mx + b$ and want to find (m and b) to minimize the fitting error (residual)

$$E = \sum_i ||y_i - mx_i - b||^2 \tag{4}$$

$$= \left\| \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix} - \begin{bmatrix} x_1 & 1 \\ \vdots & \\ x_n & 1 \end{bmatrix} \begin{bmatrix} m \\ b \end{bmatrix} \right\|^2 \tag{5}$$

$$= ||Y - Xh||^2 \tag{6}$$

- Find $h = [m, b]^T$ that minimizes E: $\frac{\partial E}{\partial h} = 0$
- $||Y - Xh||^2 = (Y - Xh)^T (Y - Xh)$
- $\frac{\partial E}{\partial h} = -2X^T Y + 2X^T Xh = 0$
- The analytical solution is: $h = (X^T X)^{-1} X^T Y$

Least square

- Distance between point $(x_i, y_i, 1)$ and line (a, b, d)
- Find (a, b, d) to minimize the sum of squared perpendicular distances: $E = \sum_{i=1}^{n} (ax_i + by_i + d)^2$
- $Ah = 0$, A is rank deficient
- Minimize $||Ah||$ subject to $||h|| = 1$
- $A = UDV^T \rightarrow$ h = last column of V

Cross-validation

- In cases where the size of the training data might be small, people sometime use a sophisticated technique for hyperparameter tuning called **cross-validation**
- The idea is that instead of arbitrarly picking the 1000 datapoints to be the validation set and rest training set, you can get a better and less noisy estimate of how well a certain value of k works by iterationg over different validations sets and averaging the performance across these.
- For example, in 5-fold cross-validation, we would split the training data into 5 equal folds, use 4 of them for training, and 1 for validation.
    1. kNN has a number of disadvantages:
    2. The classifier must remember all of the training data and store if for future comparisons with the test data
    3. Classifying a test image is expensive since it requires a comparison to all training images.

Linear Classification

This method will have two major components:

- a **score function** that maps the raw data to class scores
- a **a loss function**

A score function $f : R^D \rightarrow R^K$ maps the raw image pixels to class scores.

$$f(x_i, W, b) = Wx_i + b \tag{7}$$

- Each row of W is a classifier. One change in the row of W will results in the corresponding line in the pixel space to rotate.
- The biases b, on the other hands, allow our classifiers to translate the lines.

Support Vector Machine

- SVM loss is set up so that the SVM "wants" the correct class for each image to have a score higher than the incorrect classes by some fixed margin Δ.
- The score for the j-th class is the j-th element: $s_j = f(x_i, W)_j$
- The SVM loss for the i-th example is then formalized as follows:

$$L_i = \sum_{j \neq y_i} max(0, s_j - s_{y_i} + \Delta \tag{8}$$

Regularization

- The most common regularization penalty is the L_2 norm that discourages large weights:

$$R(W) = \sum_k \sum_l W_{k,l}^2 \tag{9}$$

- The regularization is not a function of the data, it is only based on the weights

$$L = \frac{1}{N} \sum_i L_i + \lambda R(W) \tag{10}$$

- The most appealing property is that penalizing large weights tends to improve generalization, because it means that no input dimension can have a very large influence on the score all by itself.
- Since the L_2 penalty prefers smaller and more diffuse weights vectors, the final classifier is encouraged to take into account all input dimensions to small amounts rather than a few input dimensions and very strong.

Softmax classifier

- Unlike the SVM which treats the outputs $f(x_i, W)$ as (uncalibrated and difficult to interpret) scores for each class, the Softmax classifier gives a slightly more intuitive output - normalized class probabilities.
- While the mapping function stays unchanges: $f(x_i; W) = Wx_i$, we replace the *hinge loss* with a **a cross-entropy loss** that has the form:

$$L_i = -log(\frac{e^f_{y_i}}{\sum_j e^{f_j}}) \tag{11}$$

Optimization

- The goal of optimization is to find W that minimizes the loss function.
- There are two ways to compute gradient: **numerical gradient** and **analytical gradient**
- The gradient tells us the direction in which the function has the steepest rate of increase, but it does not tell us how far along this we should step.
- The step size or the *learning rate* is one of the most important hyperparameters
- The gradient for the SVM is:

$$= -\left(\sum_{j \neq y_i} indicator_function(w_j^T x_i - w_{y_i} x_i + \Delta > 0\right) x_i \quad (12)$$

Mini-batch

- In large-scale applications, the training data can have an order of milions of examples.
- It is wasteful to compute the full loss function over the entire training set, instead compute the gradient over **batches** of the training data.
- The reason this work well is that the examples in the training data are correlated.
- Usually the batch size is: 32, 64 or 128. We use power of 2 because many vectorized operation implementations work faster when their inputs are sized in power of 2.

Data processing

- **Mean extraction** - it involves substracting the mean across every individual feature in the data, and has the geometric interpretation of centering the cloud of data arounf the origin.
- **Normalization** - normalizing the data dimensions so that they are of approximately the same scale. One would be dividing each dimension by its standard deviation.
- **PCA and Whitening** - steps:
  1. Extract mean
  2. Compute cov = np.dot(X.T,X)
  3. U,S, U = SVD(cov)
  4. decorrelate the data Xrot = np.dot(X,U)
  5. reduce Xrotreduced=X([:,1:100])
  6. Whitening takes the data in the eigenbasis and divides every dimension by the eigenvalue to normalize the scale: Xrot=X/sqrt(S+1e-10)

Batch Normalization

- Batch normalization - is forcing the activations throughout a network to take on a gaussian distribution at the begining of the training
- Insert the BachNorm layer immediatialy after the fully connected layer and before non-linearities
- Batch normalization can be interpred as doing preprocessing at every layer of the network
- Normalization is a simple differentiable operation
- Network with bad initialization are significantly more robust to bad initialization

L1 regularization

- the L1 regularization has the property that it leads the weight vectors to become sparse during optimization
- In other words, neurons with L1 regularization end up using only a sparse subset of thir inputs and become nearly invariant to "noisy inputs"
- **Elastic net regularization** - use both L1 and L2

Dropout

- Dropout is an extremly effecient, simple and recently introduced regularization techniques.
- While training, the dropout is implemented by only keeping a neuron active with some probability P or setting it to zero otherwise.
- **Inverted dropout** - performs the scaling at training time, leaving the forward pass at test time untouched.

Summary

- Preprocessing: center the data to have mean of zero, and normalize its scale to $[-1, 1]$
- Initialize the weights by drawing them from a gaussian distribution with standard deviation of $\sqrt{2/n}$ - Xavier initialization
- Use L2 regularization and dropout(the inverted version)
- Use bath normalization

Activation functions

- **Sigmoid**: takes a real-valued number and "squashes" it into range between 0 and 1
- A very undesirable property of the sigmoid neuron is that when the neuron's activation saturates at either tail of 0 or 1, the gradient at these regions is almost zero.
- **Tanh**: like the sigmoid neuron, its activation saturate, but unlike the sigmoid neuron its output is zero-centered.
- Tanh is always preffered to the sigmoid nonlinearity.
- **ReLU: Rectified Linear Unit** - it computes the function $f(x) = \max(0, x)$.
- Greatly accelerates the convergence of stochatic gradient descent
- Compared to tanh/sigmoid that involve expensive operation: Relu are fast to compute
- Cons: Relu can be fragile during training and can "die"

Bezier curve

- It is a parametric representation of curves used in computer graphics to model smooth curves
- Linear Bezier curves: $B(t) = (1-t) \cdot p_0 + t \cdot p_1$
- **Slerp** - spherical linear interpolation - in the context of quaternion interpolation

$$Slerp(p_0; p_1; t) = \frac{\sin[(1-t)\Omega]}{\sin \Omega} p_0 + \frac{\sin t\Omega}{\sin \Omega} p_1 \tag{13}$$

where $cos\Omega = p_0 \cdot p_1$

Ian Gooffellow book

Distributions:

- The Bernoulli distribution is a distribution over a single binary random variable
- It is controlled by a single parameter $\phi \in [0, 1]$, which gives the probability of the random variable being equal to 1.
- It has the following properties:

$$P(x = 1) = \phi \tag{14}$$
$$P(x = 0) = 1 - \phi \tag{15}$$
$$P(x = \S) = \phi^x (1 - \phi)^{1-x} \tag{16}$$
$$\mathcal{E}_x[x] = \phi \tag{17}$$
$$Var_x(x) = \phi(1 - \phi) \tag{18}$$

Information theory

- In the context of machine learning, we use few key ideas from information theory to characterize probability distributions or quantify similarity between probability distributions
- The basic intuition behind information theory is that learning that an unlikely event has occured is more informative that learning that likely event has occured
- This intuition is formalized as:
  - Likely events should have low information content, and in the extreme case, event that are guaranteed to happen should have no information content whatsoever
  - Less likely events should have higher information content
  - Independent events should have additive information
- To satisfy all 3 properties, the **self-information** of an event $X = x$ to be

$$I(x) = -logP(x) \tag{19}$$

nat is the unit and it represents the amount of information gained by observing an event of probability $\frac{1}{e}$

Information theory - cont

- We can quantify the amount of uncertainty in an entire probability distribution using the **Shannon entropy**:

$$H(x) = \mathbb{E}_{x\sim P}[I(x)] = -\mathbb{E}_{x\sim P}[logP(x)] \tag{20}$$

  the Shannon entropy of a distribution is the expected amount of information in an event drawn from that distribution
- Distribution that are nearly deterministic (where the outcome is nearly certain) have low entropy; distribution that are closer to uniform have high entropy.
- If we have two separate probability distributions $P(x)$ and $Q(x)$ over the same random variable x, we can measure how different these two distributions are using **Kullback-Leibler (KL) divergence**:

$$D_{KL}(P||Q) = \mathbb{E}_{x\sim P}\left[\log \frac{P(x)}{Q(x)}\right] = \mathbb{E}_x\left(\log P(x) - \log Q(x)\right) \tag{21}$$

- Is not a true distance measure because it is not symmetric: $D_{KL}(P||Q) \neq D_{KL}(Q||P)$

Kullback-Leibler divergence

- Suppose we have a distribution $p(x)$ and wish to approximate it with another distribution $q(x)$

$$q^* = argmin_q D_{KL}(p||q) \text{ or} \tag{22}$$
$$q^* = argmin_q D_{KL}(q||p)$$

- The choice of which direction of the KL divergence to use is problem-dependent
- When $q^* = argmin_q D_{KL}(p||q)$ we select q that has high probability where p has high probability
- When p has multiple modes, q choses to blur the modes together, in order to put high probability mass in all of them
- When $q^* = argmin_q D_{KL}(q||p)$ we select a q that has low probability where p has low probability

Structured Probabilistic Models

- Machine learning algorithms often involve probability distributions over a very large number of random variables
- Often, these probabilities distributions involve direct interactions between relatively few variables

Simulated Annealing

Check for a better explanation at Beyond Hill Climbing, Lesson2, Chapter 3, RL nanodegree

Overflow and Underflow

- **Underflow** occurs when numbers near zero are rounded to zero
- **Overflow** occurs when numbers with large magnitude are approximated as ∞ or $-\infty$
- Conditioning refers to how rapidly a function changes with respect to small changes in its inputs
- Functions that change rapidly when their inputs are perturbed slightly can be problematic for scientific computation because rounding errors in the inputs can result in large changes in the output.
- **condition number** is $\max_{i,j} \frac{|\lambda_i|}{|\lambda_j|}$
- When this number is large, matrix inversion is particularly sensitive to error in the input.

Hessian matrix

- Point where $f'(x) = 0$ are known as **critical points** or **stationary points**
- At a critical point, where $\nabla_x f(x) = 0$, we can examine the eigenvalues of the Hessian to determine whether the critical point is a local maximum, local minimum, or saddle point.
- When the Heassian is positive definite (all its eigenvalues are positive), the point is a local minimum
- When the Hessian is negative definite (all its eigenvalues are negative), the point is a local maxima
- When the Hessian has a poor condition number, gradien descent perform poorly
- Use Hessian information in the search $\rightarrow$ **Newton's method**. i
- Newton's method is based on using a second-order Taylor series expansion to approximate $f(x)$

Lipschitz continuous

- A Lipschitz continous function is a function f whose rate of change is bounded by a **Lipschitz constant** $\mathcal{L}$:

$$\forall x, \forall y, \ |f(x) - f(y)| \leq \mathcal{L}||x - y||_2 \tag{23}$$

- Convex optimization algorithms are applicable only to convex functions - functions for which the Hessian is positive semidefinite everywhere.

Karush-Kuhn-Tucker multipliers

- The KKT approach provides a very general solution to constraint optimization
- With the KKT approach, a **generalized Lagrangian** can be introduced:

$$L(x, \lambda, \alpha) = f(x) + \sum_i \lambda_i g^{(i)}(x) + \sum_j \alpha_j h^{(j)}(x) \qquad (24)$$

- where $g^{(i)}$ are called the **equality constraints** and the inequalities involving $h^{(j)}$ are called **inequality constraints**

Learning Alg

- Machine learning alg contain: an optimization algorithm, a cost function, a model, and a dataset to build a machine learning algorithm
- A machine learning algorithm is an algorithm that is able to learn from data
- Roughly speaking, unsupervised learning involves observing several examples of a random vector x, and attempting to implicitly or explicitly learn the probability distribution $p(x)$
- Supervized learning involves observing several examples of a random vector x and an associated value or vector y, and learning to predict y from x, usually by estimating $p(y|x)$
- There are no well-define boundaries between supervised and unsupervised. For example, the chain rule of probability states that for a vector $x \in \mathbb{R}^n$, the joint distribution can be decomposed as:

$$p(x) = \prod_{i=1}^{n} p(x_i | x_1, \ldots, x_{i-1}) \tag{25}$$

which means that we can solve the unsupervised problem of modeling $p(x)$ by spliting it into n supervised learning problems.

Transformations in 2D

- **Isometric transformations:** are transformations that preserve distances. In its most basic form, an isometry can be described as a rotation R and translation t.
- **Similarity transformations:** are transformations that preserve shape. They can do everything that isometric can do plus scaling
- **Affine transformations:** preserve points, straight lines and parallelism
- **Projective transformations:** transformations that maps lines to lines, but does not necessarily preserve parallelism. Despite not preserving parallelism, projective transformations does preserve collinearity of points, and it maps lines to lines.