Introduction

Let Γ represent the knowledge of the table. Γ is composed of $\gamma_1, \ldots, \gamma_n$, corresponding to the objects on the table, and $\gamma_{\text{wall}_1}, \gamma_{\text{wall}_2}, \gamma_{\text{wall}_3}, \gamma_{\text{wall}_4}$, corresponding to the four walls (edges) of the table. Each γ contains position and dimension information about the object in question.

Let λ be the command the user gives, consisting of a distance λ_d a direction (relation) λ_r and a reference object λ_f. For the sake of testing this algorithm, we will assume perfect ability to parse the command and extract this information. Therefore we assume there is a perfect correspondence between λ_f and γ_{ref}, where $\gamma_{\text{ref}} \in \{\gamma_1, \ldots, \gamma_n\}$. In addition, we assume that the direction is in the set {left, right, in front, behind} and map λ_r to the corresponding direction vectors $\{[1,0], [-1,0], [0,1], [0,-1]\}$. Underlying this assumption is the larger assumption that all commands map to a single direction vector - discussion of how we plan to improve this can be found later in this paper. We assume that the distance λ_d of a command is independent of the direction λ_r. Finally, we only deal with two dimensions, so don't consider commands that involve the z-axis.

As an example, if the command is 'five inches to the right of the bowl', then we have $\lambda_d = 5$, $\lambda_r = [1,0]$, $\lambda_f =$ the bowl.

Our goal then is to estimate a function $\mathbb{P}(x, y|\lambda, \Gamma)$ which gives the probability that a user was referring to the point (x, y) given the command λ and world Γ.

From the command and reference, we calculate a naive mean, which is the point you would select if you went exactly the distance specified by the command, in the direction specified by the command. From this we calculate four features for a log-linear model.

Feature Calculation

Calculating $\hat{\mu}$

The naive mean, $\hat{\mu}$, is what is obtained by going exactly the distance specified in the command from the edge of the object. This is calculated as follows:

$$\hat{\mu} = \begin{cases} \gamma_{\text{ref}}.center + \frac{1}{2}\gamma_{\text{ref}}.height + \lambda_d, & \lambda_r = [0,-1] \\ \gamma_{\text{ref}}.center - \frac{1}{2}\gamma_{\text{ref}}.height - \lambda_d, & \lambda_r = [0,1] \\ \gamma_{\text{ref}}.center + \frac{1}{2}\gamma_{\text{ref}}.width + \lambda_d, & \lambda_r = [1,0] \\ \gamma_{\text{ref}}.center - \frac{1}{2}\gamma_{\text{ref}}.width - \lambda_d, & \lambda_r = [-1,0] \end{cases}$$

Calculating T_1 and T_2

T_1 and T_2 create a gaussian-like distribution around the naive mean $\hat{\mu}$. T_1 penalizes points based on squared distance from $\hat{\mu}$ *in the direction of the command* (e.g. if the direction is 'left', then T_1 penalizes points based on squared distance from $\hat{\mu}$ in the x-axis). In addition, T_1 penalizes less if the command is a longer distance (e.g. 1 foot) than if it is a shorter distance (e.g. 1 inch). T_2 penalizes points based on squared distance from $\hat{\mu}$ in the orthogonal direction (e.g. if the direction is 'left', then T_2 penalizes points based on squared distance from $\hat{\mu}$ in the y-axis).

To arrive at these features, we make three assumptions about the data. First, that the data are distributed in a gaussian manner. Second, that the variance in the direction of the command (i.e. in the x direction for 'left' or 'right' and the y direction for 'in front' and 'behind') is independent of the variance in the orthogonal direction. Third, that variance in the direction of the command scales linearly with the distance of the command, while variance in the orthogonal direction is constant.

From this, our goal is to generate features that tell us about the probability of a point (x, y) given λ, Γ. Let $v = (x, y) - \hat{\mu}$. A gaussian version of this probability incorporating the above assumptions would be:

$$\frac{1}{Z} \exp\left(\frac{\langle v, \lambda_r \rangle^2}{k_1 \lambda_d}\right) \exp\left(\frac{[v - \langle v, \lambda_r \rangle \lambda_r]^2}{k_2}\right)$$

Turning these into features in a log-linear distribution, we then get:

$$T_1(x, y|\lambda, \Gamma) = \frac{1}{\lambda_d} \langle v, \lambda_r \rangle^2$$
$$T_2(x, y|\lambda, \Gamma) = [v - \langle v, \lambda_r \rangle \lambda_r]^2$$

with $v = (x, y) - \hat{\mu}$ as before.

Calculating T_3

T_3 is a hinge loss, designed to penalize points that are closer to an object other than the reference object (the one used in the command). The value should be zero for points where the reference object is the closest object, and increase linearly as (x, y) get closer to some other object. Let $p_{(x,y),\gamma_i}$ be the point on γ_i closest to (x, y). Then,

$$T_3(x, y|\lambda, \Gamma) = ||(x, y) - p_{(x,y),\gamma_{\text{ref}}}|| - \min_{\gamma_i \in \Gamma} ||(x, y) - p_{(x,y),\gamma_i}||$$

Calculating T_4

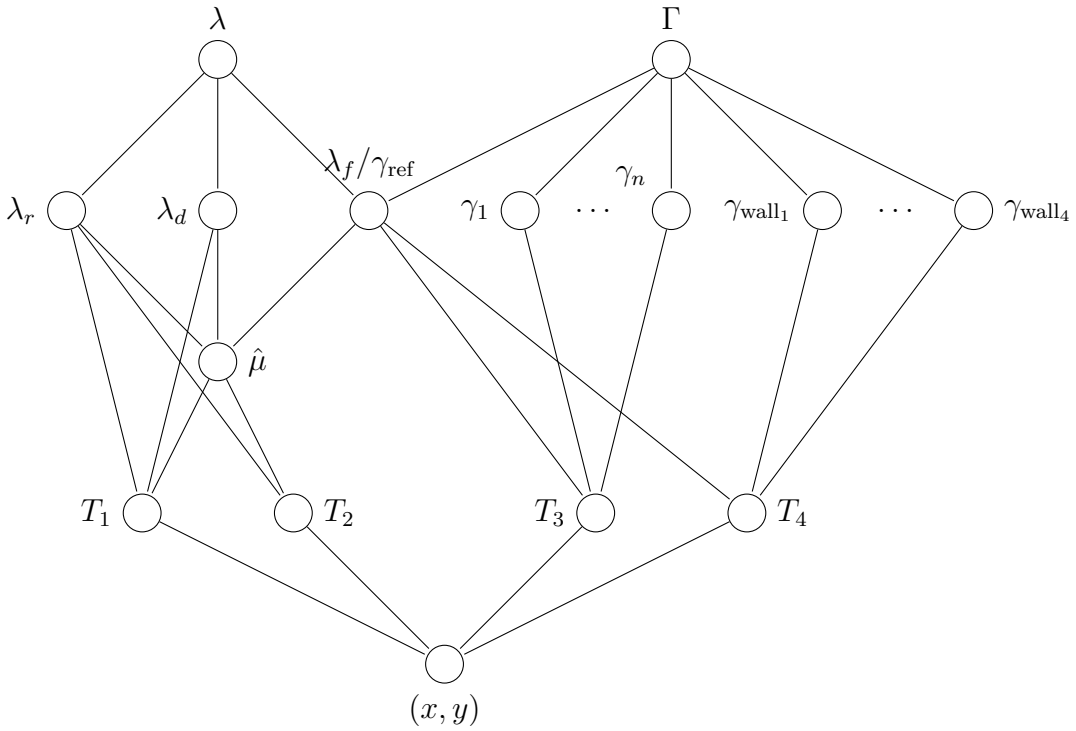
T_4 is a hinge loss, designed to penalize points that are closer to a wall than to the reference object. The value should be zero for points where the reference object is

closer than any wall, and increase linearly as (x, y) get closer to some other object. Let $p_{(x,y),\gamma_i}$ be the point on γ_i closest to (x, y). Then,

$$T_4(x, y|\lambda, \Gamma) = \max\left(0, ||(x, y) - p_{(x,y),\gamma_{\text{ref}}}|| - \min_{\gamma_{\text{wall}_i} \in \Gamma} ||(x, y) - p_{(x,y),\gamma_{\text{wall}_i}}||\right)$$

Putting the Model Together

All the features described above are related to each other via the following graphical model:



Given these features, we obtain the log-linear model

$$\mathbb{P}_w(x, y|\lambda, \Gamma) = \frac{1}{z_w(\lambda, \Gamma)} \exp\left(\sum_{c=1}^{4} w_c T_c(x, y|\lambda, \Gamma)\right)$$

where w is a vector of weights. We then learn the weights using MLE estimation.

MLE Estimation for Learning Weights

We trained the model using the data from scene #1. We assume each data point is an iid sample from $\mathbb{P}_w(x, y|\lambda, \Gamma)$. Let the 12 commands be specified by $\lambda^{(1)}, \ldots, \lambda^{(12)}$. For each command we collected 10 data points. Let X represent all the data and let $X_j^{(i)}$

be the j-th data point for the i-th command. Then the joint probability of the data points can be expressed as

$$\begin{aligned}
\mathbb{P}_w(X|\lambda^{(1)},\ldots,\lambda^{(12)},\Gamma) &= \prod_{i=1}^{12}\prod_{j=1}^{10}\frac{1}{z_w(\lambda^{(i)},\Gamma)}\exp\left(\sum_{c=1}^{4} w_c T_c(X_j^{(i)}|\lambda^{(i)},\Gamma)\right) \\
&= \left(\prod_{i=1}^{12}\prod_{j=1}^{10}\frac{1}{z_w(\lambda^{(i)},\Gamma)}\right)\left(\prod_{i=1}^{12}\prod_{j=1}^{10}\exp\left(\sum_{c=1}^{4} w_c T_c(X_j^{(i)}|\lambda^{(i)},\Gamma)\right)\right) \\
&= \left(\prod_{i=1}^{12}\prod_{j=1}^{10}\frac{1}{z_w(\lambda^{(i)},\Gamma)}\right)\exp\left(\sum_{c=1}^{4} w_c \sum_{i=1}^{12}\sum_{j=1}^{10} T_c(X_j^{(i)}|\lambda^{(i)},\Gamma)\right)
\end{aligned}$$

Our goal then is to find the argmax over w of this function. This is the same as finding the argmin of the negative log, so we have

$$\begin{aligned}
w^* &= \operatorname*{argmin}_w -\log \mathbb{P}_w(X|\lambda^{(1)},\ldots,\lambda^{(12)},\Gamma) \\
&= \operatorname*{argmin}_w \sum_{i=1}^{12}\sum_{j=1}^{10}\log(z_w(\lambda^{(i)},\Gamma)) - \sum_{c=1}^{4} w_c \sum_{i=1}^{12}\sum_{j=1}^{10} T_c(X_j^{(i)}|\lambda^{(i)},\Gamma)
\end{aligned}$$

This is the sum of exponential families, one for each i, j. Therefore the derivative with respect to w_c of the log partition function is the expected value of the c-th feature. Therefore,

$$\frac{\partial}{\partial w_c} -\log \mathbb{P}_w(X|\lambda^{(1)},\ldots,\lambda^{(12)},\Gamma) = \sum_{i=1}^{12}\sum_{j=1}^{10}\mathbb{E}_w[T_c(x,y|\lambda^{(i)},\Gamma)] - \sum_{i=1}^{12}\sum_{j=1}^{10} T_c(X_j^{(i)}|\lambda^{(i)},\Gamma)$$

We use this to perform gradient descent. In order to calculate the probabilities and the partition function, we discretize the table with a grid of step size 0.1 inches. Once we do this, we have all the components necessary to calculate the gradient, and so we perform gradient descent until the gradient falls below a threshold level.

Modeling Results