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.

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:

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

Calculating T_1 and T_2

Here, we use 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,

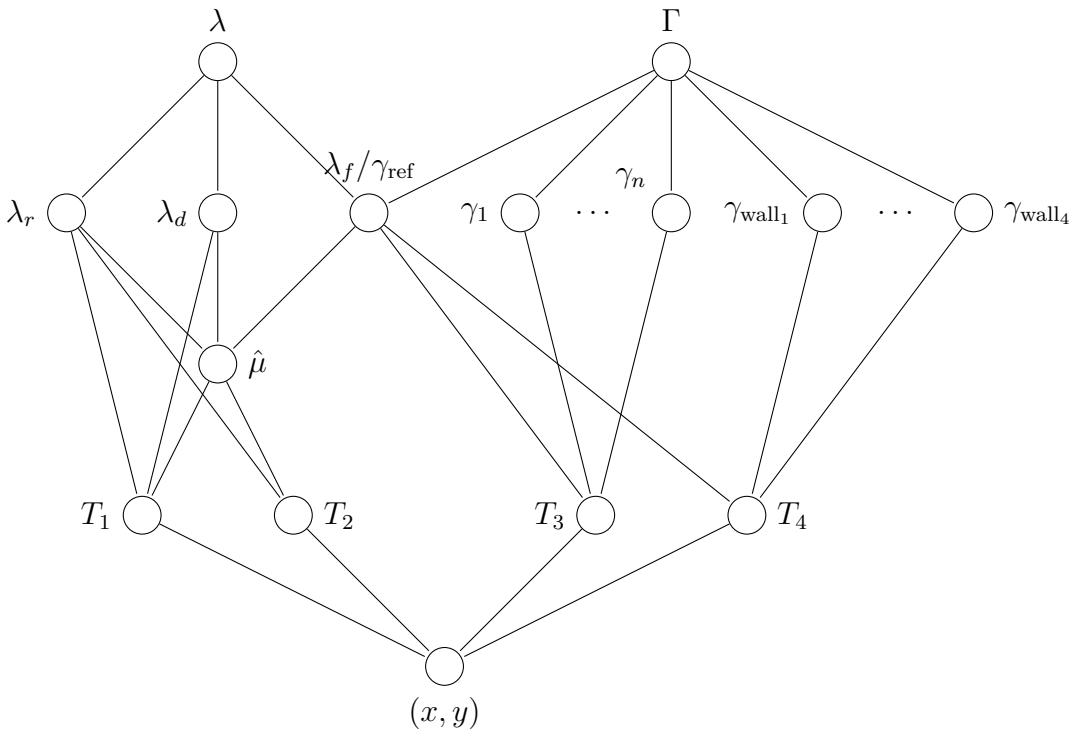
$$T_3(x, y|\lambda, \Gamma) = ||(x, y) - p_{(x,y),\gamma_{\text{ref}}}|| - \min_{\gamma_i \in \Gamma} ||(x, y) - p_{(x,y),\gamma_i}||$$

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,

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

All the terms so far described are related to each other via the following graphical model:



Putting the Model Together

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

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

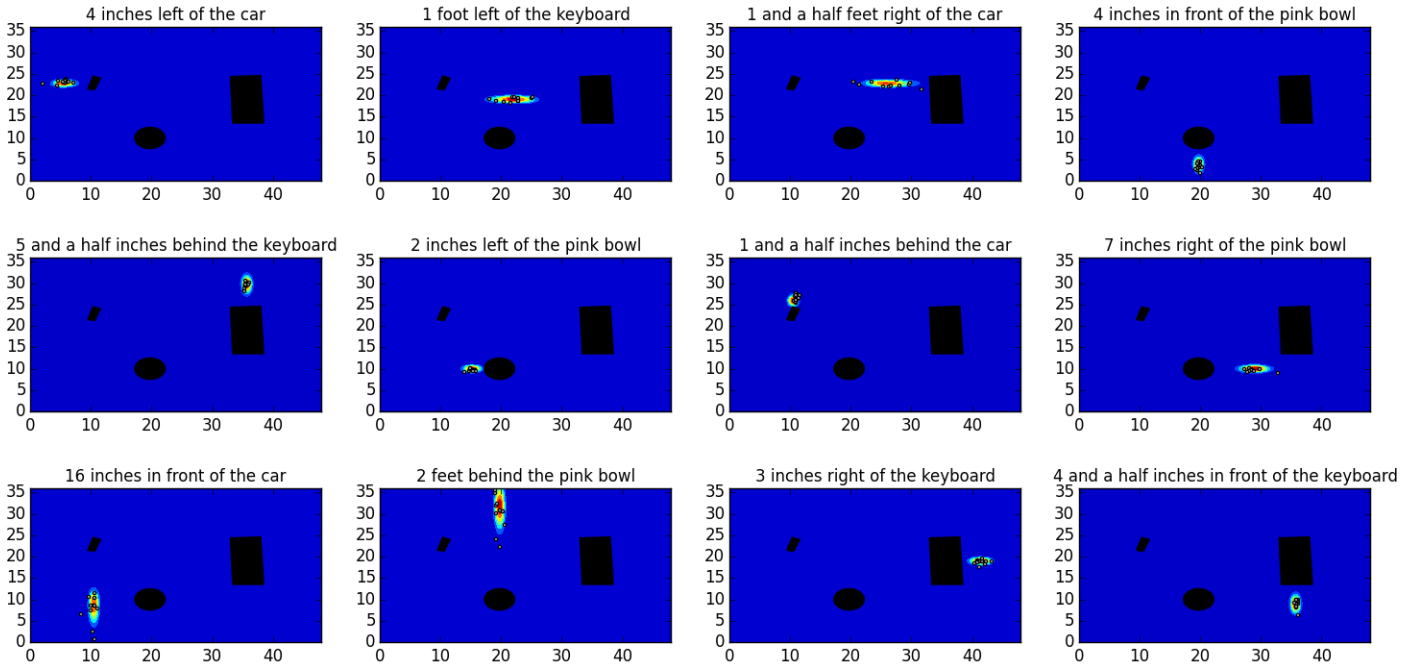
This is the sum of exponential families, one for each i, j. Therefore the derivative with respect to w_c of the log partition function is the expected value of the c-th feature. Therefore,

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

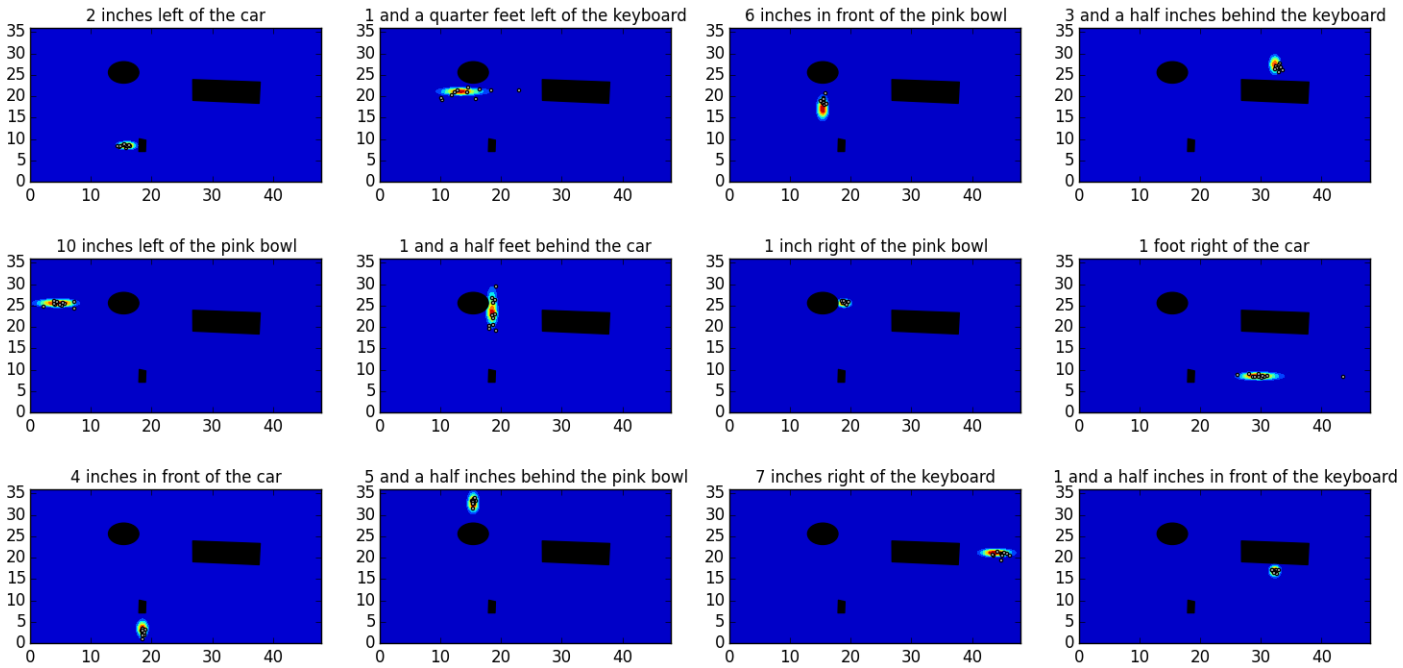
We use this to perform gradient descent. In order to calculate the probabilities and the partition function, we discretize the table with a grid of step size 0.1 inches. Once we do this, we have all the components necessary to calculate the gradient, and so we perform gradient descent until the gradient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

Modeling Results

We have successfully implemented this model in MATLAB and Python, and learned values of λ from the training data gathered in Experiment 1. Once trained using the MLE code implemented in MATLAB, our algorithm can parse a natural language command and, combined with information about the world Γ, produce a distribution over (x, y) coordinates. Plotted below are distributions generated by the log-linear model with all four features enabled:



(a) Distributions on training data

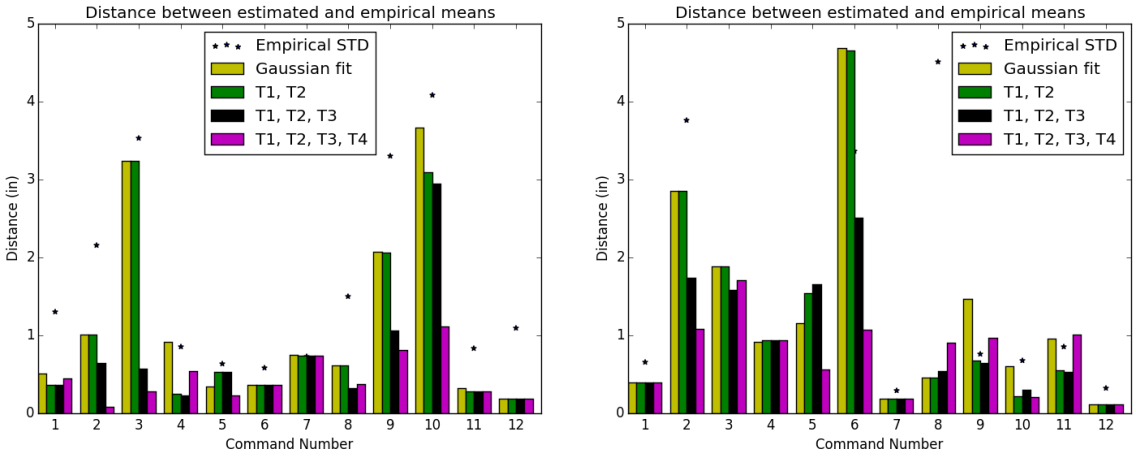


(b) Distributions on test data

Figure 1: Distributions from natural language commands and world information. Higher probability regions are in red.

Algorithm Evaluation

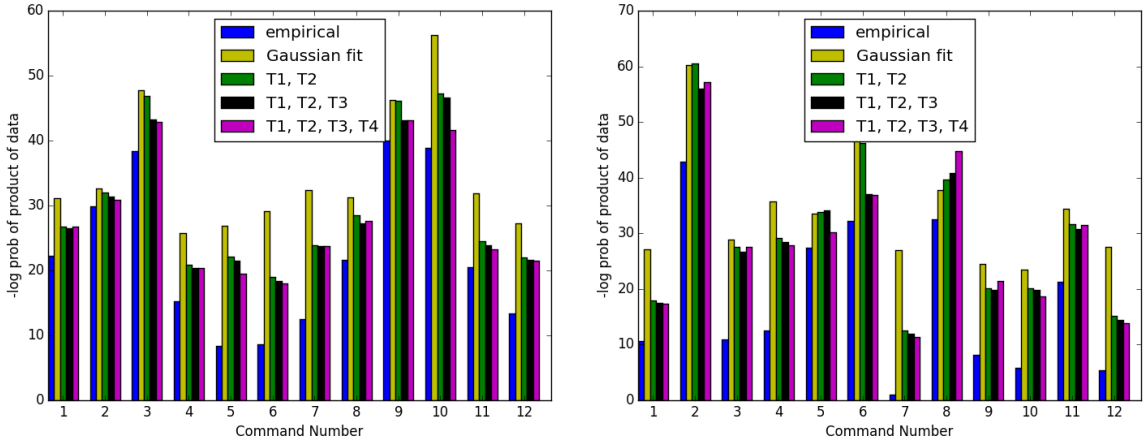
To evaluate the performance of our algorithm, we compared its performance to a baseline algorithm that fits a gaussian distribution around $\hat{\mu}$, the naive mean generated above. In addition, we compared it to log-linear algorithms trained and tested on subsets of our features, to determine the relative performance of different feature selections. All of these algorithms were compared to an empirical Gaussian distribution generated from the data in question - i.e. fitting a Gaussian to data for the specific λ, Γ pair. We used three different performance metrics to evaluate our system: the distance between predicted and empirical means, and the combined log-probability of the data given our distributions, mean-squared error (MSE) of the algorithms on our data sets. Plots of these metrics are displayed below, followed by a discussion of their meaning.



(a) Distance between empirical and distribution means on training data

(b) Distance between empirical and distribution means on test data

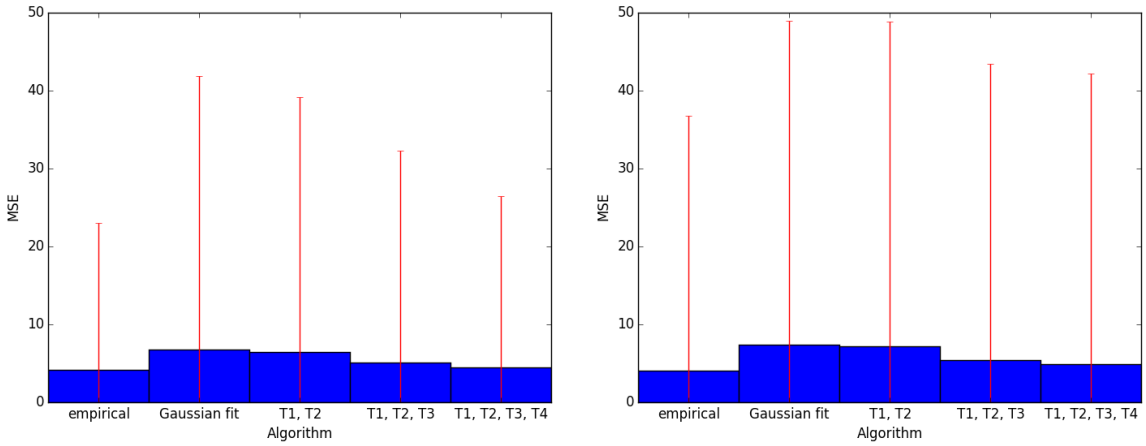
Figure 2: Distance between empirical and predicted mean for all NL phrases.



(a) Negative log probability of data by command on training set

(b) Negative log probability of data by command on test set

Figure 3: Negative log probability of algorithms



(a) MSE across all training data

(b) MSE across all test data

Figure 4: Mean-squared error by feature set. Confidence interval is twice the standard deviation of the squared error.

Discussion of Results

These evaluation metrics lead to two primary conclusions about the performance of the algorithm and possible avenues for future research. The first, and most obvious, conclusion can be derived from the command-wise plots: the performance of different feature sets, as well as the relative performance between algorithms using different

feature sets, varies across commands. This is entirely logical and consistent with the algorithms' design - in cases where objects and walls in Γ have minimal effect, the algorithms including T3 and T4 hinge losses would be expected to perform similarly to the gaussian fit and algorithms only using the first two features. As well, in commands with short distances, the distance term in feature T1 has little effect on the variance of the distributions. However, in commands with larger distances, the non-Gaussian log-linear models have a superior performance. Overall, the larger feature sets maximize log-probability of the data, although the distance between the empirical and predicted mean varies across the commands. For nearly every command, the predicted mean is within one standard deviation of the empirical mean.

The second conclusion derives from the MSE evaluation. As shown in figure 4, increasing the size of the feature sets lowers the mean-squared error across all distributions. However, the confidence interval with p=0.95 shows that the MSE of every feature set is statistically indistinguishable. Given that we only have 10-12 data points per λ, Γ pair, a large confidence interval is to be expected. In particular, the magnitude of the standard deviation is similar to that of the empirical distribution, demonstrating that the algorithms perform similarly in that regard. Even so, the small data sets do not allow a definitive conclusion on the relative performance of each feature set. While the complete feature set does look promising in regards to log-probability, distance between predicted and empirical means, and MSE, we cannot definitively say that the performance of the log-linear model with all features is superior to Gaussian or log-linear models with smaller feature sets. Discussion of how we plan to resolve this issue, and expand the capabilities of our model, is included below.

Future Research

Conclusion