Learning to Follow Navigational Route Instructions

Nobuyuki Shimizu
Information Technology Center
University of Tokyo
shimizu@r.dl.itc.u-tokyo.ac.jp

Andrew Haas
Department of Computer Science
State University of New York at Albany
haas@cs.albany.edu

Abstract

We have developed a simulation model that accepts instructions in unconstrained natural language, and then guides a robot to the correct destination. The instructions are segmented on the basis of the actions to be taken, and each segment is labeled with the required action. This flat formulation reduces the problem to a sequential labeling task, to which machine learning methods are applied. We propose an innovative machine learning method for explicitly modeling the actions described in instructions and integrating learning and inference about the physical environment. We obtained a corpus of 840 route instructions that experimenters verified as follow-able, given by people in building navigation situations. Using the four-fold cross validation, our experiments showed that the simulated robot reached the correct destination 88% of the time.

1 Introduction

Advances in technologies supporting task-oriented, deep semantic understanding are important for seamless interactions with computers. The following of navigation instructions by a computer, for example, is one area where such advances would be beneficial. Although the vocabulary in this domain is limited and strong domain knowledge is available, the need to understand unrestricted natural language instructions poses a significant challenge due to the rich structures and creative expressions that people use. Consider these quotes taken from a corpus of building navigation instructions given by undergraduate students at a U.S. university.

> "Just head straight through the hallway ignoring the rooms to the left and right of you, but while going straight your going to eventually see a room facing you, which is north, enter it."

> "Head straight then make a right turn, then head straight again ignoring the first left and right rooms, but not ignoring the second room to the right, enter it."

The examples contain a number of typos and grammatical errors. They are clearly not from a restricted command set, like one might imagine, and a syntactic parser would obviously have trouble parsing them.

We have addressed this problem by developing a model, to be used as a natural language interface for mobile robots, which we trained using machine learning methods. Our aim is to create a system capable of understanding instructions given in unrestricted natural language and display its understanding in a simulated environment. The model requires three inputs.

- An instruction for reaching a destination given in unrestricted English.
- An abstraction of the real world.
- The current position of and the direction in which the instruction follower is facing.

The task is to find the location corresponding to the destination in an instruction.

To simplify the task, a map of a building is abstracted to a graph in which the edges represent hallways and doorways and the points represent where the edges end (such as inside an office or at the end of a hallway) or meet each other (such as at an intersection of hallways or at a doorway). Since it is difficult to solve the complete input-output problem, a typical system breaks the task down into two components.

Front End: $RouteInstruction \rightarrow ActionSequence$
Back End: $ActionSequence \times Map \times Start \rightarrow Goal$

The front end of the system is an information extraction system that extracts how to move in accordance with a route instruction. The back end is a reasoning system that takes a sequence of moves and finds the destination on the map.

While most previously proposed systems are based on heuristics crafted by a person, we use the novel formulation developed by Shimizu [Shimizu, 2006], which reduces the task of following the route instructions to a sequential labeling problem. We use their frame-based flat knowledge representation to represent an **action**. An **action frame** is a tuple with four slots $\langle (G|E), (H|D), (L|R|S), (1|2|3|Z) \rangle$. The first slot takes either G or E, where G means that the desired action is to advance down the hall, then turn and face a door or side hall, and E means that the desired action is to actually enter a doorway. In the second slot, H means "a hallway" and D means "a door." In the third slot, L means "left," R means "right," and S means "straight," i.e., there is no turn in the action. The last slot represents the ordinal number that modifies an entity, and Z means either "last" or "at the end." For example, if the entity is "the third door," the ordinal number slot contains 3. If the entity is "the door at the

end," the ordinal number slot contains Z. If the third slot contains S, the last slot must contain Z. Although "first right" and "second left" make sense, there is no object that satisfies "first straight," "second straight," and so on. For example, $\langle G, H, R, 2\rangle$ means [*advance* to the second hall on the right, *turn* to face it]. $\langle E, D, L, 3\rangle$ means [*advance* to the third door on the left, *turn* to face it, *enter* it]. $\langle E, D, S, Z\rangle$ means [*advance* to the door straight ahead of you, *enter* it]. When impossible combinations, such as S with 1, are eliminated from the corpus, the total number of possible actions, i.e., labels, is 15.

Shimizu concatenated the values of frame slots to form one label for each token [Shimizu, 2006]. They annotated the instructions using the BIO tagging scheme used in syntactic chunkers [Ramshaw and Marcus, 1995] and reduced the problem to segmentation and labeling of segments, much like the noun phrase chunking in the CoNLL-2000 shared task [Sang and Buchholz, 2000]. Their use of a simplifying assumption – which states that an action can be represented by an atomic label – enables the use of machine learning methods such as the linear-chain conditional random field (CRF) [Lafferty *et al.*, 2001; Sha and Pereira, 2003], but also results in data sparseness for rare actions and lessens the performance of the learning system. We avoid this problem by combining two chains of sequential labeling into one decoding algorithm.

We next review related work. Then we evaluate Shimizu's method using a larger corpus and analyze the errors it produced. We then describe our frame-segmentation model, discuss the results of testing, examine the generalizability of our approach, and conclude with a short summary.

2 Related Work: Rule-based Systems

Previous work on giving instructions to an agent include using the SHRDLU program by Winograd [1972] to give instructions to a simulated robot. SHRDLU is a rule-based program that works in a tabletop environment. This restricted world enables SHRDLU to deduce what it can do in the environment. Combined with a basic memory, SHRDLU is capable of holding a text-based conversation.

Some researchers concentrated on the understanding of simple spatial commands in small space and focused on grounding, that is, mapping the language to the physical object. Notable works include that of Roy [Roy, 2005], who used an actual robotic hand in a tabletop environment, and Skubic et al. [Skubic *et al.*, 2004], who used a mobile robot to generate spatial language from occupancy grid maps and integrated it into a dialog.

The Instruction-based Learning for Mobile Robots (IBL) project [Bugmann *et al.*, 2004] used a miniature model of a city center with many different landmarks and roads and had a remotely controlled human-operated robot navigate through the model. Participants were asked to give the operator instructions for guiding the robot. From these instructions, a corpus of 144 spoken route instruction sets was obtained. The instructions were then annotated with functional action primitives, and a robotic system capable of following these primitives was implemented. Testing showed that people following the instructions were able to reach the destination 83% of the time, while the robot following hand-translated primitives succeeded 63% of the time.

To see how a system performs, it is important to know how successful the system is in reaching the destination from given instructions that human can follow. "Instruction follow through" measures this performance. Using this metric, their result would translate to a 75.9% success rate. We note that only 14% of the routes that were automatically annotated actually led the robot to the goal, resulting in a 16.5% success rate in terms of instruction follow through. All of the automated translations from route instructions to action primitives were done using heuristics rather than statistical learning.

MacMahon and Stankiewicz [MacMahon and Stankiewicz, 2006] took a similar approach to following route instructions. The main difference was the simulated environment - they used one very similar to a first-person shooter game like DOOM. Unlike the IBL project, they annotated the instructions with a semantic grammar. In their MARCO system, they feed a semantic tree to a "content framer" and then to an "instruction modeler," which compiles the tree into a compound action specification. During this procedure, any information missing from the tree is filled in using hand-built heuristics in accordance with a model of the environment. Their corpus consisted of 786 natural language route instructions produced by six participants. They found that a human instruction follower was capable of reaching the correct destination with a mean success rate of 69% for 682 of the instructions. For comparison, they also handcrafted gold standard semantic parses of the same instructions and fed them into the MARCO system. The system successfully followed 61% of the instructions. Again, this result translates to a 88.4% success rate in instruction follow through.

When the missing information was not inferred, system performance dropped to 28% (40.6% with instruction follow through). Considering that the performance of today's best syntactic parser is around 90% for constituents and less than 50% for whole sentences, we can infer that the performance of an end-to-end MARCO system would not be very high.

3 Supervised Machine Learning for Understanding Route Instructions

Given an input x, a multi-class linear classifier uses a linear combination of features to predict to which class (labeled as y) x belongs. That is, if the input feature vector is $\phi(x, y)$, then the most likely class y has a score $y = argmax_{y'}(\mathbf{w}^\top \phi(x, y'))$ where $\mathbf{w}$ is a real vector of weights. If we would like to predict the genre of a book, then x would be the text of a book and y the genre. For example, the features could be words in the text, and a word 'extraterrestrial' is heavily weighted for science fiction but not so much for comedy. The weight vector is learned from a set of labeled training samples.

We now explain slightly more complex prediction problems. Suppose we want to predict both the writing style (journalistic, scholarly and so on) and the genre of a book. In this case, y would be a combination of writing style and genre. If writing style and genre is independent of each other, we

may independently predict the writing style using one classifier and the genre with another; doing so will improve the performance since there would be more training examples for a label, say, science fiction, than for a combination of labels such as journalistic-comedy. Furthermore, if writing style and genre are not independent, we may predict using the sum of the scores produced by the two classifiers above and a classifier that predicts combined labels. Since a score is a linear combination of features, the summation simply results in one linear classifier. In this case, a label could be factorized into three parts, and we will call this label factorization.

The CRF applies this idea of label factorization to a sequence of labels, assuming a label is affected by only a label before and after it in the sequence. We follow the framework for structured problems described by Bartlett et al. [Bartlett *et al.*, 2004]. Each structure, or label y is now defined to be a set of parts. We assume that the feature vector for an entire structure y decomposes into a sum over feature vectors for individual parts as follows: $\phi(x, y) = \sum_{r \in y} \phi(x, r)$. Note that we have overloaded the symbol ϕ to apply to either a structure y or its parts r. Given the input sequence $x = (x_1, x_2, x_3, ...x_n)$ and the output sequence $y = (y_1, y_2, y_3, ...y_n)$, the linear chain CRF we used is an undirected graphical model where for $0 < j < n$, y_j are the states in the graph (we write state$\langle y_j \rangle_j$ to indicate these parts), and the probabilistic dependency is modeled with a linear chain of transition parts *trans*$\langle y_j \rightarrow y_{j+1} \rangle_j$ between the state y_j and y_{j+1}.

For each sentence x, we have binary state features $\phi_p(x, \textit{state}\langle y'_j \rangle_j)$ and binary transition features $\phi_q(x, \textit{trans}\langle y'_j \rightarrow y'_{j+1} \rangle_j)$, where p and q range over the number of features for state and transition, respectively. The conditional probability of y given x is:

$$P(y|x) = \tfrac{1}{Z_x} exp(\quad \sum_{j,q} \mathbf{w}_p^\top \phi_p(x, \textit{state}\langle y'_j \rangle_j) + \sum_{j,p} \mathbf{w}_q^\top \phi_q(x, \textit{trans}\langle y'_j \rightarrow y'_{j+1} \rangle_j))$$

where w is the weight for each feature and Z_x is a normalization constant for each x.

$$Z_x = \sum_y exp(\quad \sum_{j,q} \mathbf{w}_p^\top \phi_p(x, \textit{state}\langle y'_j \rangle_j) + \sum_{j,p} \mathbf{w}_q^\top \phi_q(x, \textit{trans}\langle y'_j \rightarrow y'_{j+1} \rangle_j))$$

Given a training set $T = \{(x^i y^i)\}_{i=1}^m$, the weights in the CRF is learned by maximizing the log-likelihood, $\sum_j log P(y^i|x^i)$. A detailed description of the CRF can be found in [Lafferty *et al.*, 2001; Sha and Pereira, 2003]. The inference uses the Viterbi algorithm much like a Hidden Markov Model in order to find $y = argmax_{y'} P(y'|x)$, which is

$$y = argmax_{y'} (\quad \sum_{j,q} w_p \phi(x, \textit{state}\langle y'_j \rangle_j) + \sum_{j,p} w_q \phi(x, \textit{trans}\langle y'_j \rightarrow y'_{j+1} \rangle_j))$$

Shimizu [2006] applied the CRF to the segmentation task in a straightforward manner by combining the slots of an action frame into one atomic label. He improved performance by discarding action sequences that did not make sense in context, for example, a reference to the second door on the left when there is only one door on the left. This was done by modifying the Viterbi algorithm used to extract the best action sequences from the model. This is effectively the machine learning version of filling the missing information in the semantic tree in accordance with a model of the environment [MacMahon and Stankiewicz, 2006].

Table 1: Frequencies of Labels in Dataset

Label	Count	Avg. length	Label	Count	Avg. length
GHL1	277	8.8	EDL1	201	10.1
GHL2	12	6.9	EDL2	169	11.4
GHLZ	82	15.0	EDL3	47	13.0
GHR1	315	10.4	EDR1	152	10.4
GHR2	6	15.3	EDR2	84	12.7
GHRZ	110	11.8	EDR3	8	12.5
EDLZ	47	14.6	EDRZ	25	18.0
EDSZ	112	16.0			

We evaluated the method using 840 samples of route instructions collected in the same manner. In the corpus, there were 1.96 actions per instruction on average. There were 389 cases in which a period (i.e., a boundary) appeared in the middle of a segment, which corresponded to an action. This number underestimates the number of sentence boundaries since sentence endings are often not clearly marked with punctuation in unedited text.

There were 45 distinct routes, and on average there were three destinations given that there was one starting point for these distinct routes. There were 18.6 instructions on average for each route. If we simply memorized the map and learned to associate each starting point with a destination, we would theoretically reach the correct destination around 30% of the time. However, since there were many more than three rooms within any two actions, the probability of reaching the correct destination by simply randomly picking doors was much lower than 30%.

Table 1 lists the frequencies of action frames in the corpus and the average length of the text segments describing them. The distribution of labels was highly skewed; there were many more GHL1 labels than GHL2 ones.

3.1 Error Analysis

The problem with the formulation used for the linear-chain CRFs is that all the attributes for one action are represented by a single label. For example, although "take the first door on the right" and "take the first door on the left" differ by only one word, they have completely different labels, EDR1 and EDL1, respectively. Ideally, "take X door" should contribute to the learning of the *E* and *D* slot fillers, and "first" should help learn the ordinal *1* attribute of both labels. However, with linear-chain CRFs, a label is atomic, so no such factorization takes place. This obviously causes data sparseness for each label. In fact, GHR2 was always mistaken for GHR1. This tendency is evident in Table 2. The left-hand side of $\rightarrow$ represents the CRF result, and the right-hand side represents the results of our frame-segmentation model, which is described in the next section. With our model, both the recall percentage and F_1 measure for GHR2 were improved.

Table 2: Results by Label (CRF → FS)

Label	Precision (%)	Recall (%)	$F_{\beta=1}$ (%)
GHL1	77.94 → 78.15	82.64 → 79.62	80.22 → 78.88
GHL2	100.00 → 100.00	25.00 → 83.33	40.00 → 90.91
GHLZ	76.25 → 79.49	74.39 → 75.61	75.31 → 77.50
GHR1	66.67 → 69.20	69.13 → 67.11	67.87 → 68.14
GHR2	**0.00** → 50.00	**0.00** → 33.33	**0.00** → 40.00
GHRZ	73.96 → 74.74	65.14 → 65.14	69.27 → 69.61
EDL1	81.41 → 84.74	84.38 → 83.85	82.86 → 84.29
EDL2	79.04 → 82.91	85.16 → 84.52	81.99 → 83.71
EDL3	78.95 → 82.93	66.67 → 75.56	72.29 → 79.07
EDLZ	53.66 → 77.55	47.83 → 82.61	50.57 → 80.00
EDR1	82.17 → 83.44	86.58 → 84.56	84.31 → 84.00
EDR2	81.01 → 89.33	80.00 → 83.75	80.50 → 86.45
EDR3	50.00 → 100.00	12.50 → 75.00	20.00 → 85.71
EDRZ	50.00 → 40.74	23.81 → 52.38	32.26 → 45.83
EDSZ	75.00 → 80.39	81.82 → 74.55	78.26 → 77.36

4 Frame-segmentation Model

To overcome the data sparseness problem, our proposal is to take one further step in label factorization; instead of combining the slots into one atomic label, we factorize each action frame and the slots associated with it into a segment.

In our model, we use segments and transitions between neighboring segments, as well as sequences of slot fillers within a segment and transitions between these fillers.

The frame-segment structure takes the frame-based knowledge representation and lays it on top of the input tokens. Each token carries four slot fillers. If the token at index r carries slot filler P, the structure includes *state*$\langle P\rangle$. We also write *state*$\langle P\rangle_r$ to indicate with which token the part is associated. If the tokens at indexes $r-1$ and r are in the same segment, the segment also includes *trans*$\langle P \rightarrow P\rangle$ (since the tokens at $r-1$ and r belong to the same segment, their slot fillers are the same). Such parts are called *transitions*. They represent slot fillers and transitions between the slot fillers.

If a segment extends from the token at q to the token at r inclusive, the structure includes *seg_state*$\langle I, q, r\rangle$. Also if the token at $q-1$ is the last token in a segment, while the token at q is the first token of the next segment, the second layer (segments) includes *seg_trans*$\langle I \rightarrow I\rangle$. This part is also a transition. These parts indicate the beginning and ending of segments and the transitions between the segments.

Note that since G always occurs with H, and E always with D, we can eliminate $(H|D)$ from the knowledge representation. However, doing so reduces the performance of the system. We postulate that having $(H|D)$ in the model allows the system to place more importance on the distinction between "going to a hall" and "entering a door". We limit the following discussion to the case where $(H|D)$ is included.

The score of the frame-segment structure is factorized as the sum of the scores of all the parts, in exactly the same way that the conditional random field method factorizes the score of the undirected linear-chain graph.

In the next section, we describe the decoding algorithm for the frame-segment structures.

4.1 Decoding

The decoder finds $argmax_y \mathbf{w}^\top \phi(x, y)$. We state the recursive version of the Viterbi algorithm for the Markov model.

$$\begin{aligned} V_m(i, L) = \; & max_{L'} V_m(i-1, L') \\ & + \mathbf{w}^\top \phi(x, \textit{trans}\langle L' \rightarrow L\rangle_i) \\ & + \mathbf{w}^\top \phi(x, \textit{state}\langle L\rangle_i) \\ V_m(0, L) = \; & 0. \end{aligned}$$

where $\langle L_{(y,n)}, i\rangle$ is the state part for the tag at n, and $\langle L_{(y,n-1)} \rightarrow L_{(y,n)}, n-1, n\rangle$ is the transition from $n-1$ to n.

The following is the Viterbi algorithm for the semi-Markov model [Cohen and Sarawagi, 2004].

$$\begin{aligned} V_{sm}(r, L) = \; & max_{L', d=1..r} V_{sm}(r-d, L') \\ & + \mathbf{w}^\top \phi(x, \textit{seg\_trans}\langle L' \rightarrow L\rangle_{r-d}) \\ & + \mathbf{w}^\top \phi(x, \textit{seg\_state}\langle L, r-d, r\rangle) \\ V_{sm}(0, L) = \; & 0 \end{aligned}$$

where $\langle L' \rightarrow L, r-1, r\rangle$ is the transition part, which tells us that as the sequence moved from $r-1$ to r, the transition from label L' to L took place. $\langle L, r-d, r\rangle$ is the segment part, which tells us that the sequence had a segment from $r-d$ to r labeled with L.

We attempt to combine the decoding algorithm for the Markov model and semi-Markov model. First, we modify the Viterbi algorithm for the Markov model so that the chain does not necessarily start at 0.

$$\begin{aligned} V_{m'}(q, r, L) = \; & max_{L'} V_{m'}(q, r-1, L') \\ & + \mathbf{w}^\top \phi(x, \textit{trans}\langle L' \rightarrow L\rangle_r) \\ & + \mathbf{w}^\top \phi(x, \textit{state}\langle L\rangle_r) \\ V_{m'}(q, q, L) = \; & 0 \end{aligned}$$

We embed this $V_{m'}$ into V_{sm}.

$$\begin{aligned} V_{smm}(r, L) = \; & max_{L', d=1..r} V_{smm}(r-d, L') \\ & + \mathbf{w}^\top \phi(x, \textit{seg\_trans}\langle L' \rightarrow L\rangle_{r-d}) \\ & + \mathbf{w}^\top \phi(x, \textit{seg\_state}\langle L, r-d, r\rangle) \\ & + max_{L''} V_{m'}(r-d, r, L'') \\ V_{smm}(0, L) = \; & 0 \end{aligned}$$

Our frame-segment model for following route instructions is as follows. First, we define $V_E(q, r)$ to be $V_{m'}(q, r, L)$, with L_E as the sole label. The labels of this model comprise a singleton set. This means that V_E examines the score of sequence $[L_E, L_E, L_E, .., L_E]$, which is the only sequence the model can have since there is only one label L_E. Similarly, we define $V_G, V_H, V_D, V_S, V_R, V_L, V_1, V_2, V_3, and V_Z$. The frame-segment model thereby becomes

$$\begin{aligned} V_{smm}(r, L) = \; & max_{L', d=1..r}(V_{smm}(r-d, L') + Z) \\ V_{smm}(0, L) = \; & 0, \end{aligned}$$

where

$$\begin{aligned} Z = \; & \mathbf{w}^\top \phi(x, \textit{seg\_trans}\langle L' \rightarrow L\rangle_{r-d}) \\ & + \mathbf{w}^\top \phi(x, \textit{seg\_state}\langle L, r-d, r\rangle) \\ & + max_{L_E, L_G}(V_E(r-d, r), V_G(r-d, r)) \\ & + max_{L_H, L_D}(V_H(r-d, r), V_D(r-d, r)) \\ & + max_{L_S, L_R, L_L}(V_S(r-d, r), \\ & \quad V_R(r-d, r), \\ & \quad V_L(r-d, r)) \\ & + max_{L_1, L_2, L_3, L_Z}(V_1(r-d, r), \\ & \quad V_2(r-d, r), \\ & \quad V_3(r-d, r), \\ & \quad V_Z(r-d, r))). \end{aligned}$$

We now explain the dynamic programming version of this algorithm. To shorten the discussion, we provide only a brief overview of how to integrate the backend into the algorithm. First, one Viterbi extends a segment $[q, r-1]$ to $[q, r]$, considering the state indexed by r. While this is being done, we evaluate the scores for the slot filler combinations and sort them for each segment. To do this, we make use of a sorted tree structure such as a red-black tree. The other Viterbi extends a sequence of segments $[0, q]$ to $[0, r]$, considering the state indexed by $[q, r]$. While this is being done, we find the score for the highest scoring combination that does not violate the physical environment given the current location on the map, such as entering a door where there is no door. This is much like the path-Viterbi described by Shimizu [Shimizu, 2006]. We retrieve the best scoring action_frame from the sorted tree that fits the environment and add the score of the frame to that of the action segment sequence, $[0, r]$. The algorithm returns the best score for the frame-segment structure.

To enable recovery of the structure, we simply maintain back pointers to the items that gave rise to each item in the dynamic programming table. This is exactly like maintaining back pointers in the Viterbi algorithm for sequences.

The run-time complexity of this decoding algorithm is O($c_s c_f n^2$) compared to O($c_s^2 c_f^2 n$) for linear-chain CRFs. The constant c_s is the number of slots, c_f is the number of fillers for a slot, and n is the length of the instruction. The decoding algorithm for our frame-segment model scales better than the Viterbi algorithm for the Markov model as the number of actions increases.

Although fitting a log-linear model (such as CRF) is generally thought to have a superior performance than using perceptron, it requires the computation of the marginal probability of parts. Since an effective dynamic programming algorithm for computing this value is not known, we used the averaged perceptron [Collins, 2002] to train our model.

Algorithm
4.1: AVERAGED PERCEPTRON($T, \{(x^i, y^i)\}_{i=1}^n$)

$k := 0$; $\mathbf{w}_1 := \mathbf{0}$; $c_1 := 0$;
for $t := 1$ **to** T
  for $i := 1$ **to** n
    $y := argmax_{y'}(\mathbf{w}_k^\top \phi(x^i, y'))$
    if $(y = y^i)$
      $c_k := c_k + 1$;
    else
      $\mathbf{w}_{k+1} := \mathbf{w}_k - \phi(x^i, y) + \phi(x^i, y^i)$;
      $c_{k+1} := 1$; $k := k + 1$; $c_k := c_k + 1$;
return $(\{(\mathbf{w}_1, c_1), ..(\mathbf{w}_k, c_k)\})$

We average the resulting weight vectors to obtain the model parameters of the classifier.

We tested the model using a four-fold cross validation in which each instruction was randomly assigned to a partition. The evaluation metrics were precision P (fraction of output chunks that exactly matched the reference chunks), recall R (fraction of reference chunks returned by the chunker), and their harmonic mean, the F_1 score ($F_1 = 2PR/(P + R)$).

Table 3: Overall Results

Exact match	Recall	Precision	F_1
CRF	75.4%	75.3%	75.3%
CRF w/Backend	76.0%	75.6%	75.8%
Perceptron	64.4%	58.1%	61.1%
Perceptron w/Backend	66.6%	60.2%	63.2%
Frame-Segment	75.8%	73.6%	74.7%
Frame-Segment w/Backend	77.0%	78.2%	77.6%
Overlap match	**Recall**	**Precision**	**F_1**
CRF	89.5%	89.3%	89.4%
CRF w/Backend	89.9%	89.4%	89.7%
Perceptron	88.8%	80.1%	84.2%
Perceptron w/Backend	90.9%	82.2%	86.3%
Frame-Segment	92.8%	90.1%	91.4%
Frame-Segment w/Backend	92.6%	94.1%	93.4%
Instruction follow through		**Success rate**	
CRF		80.1%(**)	
CRF w/Backend		81.0%(*)	
Perceptron		73.9%	
Perceptron w/Backend		76.8%	
Frame-Segment		82.5%(*) (**)	
Frame-Segment w/Backend		88.1%	
Heuristic		39.5%	

4.2 Results

The overall results are listed in Table 3. "Exact match" means that, to be judged a correct segment, both boundaries of the segment must match those in the annotation. "Overlap match" is a lenient measure that considers segmentation or labeling to be correct if it overlaps with any of the annotated labels. "CRF" is a straightforward application of the conditional random field method, "CRF w/Backend" discards impossible paths when running the Viterbi algorithm. "Perceptron" is the same undirected model as the CRF model but is trained with the averaged perceptron instead of the maximum log-likelihood as in the CRF model. "Frame-Segment w/Backend" is the algorithm described in the previous section. "Frame-Segment" is the version of Frame-Segment that does not discard paths that are impossible in context (i.e., paths that refer to objects not in the scene). "Heuristic" is a simple rule-based method defined by Shimizu [Shimizu, 2006]. We also conducted the experiment using the model developed by Cohen and Sarawagi [Cohen and Sarawagi, 2004], but the performance was quite low due to the lack of features that span a segment, unlike the named entity recognition task.

"Instruction follow through" measures the success rate of reaching the destination; it is the most important measure of performance in this domain. If we increase the number of features around the segment boundaries, F_1 may increase; however, this is not the objective of the application. Surprisingly, a paired samples T-test showed that the differences between CRF(*) and frame-segment (*) and between CRF with backend integration (**) and frame-segment (**) are statistically not significant, although the difference between CRF and CRF with Backend is statistically significant. The differences between the other pairs are statistically significant.

The large gain obtained by integrating the backend with the frame-segment model suggests that the frame-segment model

Table 4: Results: Frame-Segment Model w/ Backend on Various Train-Test Partition

Exact Match	Recall	Precision	F_1
Instruction Giver Partition	70.7%	72.3%	71.4%
Start Position Partition	67.9%	69.5%	68.7%
Mixed Partition	77.0%	78.2%	77.6%
Overlap Match	Recall	Precision	F_1
Instruction Giver Partition	90.7%	92.7%	91.7%
Start Position Partition	87.0%	89.0%	88.0%
Mixed Partition	92.6%	94.1%	93.4%
Instruction Follow Through			Success rate
Instruction Giver Partition			86.3%
Start Position Partition			80.0%
Mixed Partition			88.1%

has a better 2nd best or 3rd best action frame in the queue than the CRF model. We attribute this to factorization of the slot fillers. Our model allows swapping of slot fillers when the frame is found to be incompatible with the environment, whereas the linear-chain CRF model must replace the whole action frame.

To determine the generalizability of our results, we tested our frame-segment model (with the backend) for various partitions of the route instructions. First, we partitioned the instructions so that no person appeared in two different partitions. This is called "Instruction Giver Partition." This partitioning reduced the success rate by about 2% from that of the "Mixed Partition" version described in the previous sections. We then partitioned the instructions on the basis of the building. To test how training on one area of the building generalizes to the other areas, we partitioned the route instructions on the basis of the starting position when the demonstration was shown to an instruction giver. The building was partitioned into four separate areas, and the instructions starting from the four areas were placed in four corresponding partitions. Naturally, no route was shared between partitions. Again, we applied four-fold cross validation to this partition. This reduced the performance by about 8 to 80%. The results listed in Table 4 demonstrate that the system is quite robust and that its performance does not suffer much when tested in unseen environments.

5 Conclusion

We have created an end-to-end natural language understanding system for route instruction following. We started by analyzing the errors generated by a linear-chain CRF model and developed the frame-segment model, which combines the Markov and semi-Markov model. By extracting a frame-like structure, we have shown that the frame-segment model obtains a higher labeling accuracy and a much better success rate than the linear-chain CRF.

References

[Bartlett *et al.*, 2004] P. Bartlett, M. Collins, B. Taskar, and D. McAllester. Exponentiated gradient algorithms for large-margin structured classification,. In *Advances in Neural Information Processing Systems 16*, 2004.

[Bugmann *et al.*, 2004] G. Bugmann, E. Klein, S. Lauria, , and T. Kyriacou. Corpus-based robotics : A route instruction example. In *Proc. of Intelligent Autonomous System, 96?E03*, 2004.

[Cohen and Sarawagi, 2004] W. Cohen and S. Sarawagi. Exploiting dictionaries in named entity extraction: Combining semi-markov extraction processes and data integration methods. In *Proc. of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2004.

[Collins, 2002] M. Collins. Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In *Proc. of Empirical Methods in Natural Language Processing (EMNLP)*, 2002.

[Lafferty *et al.*, 2001] J. Lafferty, A. McCallum, and F. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proc. of the 18th International Conference on Machine Learning*, 2001.

[MacMahon and Stankiewicz, 2006] Matt MacMahon and Brian Stankiewicz. Human and automated indoor route instruction following. In *Proc. of the 28th Annual Conf. of the Cognitive Science Society*, Vancouver, BC, July 2006.

[Ramshaw and Marcus, 1995] L. Ramshaw and M. Marcus. Text chunking using transformation-based learning. In *Proc. of Third Workshop on Very Large Corpora. ACL*, 1995.

[Roy, 2005] D. Roy. Semiotic schemas: a framework for grounding language in action and perception. In *Artificial Intelligence 167(1?E):170?E05*, 2005.

[Sang and Buchholz, 2000] E. F. Tjong Kim Sang and S. Buchholz. Introduction to the conll-2000 shared task: Chunking. In *Proc. of the Fourth Conf. on Computational Natural Language Learning (CoNLL)*, 2000.

[Sha and Pereira, 2003] F. Sha and F. Pereira. Shallow parsing with conditional random fields. In *Proc. of the Human Language Technology Conf. (HLT)*, 2003.

[Shimizu, 2006] N. Shimizu. Semantic discourse segmentation and labeling for route instructions. In *Proc. of the COLING/ACL 2006 Student Research Workshop*, pages 31–36, Sydney, Australia, July 2006. Association for Computational Linguistics.

[Skubic *et al.*, 2004] M. Skubic, D. Perzanowski, S. Blisard, A. Schultz, W. Adams, W. Bugajska, and D. Brock. Spatial language for human-robot dialogs. In *IEEE Transactions on Systems, Man & Cybernetics ?EPart C 34(2):154?E67*, 2004.

[Winograd, 1972] T. Winograd. *Understanding Natural Language*. Academic Press, 1972.