The 2010 IEEE/RSJ International Conference on
Intelligent Robots and Systems
October 18-22, 2010, Taipei, Taiwan

A Robust Sketch Interface for Natural Robot Control

Danelle Shah, Joseph Schneider, Mark Campbell

Abstract— **A fully probabilistic command interface for controlling robots using multi-stroke sketch commands is presented. Drawing from prior work in handwriting recognition, sketches are modeled as a variable duration hidden Markov model, where the distributions on the states and transitions are learned from training data. A forward search algorithm on the gesture, stroke, and stroke transition observations is used to find the most likely sketch, which is displayed to the user for confirmation. In cases where the most likely sketch is incorrect, the user can reject it, prompting the next most likely sketch to be displayed. Upon confirmation from the user, the robot executes the desired behaviors. A prototype sketch interface was implemented using a pen tablet; two sets of search-and-identify experiments were conducted using a single robot in an indoor environment to test the usability of the proposed framework. Even novice users were able to successfully complete the missions, including those on whom the algorithm was not trained. User surveys indicate that operators generally found the interface to be natural and easy to use.**

I. Introduction

One of the challenges of making robots ubiquitous is the need to develop intelligent interfaces for more natural human-robot interaction. Especially in search-and-rescue and military applications, it is increasingly necessary that humans be able to communicate naturally and efficiently with teams of collaborating agents, be they humans, robots, or both. Specifically, the command of teams consisting of both humans *and* robots should require a single communication strategy, so the operator need not be concerned with the type or capacity of each individual agent. This motivates the development of an interface that allows humans to communicate with robots in the same flexible and natural way as they would communicate with other humans.

In this paper an architecture for controlling robots using a robust sketch interface is proposed, moving away from the more traditional keyboard and mouse interfaces. A prototype system is shown in Figure 1, in which an operator uses a pen tablet to control a robot remotely by sketching commands, which the robot must interpret and execute correctly. This type of interface could be useful in many robot-assisted operations due to its flexible nature. The sketch recognition algorithm is generalizable, in that it *learns* how gestures are drawn, rather than pre-defining the gestures' structures.

While there has been recent work in commanding robots using hand-drawn sketches, the focus primarily been on qualitative mapping [1], [2] and navigation problems [3], [4], where sketch recognition is hard-coded or nearly-trivial. For example in [3], an object is defined as a closed polygon if the Euclidian distance between the starting and end points fall within an empirically set threshold, whereas a path is interpreted as any line segment that is not recognized as a closed polygon and has a total length greater than a set threshold. The authors do not address the effect of incorrect interpretations of the sketches resulting from variations in sketching styles.

Drawing from prior work in handwriting recognition [5], [6], [7], the proposed sketch interface provides several advantages over the previous approaches. First, by training a classifier, rigid *a priori* assumptions about how gestures should be drawn, such as defining an "X" as two intersecting lines of a particular length, are largely avoided. The sketch model can be learned on many people, providing a more robust interface for a wide range of users; it can also be learned on a single user to increase recognition accuracy if the intended operator is known. This helps to shift the burden of correct sketch recognition from the user to the machine, allowing the user to sketch more naturally. Second, a probabilistic framework enables an intelligent search of the space of possible sketch interpretations, the size of which grows exponentially with the number of strokes. Additionally, if the 'best' solution (i.e. the sketch interpretation with the highest likelihood) is rejected by the user, the interface proposes the 'next best' solution. This is both more convenient to the user than re-drawing the sketch from scratch, and also useful for updating the model on-line. Lastly, by framing the sketch recognition problem as a variable duration (semi-Markov) Hidden Markov Model (HMM), the proposed method supports flexible and multi-stroke gestures, and can be extended to incorporate multi-modal communication such as verbal cues.

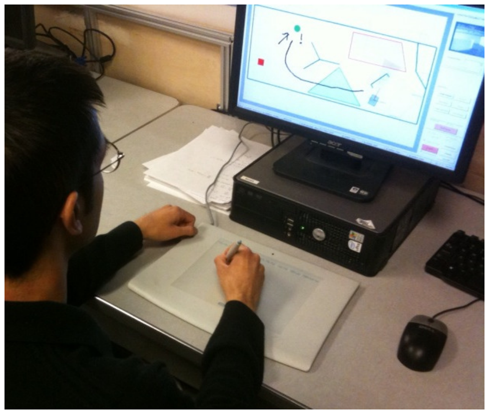


Fig. 1. An operator using the sketch interface to control a robot.

Department of Mechanical Engineering, Cornell University, Ithaca, NY {dcs45,jrs465,mc288}@cornell.edu

II. Probabilistic Sketch Definition

The definition of a sketch proposed in this paper draws from prior work in handwriting recognition by implementing a variation of the variable duration hidden Markov model (VDHMM, sometimes also referred to as hidden semi-Markov model). VDHMMs have been studied and successfully implemented for recognizing handwritten text [8], [9]. Formally, a sketch is composed of N_G gestures $G = [g_1, \ldots, g_{N_G}]$ where the set of possible gesture types is finite and known, $g_k \in \bar{G} = \{\bar{g}_1, \ldots, \bar{g}_M\}$, but the total number of gestures drawn N_G is unknown. Each gesture is made up of d_k strokes, where $d_k \in [1, \infty)$ is the *duration*. In this paper, it is assumed that each stroke belongs to a single gesture and that strokes are not interspersed, i.e. users do not return to a gesture after starting a new one. Given a gesture g_k, the set of d_k strokes comprising the gesture is given as $S_k = \left[s_k^1, \ldots, s_k^{d_k}\right]$, where $s_k^j \in \bar{S} = \{\bar{s}_1, \ldots, \bar{s}_M\}$ is the jth stroke in the kth gesture and $\bar{S}$ is the set of possible stroke types. Each stroke type corresponds to a particular gesture type (i.e. 'square-like' or 'circle-like' strokes); this is similar to existing methods that categorize a stroke as being a particular primitive, such as a line or an arc [10]. The key difference here is that the primitives are not predefined. Rather, the stroke model will be *learned* such that the most discriminating features are used for inference (e.g. it will automatically determine what makes all `square` strokes inherently different than all `triangle` strokes).

An *interstroke* is the transition between two strokes, and defines whether a stroke belongs to the same gesture as the previous stroke (i.e. a self-transition on the gesture) or it is the start of a new gesture. Figure 2 illustrates an example of a sketch of $N = 5$ strokes and $N_G = 2$ gestures modeled as a VDHMM, where white nodes represent hidden (unobserved) variables, shaded nodes are observed variables, and arrows represent conditional dependencies. In this example, the first gesture g_1 has a duration of $d_1 = 3$ strokes, and the second gesture g_2 has a duration of $d_2 = 2$ strokes. Interstroke nodes i_1^1, i_1^2, and i_2^1 represent self-transitions on the gestures, while node $i_{1:2}$ represents a transition from gesture g_1 to gesture g_2. The goal is to determine the specific gestures sketched, given the observations O (from pixel-level data) by maximizing the observation likelihood:

$$p\,(\text{sketch}|O) = p\,(G, S, I|O) \qquad (1)$$

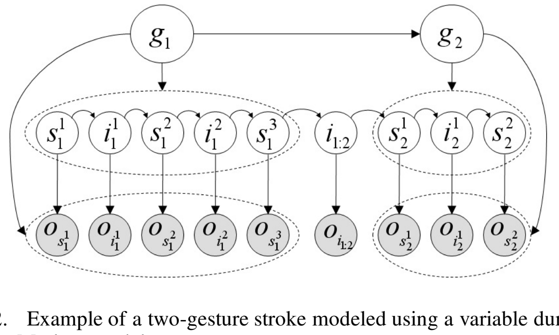


Fig. 2. Example of a two-gesture stroke modeled using a variable duration hidden Markov model.

Defining O_G, O_S and O_I as sets of observations on the gestures, strokes, and interstrokes respectively, one can factorize Equation 1:

$$p\,(G, S, I|O) = p\,(G|I, S, O_G)\, p\,(I|S, O_I)\, p\,(S|O_S) \qquad (2)$$

Specifically, the observations are features extracted from the pixel data, to be described in Section III-A. The terms on the right-hand side of Equation 2 are approximated as:

$$p\,(I|S, O_I) = \prod_{k=1}^{N-1} p\,(i_k|s_{k-1}, o_{i_k})$$

$$p\,(S|O_S) = \prod_{k=1}^{N} p\,(s_k|s_{k-1}, i_{k-1}, o_{s_k})$$

$$p\,(G|I, S, O_G) = \prod_{k=1}^{N_G} p\,(g_k|S_k, I_k, o_{g_k}, g_{k-1})$$

where lowercase letters indicate single nodes and uppercase letters indicate (possibly) more than one node. (The subscript notation above differs slightly from those shown in Figure 2, because it is not known how strokes and interstrokes compose separate gestures.) This generic framework allows for the inclusion of a variety of scripts and icons without the need to hard-code their attributes. This flexibility is key in enabling an arbitrary selection of intuitive and natural gestures for the basic units in the lexicon of robot commands.

III. Learning the VDHMM

In this work, several simplifying assumptions are made before learning the HMM. First, the probability distribution of transitioning from one gesture type to another is assumed to be uniform. This assumption is not necessary (and may be inappropriate in some applications), but it provides a reasonable balance between complexity and accuracy for the application presented here. Second, gesture 'mistakes' or otherwise unknown gestures are not considered, although this could also be easily incorporated into the framework by including an $(M + 1)$th 'other' gesture type. Lastly, gesture g_k must be of the same type as its associated strokes S_k (i.e. a *square* can only be drawn with *square-like* strokes).

Noting that the distributions $p\,(i_k|s_{k-1}, o_{i_k})$, $p\,(s_k|s_{k-1}, i_{k-1}, o_{s_k})$ and $p\,(g_k|S_k, I_k, o_{g_k}, g_{k-1})$ are partly known *a priori* (for example, stroke s_k must be of the same type as s_{k-1} if i_{k-1} is a gesture self-transition), the remaining unknown parts of the HMM require only the following distributions be defined: $p\,(i_k|o_{i_k})$, $p\,(s_k|o_{s_k})$, and $p\,(g_k|o_{g_k})$. These can each be thought of as multinomial classification problems, the distributions of which are learned using Sparse Multinomial Logistic Regression (SMLR)[11]. SMLR is a true multiclass formulation which learns a weight vector w such that the likelihood of label ℓ for a set of features x is given by:

$$\mathcal{L}_\ell\,(y) = p\,(y = \ell|x; w) = \frac{\exp\left(w^{(\ell)T}x\right)}{\sum_{j=1}^{m} \exp\left(w^{(j)T}x\right)} \qquad (3)$$

where y corresponds to g_k, s_k, or i_k and x corresponds to observations o_{g_k}, o_{s_k} or o_{i_k}, respectively. As the name implies, SMLR produces a *sparse* weight vector by using a Laplacian prior on the weights. Although calculating the maximum a posteriori multinomial regression with a Laplacian prior scales unfavorably with the number of bases (which may be very large), the component-wise update equation has a monotonically increasing closed form solution[11]. Solving component-wise SMLR results in a computation cost and storage requirement linear in the number of bases. As with relevance vector machines (RVMs), SMLR uses Bayesian inference to provide probabilistic classification and can incorporate arbitrary bases, including non-Mercer kernels. SMLR, however, converges to a unique maximum and is not at risk of converging to local minima as RVMs are [12].

A. *Multi-Stroke Data Set and Features*

Data was collected for ten gesture types, shown in Figure 3, using a pen tablet from thirteen users. Users were told to draw the gestures "naturally" and were given minimal instructions regarding how the gestures should look.

For each gesture drawn, 99 features were extracted from the pixel data (corresponding to O_G). Some of these features were adopted from a portion of the g-48 set presented in [13], which was developed to be generally well-suited for identifying multiple-stroke gestures. An additional set of features was developed by calculating the direction of the stroke at each pixel and discretizing into bins. Completing the set of observations were features corresponding to initial orientation angles, the amount of time spent drawing each gesture, the total number of strokes, and cw/ccw orientation.

The gesture classifier, corresponding to $\mathcal{L}_{\bar{g}_\ell}(g_k) = p(g_k = \bar{g}_\ell | o_{g_k})$, was learned using SMLR on 880 gestures (88 of each type $\bar{g}_\ell$). Figure 4 presents the average probabilities calculated by the learned regression model using 550 test gestures (55 of each type). The average probability of the correct gesture types is 96.9%, which is comparable to other similar recognition algorithms.

The stroke regression model, corresponding to $\mathcal{L}_{\bar{s}_\ell}(s_k) = p(s_k = \bar{s}_\ell | o_{s_k})$, was learned on single strokes using 890 strokes (89 of each type $\bar{s}_\ell$) and tested on 820 strokes (82 of each type). The extracted stroke features (corresponding to O_S) were the same as those used for gestures (with the exception of number of strokes).

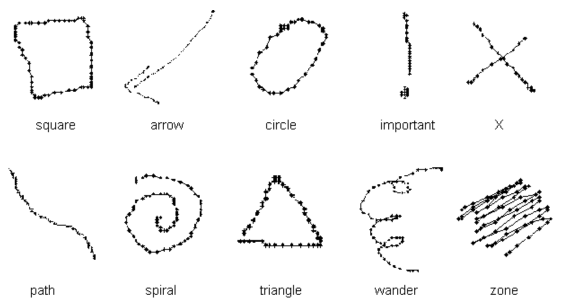
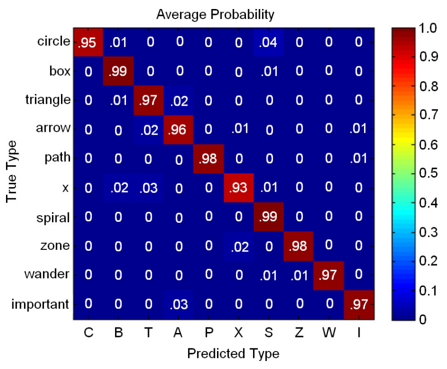


Fig. 3. The ten gestures used in the proposed sketch interface.

Figure 5 shows the average probabilities calculated by the learned regression model for stroke classes. Notice that the X-stroke has non-zero likelihoods associated with `triangles` and `arrows`. This is because `triangles` and `arrows` are often also drawn with angled straight-line strokes. Similarly, the `important`-stroke had non-zero likelihoods associated with the `box`-stroke, which is also often drawn using vertical straight-lines. Despite these expected confusions, the average likelihood calculated for the correct label is still 81.2%. Considering that half of the 10 stroke classes are likely to contain straight line strokes, this level of accuracy is still very good.

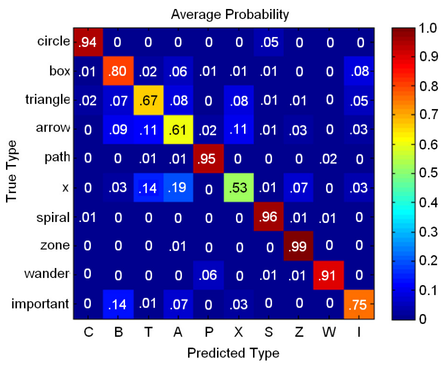
The interstroke model $\mathcal{L}_{\bar{i}_\ell}(i_k) = p(i_k = \bar{i}_\ell | o_{i_k})$ assumes that $\bar{I}$ consists of seven classes : `box-ST`, `triangle-ST`, `arrow-ST`, `X-ST`, `zone-ST`, `important-ST` and `new gesture transition` (where 'ST' indicates a gesture self-transition). Four gesture types (`circle`, `path`, `wander` and `spiral`) were never drawn with more than one stroke in the training set, so these stroke transition likelihoods were set to zero. Interstrokes corresponding to `new gesture transitions` did not specify which gestures were being transitioned to/from, but this could also be learned if necessary. Nine interstroke features (O_I) were extracted from the information from consecutive strokes, including relative positions, sizes, and orientations, as well as temporal information.

The interstroke regression model was learned on stroke transitions using 560 interstrokes (80 of each type $\bar{i}_\ell$) and



	C	B	T	A	P	X	S	Z	W	I
circle	.95	.01	0	0	0	0	.04	0	0	0
box	0	.99	0	0	0	0	.01	0	0	0
triangle	0	.01	.97	.02	0	0	0	0	0	0
arrow	0	0	.02	.96	0	.01	0	0	0	.01
path	0	0	0	0	.98	0	0	0	0	.01
x	0	.02	.03	0	0	.93	.01	0	0	0
spiral	0	0	0	0	0	0	.99	0	0	0
zone	0	0	0	0	0	.02	0	.98	0	0
wander	0	0	0	0	0	0	.01	.01	.97	0
important	0	0	0	.03	0	0	0	0	0	.97

Fig. 4. Average calculated gesture likelihoods.



	C	B	T	A	P	X	S	Z	W	I
circle	.94	0	0	0	0	0	.05	0	0	0
box	.01	.80	.02	.06	.01	.01	.01	0	0	.08
triangle	.02	.07	.67	.08	0	.08	.01	.01	0	.05
arrow	0	.09	.11	.61	.02	.11	.01	.03	0	.03
path	0	0	.01	.01	.95	0	0	0	.02	0
x	0	.03	.14	.19	0	.53	.01	.07	0	.03
spiral	.01	0	0	0	0	0	.96	.01	.01	0
zone	0	0	0	.01	0	0	0	.99	0	0
wander	0	0	0	0	.06	0	.01	.01	.91	0
important	0	.14	.01	.07	0	.03	0	0	0	.75

Fig. 5. Average calculated stroke likelihoods.

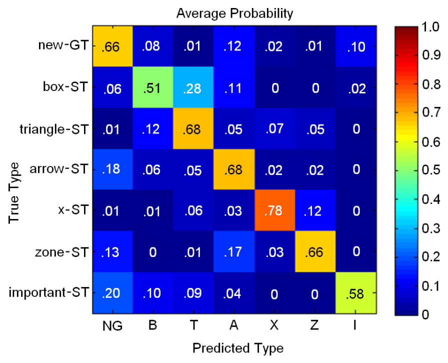


Fig. 6. Average calculated interstroke likelihoods.

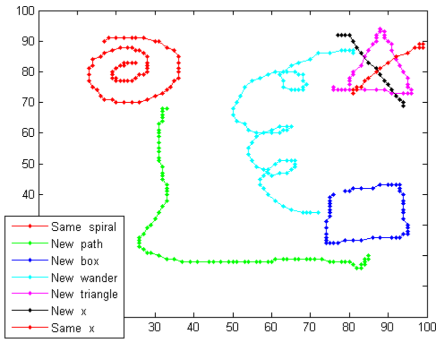


Fig. 7. An example sketch with six gestures and seven strokes.

tested on 322 interstrokes (46 of each type). Figure 6 shows the average probabilities calculated by the learned regression model. The average likelihood of correct classification likelihood was 65.1%. This lower recognition rate reflects the fact that the classification problem is very challenging; performing classification on interstrokes is analogous to determining where spaces belong in a line of text by looking only at two adjacent letters. Although the feature set is not particularly rich, this extra information can be helpful when combined with stroke and gesture level features.

B. Recognizing Sketched Commands Using VDHMM

An example sketch is shown in Figure 7. Using the learned likelihood models from the previous section, a forward tree-search algorithm is implemented to find the most likely sketch by searching for the sequence of strokes, interstrokes and gestures that maximizes Equation 1. Starting with the first stroke, a node is created for each $s_1 = \bar{s}_\ell$ where $\ell \in \{1, ..., M\}$. The node n which maximizes the likelihood $\mathcal{L}_n = \mathcal{L}_{\bar{s}_\ell}(s_1)$ from Equation 3 is expanded and a new node is created for each $i_1 = \bar{i}_\ell$. If i_1 at node n' (with parent node n) corresponds to a self-transition (i.e. stroke s_2 will be part of the same gesture as s_1), the node likelihood is $\mathcal{L}_{n'} = \mathcal{L}_n \cdot \mathcal{L}_{\bar{i}_\ell}(i_1)$. If i_1 at node n' corresponds to a gesture transition (i.e. stroke s_2 is the start of a new gesture), the node likelihood is $\mathcal{L}_{n'} = \mathcal{L}_n \cdot \mathcal{L}_{\bar{i}_\ell}(i_1) \cdot \mathcal{L}_{\bar{g}_\ell}(g_1)$. This process is repeated, always starting from the node with the highest likelihood $\mathcal{L}$ until the highest likelihood node corresponds to the last stroke of the sketch.

The number of nodes explored in the tree search is, worst-case, $M(M+1)^{N-1}$ nodes, which can be too time consuming to allow real-time evaluation of stroke sequences. Unfortunately, as the forward search algorithm progresses through the stroke sequence, the aggregate likelihood of the sketch decreases (as numbers ≤ 1 are multiplied, or numbers ≤ 0 added in the case of log-likelihoods). This has the undesirable effect of encouraging the algorithm to perform breadth-first-search (this is worst-case for trees with constant depth). Therefore, a heuristic is implemented to 'penalize' expanding nodes closer to the top of the tree:

$$\mathcal{L}'_n = \mathcal{L}_n \cdot \left(\frac{N_n}{N}\right)^\alpha \quad (4)$$

where N_n is the stroke number of node n, N is the total number of strokes, and α is a number between 0 and 1. (Note that $\mathcal{L}$ here is a log-likelihood.) This heuristic encourages a more depth-first-search behavior, but sacrifices the guarantee of optimality.

Two metrics are used to evaluate the efficiency and effectiveness of the sketch recognition algorithm: (1) the total number of misclassifications, and (2) the number of nodes explored in the tree search. A misclassification occurs either when a stroke is labeled as the incorrect type, or if an interstroke is incorrectly identified as a new gesture transition when it is actually a self-transition (or vise-versa). The number of nodes explored in the tree search indicates computational complexity, and is a key concern for a system that must run in real-time. Specifically, full sketch recognition should be performed at about the same rate as the human can draw (under one second is desirable).

The sketch recognition algorithm was tested on 77 sketches, with the number of gestures ranging from 3–10, and each gesture drawn using 1–4 strokes. Average accuracy (Fig. 8) and percent of tree searched (Fig. 9) show that while adding a penalty term ($\alpha \in [0, 1]$ in Equation 4) decreases the sketch recognition accuracy slightly, the computational cost decreases significantly. Figs 8 and 9 suggest that it may be reasonable to use a small penalty term ($\alpha < 0.4$) to help avoid breadth-first-search behavior while sacrificing little accuracy.

With no penalty ($\alpha = 0$), the sequence reconstruction algorithm yielded an average accuracy of 93.2%. Of the 77 sketches tested, only 59% were identified completely correctly on the first try. To account for these errors, users were able to confirm or reject the recognized sketch. While this may slightly increase operator workload, hypothesis generation and confirmation has been used successfully to avoid problems caused by imperfect recognition [14].

IV. Experiments

Two sets of experiments were conducted to study the usability of the proposed sketch interface. Operators controlled a single robot in a search-and-identify mission using two different control architectures: a point-and-click computer mouse interface using buttons and menus (referred to as "Waypoint mode"), and a sketch interface using a pen tablet

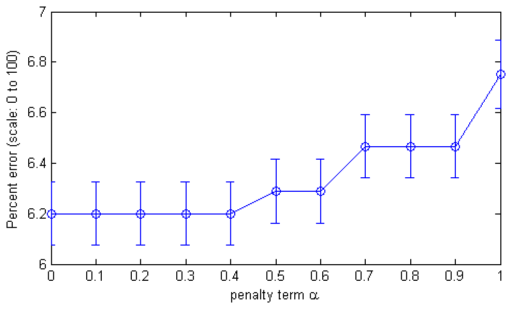


Fig. 8. Average tree search accuracy using several values of α.

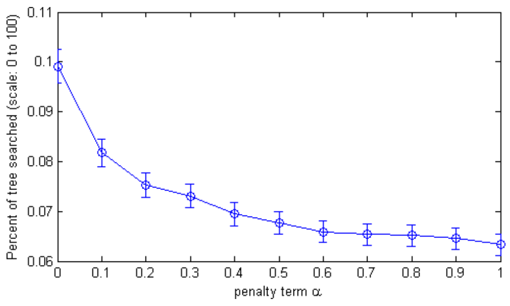


Fig. 9. Average tree search efficiency using several values of α.

("Sketch mode"). While the modes of user input were different, both architectures included the same functionalities. The mission objectives were to explore all areas displayed in green on the map, avoid all areas displayed in red on the map, take a picture of all green objects (large trash cans), search for three points of interest (orange traffic cones) and add their locations to the map. Although users controlled only a single robot in the experiments presented here, the system has also demonstrated effectiveness in multi-robot control.

The robot and experimental environment are shown in Figure 10. The robot's pose is estimated in real time using noisy odometry readings and GPS-like measurements obtained from an overhead motion capture system. The robot is also equipped with a laser range finder and a camera. Objects in the environment detected by the laser are displayed as wire polygons in the map. The robot is capable of basic obstacle avoidance; if an obstacle is seen with the laser, the object is detected and avoided using the robot's low-level path planning component. Additionally, if the robot is traveling to a waypoint location, the trajectory is planned to avoid all detected obstacles (so long as the waypoint is not placed inside an obstacle, in which case the robot waits for the user to send a new command).

Figure 11 shows screen shots of the user interface, displaying an overhead view of the environment and the robot being controlled. For experiments conducted using Sketch mode, nine gestures were available to the user from Figure 3 (the `wander` gesture was not used). Users were instructed to sketch "naturally" as if they were trying to communicate with another human, but were otherwise given no instructions as to how the gestures should be drawn. Sketches could be edited by deleting the last stroke or the entire sketch.

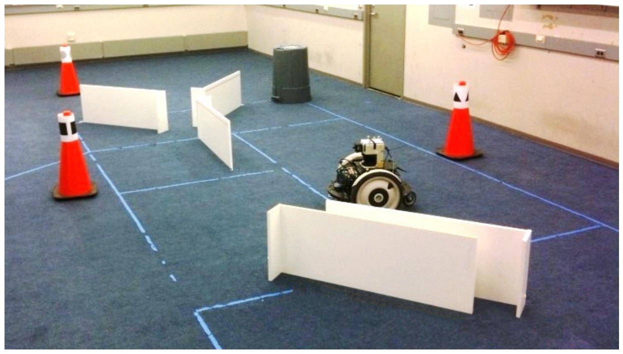


Fig. 10. Field used for sketch experiments with robot, obstacles, and points of interest (POIs).

The gesture commands were to be used as follows. `Circles` drawn around a robot selects the robot for future control. An `X` sets a single waypoint to which the robot must go, the specific path to be determined by the robot's low-level path planner. An `arrow` commands the robot to travel to the base of the arrow and turn to point in the direction indicated by the arrow head. `Paths` define a desired robot trajectory by setting a series of waypoints along the sketched path. A `zone` indicates a specific area that should be explored, and the robot visits randomly sampled points inside the specified zone. An `important` instructs the robot to take a picture of its current camera view. Lastly, `squares`, `triangles` and `spirals` represent the three types of points of interest (POIs) the robot can encounter. When one of these three gestures is drawn, a point is added to the map at the gesture centroid location.

When a user finishes drawing a sketch, he or she presses the 'Send Sketch' button, triggering the recognition algorithm to search for the most likely sketch. Once finished (typically < 1 sec), the recognized sketch is displayed on the screen for the user to inspect. If the sketch is correctly recognized, the user presses the 'Correct' button. Otherwise, pressing the 'Incorrect' button signals to display the sketch with the next-highest likelihood. Figure 11 illustrates the sequence of events for an example sketch. In the top panel, the user inputs a sketch consisting of `circle`, `path`, `arrow`, and `important` gestures. In the second panel, the robot displays the most likely sketch and waits for the user to confirm. Finally, the last panel shows the robot beginning to follow the instructed path after the sketch has been confirmed. The corresponding video submission illustrates the interface in use.

Four users have performed a total of ten missions using Waypoint mode, and an additional three users performed one mission each using both Waypoint and Sketch modes. On average, users were able to complete the missions in approximately the same amount of time (within 0.5 standard deviation) using either interface, around 8 minutes per mission. However, users of Sketch mode used on average 80 mouse clicks to perform each mission, while users of Waypoint mode used an average of 109 clicks (a 37% increase). These clicks were divided into distinct interactions; some

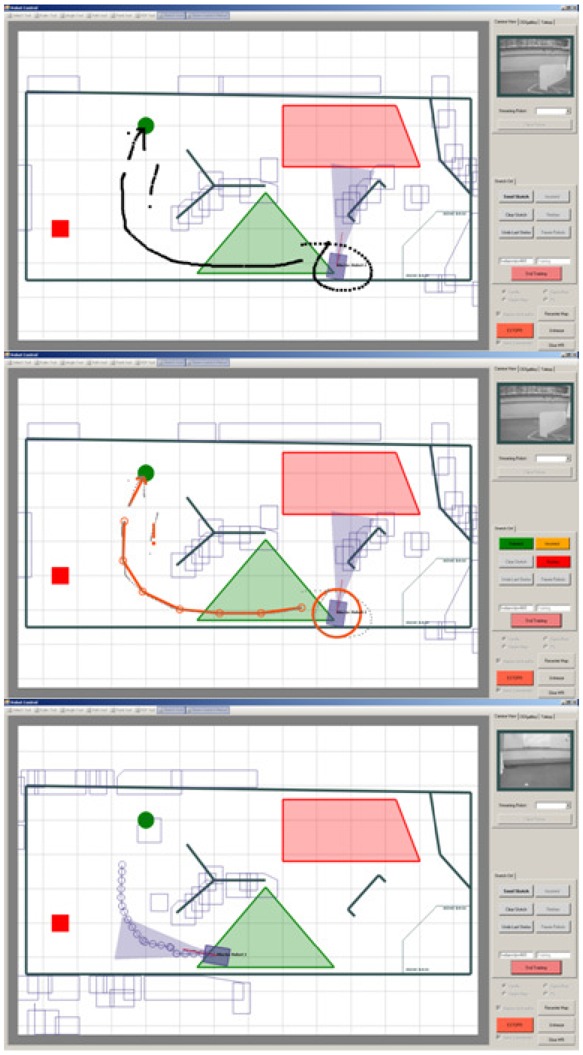


Fig. 11. Sequence of events for an example sketch with four gestures.

interactions required several clicks, for example operators using Waypoint mode used an average of 2.2 waypoints per path command. The number of interactions was similar for missions using both Sketch and Waypoint modes (23 and 20 respectively, within 0.5 standard deviation). Thus, Sketch mode uses substantially fewer mouse clicks than Waypoint mode (5.8 and 3.5, respectively, a difference of about 1.5 standard deviations).

Operators using Sketch mode could use three different gestures for robot navigation – `X`, `path`, and `zone`. Of these, `paths` were used most often (58%), followed by `zones` (29%) and `Xs` (13%). Operators preferred sending short, simple commands, rather than long sequences of gestures; 96% of sketches were drawn using only 1–2 strokes. As a result, sketches were correctly recognized 80% of the time on the first try, and another 10% on the second try.

User surveys conducted following the experiments suggest that novice users (unfamiliar with robotic systems) were quite receptive to the gesture interface, even though they were more exposed to the common interface of a keyboard and mouse. One subject indicated, “It could also probably be easy to control multiple robots with this interface.” Another subject noted, “It relies more on the robot following your commands, rather than you do [sic] everything.” This is precisely the impression that is lacking in many current control interfaces, hindering the widespread acceptance of robotic assisted tasks.

V. Conclusion

In this paper, a flexible sketch interface to accommodate natural communication between a robot and human operator was introduced. By modeling sketched commands using a variable duration hidden Markov model and learning the distributions from training data, multi-stroke gesture sketches are recognized at 93.2% accuracy without requiring the user to sketch in any particular style. This shifts the burden of recognition from the operator to machine, allowing users to focus their concentration on the task at hand. The probabilistic framework is generalizable, and can therefore be implemented using any gestures desired. The sketch interface is demonstrated in an indoor search-and-identify mission, where users controlled a single robot using sketched commands.

References

[1] Bailey, C., “A Sketch Interface for Understanding Hand-Drawn Route Maps”, MS Thesis, Columbia, MO, 2003.

[2] Parekh, G., “Scene Matching Between a Quatitative Map and a Qualitative Hand Drawn Sketch,” Ph.D. dissertation, University of Missouri-Columbia, Publication No.

[3] Skubic, M., Bailey C., and Chronis, G., “A Sketch Interface for Mobile Robots,” in Proc. of the IEEE 2003 Conf. on SMC, Washington, D.C., Oct., pp. 918–924, 2003.

[4] Skubic, M., Anderson, D., Blisard, S., “Using a hand-drawn sketch to control a team of robots,” Autonomous Robots, vol. 22, no. 4, pp. 399–410, 2007.

[5] Shilman, M., Viola, P., Chellapilla, K., ”Recognition and Grouping of Handwritten Text in Diagrams and Equations,” Frontiers in Handwriting Recognition, International Workshop on, pp. 569-574, 2004.

[6] Sezgin, T. M. and Davis, R. 2005. “HMM-based efficient sketch recognition.” In Proceedings of the 10th international Conference on intelligent User interfaces, January 10 - 13, 2005.

[7] Lee C., and Xu, Y., “Online, interactive learning of gestures for human/robot interfaces,” IEEE International Conference on Robotics and Automation (ICRA1996), pp. 2982–2987, 1996.

[8] Cho, W., Lee S., and Kim, J., “Modeling and recognition of cursive words with hidden Markov models,” Pattern Recognition, vol. 28, no. 12, pp. 1941–1953, December 1995.

[9] Kundu, A., Hines, T., Huyck, B., Phillips, J., and Van Guilder, L., “Arabic Handwriting Recognition Using Variable Duration HMM,” Proceedings of the Ninth International Conference on Document Analysis and Recognition, vo. 2, pp. 644–648, 2007.

[10] Alvarado, C., Davis, R., “SketchREAD: a multi-domain sketch recognition engine,” Proceedings of the 17th annual ACM symposium on User interface software and technology, pp. 23–32, 2004.

[11] Krishnapuram, B., Figueiredo, M., Carin, L., and Hartemink, A., “Sparse Multinomial Logistic Regression: Fast Algorithms and Generalization Bounds.” IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI), vol. 27, pp. 957968, 2005.

[12] Tipping, M., “Sparse Bayesian Learning and the Relevance Vector Machine,” The Journal of Machine Learning Research, vol. 1, pp. 211–244, 2001.

[13] Willems, D., Niels, R., Van Gerven, M., Vuurpijl, L., “Iconic and multi-stroke gesture recognition,” Pattern Recognition, vol. 42, num. 12, pp. 3303–3312, 2009.

[14] Kurnia, R., Hossain, M., Nakamura, A., Kuno, Y., “Generation of efficient and user-friendly queries for helper robots to detect target objects,” Advanced Robotics, vol. 20, pp. 499–517, 2006.