Drawings as Input for Handheld Game Computers

Mannes Poel, Job Zwiers, Anton Nijholt, Rudy de Jong, and Edward Krooman

University of Twente, Dept. Computer Science,
P.O. Box 217, 7500 AE Enschede, The Netherlands
{mpoel, zwiers, anijholt}@cs.utwente.nl

Abstract. The Nintendo DS$^{\text{TM}}$ is a hand held game computer that includes a small sketch pad as one of it input modalities. We discuss the possibilities for recognition of simple line drawing on this device, with focus of attention on robustness and real-time behavior. The results of our experiments show that with devices that are now becoming available in the consumer market, effective image recognition is possible, provided a clear application domain is selected. In our case, this domain was the usage of simple images as input modality for computer games that are typical for small hand held devices.

1 Introduction

Game user interfaces convert the actions of the user to game actions. In a standard PC setting game players mainly use keyboard and mouse. It does not necessarily mean that there are straightforward mappings from the usual keyboard and mouse actions (pointing, clicking, selecting, dragging and scrolling) to similar actions in the game. On the contrary, there are mappings to menu and button actions, to route and region selection on maps, to tasks and objects and to actor activity. The cursor can obtain context-dependent functionality, allowing for example pointing, grabbing, catching, caressing and slapping. In games the computer may also ask the player to select or draw paths and regions on a map. Drawing interfaces for games are available. For example, several games have been designed where in a multi-user setting a player enters a drawing that best expresses a word or phrase that has been assigned to him and other players have to guess the word. Hence, the interaction is based on a drawing, but the drawing is just mediated to the other players and there is no attempt to make the computer interpret the drawing and make this interpretation part of a game. A recent example of a game computer that allows game developers to include player made drawings in a game is the handheld game computer Nintendo DS. Its touch screen can be used as a sketch pad that allows for simple drawings to be made. Engine-Software [1] company is a small company that investigates

[TM] Nintendo DS is a trademark of Nintendo of America Inc.

[1] Engine-Software: http://www.engine-software.nl

the possibilities of this feature in the development of interactive games, where pen input replaces the more traditional modalities such as buttons and mouse. The type of application (interactive games) results in hard real-time constraints, and requires a fair amount of robustness: game players do not want to wait for image recognition processes, nor do they want to have an editing or correction stage. Since the device has also only limited computational power, the selection of algorithms for recognition of pen input becomes challenging. In Fig. 1 we have displayed the instruction screen of the game, which can be consulted before the game starts, while on the right, during the game the gamer has drawn an object that helps the hero to deal with a particular problem. Here a trampoline is drawn that can be used to jump over an obstacle. We have implemented and

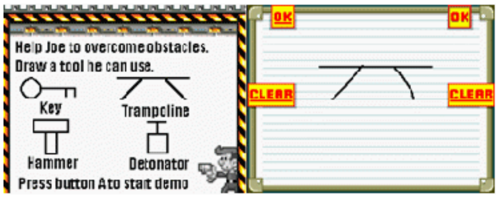


Fig. 1. Instruction screen (left) and sketch pad (right)

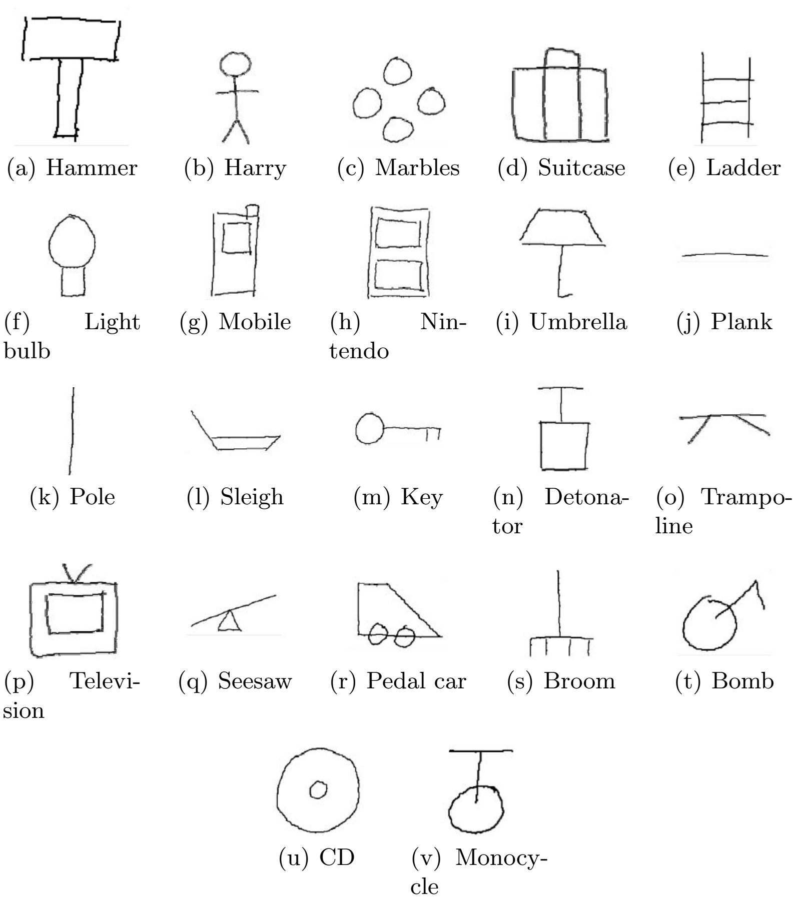
tested a possible setup for recognition of a set of simple line drawings that could play a role in games. Experimental results have been obtained for drawings of a set of 22 different classes of objects, see Fig. 2 for examples. After an (offline) training phase the classification returned correct result in 90 %, whereas the overall recognition rate was over 86 %. We do not compare this approach with respect to playing performance with the approaches using more classical input modalities (keyboard and mouse actions such as typing, clicking, dragging, etc) for supporting game interactions. The focus is on new interaction modalities for interactive games.

In the remainder, we first describe the global structure of the recognition process, consisting of line recognition, feature extraction, and template matching. Thereafter, we discuss the results of testing the performance of our classification process.

2 Global Structure

A basic assumption that is made is that users of the device will produce simple line drawings, according to a predefined set of "objects" that play a role in the game. Since users should be able to learn to draw very easily, it was decided that the drawings are built up from straight lines and circles only. The global image recognition process is built up like a pipeline:

- The first stage tries to simplify curves that are being drawn. Lines and circles that are drawn tend to be imperfect, partly due to difficulties with drawing straight lines on a slippery surface, and partly due to inaccurate drawing. This stage tries to reduce curves to either lines or circles, and tries to combine series of small line segments into larger segments.
- The second stage is concerned with extracting a set of features from lines and circles. Typical features like number of lines and circles, number of horizontal lines, number of intersections etc.
- The final stage tries to classify a drawing based upon features. We used a decision tree approach, with a tree that was created off line, from experimental data.



(a) Hammer (b) Harry (c) Marbles (d) Suitcase (e) Ladder

(f) Light bulb (g) Mobile (h) Nintendo (i) Umbrella (j) Plank

(k) Pole (l) Sleigh (m) Key (n) Detonator (o) Trampoline

(p) Television (q) Seesaw (r) Pedal car (s) Broom (t) Bomb

(u) CD (v) Monocycle

Fig. 2. Some examples of drawings and the corresponding classes

2.1 Line Recognition and Simplification

Image recognition for devices like the Nintendo DS, and the typical application that one has in mind for such devices can be characterized as follows:

- The image quality itself is not very high, due to the limited resolution of digitizing process that is built into the device. Also, the surface of the sketch pad is slippery, which makes it difficult, for instance, to draw perfectly straight lines.
- "Drawing" is used as an *input modality*, used to control a game in real-time, rather than to input complex drawings.

The image recognition process should take this into account. Consequently, many advanced techniques have been rules out in favor of simple, fast and robust recognition techniques. For instance, we have assumed that images consist of line drawings, consisting of simple curves like straight line segment or circles. In our case after some comparisons with other algorithms, we chose the algorithm by Douglas and Peucker [1]. This is a global approach where a curve is subdivided into two parts, where the division point is chosen as the point that deviates most from the straight line from start point to end point. This subdivision process is repeated recursively, until the deviation from a straight line becomes smaller than a predefined threshold. The threshold depends on the distance between start and end point in order to retain small scale features. The algorithms were tested on a set of 385 drawings, from 21 categories. For each drawing, the "correct" set of lines was established manually, based upon human judgment. The results of the three line simplification algorithms have been compared to these correct lines. It turned out that the Douglas and Peucker algorithm was able to classify correctly in 95% of all cases, outperforming the other algorithms

3 Feature Extraction

After line simplification has been applied, the segments and circles are reduced to a small set of features that are suitable for the classification phase, discussed in Sect. 4. In the literature one can find many features used for the recognition of drawings. A short and definitely not complete list of static features is given below.

- Chain code features [2].
- Line features, such as horizontal, vertical and diagonal [3].
- Characteristic points such as vertices, intersections, endpoints [4].
- Loops [4].
- Geometric forms such as circles, triangles, squares [5,6].
- Ratio between height and width [7].

In the next subsections we introduce the selected features, and consider the computational effort to calculate them. The latter is important, since we are focussing on applications that must run in real-time on low end devices.

3.1 Directions of Line Segments and Length Ratios

The direction of line segments turned out to be a sound basis for a number of features. On the one hand side, determining such features is computationally cheap. On the other side, the class of drawings consists mainly of pictures where lines are either horizontally, vertically, or diagonally. The direction of drawing a line was deemed insignificant: for instance, a horizontal line can be drawn from left to right or vice versa, but we would not classify the picture differently in these two cases. So that leaves us basically with four different directions, and simply *counting* the number of segments in each of these four categories resulted in useful features. A slightly different approach is not to count, but rather calculate the ratio of the total (i.e. accumulated) length of all segments in a certain direction to the total accumulated length of all segments.

3.2 Line Crossings and Corners

The number of line crossings and the number of corners have also been used as features. Due to the line simplification preprocessing phase, such features can be calculated in an efficient and robust way, simply by solving linear equations and classifying the intersection points: if the line intersection point is near the end points of both segments, it is a corner point, if the line intersection point is on both segments but not in the neighborhood of these end points, it is an ordinary intersection of the two segments etcetera. This rather simple approach is possible only because we assume drawings to consist of lines. For instance, in [4] intersections are calculated for character recognition in handwriting, and there, intersections have to be determined on the level of relationship between neighboring pixels.

3.3 Detecting Circles

The number of circles is an obvious feature, given the set of drawings that we are interested in. The question is mainly how such circles can be detected in a computationally cheap way. For instance, detection based on the Hough transform, that would be a good approach otherwise, is ruled out on the basis of computational cost. Moreover, it was experimentally observed that users had great difficulty in drawing “nice” circles on the screen of the Nintendo device. Usually, the result could be described best as a “polygon with a large number of edges, with shallow angles between consecutive edges”. This informal description, together with the requirement that end point of the drawing stroke should be within the neighborhood of the start point, resulting in a very simple yet effective detection algorithm for circles. As before the success here depends heavily on the constraints on drawings: pictures should consist of straight line segments and circles only. For example, drawing a shape in the form of the number eight will result in incorrectly detecting a circle shape.

3.4 The Complete Feature Vector

Summarizing, we have used the following set of robust features, cf. Table 1. These features have a high discriminative power and can be computed efficiently.

Table 1. The list of selected features for recognizing the drawings. The value for these features is given for a typical example of the class "Umbrella" and "Key".

Feature	Umbrella	Key
Number of horizontal lines	3	1
Number of vertical lines	1	2
Number of up diagonals	1	0
Number of down diagonals	1	0
Number of vertices	4	1
Number of intersections	1	1
Number of circles	0	1
Ratio horizontal/total length	0.68	0.75
Ratio vertical/total length	0.10	0.25
Ratio up diagonal/total length	0.13	0
Ratio down diagonal/total length	0.09	0
Ratio height/width	0.91	0.40

4 Classifying Drawings

The final phase in the recognition process is the classification of the drawing based on the features discussed in the previous section, Sect. 3. The approach taken is to use machine learning techniques to train a classifier based on a training set of drawings. The requirements for the classifier are that it should run on the low-end device, hence it should take a minimum of processing power, it should be fast! Moreover the classification procedure should be transparent for by humans and have a high performance. Given this requirements an obvious candidate is decision trees, [8,9]. But decision trees will classify every drawing to one of the a priori determined classes and hence it will also assign a class to drawings, for instance a drawing of a car, which are completely out of the domain. In order to determine that a drawing is out of the domain we use template matching for determining if a drawing actually belongs to the assigned class.

In order to construct the decision tree and determine the templates for template matching a set of around 940 drawings was gathered. These drawings where generated by 35 persons, for each class there are approximately 30 example drawings. This set was split in two parts: a test set of 314 drawings (chosen at random) and a training set consisting of the remaining drawings.

4.1 Decision Trees for Classifying Drawings

We use Quinlan's C4.5 algorithm [9] for learning a decision tree from the available training data. It is well known in the theory of decision trees that pruning a decision tree improves the performance on new unseen data. This pruning can be done with several confidence levels, c.f. [9]. A K-fold cross validation experiment was performed on the training set to determine the best confidence level. It turned out that pruning improved the performance but there was no statistical significant difference between the different confidence levels. Hence a confidence level of 0.1 was taken and afterwards a decision tree was learned from the training data using C4.5. For attribute selection the "information gain" criteria was taking, and afterwards the tree was pruned with confidence level 0.1.

The test results can be found in the next section, Sect. 5

4.2 Template Matching for Recognizing Out of Domain Drawings

The next step in the classification procedure is to recognize out of domain drawings. A decision tree assigns to each new drawing one of the a priori defined classes, also when the drawing is completely out the range of allowed drawings. This is not to reject drawings. One solution is to define a "reject" class and in corporate this class in the learning of the decision tree. But then we should have examples of all possible classes of drawings which should be rejected. This means that there should be almost an infinity number of drawings for this rejection class in order to learn all the drawings which should be rejected. Hence this approach does not work.

A workable solution is as follows. For a new drawing first the drawing is classified by the decision tree, say as class C. Afterwards the drawing is matched against a template constructed for class C. If the drawing differs to much from the template, i.e the difference is above a certain predetermined threshold, the drawing is rejected, i.e. considered out of domain.

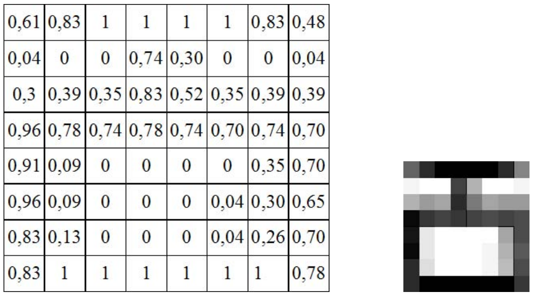
The template for each class C is constructed as follows. A template T of grid size $n \times n$ is taken. For each example e of the class under consideration a bounding box around the drawing is determined, afterwards for each grid (i, j) in the template T the number of lines crossing this grid is determined. This is the value of $T_e(i, j)$. After calculating T_e's for each example of the class in the training set the resulting T_e's are averaged over the number of elements of class c in the training set, this is the template T_C for class C.

$$T_C(i,j) = \frac{1}{N} \sum_{e \in C} T_e(i,j)$$

with N the number of elements of class C in the training set.

For the class "Detonator" the resulting template is for $n = 8$ given in Fig. 3.

After constructing the template T_C for each class C, the rejection threshold RT_C for each class needs to be determined. This threshold is the maximum



0,61	0,83	1	1	1	1	0,83	0,48
0,04	0	0	0,74	0,30	0	0	0,04
0,3	0,39	0,35	0,83	0,52	0,35	0,39	0,39
0,96	0,78	0,74	0,78	0,74	0,70	0,74	0,70
0,91	0,09	0	0	0	0	0,35	0,70
0,96	0,09	0	0	0	0,04	0,30	0,65
0,83	0,13	0	0	0	0,04	0,26	0,70
0,83	1	1	1	1	1	1	0,78

Fig. 3. The template for the class “Detonator” for $n = 8$

distance of the templates T_e and the class template T_C. The maximum is taken over all the examples e in the training set belonging to class C.

$$RT_C = max_{e \in C}\{\|T_e - T_C\|\}$$

where

$$\|T_e - T_C\| = \sum_{i,j} |T_e(i,j) - T_C(i,j)|$$

5 Test Results

To test the performance of the global system we once used again the available data set of 900 drawings. From this data set 600 examples were taken to train the decision tree, using C4.5 with pruning at a confidence level of 0.1, c.f. [9]. Also the rejection threshold for the template matching was determined using this training set, as described in the previous section, Sect. 4.

5.1 Classification Performance of the Decision Tree

On the test set the average performance of the decision tree was 90.4%. The most important confusions made by the decision tree are listed in Table 2. From this table it follows that there is a relatively large confusion between Television drawings and Nintendo drawings. Looking at the example drawings, Fig. 2, this can be explained by the fact that for the extracted features there is only a small

Table 2. The important confusions made by the decision tree. Rows correspond to actual classes, columns to classified classes.

	Mobile	Nintendo
Mobile	82 %	9 %
Nintendo	18 %	76 %

	Hammer	Television
Hammer	84 %	8 %
Television	11 %	81 %

	Bomb	CD
Bomb	90 %	2 %
CD	10 %	88 %

difference in the number of horizontal lines, namely 1. The confusion between a bomb and a CD is due to the fact that the inner circle of the CD is sometimes not detected. The confusion between a Hammer and a Television is not that easily explained. Confusion between a bomb and a CD could be detrimental for the game play. This confusion could be resolved by using other input modalities such as speech, or by letting the player select between the most likely alternatives. This last option would slow down the game speed.

5.2 Performance After Template Matching

After a drawing is classified, say as class C, then it is matched against the template of that particular class in order to determine if the drawing is out of domain, c.f. Subsect. 4.2. This lead to the following results, c.f. Table 3. It should be observed that we only tested with example drawings each of which belonged to a class, since we did not have out of domain drawings in our training and test set. From Table 3 it follows that 26 instances of misclassified drawings are

Table 3. The performance of the template matching procedure. TD stands for the decision tree classification procedure and TM for template matching procedure.

DT	TM	number
correct	not rejected	271
correct	rejected	13
not correct	rejected	26
not correct	not rejected	4

also rejected by the template matching procedure which is a good sign. But 13 correctly classified drawings are rejected by the template matching procedure, which is a bad sign.

The overall performance of the classification system, correctly classified by the decision tree and not rejected by the template matching procedure, is 271/314, which equals 86.3%.

6 Conclusions and Future Work

The results of our experiments show that with devices that are now becoming available in the the consumer market, effective image recognition is possible, provided a clear application domain is selected. In our case, this domain was the usage of simple images as input modality for computer games that are typical for small hand held devices. Since devices such as the Nintendo DS are likely to have possibilities for limited forms of speech recognition, it would be worthwhile to investigate the fusion of such speech data with the image data from the sketch pad. This could also resolve confusions, such as between a bomb and a CD, as such a confusion is detrimental for the game play.

The image recognition process itself could also be improved: The set of features that we have used turned out to be sufficient for a limited set of drawings consisting of straight lines and circles, but this might change as soon as we enlarge this class, and allow curved lines, filled areas, 3D pictures etcetera. Also, the classification that was based on decision trees might be replaced by other techniques, such as neural nets or Bayesian networks.

The current approach assumes that aspects such as training or learning are offline processes: there is no learning phase where the end user can train its own device. Of course it is questionable whether end users are willing to go through such processes. It seems more attractive to introduce processes that will fine-tune an existing classifier based upon data that is collected while the final application is in use.

References

1. Douglas, D., Peucker, T.: Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. Canadian Cartographer **10** (1973) 112–122
2. Anzai, Y.: Pattern Recognition and Machine Learning. Academic Press (1989)
3. Theodoridis, S., Koutroumbas, K.: Pattern Recognition. Academic Press (1999)
4. Pandya, A., Macy, R.: Pattern Recognition with Neural Networks in C++. CRC Press (1996)
5. Fonseca, M., Pimentel, C., Jorge, J.: Cali: an online scribble recognizer for calligraphic interfaces. In: Proc. AAAI Spring Symposium on Sketch Understanding. (2002) 51–58
6. Caetano, A., Goulart, N., Fonseca, M., Jorge, J.: JavaSketchIT: Issues in sketching the look of user interfaces. In: Proc. AAAI Spring Symposium on Sketch Understanding. (2002) 9–14
7. Parizeau, M., Lemieux, A., Gagné, A.: Character recognition experiments using unipen data. In: Proc. Int. Conference on Document Analysis and Recognition. (2001) 481–485
8. Duda, R., Hart, P., Stork, D.: Pattern Classification. Wiley New York (2001)
9. Quinlan, R.: C4.5: programs for machine learning. Morgan Kaufmann Publishers Inc. San Francisco, CA, USA (1993)