Gesture Recognition Using Motion Estimation on Mobile Phones

Sven Kratz
Media Computing Group
RWTH Aachen
skratz@post.rwth-aachen.de

Rafael Ballagas
Media Computing Group
RWTH Aachen
ballagas@cs.rwth-aachen.de

ABSTRACT

We present the mobile pervasive game REXplorer as the context of the development of a unique spell-casting user interface for mobile phones. A detailed description of the interface's algorithms, especially gesture recognition, is given. We go on to discuss alternative approaches and also focus on possible user feedback modes. Finally, we describe experimental methods which we intend to use for further improvement of our interface.

1. INTRODUCTION

REXplorer [2] is a mobile pervasive game due to be launched in Regensburg, Germany in summer 2007. Custom-built game devices, consisting of a Nokia N70 mobile phone and a Bluetooth GPS receiver housed in a wand-shaped metal casing, will be available for rent by tourists. As seen in Figure 1, spell-casting is REXplorer's main interaction paradigm. In order to support this mode of interaction, a gesture recognition system for camera-based motion has been designed. Camera-based motion data is problematic to use for gesture recognition due to the low quality of the data, caused by a low sample rate combined with high noise.

We have designed a gesture recognition algorithm that tries to solve this problem by using state machines, modeled from a gesture rule set, that parse the motion data and interpret the gesture the user has performed.

We will show that our algorithm adapts well to REXplorer's requirements and also discuss the two possible user experiences the algorithm can provide: either running the recognition while the user is performing the gesture, or after the user has explicitly terminated the gesture. Finally, we will describe how we intend to further improve our gesture recognition engine.

2. RELATED WORK

Recent research has shown how modern mobile phones support motion detection using their on-board camera.

Camera-based motion estimation was first proposed as positioning input for mobile phones by Rohs [8]. The "Sweep" Technique [1] extends this concept by using motion estimation on mobile phones as an input method for large public displays.

TinyMotion [10] offers a standard camera-based motion estimation library for Linux-based mobile phones. A set of

Figure 1: A REXplorer user performing a gesture

example applications for it's target device, a Motorola V710 Mobile Phone were developed and user-tested by the authors. The example applications explore numerous uses for camera-based motion estimation, for example for text entry or gaming. What is especially relevant is that a commercial handwriting recognition system was used successfully for text input using TinyMotion's motion estimation. However, Wang et al. [9] state that gestural interfaces may be problematic due to the low image frame rate. REXplorer, however is the first application to use motion estimation technology to enable spell-casting input on mobile phones.

A lot of work has been done in the field of handwriting recognition systems that could be adapted to our needs. However, most of these algorithms are very complex, employing neural network classifiers [6] or using stochastical methods such as Hidden Markov Models [4] for classification. These traditional gesture recognition engines typically use a library of predefined gesture traces, which are often entered by the user in a learning phase. As the typical REXplorer user will only use the device once, a learning phase for the gesture classifier is undesirable. While these algorithms feature a very good recognition rate operating on large and complex

gesture vocabularies, we feel that they are too complex and resource-intensive for our target platform and application area.

3. IMPLEMENTATION

Our gesture recognition engine can be divided into three components. Movement data is obtained via motion estimation. While the gesture is being performed by the user, a "live" graphical trace motion data is provided. This aids the user in performing the intended gesture correctly. In a final step, the accumulated motion information is evaluated by the gesture recognition algorithm to determine which gesture has been performed.

Motion Estimation

The motion data for our gesture recognizer is obtained using camera motion estimation. We employ a block-matching algorithm similar to the ones described in [8, 10]. The last two frames of video input are compared. Both frames are subsampled to roughly $\frac{1}{8}$th of their former size. We test the MSE (Mean Square Error) between the old and a shifted version of the new frame, with a shift range of 3×3 pixels. The candidate with the lowest MSE then delivers the motion vector.

User Feedback

Feedback is provided to the users by rendering a trace of the current gesture's progress to the device's screen. Preliminary user testing has suggested that users have difficulty coping with large amounts of noise in the gesture visualization. Several users aborted gestures prematurely because of noise in the feedback, even though the gesture recognition system would likely correctly identify a completed gesture. This suggests that we need to find ways to reduce noise in the user feedback.

Gesture Recognition

For REXplorer, we decided to implement a gesture recognition algorithm that incrementally matches the input to a gesture by verifying the entered motion data reaches certain predefined distance offsets.

Let $M = m_1, \ldots, m_n$ be a sequence of motion tuples with $m_i \in \mathbb{N}^2$, obtained by user input. When adding up the motion tuples we obtain a "gesture trace" $T = t_1, \ldots, t_n$ with

$$t_j = \sum_{i=0}^{j} m_i \quad m_i \in M$$

As shown in Figure 4, a gesture trace can be plotted in 2D space to obtain a graphical representation of the gesture.

Due to the fact that users perform gestures that cover a varying area of pixels and that we only specify fixed offsets in our gesture configuration files, the input gesture trace is normalized to make its points lie within an area of 100×100 pixels before gesture recognition.

Our algorithm determines which predefined rule set is matched best by the entered motion data. A rule set defines a sequence of distance offsets that must be fulfilled by the trace of the entered motion data.

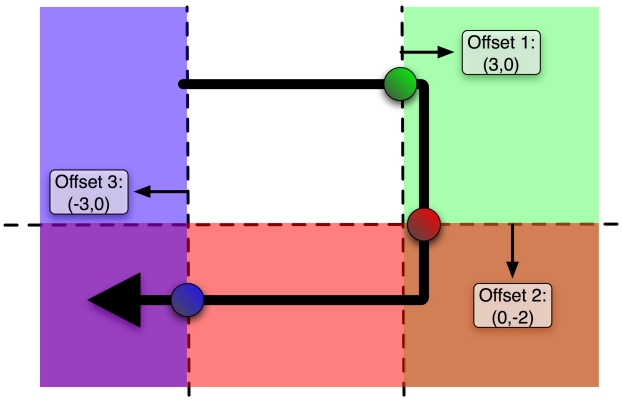

Figure 2: Recognizing the gesture using offsets

We model each gesture as the acceptance state of a state machine $\mathfrak{G}$. Upon initialization, a gesture configuration file with the sequence of predefined offset rules is loaded. Each state q_i of a state machine $\mathfrak{G}_n$ represents an offset rule. An offset specifies the (Manhattan-)distance (x, y) in pixels between the states q_i and q_{i-1}, in other words the first occurrence of the value $\delta = t_j - t_i$ with $j > i$ and t_i, t_j being two points in the gesture trace. A gesture is accepted if $\mathfrak{G}_n$ reaches it's acceptance state q_{accept}.

For instance, if, as in Figure 2, q_2 is defined as being $(-2, 0)$ pixels away from q_1, which was matched at the coordinates $t_i = (2, 6)$, then the automaton will change it's state to q_2 when incoming motion data can be traced to a location $t_j = (2 - k, n)$, where $k > 3$ and $j > i$.

The gesture recognizer A is thus modeled as the following union automaton:

$$\mathfrak{A} = \mathfrak{G}_1 \cup \mathfrak{G}_2 \cup \ldots \cup \mathfrak{G}_k$$

So it is evident that

$$F = \{q^{(1)}_{\text{accept}}, \ldots, q^{(k)}_{\text{accept}}\}$$

is the acceptance set of $\mathfrak{A}$.

Because it is theoretically possible that multiple gestures can be recognized from a single user input. We pragmatically modify the union criterion so that the first automaton that accepts, be it $\mathfrak{G}_n$, will yield the recognition of gesture n, while the results of the remaining automata that have accepted will be discarded. This modification is justified because we use a carefully chosen gesture vocabulary (see Figure 3) composed of a few very distinct gestures that not only lowers the user's learning curve but also eliminates almost all falsely recognized gestures.

Our algorithm has proven itself to robustly handle the mediocre motion data. This is because, conceptually, our system is very forgiving and usually leads to a correctly recognized gesture, even if the graphical representation of the gesture trace might not be similar at all to the predefined shapes. An explanation for this is that we are, abstractly seen, testing if

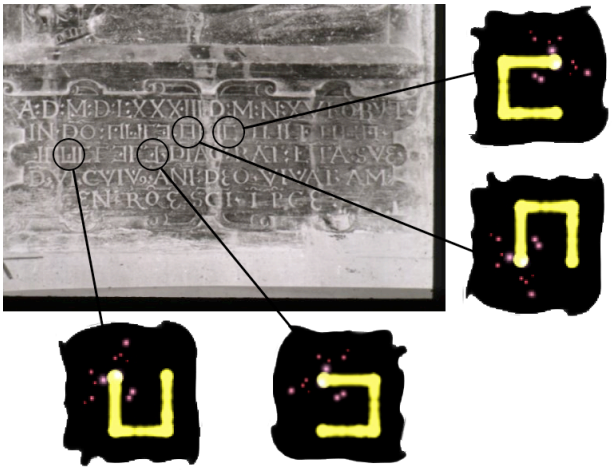

Figure 3: The gesture vocabulary is inspired by a medieval tombstone in Regensburg, inscribed with a mysterious secret language. Players are told that these shapes represent spells that awaken spirits trapped inside of historical buildings.

our internal cursor has entered the infinite plain[1] specified by the next automaton state's offset vector. This cancels out noise from inadequate motion data quite effectively as it is possible to move away from or orthogonally from the specified direction before moving correctly and finally reaching the next state.

Due to our algorithm's coarse matching criteria, it does not perform well with large and complex gesture vocabularies. Here a finer analysis of the data is required. For instance, angle sums, point distances or mean absolute distances can be used for more detailed classification. But taking all into account, our algorithm's low complexity but relatively high recognition rate allows for an efficient implementation that performs well with a simple gesture set on our target platform.

4. ALTERNATIVE IMPLEMENTATIONS

As mentioned earlier, a wide variety of gesture recognition algorithms exist. This assertion also applies to motion estimation algorithms. Furthermore, our gesture recognition algorithm supports a "live" operation mode that can lead to new feedback options. We are currently evaluating the following alternative solutions to the challenges posed by REXplorer.

Motion Estimation

We have seen so far that a gesture recognition algorithm that uses motion data from camera-based motion estimation is required to be tolerant of a low sampling rate and of a high noise ratio.

A way to globally improve gesture recognition from camera-based motion estimation is to augment the quality of the motion estimation data itself by using more complex (but not less efficient) motion estimation algorithms from the field of video compression, such as Three-Step-Search [7].

However, an evaluation of these algorithms must be undertaken to see how they adapt to our application domain. For instance, a comparison of the complexity and mean square error (MSE) rates of fast block-matching motion estimation algorithms (BMA) used in video compression can be found in [5]. These algorithms feature a low computational complexity and show a low MSE when detecting motion in the individual blocks that a video image is composed of. We, in contrast, have not divided our source image into multiple blocks, but instead analyze only one block due to the high complexity of our BMA. Because we assume that the presented block-matching algorithms are better suited to estimate local movement in the individual blocks (from which a frame is composed), but less suited to estimate global movement of an entire frame sub sampled as a single block, it appears necessary for us to change and re-evaluate our sub-sampling strategy if we are to use the presented results to chose our BMA.

[3] presents a comparison of different block sub-sampling strategies, but still assumes that the image is completely divided into blocks before the BMA is applied. We intend to evaluate which (block) sub sampling strategy best fits our needs (i.e. if we need to divide the whole image into blocks or if it is sufficient to analyze blocks from select regions of the image) and if using a fast BMA with multiple blocks from the image will yield higher-precision (global) motion data, as opposed to our current single-block approach.

User Feedback

Our state-machine based algorithm can be employed in two ways, providing two different user experiences. It can either run "live", processing motion data while the gesture is being performed by the user, or as is presently the case, the motion data can be saved for gesture recognition after the user terminates the gesture.

Running the gesture recognizer in the "live" mode allows for incremental feedback. This Gesture-Subtarget visualizer mode works together with our state-machine-based algorithm and displays a red circle at each coordinate in the gesture trace where an automaton has matched a target rule. This implementation allows us to give multimodal feedback (haptic, visual, or audio) to the users after each target rule has been met. For example, an idea for audio feedback is to play sounds with a gradually rising tension at each matching point to indicate how close the gesture is to completion. See Figure 4 for a depiction of some of the proposed visualization modes.

We would like to see which of the algorithm's operational modes is more acceptable to our target users. For example, does "live" gesture recognition confuse the user when motion estimation is stopped as soon as a gesture is recognized? Maybe the proposed incremental feedback might lift this confusion. On the other hand, the higher processing time reserved for "offline" gesture recognition might permit some more detailed processing of the motion data as the CPU-intensive motion estimation is not performed simulta-

[1]These plains are represented by the colored areas in Figure 2

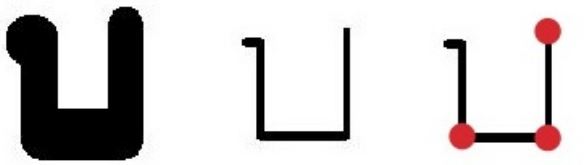

Figure 4: Thick, Thin and Gesture-Subtarget Visualizer modes, generated from actual user input

neously. Also, live gesture recognition prevents normalization of motion data, which may decrease gesture recognition performance.

Gesture Recognition

Improved gesture recognition algorithms need to be explored in terms of correct gesture recognition rate and performance costs. We would like to evaluate a more classical approach using an algorithm that matches the difference between the trace obtained by user input and that of a set of predefined traces. This approach could be implemented as follows:

1. Normalize the trace of the motion data
2. Normalize the number of points in the trace. This means reducing the number of points in the gesture trace to n equidistant coordinates.
3. Calculate the (average) difference between the normalized trace's points and the set of predefined traces
4. The predefined trace with the lowest (average) difference is the recognized gesture

The advantage of this method is that while it is more CPU intensive than our current algorithm, it might improve recognition of more complex gestures, and reduce the number of falsely recognized gestures. However, it remains to be seen how such an algorithm performs in terms of the low-quality motion data as previously discussed and in terms of the computational resources of our target device.

5. FUTURE WORK

We intend to optimize our motion estimation algorithm while evaluating different user feedback mechanisms and gesture recognition algorithms. Our goal is to achieve a high gesture recognition rate while retaining a high user acceptance rate.

We are specifically interested in how the quality of the motion data, visual and/or aural feedback and the "online" and "offline" versions of our gesture recognition algorithm interact. To this aim, we will use a high-end webcam mounted on the REXplorer device and optical tracking of the user's movements to generate baseline motion data. The webcam's higher frame rate and resolution compared with the cameras present on most available mobile phones today allows us to experiment with a wider numerical range of parameters. Thus, this will also enable us to predict how future camera-based motion estimation applications might perform.

We would ultimately like to find out what are the minimum requirements for a usable gestural interface based on camera motion estimation and how they can be met on a given platform.

6. CONCLUSION

Spell-casting is a novel way to use mobile phones to interact with our environment. In order to successfully implement spell-casting with a mobile phone, one must carefully engineer the motion estimation, gesture recognition, and user feedback mechanism. We have presented our current prototype implementation of gesture recognition using mobile phones in REXplorer, as well as proposed potential improvements based on preliminary user testing. By empirically testing these improvements in future work, we hope to inform future camera-based gesture input using mobile phones.

7. REFERENCES

[1] Rafael Ballagas, Michael Rohs, Jennifer G. Sheridan, and Jan Borchers. Sweep and Point & Shoot: Phonecam-based interactions for large public displays. In *CHI '05: Extended abstracts of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1200–1203, New York, NY, USA, 2005. ACM Press.

[2] Rafael Ballagas, Steffen P. Walz, Sven Kratz, Claudia Fuhr, Eugen Yu, Martin Tann, Jan Borchers, and Ludger Hovestadt. Rexplorer: A mobile, pervasive spell-casting game for tourists. In *To appear in CHI '07 extended abstracts on Human factors in computing systems*, San Jose, CA, USA, 2007. ACM Press.

[3] Eric Debes Fulvio Moschetti. A fast block matching for simd processors using subsampling. In *ISCAS 2000 - IEEE International Symposium on Circuits and Systems, May 28-31, 2000, Geneva, Switzerland.*

[4] Brown M. Hu, J. and W. Turin. Hmm based online handwriting recognition. In *IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. PAMI-18, pp. 1039-1045*, 1996.

[5] Yilong Liu and Soontorn Oriantara. Complexity comparison of fast block-matching motion estimation algorithms. In *IEEE Int. Conf. Acoust., Speech, Signal Processing 2004.*

[6] Dzulkifli Mohamad Muhammad Faisal Zafar and Razib M. Othman. On-line handwritten character recognition: An implementation of counterpropagation neural net. In *Transactions on Engineering, Computing and Technology ISSN 1305-5313*, December 2005.

[7] B. Zeng R. Li and M. L. Liou. A new three-step search algorithm for block motion estimation. In *IEEE Trans. Circuits Syst. Video Technol., vol. 4, no: 4, pp. 438-442, Aug. 1994.*

[8] Michael Rohs. Real-world interaction with camera phones. In *International Symposium on Ubiquitous Computing Systems (UCS 2004), Tokyo, Japan, http://www.vs.inf.ethz.ch/publ/papers/rohs2004-visualcodes.pdf Also published under Revised Selected*

Papers, pp. 74-89, LNCS 3598, Springer, July 2005.

[9] Jingtao Wang and John Canny. End-user place annotation on mobile devices : A comparative study. In *Work-in-Progress of ACM CHI 2006, Montreal, Canada, April 24-27, 2006.*

[10] Jingtao Wang, Shumin Zhai, and John Canny. Camera phone based motion sensing: Interaction techniques, applications and performance study. In *UIST '06: Proceedings of the 19th annual ACM Symposium on User Interface Software and Technology.* ACM, 2006.