iGesture: A General Gesture Recognition Framework

Diploma Thesis

Ueli Kurmann
<kurmannu@ethz.ch>

Prof. Dr. Moira C. Norrie
Dr. Beat Signer

Global Information Systems Group
Institute of Information Systems
Department of Computer Science

9th January 2007

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

globis

Abstract

Gestures have high potential to be used in paper and pen-based user interfaces. However, so far there exists no framework that on the one hand is powerful enough to satisfy the requirements of recent applications and on the other hand is easy to integrate. We provide an overview of existing approaches and present iGesture, our new solution to solve this problem. iGesture is a Java-based extensible framework and a set of related tools for gesture recognition, the creation and management of gestures as well as the evaluation of recognition algorithms. As part of the project, we described and implemented two existing and a new algorithm. The existing algorithms were extended to improve their recognition quality. All algorithms have been tested with different kind of gesture sets and users proving that our framework is sound. Finally, high recognition rates are achieved using our improved algorithms introduced in this thesis.

Contents

1 Introduction

The Global Information Systems Group at ETH Zurich provides frameworks for the development of pen and paper-based applications. However, so far there is no support for gesture recognition. Pen gestures are an important technique to execute commands in pen-based applications and become more relevant in recent projects.

The topic of this diploma thesis was first to analyse existing gesture recognition frameworks and the corresponding recognition techniques. Based on these investigations a recognition framework has been developed and tools to create gestures as well as mechanisms to evaluate and test algorithms have been designed.

The resulting framework provides a simple application programming interface (API) to recognise gestures provided in a specific format. A user application can either handle results directly or make use of an event manager that executes commands based on recognised gestures. We have further implemented three different algorithms to be used in the recognition process.

With our new tool, gesture sets can be created and managed. A test-bench enables the manual testing of algorithms and special functionality is provided to create the test data. Last but not least we provide tools to evaluate different algorithms and their configurations in batch mode and visualises the results.

Chapter 2 provides a survey about related work in the area of mouse and pen-based gesture recognition. Beside research papers different existing frameworks are presented and analysed. The implementation and architecture of our new gesture recognition framework and the related tools are described in Chapter 3. We provide class diagrams for important parts of the architecture and discuss some of our design decisions. Chapter 4 provides theoretical information about the algorithms and introduces our new features for the classification process. In the first part two existing and a new algorithm, which was developed as part of this diploma thesis, are described in detail. The second part proposes enhancements in terms of additional features for an existing algorithm to improve its quality. The user guide in Chapter 5 explains the functionality and usage of the graphical iGesture tool for creating and managing gestures. With a simple application the usage of the iGesture API is demonstrated and last but not least the use of the batch tool to evaluate different recognition algorithms is described. Finally, Chapter 6 defines key figures which can be used to compare different algorithms and their configuration. We have conducted eight experiments with different gesture sets, training data and users and the different algorithms have been ranked.

2 Related Work

Gestures are omnipresent in communication between humans and more recently they became more interesting for the interaction with computers. On the one hand there are the human hand and body gestures which are recorded with cameras and sensors. On the other hand there are two dimensional gestures for pen and mouse-based applications which are also the focus of this thesis.

Pen-based applications became widely adapted with the emergence of PDA computers. A new kind of user interface was needed because a keyboard was not handy enough for these small devices. The PDA systems support gesture or handwriting recognition with vendor and application-specific tools. An early example of such a vendor-specific tool is Palm's *Graffiti* font for handwriting recognition.

The use of pens also became more popular in the field of Tablet PCs mainly pushed by Microsoft. Consequently, Microsoft provides an SDK to develop applications with gesture-based user interfaces. However, such interfaces are often not using the full potential of pen-based solutions since existing mouse and keyboard-based interfaces are just adapted or enhanced with some gestures. These adoptions often lead to constrained interfaces with a low user acceptance.

Another example are CAD applications with graphic-tablet-based gesture support where the user does not have to change input devices between the drawing and the execution of common commands such as for example an undo operation. These applications are popular and can enhance productivity.

In the remaining part of this chapter we present a short overview of existing mouse or pen-based gesture recognition frameworks and user interfaces.

2.1 Specifying Gestures by Example

Specifying Gestures by Example [20] was published by Rubine in 1991. It describes *GRANDMA* (Gesture Recognizers Automated in a Novel Direct Manipulation Architecture), an object-oriented toolkit for rapidly adding gestures to direct manipulation interfaces and introduces a specific classification algorithm.

The idea behind this project is the creation of gestures by example. In this context a gesture is a two dimensional sketch or symbol drawn with a computer mouse. Gesture by example means that a specific gesture, also termed *gesture class*, is described by some sample gestures and does not require any further programmatic adaption of the recognition algorithm. The algorithm itself is therefore independent from the sample sets defining the gestures.

The set of gesture classes used for the recognition process is called *gesture set*. A set has to be defined when the algorithm is initialised. Note that a gesture class may occur in different gesture sets.

An evaluation shows that the rate of correctly classified gestures depends on the size of the gesture set and the number of examples provided for each class. For gesture sets with 15 different classes and at least 15 examples per class up to 98% of the test gestures are recognised correctly. Also sets of 30 classes achieve an error rate of only 3% with 40 examples or 4% with 15 examples, respectively.

The classification algorithm uses statistical single-stroke gesture recognition based on 13 different features proposed by Rubine. A small change in the gesture should lead only to a small change in the computed feature value and there should be enough features to distinguish all gestures of a specific set.

The training problem is to determine the weights for each gesture class and feature based on a given set of examples. In a first step, the feature vector for each example is computed and the outcomes are summarised in a mean feature vector for each gesture class. With the help of a covariance matrix the weights per class and feature can be computed. This computation has to be done only once during the algorithm's initialisation phase. Afterwards, an arbitrary number of gestures can be recognised with a simple linear computation per class. Because a linear classifier will always find a gesture which matches best, a mechanism to reject input is necessary. If two gestures have nearly the same classification value or a test gesture is detected as outlier, the input should be rejected.

The publication by Rubine is well known in the gesture recognition research area and it is cited by many later papers.

2.2 SATIN: A Toolkit for Informal Ink-based Applications

Hong and Landay proposed *SATIN* [14], a toolkit for informal ink-based applications. SATIN is an open source project available at Sourceforge [7] and is written in Java. It provides various components including different recognisers and an interpreter that supports the creation of pen-based applications.

As part of the project they did a survey of existing pen-based applications and extracted common functionality such as input as ink, input as gestures, input for selecting and moving, interpreters that act on ink input or the transformation of ink to cleaned-up objects.

For the recognition process, SATIN uses the Rubine algorithm. The recogniser is responsible for the classification only, whereas interpreters perform actions based on the input. There are two kinds of interpreters: the gesture interpreters to execute commands and the ink interpreters which clean up the input and display the result.

A new concept of SATIN is the *multi-interpreter*. This is a collection of different interpreters combined with policies to decide which concrete interpreter should be used. A default multi-interpreter iterates over the algorithms and stops as soon as one of them returns a result.

SATIN uses a tree-like data structure to arrange graphical objects which are defined by multiple strokes, the simplest primitive graphical object. Graphical objects have two dimensional coordinates as well as a layer coordinate to arrange them on them z-axis.

An important part of SATIN is the extension and optimisation of Java Swing components for pen-based applications. A new component is the *pie menu* to replace common popup menus. Furthermore, SATIN provides tools for manipulating strokes and an algorithm is proposed to simplify strokes by removing non relevant points to speed up an application's performance.

Two sample applications have been implemented with SATIN. One of them is DENIM, a sketch-based website design tool. The other is SketchySPICE which is a simple circuit CAD tool. Both use a pen as the primary input device.

Unfortunately, SATIN is no longer maintained since 2001.

2.3 quill: Providing Advice for Pen-based Gesture Design

quill [16] is a toolkit to design gestures and has been developed by Long, Landay and Rowe. A problem of gesture-based interfaces is often the similarity of gestures by what it is difficult to develop recognisers with a low error rate. quill, which is based on *gdt* (gesture design tool) [17], supports the gesture designer with advices when two different gestures are ambiguous either for humans or recognisers.

They did various studies about the user acceptance of gesture-based applications and found out that users like the concept of gesture-based user interfaces. Furthermore, they recognised that users wanted gestures to be easy to memorise and more fault-tolerated in terms of the recognition.

To measure the similarity and how memorisable a gesture is for a user Long et al. constructed a computational model based on geometric gesture features. This model allows quill to predict when people will perceive gestures to be very similar and therefore may have problems to learn and distinguish.

The most novel feature in quill is the active feedback. The similarity for recognisers and humans, outlying categories and gestures and the number of misrecognised training examples is monitored, quill then warns the gesture designer about these problems and assists them with textual advices to create more reliable gestures.

2.4 SketchREAD: A Multi-Domain Sketch Recognition Engine

SketchREAD [11] is a multi-domain sketch recognition engine developed by Alvarado and Davis. The engine enables the recognition of two dimensional hand-drawn diagrammatic sketches. The strokes that a user draws are interpreted as objects in a domain of interests.

The information about basic shapes is separated from the interpretation in a certain domain. This has the advantage that recognisers can be reused in different domains. For the recognition dynamically constructed Bayesian networks are used. These networks are influenced not only by strokes but also by higher-level shape information.

SketchREAD uses a hierarchical shape description language to describe the interpretation of shapes in a specific domain. A shape may consist of subshapes and have several constraints describing the relationship between them. The context of the shape is also taken into account. Therefore, a shape may have different interpretations in different contexts.

Two example applications have been implemented based on the SketchREAD framework. First, there is a family tree application and second an application to draw simple circuit diagrams is presented.

2.5 LipiTk: A Generic Toolkit for Online Handwriting Recognition

LipiTk [18] is a generic toolkit for online handwriting recognition and has been developed by the Hewlett-Packard Laboratories in India. The toolkit aims to simplify the creation and integration of recognisers for new scripts and should be useful for various user groups. For researchers it is important to be able to implement and test new algorithms whereas an application developer wants a simple API to integrate functionality in applications.

Most of the toolkit is implemented in C++ and runs on both, Linux and MS Windows systems. It provides generic pre-processing operations as size-normalisation or equidistant re-sampling. LipiTk is bundled with two recognition algorithms: Subspace-based classification and Nearest-Neighbour classification. There are also utilities for the data collection and the evaluation of different algorithms.

LipiTk was released in a first version in April 2006 and is hosted at SourceForge [3]. The development is still in progress and it has been announced that other pen interfaces should be added and open standards like InkML should be supported.

2.6 Microsoft Tablet PC SDK

The *Microsoft Table PC SDK* [5] contains a component for gesture recognition. Thereby Microsoft distinguish between *system* and *user application gestures* [4]. System gestures are the pen-based equivalent representation of mouse events. With the system gestures the mouse can be replaced and all mouse actions can be done with the pen. User application gestures are a set of about 40 gestures proposed by Microsoft to interact with Tablet PC applications. An overview of these gestures is provided in Appendix A.2.

Some of these gestures have a fixed interpretation and it is recommended to follow this guidelines to prevent confusion. Unfortunately, the recogniser is limited to these gestures and other gestures can only be integrated by implementing new recognisers.

The recogniser itself has different modes where the developer can decide whether gestures and ink or only one of them should be recognised. If the recogniser should detect ink and gestures, only single stroke gestures are possible. On the screen only ink and no gestures are shown. The recognition is done after each stroke and based on its result an event is fired or the stroke is interpreted as ink.

Beside gesture recognition, the Tablet PC SDK also supports the recognition of handwritten text. The SDK is based on Microsoft .NET and depends on Windows XP.

2.7 SwingGestures

SwingGestures [9] is a SourceForge project initiated by Alberto Bengoa Moreno. The framework is targeted to add simple gestures to Java Swing applications. Eight basic gestures, namely up, down, left, right and the four diagonals are hard coded. Other gestures can only be constructed out of these basic gestures which limits the power of the framework.

There is a single instance of the GestureCenter which is a kind of a facade to use the framework. Event listeners for specific gestures can be added to and also a simple event system is implemented to run actions based on the result of the recognition. Due to these architectural choices the framework is easy to use.

The recogniser itself runs in two modes to support simple and complex gestures and uses a simple algorithm to classify the gestures. The algorithm compares each mouse event with the simple or complex gesture set and calculates a fit value. On the basis of these values, the gesture which matches best is returned.

In our opinion this framework has some drawbacks. First, only a limited number of gestures can be represented because of the eight basic gestures. For example, gestures with curves are nearly impossible. Furthermore, the program code has to be adapted each time, when a new gesture has to be added to the application. Finally, the framework was not designed to support different algorithms.

SwingGestures was published in 2004 and has not been maintained or extended since then.

2.8 Design and Analysis of Delimiters for Selection-Action Pen Gesture Phrases in Scriboli

Hineckley, Baudisch, Ramos and Guimbretière analyse delimiters for select-action pen gesture phrases in *Scriboli* [13]. A select-action gesture phrase is a selection of an object and the choice of a command executed on that object. The selection is done with a lasso around the object and the command is chosen from a pie menu. In their work they compare four different delimiters to separate the selection part from the pie menu.

The *Pigtail* uses a small loop to delimit the lasso and the command to execute. The loop intersects the lasso and the part after the intersection describes the command. So the intersection point is the centre of the pie menu.

The *Handle* delimiter attaches a small box at the end of the stroke. This box acts as activation point and the command accordingly is drawn in a second stroke. This method has advantages in case of selecting complex objects or for the reuse of a selection.

Using the *Timeout* delimiter, the user has to observe a timeout of 500 ms after the selection and before the choice of the command. After the timeout the pie menu appears.

The fourth delimiter uses a *Button*. After the selection, the button is pressed and the command can be chosen. In the prototype the CTRL key is pressed with the free hand.

Hineckley et al. implemented all four delimiters and did the same experiments to evaluate time efficiency and error rates. The Handle technique had the best time performance where the timeout delimiter is slow and only a few testers liked the method with the button because of difficulties to catch the right moment. Pigtail seems to split the users. Some of them liked the method because it is intuitive where others could not manage to use it efficiently.

2.9 PapierCraft: A System for Interactive Paper

PapierCraft: A System for Interactive Paper [15] is written by Liao, Guimbretière and Hinckley in a joint work between the University of Maryland and Microsoft Research in Redmond. They analysed the drawback of computer systems versus traditional paper solutions in terms of knowledge workers. Printouts have many advantages to arrange information and to network different sources. Unfortunately, these annotations are only available on printouts and cannot be transferred to digital systems.

The chosen approach considers paper printouts as proxies of digital documents stored in the computer. As a user interface they used a gesture-based command system. A user can copy and paste parts from one document into another document, create links or stitch two paper documents together. As input device they use an Anoto pen and the synchronisation process executes all recorded commands in the digital version and the result is available in a digital document browser.

The most important point of PapierCraft was the development of a simple and reliable command system which respects current paper-based practices. The command system also should be readable for humans. The only immediately available feedback is the strokes on paper drawn with the pen. Therefore, the strokes should not only be understandable by computers but also by the users.

To distinguish between ink and gesture strokes, PapierCraft uses a foot pedal. Whenever the pedal is pressed during a stroke, it is recognised as gesture. For the distinction between the scope and command selection Scriboli's Pigtail approach has been adopted. Five different scope selectors allow the selection of single words, multiple lines or lassoing whole passages. The four most commonly used commands are directly available as in Pigtail. Other commands can be executed by writing down a unique prefix of the command name. Named commands have the advantage that they are easier to memorise and understand.

PapierCraft is implemented in C++ and uses the Microsoft Tablet PC SDK and the Anoto SDK.

2.10 Summary

The frameworks proposed by the different authors do not satisfy the needs of a general gesture recognition framework for iPaper. Such a framework should be easy to use for application developers, but also be powerful enough and extendable for upcoming requirements of new applications.

SwingGestures is too limited to be used as a general gesture recognition framework. Only a limited set of gestures can be created and the classification algorithm is not reliable enough. For simple Swing-based Applications it could be useful, but as soon as the set of gestures is becomes larger it is ineffectual.

Microsoft's Tablet PC SDK is designed for screen-based applications and implemented for the .NET environment. Also the limitation to a fixed set of gestures does not make this platform interesting to elaborate new pen-based user interfaces and applications.

SATIN is powerful but targeted to screen-based applications. A big part of SATIN is not the gesture recognition itself but rather the interpretation or beautification of strokes to build ink-based graphical Swing applications. Another drawback is that SATIN has not been maintained for the last few years and is still based on Java 1.3.

3 Implementation

iGesture is developed on the *Java* 1.6 platform and is based on *sigtec* [8] and *iPaper* [21]. sigtec is a library of common tool classes and is used for the ink representation and the XML functionality whereas iPaper provides access to Anoto-based input devices. Other libraries as the *Apache Jakarta Commons* [1] are used and mentioned in Section 3.11. Note that we paid attention to use only third-party libraries which are available on the basis of an open source license.

The iGesture framework consists of different partially independent components. Roughly there is the recogniser, a management console and tools for testing and optimising algorithms as shown in Figure 3.1. In addition to these components there exist common data structures and model classes that are used by all parts.

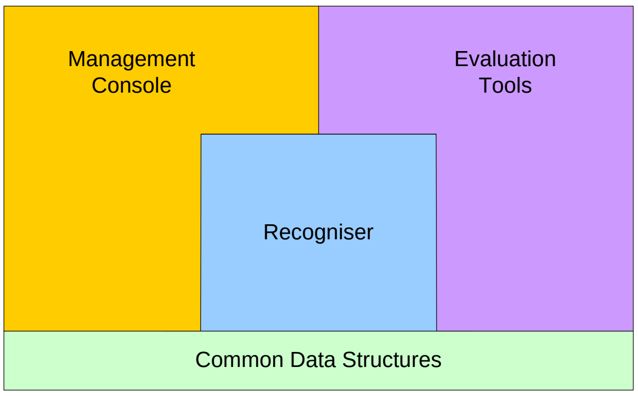


Figure 3.1: iGesture overview

The following sections provide information about the implementation of the framework and on how the components interact with each other.

3.1 Gesture Representation

The representation of gestures in the framework is an important design decision and has implications on all other parts. It was a requirement for the data structures to manage gestures and groups of gestures. Furthermore, different algorithms need different descriptions of a gesture. Therefore, it is important that the model classes do not make any assumptions about specific algorithms or provide algorithm-specific data. Figure 3.2 shows the UML diagram of our data structure for representing gestures.

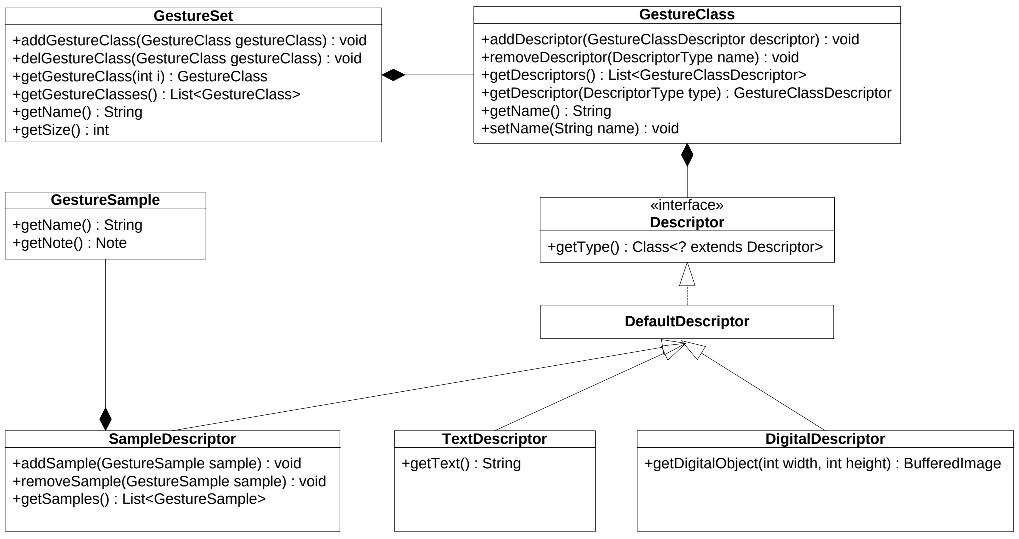

GestureSet
+addGestureClass(GestureClass gestureClass) : void
+delGestureClass(GestureClass gestureClass) : void
+getGestureClass(int i) : GestureClass
+getGestureClasses() : List<GestureClass>
+getName() : String
+getSize() : int
GestureClass
+addDescriptor(GestureClassDescriptor descriptor) : void
+removeDescriptor(DescriptorType name) : void
+getDescriptors() : List<GestureClassDescriptor>
+getDescriptor(DescriptorType type) : GestureClassDescriptor
+getName() : String
+setName(String name) : void
GestureSample
+getName() : String
+getNote() : Note
«interface»
Descriptor
+getType() : Class<? extends Descriptor>
DefaultDescriptor
SampleDescriptor
+addSample(GestureSample sample) : void
+removeSample(GestureSample sample) : void
+getSamples() : List<GestureSample>
TextDescriptor
+getText() : String
DigitalDescriptor
+getDigitalObject(int width, int height) : BufferedImage

Figure 3.2: Gesture representation class diagram

GestureClass
The `GestureClass` class represents an abstract gesture and therefore holds the name of the class and a list of descriptors characterising it. For example to gather circles as gestures, we instantiate a new `GestureClass` and set the name to 'Circle'. The class itself does not hold any information on how the gesture looks like and needs at least one descriptor characterising the circle as a graphical object.

GestureSet
The class `GestureSet` contains a collection of gesture classes. This aggregation is necessary to be able to initialise an algorithm with a specific set of gestures whereas a gesture class can be a member of different gesture sets.

Descriptor
`Descriptor` is an interface that all gesture class descriptors have to implement. It does not specify any methods concerning the retrieval of gesture descriptions and is therefore more a marker than a functional interface. The idea behind this is that we do not want to limit the functionality of descriptors and it is not possible to provide methods for arbitrary descriptors which cannot be specified in advance. Each implementation of an algorithm is responsible for the necessary type checks and casts to work with the desired descriptor. Therefore, algorithms have to know on which descriptors they can operate and have to check if the necessary descriptors are available for all classes in the gesture set.

SampleDescriptor
The `SampleDescriptor` class implements the `Descriptor` interface and describes gestures by samples whereas a sample is a instance of a gesture. This descriptor is used for training-based algorithms.

GestureSample
The `GestureSample` class is the instance of a gesture. In our case, it contains the time stamped locations gathered from the input device summarised as a `Note` [21]. Despite the stroke detection the data gathered from the input device is not modified. This allows the algorithms to work on the original data and delegates the preprocessing to them.

TextDescriptor
The `TextDescriptor` class implements the `Descriptor` interface and specifies a gesture in terms of text. This could for example be a character string which describes the directions between characteristic points of the gesture.

DigitalDescriptor
`DigitalDescriptor` is an abstract class describing the gesture as digital image. This descriptor is not suitable for the recognition but rather for the digital representation and can provide a digital image of the gesture. Therefore, our gesture representation is powerful enough to build design oriented applications. For example, we can draw a circle that is recognised with the help of another descriptor and the application can then present its digital version based on the digital descriptor.

3.2 Recognition Algorithm

One of the main goals of iGesture is the support of different algorithms. To provide maximal flexibility in the design and use of algorithms, we decided to provide a minimal interface as shown in Figure 3.3. The `Algorithm` interface has methods for the initialisation, the recognition, the registration of an event manager and to retrieve possible parameters and their default values.

An algorithm is initialised with an instance of the `Configuration` class. This object contains gesture sets, an event manager and algorithm-specific parameters which are managed in a name/value collection. The init method throws an exception if the initialisation process fails. This exception can have different types informing about the specific error that happened.

The configuration object can be managed using the Java API or by importing the data from an XML document with the structure described in Appendix B.2. The storage in an external XML file has the advantage that parameters and the gesture sets can be adjusted whiteout recompiling the application source code. However, the full power of the configuration object can only be accessed based on the Java API. It is for not possible to set an event manager within the XML file. However, event managers can be added based on the Java API after loading the XML file.

The implementation of an algorithm is responsible for the validation of the configuration object. This means that algorithms have to check if all the necessary parameters have been set and if all required descriptors are available. If the initialisation process does not throw an exception, the user can assume that the algorithm is ready to recognise gestures. Furthermore,

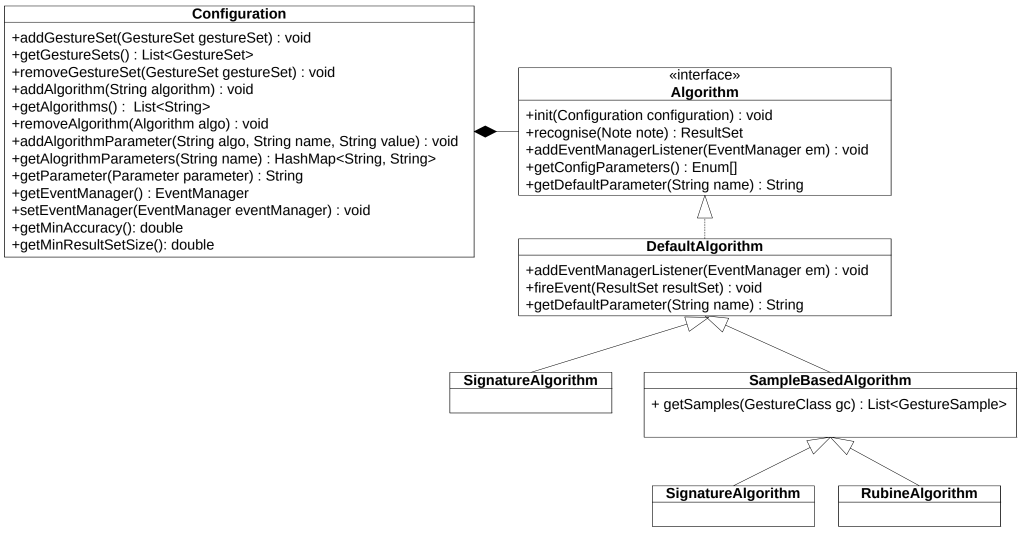

Configuration
+addGestureSet(GestureSet gestureSet) : void
+getGestureSets() : List<GestureSet>
+removeGestureSet(GestureSet gestureSet) : void
+addAlgorithm(String algorithm) : void
+getAlgorithms() : List<String>
+removeAlgorithm(Algorithm algo) : void
+addAlgorithmParameter(String algo, String name, String value) : void
+getAlogrithmParameters(String name) : HashMap<String, String>
+getParameter(Parameter parameter) : String
+getEventManager() : EventManager
+setEventManager(EventManager eventManager) : void
+getMinAccuracy(): double
+getMinResultSetSize(): double
«interface»
Algorithm
+init(Configuration configuration) : void
+recognise(Note note) : ResultSet
+addEventManagerListener(EventManager em) : void
+getConfigParameters() : Enum[]
+getDefaultParameter(String name) : String
DefaultAlgorithm
+addEventManagerListener(EventManager em) : void
+fireEvent(ResultSet resultSet) : void
+getDefaultParameter(String name) : String
SignatureAlgorithm
SampleBasedAlgorithm
+ getSamples(GestureClass gc) : List<GestureSample>
SignatureAlgorithm
RubineAlgorithm

Figure 3.3: Recognition class diagram

the implementation is obligated to observe the minimal accuracy and the maximal result set size stored in the configuration object and notifies the event manager about recognised gestures.

The `AlgorithmFactory` class provides static methods to create algorithms with a configuration instance and uses dynamic class loading to instantiate the algorithms.

3.3 Return Values

The result returned by the recogniser is structured into a result set which contains a list of results as shown in the class diagram in Figure 3.4. We decided to return a set of possible results instead of a single one to delegate the selection of specific results to the application making use of the recogniser. The advantage of this approach is that the application may choose a result based on additional contextual information about the captured gesture.

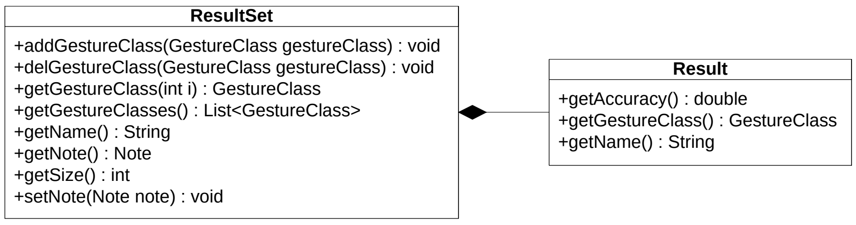

ResultSet
+addGestureClass(GestureClass gestureClass) : void
+delGestureClass(GestureClass gestureClass) : void
+getGestureClass(int i) : GestureClass
+getGestureClasses() : List<GestureClass>
+getName() : String
+getNote() : Note
+getSize() : int
+setNote(Note note) : void
Result
+getAccuracy() : double
+getGestureClass() : GestureClass
+getName() : String

Figure 3.4: Class diagram return value

The result set also contains a reference to the original note which was used in the gesture recognition process. This would not be necessary for the direct use of the gesture set returned to the application but enables events working on the note. Otherwise they would not be aware of this information.

The list of returned results is always ordered by the accuracy of the result or is empty if the recognition was unsuccessful. To simplify the access to the most likely result a convenience method is provided which directly returns the first result.

3.4 Event Manager

iGesture has an event manager providing an alternative method to react on recognised gestures in addition to the result set returned by the algorithm. Listeners can be added to an instance of an algorithm and an event manager is notified after each recognition process as shown in Figure 3.5.

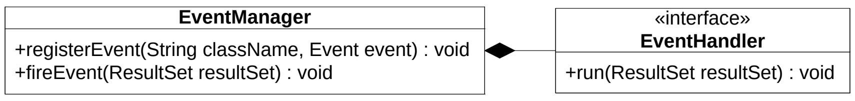

EventManager
+registerEvent(String className, Event event) : void
+fireEvent(ResultSet resultSet) : void
«interface»
EventHandler
+run(ResultSet resultSet) : void

Figure 3.5: EventManager class diagram

An event manager has advantages compared to the processing of result sets by the caller in terms of executing different actions based on the result. Thereby, it is not necessary to implement different behaviour for various results on the client side which reduces code complexity.

After a notification has been triggered by an algorithm the event manager looks up in the event table if the recognised gesture is bound to an action. If so, the action is executed by the event manager.

Actions have to implement the `EventHandler` interface demanding for a `run` method with a `ResultSet` as parameter.

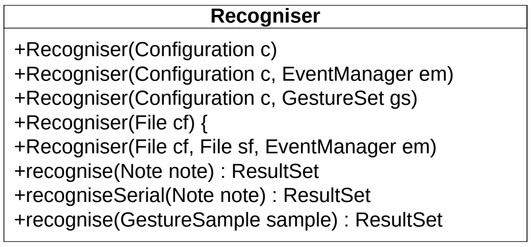
To be more flexible, the event manager contains instantiated objects. This has the advantage that the implementation of an event can be constructed with arbitrary parameters and therefore it is able to operate on all application data.

By definition, the algorithm has to notify the event manager after each call of the recognise method.

3.5 Recogniser

The `Recogniser` class shown in Figure 3.6 is the front end of the framework and hides the complexity with a facade [12]. An application developer using iGesture does not have to bother about the classes in the background. They only have to define an XML configuration file as shown in Listing B.2, create a gesture set and use the `Recogniser` to recognise gestures. In general, the recogniser is instantiated with a configuration object containing information about the algorithms to be used. To make the recogniser more comfortable we provide a broad set of constructors which allow to pass the configuration files directly.

A configuration object may have multiple algorithms for the recognition process. The recogniser provides two methods which have a different behaviour regarding to the use of multiple



```
Recogniser
+Recogniser(Configuration c)
+Recogniser(Configuration c, EventManager em)
+Recogniser(Configuration c, GestureSet gs)
+Recogniser(File cf) {
+Recogniser(File cf, File sf, EventManager em)
+recognise(Note note) : ResultSet
+recogniseSerial(Note note) : ResultSet
+recognise(GestureSample sample) : ResultSet
```

Figure 3.6: Recogniser class diagram

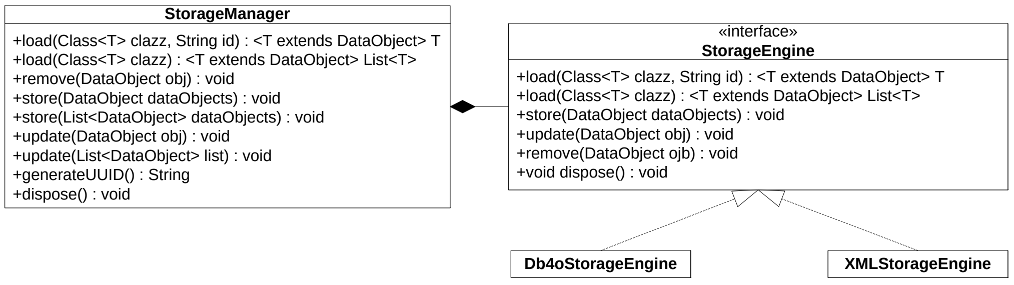
algorithms. The `recognise(Note note)` method goes through the algorithms in sequential order and stops the recognition process as soon as one algorithm returns a valid result whereas the `recognise(Note note, boolean recogniseAll)` method combines the results returned by different algorithms.

For instance, there is an application which recognises handwritten text and reacts on command gestures. For both tasks a different recognition algorithm is needed. These two algorithms are executed in serial order where the first recognises commands and the second recognises handwritten text. We draw the sign to delete a text block. So the first algorithm will recognise the gesture and the recogniser stops. Afterwards a note representing a word is processed. The first algorithm fails to recognise the input and so the second algorithm recognises it as handwritten text.

The problem of the second method is the order of the result set. The results have an accuracy value but this value is computed by the algorithm and it cannot be assumed that the accuracy of different algorithms has the same meaning. Therefore, the result with the highest accuracy value does not have to be the best result.

3.6 Persistence Mechanism

The `StorageManager` encapsulates the access to persistent data objects and uses a realisation of the `StorageEngine` interface to interact with the data source. As shown in Figure 3.7, the `StorageEngine` interface requests the four basic functions to create, read, update and delete data objects (CRUD).



```
StorageManager
+load(Class<T> clazz, String id) : <T extends DataObject> T
+load(Class<T> clazz) : <T extends DataObject> List<T>
+remove(DataObject obj) : void
+store(DataObject dataObjects) : void
+store(List<DataObject> dataObjects) : void
+update(DataObject obj) : void
+update(List<DataObject> list) : void
+generateUUID() : String
+dispose() : void
```

```
«interface»
StorageEngine
+load(Class<T> clazz, String id) : <T extends DataObject> T
+load(Class<T> clazz) : <T extends DataObject> List<T>
+store(DataObject dataObjects) : void
+update(DataObject obj) : void
+remove(DataObject ojb) : void
+void dispose() : void
```

Figure 3.7: Storage Manager UML diagram

In contrast, the storage manager provides an extended version of the CRUD methods and would also allow to cache data to prevent continuous updates while using data sources which cannot store data on object level.

Persistent objects have to implement the `DataObject` interface shown in Figure 3.8. It demands for an universally unique identifier which is used for the identification of persistent objects as for example in XML documents. Not all persistent storage engines need such an identifier but anyway it simplifies the use of different storage engines.

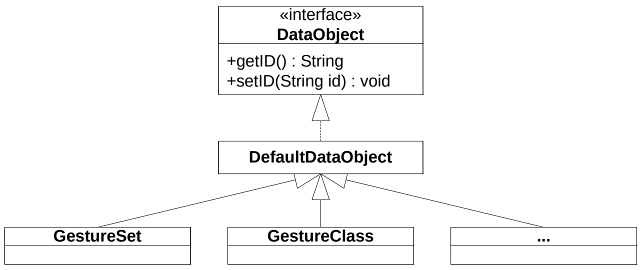


Figure 3.8: DataObject UML diagram

We decided to use *db4objects* [2] as the primary storage container which is an object oriented database and among others available under the GPL license. The class `db4oStorageEngine` implements the `StorageEngine` interface and realises the access to db4objects.

The second storage engine is implemented in the class `XMLStorageEngine` and serialises the data objects into an XML document. For the serialisation process we use the open source library x-stream [10] which is able to serialise/deserialise arbitrary Java objects and therefore enormously simplifies the implementation.

To support the transformation of different persistent data sources we can instantiate two instances of the storage manager with different storage engines and copy data from one to the other data source.

3.7 Management Console

The management console of iGesture is a Java Swing application and basically consists of three parts to define gestures, test them manually and a tool to create test sets. The application is implemented using the MVC pattern [12] and is extendable for additional functionality.

The backbone is the model class `GestureMainModel`. It provides access to data objects and uses the `StorageManager` to make them persistence. Beside that, the main model provides a set of listeners where views can register to be informed about changed data. Also other component models are based on the main model and so it is the only instance holding data and all alterations are propagated to it. Furthermore, the main model provides access to registered input devices.

The main model is initialised with a `StorageEngine` and a list of input devices. The fact that input devices are passed to the constructor allows the use of different input devices.

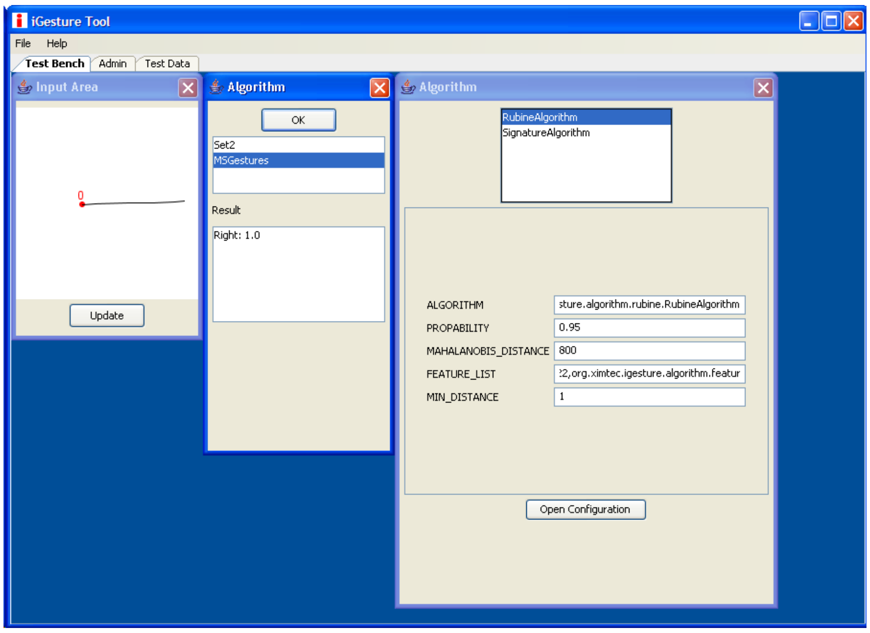
The most recent note that has been captured by the input device is stored in the main model and a listener informs registered views when it is altered. To prevent the destruction of the note by any views, the getter method always returns a clone of the original note. To pass the original reference would lead to non deterministic behaviour because some operations on the note change the note itself.

The management tool is structured into different tabs. Each tab contains functionality for a specific task and different tabs do not have any dependencies. This design decision has the advantage that new tabs can be added in a simple manner. The tabs are not hard coded in the main view and have to extend the abstract class `GestureTab`. The list of tabs is passed to the constructor of the main view and it creates the instances of the tabs using dynamic class loading and shows them in the order given by the list. This allows adding new tabs without a recompilation of the main view. The list of tabs is stored in a global property file.

A tab has a `JDesktopPane` as basis and all other components are attached to it. The existing implementations use `JInternalFrames` as main components on the base pane. The `GuiFactory` furthermore provides static methods to simplify the creation and reuse of components.

3.7.1 Test Bench Tab

The *Test Bench Tab* shown in Figure 3.9 provides functionality to acquire a gesture from the input device and recognise it with the gesture set and algorithm of the user's choice. This enables a simple and quick test of specific gestures. All gesture sets of the currently opened database are available and the algorithms can be selected. The available algorithms have to be set in the global property file in a comma delimited list. For the recognition process default parameters are used.



Set2
MSGestures

Result

Right: 1.0

RubineAlgorithm
SignatureAlgorithm

ALGORITHM	sture.algorithm.rubine.RubineAlgorithm
PROPABILITY	0.95
MAHALANOBIS_DISTANCE	800
FEATURE_LIST	:2,org.ximtec.igesture.algorithm.featur
MIN_DISTANCE	1

Figure 3.9: Test bench tab screenshot

3.7.2 Admin Tab

The *Admin Tab* enables us to administrate gesture sets, classes and descriptors. This view as shown in Figure 3.10 has three main components which are always visible. These are a frame for capturing gestures, a list of all available classes and a tree structure to manage the sets.

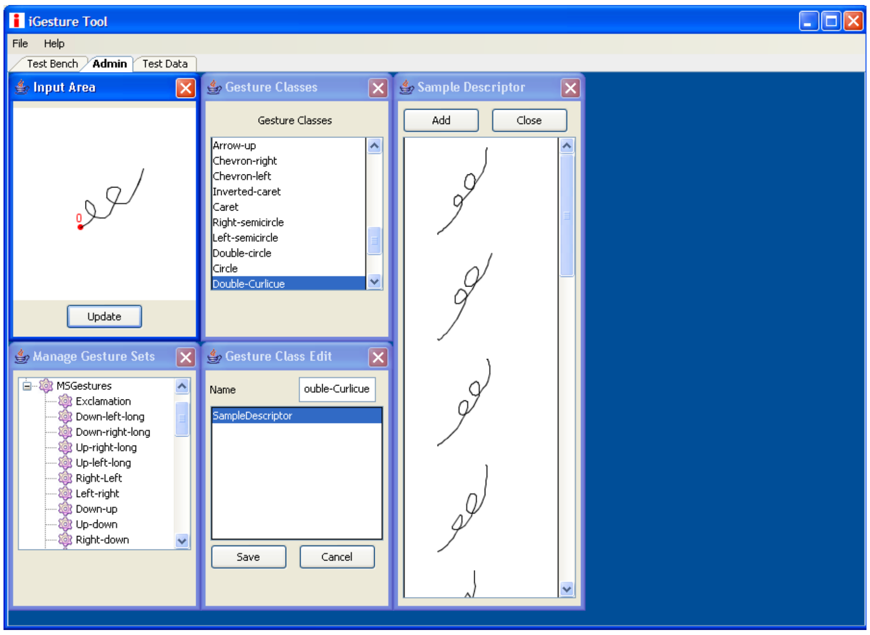


Figure 3.10: Screenshot admin tab

Gesture classes can be created, edited and deleted in the list. These operations are accessible over a context menu. For editing a gesture class, a new frame opens and lists the existent descriptors. Again the three basic functions create, edit and delete are applicable on the descriptors.

To manipulate a descriptor another frame opens. At the moment we only have an implementation for a sample-based descriptor providing functionality to add and delete gesture samples in a list. To add a sample, the gesture needs to be gathered with the capture component and it can be copied to the list of samples.

The tree representing the gesture sets and the containing classes has several context menus for the operations create, delete, export and import a gesture set and to add gesture classes. For adding classes to the set, another window pops up to select classes from the list.

The export and import functionality of set level allows creating and reading XML files containing the set and the corresponding classes with their descriptors. These files can be used for initialising the recogniser and makes it therefore independent of the storage manager.

3.7.3 Test Data Tab

The *Test Data Tab* shown in Figure 3.11 can be used to create test sets for testing algorithms and their configuration. It also has a capture component, a list of available test sets and a frame to select the gesture class the input belongs to.

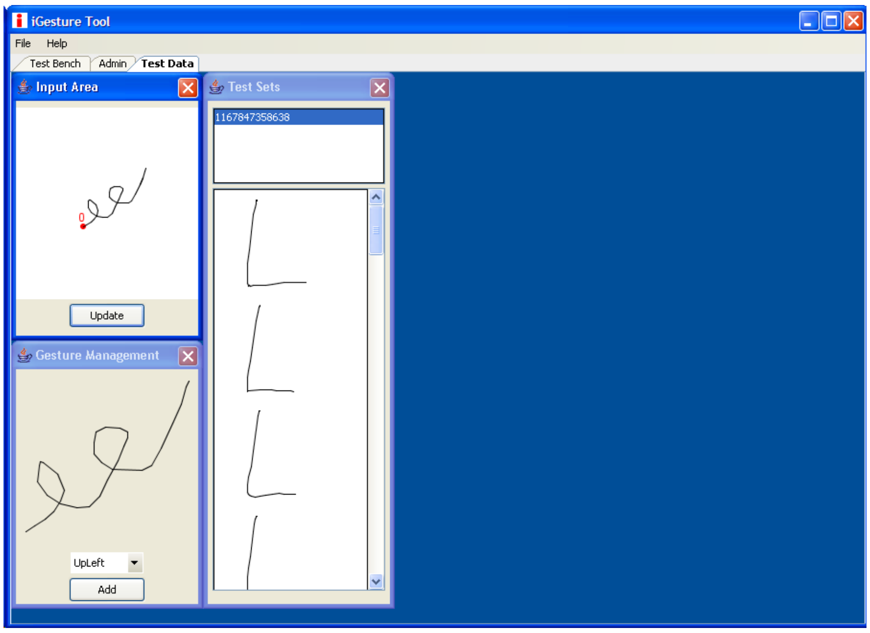


Figure 3.11: Screenshot test tab

Test sets can be created, edited and deleted. Furthermore, we provide functionality to export and import test sets from the corresponding XML representation.

3.7.4 Property File

The management console is configured with a Java property file. It contains all parameters of the application which can be set dynamically. Another resource file provides language dependent names and identifier used for the graphical user interface.

The global configuration file is named *igestureTool.properties* and contains the following parameter definitions:

- Path to the input device property file
- List of tabs to be shown
- Filename of the database
- List of available algorithms

All this parameters are mandatory to run the iGesture tool. An example of a configuration file is shown in Figure 3.12.

```
INPUT_DEVICE_CONFIG = magicommPen.properties

TABS = org.ximtec.igesture.tool.CaptureTab, \
       org.ximtec.igesture.tool.AdminTab, \
       org.ximtec.igesture.tool.TestTab

ALGORITHM = org.ximtec.igesture.algorithm.rubine.RubineAlgorithm, \
org.ximtec.igesture.algorithm.signature.SignatureAlgorithm, \
org.ximtec.igesture.algorithm.siger.SigerRecogniser

DATABASE = igesture.db
```

Figure 3.12: Property file igestureTool.properties

3.8 Batch Processing

iGestures provides a tool to test and evaluate different recognition algorithms. The idea behind this tool is to simplify the evaluation of newly developed algorithms and the comparison of different algorithms. Beside the comparison it allows us to test a specific algorithm with different parameters to adjust them in the best possible way for a given gesture set. Figure 3.13 shows the main classes belonging to the batch process handling.

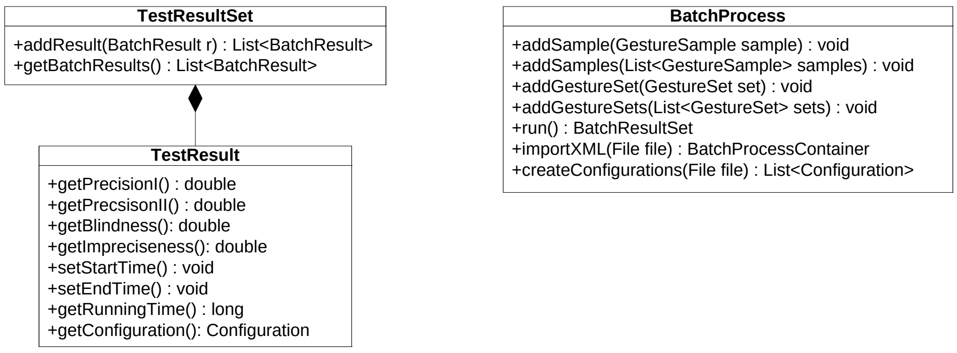

TestResultSet
+addResult(BatchResult r) : List<BatchResult>
+getBatchResults() : List<BatchResult>
TestResult
+getPrecisionI() : double
+getPrecsisonII() : double
+getBlindness(): double
+getImpreciseness(): double
+setStartTime() : void
+setEndTime() : void
+getRunningTime() : long
+getConfiguration(): Configuration
BatchProcess
+addSample(GestureSample sample) : void
+addSamples(List<GestureSample> samples) : void
+addGestureSet(GestureSet set) : void
+addGestureSets(List<GestureSet> sets) : void
+run() : BatchResultSet
+importXML(File file) : BatchProcessContainer
+createConfigurations(File file) : List<Configuration>

Figure 3.13: BatchProcess class diagram

A gesture test set is encapsulated in a `TestSet` instance which consists of a set of `GestureSample` instances. Each instance contains the name of the gesture class the sample belongs to or the name is empty, if the gesture does not exist in the gesture set. This naming is necessary to check whether the input was recognised correctly.

3.8.1 Test Configuration

A batch process is configured with an XML file and out of it the different configuration objects are created. To be able to test different parameter adjustments we provide a mechanism to define various parameters.

As usual it is possible to set a parameter to a fixed value. This is realised with the following XML construct:

```
<parameter name="MIN_ACCURACCY">
  <value>0.85</value>
</parameter>
```

Then we can define a sequence of values a parameter can have. This is done as follows:

```
<parameter name="RESULTSET_SIZE">
  <sequence>
    <value>1</value>
    <value>8</value>
  </sequence>
</parameter>
```

The next construct acts like a for-loop. Each value in the given sequence is taken as a parameter. All three arguments are processed as double values.

```
<parameter name="MIN_DISTANCE">
  <for start="1" end="3.5" step="0.5"/>
</parameter>
```

The last construct creates the power-set of the specified length out of a comma delimited list. The two parameters *min* and *max* denotes the minimal and maximal length of the created power-set. The power-set of a list has $2^{List.length}$ elements in general.

```
<parameter name="FEATURE_LIST">
    <powerset min="4" max="6">F1,F2,F3,F4,F5,F6</powerset>
</parameter>
```

Out of this XML configuration file all possible parameter permutations are collected. For each configuration the batch process instantiates the algorithm and processes the test set. The results of the batch process are collected in a `TestSetResult` data structure. For each configuration the key figures are computed and collected.

Note that this batch process may be very memory and time consuming.

3.9 Name Value Mapping

To realise a mapping between gesture classes and an integer as identifier we provide a helper class named `Mapping`. The idea of this mapping is to be able to use switch statements to handle actions triggered by a gesture.

The mapping between the name and the value is managed in an XML file which uses the schema shown in Appendix B.3.

`Mapping` is a static class to be able to access the identifiers directly. Although the class is static, it has to be initialised with the filename of the mapping. If the class is not initialised, a runtime exception is thrown.

3.10 org.sigtec.ink.Note

iGesture uses the `org.sigtec.ink.Note` class to represent gestures captured from an input device. A `Note` consists of `Traces` defined by time stamped `Points`. Beside the coordinates and the timestamp a point can contain additional information such as the force, rotation, pitch and yaw. The schema of the XML note representation is shown in Listing B.1.

3.11 Dependencies

The libraries listed in the following table are used by iGesture.

Name	**Filename**	**License**
Bloat	bloat-1.0.jar	GNU
db4objects	db4o-5.5-java5.jar	GPL
db4objects	db4o-5.5-nqopt.jar	GPL
iPaper	ipaper.jar	ETH
iText	itext-1.4.8.jar	MPL
Jakarta Commons CLI	commons-cli-1.0.jar	Apache 2.0
Jakarta Commons Collections	commons-collections-3.1.jar	Apache 2.0
Jakarta Mathematics Library	commons-math-1.1.jar	Apache 2.0
Jdom	jdom.jar	Apache Style
Jdom Contributions	jdom-contrib.jar	Apache Style
NativeCall	NativeCall.dll	BSD
NativeCall	nativecall-0.4.1.jar	BSD
NativeCall	nativeloader-200505172341.jar	BSD
OfficeLnFs	officeLnFs_1.1.2.jar	BSD
sigtec	sigtec.jar	ETH
Spline	spline.jar	LGPL
Universal Java XPath Engine	jaxen-1.1-beta-11.jar	Apache Style
UUID Specification	jug-asl-2.0.0.jar	Apache 2.0
Xalan	xalan.jar	Apache 2.0
xstream	xstream-1.2.jar	BSD

4 Algorithm

4.1 Rubine Algorithm

The Rubine algorithm introduced in Section 2.1 was developed in 1991 and represented one of the first algorithms published for the recognition of mouse and pen-based gestures. An important feature of this algorithm is that gestures are not described programmatically but learnt by examples. With the appropriate tools, such as the management console of iGesture, it becomes a simple and quick task to create new gestures and add them to the gesture recognition engine.

Features are extracted from the gestures consisting of time stamped points and used in the recognition process. The classification itself does not depend on specific features which allows us to use it for different recognition tasks as long as it is possible to describe the classifiable objects by feature vectors.

In a first step for each example gesture e the feature vector f is computed based on the selected features F. These vectors are then summarised in a mean vector per gesture class $\hat{c}$. This mean vector $\bar{f}$ is simply the average of the classes' example feature vectors as illustrated in Equation 4.1.

$$\bar{f}_{\hat{c}i} = \frac{1}{E_{\hat{c}}} \sum_{e=0}^{E_{\hat{c}}-1} f_{\hat{c}ei} \tag{4.1}$$

On the basis of these vectors the covariance matrix $M_{\hat{c}}$ shown in Equation 4.2 is computed for each gesture class and these covariance matrices are averaged to a single covariance matrix M shown in Equation 4.3. With this single covariance matrix it is possible to estimate the weights $\omega_{\hat{c}j}$ of the vector components shown in Equation 4.4 and the initial weight $\omega_{\hat{c}0}$ for each gesture class as illustrated in Equation 4.5.

$$M_{\hat{c}ij} = \sum_{e=0}^{E_{\hat{c}}-1} \left(f_{\hat{c}ei} - \bar{f}_{\hat{c}i}\right)\left(f_{\hat{c}ej} - \bar{f}_{\hat{c}j}\right) \tag{4.2}$$

$$M_{ij} = \frac{\sum_{c=0}^{C-1} \frac{M_{\hat{c}ij}}{E_{\hat{c}}-1}}{-C + \sum_{c=0}^{C-1} E_{\hat{c}}} \tag{4.3}$$

$$\omega_{\hat{c}j} = \sum_{i=1}^{F} \left(M^{-1}\right)_{ij} \bar{f}_{\hat{c}i} \tag{4.4}$$

$$\omega_{\hat{c}0} = -\frac{1}{2}\sum_{i=1}^{F} \omega_{\hat{c}i}\bar{f}_{\hat{c}i} \tag{4.5}$$

All these steps can be done during the initialisation phase of the algorithm and the weights computed per gesture class do not change during classifications.

The classification itself is realised with the linear function shown in Equation 4.6. For an input gesture the feature vector with the same features is computed and each component of this vector is multiplied with the corresponding weight of the gesture class. The gesture class which yields the maximal value denotes the classified gesture.

$$v_{\hat{c}} = \omega_{\hat{c}0} + \sum_{i=1}^{F} \omega_{\hat{c}i} f_i \tag{4.6}$$

This kind of classification has the problem that a result always will be returned even if the input gestures does not have any similarities with a trained example gesture. Therefore, mechanisms are required to reject gestures which are not similar to the trained ones or when several gestures classes have a similar probability to be selected and therefore the result is ambiguous.

To check whether the result is non-ambiguous the classification sums of the gesture classes are compared with the classified gestures class illustrated in Equation 4.7. If several classes are near this maximum class, it is assumed that the result is ambiguous. The border for this decision can be set by a specific parameter.

$$\tilde{P}\left(\hat{i}|g\right) = \frac{1}{\sum_{j=0}^{C-1} e^{(v_j - v_i)}} \tag{4.7}$$

To detect outliers the Mahalanobis distance [19] introduced in Equation 4.8 is used.

$$\delta^2 = \sum_{j=1}^{F}\sum_{k=1}^{F} \left(M^{-1}\right)_{jk} \left(f_j - \bar{f}_{ij}\right)\left(f_k - \bar{f}_{ik}\right) \tag{4.8}$$

4.1.1 Implementation

We implemented this algorithm as described in Rubin's paper and the following parameters can be set with the configuration object.

Parameter name	Description
MIN_DISTANCE	The minimal distance denotes the minimal space between two succeeding points of a gesture. Succeeding points which are too close may have a negative influence for the computation of particular features and are not meaningful in the context of the entire gesture.
FEATURE_LIST	A feature has to implement the `Feature` interface. This parameter holds a list of comma separated full qualified class names of feature objects. These classes are instantiated during the initialisation of the algorithm using dynamic class loading.
MAHALANOBIS_DISTANCE	This parameter holds the maximal distance an input gesture can have so that it is not detected as outlier. Rubine proposed to use half of the squared number of features as value for this parameter, but in practise this value seems to be too small.
AMBIGUITY	This parameter is used to prevent ambiguous results. Rubine proposes to reject results which have a value lower than 0.95.

4.2 SiGeR Algorithm

The *SiGeR* (Simple Gesture Recogniser) algorithm was developed by Scott Swigart for the Microsoft Developer Network to illustrate the implementation of custom gesture recognisers for the Microsoft Tablet PC platform. The algorithm classifies gestures on the basis of regular expressions.

Gestures are described with the eight cardinal points `(N, NE, E, SE, S, SW, W and NW)` and some statistical information. Out of this description a regular expression is created. These regular expressions are applied to input gestures and if a class description matches the input string, the gesture class is recognised as result. Therefore the classification is binary and it is not possible to rate different results.

For example the gesture shown in Figure 4.1, starting at the red dot, can be described with the following character string: `E, N, W, S`. Out of these characters the regular expression `(E)+(N)+(W)+(S)+` can be created. Because hand drawn lines may not be always straight, the author proposed a more general form of the regular expression which also accepts neighbouring distances. So the extended regular expression for this example gesture would be `(NE|E|SE)+(NW|N|NE)+(SW|W|NW)+(SE|S|SW)+`.

The input gesture is transformed into a character string and the distance between two points is approximated with the directions corresponding to the cardinal points. Additionally, statistical information is extracted out of the input gesture. Therefore each direction is

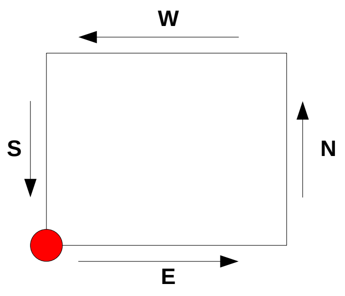


Figure 4.1: Example Gesture

counted and information about the proportion of the directions is provided. The proximity of the start and endpoints and the number of stop points is counted to enabling a more reliable description of gestures.

The description of gesture classes may set constraints concerning this statistical information. For example for a circle we can set constraints that there need to be equally many straight and diagonal elements and that the start- and endpoint of the gesture are close together.

Start- and endpoints of strokes may contain scar points which do not belong to the gesture itself and can be removed. For this reason the regular expression is extended allowing some points at the beginning and at the end not described in the gesture class description. During the recognition process gestures may also be flipped and mirrored to enable different drawings.

SiGeR was originally implemented in VB.NET and hosted as open source project at Sourceforge. For each gesture class an implementation of an interface is necessary to describe it programmatically.

4.2.1 Implementation

Our implementation differs in some points from the original SiGeR version. The most important change is the possibility to describe gesture classes in textual form using some keywords to describe the directions and to make use of the statistical information. This allows us to use the existing text descriptor for gesture classes.

Gestures are described using the following language:

```
Description = Directions [";" Constraints];
Directions = Direction ["," Directions];
Direction = "N"|"NE"|"E"|"SE"|"S"|"SW"|"W"|"NW";
Constraints = Constraint ["AND" Constraints];
Constraint = Operand Operator Operand;
Operator = "EQ"|"NEQ"|"GT"|"GTE"|"LT"|"LTE";
Operand = "N"|"NE"|"E"|"SE"|"S"|"SW"|"W"|"NW"|"DIAGONAL"|
"STRAIGHT"|"PROXIMITY"|"STOPPOINTS";
```

For example a rectangle would be described as `E,N,W,S;STRAIGHT GT 0.8 AND PROXIMITY LT 0.2`. The part before the semicolon describes the form of the rectangle with the directions. The two constraints state that at least 80% of the directions have to be straight and that the start and endpoint is in maximum 20% of the gesture diagonal distance away.

The following parameter can be set with the configuration object.

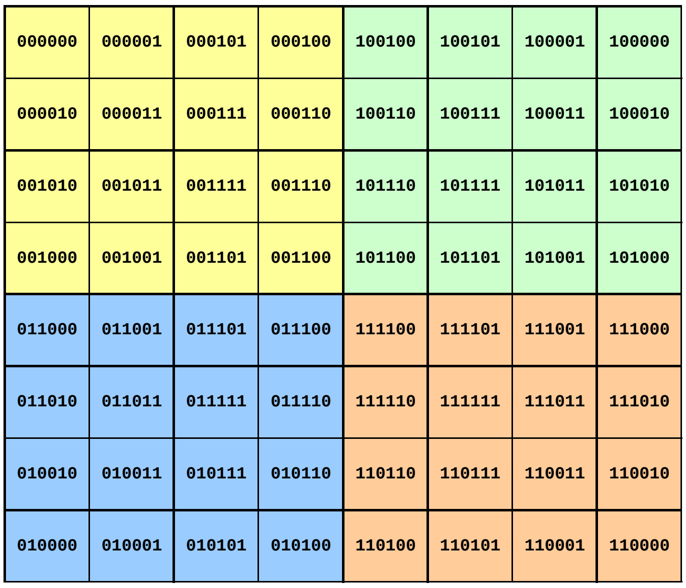
Parameter name	**Description**
MIN_DISTANCE	The minimal distance denotes the minimal space between two succeeding points describing a gesture.

4.3 Signature Algorithm

The *Signature* algorithm was designed as part of this diploma thesis. The idea behind this algorithm is to approximate gestures with a signature whereas the signature is computed from example gestures as done in the Rubine algorithm. The signatures are compared based on distance functions leading to a classification.

For the creation of the signature we use a grid which consists of squares of the same size. Each square is identified with a bit string and two neighbouring squares always differ in exactly one bit.

During the preprocessing phase the gestures are stretched to a uniform size and mapped onto the grid. Each point of the gesture can now be represented with the bit string of its related square as shown in Figure 4.2. The full signature consists of the concatenation of these bit strings.



000000	000001	000101	000100	100100	100101	100001	100000
000010	000011	000111	000110	100110	100111	100011	100010
001010	001011	001111	001110	101110	101111	101011	101010
001000	001001	001101	001100	101100	101101	101001	101000
011000	011001	011101	011100	111100	111101	111001	111000
011010	011011	011111	011110	111110	111111	111011	111010
010010	010011	010111	010110	110110	110111	110011	110010
010000	010001	010101	010100	110100	110101	110001	110000

Figure 4.2: Example Grid 8×8

While mapping the points onto this grid a fuzziness arises. This is furthermore increased in removing succeeding points which fall in the same square. Thereby, only significant points will be remaining.

Alternatively, this algorithm uses an mechanism to remove points which do not change the signature significantly. A point is denoted as irrelevant if the angle of the direction remains in a defined range.

We implemented two distance functions which can be selected by a parameter. Additional functions can be added by implementing the `DistanceFunction` interface.

Hamming Distance
The first function is the Hamming distance. It counts the number of bits which have to be flipped to make two signatures equal. The drawback of this function is that a displacement may trigger a lot of after-effects which may lead to bad results.

Levenshtein Distance
The Levenshtein distance is a generalisation of the Hamming distance. Beside the flipping of bits this measurement is also able to add or remove bits to achieve the smallest possible distance between two bit strings. Across this fact, after-effects can be minimised. The drawback of this function is that signatures might be changed drastically and so the result of the recognition is falsified.

For each example gesture the signature is created and an input gesture is compared with each of them. The accuracy of the result denotes the number of coinciding bytes.

The following parameters can be set with the configuration object.

Parameter name	**Description**
GRID_SIZE	The grid size defines the number of cells the grid should have within a line.
RASTER_SIZE	The raster size defines the width of and height a gesture is stretched to.
DISTANCE_FUNCTION	The full qualified class name of the distance function. The class has to implement the DistanceFunction interface.

4.4 Features

This section provides an overview about the implemented features describing gestures. The first part lists features proposed by Rubine which is followed by new features developed as part of this diploma thesis.

4.4.1 Rubine Features

The following 13 features are elaborated by Dean Rubine. This are the original features of his algorithm and all of them are implemented in our framework. Figure 4.3 illustrates some of this features.

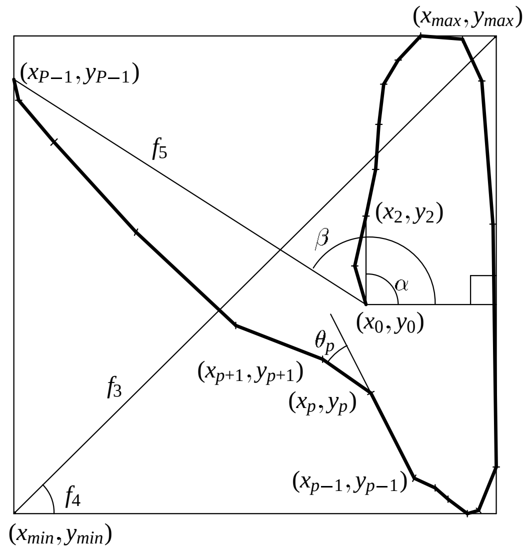


Figure 4.3: Rubine Features taken from [20]

Cosine and sine of the initial angle with respect to the X axis

These features use x_2 instead of x_1 because the larger distance between the two points would be more meaningful. Our implementation automatically takes the next possible point if x_2 is equal to x_0.

$$f_1 = \cos\alpha = \frac{(x_2 - x_0)}{\sqrt{(x_2 - x_0)^2 + (y_2 - y_0)^2}}$$

$$f_2 = \sin\alpha = \frac{(y_2 - y_0)}{\sqrt{(x_2 - x_0)^2 + (y_2 - y_0)^2}}$$

Length of the bounding box diagonal

$$f_3 = \sqrt{(x_{max} - x_{min})^2 + (y_{max} - y_{min})^2}$$

We think that this feature can be problematic because gestures do not always have the same size and the absolute size itself often does not characterise a gesture very well. This problem can be solved by scaling gestures to a specific size.

Angle of the bounding box

$$f_4 = \arctan\frac{y_{max} - y_{min}}{x_{max} - x_{min}}$$

Distance between first and last point

$$f_5 = \sqrt{(x_{N-1} - x_0)^2 + (y_{N-1} - y_0)^2}$$

We think that this feature could be problematic because of the same reason already mentioned for f_3.

Cosine and sine of angle between first and last point

$$f_6 = \cos\beta = \frac{x_{N-1} - x_0}{f5}$$

$$f_7 = \sin\beta = \frac{y_{N-1} - y_0}{f5}$$

Total gesture length

$$\Delta x_i = x_{i+1} - x_i$$

$$\Delta y_i = y_{i+1} - y_i$$

$$f_8 = \sum_{i=0}^{N-2} \sqrt{\Delta x_i^2 + \Delta y_i^2}$$

We think that this feature could be problematic because of the same reason already mentioned for f_3.

Total angle traversed

$$\Theta_i = \arctan \frac{\Delta x_i \Delta y_{i-1} - \Delta x_{i-1} \Delta y_i}{\Delta x_i \Delta x_{i-1} + \Delta y_i \Delta y_{i-1}}$$

$$f_9 = \sum_{i=1}^{N-2} \Theta_i$$

$$f_{10} = \sum_{i=1}^{N-2} |\Theta_i|$$

$$f_{11} = \sum_{i=1}^{N-2} \Theta_i^2$$

We observed that f_9 and f_{10} often yields to the same value. The difference between these two features only takes effect for gestures containing changes of directions. If features are to similar, the weights cannot be computed.

Maximum speed squared

$$f_{12} = \max_{i=0}^{N-2} \frac{\Delta x_i^2 + \Delta y_i^2}{t_{i+1} - t_0}$$

Stroke Duration

$$f_{13} = t_{N-1} - t_0$$

4.4.2 Single Stroke Features

The following new features were developed as part of this diploma thesis. They are an extension to Rubine's single stroke features.

Curve of strokes
This feature tests how straight a gesture is.

$$f_{14} = \frac{\sqrt{(x_0 - x_{N-1})^2 + (y_0 - y_{N-1})^2}}{f8}$$

Number of stop points
A stop point occurs when the drawing speed falls below a defined lower bound. The average speed for the whole gesture is computed and the lower bound is set to a third to it. This lower bound is compared with the velocity between two points. The number of stop points denotes the number sequences having a speed below the specified lower bound. Usually stop points appear on directional changes and at the end of strokes.

$$f_{15} = |StopPoints|$$

Proportion of the direction start-/middle-point to the diagonal
This value gives a clue if the first or second part of the gesture is more curvy. To determine the middle-point we use integer division.

$$f_{16} = \frac{\sqrt{\left(x_0 - x_{N/2}\right)^2 + \left(y_0 - y_{N/2}\right)^2}}{f3}$$

Sine of angle between first and second part of the stroke
This feature describes the change of direction with respect to the middle point of the gesture.

$$f_{17} = \frac{\left(y_{N/2} - y_0\right)}{\sqrt{\left(x_{N/2} - x_0\right)^2 + \left(y_{N/2} - y_0\right)^2}} - \frac{\left(y_N - y_{N/2}\right)}{\sqrt{\left(x_N - x_{N/2}\right)^2 + \left(y_N - y_{N/2}\right)^2}}$$

Sine of angle between the direction first- to middle-point with respect to the X axis

$$f_{18} = \frac{\left(y_{P/2} - y_0\right)}{\sqrt{\left(x_{P/2} - x_0\right)^2 + \left(y_{P/2} - y_0\right)^2}}$$

Cosine of angle between the direction middle- to end-point with respect to the X axis

$$f_{19} = \cos\alpha = \frac{\left(x_P - x_{P/2}\right)}{\sqrt{\left(x_P - x_{P/2}\right)^2 + \left(y_P - y_{P/2}\right)^2}}$$

Proportion Start/Endpoint to Diagonal of the bounding box This feature is a gesture size independent replacement for feature f_5.

$$f_{20} = \frac{f5}{f3}$$

4.4.3 Multi Stroke Features

To improve the recognition of multi-stroke gestures, we designed some features characterising them. These features are used in combination with a subset of Rubine's features.

Number of strokes
This feature denotes the number of strokes a gesture has. If most of the gestures contain the same number of strokes this feature may not be a good classifier.

$$f_{21} = |Strokes|$$

Sum of distances between strokes
This feature sums up the distances between strokes and informs about the distribution of them. In order that this feature is size independent the sum is divided by the diagonal of the bounding box.

$$f_{22} = \frac{1}{f_3}\sum_{i=1}^{S-1}\sqrt{\left(S_{i-1x_{P-1}} - S_{ix_0}\right)^2 + \left(S_{i-1y_{P-1}} - S_{iy_0}\right)^2}$$

Angle between strokes
This feature sums up the angles between the strokes. Therefore, the angle for each stroke is computed with respect to the X axis and subtracted from the previous one. To compute the angle the stroke is approximated as a direct line from the first to the last point.

$$f_{23} = \sum_{i=0}^{S-2} Angle\ between\ Stroke\ S_i\ and\ S_{i+1}$$

Proportion of stroke length
This feature informs about the proportion of the different strokes.

$$S_{i\ Length} = \sqrt{\left(S_{ix_0} - S_{ix_{P-1}}\right)^2 + \left(S_{iy_0} - S_{iy_{P-1}}\right)^2}$$

$$f_{24} = S_{0\ Length}\prod_{i=1}^{S-1}\frac{1}{S_{i\ Length}}$$

5 User Guide

5.1 Tool

This chapter gives an overview of the graphical iGesture tool shown in Figure 5.1.

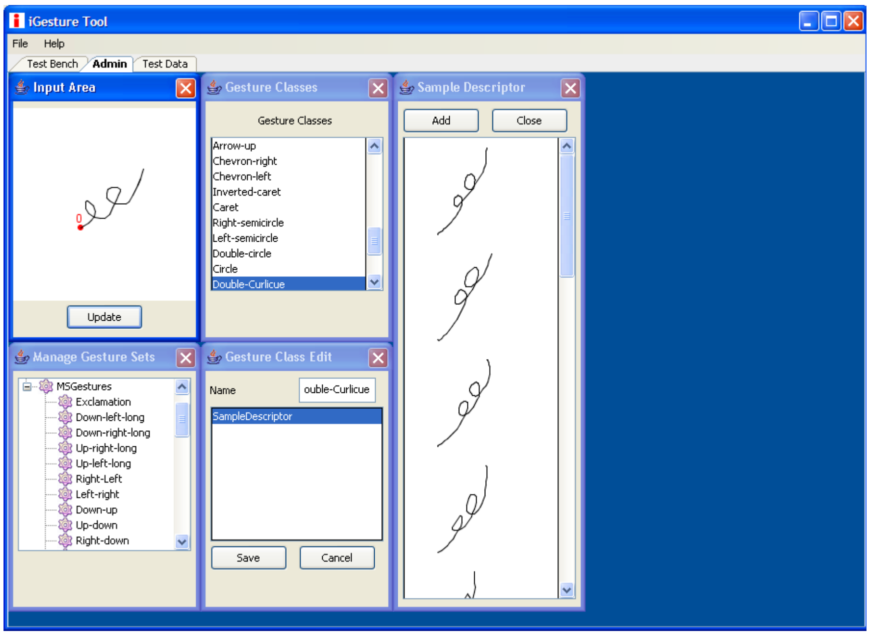


Figure 5.1: Screenshot iGesture tool

5.1.1 Capturing Gestures

On each tab there is a capture area as shown in Figure 5.2. It displays the gesture that has been been captured most recently by a specific input device. This gesture input can be used for various tasks as described in the following sections. Note that a click on the update button updates the gesture.

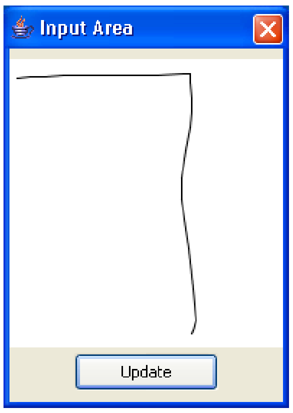


Figure 5.2: Capturing a gesture

5.1.2 Managing Gesture Sets and Classes

The *Admin Tab* provides functionality to create and manage gesture classes and sets. Classes and the related descriptors can be created and summarised to gesture sets.

Create and Edit Gesture Classes

A new gesture class can be created from the context menu which is reachable with a right mouse click in the *Gesture Classes* frame as shown in Figure 5.3.

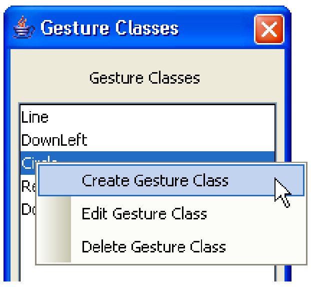


Figure 5.3: Create a new gesture class

After selecting *Create new Gesture Class* the new *Gesture Class Edit* frame appears and the gesture class can be named. The list below the name shows the available descriptors of this class. Right after the creation of a new gesture class no descriptor is visible. A mouse click in the list with the right button shows the context menu as outlined in Figure 5.4. The add button allows to create sample and text descriptors. By the same menu descriptors can also be edited and deleted.

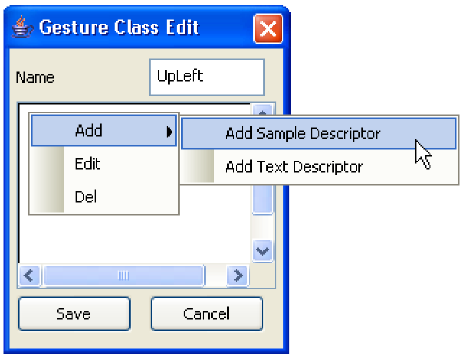
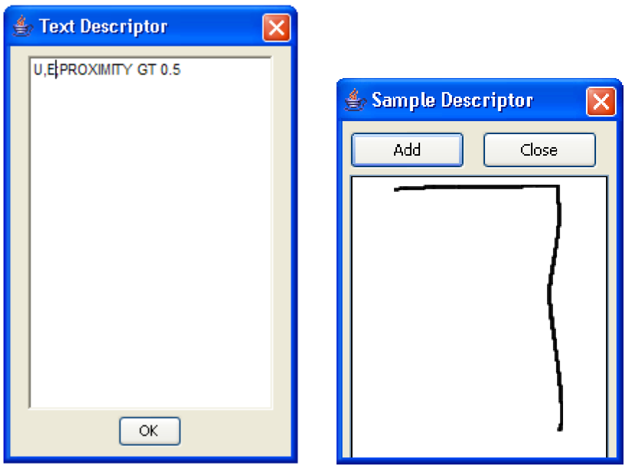


Figure 5.4: Create a new descriptor

Figure 5.5 shows the two available descriptor frames. The *Text Descriptor* contains a text field where the description can be made. A click on the `OK` button stores the description and closes the frame.

In the *Sample Descriptor* frame the gesture currently shown in the capturing area is added as new sample with a mouse click on the `Add` button. Samples can be deleted by using the context menu of the list showing the samples.



U,E|PROXIMITY GT 0.5

(a) Text descriptor (b) Sample descriptor

Figure 5.5: Descriptors

Create and Edit Gesture Sets

Gesture sets are created and managed in the *Gesture Sets* frame as illustrated in Figure 5.6. The sets and the related classes are shown in a tree structure. Each level of the three has a context menu for specific operations. A click on the root node allows the creation of new gesture sets. Alternatively, existing sets can be imported from an XML file.

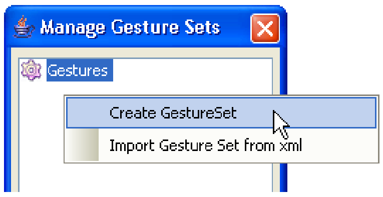


Figure 5.6: Create a new gesture set

Figure 5.7 shows the window that appears when a new set is created. In the text field the set can be named and it is added after a click on the `Create` button.

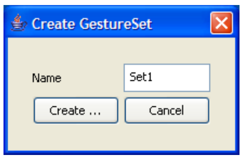


Figure 5.7: Set name of the gesture set

The context menu of the gesture set level shown in Figure 5.8 allows to add classes, delete the set, export the set as XML file or PDF document and create a test set out of the gesture set.

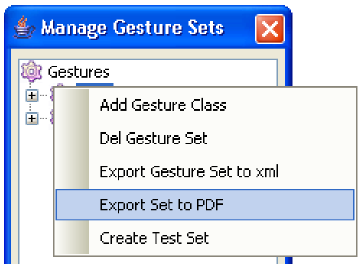


Figure 5.8: Add a gesture class to the set

The window shown in Figure 5.9 allows to add gesture classes to the selected gesture set. Multi-select is also possible and the marked classes are added after a click on the Add button.

Figure 5.10 shows the context menu of the gesture class level. It allows to remove selected classes from the gesture set.

5.1.3 Test Bench

The *Test Bench Tab* allows the manual test and creation of algorithm configurations. Figure 5.11 shows the frame with the available configurations and a list of parameters which can be edited.

Configurations can be created, deleted and edited using the context menu shown in Figure 5.12. Another feature is the possibility to export a configuration to an XML file whereas this file can be used for the initialization of the recogniser.

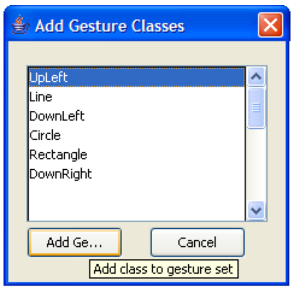


Figure 5.9: Select gesture classes

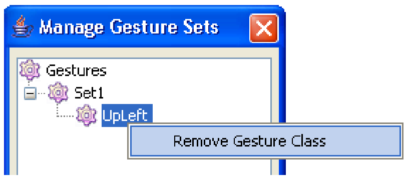
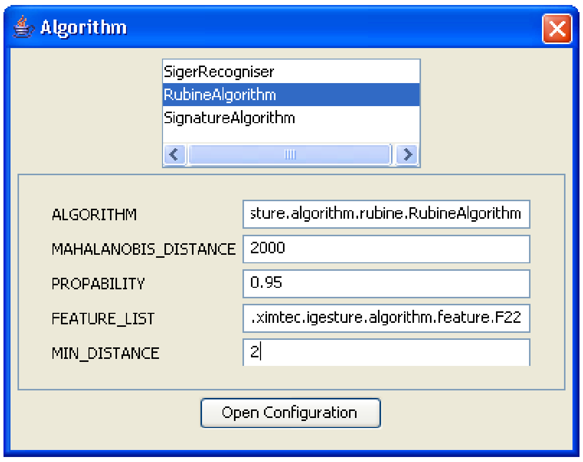


Figure 5.10: Remove a gesture class



ALGORITHM	sture.algorithm.rubine.RubineAlgorithm
MAHALANOBIS_DISTANCE	2000
PROPABILITY	0.95
FEATURE_LIST	.ximtec.igesture.algorithm.feature.F22
MIN_DISTANCE	2

Figure 5.11: Algorithm configuration

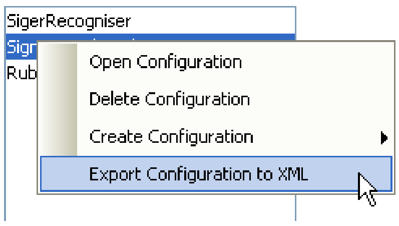
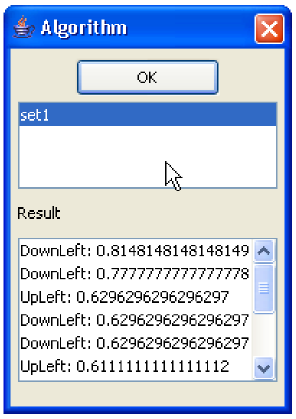


Figure 5.12: Manage configurations

The *Algorithm* frame shown in Figure 5.13 can test the recognition of a gesture shown in the capture area. The configuration of the algorithm to be used for the recognition needs to be open and a gesture set has to be selected. Afterwards, the recognition process can be started and the result list is shown.



```
DownLeft: 0.8148148148148149
DownLeft: 0.7777777777777778
UpLeft: 0.6296296296296297
DownLeft: 0.6296296296296297
DownLeft: 0.6296296296296297
UpLeft: 0.6111111111111112
```

Figure 5.13: Result of the recognition

5.1.4 Test Data

With the *Test Data Tab* test sets can be created. Figure 5.14 shows the frame for selecting the name of the captured gesture. Additionally to the available class names there is an entry called `None`. This option is used to declare test gestures which are not part of the gesture set and should be rejected by the algorithm. A click on the `Add` button copies the gesture to the opened test set.

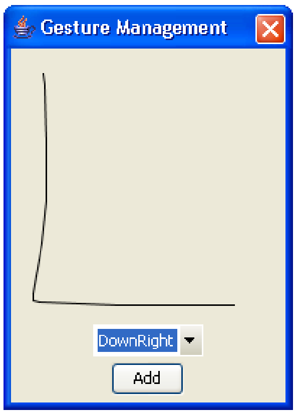


Figure 5.14: Name test gestures

The upper part of the *Testset List* shown in Figure 5.15 lists the available test set. Sets can be created, deleted, edited and also imported and exported to an XML file. The list below display the content of the selected test set. Using the context menu of this list enables the deletion of particular test gestures.

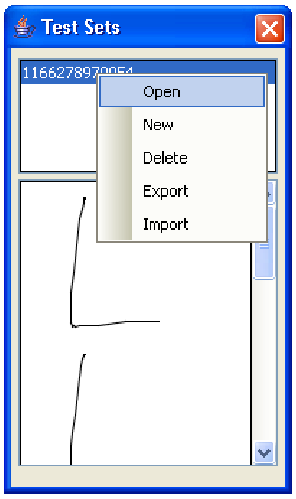


Figure 5.15: List of test sets

5.2 Framework

The use of the framework itself is demonstrated by a simple demo application. This application knows three gestures, namely LeftRight, DownRight and UpLeft as shown in Figure 5.16. In the first part all configuration issues are done programmatically. In a second step the use of XML files is described.

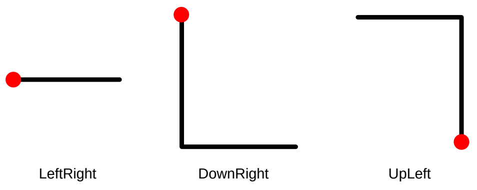


Figure 5.16: Example Gestures

This example uses the SiGeR algorithm which needs a text descriptor for each gesture class. The Listing 5.1 shows how these gestures are created and grouped in a gesture set.

Listing 5.1: Creation of gesture classes

```
GestureClass leftRightLine = new GestureClass("LeftRight");
leftRightLine.addDescriptor(new TextDescriptor("E"));

GestureClass downRight = new GestureClass("DownRight");
downRight.addDescriptor(new TextDescriptor("S,E"));

GestureClass upLeft = new GestureClass("UpLeft");
upLeft.addDescriptor(new TextDescriptor("N,W"));

GestureSet gestureSet = new GestureSet("GestureSet");
gestureSet.addGestureClass(leftRightLine);
gestureSet.addGestureClass(upLeft);
gestureSet.addGestureClass(downRight);
```

In the next step, the `Configuration` object is created. The gesture set created before is added to the configuration and the SiGeR algorithm is set as shown in Listing 5.2. With this configuration the `Recogniser` can be instantiated.

Listing 5.2: Creation of gesture classes

```java
Configuration configuration = new Configuration();
configuration.addGestureSet(gestureSet);
configuration.addAlgorithm(SigerRecogniser.class.getName());
recogniser = new Recogniser(configuration);
```

To capture the input of an appropriate device the `InputDeviceClient` is used. A list of devices has to be instantiated and the `MouseReader` is added. This allows drawing the gesture with the mouse while pressing the middle mouse button. After releasing the button, the gesture is recognised. To be able to react on this event, this example class has to implement the `ButtonDeviceEventListener` interface. These steps are shown in Listing 5.3.

Listing 5.3: Creation of gesture classes

```java
List<InputDevice> devices = new ArrayList<InputDevice>();
devices.add(new MouseReader());
client = new InputDeviceClient(devices);
client.addButtonDeviceEventListener(this);
```

The method shown in Listing 5.4 is executed after releasing the mouse button. The `Note` is created with the data stored in the buffer of the `InputDeviceClient` which is cleared afterwards. With this Note the recognise method is called and the result is stored in the `ResultSet`. Depending on the result, the name of the classified gesture or 'recognition failed' is printed on the console.

Listing 5.4: Creation of gesture classes

```java
public void handleMouseUpEvent(InputDeviceEvent event) {
    ResultSet result = recogniser.recognise(client.createNote(0, event.getTimestamp
         (), 70));
    client.clearBuffer();
    if(result.isEmpty()){
        System.out.println("recognition failed");
    }else{
        System.out.println(result.getResult().getName());
    }
}
```

Alternatively to defining the gesture set and classes programmatically it can be done in an XML file as illustrated in Listing 5.5. This has the advantage that gestures can be defined independently of the source code and the instantiation of an algorithm is much shorter.

Listing 5.5: XML Configuration

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
  <algorithm name="org.ximtec.igesture.algorithm.siger.SigerRecogniser" />
  <set name="gestureSet1" id="1">
    <class name="LeftRight" id="2">
      <textDescriptor><text>E</text></textDescriptor>
    </class>
    <class name="DownRight" id="3">
      <textDescriptor><text>S,E</text></textDescriptor>
    </class>
    <class name="UpLeft" id="4">
      <textDescriptor><text>N,W</text></textDescriptor>
    </class>
  </set>
</configuration>
```

If the configuration is done with an XML file, Listing 5.6 replaces Listing 5.1 and 5.2. Note that semantically the two declaration are identical.

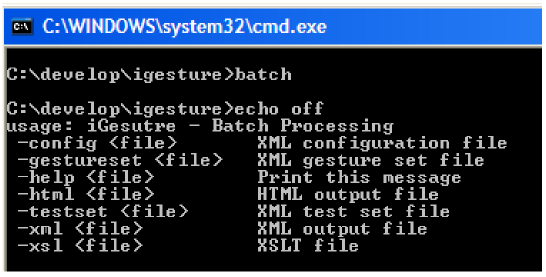
Listing 5.6: Creation of gesture classes

```
recogniser = new Recogniser(ConfigurationTool.importXML(new File("config.xml")));
```

5.3 Batch Processing

Our batch processing front end has a command line interface. It is started with the batch.bat file located in the root directory of iGesture. Starting the program without any parameters will show a help screen as outlined in Figure fig:batchhelp. The following table explains the possible parameters.

config	Denotes the path to an XML file containing the configuration of the batch process. The syntax of this configuration file is described in the implementation section and an example is given in Listing 5.7.
gestureset	Denotes the path to an XML file containing a gesture set. This file can be created using the graphical iGesture tool. This is the gesture set the algorithm specified in the configuration file will work on.
testset	Denotes the path to an XML file containing a test set. Again this file can be created in the graphical iGesture tool and holds valid as also invalid gesture samples. This set is used to measure the quality of an algorithm.
xml	Denotes the path to the output XML file. It contains the results of the batch process in a raw format.
xsl	Denotes the path to an XSLT file. It is used to render an HTML page out of the raw data. An example XSLT file is located in the XML resource folder of iGesture.
html	Denotes the path to the output HTML document which is rendered with the XSLT file.



```
C:\develop\igesture>batch

C:\develop\igesture>echo off
usage: iGesutre - Batch Processing
 -config <file>         XML configuration file
 -gestureset <file>     XML gesture set file
 -help <file>           Print this message
 -html <file>           HTML output file
 -testset <file>        XML test set file
 -xml <file>            XML output file
 -xsl <file>            XSLT file
```

Figure 5.17: Batch processing help screen

Listing 5.7: XML Configuration

```xml
<?xml version="1.0" encoding="UTF-8"?>
<iGestureBatch>
  <algorithm name="org.ximtec.igesture.algorithm.signature.SignatureAlgorithm">
      <parameter name="GRID_SIZE">
          <for start="8" end="16" step="2" />
    </parameter>
    <parameter name="RASTER_SIZE">
      <for start="120" end="240" step="10" />
    </parameter>
    <parameter name="DISTANCE_FUNCTION">
      <sequence>
        <value>org.ximtec.igesture.algorithm.signature.HammingDistance</value>
        <value>org.ximtec.igesture.algorithm.signature.LevenshteinDistance</value>
      </sequence>
    </parameter>
    <parameter name="MIN_DISTANCE">
      <for start="1" end="5" step="1" />
    </parameter>
  </algorithm>
</iGestureBatch>
```

6 Evaluation

This section deals with the evaluation of the implemented algorithms using different configurations and gesture sets. It is mainly divided into three parts each one based on a different kind of gesture sets.

As described in Chapter 4 our framework currently supports three algorithms. However, in the following experiments we distinguish between the original algorithm presented by Rubine and the one which has been extended as part of this diploma thesis. Therefore, in total we evaluate four algorithms, three of them using samples as descriptors and one using textual definitions.

6.1 Key Figures

There are four different result categories how an input can be classified by an algorithm. First, the algorithm recognises the input correctly. Correctly means that the input is recognised as the gesture it actually represents. This category is named *Correct*.

The category *Error* concerns incorrectly recognised gestures. In this case the algorithm returns a wrong result which may lead to wrong actions triggered by the recognised gesture.

The third and fourth categories contain rejected gestures. A gesture can be rejected for good reasons because it is not member of the gesture set. This category is denoted *Reject Correct*. Gestures which are rejected although they are part of the gesture set are named *Reject Error*.

In the field of pattern recognition the key figures *Precision* and *Recall* are normaly used. The Precision illustrated in Equation 6.1 denotes the amount of correct results in the result set. The Recall shown in Equation 6.2 denotes how many of the correct results are returned at all.

$$Precision = \frac{Pattern\ Candidates\ \cap\ Control\ Set}{Pattern\ Candidates} \qquad (6.1)$$

$$Precision = \frac{Pattern\ Candidates\ \cap\ Control\ Set}{Control\ Set} \tag{6.2}$$

Transformed to our environment this gives the following key figures:

Precision
The *Precision* shown in Equation 6.3 denotes the proportion of correct results versus all results. A value of 1 means that all recognised gestures are identified correctly. In other words the result does not contain any errors.

$$Precision = \frac{||Correct||}{||Correct|| + ||Error||} \tag{6.3}$$

Recall
The *Recall* illustrated in Equation 6.4 terms the amount of correct recognised gestures versus the size of the test set without any noise. This means that the test set contains only gestures which should be recongised correcty.

$$Recall = \frac{||Correct||}{||Samples|| - ||Noise||} \tag{6.4}$$

F-Measure
The *F-Measure* shown in Equation 6.5 is the weighted harmonic mean of precision and recall. We decided to weight precision and recall equally. Note that this measure is used to rank different algorithms and their configurations.

$$F - Measure = \frac{2 * Precision * Recall}{Precision + Recall} \tag{6.5}$$

6.2 Palm Graffiti

The following experiments consider the 26 letters and 10 numbers of the Palm Graffiti alphabet as shown in Appendix A.1. They cannot be combined into a single gesture set because several gestures in the number set are similar to letters. Overall, we collected training and test data from four persons which allows a more objective estimation than with a single user only. The setup of each experiment is shortly described and the key figures of the best configuration are presented.

6.2.1 Experiment 1: Graffiti Numbers

Experiment number 1 uses the Graffiti numbers as gesture set trained with 15 examples for each gesture class by one person. The test set has a size of 150 valid samples and has been collected by three different persons. To test the SiGeR algorithm a textual description of the numbers was constructed manually.

Tables 6.1 and 6.2 show the key figures for the best configuration of each algorithm. The best configuration is the one with the maximal *F-Measure* which is a compromise of precision and recall. This means that the configuration should have a low error rate as well as a high number of correctly recognised gestures.

	Correct	Error	Reject Correct	Reject Error	Sample	Noise
Rubine Extended	147	3	0	0	150	0
Rubine Original	134	15	0	1	150	0
Signature	126	9	0	15	150	0
SiGeR	131	6	0	13	150	0

Table 6.1: Experiment 1: Absolute values

	Precision	Recall	F-Measure
Rubine Extended	0.980	0.980	0.980
Rubine Original	0.899	0.893	0.896
Signature	0.933	0.840	0.884
SiGeR	0.956	0.873	0.913

Table 6.2: Experiment 1: Key figures

Although the example-based algorithms are trained by only one person, nearly all algorithms reached a *Precision* of at least 90%. The result shows that our new features used in the extended version of the Rubine algorithm significantly improve the recognition quality. Even the simple Signature algorithm has a higher Precision than the original implementation of the Rubine algorithm.

Surprisingly, also the SiGeR algorithm has good test results. However, note that the textual description of the gesture classes were done on the basis of this test set and therefore they are optimised for this specific experiment.

6.2.2 Experiment 2: Graffiti Numbers

As in experiment number 1 we use the Graffiti numbers as gesture set. But this time the training data was collected by four different persons. We trained each gesture class with 4 times 4 examples and the test set has a size of 140 samples which are collected from single person. We compared only sample-based algorithms and so the SiGeR algorithm is not taken into account anymore. Again the test set does contain only valid samples which should all be recognised.

	Correct	Error	Reject Correct	Reject Error	Sample	Noise
Rubine Extended	140	0	0	0	140	0
Rubine Original	135	4	0	1	140	0
Signature	120	13	0	7	140	0

Table 6.3: Experiment 2: Absolute values

The Rubine algorithms provide better key figures than in experiment number 1 as shown in Table 6.3 and 6.4. The extended Rubine algorithm achieved a perfect result whereas this time the Signature algorithm has a higher error rate than the original Rubine algorithm. It

	Precision	Recall	F-Measure
Rubine Extended	1.000	1.000	1.000
Rubine Original	0.971	0.964	0.968
Signature	0.902	0.857	0.879

Table 6.4: Experiment 2: Key figures

can be assumed that the Rubine algorithm works significantly better with a broader variety of training data.

6.2.3 Experiment 3: Graffiti Letters

This experiment uses the 26 Graffiti letters as gesture set. As in experiment number 1 the training data is collected from a single person and the algorithms are trained with 15 examples per gesture class. The test set has a size of 390 samples collected from three different persons. As in the experiments before the test set contains valid data only.

	Correct	Error	Reject Correct	Reject Error	Sample	Noise
Rubine Extended	334	52	0	4	390	0
Rubine Original	280	107	0	3	390	0
Signature	261	126	0	3	390	0

Table 6.5: Experiment 3: Absolute values

	Precision	Recall	F-Measure
Rubine Extended	0.865	0.856	0.861
Rubine Original	0.724	0.718	0.721
Signature	0.674	0.669	0.672

Table 6.6: Experiment 3: Key figures

The key figures shown in Tables 6.5 and 6.6 are significantly worse compared with experiment number 1. The reason is the size of the gesture set which is 2.5 times larger and that we have not increased the size of the training data. These results highlight that the number of samples has to grow with the size of the gesture set.

It attracts attention that several letters have a very high error rate whereas others are recognised perfectly. The reason for this behaviour is that the training data is collected by a single person. Better results are expected using training data from different users.

6.2.4 Experiment 4: Graffiti Letters

As in experiment number 2 each gesture class is trained with 4 times 4 examples from different users. The test set has a size of 363 samples and was produced by the same persons used for the training of the algorithm. Again the test set does only contain valid gestures.

Table 6.2.4 and 6.2.4 show the key figures which are distinctively higher than in experiment number 3 where the algorithms were trained by just one person. It is assumed that these values can be further increased by using more examples for each gesture class. In comparison to experiment number 3 there are no longer several gesture classes which are recognised badly.

	Correct	Error	Reject Correct	Reject Error	Sample	Noise
Rubine Extended	342	18	0	3	363	0
Rubine Original	305	48	0	10	363	0
Signature	253	32	0	78	363	0

Table 6.7: Experiment 4: Absolute values

	Precision	Recall	F-Measure
Rubine Extended	0.950	0.942	0.946
Rubine Original	0.864	0.840	0.852
Signature	0.888	0.697	0.781

Table 6.8: Experiment 4: Key figures

6.2.5 Experiment 5: Graffiti Numbers

Experiment number 5 uses the same gesture set as experiment number 2 but the test set is extended with Graffiti letters as noise which should be rejected by the recogniser. The test set has a total size of 470 samples and is collected from four persons. Letters which are similar to numbers are excluded from the test set because they would falsify the result. Therefore there are 22 different gesture classes used as noise.

The key figures shown in Table 6.9 and 6.10 are definitely worse than in experiment number 2. The reason for this behaviour is that gestures have to be rejected earlier to retain the error rate as low as possible. Even the extended Rubine algorithm has a precision lower than 92% which seems to be too low for practical use. We assume that these results can be improved by increasing the number of training data.

	Correct	Error	Reject Correct	Reject Error	Sample	Noise
Rubine Extended	114	11	319	26	470	330
Rubine Original	131	89	243	7	470	330
Signature	111	39	294	26	470	330

Table 6.9: Experiment 5: Absolute values

	Precision	Recall	F-Measure
Rubine Extended	0.912	0.814	0.860
Rubine Original	0.595	0.936	0.728
Signature	0.740	0.793	0.766

Table 6.10: Experiment 5: Key figures

6.3 Microsoft Application Gestures

This section uses the Microsoft Application Gestures shown in Appendix A.2 as gesture set. Originally this set consists of 42 gestures but two of them which represents mouse- or pen-clicks cannot be recognised with our feature based algorithm because the gesture consist of only one or two points. Therefore the set used in the evaluation has a size of 40 gestures.

6.3.1 Experiment 6: Microsoft Application Gestures

The 40 gestures shown in Appendix A.2 are trained with 15 examples for each gesture class by one person and it is tested with 5 instances of each gesture class provided by the same person. Again the test set contains valid gestures only.

Tables 6.11 and 6.12 highlight that the extended Rubine algorithm has a good performance although the number of used training data used is relatively small compared to the gesture set size.

	Correct	Error	Reject Correct	Reject Error	Sample	Noise
Rubine Extended	196	2	0	2	200	0
Rubine Original	178	19	0	3	200	0
Signature	145	32	0	23	200	0

Table 6.11: Experiment 6: Absolute values

	Precision	Recall	F-Measure
Rubine Extended	0.990	0.980	0.985
Rubine Original	0.904	0.890	0.897
Signature	0.819	0.725	0.769

Table 6.12: Experiment 6: Key figures

It is evident that this experiment is not realistic because the training data and the test data were produced by the same person. Anyway, it shows that with the extended Rubine algorithm good results can be achieved even with a small number of training data if the coverage is good.

6.4 Multi-Stroke Gestures

We defined a set of 15 multi-stroke gestures as shown in Appendix A.3. Two of them consist of a single stroke only and are prefixes of other multi-stroke gestures in the set. The set is quite small but it should be a proof of concept that also multi-stroke gestures can be recognised by the implemented algorithms.

6.4.1 Experiment 7: Multi-Stroke Gestures

In this experiment the algorithms are trained with 15 examples collected from one person and tested with 5 samples for each gesture class collected from the same person.

The results shown in Tables 6.13 and 6.14 are good. All algorithms have a precision higher than 96%. The reason for these results is that the algorithm is trained and tested by the same person. Furthermore, the relatively high number of training-examples for each gesture class has a positive effect.

	Correct	Error	Reject Correct	Reject Error	Sample	Noise
Rubine Extended	75	0	0	0	75	0
Rubine Original	72	3	0	0	75	0
Signature	72	1	0	2	75	0

Table 6.13: Experiment 7: Absolute values

	Precision	Recall	F-Measure
Rubine Extended	1.000	1.000	1.000
Rubine Original	0.960	0.960	0.960
Signature	0.986	0.960	0.973

Table 6.14: Experiment 7: Key figures

6.4.2 Experiment 8: Multi-Stroke Gestures

In this experiment the algorithms are trained with 20 examples collected from one person and tested with 5 samples for each gesture class collected from another person.

	Correct	Error	Reject Correct	Reject Error	Sample	Noise
Rubine Extended	74	1	0	0	75	0
Rubine Original	73	1	0	1	75	0
Signature	74	1	0	0	75	0

Table 6.15: Experiment 8: Absolute values

Surprisingly, all algorithms achieve nearly the same very good results as shown in Tables 6.15 and 6.16. The reasons for these good results are the high number of training samples compared to the size of the gesture set and probably also the kind of gestures we used. They

	Precision	Recall	F-Measure
Rubine Extended	0.987	0.987	0.987
Rubine Original	0.986	0.973	0.980
Signature	0.987	0.987	0.987

Table 6.16: Experiment 8: Key figures

are not similar to each other and therefore it is easier for the algorithms to classify them correctly.

6.5 Summary

The experiments have shown the behaviour of the chosen algorithms in different setups. Having a look at the results, only the extended Rubine algorithm seems to be good enough for practical usage. However, the other algorithms may have other strengths as for rapid prototyping.

Simple geometric figures in a small gesture set could be recognised as well with the SiGeR algorithm whiteout having to create a large amount of example data. This has the advantage that an application can be extended easily with gestures for case studies.

The Signature algorithm can also lead to good results if the gesture set is not too big and only a few gesture class examples are available. In this setting it can be even superior to the Rubine algorithm because this one needs several examples per gesture class to compute the weights. If the number of gesture examples is too small, it is actually possible that the Rubine algorithm cannot be instantiated because the determinant of the covariance matrix is zero and therefore the inverse cannot be computed.

In general the features defined as part of this diploma thesis helped to improve the Rubine algorithm's performance significantly.

7 Future Work

iGesture as framework and tool has reached a stable state and in a next step it has to be integrated in applications for evaluating it in practice and enhance it on the basis of these results. First simple sample applications are showing that the framework is easy to integrate and that the functionality is quite solid. The framework is designed to be extendable in different ways for adding functionality and fulfilling any practical needs and requirements.

The following section describes some extension which would be desirable but were not implemented as part of this diploma thesis.

Capturing of sample Gestures
The gathering of sample gestures which are used in example-based algorithm is a time consuming task using the iGesture tool. This task could be improved with an iPaper application for collecting the sample data. We could imagine to use a form containing a table. The first cell in each row shows the sample gestures. The other cells in the row can collect sample gestures. Preferably, such a form could be generated automatically by the iGesture tool.

Beside the reduced amount of time required for collecting the required data this would also enable us to gather sample gestures from untrained users resulting in a better coverage of the test data.

Algorithm
Up to now we have not implemented algorithms using techniques from the field of artificial intelligence. This technique is broadly used in speech and image-recognition and should also work reliable for gesture recognition.

Neuronal networks and hidden Markov models are mentioned in work dealing with pattern recognition. There already exist frameworks implementing these algorithms.

Interpreter
For the recognition of single gestures recognisers are appropriate. However, if we have more complex elements consisting of several gestures we need another abstraction level that can

handle such higher level objects. An interpreter could use the recognisers to detect gestures and assemble them to higher level objects.

In addition to the recognisers an interpreter also uses contextual information to achieve more reliable results and to get the semantic of a set of drawn gestures.

Extended Evaluation
The existing algorithms have been tested with a small number of users only. It would be interesting to know how good they perform with gestures captured from a larger group of users. Also a comparison with the Microsoft Tablet PC SDK regarding their application gesture set would be interesting. Therefore, a wrapper for the Microsoft recogniser would have to be implemented based on the Java Native Interface.

Test Suite
It would be more user friendly if the tool for testing algorithms in batch mode would be integrated into the graphical tool. This would also provide more functionality concerning the analysis of the results in comparison to the presentation of a simple HTML file. A wizard to create batch configurations would be another improvement.

8 Conclusion

The goal of this diploma thesis was the design and implementation of a general gesture recognition framework. With iGesture such a gesture recognition framework and the necessary tools to create gesture-based pen applications was developed. It supports various algorithms and can easily be extended. The tools enable the creation of gesture sets and provide functionality to test algorithms manually or in batch mode.

In addition to the framework we designed a new simple algorithm for the recognition process and realised significant improvements for the Rubine algorithm. The existing set of Rubine's single-stroke features was extended and we also designed some special features to classify multi-stroke gestures.

The evaluation in Chapter 6 proved that good recognition results can be achieved as long as the algorithms are trained with adequate training data and the gesture set itself is well designed. Furthermore, it is shown that our newly defined features increase the quality of the recognition process significantly.

A Gestures

This chapter shows images of the gesture sets used in Chapter 6.

A.1 Palm Graffiti

The images showing the Graffiti alphabet are taken from [6].

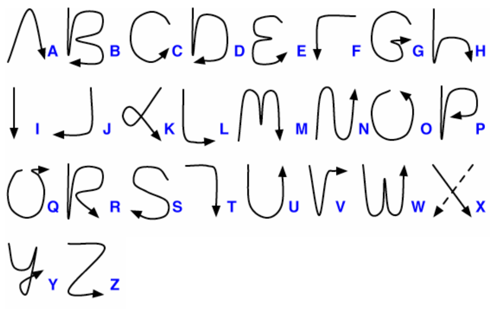


Figure A.1: Palm Graffiti letters

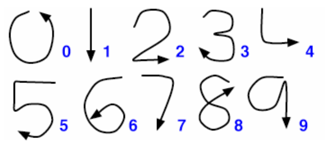
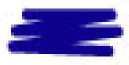
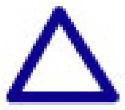
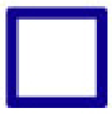
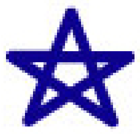
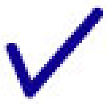
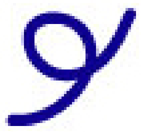
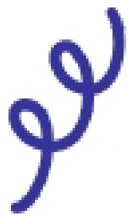
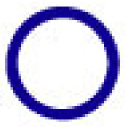
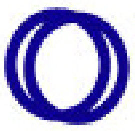
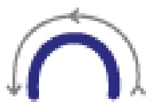
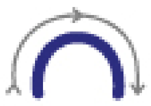
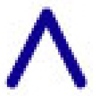
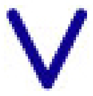


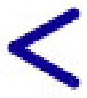
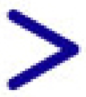
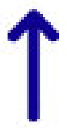
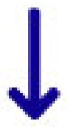
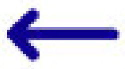
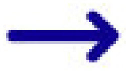
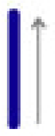
Figure A.2: Palm Graffiti numbers

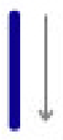
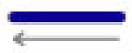
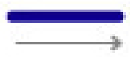
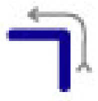
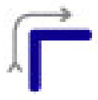
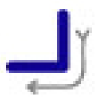
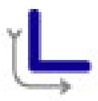
A.2 Microsoft Application Gestures

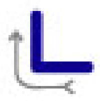
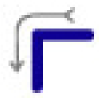
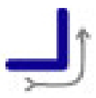
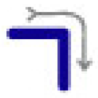
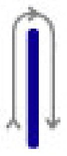
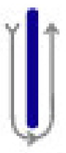
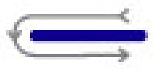
The images of the Microsoft Application Gestures are taken from [4].

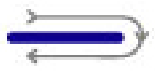
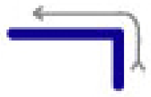
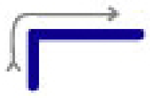
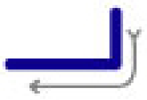
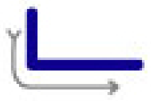
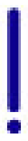
Gesture	Gesture name	Action
	Scratch-out	Erase content
	Triangle	Insert
	Square	Action item
	Star	Action item
	Check	Check-off
	Curlicue	Cut

	Double-Curlicue	Copy
	Circle	Application-specific
	Double-circle	Paste
	Left-semicircle	Undo
	Right-semicircle	Redo
	Caret	Paste/Insert
	Inverted-caret	Insert

	Chevron-left	Application-specific
	Chevron-right	Application-specific
	Arrow-up	Application-specific
	Arrow-down	Application-specific
	Arrow-left	Application-specific
	Arrow-right	Application-specific
	Up	Application-specific

	Down	Application-specific
	Left	Backspace
	Right	Space
	Up-left	Application-specific
	Up-right	Application-specific
	Down-left	Application-specific
	Down-right	Application-specific

	Left-up	Application-specific
	Left-down	Application-specific
	Right-up	Input Method Editor (IME) convert
	Right-down	Application-specific
	Up-down	Undo
	Down-up	Application-specific
	Left-right	Move cursor left

	Right-Left	Move cursor right
	Up-left-long	Decrease indent
	Up-right-long	Tab
	Down-left-long	Enter
	Down-right-long	Space
	Exclamation	Application-specific
	Tap	Click

	Double-tap	Left-Double-click

A.3 Multi-Stroke Gestures

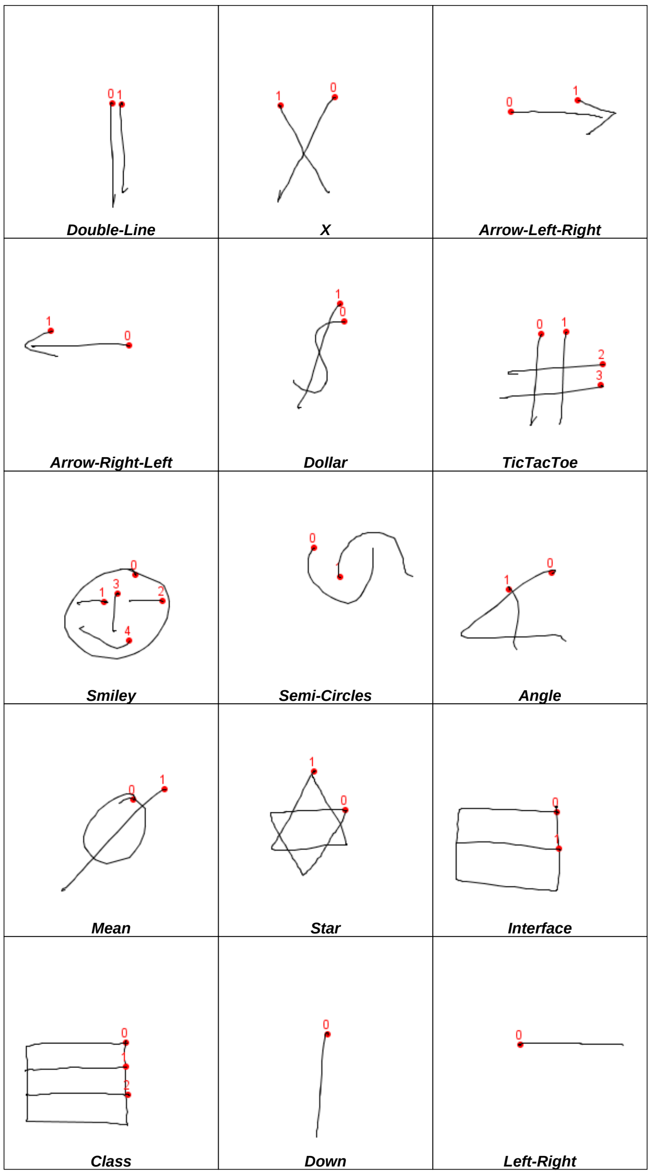


Figure A.3: Multi-Stroke Gestures

B XML Schema

B.1 org.sigtec.ink.Note

Listing B.1: XML Schema note.xsd

```

<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:element name="note" type="NoteType" />

  <xs:complexType name="NoteType">
    <xs:sequence minOccurs="1" maxOccurs="unbounded">
      <xs:element name="trace" type="TraceType" />
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="TraceType">
    <xs:sequence minOccurs="1" maxOccurs="unbounded">
      <xs:element name="point" type="PointType" />
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="PointType">
    <xs:sequence>
      <xs:element name="x" type="xs:double"></xs:element>
      <xs:element name="y" type="xs:double"></xs:element>
      <xs:element name="timestamp" type="xs:long"></xs:element>
      <xs:element name="force" type="xs:int"></xs:element>
      <xs:element name="yaw" type="xs:double"></xs:element>
      <xs:element name="pitch" type="xs:double"></xs:element>
      <xs:element name="rotation" type="xs:double"></xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```

B.2 org.ximtec.configuration.Configuration

Listing B.2: XML Schema configuration.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://org.ximtec.ipaper.gesture.configuration"
  xmlns:tns="http://org.ximtec.ipaper.gesture.configuration">

  <element name="configuration" type="tns:MappingType" />

  <complexType name="MappingType">
    <sequence>
      <element name="algorithm" type="tns:AlgorithmType"
               minOccurs="0" maxOccurs="unbounded" />
      <element name="set" type="tns:SetType" minOccurs="0"
               maxOccurs="unbounded" />
    </sequence>
  </complexType>

  <complexType name="AlgorithmType">
    <sequence minOccurs="1" maxOccurs="unbounded">
      <element name="parameter" type="tns:ParameterType"></element>
    </sequence>
    <attribute name="name" type="string"></attribute>
  </complexType>

  <complexType name="ParameterType">
    <simpleContent>
        <extension base="string">
        <attribute name="name" type="string"></attribute>
      </extension>
    </simpleContent>
  </complexType>

    <complexType name="SetType">
        <sequence minOccurs="0" maxOccurs="unbounded">
            <element name="class" type="tns:ClassType"></element>
        </sequence>
        <attribute name="name" type="string"></attribute>
    </complexType>

  <complexType name="ClassType">
    <sequence minOccurs="1" maxOccurs="unbounded">
      <element name="descriptor" type="tns:DescriptorType"></element>
    </sequence>
    <attribute name="name" type="string"></attribute>
  </complexType>

  <complexType name="DescriptorType">
    <sequence minOccurs="1" maxOccurs="unbounded">
      <element name="note" type="tns:SampleType"></element>
    </sequence>
    <attribute name="type" type="string"></attribute>
  </complexType>

  <complexType name="NoteType">
    <sequence minOccurs="1" maxOccurs="unbounded">
      <element name="trace" type="tns:TraceType"></element>
    </sequence>
  </complexType>

  <complexType name="TraceType">
    <sequence minOccurs="1" maxOccurs="unbounded">
      <element name="Point" type="tns:PointType"></element>
    </sequence>
  </complexType>

  <complexType name="PointType">
    <sequence>
      <element name="x" type="double"></element>
      <element name="y" type="double"></element>
      <element name="timestamp" type="long"></element>
      <element name="force" type="int"></element>
      <element name="yaw" type="xs:double"></element>
      <element name="pitch" type="xs:double"></element>
      <element name="rotation" type="xs:double"></element>
    </sequence>
  </complexType>

  <complexType name="SampleType">
    <sequence>
      <element name="note" type="tns:NoteType"></element>
    </sequence>
    <attribute name="name" type="string"></attribute>
  </complexType>
</schema>
```

Listing B.3: XML Schema mapping.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
   targetNamespace="http://org.ximtec.ipaper.gesture.mapping"
   xmlns:tns="http://org.ximtec.ipaper.gesture.mapping">

  <element name="mapping" type="tns:MappingType" />

  <complexType name="MappingType">
    <sequence>
      <element name="map" type="tns:MapType"
               minOccurs="0" maxOccurs="unbounded" />
    </sequence>
  </complexType>

  <complexType name="MapType">
    <attribute name="name" type="string"></attribute>
    <attribute name="value" type="integer"></attribute>
  </complexType>
</schema>
```

C Evaluation Statistics

C.1 Experiment 1

C.1.1 Extended Rubine Algorithm

Configuration

Parameter	Value
MAHALANOBIS_DISTANCE	13600.0
MIN_DISTANCE	2.0
FEATURE_LIST	F1, F2, F3, F4, F5, F6, F7, F8, F9, F10, F11, F12, F13, F16, F17, F18, F19, F20, F15, F14
PROPABILITY	0.95

Absolute values

Correct	Error	Reject Correct	Reject Error	Samples total	Noise
147	3	0	0	150	0

Absolute values per gesture class

Gesture Class	Correct	Error	Reject Correct	Reject Error
0_Graffiti	15	0	0	0
1_Graffiti	15	0	0	0
2_Graffiti	13	2	0	0
3_Graffiti	15	0	0	0
4_Graffiti	15	0	0	0
5_Graffiti	15	0	0	0
6_Graffiti	15	0	0	0
7_Graffiti	15	0	0	0
8_Graffiti	14	1	0	0
9_Graffiti	15	0	0	0
None	0	0	0	0

Key figures

Precision	Recall	F-measure
0.98	0.98	0.98

C.1.2 Original Rubine Algorithm

Configuration

Parameter	Value
MAHALANOBIS_DISTANCE	6600.0
MIN_DISTANCE	3.0
FEATURE_LIST	F1, F2, F3, F4, F5, F6, F7, F8, F9, F10, F11, F12, F13
PROPABILITY	0.95

Absolute values

Correct	Error	Reject Correct	Reject Error	Samples total	Noise
134	15	0	1	150	0

Absolute values per gesture class

Gesture Class	Correct	Error	Reject Correct	Reject Error
0_Graffiti	15	0	0	0
1_Graffiti	15	0	0	0
2_Graffiti	14	1	0	0
3_Graffiti	14	1	0	0
4_Graffiti	14	0	0	1
5_Graffiti	15	0	0	0
6_Graffiti	15	0	0	0
7_Graffiti	8	7	0	0
8_Graffiti	14	1	0	0
9_Graffiti	10	5	0	0
None	0	0	0	0

Key figures

Precision	Recall	F-measure
0.8993288590604027	0.8933333333333333	0.8963210702341138

C.1.3 Signature Algorithm

Configuration

Parameter	Value
DISTANCE_FUNCTION	org.ximtec.igesture.algorithm.signature.HammingDistance
RASTER_SIZE	250.0
MIN_ACCURACY	0.7000000000000001
MIN_DISTANCE	1.0
GRID_SIZE	8.0

Absolute values

Correct	Error	Reject Correct	Reject Error	Samples total	Noise
126	9	0	15	150	0

Absolute values per gesture class

Gesture Class	Correct	Error	Reject Correct	Reject Error
0_Graffiti	11	1	0	3
1_Graffiti	15	0	0	0
2_Graffiti	12	2	0	1
3_Graffiti	14	0	0	1
4_Graffiti	14	0	0	1
5_Graffiti	11	3	0	1
6_Graffiti	15	0	0	0
7_Graffiti	12	1	0	2
8_Graffiti	13	0	0	2
9_Graffiti	9	2	0	4
None	0	0	0	0

Key figures

Precision	Recall	F-measure
0.9333333333333333	0.84	0.8842105263157894

C.1.4 SiGeR Algorithm

Configuration

Parameter	Value
MIN_DISTANCE	2.0

Absolute values

Correct	Error	Reject Correct	Reject Error	Samples total	Noise
131	6	0	13	150	0

Absolute values per gesture class

Gesture Class	Correct	Error	Reject Correct	Reject Error
0_Graffiti	15	0	0	0
1_Graffiti	14	0	0	1
2_Graffiti	13	2	0	0
3_Graffiti	15	0	0	0
4_Graffiti	12	3	0	0
5_Graffiti	13	1	0	1
6_Graffiti	10	0	0	5
7_Graffiti	15	0	0	0
8_Graffiti	14	0	0	1
9_Graffiti	10	0	0	5
None	0	0	0	0

Key figures

Precision	Recall	F-measure
0.9562043795620438	0.8733333333333333	0.9128919860627178

C.2 Experiment 2

C.2.1 Extended Rubine Algorithm

Configuration

Parameter	Value
MAHALANOBIS_DISTANCE	6000.0
MIN_DISTANCE	1.0
FEATURE_LIST	F1, F2, F3, F4, F5, F6, F7, F8, F9, F10, F11, F12, F13, F16, F17, F18, F19, F20, F15, F14
PROPABILITY	0.95

Absolute values

Correct	Error	Reject Correct	Reject Error	Samples total	Noise
140	0	0	0	140	0

Absolute values per gesture class

Gesture Class	Correct	Error	Reject Correct	Reject Error
0_Graffiti	14	0	0	0
1_Graffiti	14	0	0	0
2_Graffiti	14	0	0	0
3_Graffiti	14	0	0	0
4_Graffiti	14	0	0	0
5_Graffiti	14	0	0	0
6_Graffiti	14	0	0	0
7_Graffiti	14	0	0	0
8_Graffiti	14	0	0	0
9_Graffiti	14	0	0	0
None	0	0	0	0

Key figures

Precision	Recall	F-measure
1.0	1.0	1

C.2.2 Original Rubine Algorithm

Configuration

Parameter	Value
MAHALANOBIS_DISTANCE	2800.0
MIN_DISTANCE	3.0
FEATURE_LIST	F1, F2, F3, F4, F5, F6, F7, F8, F9, F10, F11, F12, F13
PROPABILITY	0.95

Absolute values

Correct	Error	Reject Correct	Reject Error	Samples total	Noise
135	4	0	1	140	0

Absolute values per gesture class

Gesture Class	Correct	Error	Reject Correct	Reject Error
0_Graffiti	14	0	0	0
1_Graffiti	14	0	0	0
2_Graffiti	14	0	0	0
3_Graffiti	14	0	0	0
4_Graffiti	13	1	0	0
5_Graffiti	14	0	0	0
6_Graffiti	14	0	0	0
7_Graffiti	13	1	0	0
8_Graffiti	12	2	0	0
9_Graffiti	13	0	0	1
None	0	0	0	0

Key figures

Precision	Recall	F-measure
0.9712230215827338	0.9642857142857143	0.967741935483871

C.2.3 Signature Algorithm

Configuration

Parameter	Value
DISTANCE_FUNCTION	org.ximtec.igesture.algorithm.signature.HammingDistance
RASTER_SIZE	250.0
MIN_ACCURACY	0.7500000000000001
MIN_DISTANCE	1.0
GRID_SIZE	8.0

Absolute values

Correct	Error	Reject Correct	Reject Error	Samples total	Noise
120	13	0	7	140	0

Absolute values per gesture class

Gesture Class	Correct	Error	Reject Correct	Reject Error
0_Graffiti	11	3	0	0
1_Graffiti	14	0	0	0
2_Graffiti	12	2	0	0
3_Graffiti	11	2	0	1
4_Graffiti	13	0	0	1
5_Graffiti	11	3	0	0
6_Graffiti	13	0	0	1
7_Graffiti	13	0	0	1
8_Graffiti	14	0	0	0
9_Graffiti	8	3	0	3
None	0	0	0	0

Key figures

Precision	Recall	F-measure
0.9022556390977443	0.8571428571428571	0.879120879120879

C.3 Experiment 3

C.3.1 Extended Rubine Algorithm

Configuration

Parameter	Value
MAHALANOBIS_DISTANCE	3600.0
MIN_DISTANCE	3.0
FEATURE_LIST	F1, F2, F3, F4, F5, F6, F7, F8, F9, F10, F11, F12, F13, F16, F17, F18, F19, F20, F15, F14
PROPABILITY	0.95

Absolute values

Correct	Error	Reject Correct	Reject Error	Samples total	Noise
334	52	0	4	390	0

Absolute values per gesture class

Gesture Class	Correct	Error	Reject Correct	Reject Error
A_Graffiti	15	0	0	0
B_Graffiti	11	3	0	1
C_Graffiti	9	5	0	1
D_Graffiti	5	10	0	0
E_Graffiti	14	1	0	0
F_Graffiti	15	0	0	0
G_Graffiti	14	1	0	0
H_Graffiti	10	4	0	1
I_Graffiti	13	2	0	0
J_Graffiti	15	0	0	0
K_Graffiti	15	0	0	0
L_Graffiti	15	0	0	0
M_Graffiti	15	0	0	0
N_Graffiti	15	0	0	0
None	0	0	0	0
O_Graffiti	11	4	0	0
P_Graffiti	8	7	0	0
Q_Graffiti	13	2	0	0
R_Graffiti	15	0	0	0
S_Graffiti	15	0	0	0
T_Graffiti	15	0	0	0
U_Graffiti	15	0	0	0
V_Graffiti	14	1	0	0
W_Graffiti	14	1	0	0
X_Graffiti	15	0	0	0
Y_Graffiti	11	4	0	0
Z_Graffiti	7	7	0	1

Key figures

Precision	Recall	F-measure
0.8652849740932642	0.8564102564102564	0.8608247422680412

C.3.2 Original Rubine Algorithm

Configuration

Parameter	**Value**
MAHALANOBIS_DISTANCE	3200.0
MIN_DISTANCE	3.0
FEATURE_LIST	F1, F2, F3, F4, F5, F6, F7, F8, F9, F10, F11, F12, F13
PROPABILITY	0.95

Absolute values

Correct	**Error**	**Reject Correct**	**Reject Error**	**Samples total**	**Noise**
280	107	0	3	390	0

Absolute values per gesture class

Gesture Class	Correct	Error	Reject Correct	Reject Error
A_Graffiti	2	13	0	0
B_Graffiti	12	2	0	1
C_Graffiti	9	6	0	0
D_Graffiti	5	10	0	0
E_Graffiti	14	1	0	0
F_Graffiti	11	4	0	0
G_Graffiti	13	2	0	0
H_Graffiti	12	3	0	0
I_Graffiti	15	0	0	0
J_Graffiti	11	4	0	0
K_Graffiti	14	1	0	0
L_Graffiti	7	8	0	0
M_Graffiti	15	0	0	0
N_Graffiti	15	0	0	0
None	0	0	0	0
O_Graffiti	11	4	0	0
P_Graffiti	11	3	0	1
Q_Graffiti	10	5	0	0
R_Graffiti	15	0	0	0
S_Graffiti	15	0	0	0
T_Graffiti	10	5	0	0
U_Graffiti	4	11	0	0
V_Graffiti	5	10	0	0
W_Graffiti	12	3	0	0
X_Graffiti	13	2	0	0
Y_Graffiti	13	2	0	0
Z_Graffiti	6	8	0	1

Key figures

Precision	Recall	F-measure
0.7235142118863049	0.717948717948718	0.7207207207207208

C.3.3 Signature Algorithm

Configuration

Parameter	**Value**
DISTANCE_FUNCTION	org.ximtec.igesture.algorithm.signature.HammingDistance
RASTER_SIZE	250.0
MIN_ACCURACY	0.6
MIN_DISTANCE	1.0
GRID_SIZE	8.0

Absolute values

Correct	**Error**	**Reject Correct**	**Reject Error**	**Samples total**	**Noise**
261	126	0	3	390	0

Absolute values per gesture class

Gesture Class	Correct	Error	Reject Correct	Reject Error
A_Graffiti	10	5	0	0
B_Graffiti	10	5	0	0
C_Graffiti	9	6	0	0
D_Graffiti	6	9	0	0
E_Graffiti	12	3	0	0
F_Graffiti	10	5	0	0
G_Graffiti	12	2	0	1
H_Graffiti	6	9	0	0
I_Graffiti	12	2	0	1
J_Graffiti	14	1	0	0
K_Graffiti	9	5	0	1
L_Graffiti	15	0	0	0
M_Graffiti	15	0	0	0
N_Graffiti	7	8	0	0
None	0	0	0	0
O_Graffiti	13	2	0	0
P_Graffiti	4	11	0	0
Q_Graffiti	4	11	0	0
R_Graffiti	8	7	0	0
S_Graffiti	6	9	0	0
T_Graffiti	15	0	0	0
U_Graffiti	6	9	0	0
V_Graffiti	8	7	0	0
W_Graffiti	14	1	0	0
X_Graffiti	9	6	0	0
Y_Graffiti	12	3	0	0
Z_Graffiti	15	0	0	0

Key figures

Precision	Recall	F-measure
0.6744186046511628	0.6692307692307692	0.6718146718146718

C.4 Experiment 4

C.4.1 Extended Rubine Algorithm

Configuration

Parameter	**Value**
MAHALANOBIS_DISTANCE	2800.0
MIN_DISTANCE	3.0
FEATURE_LIST	F1, F2, F3, F4, F5, F6, F7, F8, F9, F10, F11, F12, F13, F16, F17, F18, F19, F20, F15, F14
PROPABILITY	0.95

Absolute values

Correct	**Error**	**Reject Correct**	**Reject Error**	**Samples total**	**Noise**
342	18	0	3	363	0

Absolute values per gesture class

Gesture Class	Correct	Error	Reject Correct	Reject Error
A_Graffiti	14	0	0	0
B_Graffiti	9	5	0	0
C_Graffiti	14	0	0	0
D_Graffiti	14	0	0	0
E_Graffiti	14	0	0	0
F_Graffiti	14	0	0	0
G_Graffiti	14	0	0	0
H_Graffiti	13	1	0	0
I_Graffiti	13	0	0	0
J_Graffiti	14	0	0	0
K_Graffiti	14	0	0	0
L_Graffiti	14	0	0	0
M_Graffiti	14	0	0	0
N_Graffiti	14	0	0	0
None	0	0	0	0
O_Graffiti	11	1	0	2
P_Graffiti	14	0	0	0
Q_Graffiti	14	0	0	0
R_Graffiti	12	1	0	1
S_Graffiti	12	2	0	0
T_Graffiti	14	0	0	0
U_Graffiti	14	0	0	0
V_Graffiti	10	4	0	0
W_Graffiti	14	0	0	0
X_Graffiti	14	0	0	0
Y_Graffiti	11	3	0	0
Z_Graffiti	13	1	0	0

Key figures

Precision	Recall	F-measure
0.95	0.9421487603305785	0.946058091286307

C.4.2 Original Rubine Algorithm

Configuration

Parameter	Value
MAHALANOBIS_DISTANCE	1600.0
MIN_DISTANCE	3.0
FEATURE_LIST	F1, F2, F3, F4, F5, F6, F7, F8, F9, F10, F11, F12, F13
PROPABILITY	0.95

Absolute values

Correct	Error	Reject Correct	Reject Error	Samples total	Noise
305	48	0	10	363	0

Absolute values per gesture class

Gesture Class	Correct	Error	Reject Correct	Reject Error
A_Graffiti	3	11	0	0
B_Graffiti	8	6	0	0
C_Graffiti	14	0	0	0
D_Graffiti	13	1	0	0
E_Graffiti	14	0	0	0
F_Graffiti	12	1	0	1
G_Graffiti	14	0	0	0
H_Graffiti	13	1	0	0
I_Graffiti	13	0	0	0
J_Graffiti	10	3	0	1
K_Graffiti	14	0	0	0
L_Graffiti	9	3	0	2
M_Graffiti	14	0	0	0
N_Graffiti	14	0	0	0
None	0	0	0	0
O_Graffiti	11	2	0	1
P_Graffiti	14	0	0	0
Q_Graffiti	11	2	0	1
R_Graffiti	14	0	0	0
S_Graffiti	14	0	0	0
T_Graffiti	7	6	0	1
U_Graffiti	12	1	0	1
V_Graffiti	4	9	0	1
W_Graffiti	13	1	0	0
X_Graffiti	14	0	0	0
Y_Graffiti	14	0	0	0
Z_Graffiti	12	1	0	1

Key figures

Precision	Recall	F-measure
0.8640226628895185	0.8402203856749312	0.8519553072625698

C.4.3 Signature Algorithm

Configuration

Parameter	**Value**
DISTANCE_FUNCTION	org.ximtec.igesture.algorithm.signature.HammingDistance
RASTER_SIZE	250.0
MIN_ACCURACY	0.7500000000000001
MIN_DISTANCE	1.0
GRID_SIZE	12.0

Absolute values

Correct	**Error**	**Reject Correct**	**Reject Error**	**Samples total**	**Noise**
253	32	0	78	363	0

Absolute values per gesture class

Gesture Class	Correct	Error	Reject Correct	Reject Error
A_Graffiti	5	0	0	9
B_Graffiti	10	2	0	2
C_Graffiti	11	1	0	2
D_Graffiti	8	2	0	4
E_Graffiti	12	2	0	0
F_Graffiti	14	0	0	0
G_Graffiti	9	4	0	1
H_Graffiti	8	0	0	6
I_Graffiti	11	0	0	2
J_Graffiti	12	0	0	2
K_Graffiti	7	1	0	6
L_Graffiti	12	1	0	1
M_Graffiti	11	0	0	3
N_Graffiti	12	0	0	2
None	0	0	0	0
O_Graffiti	10	4	0	0
P_Graffiti	9	1	0	4
Q_Graffiti	10	3	0	1
R_Graffiti	9	4	0	1
S_Graffiti	7	3	0	4
T_Graffiti	9	1	0	4
U_Graffiti	13	1	0	0
V_Graffiti	8	1	0	5
W_Graffiti	6	1	0	7
X_Graffiti	7	0	0	7
Y_Graffiti	10	0	0	4
Z_Graffiti	13	0	0	1

Key figures

Precision	Recall	F-measure
0.887719298245614	0.696969696969697	0.7808641975308642

C.5 Experiment 5

C.5.1 Extended Rubine Algorithm

Configuration

Parameter	Value
MAHALANOBIS_DISTANCE	400.0
MIN_DISTANCE	3.0
FEATURE_LIST	F1, F2, F3, F4, F5, F6, F7, F8, F9, F10, F11, F12, F13, F16, F17, F18, F19, F20, F15, F14
PROPABILITY	0.95

Absolute values

Correct	Error	Reject Correct	Reject Error	Samples total	Noise
114	11	319	26	470	330

Absolute values per gesture class

Gesture Class	Correct	Error	Reject Correct	Reject Error
0_Graffiti	7	0	0	7
1_Graffiti	14	0	0	0
2_Graffiti	11	0	0	3
3_Graffiti	14	0	0	0
4_Graffiti	14	0	0	0
5_Graffiti	13	0	0	1
6_Graffiti	12	0	0	2
7_Graffiti	13	0	0	1
8_Graffiti	3	0	0	11
9_Graffiti	13	0	0	1
None	0	11	319	0

Key figures

Precision	Recall	F-measure
0.912	0.8142857142857143	0.860377358490566

C.5.2 Original Rubine Algorithm

Configuration

Parameter	Value
MAHALANOBIS_DISTANCE	400.0
MIN_DISTANCE	3.0
FEATURE_LIST	F1, F2, F3, F4, F5, F6, F7, F8, F9, F10, F11, F12, F13
PROPABILITY	0.95

Absolute values

Correct	Error	Reject Correct	Reject Error	Samples total	Noise
131	89	243	7	470	330

Absolute values per gesture class

Gesture Class	Correct	Error	Reject Correct	Reject Error
0_Graffiti	14	0	0	0
1_Graffiti	14	0	0	0
2_Graffiti	14	0	0	0
3_Graffiti	14	0	0	0
4_Graffiti	13	1	0	0
5_Graffiti	14	0	0	0
6_Graffiti	14	0	0	0
7_Graffiti	13	1	0	0
8_Graffiti	9	0	0	5
9_Graffiti	12	0	0	2
None	0	87	243	0

Key figures

Precision	Recall	F-measure
0.5954545454545455	0.9357142857142857	0.7277777777777779

C.5.3 Signature Algorithm

Configuration

Parameter	Value
DISTANCE_FUNCTION	org.ximtec.igesture.algorithm.signature.HammingDistance
RASTER_SIZE	250.0
MIN_ACCURACY	0.8000000000000002
MIN_DISTANCE	1.0
GRID_SIZE	8.0

Absolute values

Correct	Error	Reject Correct	Reject Error	Samples total	Noise
111	39	294	26	470	330

Absolute values per gesture class

Gesture Class	Correct	Error	Reject Correct	Reject Error
0_Graffiti	13	0	0	1
1_Graffiti	14	0	0	0
2_Graffiti	10	0	0	4
3_Graffiti	7	0	0	7
4_Graffiti	13	0	0	1
5_Graffiti	11	2	0	1
6_Graffiti	13	0	0	1
7_Graffiti	11	0	0	3
8_Graffiti	12	0	0	2
9_Graffiti	7	1	0	6
None	0	36	294	0

Key figures

Precision	Recall	F-measure
0.74	0.7928571428571428	0.7655172413793102

C.6 Experiment 6

C.6.1 Extended Rubine Algorithm

Configuration

Parameter	Value
MAHALANOBIS_DISTANCE	10800.0
MIN_DISTANCE	2.0
FEATURE_LIST	F1, F2, F3, F4, F5, F6, F7, F8, F9, F10, F11, F12, F13, F21, F22, F14, F23, F24, F25, F15, F16, F17, F18, F19
PROPABILITY	0.95

Absolute values

Correct	Error	Reject Correct	Reject Error	Samples total	Noise
196	2	0	2	200	0

Absolute values per gesture class

Gesture Class	Correct	Error	Reject Correct	Reject Error
Arrow-down	5	0	0	0
Arrow-left	5	0	0	0
Arrow-right	5	0	0	0
Arrow-up	5	0	0	0
Caret	5	0	0	0
Check	4	1	0	0
Chevron-left	5	0	0	0
Chevron-right	5	0	0	0
Circle	5	0	0	0
Curlicue	5	0	0	0
Double-circle	5	0	0	0
Double-Curlicue	5	0	0	0
Down	5	0	0	0
Down-left	5	0	0	0
Down-left-long	5	0	0	0
Down-right	5	0	0	0
Down-right-long	5	0	0	0
Down-up	5	0	0	0
Exclamation	5	0	0	0
Inverted-caret	4	0	0	1
Left	5	0	0	0
Left-down	5	0	0	0
Left-right	5	0	0	0
Left-semicircle	5	0	0	0
Left-up	5	0	0	0
None	0	0	0	0
Right	5	0	0	0
Right-down	5	0	0	0
Right-Left	4	1	0	0
Right-semicircle	5	0	0	0
Right-up	5	0	0	0
Scratch-out	5	0	0	0
Square	5	0	0	0
Star	5	0	0	0
Triangle	5	0	0	0
Up	5	0	0	0
Up-down	5	0	0	0
Up-left	5	0	0	0
Up-left-long	5	0	0	0
Up-right	4	0	0	1
Up-right-long	5	0	0	0

Key figures

Precision	**Recall**	**F-measure**
0.98989898989899	0.98	0.9849246231155778

C.6.2 Original Rubine Algorithm

Configuration

Parameter	**Value**
MAHALANOBIS_DISTANCE	3200.0
MIN_DISTANCE	3.0
FEATURE_LIST	F1, F2, F3, F4, F5, F6, F7, F8, F9, F10, F11, F12, F13
PROPABILITY	0.95

Absolute values

Correct	**Error**	**Reject Correct**	**Reject Error**	**Samples total**	**Noise**
178	19	0	3	200	0

Absolute values per gesture class

Gesture Class	Correct	Error	Reject Correct	Reject Error
Arrow-down	5	0	0	0
Arrow-left	4	1	0	0
Arrow-right	5	0	0	0
Arrow-up	5	0	0	0
Caret	4	1	0	0
Check	3	2	0	0
Chevron-left	4	1	0	0
Chevron-right	3	2	0	0
Circle	5	0	0	0
Curlicue	5	0	0	0
Double-circle	5	0	0	0
Double-Curlicue	5	0	0	0
Down	5	0	0	0
Down-left	5	0	0	0
Down-left-long	5	0	0	0
Down-right	4	0	0	1
Down-right-long	5	0	0	0
Down-up	4	1	0	0
Exclamation	5	0	0	0
Inverted-caret	5	0	0	0
Left	5	0	0	0
Left-down	1	4	0	0
Left-right	1	4	0	0
Left-semicircle	4	0	0	1
Left-up	3	2	0	0
None	0	0	0	0
Right	5	0	0	0
Right-down	5	0	0	0
Right-Left	4	1	0	0
Right-semicircle	5	0	0	0
Right-up	5	0	0	0
Scratch-out	5	0	0	0
Square	5	0	0	0
Star	5	0	0	0
Triangle	5	0	0	0
Up	5	0	0	0
Up-down	4	0	0	1
Up-left	5	0	0	0
Up-left-long	5	0	0	0
Up-right	5	0	0	0
Up-right-long	5	0	0	0

Key figures

Precision	**Recall**	**F-measure**
0.9035532994923858	0.89	0.8967254408060454

C.6.3 Signature Algorithm

Configuration

Parameter	**Value**
DISTANCE_FUNCTION	org.ximtec.igesture.algorithm.signature.HammingDistance
RASTER_SIZE	250.0
MIN_ACCURACY	0.7000000000000001
MIN_DISTANCE	1.0
GRID_SIZE	12.0

Absolute values

Correct	**Error**	**Reject Correct**	**Reject Error**	**Samples total**	**Noise**
145	32	0	23	200	0

Absolute values per gesture class

Gesture Class	Correct	Error	Reject Correct	Reject Error
Arrow-down	4	0	0	1
Arrow-left	4	1	0	0
Arrow-right	5	0	0	0
Arrow-up	0	1	0	4
Caret	4	0	0	1
Check	3	2	0	0
Chevron-left	5	0	0	0
Chevron-right	1	0	0	4
Circle	5	0	0	0
Curlicue	3	1	0	1
Double-circle	4	0	0	1
Double-Curlicue	5	0	0	0
Down	1	4	0	0
Down-left	5	0	0	0
Down-left-long	5	0	0	0
Down-right	3	2	0	0
Down-right-long	4	1	0	0
Down-up	5	0	0	0
Exclamation	3	2	0	0
Inverted-caret	3	2	0	0
Left	4	1	0	0
Left-down	4	1	0	0
Left-right	5	0	0	0
Left-semicircle	3	0	0	2
Left-up	5	0	0	0
None	0	0	0	0
Right	2	1	0	2
Right-down	2	3	0	0
Right-Left	4	1	0	0
Right-semicircle	3	2	0	0
Right-up	5	0	0	0
Scratch-out	4	0	0	1
Square	3	0	0	2
Star	3	0	0	2
Triangle	2	1	0	2
Up	3	2	0	0
Up-down	5	0	0	0
Up-left	5	0	0	0
Up-left-long	5	0	0	0
Up-right	1	4	0	0
Up-right-long	5	0	0	0

Key figures

Precision	Recall	F-measure
0.8192090395480226	0.725	0.7692307692307693

C.7 Experiment 7

C.7.1 Extended Rubine Algorithm

Configuration

Parameter	Value
MAHALANOBIS_DISTANCE	2000.0
MIN_DISTANCE	3.0
FEATURE_LIST	F1, F2, F3, F4, F5, F6, F7, F8, F9, F10, F11, F12, F13, F21, F22, F14, F23, F24, F25, F15, F16, F17, F18, F19
PROPABILITY	0.95

Absolute values

Correct	Error	Reject Correct	Reject Error	Samples total	Noise
75	0	0	0	75	0

Absolute values per gesture class

Gesture Class	Correct	Error	Reject Correct	Reject Error
Angle	5	0	0	0
Arrow-Left-Right	5	0	0	0
Arrow-Right-Left	5	0	0	0
Class	5	0	0	0
Dollar	5	0	0	0
Double-Line	5	0	0	0
Down	5	0	0	0
Interface	5	0	0	0
Left-Right	5	0	0	0
Mean	5	0	0	0
None	0	0	0	0
Semi-Circles	5	0	0	0
Smiley	5	0	0	0
Star	5	0	0	0
TicTacToe	5	0	0	0
X	5	0	0	0

Key figures

Precision	**Recall**	**F-measure**
1.0	1.0	1

C.7.2 Original Rubine Algorithm

Configuration

Parameter	**Value**
MAHALANOBIS_DISTANCE	1200.0
MIN_DISTANCE	1.0
FEATURE_LIST	F1, F2, F3, F4, F5, F6, F7, F8, F9, F10, F11, F12, F13
PROPABILITY	0.95

Absolute values

Correct	**Error**	**Reject Correct**	**Reject Error**	**Samples total**	**Noise**
72	3	0	0	75	0

Absolute values per gesture class

Gesture Class	**Correct**	**Error**	**Reject Correct**	**Reject Error**
Angle	5	0	0	0
Arrow-Left-Right	5	0	0	0
Arrow-Right-Left	5	0	0	0
Class	5	0	0	0
Dollar	5	0	0	0
Double-Line	5	0	0	0
Down	5	0	0	0
Interface	5	0	0	0
Left-Right	5	0	0	0
Mean	2	3	0	0
None	0	0	0	0
Semi-Circles	5	0	0	0
Smiley	5	0	0	0
Star	5	0	0	0
TicTacToe	5	0	0	0
X	5	0	0	0

Key figures

Precision	Recall	F-measure
0.96	0.96	0.96

C.7.3 Signature Algorithm

Configuration

Parameter	Value
DISTANCE_FUNCTION	org.ximtec.igesture.algorithm.signature.HammingDistance
RASTER_SIZE	250.0
MIN_ACCURACY	0.7000000000000001
MIN_DISTANCE	1.0
GRID_SIZE	8.0

Absolute values

Correct	Error	Reject Correct	Reject Error	Samples total	Noise
72	1	0	2	75	0

Absolute values per gesture class

Gesture Class	Correct	Error	Reject Correct	Reject Error
Angle	5	0	0	0
Arrow-Left-Right	5	0	0	0
Arrow-Right-Left	5	0	0	0
Class	4	1	0	0
Dollar	5	0	0	0
Double-Line	5	0	0	0
Down	5	0	0	0
Interface	5	0	0	0
Left-Right	5	0	0	0
Mean	3	0	0	2
None	0	0	0	0
Semi-Circles	5	0	0	0
Smiley	5	0	0	0
Star	5	0	0	0
TicTacToe	5	0	0	0
X	5	0	0	0

Key figures

Precision	Recall	F-measure
0.9863013698630136	0.96	0.9729729729729729

C.8 Experiment 8

C.8.1 Extended Rubine Algorithm

Configuration

Parameter	Value
MAHALANOBIS_DISTANCE	12800.0
MIN_DISTANCE	1.0
FEATURE_LIST	F1, F2, F3, F4, F5, F6, F7, F8, F9, F10, F11, F12, F13, F16, F17, F18, F19, F20, F15, F14
PROPABILITY	0.95

Absolute values

Correct	Error	Reject Correct	Reject Error	Samples total	Noise
74	1	0	0	75	0

Absolute values per gesture class

Gesture Class	Correct	Error	Reject Correct	Reject Error
Angle	5	0	0	0
Arrow-Left-Right	5	0	0	0
Arrow-Right-Left	5	0	0	0
Class	5	0	0	0
Dollar	5	0	0	0
Double-Line	5	0	0	0
Down	5	0	0	0
Interface	5	0	0	0
Left-Right	5	0	0	0
Mean	5	0	0	0
None	0	0	0	0
Semi-Circles	5	0	0	0
Smiley	5	0	0	0
Star	4	1	0	0
TicTacToe	5	0	0	0
X	5	0	0	0

Key figures

Precision	Recall	F-measure
0.9866666666666667	0.9866666666666667	0.9866666666666668

C.8.2 Original Rubine Algorithm

Configuration

Parameter	Value
MAHALANOBIS_DISTANCE	5200.0
MIN_DISTANCE	3.0
FEATURE_LIST	F1, F2, F3, F4, F5, F6, F7, F8, F9, F10, F11, F12, F13
PROPABILITY	0.95

Absolute values

Correct	Error	Reject Correct	Reject Error	Samples total	Noise
73	1	0	1	75	0

Absolute values per gesture class

Gesture Class	Correct	Error	Reject Correct	Reject Error
Angle	5	0	0	0
Arrow-Left-Right	5	0	0	0
Arrow-Right-Left	4	0	0	1
Class	5	0	0	0
Dollar	5	0	0	0
Double-Line	5	0	0	0
Down	5	0	0	0
Interface	5	0	0	0
Left-Right	5	0	0	0
Mean	5	0	0	0
None	0	0	0	0
Semi-Circles	5	0	0	0
Smiley	5	0	0	0
Star	4	1	0	0
TicTacToe	5	0	0	0
X	5	0	0	0

Key figures

Precision	Recall	F-measure
0.9864864864864865	0.9733333333333334	0.9798657718120806

C.8.3 Signature Algorithm

Configuration

Parameter	Value
DISTANCE_FUNCTION	org.ximtec.igesture.algorithm.signature.HammingDistance
RASTER_SIZE	200.0
MIN_ACCURACY	0.6
MIN_DISTANCE	1.0
GRID_SIZE	8.0

Absolute values

Correct	Error	Reject Correct	Reject Error	Samples total	Noise
74	1	0	0	75	0

Absolute values per gesture class

Gesture Class	Correct	Error	Reject Correct	Reject Error
Angle	5	0	0	0
Arrow-Left-Right	5	0	0	0
Arrow-Right-Left	5	0	0	0
Class	5	0	0	0
Dollar	5	0	0	0
Double-Line	5	0	0	0
Down	5	0	0	0
Interface	5	0	0	0
Left-Right	5	0	0	0
Mean	5	0	0	0
None	0	0	0	0
Semi-Circles	5	0	0	0
Smiley	4	1	0	0
Star	5	0	0	0
TicTacToe	5	0	0	0
X	5	0	0	0

Key figures

Precision	Recall	F-measure
0.9866666666666667	0.9866666666666667	0.9866666666666668

Acknowledgements

I would like to thank my supervising assistant Dr. Beat Signer for having always time to support my diploma thesis in every respect, for his valuable feedback and last but not least for reviewing my report. I am also grateful to Professor Dr. Moira C. Norrie for giving me the opportunity to accomplish my diploma thesis in her group.

Bibliography

[1] Apache Jakarta Commons, http://jakarta.apache.org/commons.

[2] db4objects, http://www.db4o.com.

[3] LipiTk, http://sourceforge.net/projects/lipitk.

[4] Microsoft Application Gestures, http://msdn.microsoft.com/library/default.asp?url=/library/en-us/tpcsdk10/lonestar/whitepapers/designguide/tbconusingapplicationgesturesandtheirsemantics.asp.

[5] Microsoft Tablet PC SDK, http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnanchor/html/tabletpc.asp.

[6] Palm Graffiti, http://en.wikipedia.org/wiki/Graffiti_%28Palm_OS%29.

[7] SATIN, http://sourceforge.net/projects/satin.

[8] sigtec, http://www.sigtec.org.

[9] SwingGestures, http://sourceforge.net/projects/swinggestures.

[10] XStream, http://xstream.codehaus.org.

[11] Christine Alvarado and Randall Davis. SketchREAD: A Multi-Domain Sketch Recognition Engine. In *UIST '04: Proceedings of the 17th Annual ACM Symposium on User Interface Software and Technology*, pages 23–32, New York, NY, USA, 2004. ACM Press.

[12] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns*. Addison Wesley, 1994.

[13] Ken Hinckley, Patrick Baudisch, Gonzalo Ramos, and François Guimbretièrere. Design and Analysis of Delimiters for Selection-Action Pen Gesture Phrases in Scriboli. In *CHI '05: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 451–460, New York, NY, USA, 2005. ACM Press.

[14] Jason I. Hong and James A. Landay. SATIN: A Toolkit for Informal Ink-Based Applications. In *UIST '00: Proceedings of the 13th Annual ACM Symposium on User Interface Software and Technology*, pages 63–72, New York, NY, USA, 2000. ACM Press.

[15] Chunyuan Liao, François Guimbretièrere, and Ken Hinckley. PapierCraft: A Command System for Interactive Paper. In *UIST '05: Proceedings of the 18th Annual ACM Symposium on User Interface Software and Technology*, pages 241–244, New York, NY, USA, 2005. ACM Press.

[16] A. Christ Long, James A. Landay, and Lawrence A. Row. quill: Providing Advice for Pen-Based Gesture Design. 2003.

[17] Allan Christian Long, James A. Landay, and Lawrence A. Rowe. Implications for a Gesture Design Tool. In *CHI '99: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 40–47, New York, NY, USA, 1999. ACM Press.

[18] Sriganesh Madhvanath, Deepu Vijayasenan, and Thanigai Murugan Kadiresan. LipiTk: A Generic Toolkit for Online Handwriting Recognition. 2006.

[19] P.C̃. Mahalanobis. On the Generalized Distance in Statistics. In *National Institute of Science of India*, pages 49–55, Calcutta, India, 1936.

[20] Dean Rubine. Specifying Gestures by Example. In *SIGGRAPH '91: Proceedings of the 18th Annual Conference on Computer Graphics and Interactive Techniques*, pages 329–337, New York, NY, USA, 1991. ACM Press.

[21] Beat Signer. *Fundamental Concepts for Interactive Paper and Cross-Media Information Spaces*. PhD thesis, ETH Zurich, 2006.