Kinetic online trajectory recovery from static images of handwriting

Paul Konstantin Gerke - 0616427

***Abstract*—In this Bachelor thesis I propose two new approaches for extracting online handwriting data from scanned images of handwriting (offline data) using the mechanical concept of the momentum. The momentum describes the movement of a body and can only be changed by exerting forces, as first described by Sir I. Newton [1].**

The first approach extracts pen strokes directly from an offline handwriting by tracing lines with a tracing point that uses a momentum. The second approach tries to sequence pen strokes to whole pen trajectories by using a kinetic cost function that is based on concepts derived from the definition of the momentum.

An exploration of the limits of the first approach shows that it is not capable of dealing with noise that occurs in normal handwriting. The kinetic cost function of the second approach is compared to a traditional Euclidean distance based cost function for stroke sequencing. Using the kinetic cost function for stroke sequencing leads to significantly better pen trajectories than using the Euclidean distance cost function. Using a momentum based cost function for sequencing pen strokes can improve the quality of extracted pen trajectories.

I. Introduction

The increasing performance of modern handwriting recognition systems has led to many useful applications, such as automatic form processing or forensic applications like signature verification. Applications as automated form processing often require the user to write block letters because automated processing of images of cursive handwriting does not work reliably. In this Bachelor thesis, I investigate a method to improve information extraction from scans of cursive handwriting. My approach is based on the fact that *online handwriting recognition* performs better than *offline handwriting recognition* [2].

In online handwriting recognition, data about the writing process itself is used. This is usually done when a user writes with a stylus onto a tablet PC with a touch screen interface that translates the writing into digital text. The online data consists of a list of subsequent pen tip coordinates that are recorded at a constant sample rate. From this data the sequence and velocities of pen movements can be derived. Such dynamic pen information is valuable, for example, to guide character recognition within words [2]. Optionally, information about writing pressure and pen tilt can be recorded in online data, too [3]. These additional information can improve the recognition rates of online recognition systems.

Systems that do not use online data are called offline recognition systems. Offline recognition systems cannot use online data because they process scanned images, which are usually saved as pixel rasters. An example of a handwritten signature is shown in figure 1.a). A common technique that is used to recognize such offline handwritings is to try to recalculate the related online data (i.e., the related pen trajectory). Systems that recalculate online data from offline data are called trajectory recovery or trajectory extraction systems.

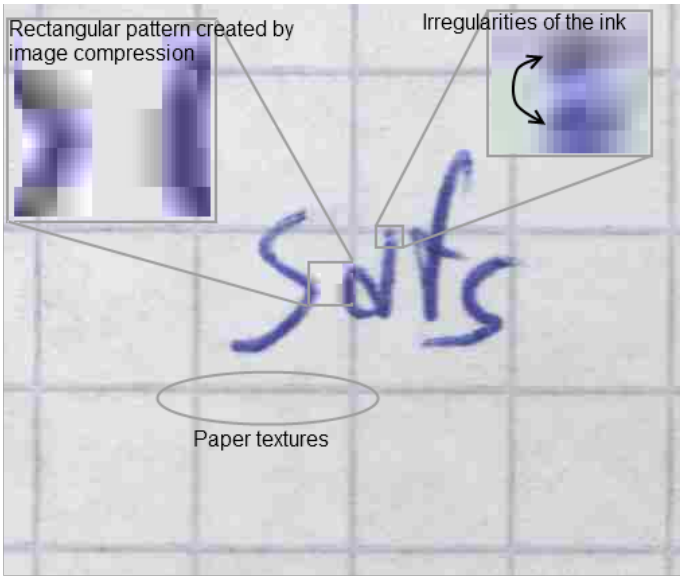

Fig. 2. Typical noise sources that can interfere with an online trajectory recovery system or an offline handwriting recognition system.

Online systems work better than offline recognition systems. Online recognition systems have been reported to achieve a 80% correct recognition rate using a lexicon of 21000 recognizable words. The lexicon is used to limit the amount of words that the system has to distinguish. Offline handwriting recognition systems using a lexicon size of 1000 words only achieve a recognition performance of 78% [2].

The most important reason for the performance difference is that the recovery of online data from offline data is hard. There is no system which is capable of recomputing online data from offline data and some researchers doubt if all aspects of online data are in fact reconstructible [4]. Present trajectory recovery systems usually first try to extract basic pen strokes from a scanned image (see figure 1b). These basic pen strokes are parts of original pen strokes and describe parts of pen movements that were executed during writing. The extraction of basic pen strokes is complicated by noise in the scanned image, as illustrated in figure 2.

After the extraction of basic pen strokes, the following step is to sequence the basic pen strokes in a way that replicates the most probable writing order to obtain a full pen trajectory. This sequencing problem is a combinatorial problem: Usually the fitness of a given candidate pen trajectory can be judged on

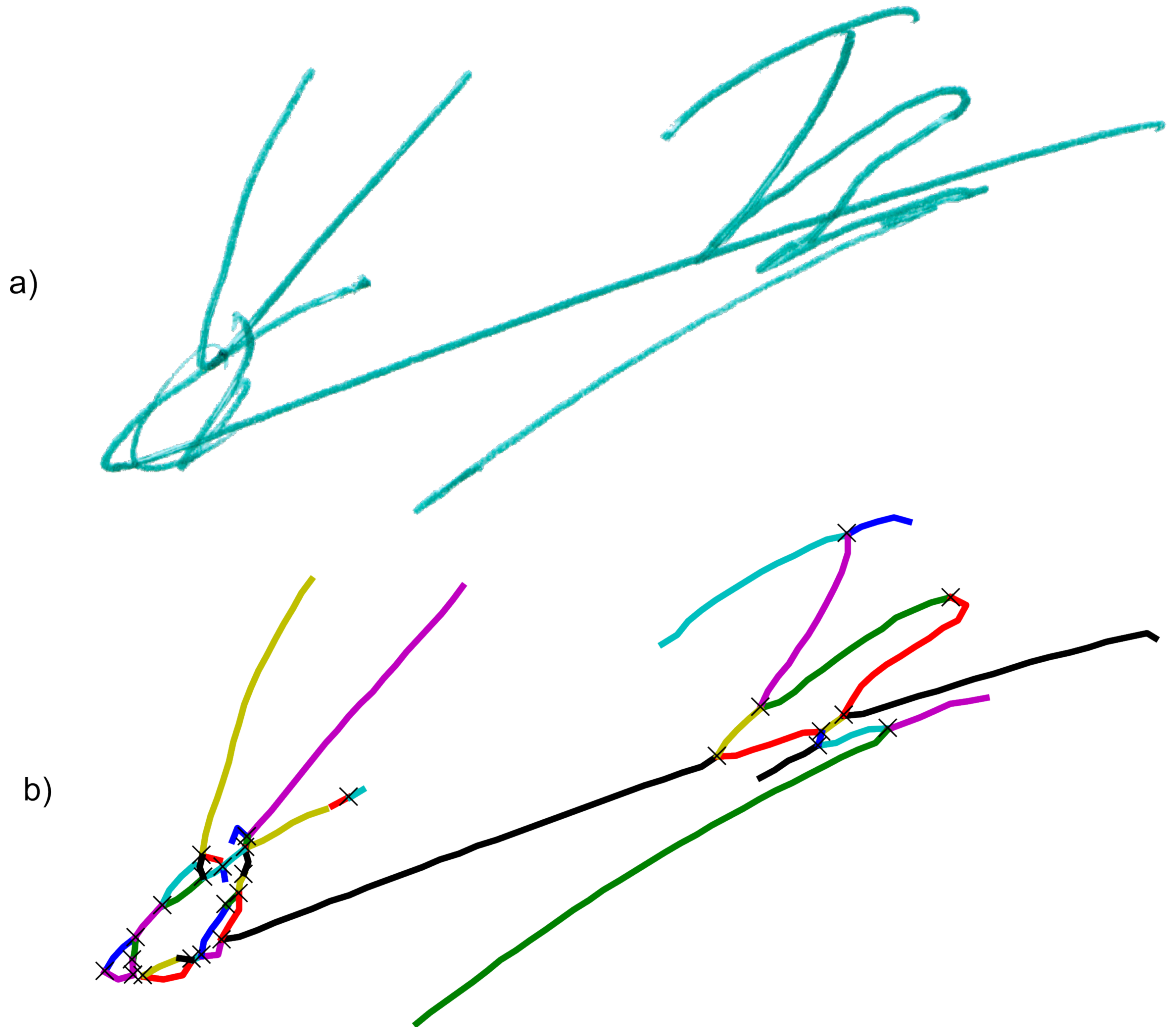

Fig. 1. a) A scanned image of a handwritten signature. b) basic pen strokes extracted from the signature in a). The beginning and end of pen strokes are marked with crosses.

the basis of a defined cost function. To find the pen trajectory with the optimal fitness all possible combinations of strokes have to be considered. Such a problem is a typical problem of classical artificial intelligence and can be solved by search algorithms [5]. However, for guiding the search only the basic strokes and sub-stroke data like, for example, striations in the ink are available. Until today there is no known effective way to sequence pen strokes to obtain original pen trajectories.

Both basic pen stroke extraction and stroke sequencing present large challenges for current trajectory recovery systems and do not work well enough to build reliable offline handwriting recognitions systems. In this Bachelor thesis, I explore new methods for extracting basic pen strokes from offline data and to sequence pen strokes in such a way that they resemble the correct writing order.

A. Standard online data recovery systems

In the past, several attempts have been made for building such a pen trajectory extraction system. Most of them follow

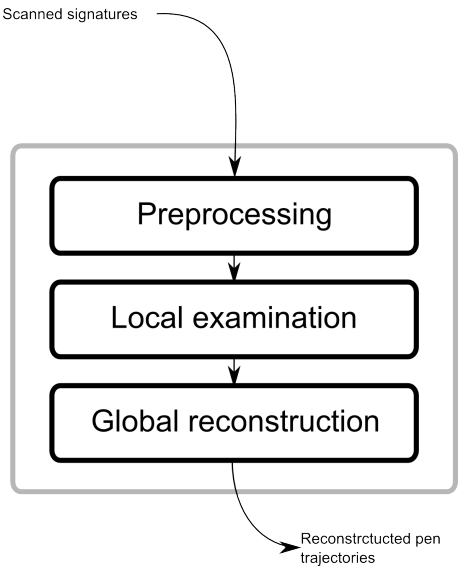

Fig. 3. Processing stages a typical trajectory recovery system consists of.

a standard design principle, which consists of three different stages (figure 3).

The first stage is the preprocessing stage. During preprocessing, noises may be filtered out or image processing techniques like gray scaling or image smoothing may be applied. These actions stabilize the quality of the input image to improve the final recovery results. In addition, a thinning or contour extraction algorithm is applied in order to extract basic pen strokes from the offline image. As previously explained, the basic strokes are usually the first line representation using coordinates. They are used by extraction systems for describing parts of the final pen trajectory.

The second stage is the local examination stage. At this stage an image of handwriting is analyzed thoroughly in order to extract clues on the sub-stroke level on how the handwriting sample was written. Clues may be, for example, detection of double traces [6], feathers, or striations in the ink [7]. The extracted clues are later on used during the global reconstruction stage.

At the third and final stage, global reconstruction, the pen strokes are sequenced to build a full pen trajectory. The sequencing is achieved with a classical artificial intelligence search algorithm that tries to minimize a certain global cost. The global cost is defined in such a way that the built pen trajectory resembles the correct writing order of the original handwriting as well as possible. The global cost is computed by a *cost function* that depends on stroke properties and clues extracted during the local examination stage.

In this Bachelor thesis, I propose two different ways of extracting pen trajectories from offline images of handwriting. Both approaches are based on the idea that a pen-tip's movement can be described by the mechanical concept of the momentum. In the following, I will explain the mechanical concept of the momentum. Then I will describe how both approaches employ the momentum for online pen trajectory recovery.

B. Mechanical concepts

I was inspired by the mechanical notion of a body's momentum and how the momentum is influenced by forces, to apply the same principles to online data recovery. The relation between force $\vec{F}$ and a body's momentum $\vec{p} = m\vec{v}$ is given by Newton's second law [1]:

$$\vec{F} = \frac{d}{dt}(m\vec{v}) \tag{1}$$

m refers to a body's mass and and $\vec{v}$ to a body's velocity. From equation 1 it can be derived that the momentum of a body changes as a force is applied to it over time:

$$\int \vec{F} dt = m\vec{v} \tag{2}$$

If the momentum changes, the velocity and/or the movement direction of the body changes, due to the relation $\vec{p} = m\vec{v}$. To alter a body's momentum by a applying a force to it, a certain amount of energy is necessary. The energy E needed to change the momentum is:

$$E = \int_C \vec{F} d\vec{s} \tag{3}$$

The vector $\vec{s}$ and limit C describe the movement the body during exertion of force $\vec{F}$.

C. Application of mechanics to trajectory recovery

In this thesis I investigate if a pen movement can be reconstructed from offline images using the given physical equations. I propose two different approaches based on the principle of momentum.

For the first approach, *low-level kinetic simulation*, it is assumed that a pen movement follows the basic principle of momentum. In this case, the pen trajectory could be extracted by simulating a moving ball with a momentum that follows the lines of an image of handwriting. This way strokes could be extracted from an image that continue at crossings. Normal thinning-based stroke extraction algorithms are not capable of deciding immediately how to continue a stroke at a crossing. The simulated ball would simply roll over the crossing favoring a straight way and be able to extract longer strokes from handwriting images.

For the second approach, *high-level kinetic reconstruction*, a standard trajectory recovery system is used (as introduced in section I-A). At the global reconstruction stage, the sequence of strokes is obtained by defining a kinetic cost function. The kinetic cost function computes a cost for a given pair of stroke ends. A curve is fitted from one end of a stroke to the other. The fitted curve is defined by the same type of function that describes the movement of objects that underly a momentum. Such a movement is described by a quadratic equation. This equation follows from Newton's second law (equation 1) and the definition of speed, which is $\frac{d\vec{x}}{dt} = \vec{v}$:

$$\vec{x}(t) = \frac{\iint \vec{F}\, dt^2 + \int \vec{p}\, dt}{m} + \vec{x}(0) \tag{4}$$

Assuming that the force $\vec{F}$ and the momentum $\vec{p}$ of an object stay constant over time, it is possible to derive the second order polynomial from equation 4:

$$\vec{x}(t) = \frac{\vec{F} \cdot \frac{t^2}{2} + \vec{p} \cdot t}{m} + \vec{x}(0) \tag{5}$$

If a pen's trajectory underlies a momentum, it should be possible to describe the pen movement between strokes with a function as $\vec{x}(t)$. The kinetic cost function fits the polynomial $\vec{x}(t)$ (equation 5) to a transition from one stroke to another by calculating $\vec{F}$ and $\vec{p}$. Then equation 3 is applied to compute the energy that is needed to make the transition. This energy is the cost computed by the kinetic cost function. When the global reconstruction stage minimizes the global cost, it minimizes the sum of stroke transition energies. This method assumes that people write cursive hand writing in a way that minimizes the over-all cost of stroke transitions.

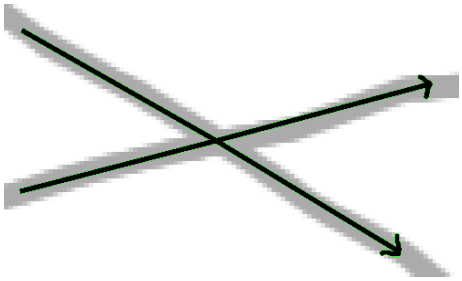

Fig. 4. A line crossing in a scanned image. If the lines are traced by a program using a simulated momentum it should be likely that it traces the lines as illustrated by the black arrows because the momentum does not allow abrupt changes of direction during tracing.

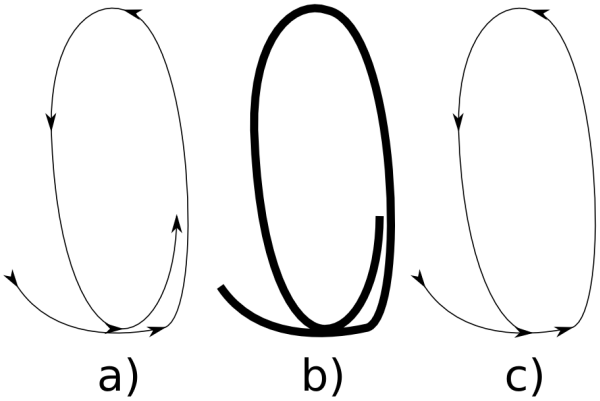

Fig. 5. a) The original pen trajectory used to produce the handwriting. b) The resulting offline image. c) The trajectory as expected to be extracted by the low-level kinetic simulation. Notice that the line end in the middle would not be traced and extracted because the momentum pushes the tracing-point to the outside.

The structure of this Bachelor thesis is as follows: In section II, I will describe the general concepts of kinetic path extraction and explain how the low-level kinetic simulation and high-level kinetic reconstruction make use of a momentum. In section III, I will describe how I tested the low-level kinetic simulation and discuss the results of these exploratory tests. In section IV, I describe the experiment design for testing the cost function of the high-level kinetic reconstruction and the results of the corresponding experiments. In the last section (V), I discuss the implications of my findings and make some suggestions for further research.

II. Kinetic path extraction

I explore two different trajectory extraction algorithms based on the principle of using a pen momentum to retrieve sequential and temporal trajectory data from offline images. I assume that the momentum helps solving ambiguities that occur during the trajectory extraction as, for example, how to continue a stroke at crossings in the image (Figure 4). During a basic thinning-based pen stroke extraction such a crossing would be saved as an ambiguous point because it is not clear which lines belong to each other. The ambiguity would later be solved during the global reconstruction stage. However, when using low-level kinetic reconstruction the used tracing ball would roll straight across such a crossing due to its momentum. This way, it would immediately solve the ambiguity during the basic stroke extraction process. When using high-level kinetic reconstruction, the ambiguity is solved more traditionally during the global reconstruction phase. The used kinetic cost function computes the lowest over-all cost when connecting the crossing trajectories as straight lines. Therefore, both approaches would favor straight lines across the crossing like illustrated by the arrows in figure 4. This solves the ambiguity that arises at normal stroke crossings.

However, there are also cases where two pen strokes only touch each other. In this case the usage of the momentum does not always lead to correct results. Figure 5 illustrates such a problematic handwriting sample that is traced by the low-level kinetic simulation: The tracing point is forced by its momentum to stay on the outside stroke, which leads to an infinite loop.

The high-level approach could prove more valuable in the two strokes touch each other. The search function used during the global reconstruction stage can consider multiple combinations of pen strokes and favor the one with the least *over-all* cost. However, locally, the momentum-based kinetic cost function would still favor connecting the inner curve to the outer curve in the touching point: The function depends on the energy necessary for a transition from one stroke to another and a stroke transition curve with slight bend (like the outer loop curve of figure 5) needs less energy than one with a large bend (like the inner loop curve of figure 5).

Both examples illustrate that the idea of using a momentum for solving ambiguities at crossing-like points of a handwriting might have advantages and disadvantages. I expect that there are more ”normal crossings” than touching strokes in normal handwriting. Therefore, I expect that both proposed ways of using a momentum for online trajectory recovery are valuable. However, this needs to be tested. In the following I will describe in detail how low-level kinetic simulation and high-level kinetic reconstruction work.

A. Low-level kinetic simulation

The low level approach is supposed to work directly with preprocessed image data. It is a replacement for thinning-based pen stroke extraction. It is designed to directly extract pen strokes from offline handwriting data, which are longer than pen strokes that can be extracted by thinning-based extraction techniques.

In order to derive a path directly from an image of handwriting, the image is reinterpreted as a height map. Dark pixels represent deep places in the height map and bright pixels represent high places in the landscape. This way the ink of a handwritten piece of text resembles a canyon in the derived height map. Height values between pixels are interpolated using a linear mixing function of the surrounding pixels. Values outside the image’s area of the image are extruded from the edges of the image. This way the height map is continuously defined.

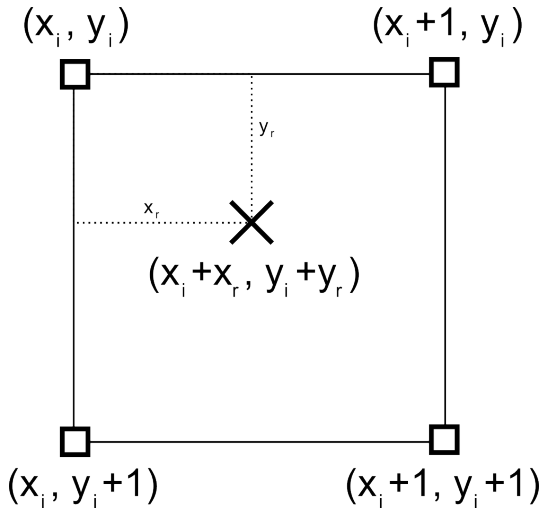

Fig. 6. Situation where a coordinate $(x_i + x_r, y_i + y_r)$ lies between four pixels (x_i, y_i), $(x_i + 1, y_i)$, $(x_i, y_i + 1)$, and $(x_i + 1, y_i + 1)$. The naming is the same as used in equation 6 for inter-pixel height value interpolation.

The linear mixing function calculates for a given coordinate $(x, y) = (x_i + x_r, y_i + y_r)$ with $x, y \in \mathbb{R}$ a height value using the function $h(x, y)$. If a point (x, y) lies between integer pixel coordinates (see figure 6), the height value is calculated as follows:

$$
\begin{array}{l}
x_i, y_i \in \mathbb{N} \\
x_r, y_r \in [0; 1[\\
c(x, y) \in [0; 1[
\end{array}
\tag{6}
$$

$$
\begin{array}{lll}
h(x_i + x_r, y_i + y_r) = & & \\
(c(x_i, y_i)x_r & +(1-x_r)c(x_i+1, y_i)) & y_r+ \\
(c(x_i, y_i+1)x_r & +(1-x_r)c(x_i+1, y_i+1)) & (1-y_r)
\end{array}
$$

The function $c(x, y)$ returns a gray-scale color value for the pixel at coordinate (x, y).

Using the height map that results from the given function, it is possible to simulate a ball that runs through the handwriting-canyon. The tracing ball has a simulated momentum. Its behavior is not accurately simulated but it is approximated by moving a point that has a certain impulse through the canyon. This simplification avoids some complications that would arise from the simulation of a ball with a volume. Most importantly, the simulation of multi-point collisions as illustrated in figure 7 is avoided when only simulating a point.

The simulated ball is placed inside the canyon and given a certain impulse. While the ball moves along the canyon, the ball coordinates ($ball_x(t)$ and $ball_y(t)$ with t as time) are recorded. The sequence of coordinates is used as extracted pen stroke. Thereby, the slant of the canyon is pushing the ball back into the canyon using a simulated gravity (see figure 8). This should prevent the ball from leaving the handwriting. In order to prevent the ball from stopping after being pushed back by a wall of the canyon, the law of conservation of energy is used. The height of the ball ($ball_z(t)$) gives its potential energy and the velocity of the ball ($ball_v$) its kinetic energy. The potential energy of a body is defined by its height times

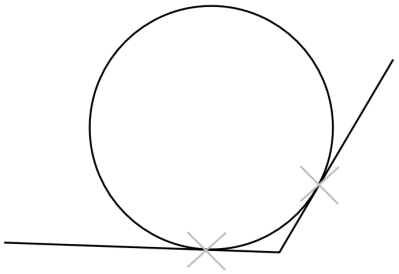

Fig. 7. An example of a multi-point collision of a two dimensional circle touching two lines at once as indicated by the gray crosses. In three dimensions a sphere could touch three planes at once; in special cases even more. These kinds of collisions do require more complex calculations.

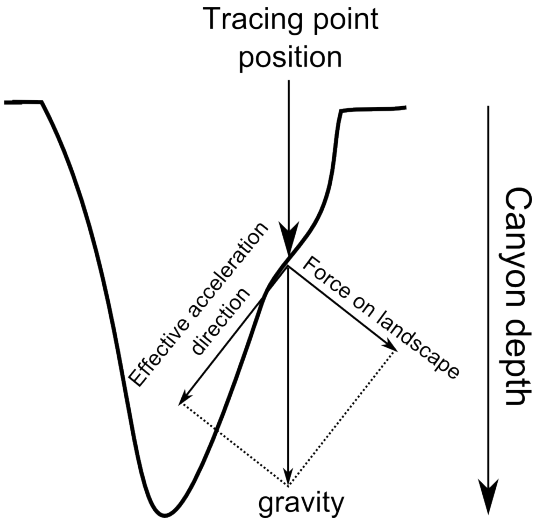

Fig. 8. Cross section of a handwriting canyon with a tracing point in it. The shown parallelogram of forces illustrates how the tracing point is pushed back into the canyon by and the slant of the canyon.

the local acceleration times the body's mass. In the case of the ball it is $E_{potential} = ball_z(t) \cdot gravity \cdot ball_m$. The kinetic energy of a body is defined by the body's speed ($ball_v$) and its mass ($ball_m$) which is for the ball $E_{kinetic} = \frac{ball_v^2}{2} \cdot ball_{mass}$. The sum of both energies is assumed to stay constant during the movement simulation. It is assumed that the ball or tracing-point never bounces and has no friction. It is always attached to the surface of the handwriting canyon. The constant relation is defined by:

$$
\overbrace{ball_z \cdot gravity \cdot ball_m}^{E_{potential}} + \overbrace{\frac{ball_v^2}{2} \cdot ball_m}^{E_{kinetic}} = constant \tag{7}
$$

The mass-factor can be contained in the constant by dividing both sides by $ball_m$. The resulting relation is:

$$
ball_z \cdot gravity + \frac{ball_v^2}{2} = constant \tag{8}
$$

This relation does not require a value for the mass of the ball. The influence of gravity on the tracing ball can therefore be computed without defining a mass for the tracing point. The

influence of gravity includes the deflections from the canyon walls like illustrated by the parallelogram of forces in figure 8.

Equation 8 should also allow the algorithm to extract pen speed information from a trajectory. The narrower a curve is, the more a ball is pressed into the curve by its momentum. It will climb up the slant and thereby get slower. This would cover the fact that narrow curves are drawn more slowly than wide curves [8].

After every line of the image has been traced using this technique the pen strokes must be sequenced to form a whole pen trajectory. This is done in a final global reconstruction stage.

Low-level kinetic simulation presents a new way of extraction pen strokes from offline images. In section III I test if the algorithm works as expected. In the next section I present the different approach of high-level kinetic reconstruction for online data recovery.

B. High-level kinetic reconstruction

The high-level approach uses standard design for an online trajectory recovery system. It consists of the same three stages as described earlier in section I-A (see also figure 3).

The idea is to define a cost function for the global reconstruction stage that is based on the physical principle of the momentum. The cost function calculates the cost (i.e., the energy needed) for moving the pen from the end of one stroke to the beginning of another stroke. This pen-up trajectory is modeled with the function $\vec{x}(t)$ from equation 4. The function $\vec{x}(t)$ uses the mechanical model that describes the movement of a body to which a constant force $\vec{F}$ is applied. By fitting the function $\vec{x}(t)$ to a stroke transition, a value is assigned to the force component $\vec{F}$. Using this force component $\vec{F}$ in equation 3 it is possible to calculate an energy value for the given translation. I assume that the higher the energy for a transition, the less likely it is to be made.

In the following I will describe in detail how the transition energy is computed.

1) Deriving the stroke transition energy: Handwritten pen trajectories consist of curves that are created by varying forces. Therefore, the definition of $\vec{x}(t)$ from equation 4 is insufficient. The most simple model to describe forces that change over time t is a linear function:

$$\vec{F}(t) = \vec{c} \cdot t + \vec{F}(0) \tag{9}$$

The new model of changing forces as a function of time t has to be embedded in the function $\vec{x}(t)$ of equation 4:

$$\vec{x}(t) = \frac{\iint \vec{c} \cdot t + \vec{F}(0)\, dt^2 + \int \vec{p}\, dt}{m} + \vec{x}(0) \tag{10}$$

$$\Rightarrow \vec{x}(t) = \vec{c} \cdot \frac{t^3}{6 \cdot m} + \vec{F}(0) \cdot \frac{t^2}{2 \cdot m} + \vec{p} \cdot \frac{t}{m} + \vec{x}(0) \tag{11}$$

The function $\vec{x}(t)$ of equation 11 is used to interpolate pen coordinates between strokes. The general layout of the new definition of $\vec{x}(t)$ matches the layout of a third-order polynomial:

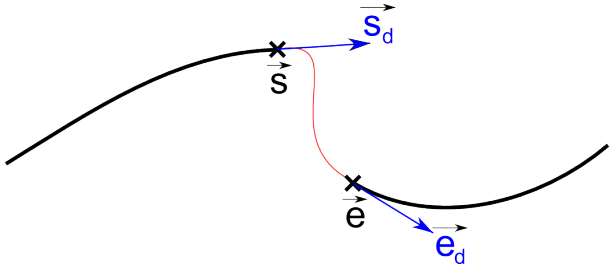

Fig. 9. Values that need to be fit by the third-order polynomial in order to describe the pen-up transition from one pen stroke to the next.

$$\vec{f}(t) = \vec{a_0} + \vec{a_1} t + \vec{a_2} \frac{t^2}{2} + \vec{a_3} \frac{t^3}{6} \tag{12}$$

The following definitions give the relation between $\vec{f}(t)$ and $\vec{x}(t)$:

$$\begin{aligned} \vec{a_0} &= \vec{x}(0) \\ \vec{a_1} &= \frac{\vec{p}}{m} \\ \vec{a_2} &= \frac{\vec{F}(0)}{m} \\ \vec{a_3} &= \frac{\vec{c}}{m} \end{aligned} \tag{13}$$

These relations give that $\vec{f}(t) = \vec{x}(t)$. To compute the transition energy, the function $\vec{f}(t)$ is fit to the stroke transition. There are four values that need to be fit by the function. Figure 9 illustrates the role of each of the values:

$\vec{s}$ and $\vec{s_d}$ describe the end point of the stroke that the polynomial is fitted *from* and the direction of the tangent in that point.
$\vec{e}$ and $\vec{e_d}$ describe the starting point of the stroke that the polynomial is fitted *to* and the direction of the tangent in that point.

The time parameter t of the fit polynomial $\vec{f}(t)$ is defined to be of the domain $t \in [0; 1]$. The values a_0 to a_3 must be fit so that for the function $\vec{f}(t)$ holds that:

$$\begin{aligned} f(0) &= \vec{s} \\ \tfrac{df}{dt}(0) &= \vec{s_d} \\ f(1) &= \vec{e} \\ \tfrac{df}{dt}(1) &= \vec{e_d} \end{aligned} \tag{14}$$

Solving the resulting system of linear equations gives:

$$\begin{aligned} \vec{a_0} &= \vec{s} \\ \vec{a_1} &= \vec{s_d} \\ \vec{a_2} &= 5\vec{e} - 7\vec{s} + 3\vec{e_d} + 2\vec{s_d} \\ \vec{a_3} &= -12 \left(\vec{e} - \vec{s} + \tfrac{\vec{e_d} - \vec{s_d}}{2} \right) \end{aligned} \tag{15}$$

The transition energy is computed using the energy-force relation from (3) and the force function from (9):

$$E = \int_C \vec{c} \cdot t + \vec{F}(0)\, d\vec{s} \tag{16}$$

Applying the $\vec{x}(t)$-$\vec{f}(t)$ relations from (13) gives:

$$E = m \cdot \int_C \vec{a_3} \cdot t + \vec{a_2}\, d\vec{s} \tag{17}$$

From this equation the first important inference can be made: The mass m is only a scaling factor for the final energy value E. Since only energy values produced by this function are compared with each other the mass does not serve an important purpose for the given energy function. Therefore, I will leave out the mass from any following calculations by defining it as $m = 1$.

In the integral part of equation 17, the movement defined by $\vec{s}$ and C is dependent on the curve described by $\vec{f}(t)$ (or $\vec{x}(t)$ since $\vec{x}(t)$=$\vec{f}(t)$). This specifically means that $d\vec{s}$ is dependent on t because the pen speed given by $\frac{d\vec{f}}{dt}$ depends on t. This relation between $\vec{s}$ and t makes the integral in (17) hard to solve. However, it can still be solved iteratively by a computer program. The following pseudo-code defines a function, which computes the integral (note that $t \in [0; 1]$):

```
// During computation a_2, a_3, s and oldS
// contain vectors

function integrateEnergy(a_2, a_3)
  res = 0.0

  for i = 0 to nIntegrationSteps
    t = i / nIntegrationSteps
    s = a_2 * t^2 / 2 + a_3 * x^3 / 6

    if i > 0 then
      ds = s.distanceTo(oldS)
      res += (a_2 + (a_3 * t)).len() * ds
    endif

    oldS = s
  endfor

  return res
end
```

This function is used to compute the transition energy used as cost during the global reconstruction stage. The cost function is tested in section IV.

III. Experiment 1: Low-level kinetic simulation

In this section, I will first describe the testing platform for testing the low-level kinetic simulation which makes use of a tracing point to extract pen strokes. Then I discuss the problems I found with this approach.

A. The testing platform

I programmed a small testing platform to assess the basic capabilities and problems of the low-level kinetic simulation. It loads offline images and tries to trace a single pen stroke somewhere in the image. The constant which relates the height of the tracing point to its speed (equation 8) is adjustable. A starting point for beginning the tracing is searched automatically by looking for a point that lies on the handwriting. The stopping condition for the tracing point simulation is a hard-coded limit of simulation update cycles.

Fig. 10. A part of a height map as it is constructed directly by converting dark points of a scanned signature into depth values. The resulting canyon is relatively rough and has no smooth ground.

Since I early on encountered severe problems with the low-level approach, the testing platform is not capable of extracting all pen strokes of a handwriting image automatically. Due to the problems I canceled the further development of this algorithm. The problems I encountered are discussed in the following section.

B. Problems with the low-level approach

I encountered different problems with the low-level kinetic simulation. In the following, I will describe the two most problematic ones: oscillations and chaotic deflections.

1) Oscillation: Sometimes, when the tracing point moves along the canyon, it seems to loose momentum and stop moving. However, the tracing point should not slow down like this while it is in the canyon. Per definition, the speed of the ball depends on its height in the height map ($ball_z$) as formalized in equation 8, but as long as the ball remains in the canyon, the height should not change enough to cause such a deceleration.

A further analysis of the problem showed that the movement direction of the tracing point slowly changes from moving along the canyon to oscillating between two facing canyon walls. This behavior occurred especially at double traced parts of a handwriting image where the image was very dark and therefore the canyon was very deep.

I found no reliable way of solving this problem. I found a way of damping the oscillations by applying techniques I found in computer-game physics engines. The fix reduced the number of occurrences of oscillations between canyon walls but did not solve the problem. Therefore, I will not further elaborate on the fix.

2) Chaotic deflections at crossings: The biggest problem of the low-level approach is that is does not produce stable

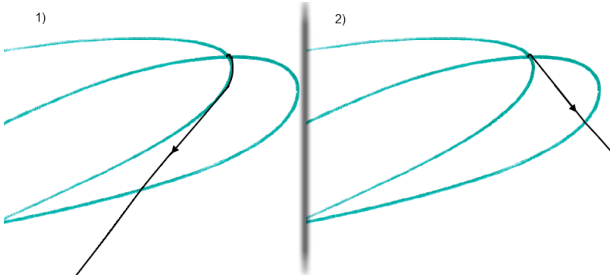

Fig. 11. Chaotic behavior of the proposed low-level trajectory extraction technique. The green line is the original offline data. The black line is the trace of the tracing point. In 1) the tracing point is launched into a certain direction. For 2) the same handwriting image is used as for 1) and the tracing point is launched from the same point as for 1), but in a by one degree different direction. You can see that the small change of direction has a major effect on the resulting trace.

Fig. 12. Two close lines are blurred into each other by the Gaussian smoothing function that is applied to prevent the chaotic deflection of the tracing point by unsmooth landscape. This can lead to problems since the tracing point now has a bridge to the other line which is not a desired effect.

results. The ink of scanned handwriting is not smooth. Scanned ink is textured, which does not lead to a smooth landscape when translated to a height map (see figure 10). The rough ground of the landscape makes the kinetic tracing point deflect chaotically. This leads to problems at crossings. The expected advantage of using a momentum to encourage the tracing point to extract straight lines is not observable. Instead, the tracing point regularly takes erratic turns. Figure 11 illustrates another example of the chaotic behavior of the ball. A small change of the ball's starting direction results in a large change of its resulting trace.

I was expecting this behavior and I had two ideas to deal with this problem. The first idea was to increase the influence of the momentum by decreasing the simulated gravity, which leads to less deflections by the terrain (compare figure 8). This would make the effects of a rough terrain less severe. The other idea was to apply a Gaussian smooth function to the offline images, before starting to trace the lines of the image. This would smooth the landscape creating a plainer ground for the canyon. However, both attempts did not solve the problem.

Increasing the effect of the momentum by decreasing the simulated gravity works well for avoiding the influence of small irregularities in the landscape. However, large irregularities like the "hill" shown in figure 10 still have a major effect on the movement direction of the tracing point. As the tracing point climbs up a large irregularity in the landscape, it gets slower. This means that the tracing point stays longer at the slant of such an irregularity and therefore is deflected even with the low gravity.

Smoothing the handwriting image has the advantage that it smears out irregularities in the image and thereby helps to keep the tracing point at the middle of a pen stroke. The smoothing helps to prevent chaotic deflections from irregularities in the terrain, but also introduces new problems. Figure 12 shows how two theoretically distinct lines are blurred into each other after the application of the smoothing function. This reduces the height of the terrain between canyons, which allows the tracing point to crossover to other strokes, even if pen strokes do not touch in the original handwriting. In order to avoid chaotic deflections, so much blurring is necessary that there are severe problems with lines blurring into each other.

C. Conclusion

In the end, I dropped this approach because there were too many problems, mainly because of the chaotic behavior. Small changes of the starting position of the tracing point could result in completely different extracted strokes. Without having solved this problem, some questions about the algorithm design remain open, for example, how to prevent the re-tracing of already traced strokes. These challenges seemed to be not solvable as part of this Bachelor thesis so I dropped this approach.

IV. Experiment 2: High-level kinetic reconstruction

In this section, I will describe how I tested the kinetic cost function and present the results of these tests. In the first section I will describe the testing platform that was used for testing the kinetic cost function. In the next section, I will describe how I used the testing platform to test the pen stroke sequencing performance of the kinetic cost function. In the following section, I present the results of the tests and discuss them in the final section.

A. Extraction system

The extraction system is the testing platform for the cost function that has been presented for the high-level approach. In order to test whether the proposed cost function works for pen trajectory recovery, it is most logical to test it in a real trajectory extraction environment. Therefore, I implemented a trajectory extraction system that provides for easy exchange of different cost functions. The extraction system is built as a processing pipeline that consists of the five processing steps shown in figure 13. You will notice that the system does not include any noise removal step unlike other extraction systems [4], [9]. Therefore, images that are used with the trajectory extraction system need to be pre-filtered by a background removal algorithm, which was the case for the test set in this experiment.

In the following five sections I will describe every step of the extraction system pipeline shown in figure 13. In an additional section (IV-A6) I will summarize parameters the extraction system uses for online data recovery.

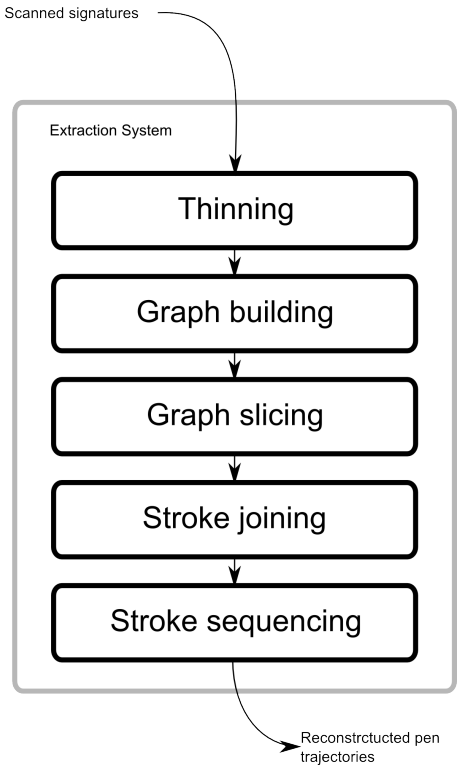

Fig. 13. Processing steps the used extraction system is built of.

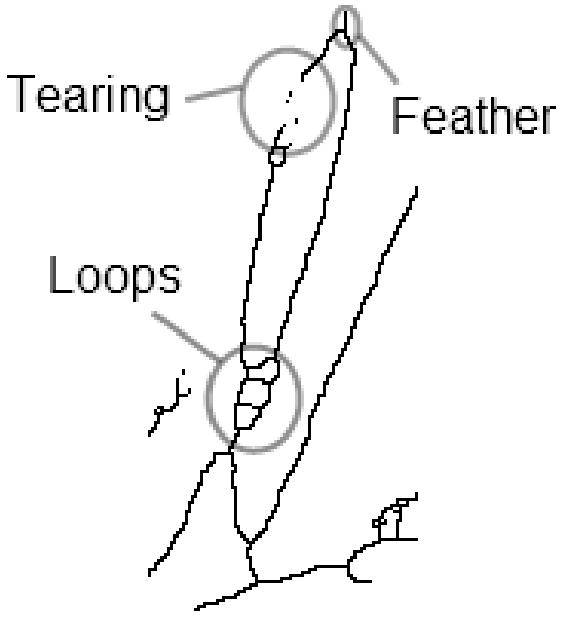

Fig. 14. Thinned image of a handwritten l. The little loops at the line crossing of the l, the additional line piece at the top (called feathering) and the fact that the loop itself is torn apart are typical artifacts that are created by thinning algorithms.

1) Thinning: The thinning processing step of the pipeline incrementally thins lines of a given image until only a one pixel wide lines are left. Thinned images are easy to convert to a line list that describes the skeleton of a given pen trajectory. This technique is widely used in different trajectory recovery systems [10]–[13]. I use a thinning algorithm that is based on the approach described in [14].

Thinning techniques are known to be very sensitive to irregularities in the ink and introduce large scale artifacts in the resulting images [4]. This is illustrated in figure 14. I try to filter out those artifacts in the following graph building processing step.

2) Graph building: The graph building process takes the thinned image as input and turns the lines that are still represented as color in a pixel raster into line representations on the basis of coordinate lists. The graph builder first looks for a starting point in the thinned image that lies on a thinned line. Then builds a graph by beginning to grow branches to all adjacent, colored pixels that are not yet added to any other graph-branch. This is repeated until all colored pixels of the handwriting are covered by the resulting graph. If there are multiple unconnected pen strokes in the handwriting multiple graphs are used to cover all pixels. The following pseudo-code illustrates how this works:

```
GRAPHS = []

for START in getAllInkPixels(thinnedImg)
 if not isUsed(START) then
  markUsed(START)
  GRAPH = new Graph(START)

  ACTIVE_BRANCHES = []

  NEIGHBORS = getAdjacentInkPixels(START)
  for N in NEIGHBORS
   if not isUsed(N) then
    NEW_BRANCH = new Branch(N)
    GRAPH.branches.add(NEW_BRANCH)
    ACTIVE_BRANCHES.add(NEW_BRANCH)
   endif
  endfor

  while ACTIVE_BRANCHES.length > 0
   for BRANCH in ACTIVE_BRANCHES
    NEXT_PIXELS = getAdjecantInkPixels
                    (BRANCH.endpoint)

    if NEXT_PIXELS.length = 0 then
     ACTIVE_BRANCHES.remove(BRANCH)
    if NEXT_PIXELS.length = 1 then
     BRANCH.addPixel(NEXT_PIXELS)
     markUsed(NEXT_PIXELS)
    endif
    if NEXT_PIXELS.length > 1 then
     ACTIVE_BRANCHES.remove(BRANCH)
     for P in NEXT_PIXELS
      NEW_BRANCH = new Branch(BRANCH, P)
```

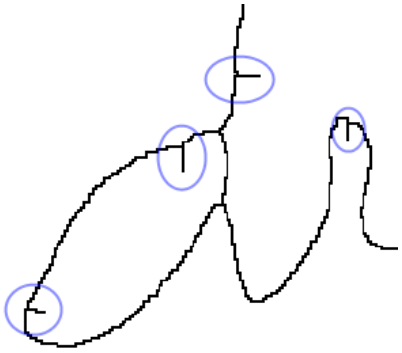

Fig. 15. Part of a thinned handwriting. The circles mark the most common artifact that occurs during thinning which is feathering.

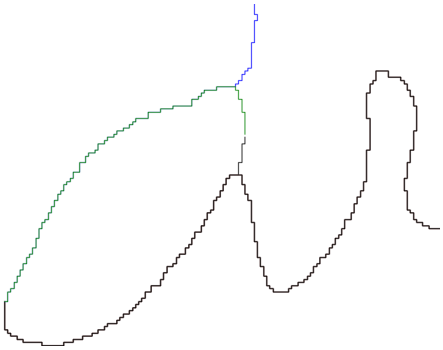

Fig. 16. The same piece of a thinned handwriting as figure 15 after filtering. Comparing to image, 15 you see that this image does not contain any feathers.

```
      ACTIVE_BRANCHES.add(NEW_BRANCH)
      markUsed(P)
     endfor
    endif

   endfor
  endwhile

  GRAPHS.add(GRAPH)
 endif
endfor

return GRAPHS
```

This way all pixels of the thinned image are translated into multiple graph representations. The graphs describe pen strokes pixel-wise as a list of coordinates. Adjacent pixels are connected by chains to form rudimentary line representations. However, the graph representations can still contain different artifacts described in the previous section "Thinning" (see also figure 14).

The thinning artifacts are removed by a filter process that is also applied during this phase. The filter function takes a hard coded parameter which is the ink width (ink_width) measured in pixels. In stead of trying to determine the ink width automatically, this parameter is set beforehand to the specifics of the used scanned handwriting images. The value can be measured manually by using appropriate graphics software given that the ink width of the given handwriting sample is homogeneous. If it is not, the handwriting sample cannot be used with this extraction system.

The ink width is necessary to filter out artifacts that are created during the thinning process, especially feathers like shown in figure 15. The filtering process checks for every branch of the extracted list of graphs whether the line from the root to the given branch-leaf is more distant than "ink width" from every other branch over its whole length. If all points of such a root-leaf line are closer than "ink width" to another branch the checked branch is erased. Image 16 shows the same part of a scanned handwriting as 15 but after the filtering has taken place. All feather artifacts have been removed by the filter process.

3) Graph slicing: The graph slicing processing step cuts the previously derived graph in stroke pieces. The used algorithm cuts the graph derived in the graph building step in every of its branching points. This way only stroke pieces with one start- and end-point remain when this processing step finishes. The coordinates of previous branching points are saved in an array of coordinates of possible line crossings in the handwriting. This array will be used in the following processing step (see section IV-A4) which tries to rejoin continuous strokes at crossings that were sliced during the graph slicing process.

Subsequently, the algorithm re-samples the extracted stroke pieces to abstract from the pixel-wise representation that the graph representation of the previous processing step still encodes. It uses a value k to replace every successive k pixel coordinates of a stroke piece by a single line. The single lines replicate the translations of every k pixel coordinates they do replace. The value k that describes the amount of coordinates that are replaced by single lines thereby is determined dynamically to fit to different stroke piece lengths. To determine k the number of coordinates a path piece consists of N_COORDS and the extraction system parameter $n\_resample$ are used. $n\_resample$ determines the reference value for the amount of coordinates that are re-samples to lines. Given these values, k is defined as:

$$k = \min\left(N\_COORDS,\ \frac{N\_COORDS}{\text{ceil}\left(\frac{N\_COORDS}{n\_resample}\right)}\right) \tag{18}$$

The function $\text{ceil}(x)$ rounds a given value x up, the function $\min(x, y)$ returns the smaller one of the two values x and y. Using this definition for k also pixel-wise stroke pieces shorter than $n\_samples$ are re-sampled correctly by replacing all coordinates they consist of by a single line from starting point to end point. For longer stroke pieces, the reference value $n\_resample$ is used to re-sample a given stroke piece. In this case, it is guaranteed that for k holds that $k \in]n\_samples/2, n\_samples]$. This way the extraction system parameter $n\_samples$ determines the maximum line element length a stroke piece consists of after re-sampling.

After the graph slicing processing step has been applied, extracted stroke pieces look like illustrated in figure 17. The figure shows the same piece of handwriting as 16. You can see

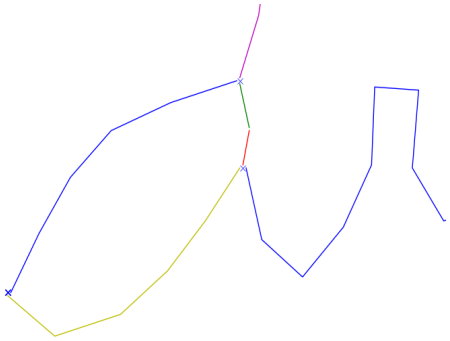

Fig. 17. Stroke pieces as they are represented after the graph slicing processing step. The crosses mark previous branching points used for rejoining continuous strokes during "Stroke joining" phase.

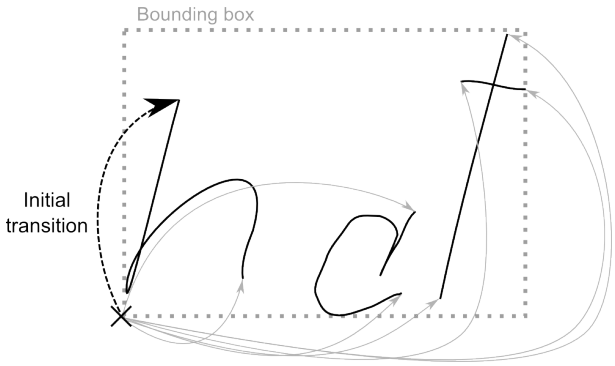

Fig. 18. The initial transition of which the cost is calculated to determine the initial cost for a candidate trajectory starting with the letter h. The gray arrows show all other transitions that are made during the initialization phase of the greedy sequencing algorithm to determine the initial costs of different starting points for the final trajectory.

comparing the two figures how the jagged graphs are replaced by stroke pieces consisting of longer, smoother line elements.

4) Stroke joining: During the path joining processing step, stroke pieces that are close to each other are rejoined to reconstruct connected strokes that have been cut during the graph slicing step. The calculation uses the cost function $c(endpoint_1, endpoint_2)$ that must be specified for running the extraction system. It collects for every crossing that has been found during the path slicing step all ends of stroke pieces that are closer than the ink width ink_width to the given crossing. As long as there are more than two found ends for a crossing, two of the associated strokes are joined together. The joining is different from the stroke sequencing that is done in the following stroke sequencing step. Two stroke pieces that are joined together will be replaced by a continuous new stroke piece. During stroke sequencing processing only the sequence in which stroke pieces are being drawn is determined.

The two strokes that are joined together are determined by choosing the stroke pair with the minimal stroke transition cost at the crossing. If all stroke piece ends that are candidates for a stroke joining are given by the set P, this function can be formalized as follows:

$$(i, j) \in \{(u, v) \in P \times P | u \neq v\} : \underset{i,j}{\operatorname{argmin}}\, c(i, j) \qquad (19)$$

The two strokes belonging to the two found end-points with minimal distance to each other (i, j) are combined to a new stroke piece. The new stroke piece has an added line between stroke i to stroke j. The original stroke pieces are removed. This is repeated until for every crossing there are less than two ends of stroke pieces that can be considered for path joining.

5) Stroke sequencing: The stroke sequencing step is the final part of the processing pipeline. It sequences the extracted stroke pieces to a complete pen trajectory. It is assumed that the stroke pieces that have been found by the previous processing steps resemble the strokes a given handwriting sample consists of.

The sequencing algorithm sequences strokes by using the extraction system's cost function $c(endpoint_1, endpoint_2)$ and the set of extracted stroke pieces. It tries to find a good pen trajectory by minimizing the over-all costs between strokes that is determined by the function $c(endpoint_1, endpoint_2)$. Finding the optimal solution would require to consider all possible combinations of strokes. This is intractable due to the combinatorial explosion caused by multiple available pen strokes (compare [5]). To avoid this problem a *greedy search* algorithm is used which does not find optimal paths, but probably good ones [5].

The greedy search uses $c(endpoint_1, endpoint_2)$ as cost function for calculating the cost of connecting a stroke to another stroke. The set of strokes used for sequencing contains two versions for every previously found stroke: one describes a certain stroke in one direction, the other in the opposite direction. A full reconstructed trajectory may contain, independently of its direction, every stroke only once.

The greedy search algorithm is initialized by using all available strokes as a starting points for candidate trajectories. The candidate trajectories are initialized with an initial cost. That is the cost for the transition from the lower left of the bounding box of collection of strokes to a candidate the start point (figure 18). The cost is calculated using the cost function $c(endpoint_1, endpoint_2)$. The idea of the initial cost is to embed the knowledge in the system that words in Latin words are usually written from left to right. Therefore, giving lower costs to trajectories starting at the left of the baseline of a handwriting seems to be a decent decision for reconstructing a trajectory.

After the initialization phase, strokes are appended to the candidate trajectory that has currently the lowest overall-cost. A stroke is selected for being appended by calculating the transition costs from the end of the candidate trajectory to every start point of all available strokes. The stroke with the lowest transition cost is appended to the candidate trajectory and the trajectory's overall-cost is increment by the transition cost. Then all versions of the given stroke (back and forth versions) are removed from the list of available strokes for the candidate trajectory.

If the candidate trajectory with the lowest overall-cost has no more strokes left that can be appended, the trajectory is completed and returned as the extracted pen trajectory. An example of a extracted trajectory can be seen in the results section of this Bachelor thesis.

6) Extraction system parameters: For the presented extraction system several parameters have to be set. The three parameters that the system depends on are:

1) The definition of the cost function $c(endpoint_1, endpoint_2)$ used for sequencing stroke pieces.
2) ink_width that describes the ink width of the used handwriting images in pixels.
3) $n\_samples$ that describes the reference value for the line lengths that stroke representations are re-sampled to (see section IV-A3).

Note that when using a parameter like int_width, this restricts the domain of offline data that can be used with the system to data with a constant ink width. This means that no handwriting images that were written with different pens and no stretched images where the horizontal ink-width differs from the vertical ink-width can be processed by the system. However, if these restrictions are acceptable the ink-width determination could be automatized in a future version of the recovery system, for example, as described by K. Franke [15].

B. Method for testing the kinetic cost function

The proposed kinetic cost function is tested by comparing extracted pen trajectories to online data that describe the same handwriting samples. By defining a similarity measure between online trajectories and extracted trajectories, different cost function performances can be tested against by analyzing the similarity values.

In the next sections, I will describe in detail how the kinetic cost function is tested. In section IV-B1, I describe the data set I extract pen trajectories from using the proposed extraction system together with the the kinetic cost function. In section IV-B2, I define the quantitative measure I use for comparing extracted pen trajectories to corresponding online data (ground truth). In section IV-B3, I define the cost function that the kinetic cost function is compared to. In the final section (IV-B4), I formalize the hypotheses I have about the distributions of the ground truth comparison values (as defined in IV-B2) for both tested cost functions.

1) Testing samples: The handwriting samples that are used for the experiment are handwriting images of signatures taken from the ICDAR Signature Verification Competition 2009 [16]. For each signature corresponding online data are available. The 1823 used signatures have been recorded by writing with an electronic, pressure-sensitive pen on a paper that was lying on a digital graphics board. This way, online and offline data were recorded at the same time. The online data were recorded at a constant sample rate of 200 Hertz. They consist of a sequence of data-points that are split into pen-up and pen-down chunks. Every data-point consists of X and Y data as position data and Z data that give the pressure that was exerted on the paper.

The offline images are preprocessed, so that it was guaranteed that they only contain pixels that are part of the ink (background-filtering). All other pixels are set to perfect white (RGB = $(255, 255, 255)$).

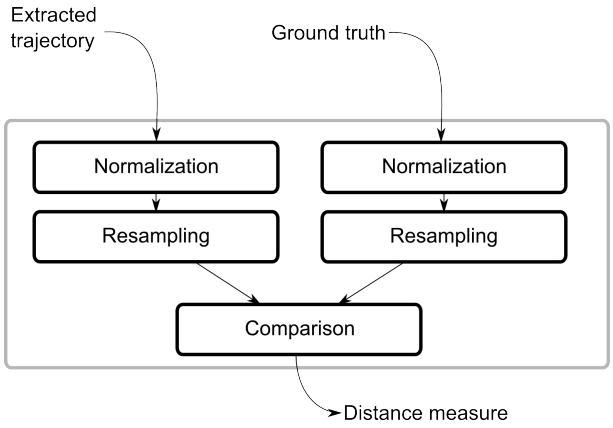

Fig. 19. Pipeline of the algorithm that computes the ground truth distance measure

The online data for a given handwriting image are used as ground truth for extracted trajectories. An extracted trajectory is compared to this ground truth using a similarity measure.

2) Ground truth distance measure: To compare the qualitative performance of different cost functions for path sequencing, a quantitative measure is useful that encodes the similarity between an extracted trajectory and the ground truth. With this measure it can be determined how good a trajectory is, by comparing it to the ground truth.

The similarity value I use is describes the distance between two trajectories. Thus the *similarity* is optimal if the distance is zero. The trajectory distance is computed by a three step algorithm (see figure 19). First the ground truth and the extracted trajectory are normalized. Then they are re-sampled so that the pen-down chunks of both trajectories are represented by the same amount of lines. The final comparison step computes the average distance between every line pair of extracted pen trajectory and ground truth.

The normalization of the trajectories is done to make their coordinates use the same scales. During normalization, the aspect ratio of the original trajectory needs to be preserved. To achieve this, the coordinates of the lines that describe a trajectory are first recomputed to fit in the rectangular space given by the point $(-0.5, -0.5)$ and $(0.5, 0.5)$. Then, the y-coordinates are multiplied with the aspect ratio ($\frac{height}{width}$) of the original image.

During the re-sampling step, the line elements that describe the pen-down parts of a trajectory are collected. All pen-down parts are re-sampled to a representation that consists of 10000 line parts of even length. The direction and sequence of single strokes is encoded in the sequence of the 10000 lines. The re-sampling step thereby removes dynamic information from the ground truth data like, for example, encoded pen-speeds. This is a desired effect because the extraction system does not try to reconstruct this type of data.

The final comparison step calculates the distance value between the normalized ground truth and the normalized extracted trajectory. The 10000 lines of both trajectories are paired by pairing the first line of ground truth with the first line of the extracted trajectory, the second line of the ground truth with the second line of the extracted trajectory, and so

on. The ground truth distance value is defined as the average distance between all these line pairs. The distance between the two lines of one pair is calculated as the sum of the distances between their starting-points and between their end-points.

3) Reference cost function: The reference cost function I am using is the Euclidean distance between two stroke piece end-points. It is a simple cost function that can be found as a part of many other extraction systems [12], [13], [17]. As a a very basic cost function it is used as a baseline test to test the capabilities of the kinetic cost function. Using the notion of the vectors from figure 9, the cost $c_{Euclidean}$ is given by:

$$c_{Euclidean} = |\vec{s} - \vec{e}| \tag{20}$$

This very simple cost function still captures the intuition that close points should be connected to each other because they were probably drawn after each other. However, this cost function does not take any pen direction changes into account for its calculations. Therefore, transitions from stroke pieces that describe harsh changes of a pen's movement direction should more likely be given a low cost than when using the kinetic cost function (equation 17). A major advantage of the kinetic cost function over the Euclidean cost function should therefore be that it not only considers transitions distances but also direction changes of the pen movement.

4) Hypotheses and testing: As discussed it is assumed that the kinetic cost function (equation 17) performs better than the Euclidean-distance based cost function (equation 20). This can be tested by extracting pen trajectories using both cost functions and calculating their ground truth distances using the distance function $d(t, g)$ described in the previous section. t gives a trajectory and g a ground truth trajectory. The computed ground truth distance values for both cost functions can then be compared with each other by using a paired t-test.

$e(i, f)$ is the extraction algorithm that extracts from a certain handwriting image i using cost function f a trajectory. The set of all available test data is set A with $(i, gt) \in A$: i designates a handwriting image and gt is the associated ground truth. $\mathbf{kinetic}$ and $\mathbf{Euclidean}$ refer to the corresponding cost functions. Using these definitions it is possible to define the experiment's hypothesis as follows:

$$\begin{aligned} &D_{kinetic} = \{(i, gt) \in A : d(e(i, \mathbf{kinetic}),\, gt)\} \\ &D_{Euclidean} = \{(i, gt) \in A : d(e(i, \mathbf{Euclidean}),\, gt)\} \\ \\ &H_0 : \mu(D_{kinetic}) >= \mu(\mathrm{D_{Euclidean}}) \\ &H_a : \mu(D_{kinetic}) < \mu(\mathrm{D_{Euclidean}}) \end{aligned} \tag{21}$$

C. Results

In this section, I will present the results of the pen trajectory extraction performances of the two different cost function I test. In section IV-C1, I show the parameters settings for the pen trajectory extraction system that are used for extracting pen trajectories with the two different cost functions. In section IV-C2, I list the observed running times of different parts of the trajectory extraction system's processing pipeline. In section IV-C3, I analyze the histograms of ground truth distances for both tested cost functions. The test results for testing the hypothesis defined in section IV-B4 can be found in section IV-C4. In section IV-C5, three examples of extracted trajectories are shown.

TABLE I
TEST PARAMETER SETTINGS FOR THE EXTRACTION SYSTEM

Parameter	Value
ink_width	10 pixels
$n\_samples$	30
$c(endpoint_1, endpoint_2)$	Kinetic cost function or Euclidean-distance based cost function

TABLE II
MEAN AND STANDARD DEVIATIONS OF THE SIMILARITY MEASURE FOR THE DIFFERENT COST FUNCTIONS

Cost Function	Euclidean	kinetic	kinetic - Euclidean
Mean	0.409	0.371	-0.037
Standard Deviation	0.212	0.227	0.173

1) Parameter settings: Prior to running the tests to compare the two defined cost functions with each other, the extraction system parameters have to be set (see section IV-A6). To determine the ink width ink_width of the used offline images I used the graphics program GIMP and measured the ink width at different spots in different offline images. This way, I obtained an ink width of 10 pixels. Furthermore, I set the value of $n\_samples$ to 30. For the cost function $c(endpoint_1, endpoint_2)$ I used the kinetic cost function and the Euclidean-distance based cost function to obtain different ground-truth similarity values for the extracted pen trajectories. The parameters settings are summarized in table I.

2) Running time: Running the path extraction on the 1823 signature image took about 52 hours on a single PC, partly using dual-core features. Since the extraction program was split up into different parts that were responsible for different stages of the extraction process, the different stages of the extraction process can partly be characterized by their running time:

1) The thinning process including initial graph building and filtering was implemented in a C++ program and took about three hours to run single threadedly on a 2 GHz PC.
2) The path slicing process was implemented in a python-script. The python script was single threadedly on a dual core 2 GHz PC. The process took about 25 hours to finish for all 1823 signatures.
3) The trajectory sequencing and ground truth comparison was also realized with a python-script. It was run single-threadedly on a different 2 GHz machine than for tests before and took a full day to complete. Later on it has been reprogrammed so that it could be distributed across a computer cluster all consisting of the same type of 2 GHz dual core machines. With 26 processes distributed across 13 machines the process of calculating trajectory similarities only took one hour to complete.

3) Dataset analysis: The histograms in figure 20 show the ground truth distance distributions for both tested cost functions. In table II, statistical information about the distance

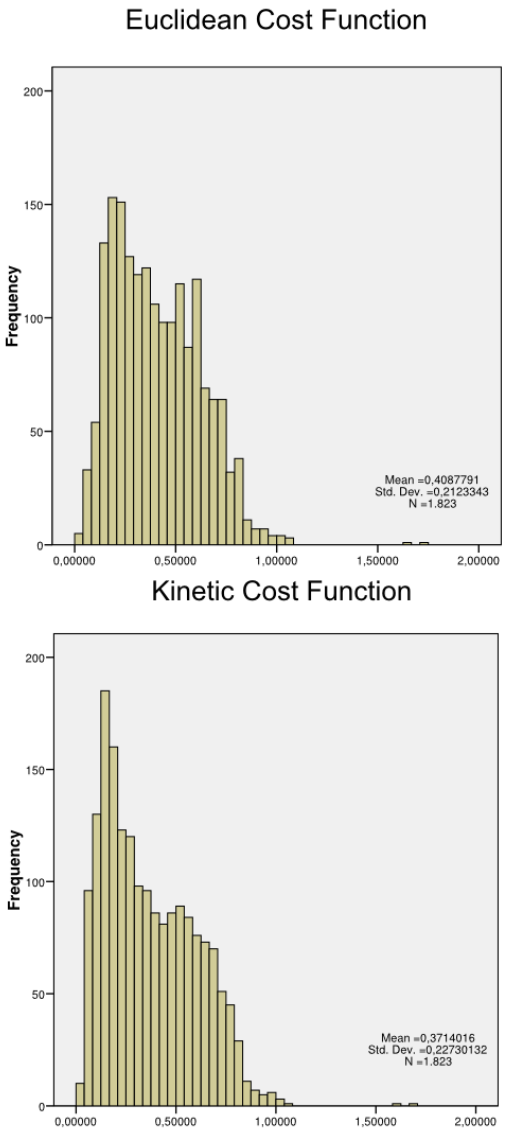

Fig. 20. Histograms showing the distributions of ground truth distance values for the pen trajectories extracted using the given cost function.

values are shown. The mean and standard deviation are calculated using all 1823 signature samples. Also data for the difference between the results of the kinetic cost function and the Euclidean cost function are shown ($kinetic - Euclidean$).

The ground truth distance histograms in figure 20 show homogeneous gamma-like distributions for both used cost functions. Neither of the used cost functions seem to have performed exceptionally bad on a distinct subset of the data samples. Therefore, the data set can be analyzed as a whole.

The trajectories created using the kinetic cost function are less distant from the ground truth than the ones created with the Euclidean cost function. However, compared to the standard deviation, the difference between the means is rather small: The standard deviation is more than five times larger than the difference between the means.

4) Hypothesis testing: The paired t-test on the results of the Euclidean cost function and the kinetic cost functions shows with a t-value of -9.189 that $p < 0.001$. However,

TABLE III
RESULTS OF WILCOXON PAIRED-RANK TEST USING $kinetic - Euclidean$ AS COMPARISON FUNCTION

	N	Mean Rank	Sum of Ranks
Negative Ranks	1102	948.03	1044726.0
Positive Ranks	720	855.59	616027.0
Ties	1		
Total	1823		

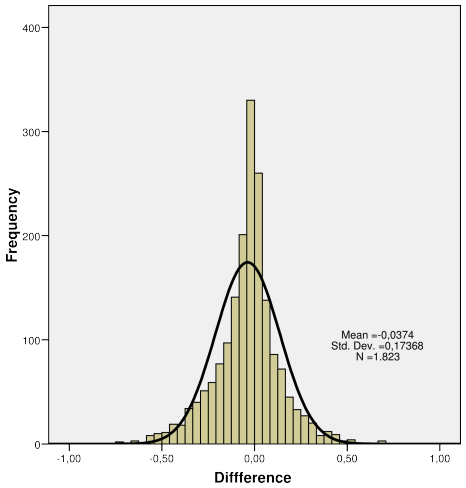

Fig. 21. Histogram of the difference between the ground truth distances of the trajectories generated with the kinetic and with the Euclidean cost function. The used value comparison function is $kinetic - Euclidean$. Compared to the Gaussian distribution (black curve) there is a pile-up of cases around zero.

figure 21 shows that the paired values may not be distributed according to a normal distribution. Therefore, I also applied a non-parametric test for paired samples: the Wilcoxon paired-rank test.

Table III shows data of the Wilcoxon paired-rank test. The resulting z-value for the test is -9.54 that gives that $p < 0.001$. This means that the null hypothesis H_0 (see section IV-B4) is rejected. There is a significant difference between using the kinetic cost function or the Euclidean cost function for trajectory extraction. The kinetic cost function produces on average trajectories which are less distant from the ground truth than the Euclidean cost function, according to the defined ground truth distance measure.

However, compared to the standard deviation of the data the found effect is rather small. When using the t-test we can compute the effect size by computing $\frac{t}{\sqrt{N}} = \frac{9.189}{\sqrt{1823}} = 0.215$. This is a small effect.

5) Sample examinations: In this section I will present general results derived from sample examinations as well as two concrete examples of extracted pen trajectories for both cost functions. One example shows a signature that has better been retrieved by the kinetic cost function, and the other example a signature of the same writer that has better been retrieved by the Euclidean cost function.

By inspecting several samples of reconstructed pen trajectories it appeared that basic pen strokes were retrieved correctly from the offline data by the online extraction algorithm (see figure 22).

Furthermore, it seems that the kinetic cost function and

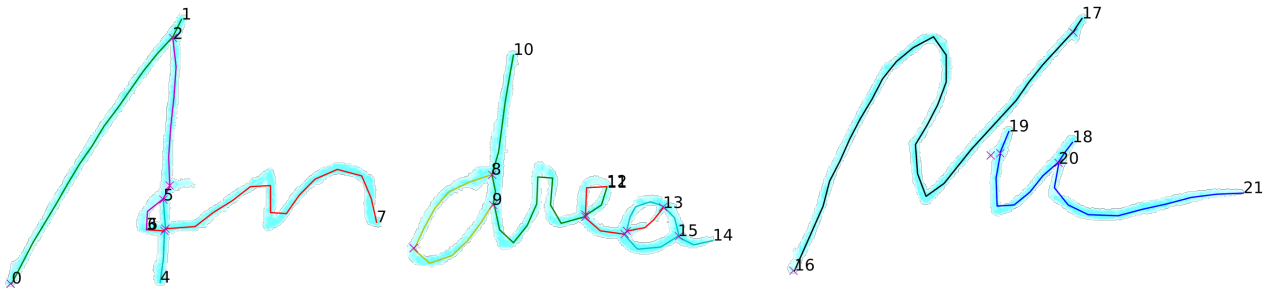

Fig. 22. Extracted pen trajectory superimposed on its offline signature. The online trajectory has been extracted by the described extraction system using the kinetic cost function. The writing order of the strokes is given by the numbers. Even numbers (including zero) determine the starting point of a stroke and uneven numbers the endpoint of a stroke. Note that in the visualization the numbers 3, 4 and 11, 12 are overlapping.
The extracted strokes describe the original offline signature well: All handwriting is covered by extracted strokes. The sequence of the strokes is also reconstructed relatively well but with some exceptions: The recovered pen trajectory writes the letter ”u” (on the right) from right to left which is not the natural writing order and does not match the original pen trajectory.

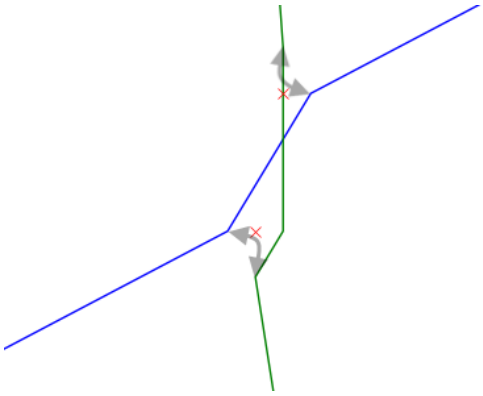

Fig. 24. A closeup of the left crossing of the signature of figure 23. The pen trajectory extracted by using the kinetic cost function is shown. The gray arrows show how strokes would have been connected by the Euclidean cost function.

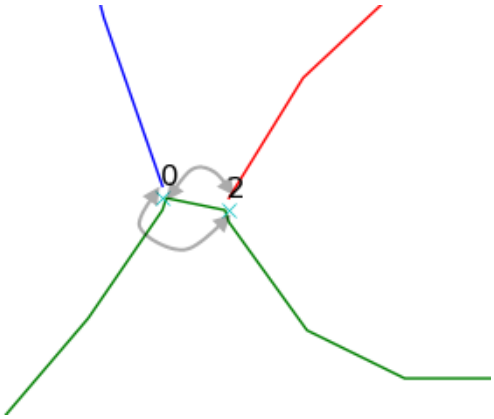

Fig. 26. A closeup of the left crossing of the signature of figure 25. The pen trajectory extracted by using the kinetic cost function is shown. The gray arrows show how strokes would have been connected by the Euclidean cost function.

the Euclidean cost function both connected strokes the same way when stoke transitions are relatively unambiguous. Those unambiguous spots are places where only two isolated stroke ends lie close to each other. In those cases the two stroke pieces were reliably chained after each other as expected. However, there were cases where the kinetic cost function did perform a lot better than the Euclidean cost function and the other way around, even on different signature samples of the same writer.

Figure 23 shows an example where the kinetic cost function performed better than the Euclidean cost function. A closeup of the crossing on the left of the signature shown in 23 can be seen in figure 24. The figure shows a close up of the trajectory extracted using the kinetic cost function and shows how lines are continued due to their orientation. The gray arrows mark how different strokes are connected when the Euclidean cost function is used. For this sample the kinetic cost function works better with a ground-truth distance of 0.020 compared to the ground-truth distance of 0.325 for the Euclidean cost function.

There are also samples of the same signatures for that the kinetic cost function does not compute advantageous costs for the pen trajectory extraction. Figure 25 shows the trajectories for such a sample. The trajectories have a ground-truth distance of 0.723 for the kinetic cost function and a ground-truth distance value of 0.025 for the Euclidean cost function. Figure 26 shows a closeup of the stroke crossing at the left of the signature. The gray arrows show how strokes are connected when using the Euclidean cost function for reconstruction. The Euclidean cost function produces in this case a much better result than the kinetic cost function.

D. Conclusion

There are several conclusions that can be made based on the data presented in the previous section. In the first section (IV-D1), I will compare the general trajectory recovery performance of the kinetic cost function to the performance of the Euclidean cost function. In the second section (IV-D2), I will discuss conclusions that can be made on the basis of the example pen trajectories presented in section IV-C5. In section IV-D3, the testing methods are discussed and in section IV-D4, problems of the used extraction system are discussed.

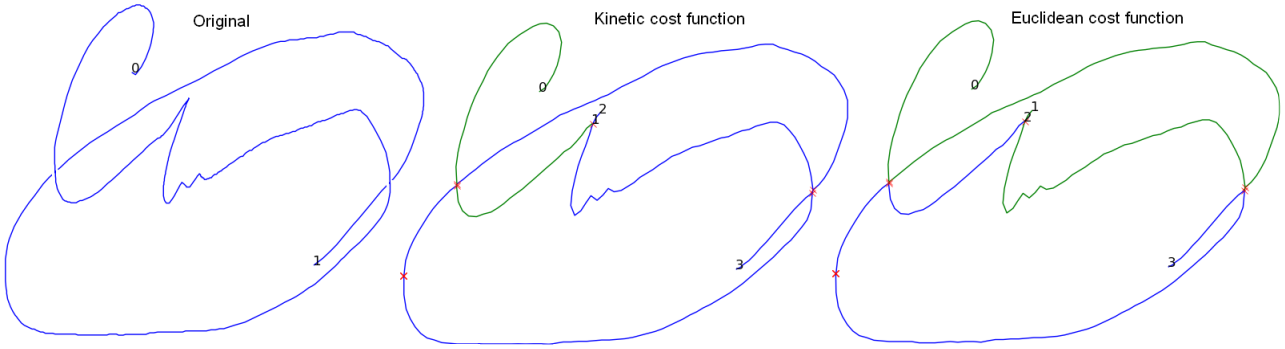

Fig. 23. Original pen trajectory, pen trajectory reconstruction using kinetic cost function and pen trajectory reconstruction using the Euclidean cost function of one handwriting sample. The reconstructed pen trajectory of the kinetic cost function has a ground-truth distance of 0.020 while the reconstructed pen trajectory using the Euclidean cost function has ground-truth distance of 0.325.

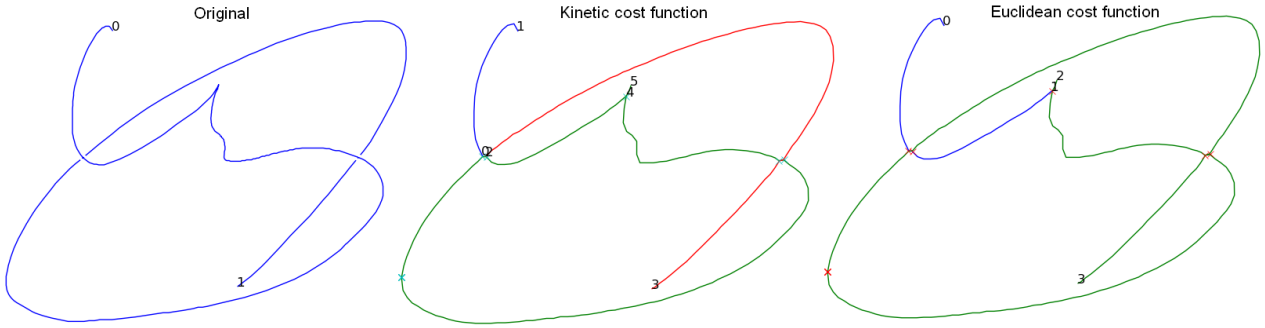

Fig. 25. Original pen trajectory, pen trajectory reconstruction using kinetic cost function and pen trajectory reconstruction using the Euclidean cost function of one handwriting sample. The reconstructed pen trajectory of the kinetic cost function has a ground-truth distance of 0.723 while the reconstructed pen trajectory using the Euclidean cost function has ground-truth distance of 0.025.

1) General cost function performances: Using the kinetic cost function for connecting pen strokes to reconstruct pen trajectories leads to better results than using a simple Euclidean distance based cost function. To consider the change of movement direction at stroke transitions seems to lead to a little improvement of the quality of the final reconstructed full trajectories. Considering the combinatorial problem of finding an optimal sequence for pen strokes and the simplification of using a greedy search to circumvent the problem (see section IV-A5), the little improvement of the quality of the extracted paths is promising result. If the principle of using a momentum-based cost function is used as part of a more sophisticated trajectory extraction system, the quality improvement of extracted pen trajectories could be even higher.

2) Conclusions from sample examinations: Trajectory extraction using the kinetic cost function leads to better pen trajectories than trajectory extraction using the Euclidean cost function. However, the sample examinations of section IV-C5 show that using the kinetic cost function is not advantageous for the extracted trajectories of all signatures. For one of the examples the kinetic cost function performs much better than the Euclidean cost function according to the ground truth distance measure. The other example shows a signature sample where the Euclidean cost function performs much better than the kinetic cost function.

The large difference between the two examples comes from the different choice of trajectory starting points made by the two different cost functions. The stroke sequencing algorithm uses as initial costs of a candidate trajectory the transition cost from the bottom left corner of a signature to the starting point of a candidate trajectory. This means that the kinetic cost function, due to its consideration of the pen movement direction change, favors starting points for pen trajectories that face the lower left corner of a signature's bounding box.

The fact that the algorithm selects a wrong starting point due to stroke-end orientations can be seen in samples of other signatures as well. Reviewing a few samples of the tested handwriting samples suggests that the wrong choice of the starting points of a trajectory has led to many sub-optimal trajectories. The pure spatial distance from the lower left corner of a trajectory's bounding box to the starting point of a trajectory is large compared to distances between strokes. This means that the initial cost of a candidate trajectory has a heavy impact on the over-all cost of a candidate trajectory. Therefore, it is unlikely for a trajectory to get the lowest over-all cost during the greedy search, when it starts with a sub optimal stroke, according to the used cost function. This means that pen trajectories might have been extracted that have an unreasonably high sum of transition costs between paths just because they had a low-cost starting point.

Since I found that the selection of wrong starting points occurs very often, I assume that it also has caused a large part of the variations found in the ground truth distances for both cost functions.

3) Testing methods: It is to note that the comparison of the kinetic cost function to the Euclidean cost function serves as a baseline test for using a kinetic cost function. There are much more advanced cost functions that have been tested in the literature that the kinetic cost function should be tested against. R. Niels and L. Vuurpijl for example used a curvature measure that also uses directional information for path extraction [17].

Another point is that only one quantitative measure was used to assess the quality of the extracted trajectories. This gives only limited insights in how the kinetic cost function outperforms the Euclidean cost function. By using different quantitative values like, for example, the percentage of correctly sequenced trajectories it is possible to explore the strengths and weaknesses of the different cost functions more precisely.

Therefore, it would be desirable to test the kinetic cost function again, but not only comparing it to more advanced cost functions but also using different measures like the percentage if correctly sequenced trajectories to assess the qualities of extracted pen trajectories.

4) Testing platform development: Besides the pure cost-function related results, a further product of this Bachelor thesis is the testing platform which allows to quickly test new cost functions for pen trajectory reconstruction. The cost functions themselves can be easily exchanged as they are implemented in the scripting language Python. However, there are three problems with the system that should be fixed in the future. The problems are:

- The long running time
- The manual setting of the extraction system's parameters
- The bad choice of trajectory starting points

It took one day and nine hours to process 1823 signature samples. Distributing the computations of the extraction system helps to heavily reduce the computation time. Increasing the processing speed would be a desirable improvement for the system since it would allow more test runs in less time.

Furthermore, parameters must be set manually to work with a given set of signatures (The parameters are listed in section IV-A6). Those parameters should be set in a data driven way to make the extraction system more general. For example, the parameter ink_width can be derived as described by K. Franke [15], assumed only offline images with a homogeneous ink-width are used (see section IV-A6). I assume that the reference value for the re-sampling stage $n\_samples$ could also be set automatically depending on the parameter ink_width. The correct relation between the two parameters must be found by further experiments. The extraction system should also be supplemented by an extra noise filter layer to make it work with unfiltered images.

A large problem of the extraction system, as already discussed, is the current way of selecting trajectory starting points. The wrong selection of trajectory starting points might have had a major impact on the data variances. A more reliable way of selecting trajectory starting points needs to be implemented. It could, for example, be tested if the initialization costs for candidate trajectories during global reconstruction can be improved by calculating them from the top-left of a trajectory's bounding box. This could lead to improvements because Latin handwriting is usually written from the top-left to the bottom right [4].

Another way of improving the selection of trajectory starting points is to use the starting points of the ground truth. If only cost functions are compared with each other, it does not matter if the starting point of a trajectory can be determined automatically. Such a system, however, could not be used for offline handwriting recognition anymore because it would rely on online data. Nevertheless, using the ground truth's starting point for a trajectory during global reconstruction should be valid for the purpose of comparing cost functions with each other.

V. Future of the kinetic cost function

The low-level kinetic simulation showed major problems during the implementation phase. To make it work, a solution has to be found to deal with the irregularities in the used handwriting-landscape. Since there are so many open question about how to implement this algorithm to make it work reliably, I do not think that it will be of help for online data recovery in the near future.

The high-level kinetic reconstruction system is built on the basis of a standard design for online data recovery systems. The proposed kinetic cost function leads to better pen trajectories than the very simple Euclidean cost function that is used as a baseline test. This is a very promising result due to the simple construction system that has been used for testing. However, in future research the kinetic cost function should be compared to the performance of more advanced cost functions. It should also be embedded in more advanced trajectory systems to test the quality of its sequenced paths more thoroughly.

To conclude, the kinetic cost function worked better than the Euclidean cost function. Future studies need to show to which extent the proposed method can be used to improve existing trajectory extraction systems. This Bachelor thesis showed that using the physical principle of the momentum to describe pen movements might help to improve offline handwriting recognition systems.

References

[1] R. A. Serway and J. W. Jewett, *Physics for Scientists and Engineers*. Brooks Cole, 2010, ch. 5.

[2] R. Plamondon and S. N. Srihari, "On-line and off-line handwriting recognition: A comprehensive survey," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 22, no. 1, pp. 63–84, 2000.

[3] I. Guyon, L. Schomaker, R. Plamondon, M. Liberman, and S. Janet, "Unipen project of on-line data exchange and recognizer benchmarks," in *In Proceedings of International Conference on Pattern Recognition*, 1994, pp. 29–33.

[4] V. Nguyen and M. Blumenstein, "Techniques for static handwriting trajectory recovery: a survey," in *DAS '10: Proceedings of the 9th IAPR International Workshop on Document Analysis Systems*. New York, NY, USA: ACM, 2010, pp. 463–470.

[5] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*. Pearson Education Inc., 2003, ch. 2.

[6] Y. Kato and M. Yasuhara, "Recovery of drawing order from single-stroke handwriting images," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 22, no. 9, pp. 938–949, 2000.

[7] D. S. Doermann and A. Rosenfeld, "Recovery of temporal information from static images of handwriting," *International Journal of Computer-Vision*, vol. 15, pp. 143–164, 1995.

[8] P. Viviani and T. Flash, "Minimum-jerk, two-thirds power law, and isochrony: converging approaches to movement planning," *J Exp Psychol Hum Percept Perform*, vol. 21, no. 1, pp. 32–53, 1995.

[9] S. Jaeger, "Recovering dynamic information from static, handwritten word images," Ph.D. dissertation, Albert-Ludwigs University Freiburg - Faculty of Applied Sciences, 1998.

[10] V. Govindaraju and R. K. Krishnamurthy, "Holistic handwritten word recognition using temporal features derived from off-line images," *Pattern Recogn. Lett.*, vol. 17, no. 5, pp. 537–540, 1996.

[11] H. Bunke, R. Ammann, G. Kaufmann, T. M. Ha, M. Schenkel, R. Seiler, and F. Eggimann, "Recovery of temporal information of cursively handwritten words for on-line recognition," in *ICDAR '97: Proceedings of the 4th International Conference on Document Analysis and Recognition*. Washington, DC, USA: IEEE Computer Society, 1997, pp. 931–935.

[12] Y. Qiao and M. Yasuhara, "Recovering dynamic information from static handwritten images," in *Frontiers in Handwriting Recognition, 2004. IWFHR-9 2004. Ninth International Workshop*. Graduate School of Information Systems, University of Electro-Communications, Tokyo, Japan, 12 2004, pp. 118–123.

[13] M. Y. Chen, A. Kundu, and J. Zhou, "Off-line handwritten word recognition using a hidden markov model type stochastic network," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 16, no. 5, pp. 481–496, 1994.

[14] R. C. Gonzalez and R. E. Woods, *Digital Image Processing*. Upper Saddle River, New Jersey 07458: Pearson Prentice Hall, 2008, ch. 9.

[15] K. Franke, "The influence of physical and biomechanical processes on the ink trace," Ph.D. dissertation, Rijksuniversiteit Groningen, 11 2005.

[16] V. L. Blankers, C. E. van den Heuvel, K. Y. Franke, and L. G. Vuurpijl, "Icdar 2009 signature verification competition," *Document Analysis and Recognition, International Conference on*, vol. 0, pp. 1403–1407, 2009.

[17] R. Niels and L. Vuurpijl, "Automatic trajectory extraction and validation of scanned handwritten characters," in *10th International Workshop on Frontiers In Handwriting Recognition*, 10 2006, pp. 343–348.