HOCHSCHULE
MITTWEIDA
UNIVERSITY OF
APPLIED SCIENCES

BACHELOR THESIS

Tobias Mann

Supervision of Mechanic Processes Using Computer Vision

2017

Faculty of Computer Science and Biosciences

BACHELOR THESIS

Supervision of Mechanic Processes Using Computer Vision

Author:
Tobias Mann

Course of Studies:
Applied Informatics/Software Development

Seminar Group:
IF13wS-B

First examiner:
Prof. Dr. rer. nat. habil. Thomas Haenselmann

Second Examiner:
M.Sc. Frederic Ringsleben

Date of Submission:
Mittweida, 2017/04/13

Mann, Tobias:

Supervision of Mechanic Processes Using Computer Vision

Mittweida, 2017

31 pages

Hochschule Mittweida (FH), University of Applied Sciences,

Faculty of Computer Science and Biosciences, bachelor thesis, 2017

Abstract

This thesis proposes a solution to the practical problem of supervising relatively basic mechanic processes in robotics by means of computer vision. Supervision happens by comparing the tracked movement with a known, ideal recording of the movement that acts as a model.

First, this thesis analyzes possible approaches to the problem regarding data structures and representation, ways of extracting the data from the recording and ways to compare the data sets of two recordings. Then, a specific solution is implemented in C++ and explained.

Table of Contents

List of Illustrations

List of Tables

1 Introduction

Automation is a highly relevant topic whose importance is yet to increase even further in the coming years. For rather simple, mechanical, repetitive tasks, robotics have long replaced human workers.

However, even automated workers still require supervision, as malfunctions leading to erratic behavior can cause severe issues in highly organized assembly lines. A wide variety of reasons can cause such faulty behavior, ranging from malfunctioning hardware like joints or the unexpected intrusion of other objects to sensors giving wrong readings or the processing code containing mistakes.

For this reason, it is important that a supervisor gathers data from independent sources, in order to be able to judge the accuracy of as many parts of the system as possible. One of those independent sources would be a visual recording of the process, using computer vision to detect and evaluate the movement.

2 Problem Statement

Two main inputs are required to make such an evaluation: a description of what constitutes the ideal case and a description of the de facto case that is to be judged. Both will be recordings in the shape of series of image frames with a set resolution and frame rate. Hereafter, the recording of the ideal case shall be referred to as the *model movement* and the recording of the process that is to be judged for its accuracy shall be referred to as the *judged movement*.

The model movement will depict exactly one iteration of the process in question, from beginning to end. The judged movement may contain any number of iterations and is not required to start or end at any specific point. The resolutions of both recordings can be assumed to equal.

Given those two recordings, the task is to explore the feasibility of various approaches to finding all occurrences of the model movement in the judged movement and deciding which of them are sufficiently similar to the model movement. As requiring perfect equivalence of both movements might not be appropriate in many contexts, the threshold beyond which the difference will be considered unacceptable will need to be config-

urable. That includes both how large the deviation from the specified path of movement is allowed to be and how much the movement may be sped up or slowed down.

After the analysis, one of those approaches is to be demonstrated with a concrete implementation.

As all recordings would be highly limited in scope and show a controlled environment, it can be assumed that they are free of any irrelevant movement that is not either part of the supervised process or an unplanned intrusion that should raise a warning.

2.1 Relevant Problem Cases

Problematic behavior of a machine might manifest itself in a few different ways. For example, parts of it could deviate from their preset path by an unreasonable amount, moving too far in one direction or becoming offset to the side. Even discrepancies that only last for a second should be detectable.

The movement could also stay on the preset path, but differ in terms of velocity. This might be desirable up to a certain degree of variation, for example to account for changing load, but any violation of that threshold should be reported. The change of velocity might again only last for a short time, or it might apply to the entirety of the movement. Sudden, unexpected stops are a subset of this.

3 Possible Approaches and Literature Review

3.1 Requirements for the Extracted Data

As the problem involves recognizing movement, a helpful concept would be that of *optical flow*, as first described by Gibson [1]. Optical flow describes the apparent motion of certain components of a visual scene over time, for example in the shape of a field of displacement vectors for some or all of the subdivisions or features of an image that acts as a frame in a video recording. Each of those vectors would denote the change in location for each subdivision between the given frame and the subsequent one.

There is a variety of existing algorithms that can extract the optical flow between two frames of a video recording [2][3]. Those that attempt to only track the motion of a select few key features are called *sparse*, while those that try to find a displacement vector for every part of the image are called *dense*.

Ultimately, the data that is extracted from the judged movement's recording should serve the goal of comparing it that extracted from the model movement's. Displacement vectors are rather ill-suited to that, as small differences between them would be deemed acceptable by themselves, but could possibly build up to a large error regarding the position of the tracked features or subdivisions over the course of a longer recording.

Therefore it would be more useful to extract and compare the position of the tracked object in each frame, which would not allow such an error to accumulate over time. In order to do that, the moving foreground will need to be split from the unchanging background.

In order to be easily storable and comparable, the location data should be compressed down to a few key points whose coordinates are easy to make calculations with. For example, certain key features or the center of the object's depiction in the image could stand in for the whole shape of the object in each frame. However, it is important that those points are chosen in a consistent way and contain enough information to ensure that the similarity between the model movement's and the judged movement's key points reflects the resemblance between the two movements at a given point in time in a reasonable way.

As the apparent size of internally rigid parts of the object that is being tracked is unlikely to change across different recordings, having one key point for each such part would be a good idea, as it is mostly the position of those parts relative to each other that decides the current configuration of the tracked object. Rotations are an exception to that and would require at least two key points two differentiate between angles of rotation.

If only the shape of the movement is relevant rather than the location of the movement within the frame, all location data could be normalized by recording coordinates in relation to some point within the movement instead of a fixed point like one of the corners of the frame. If the movement should not be limited to a very specific point at any time

but to a more general area, a second model movement could be used. At a given time, the two model movements would describe the acceptable area for a point to be in by spanning a rectangle.

3.2 Approaches to Frame Parsing

Even though it is clear what kind of information should be gathered from the recording, there are multiple ways to achieve that. One option would be to use one of the previously mentioned existing methods for determining optical flow.

For example, the sparse Lucas-Kanade feature tracking algorithm [2], when given a set of points of features, will return the estimated position of those features in the following frame. That return value would therefore be well suited for later comparison. The main problem with this approach is that it requires a fixed set of points before the optical flow analysis can be started. While it is possible to automatically detect corners and edges that are well-suited to tracking, it would not be possible to judge whether those features are part of the foreground or the background. A possible solution to that problem would be to attach markings to the tracked object which would be identified in the first frame of the recording and followed from there.

A second possibility would be using a dense optical flow algorithm, such as the one suggested by Farnebäck [3], in order to separate the moving foreground from the background. In the returned field of displacement vectors, all subdivisions of the frame with a displacement of zero would be considered unmoving. This would yield one or more blobs each representing an object that has moved, the center of which could be calculated and used for the comparison. A lack of movement, both in total and within single parts of the image, could be problematic here and may require reusing the location data of past frames.

The calculations required to find the exact displacement for every part of the frame would be largely wasted however, as for the purposes of this problem it only matters if there was movement at all, not how much of it occurred.

A more simplified alternative to this would be to forego the optical flow calculation in favor of using background subtraction techniques. The simplest option would be calculating the difference in the color values of each pixel between the current frame and the

subsequent one, as for example described by Nishu [4]. A certain threshold for the difference would need to be established, so that noise and subtle changes in lighting don't get interpreted as movement. Applying that threshold to the result of the subtraction would result in a binary mapping of each pixel to either the foreground, if there was a sufficient change, or the background.

The downside to this simple subtraction is that all differences are effectively duplicated, as both the disappearance of the object at the old location and the appearance at the new location count towards them. The center of that shape would describe the interpolated position of the object between both frames.

Areas where the object in the old and new frame overlap could appear to have no change at all. This can easily cause a fragmentation of the shapes describing the position of a single object. As long as only one object is being tracked and the shape formed by the duplication is symmetrical about the axis that is perpendicular to the direction of the motion, this would not cause big issues. For more complex recordings, it would be unsuitable, however.

Instead of simple subtracting two adjacent frames, identification of the background could also be based on more or even all frames, like calculating a rolling average or median. Multiple such methods have been compiled and analyzed by [5]. They generally require more time and storage space to calculate, but do not have the same issues with duplication and fragmentation of perceived changes between frame.

All of the methods based on background subtraction do not cope well with recordings or processes that involve parts moving against each other without losing their general applicability due to specializing on one kind of motion. The lack of predetermined points to track means that the isolation of separate moving objects must be done in real time based solely on whether and how the sections of foreground in the image are connected.

Finally, there is the option of returning to the idea of attaching markings to the tracked object which are identifiable in a frame through properties like brightness or color. This could be done through colored tags pasted to the moving parts of the machine. Their position can then be found by scanning every pixel of the image for the color or brightness value characteristic for the tags used. This leads to a similar binary mapping as background subtraction, deciding whether each pixel is relevant or not. Unlike the map-

ping resulting from background subtraction, however, it can be assumed that every separate blob of pixels marked as relevant stands for a separate object that is to be tracked. Blob size can be used as a threshold to keep noise pixels that incidentally share characteristics with the tags from causing wrong readings.

The downside to a tag-based approach is that a certain amount of preparation and configuration is required – the tags need to be fastened to the machine and their defining characteristic needs to be identified.

On the other hand, once that preparation is done, the object tracking will be very robust. It will be mostly unaffected by noise or other disturbances as long as the tags are still visible and will be able to accurately and consistently track an arbitrary amount of tags.

All of these alternative approaches, except the first one, by themselves result in a binary image. Unless it can be assumed that there is only one moving object in the frame, the blob or connected component that each pixel marked as relevant belongs to will need to be worked out first. This is known as blob extraction or connected-component labeling, multiple algorithms for which have been suggested. For binary images represented by two-dimensional arrays, there are two main approaches.

The first option is to iterate through all rows or columns of the image, using temporary labels to identify parts of the image that have already been found to be connected and merging labels if needed. Label equivalences can either be resolved by repeatedly scanning the image in alternating directions until no further changes happen, or a data structure is used to keep track of labels that are equivalent so they can be fully resolved in a single additional scan. Examples of the latter include the algorithm suggested by Dillencourt et al. [6] and the improved version described by He et al. [7].

The second base approach would be to follow the outline of the object until the starting point is returned to, assigning a label to all pixels inside the traced shape. An example of this would be the approach suggested by Chang et al. [8].

The end result of a connected-component labeling algorithm will be the binary image with every active pixel – in this context, those representing the foreground – annotated with a label that uniquely identifies the internally connected blob it is a part of. Once this is done and all disjoint blobs that describe the same object have been merged, all

labels can be iterated through, averaging the coordinates of every matching pixel for each of them. This results in one or more key points that are stored for comparison.

It might be desirable to additionally label the key points, in order to know exactly which points to compare with which later on. This would require the labeling to be consistent across recordings, which is not possible with any of the background subtraction approaches unless it can be guaranteed that both recordings start with the tracked objects in the exact same configuration. If that can not be ensured, then a key point's location can not be used to deduce the label. A tag-based approach would be able to use different types of tags to possibly infer the label from.

3.3 Approaches to the Comparison of Movements

At the most basic level, a comparison between the data sets representing the movements consists of a lot of comparisons between single points. The main way in which points will be compared is by calculating the Euclidian distance between them, as it most closely models the distance relations of the real world. If an implementation using two model movements is used to allow for more leeway in the exact path of the movement, a special case is introduced: If the position is inside the rectangle spanned by the two extreme points at the given time, then the distance is considered to be zero. Otherwise, the distance to the closest of the two points is used.

The main question is which points are compared. The simplest approach to this would be to start at the beginning of the model and judged movement and to iterate through both at the same speed, comparing the object's positions extracted from the respective frames in each iteration. As both recordings might not start at the same stage of the tracked process, a specific position of the object could be chosen to be used for synchronization. The first occurrence of that position in both recordings would need to be found and the recordings should be offset against each other so they reach that point at the same time.

As it is likely that even very slight differences between the model and judged movement could make the position not match the synchronization position exactly, a certain radius is required within which the similarity between them is considered big enough to count as a match. In turn, this could make multiple subsequent frames with slow movement and few changes trigger the synchronization. This could be solved by waiting until

the differences are larger than the matching radius and then choosing one frame to represent the synchronization, for example the first or last frame to match or the one halfway between the first and last match.

That synchronization position can also be used to tell when the next iteration of the tracked process starts, which would require restarting the iteration through the model movement. It should be taken into account that the synchronization position might be crossed multiple times within a single iteration of the process, however since the model movement represents exactly one iteration, the number of occurrences per iteration can be worked out.

The object's position in the first frame of the model movement would be a good choice as a synchronization position. Ultimately the choice of that point is arbitrary in the context of this comparison, but it should be one that is unlikely to not be crossed due to faulty behavior, as a missed synchronization could potentially cause the comparison algorithm to misinterpret data. Local extrema could also be used, as long as care is taken to not count those that only result from minor fluctuations in the movement by a few pixels, since the placement of those would not be consistent or even predictable.

This approach is not able to deal with movements that are slightly stretched or compressed. Using a synchronization position, the factor by which the movement is stretched on the time axis will only be known once that point has been crossed at least twice. It might therefore be a good idea to first extract the data of one full iteration of the tracked process based on the synchronization positions. Then, the stretching factor would be known before the comparison is started, allowing it to be adjusted accordingly.

A uniform stretching or compression of the movement could be accounted for by changing the speed of the iteration through the judged movement. Since a of change in duration of the movement with a constant frame rate will cause the amount of frames or position data sets in the judged movement to change, a one-to-one comparison between the data sets of both recordings is now no longer always possible. Cases where no matching data set of the other movement is available because of this could be handled either by skipping past them until comparable data from both movements is available again or by interpolating between the two closest frames to estimate what the object's position at that point in time would be.

Skipping would be counterproductive in cases where the stretching factor causes the data sets of both recordings to very rarely coincide in time. There could be long stretches where no comparisons are done at all, leading to an inability to properly detect differences between the movements.

Multiple interpolation strategies exist, such as assuming that there is a linear path between every point and its adjacent points (linear interpolation), trying to find a polynomial function through all points (polynomial interpolation) or finding separate polynomial functions, each of which interpolates between a small set of adjacent points (spline interpolation).

Of those, polynomial interpolation is the least applicable in this situation, as the amount of points that would need to be connected by a single function would be very large, requiring a very high polynomial degree. Linear interpolation would most likely suffice in most situations, as the resolution of the recordings and generally short time difference between frames limit the use of additional accuracy.

As stretching factors can only be determined between two synchronization positions, allowing a difference in velocity only for smaller subdivisions of the movement would require user-defined synchronization positions. This could possibly be done through a graphical user interface. Special error handling would be needed in case a faulty movement misses all or some of the defined points.

As the data set of a frame may contain multiple points, it must be decided how the points to compare with each other are selected from that set. If the frame parsing results in labeled key points and only one point in each of the movements has a certain label, then those can be compared. If multiple points per movement share the same label or don't have any at all, then those two points from different movements that are closest to each other will need to be assumed to be their respective equivalent in the other movement.

Over the course of the comparison, certain metrics will need to be collected, such as the average and maximum deviation between the positions of matched points per frame and in total or the required stretching factor to make the different movements or parts of them match each other. If at some point the amount of key points in a certain frame varies between the movements, that should also be recorded. For each of those metrics, a certain threshold will need to established, beyond which the algorithm con-

siders the difference too great and rejects that particular iteration of the tracked process as faulty. The user should be able to modify those threshold values as needed.

Making these comparisons between the movements in real time, with the judged movement being read from a video feed instead of a recording of known length, would be more complicated, since it requires multitasking to keep reading frames even while making the comparison calculations. If the synchronization position approach is used, all frames from the video feed will need to be temporarily stored until one iteration of the tracked process seems to have been completed. Reading and writing to that temporary storage at the same time could potentially lead to race conditions and other synchronization issues.

4 Implementation of one Approach

4.1 Resources Used

The example implementation has been written in C++ and compiled with GCC 4.9.3 [9]. Throughout the source code, language features from the C++11 standard have been used multiple times, such as lambda expressions or the to_string method as part of the standard library. It should be ensured that alternative compiler versions support those features.

The implementation also uses version 2.4.13 of the third-party library openCV [10]. Its header and library files will need to be included during the compilation process.

4.2 Data Structures

The most basic data structure is called map point and represents a single pixel in a frame, consisting of an X and Y coordinate, which are integers. They are based on a Cartesian coordinate system, with the positive X and Y axes pointing to the right and down, respectively, and the origin being in the top left corner of the image. Map points are an extension of openCV's Point2i data structure, but additionally have a validity flag which can used to denote points calculated based on faulty input values. They also

have a comparison function defined, which allows them to be used as key values in ordered maps.

The movement of a single map point over time is described by the point graph data structure. It essentially consists of an ordered list of map points, with an additional data member describing the time that is assumed to pass between two subsequent point positions, called the time step.

There is also an extended version of a map point, called turning point, that specifically describes a local maximum or minimum in the movement of a point graph. The extrema for movement along the X and Y axis are kept separately. In the current comparison strategy, they are not used.

Similarly, the collection of all map points extracted from a single frame is also an ordered list of map points, but contains no further data.

Finally, the result of a complete parsing of a video recording of a movement is an object movement data structure. It is a sorted list of point graphs, each of which describes the movement of one tracked part of the object. It has data members deciding the time step between positions, which is synchronized with the time steps of the individual point graphs.

Additionally, there is an unrelated data structure representing the difference between two movements. Its data members are the mean and average error across all frames for the X and Y direction each and the maximum difference between the amount of tracked points in each frame between both movements.

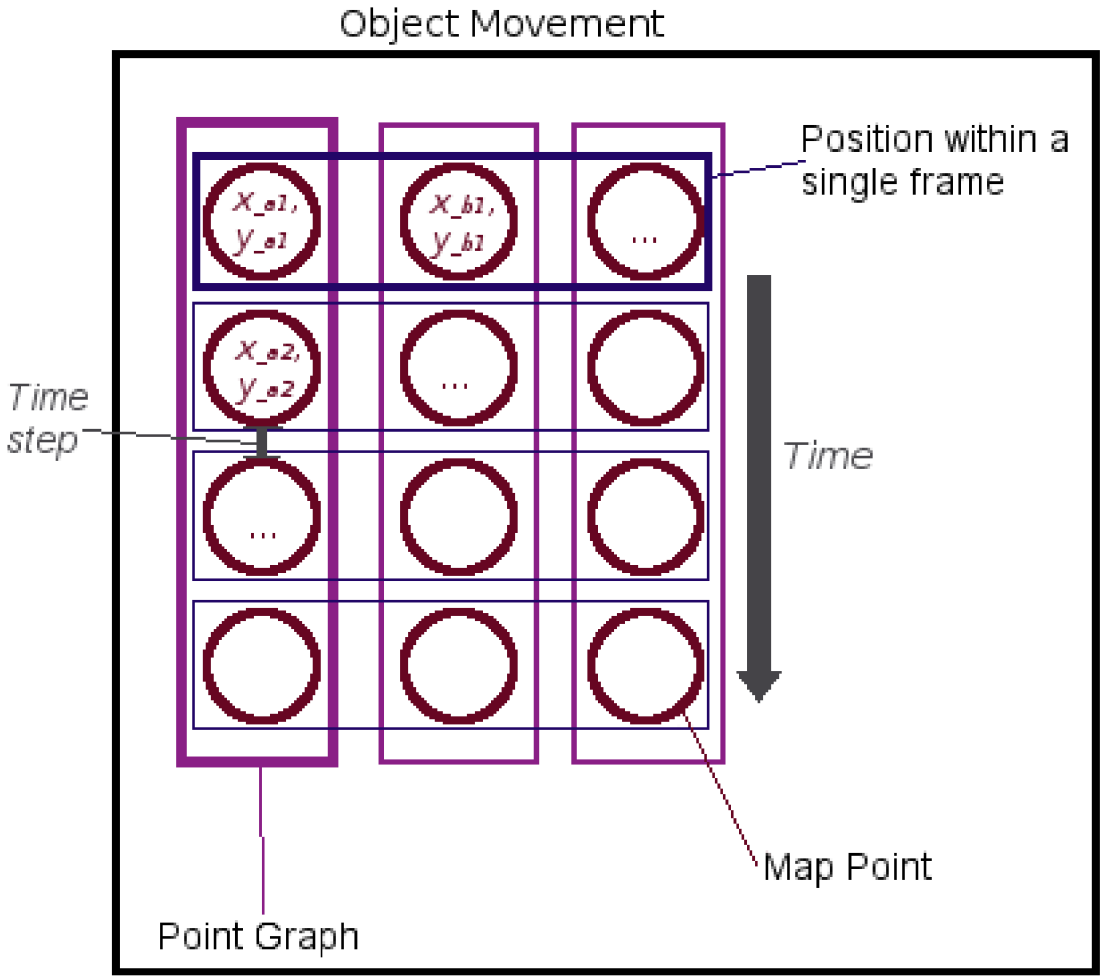

Illustration 1: Main data structures and their relation to each other

4.3 Extraction of Movement Information

The implementation is mainly centered around using colored tags on the tracked device, which are then identified in every frame. While a certain amount of preparation is required, that approach is highly versatile, robust and not very demanding in computing power. Apart from applying the physical tags, the color range that identifies a tag also needs to be specified, this is done through a configuration file. A certain amount of trial-and-error may be required until a highly contrasting color is found.

In order to compliment that approach, the configuration file can also be used to instead enable a method based on background subtraction using a given frame and its predecessor. This is markedly less robust than the default choice, being much more prone to

get disturbed by random noise and becoming less applicable the more complex the tracked object becomes. In this mode, only one key point will be returned per frame, regardless of any other settings. Between two frames, a pixel will be considered sufficiently changed when the differences for each color channel add up to more than a user-defined amount.

Blob extraction, i.e. working out which internally connected component a pixel is part of, is done with a version of a two-scan algorithm as described by He et al. [7]. Temporary labels are applied to active pixels in the first iteration and a data structure is used to record equivalences between labels. In the second iteration, all labels that have been revealed to be equivalent are set to the represented by a single label. In this implementation, the algorithm uses 4-connectivity as the metric to decide whether two pixels are connected.

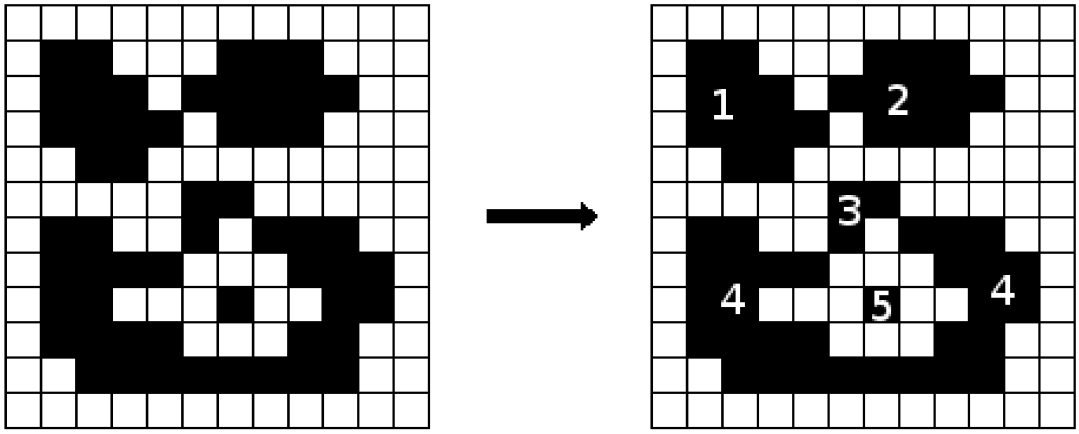

Illustration 2: A binary example image before and after the connected-component labeling process

Since usually only a small part of the frame consists of active pixels, the labeling process does not use a fixed-size structure like a two-dimensional array to keep track of the label of every pixel in the frame. Instead it uses a dynamic mapping of labels to map points, not requiring any space and computation time for irrelevant pixels. In preparation of the next step, the mapping is reversed after the labeling process to allow for easy look-up of all pixels that have been assigned a specific label.

If the colored-tag approach is used, the resulting blobs are then sorted by size and the *n* largest ones are used further, where *n i*s the user-defined amount of points to track in each frame. All smaller ones are discarded. The center of each remaining blob is then

calculated and used as a key point. If the background subtraction approach is used, blobs will instead be discarded if they are smaller than 5 pixels and the center of the pixels of all remaining blobs will be considered the sole key point for the current frame.

That list of key points, which represents the object's position, will be appended to the object movement data structure of the current recording. The appending is done by finding the distances between every key point of the current frame and the previous key point of every point graph, then adding the key point to the point graph that was closest to it. After that, the point graph and key point are removed from the matching pool so they are not used multiple times. This results in a greedy matching without consideration of the previous direction of the point's movement, which might not always be correct for key points very close to each other. However, since the current comparison strategy uses its own greedy matching between the key points of different movements, this is presently not relevant.

4.4 Comparison of Movements

The comparison algorithm uses the starting position of the model movement as sole synchronization position. The judged movement is parsed in its entirety and then scanned for occurrences of that point, allowing for a user-defined radius within which two points count as matching for that purpose. If there is a match with a synchronization position for multiple frames in a row, the one halfway between the first and last matching frame will be used. The object movement is split into subdivisions, or sub-movements, at those frames. The frame representing the matching with the synchronization position will be the first frame of every sub-movement.

The amount of occurrences of synchronization positions in the model movement is counted first. Any such match that extends until the last frame of the model movement will be disregarded, because it will be merged with the match at the beginning of the process. That information is then used to skip the appropriate amount of occurrences when dividing up the judged movement.

Every sub-movement that results from this division is considered one iteration of the tracked process and compared to the model movement individually. The part of the judged movement before the first or after the last synchronization position is discarded, since not enough information is available to assess them accurately anyway. However,

if the length of such a fragment is already greater than the length of the entire model movement multiplied with the maximum allowed stretching factor, then there would be no way for the fragment to be part of an acceptable iteration of the tracked process. The user is notified about this, since it most likely results from unintended behavior.

The duration of every such sub-movement is then calculated, in order to determine by how much it must be stretched or compressed to be as long as and therefore comparable to the model movement. It is calculated as follows:

$$t_{total} = n * (t_{step} - 1)$$

Here, t_{total} is the duration of the movement, t_{step} is the time step between frames and n is the amount of frames the movement covers.

Then the ratio between the duration of the model movement and that of the judged movement is multiplied with the judged movement's time step, thus effectively giving them the same duration. This ratio, the stretching factor, is also stored for later, since outside of a specific range it can cause the sub-movement to get rejected.

Next, every position of whichever movement has less elements is compared to the position of the movement with more elements that happens at that same time. If no data set is available at that time for the short movement, one is interpolated linearly from the two nearest existing ones. Interpolating the longer movement's data sets leads to a lower possible error, because there is less movement between two positions of the longer movement, since it shows the same movement but slowed down.

In a similar calculation to the one that happens when appending a positioning to an object movement, the closest point pairs between the two positions are determined. The deviation within the frame is calculated by averaging the distances between each matched point pair. The total average and maximum deviation across the whole movement are stored in a movement difference data structure, along with the maximum difference in tracked points between the two movements that has occurred.

The sub-movement is rejected as not similar enough to the model movement if any of the parameters in the movement difference data structure or the stretching factor exceed the limits that have been defined by the user, otherwise it is accepted.

Currently, the implementation of comparisons i not designed for real-time application by simultaneously reading from a video feed.

4.5 Application of the Implementation on the Specific Problem Cases

If the movement deviates from the set path by continuing in an unexpected direction, there is a high change of that case being caught. Even if that happened only for a short time, it will reflect in the maximum frame deviation, unless it was so small that it does fall within the tolerance range.

If the movement speeds up or slows down a lot, this will be detected by requiring a stretching factor that is outside the specified range. A change in velocity compared to the model in parts of the movement would almost always be rejected, since the stretching is applied to the whole movement uniformly, which can cause quite high deviation values. If the change in velocity is immediately undone again, it might however still be within allowed limits. This also extends to an unexpected stopping of the movement, albeit the effect will be even more extreme there.

A complete stop of the movement or a deviation so great and long-lasting that the synchronization position is never crossed again will be detected if the resulting fragment at the end of the recording is longer than the longest possible acceptable movement, based on the maximum stretching factor by the user.

5 Usage of the Program

The program is launched by executing the file that results from the compilation process. This should happen in a console window, as otherwise there will be no chance to inspect the results produced by the program since it closes itself as soon as execution as finished.

The program accepts one command-line argument, which is a path to the configuration file. In the absence of such an argument, it will attempt to read the configuration from a file called “config.cfg” in the execution directory. All other information, including which

recordings to parse and compare, is passed to the program through the contents of that configuration file.

5.1 Contents of the Configuration File

The configuration file is a plain text file. Every line contained therein is either a comment or sets a single configuration option to a specific value. All lines that start with the hash character '#' are considered to be comments and have no influence on the behavior of the program. They can be used to document other options or values, or to temporarily prevent an option from taking effect.

All other lines are split on the first occurrence of a space or tabulator character, this excludes any whitespace characters that appear at the very beginning of the line. Everything to the left of that separator is considered to be the name of the configuration option that is being set. Everything to the right of it is considered to be the value.

Depending on what configuration option is being set, the value may be interpreted in different ways by the program. A value may be interpreted as an decimal or hexadecimal integer, a floating point number, a Boolean value or a string. Integer values are interpreted as hexadecimals when the value starts with the prefix “0x” and as decimals otherwise. Floating point values are always assumed to be decimals. Boolean options will be interpreted as *false* if the value is either of “false”, “0”, “no” or empty, any other values are considered as *true*.

Configuration option names are not case-sensitive, but the values generally are, with the exception of Boolean ones.

5.1.1 Parsing Configuration Options

Option Name	Option Type	Description	Default Value
VideoPath	String	Path to the file containing the recording of the judged movement	none
ModelVideo-Path	String	Path to the file containing the recording of the model movement	none
PointCout	Integer	Number of points that are to be tracked in each recording, ignored when option “AutoTrack” is enabled	1
AutoTrack	Boolean	Whether to enable automatic tracking using background subtraction	false
Visualize	Boolean	Whether to display the judged recording on screen along with the location of the key points in each frame so that the tracking can be tested and verified, will not do any comparisons.	false
MaxRed, MaxGreen, MaxBlue	Integer	Upper boundaries for the red, green and blue color values (0-255) of a pixel for it to be considered part of a tag that is to be tracked, ignored when option “Auto-Track” is enabled	255
MinRed, MinGreen, MinBlue	Integer	Lower boundaries for the red, green and blue color values (0-255) of a pixel for it to be considered part of a tag that is to be tracked, ignored when option “Auto-Track” is enabled	0
AutoTrackRel-evantChange	Integer	Lower boundary for the sum of differences between the color values of two pixels for them to be counted as different enough to signify movement, ignored when option “AutoTrack” is disabled	80

Table 1: Parsing options

5.1.2 Comparison Configuration Options

Option Name	Option Type	Description	Default Value
MaxAvgXOffset, MaxAvgYOffset	Double	Maximum allowed average deviation across the entire movement for it to still be accepted, on the X and Y axis respectively	4
MaxXOffset, MaxYOffset	Double	Maximum allowed average deviation within a frame for the movement to still be accepted, on the X and Y axis respectively	7
MaxStretchingFactor	Double	Maximum factor with which the judged movement's time step can be multiplied to match the model movement while still being accepted	1.1
MinStretchingFactor	Double	Minimum factor with which the judged movement's time step can be multiplied to match the model movement while still being accepted	0.9
CollisionTriggerRadius	Double	Maximum average distance in a frame between a synchronization position and another position for the synchronization to trigger, note that this has no effect on anything other than splitting the	2
RunTests	Boolean	Whether to run the small included unit test suite, will not do any comparisons, but can confirm that the comparison functionality works as intended	false

Table 2: Comparison options

5.2 Output of the Program

The program outputs directly to the console and exits as soon as all calculations are finished. Depending on the length and resolution of the recordings, this may take a few seconds. In order to ease the debugging process in case of errors, it will also display what task it is about to attempt next. Those tasks will be, in order, which configuration file it loads, which video files are opened and parsed, and which sub-movements are detected.

After that, a few key figures are displayed for every sub-movement, which are the stretching factor, average error and maximum error within a frame on the X and Y axis each. Depending on whether any of those figures exceeded the limits set in the config-

uration file, the user will be notified that the sub-movement has been accepted or rejected. If the beginning or end fragment's length exceeds that of the greatest acceptable stretching of the model movement, the user will be notified about this and the minimum required stretching factor as well.

6 Conclusion and Future Directions

The provided implementation is able to detect all of the problem cases presented earlier. Despite this, it is currently more of a proof of concept rather than a solution that is ready for industrial use. It serves as a base from which more concentrated effort can be made to improve certain parts.

Implementing a real-time version would be a high-priority next step to make the solution more applicable to real-world problems. This might require using more refined parsing techniques to make sure computation can keep up with the constant stream of frames that real-time use would entail. Currently computation time and the total duration of parsed video material are within one order of magnitude of each other, so this might not always be the case.

The user interface is another area where advancements could be made, as presently the configuration file is the only form of interaction between the program and the user. Given the graphics-based nature of the task, even a command line interface might not serve, instead requiring a fully capable graphical user interface.

References

[1] Gibson, James J.: The Perception of the Visual World. Boston, 1950
[2] Lucas, Bruce D.; Kanade, Takeo: An Iterative Image Registration Technique with an Application to Stereo Vision. Proceedings of Imaging Understanding Workshop, pp. 121 – 130, 1981
[3] Farnebäck, Gunnar: Two-Frame Motion Estimation Based on Polynomial Expansion. Linköping University, Linköping, 2003
[4] Singla, Nishu: Motion Detection Based on Frame Difference Method. International Journal of Information & Computation Technology Volume 4, Number 15, pp. 1559 – 1565, 2014
[5] Piccardi, Massimo: Background Detection Techniques: a Review. University of Technology, Sydney, 2004
[6] Dillencourt, Michael; Samet, Hanan; Tamminen, Markku: Connected component labeling for arbitrary binary image representations. Journal of the ACM, Volume 39, Issue 2, pp. 253 – 280, New York City, 1992
[7] He, Lifeng; Chao, Yuyan; Suzuki, Kenji; Wu, Kesheng: Fast Connected-component Labeling. Pattern Recognition, Volume 42, Issue 9, pp. 1977 – 1987, 2009
[8] Chang, Fu; Chun-Jen, Chen; Chi-Jen, Ju: A Linear-Time Component-Labeling Algorithm Using Contour Tracing Technique. Institute of Information Science, Academia Sinica, Taipei, 2003
[9] GCC (GNU Compiler Collection), https://gcc.gnu.org/
[10] openCV (Open Source Computer Vision Library, http://opencv.org/

Enclosure

Enclosed is a CD-ROM that contains the following:

- a digital version of this thesis
- a directory named “src” with the source code of the implementation
- a text file named “config.cfg” that is a valid and commented example of a configuration file

Declaration of Authorship

I hereby certify that this thesis has been composed by me and is based on my own work, unless stated otherwise. No other person's work has been used without due acknowledgment in this thesis. All references and verbatim extracts have been quoted, and all sources of information, including graphs and data sets, have been specifically acknowledged. This thesis has not been submitted for another degree or diploma at any university or other institute of tertiary education, neither in this form nor in one that is similar to it.

_____