João Miguel Ferreira da Silva

Licenciado em Engenharia Informática

People and Object Tracking for Video Annotation

Dissertação para obtenção do Grau de Mestre em Engenharia Informática

Orientador :	Nuno Manuel Robalo Correia, Prof. Catedrático, Universidade Nova de Lisboa

	Júri:
Presidente:	Prof. Doutor José Alberto Cardoso e Cunha (FCT-UNL)
Arguente:	Prof. Doutor Manuel João Caneira Monteiro da Fonseca (IST-UTL)
Vogal:	Prof. Doutor Nuno Manuel Robalo Correia (FCT-UNL)

FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Setembro, 2012

People and Object Tracking for Video Annotation

To new begginnings.

Acknowledgements

I would like to thank my advisor, Nuno Correia, for a near infinite amount of patience and for the encouragement to submit papers to academic conferences. That work has surely increased the value of this document and has prompted personal development on my part.

A big thank you goes to Diogo Cabral for being a constant help and an amazing brainstorming partner throughout the whole development process. Many, many issues have been solved using his invaluable input.

This work is partially funded by the project "TKB – A Transmedia Knowledge Base for contemporary dance" (PTDC/EAT/AVP/098220/2008 funded by FCT/MCTES), the UTAustin -— Portugal, Digital Media Program (SFRH/BD/42662/2007 FCT/MCTES) and by CITI/DI/FCT/UNL (Pest-OE/EEI/UI0527/2011). We would like to thank the choreographers Rui Horta and Stephan Jürgens for their suggestions and help, and to acknowledge João Valente for his prior work and continued support. A big thank you needs to go to Urândia Aragão, for her help with the new interface elements, and, of course, to Carla Fernandes, the TKB project coordinator, for getting everyone on the same page, and making sure we come together in order to make something great.

Finally, I'd like to thank my family and friends for their unwavering support.

Abstract

Object tracking is a thoroughly researched problem, with a body of associated literature dating at least as far back as the late 1970s. However, and despite the development of some satisfactory real-time trackers, it has not yet seen widespread use. This is not due to a lack of applications for the technology, since several interesting ones exist. In this document, it is postulated that this *status quo* is due, at least in part, to a lack of easy to use software libraries supporting object tracking. An overview of the problems associated with object tracking is presented and the process of developing one such library is documented. This discussion includes how to overcome problems like heterogeneities in object representations and requirements for training or initial object position hints.

Video annotation is the process of associating data with a video's content. Associating data with a video has numerous applications, ranging from making large video archives or long videos searchable, to enabling discussion about and augmentation of the video's content. Object tracking is presented as a valid approach to both automatic and manual video annotation, and the integration of the developed object tracking library into an existing video annotator, running on a tablet computer, is described. The challenges involved in designing an interface to support the association of video annotations with tracked objects in real-time are also discussed. In particular, we discuss our interaction approaches to handle moving object selection on live video, which we have called "Hold and Overlay" and "Hold and Speed Up". In addition, the results of a set of preliminary tests are reported.

Keywords: multimedia, people tracking, object tracking, video, video annotation, computer vision.

Resumo

O seguimento de objectos em vídeo é um problema investigado em profundidade, tendo literatura científica associada, pelo menos desde os finais da década de 1970. Contudo, e apesar do desenvolvimento de vários seguidores de objectos que funcionam satisfatoriamente e em tempo real, o seu uso ainda não se massificou. Tal não se deve a uma falta de aplicações para a tecnologia, já que elas são variadas e interessantes. Defende-se a posição de que este *status quo* se deve, pelo menos em parte, à falta de bibliotecas de *software* fáceis de utilizar, que suportem o seguimento de objectos. Documenta-se o processo do desenvolvimento de uma tal biblioteca. Esta discussão inclui abordagens para a superação de problemas como heterogeneidades nas representações dos objectos, bem como nos requisitos de treino ou de dicas para a posição inicial dos objectos.

A anotação de um vídeo consiste na associação de dados com o seu conteúdo. A associação de dados a um vídeo tem aplicações diversas, incluindo tornar pesquisáveis grandes arquivos de vídeos, ou vídeos longos, e permitir ou facilitar a discussão sobre uma porção do vídeo. O seguimento de objectos é apresentado como uma aproximação válida para a anotação de vídeos, quer automática, quer manual, e descreve-se a integração da biblioteca de seguimento de objectos num anotador de vídeo já existente. Discutem-se os desafios do processo de associar anotações a objectos que estejam a ser seguidos e descrevemos as nossas aproximações à selecção de objectos em vídeo ao vivo. Reportam-se também os resultados de uma bateria de testes preliminares.

Palavras-chave: multimédia, seguimento de pessoas, seguimento de objectos, vídeo, anotação de vídeo, visão por computador.

Contents

List of Figures

List of Tables

Listings

1

Introduction

Object tracking is a much researched field in computer vision. It consists in estimating an object's path throughout a scene. To understand it, it is important to start by defining what an object is, in the context of tracking. An object is, then, anything that is of interest for further analysis [YJS06]. For this reason, the objects to track will be different in each application. For example, they might be vehicles or people in a surveillance video, animals in a wildlife video, for a biologist to review. From this, we can gather that people tracking is a sub-problem of object tracking. Object tracking can be useful for many applications, including:

- automated surveillance, for instance, the detection events that are out of the ordinary;
- automatic video annotation, allowing for searchable video archives;
- motion recognition, for use in human-computer interfaces, including gaming applications and scene-aware video annotation;
- motion analysis, or establishing the motion patterns for the tracked objects.
- automated vehicle driving.

Note-taking is a centuries-old activity, done over various platforms, including pieces of slate and notebooks. Having affordable video capture and processing in modern computing platforms, it is normal to want to extend this process to video, both in order to enhance the video with additional details, for future reference or sharing, and to describe the video or some detail in it, for improved searchability.

1.1 Motivation

Several object trackers are described in scientific literature, going at least as far back as the late 1970s. Despite this, and the development of some satisfactory object trackers, such as, for instance [YLS04, HHD00, SFC+11, KMM12], it has only seen limited real-world application, despite the recent and significant inroads made with the release of the Microsoft Kinect Sensor and the availability of programming interfaces for it, first in OpenNI, and then in the official Kinect SDK[1].

Meanwhile, in recent years, since affordable digital video cameras, bountiful storage capacity, and widespread high-speed Internet access became available, the amount of video content on the Internet has seen an immense growth, driven by the popularity of video sharing websites, such as YouTube or Vimeo.

Video annotation can play an important role for facilitating this task, by describing the video, in a format that current search engines can understand. In addition, and more relevantly for this work, it can also facilitate the process of analyzing and commenting the video, by allowing users to share and archive their thoughts on the video, as expressed by their annotations. This work is focused on said analysis and comment.

Modern-day Tablets offer the possibility of adding handwritten notes to any media document [BM03]. Recent developments in this technology made their size similar to a regular sheet of paper, their computational power comparable to that of a regular laptop, and also reduced their weight, thus making current Tablets more suitable for mobile use. Tablets provide different input modalities, such as touch, pen or physical keyboard. The combination of these different features (mobile affordances, pen and touch inputs and cameras) can foster the Tablet's usage to add notes to content captured by the cameras (images or video), working as digital multimedia notebook [CC09]. In the case of still images a simple overlay of the annotations is sufficient to maintain its association with the content, whereas in the case of video, with all its moving features, the solution is not so simple and direct. The same solution for still images can be adopted but tracking methods have to be added, in order to maintain the association between the annotation and the annotated moving element [CC09]. An important task of annotations is serving as an anchor between a point of interest of any media document and additional information [Mar98, MFAH+11]. Therefore, video annotations can also work as "anchors" by linking moving objects to other annotations. In this work, we refer to an object anchor as a graphical element, drawn over the tracked object. We have chosen colored rectangles as our anchors. These anchors allow the user to attach further annotations onto them, which causes those annotations to follow the object around the scene, thus maintaing their context.

Additional challenges are presented when annotating on live video. Since the user is annotating moving video features, an annotation may lose its relevance in regards to a

[1] OpenNI is available at `http://www.openni.org` and the Kinect SDK can be found at `http://kinectforwindows.com`.

video feature between the time when it starts, and the time when it is finished. Object tracking is very relevant in this context, since it can be used to create an anchor for a video feature, onto which additional annotations can be attached. Low-level computer vision primitives are available in Intel's OpenCV library[2], but it is far from trivial to form a robust object tracker from them. A high-level software library for object tracking is, then, desirable, but does not exist. This is no coincidence. Object trackers vary wildly in what kind of objects they are designed to track, how they represent these objects' shape and appearance, which means that there are difficulties involved in creating a common software programming interface for them.

1.2 Problem description

Object tracking consists of following an object's location throughout the course of a scene. This can be difficult, due to numerous factors [YJS06]:

- loss of information about depth, in the process of capturing the image[3];
- image noise;
- objects with unpredictable motion patterns;
- full or partial object occlusion;
- complex object shapes;
- changes in scene illumination;
- real-time processing requirements.

There are significant challenges involved in creating a high-level framework, providing a common programming interface for object trackers. These include the variability of object shape and appearance representations, trackers' differing requirements in regard to training data or knowledge of an object's initial position, and different expectations on the number of cameras needed for trackers to work.

As for video annotation, the main concerns are the definition of a data model and user interface design. In general, it is preferred that the video's annotations and metadata are stored separately from the video itself, to avoid copyright issues with the video [AP05]. Video annotation interface design can be tricky, since the user annotating the video needs to view two sets of time-dependent data: the video stream, and the associated annotations. In addition, annotation display should not obscure relevant parts of the video's data, even if this can be alleviated if the annotations are editable and created by the user. Object tracking can also aid manual annotations, allowing them to retain their context [CVS+11], as well as serving as anchors for further annotations or comments.

[2] OpenCV can be found at `http://opencv.willowgarage.com`.

[3] This problem can be alleviated by using multiple cameras [BK99, KCM04] or depth cameras [SFC+11].

Even after establishing that tracked objects serve as anchors to further annotations, some problems remain. For instance, if the object trackers require a hint to the objects' initial position, the user needs to specify it. In our application, this is done by drawing a rectangle around the object. When this is done on a live or playing video stream, those objects are moving, and are, thus, difficult for a user to select [AHFMI11]. The same problem applies later, when creating annotations to be attached to that anchor.

1.3 Main contributions

The main contributions for this thesis include, then, the development of a high-level object tracking framework[4], presenting a common interface for tracking that:

- Can use varied object representations;
- Supports both single, and multi-camera trackers;
- Supports tracking algorithms that need prior training or not;
- Supports tracking algorithms that need or do not need a hint to the object's initial position.

In order to test this framework, a sample tracker, based on FAST features [RD06] and BRIEF descriptors [CLSF10], was developed and three others (CAMSHIFT [Bra98], Kinect [SFC^{+}11] and TLD [KMM12]) were adapted so they can be used from it. In the process of creating this work, some bug fixes and slight performance improvements were also contributed to Georg Nebehay's fork of OpenTLD[5], an improved version of the TLD tracker.

The other main contribution of this work is the integration of this library into the Creation Tool [CVS^{+}11, CVAa^{+}12], a video annotator previously developed by João Valente [Val11]. This integration serves both as a test to the library's design, and as an opportunity to explore the unique user interface challenges involved in annotating video with tracked objects in real-time. In particular, the annotation approach we have decided on was to have anchors to moving objects on the scene, onto which further annotations can be attached. In order to create, select, and attach annotations to those anchors in real-time, we have also developed two approaches to moving object selection on live video. These approaches are based on the one presented in [AHFMI11], and we have called them "Hold and Ovelay" and "Hold and Speed-Up".

Both the Creation Tool and this work are being done in the context of the TKB project, funded by FCT (Fundação para a Ciência e Tecnologia). The Transmedia Knowledge Base (TKB) for contemporary dance is an interdisciplinary project, coordinated by Dr. Carla

[4] The source code is available at `https://github.com/jmfs/libobtrack`.

[5] The code is available at `https://github.com/gnebehay/OpenTLD` and the author's improvements are described in [Neb12].

Fernandes, from the "Faculdade de Ciências Sociais e Humanas da Universidade Nova de Lisboa". Its objective is the creation of an open-ended multimodal knowledge base to document, annotate and support the creation of contemporary dance pieces.

1.4 Published work

Papers detailing the development of this work have been published in a few conferences, such as an extended abstract in the 2011 ACM SIGCHI Conference on Human Factors in Computing Systems (CHI EA '11) [CCS+11], a short paper on the 2011 ACM Multimedia Conference (MM '11) [CVS+11] and a full paper on the 2012 International Conference on Mobile and Ubiquitous Multimedia (MUM 2012) [SCFC12].

1.5 Document outline

In Chapter 2, related literature on object and people tracking is presented and analyzed. In Chapter 3, the related work on video annotation is presented. Later, in Chapter 4 we discuss the challenges and solutions required to build a generic, high-level, object tracking framework and then, in Chapter 5, we discuss the process of finding whether it was feasible to attach annotations to tracked objects, and then the integration of the tracking framework into the Creation Tool, along with the user interface challenges it posed. The test and performance results of our proposed solution are then discussed in Chapter 6. Finally, in Chapter 7, we take conclusions and offer possible new directions for further research.

2

Object Tracking Literature Review

In order to maintain the association between annotations and the annotated objects, object tracking methods are used in order to create anchors for the objects of interest, onto which further annotations can be attached.

In this chapter, an overview of several problems pertaining to object and people tracking, along with previous proposed solutions in the literature. The problems regarding object detection and how to represent objects are introduced, followed by the presentation of several proposed object trackers, grouped in categories. Along the way, the image features that object trackers generally use to keep track of the objects' position are described. Section 2.4 introduces people tracking as a sub-problem of object tracking and examines ways of dealing with the problem, as found in the scientific literature.

The problem of object tracking, as well as several approaches to solving it, are neatly summarized in Yilmaz, Javek and Shah's "Object Tracking – A Survey" [YJS06], even though there is, of course, a need to delve deeper into some of the subjects therein.

To effectively perform object tracking, one needs to perform three key operations: object detection, object tracking and track analysis. The first one consists of detecting the position of one or more objects, in a frame. Object tracking aims to decipher an object's trajectory over time, in every frame of the video. Finally, track analysis aims to extract meaning out of the object tracks, for example, to conclude what type of motion a person is performing, based on that person's track.

Before looking into these however, it pays to know how the objects can be represented in memory, so one can better understand the output those approaches produce and how to work with it, in order to produce tracks, or to perform higher-level analysis.

2.1 Object representation

Objects can be represented by their shapes, appearances or both. Let us, then, look at some common object shape representations for object tracking.

Points: the object is represented by its centroid [VRB01] or by a set of points. This representation is especially suitable for small objects in the scene.

Primitive geometric shapes: the object's representation is a rectangle, ellipse, or some other shape. It is mostly used to represent rigid objects.

Contour or silhouette: a shape around the object's boundaries is called a contour. The contour's interior is the object's silhouette.

Articulated shape models: The object's body is represented by several parts, each of which is modeled by a geometric shape, and joints, responsible for both connecting body parts, and constraining their movement.

Skeletal models: A model of the object's skeleton is used. This approach can, therefore, only model rigid, and articulated objects.

Several representations for an object's appearance exist. Yilmaz et al. [YJS06] enumerate the following as relevant for object tracking:

Appearance probability density: The object's appearance is estimated by a probability density function of some of its visual features, such as color or texture. These features can be computed from a region corresponding to the object's shape model. Comaniciu et al. [CRM03], for example, use a color histogram to model the object's appearance.

Templates: Templates are a combination of one or more image regions, such as shapes or silhouettes [FT97]. They include information about each region's appearance and, possibly, relationships between the appearance of all regions. Hence, templates store information on an object's shape and appearance. However, they only do so for a single camera angle.

Active appearance models: active appearance models encode information about an object's shape and its appearance [CET98]. They need to be trained to recognize an object's shape and appearance, by processing a set of images containing that object. These images are annotated with the location of the object's landmark points, thus modeling the object's shape. The appearance is modeled by analyzing image features, such as color or texture[0], in the points' neighborhood.

Multi-view appearance models: as their name implies, multi-view appearance models encode information about an object, in multiple views. The models need to be trained by several sample images, in order to extract the image set's principal components [BJ98]. The use of multi-view appearance models requires, therefore, prior sample images of every view of every object to be detected on the scene.

Usually, object representations are chosen based on the application domain. For instance, point representations are adequate in situations where the objects one wishes to track are small, like seeds, as in [VRB01] or birds in the sky, as in [SS05]. A rectangular or elliptical representation will do, if the object's shape can be reasonably well approximated by that shape. Comaniciu et al. [CRM03] use ellipses as their object shape representation. Contour and silhouette representations are best for objects with complex shapes, such as humans, or non-rigid objects. As an example, Haritaoglu et al. [HHD00] use a silhouette representation in order to track humans, in a surveillance application.

As we can see, there is a huge diversity in object shapes and appearances. This makes it difficult to create a common programming interface for trackers that use different ones, or even for trackers that combine several of them.

Let us now look at how to actually build these object representations.

2.2 Object detection

In order to perform object tracking, one must first detect objects in a scene. Depending on the approach used, this must be done either on every frame, or on the first frame the object is present [YJS06]. It is common to use a single frame for object detection, but a sequence of several consecutive frames can be used in order to improve the detector's reliability. There are several approaches to object detection, which can be grouped into four main categories: interest point detectors, background modeling, image segmentation and supervised learning classifiers.

2.2.1 Interest point detectors

As their name implies, interest point detectors are responsible for detecting points of interest in an image. What makes a point interesting depends on the particular algorithm used but, in general, an interesting point is one that can be located unambiguously in different views of a scene [Mor81]. This makes corners particularly interesting. The notable algorithms for interest point detection used throughout the course of this work were the Scale Invariant Feature Transform (SIFT)[Low99], Scaled Up Robust Features (SURF) [BETVG08], and Features from Accelerated Segment Test (FAST) [RPD10].

The SIFT detector, first proposed in [Low99], works by first scaling the input image by a factor of 2, using bilinear interpolation, and then applying two, differently parametrized, Gaussian filters to the image. An image, corresponding to the difference between Gaussians, is created. The images are, then, shrunk progressively, using bilinear interpolation, and the filters are applied and Differences of Gaussians (DoGs) are calculated for every scale. This is called a scale pyramid, with the smaller scales being on top. Pixels with maximum or minimum intensities, relative to their eight neighboring pixels, are extracted, at every scale. Interest points, or SIFT keys, are the pixels that are maxima or minima across all scales, thus assuring that they are invariant to scale. SIFT then proceeds to calculate the gradient's magnitude and orientation of the pixels in a neighborhood of the interest point, in order to build an orientation histogram. The peak of this histogram is considered the main orientation of the interest point, and further interest points are created in the same spot, with orientations that have values within 80% of the peak. Lowe has further developed SIFT [Low04], with the main developments being the application of more than two Gaussians to the function, thus generating more DoG images per scale; a greater image shrinkage, in every step of the pyramid; the maxima and minima calculation also being done across scales; and, finally, the elimination of less interesting points, located in areas of low contrast or along edges. SIFT does not, unfortunately, run in real-time.

Mikolajczyk and Schmid [MS05] have evaluated the performance of several interest point detectors, concluding that SIFT-based methods and, in particular, their own variation of SIFT, Gradient Location and Orientation Histogram (GLOH), perform better than any of the other algorithms.

Since this evaluation, the SURF detector has been developed by Bay et al. [BETVG08], and its authors claim that it performs better, both computationally and with regard to its results' quality. It works by first convolving the image with 9×9 box filters, which approximate the second derivative of a two-dimensional Gaussian. In other words, they approximate the components of the Gaussian's Hessian matrix. Scale invariance is achieved by filtering the original image with progressively bigger masks, also using bigger sampling intervals. For each sampled pixel, in each scale, the determinant of this matrix is calculated and interest points are the maxima of a $3 \times 3 \times 3$ window (a 3×3 window, across 3 scales). Although much faster than SIFT, SURF is still not fast enough to run in real-time, with the authors reporting times of $610ms$ for detecting the points and creating the descriptors for a single image.

Besides detecting interest points, SIFT and SURF are also full-blown point trackers. The points are tracked by creating descriptors for them and then matching these descriptors across frames.

The FAST and FAST-ER [RPD10] detectors classify a pixel as a corner if, in a circle of 16 pixels around it, at least n contiguous pixels are lighter or darker than it, by some threshold t. FAST-ER improves on this by deforming the image and only classifying the pixel as a corner if the same holds through all the deformations. These methods are very computationally efficient.

2.2.2 Background subtraction

Background subtraction works by building a model of the scene's background, and then marking every pixel that does not fit the model as a candidate for further processing. Finally, these pixels should be connected to obtain regions, representing the objects [YJS06].

The first challenge to address is, then, to build the scene's background model. This can be problematic, due to a number of reasons, namely: changes in illumination, image noise and camera or background motion.

Although it is possible to simply calculate the difference between frames and threshold that difference to obtain the moving regions, this approach is not satisfactory, since the image is subject to noise, and there may be background motion. These problems may be addressed by building a probabilistic model for the background's appearance. Wren et al. [WADP97] model the background with a 3D Gaussian for each pixel, representing the probability densities for the three color components, Y, U and V. The Gaussian's mean and covariance are learned from the observed colors, at that pixel. A mixture of Gaussians may also be used to model the background [YJS06]. It is used when a single Gaussian cannot account for all the variability in the background. This is especially important in outdoor scenes, since the background can change significantly. In addition, different models can be built for different image regions.

In this work, we have tried to use the background subtraction approach presented by Zivkovic and Heijden[1], in order to develop a test tracker for our object tracking library. Their work uses a difference of Gaussians in order to smooth over changes in the lighting and image noise. We have, however, found that it is still too sensitive to lighting changes and image noise to be useful for our purposes.

Background subtraction is, in general, only possible with stationary cameras. In order to cope with camera motion, the background model may be periodically rebuilt, or extra sensor information may be used.

2.2.3 Image segmentation

Image segmentation aims to the divide an image into perceptually similar regions [SM00]. An image segmentation algorithm must concern itself with both how to achieve a good partition, and how to do it in a computationally efficient way. Some of the relevant techniques in the context of object tracking are [YJS06]:

Mean-shift: The mean-shift procedure is used on the image, in order to create a partition. Comaniciu and Meer [CM02] and the CAMSHIFT tracker [Bra98] use this approach.

Image segmentation using graph cuts: Every pixel in the image is represented by a node in a graph, G, and connected to adjacent nodes by edges whose weights are some measure of the similarity between the pixels [YJS06]. This graph is then partitioned

[1] Their work is described in [ZvdH06] and the associated source code is available at `http://staff.science.uva.nl/~zivkovic/Publications/CvBSLibGMM.zip`.

into N disjoint subgraphs, by pruning certain edges. The sum of the pruned edges' weight is called a *cut*. Segmentation can, for instance, be achieved by finding the partitions that minimize the cut [WL93].

Active Contours: A contour is evolved around each image region, based on an energy function. For more information, see the paragraphs on contour evolution, in Section 2.3.4.

2.2.4 Supervised learning classifiers

Objects can be detected on a scene by a classifier, which has been trained to recognize them, through some supervised learning mechanism [YJS06]. These methods work by building a function, mapping object appearances to several different object classes. In order to do this, they must be trained by processing several examples, each of them bearing a representation of the object and its associated object class.

Since the learning algorithms can learn to recognize the object's appearance, in several different views, it is not needed to save the object's appearance after the classifiers have been trained. On the other hand, if the classifiers are used on their own, a large number of manually labeled samples is needed to train them.

The Microsoft Kinect [SFC$^+$11] uses a depth camera to capture depth images, which it then feeds to a previously trained randomised decision forest classifier. The Kinect's main contribution to the field of object tracking is to treat pose estimation as object recognition. It does this by classifying each pixel in the depth image as either being part of the background, or as one of 31 body parts: LU/RU/LW/RW head, neck, L/R shoulder, LU/RU/LW/RW arm, L/R elbow, L/R wrist, L/R hand, LU/RU/LW/RW torso, LU/RU/LW/RW leg, L/R knee, L/R ankle, L/R foot (Left, Right, Upper, loWer). In order to train the classifier, a large database of motion captured depth images and associated ground truths was created. The features $f_\theta(I, \mathrm{x})$ used to train the classifier are derived from differences between depth of the current pixel and the depths of two arbitrary pixels in its vicinity, as per Equation (2.1).

$$f_\theta(I, \mathrm{x}) = d_I(\mathrm{x} + \frac{\mathbf{u}}{d_I(\mathrm{x})}) - d_I(\mathrm{x} + \frac{\mathbf{v}}{d_I(\mathrm{x})}) \tag{2.1}$$

Where I is a depth image, $d_I(x)$ is the depth at pixel x in I, x is a pixel and $\theta = (\mathbf{u}, \mathbf{v})$ describes the pixel offsets $\mathbf{u}$ and $\mathbf{v}$.

The classifier is a randomized decision forest, which consists of a set of decision trees, each containing split nodes and leaf nodes. In this case, each split node consists of a feature and a threshold. If a given feature evaluates below the threshold, the left branch is followed, otherwise the right one is followed. Leaf nodes then store a learned probability distribution of body parts for a given pixel, based on the features. The final classification for the pixel is reached by averaging the probabilities of all body parts across all trees, and retaining the highest one.

Joints are then computed by doing a local mode finding operation on each body part's pixels, which are weighed according to their real world surface and the confidence with which they were thought to be that part of the body. Since these modes lie on the body's surface, they are then pushed on the z axis to produce the joint's final estimate.

Kalal et al. [KMM12] also use supervised learning for their tracker. They have called it TLD, which stands for Tracking-Learning-Detection[2]. Their main assumption is that neither estimating the object's position from the previous one (Tracking), nor detecting the object in every frame (Detection) is sufficient to follow the object in the long term. So they combine them by having a tracker they developed, based on an image region's optical flow, a detector which checks several image patches and classifies them as being the object or not, and a learning component that detects probable errors of the tracker and the detector, and tries to correct them so those errors aren't made in the future. A more in-depth look at these components follows.

The detector works by generating a large set of bounding boxes where where the object might be and then running a three-stage cascaded classifier on each of the image patches, which means the patch is rejected if it doesn't pass either of the classifiers. The first stage consists of a simple patch variance check. The second stage is done by a randomized decision forest classifier, previously trained by the earlier frames in the video sequence. Based on the previously observed features of image patches labeled as the object (positive patches) or not (negative patches), it returns the probability that the current patch contains the object. Finally, a nearest neighbor classifier is used. If the patch is similar to one of the previously observed patches, the bounding box is classified as containing the object.

The tracker works by calculating the optical flow of several points within the bounding box, estimates the reliability of that flow, and uses the most reliable displacements to estimate the object's movement. It also includes an algorithm to detect failures in tracking. The estimate of the object's current position (or its absence) is then calculated by by combining the estimates of the tracker and the detector.

After this process, the learning component is engaged to estimate and correct errors in the detector and the tracker. It consists of two "experts" that estimate errors of the other components, the P-expert, which estimates false negatives and adds them to the training set as a positive, and the N-expert, which estimates false positives. In every frame, the P-expert calculates the reliability of the object's trajectory and, if the current location is reliable, it produces 10 transformations of its current appearance (20 in the first frame), normalizes them to a 15×15 pixel image patch, applies Gaussian noise to them, and adds the result to a set of positive examples. The N-expert is based on a key assumption: since we're only tracking one object, parts of the image that aren't part of the object must be part of the background. If the object's current position is reliable, patches that are far from the current position and weren't rejected by the detector's first two

[2] TLD's homepage can be found at `http://info.ee.surrey.ac.uk/Personal/Z.Kalal/tld.html` and its source code is available at `https://github.com/zk00006/OpenTLD`.

classifiers are transformed in the same manner and added to the training set as negative examples. Naturally, both these experts also make errors in estimation. However, the authors demonstrate that the errors in each component cancel each other out and that the system as whole tends to improve long-term tracking performance.

Work on improving TLD's tracking and computational performance was done by Nebehay [Neb12]. He implements TLD in C++, with a few changes. A new, optional, first step is added to the classifier cascade. It rejects patches which are part of a learned background model. After the nearest neighbor classifier is run, he applies non-maximal suppression to the classifier's responses, in order to eliminate similar bounding boxes near the object's location. Another main difference is that the author does not warp the samples used for training the randomized decision forest classifier. These changes cause the algorithm's precision and recall to differ from Kalal's original implementation, depending on the video sequence, but not conclusively for better or worse. In this work, we use Nebehay's implementation of TLD[3].

Other supervised learning approaches include neural networks, adaptive boosting [VJS05] and support vector machines [PEP98].

2.3 Finding object tracks

With the objects detected, one must now track their movement through the scene. Depending on the tracking method, this is accomplished either by detecting objects in the new frame, and corresponding them to the ones in one or more previous frames, or by finding the known objects' position in the new frame. The object representations enumerated in Section 2.1 limit the objects' possible motion patterns [YJS06]. For example, in the case of a point representation, "only a translational model can be used" [YJS06]. In the case of a geometric shape representation, motion can also be modeled after affine or projective transformations, while silhouette or contour representations offer the most flexibility, by being able to model non-rigid objects.

In this section, tracking approaches for points, kernels and silhouettes are presented, but first, one needs to look at what image features should be used to track the objects.

2.3.1 Features for object tracking

In order to track objects, one must select which image features to use, in order to establish the correspondence between an object's previous position and its new one. The process of selecting a feature depends on the chosen object representation[4]. For example, if the object's appearance is represented by a color histogram, it makes sense to use color as a feature to track. Some common visual features used for object tracking are [YJS06]:

Color: the object's color is used, in order to find it in the scene. The object's color is

[3] The code for this implementation of TLD can be found at https://github.com/gnebehay/OpenTLD.

sensitive to illumination. Namely, it depends on both the scene's light source(s) intensity and position and its own reflectance. Since the human eye is more sensitive to brightness than it is to color, the color information can be discarded if it is not needed, or to save memory and/or storage space.

Edges: Along an object's edges, there are usually big changes in image intensities. The object's edges can be used to track the object's boundaries and, thus, find its position. There are numerous edge detection algorithms, although the Canny edge detector [Can87] is the most popular due to its simplicity and accuracy [YJS06].

Optical flow: Optical flow is a pixel's displacement from a frame to the next. To compute optical flow, it is assumed that a pixel corresponding to some image feature will maintain its brightness in consecutive frames. Computing optical flow consists, then, in finding where each pixel has moved to, based on its brightness constancy. A performance evaluation of optical flow algorithms has been made by Barron et al. [BFB94], concluding that the Lucas-Kanade [LK81] and Fleet-Jepson [FJ90] algorithms are the most reliable, meaning that their accuracy does not change significantly, when used for different image types.

Texture: Texture is described by the change of a surface's intensity. This allows us to conclude how smooth or regular that surface is. However, some image processing is needed, in order to find the desired texture(s). Texture extraction can be achieved by using Gabor filters, for instance, such as in [KZL03]. Like edges, textures are less sensitive to illumination than color.

Though these features are described in isolation, tracking algorithms can and do use several of them at once. The choice of the features to use for tracking is usually left to the tracker's developer.

2.3.2 Point tracking

In point tracking, objects are represented by one or more points, and these must be detected in every frame. Tracking is performed by matching the points in a new frame with the ones in one or more previous frames. This can be a tough problem to solve, since there can be points without any correspondence, points that correspond to multiple image features or several points that correspond to the same feature [VRB01]. These problems are aggravated by the possibility of occlusions, misdetection and object entry and exit. A visual representation of this problem can be found in Figure 2.1.

Point tracking approaches can be divided in two classes: deterministic and statistical. Deterministic methods use *qualitative motion heuristics* [VRB01] or descriptor matching in order to establish point correspondence. Probabilistic methods rely on statistical properties of the object's appearance to match the previous points to the new ones and can compensate for image noise.

[4] Appropriate feature selection was discussed in Section 2.1.

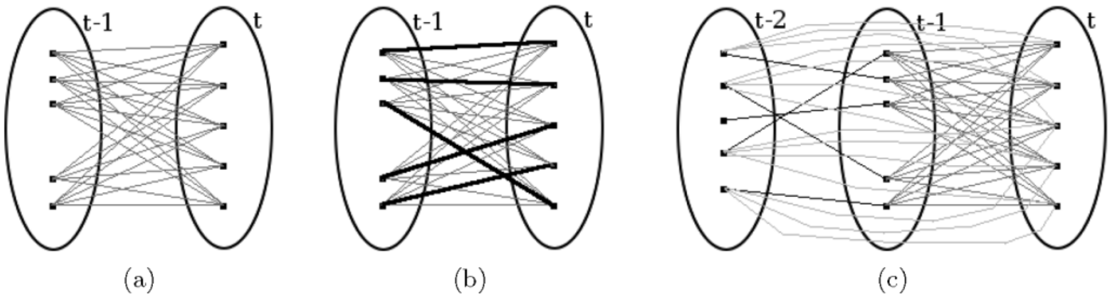


Figure 2.1: Point correspondence [YJS06].
(a) All possible associations of a point in frame $t-1$ with points in frame t, (b) unique set of associations plotted with bold lines, (c) multi-frame correspondences.

2.3.2.1 Deterministic Methods

Deterministic methods for point correspondence rely on previously built object motion models [VRB01].

Deterministic point correspondence methods are, then, responsible for assigning a cost of associating each object in a frame to a single object in the next one [YJS06].

A motion model is a set of constraints on an object's motion. For instance, trackers can assume the object can't move very far, or that its acceleration must be small. In these circumstances, minimizing the cost of the points' association is a combinatorial optimization problem.

Point trackers using motion models differ in what motion constraints are used, and whether and how they handle object occlusions, entries and exits on the scene.

Other trackers use deterministic methods to match their points by first creating descriptors for those points in each frame, and then finding the best match between those descriptors in each frame. These descriptors contain information about the point's position and the image's appearance in its neighborhood.

While detecting its interest points, SIFT [Low04] already has data on the scale and orientation of the image features associated with those points. Its descriptor relies on this data as well as a histogram of gradient orientations and gradient magnitudes around the point. The orientations are weighted by a Gaussian function so that the ones nearest to the interest point count more. These data form a part of the descriptor.

In SURF [BETVG08], the interest points are assigned a general orientation, by calculating the Haar wavelet responses in the x and y directions, for every pixel in a neighborhood of the point. The sum of all responses inside a sliding window, which represents a slice of a circle, is then calculated, and the maximum between all the sums is extracted as the interest point's orientation. This makes the interest point invariant to rotation. The point is further described by a square region, with that orientation, centered on the point and subdivided. For each of these subdivisions, several Haar wavelet responses are calculated, and both their sum, as well as the sum of their absolute values, are saved as the point's descriptor.

In both SIFT and SURF, descriptors are comprised of multi-dimensional data. Descriptors are matched by calculating the Euclidean distance between the descriptors' data, and finding the nearest neighbor from the previous frame's pixels. Since for a large number of feature points matching would be computationally expensive, both approaches apply indexing approaches to make the problem more tractable. The authors do not discuss how to consider matches failed in frame by frame data[5].

There also exist standalone approaches to create descriptors for interest points. The BRIEF descriptor [CLSF10], which we have used to create one of our sample trackers[6], is one such approach. It compares the image intensity of several randomly sampled pixels around the interest point, where every pixel's position follows a Gaussian distribution. Each of these comparisons yields a 1 if the intensity of pixel x is less than the intensity from pixel y and a 0 otherwise. To make the results less sensitive to image noise, the image is smoothed prior to these tests. The result of these tests is, then, a binary string, which makes it possible to match points using the Hamming distance between them, rather than the Euclidean distance. Using the Hamming distance and having smaller descriptors than SIFT or SURF makes BRIEF descriptors much faster to compute and match. Matching is, again, done by finding a descriptor's nearest neighbor in the next frame.

2.3.2.2 Statistical methods

Statistical methods for object correspondence allow us to model and compensate for both image noise and object motion anomalies. This is accomplished by modeling the object's state, based on motion constraints, such as the ones mentioned in the discussion about deterministic point tracking, and, of course, its previous state, while taking image noise into account. This problem can be expressed as the calculation of the probability density function of the current object state at instant t, knowing all the motion measurements from instants 1 to t [YJS06]. When tracking multiple objects, one needs match the measurements to each object's state.

A caveat of using statistical methods for point correspondence is that since they model image noise, object appearance, and possibly entry, exit, and occlusion events as probabilities, they are sensitive to the parameters used to initialize them, and finding optimal values for those parameters is no easy task [VRB01].

2.3.3 Kernel tracking

In the context of object tracking, a kernel is a representation of the object's shape and appearance [YJS06]. Tracking is performed by computing the kernel's motion from one

[5] Lowe [Low04] does, however, describe how to consider failed matches against a large database of previously detected interest points. If the nearest match differs substantially from the second nearest match, the match is considered a failure.

[6] See Section 4.2.

frame to the next. Kernel tracking methods can be divided according to the object representation they use. They can either use template-based appearance models, or multi-view appearance models.

2.3.3.1 Template or probability density appearance models

These approaches tend to have a low computational cost [YJS06]. They can be divided by whether they are able to track just one, or multiple objects.

For a single object: A common way of tracking the object can be done in the form of template matching, where the image is searched for a region similar to the object's appearance, which is usually modeled by its color, intensity or intensity gradient. This search can be computationally expensive, so some means of limiting the problem domain must be sought after. Usually, this is done by constraining the search to a neighborhood of the object's previous position. Another method of constraining the matching is to tentatively try match only a subset of the template's pixels, and only perform a full matching when the tentative match reasonably approximates the object's appearance [SBW02]. A different approach for kernel tracking consists in computing the subset of pixels inside the object's shape.

Proposed single-object trackers using these object models include:

- Comaniciu et al.'s [CRM03] mean-shift tracker. In it, the object's shape is approximated by a circle, and the object's appearance is represented as a weighted histogram of the pixels inside that circle, with the pixels nearer to the center having more weight than those near the circumference. Tracking is done by maximizing histogram similarity between frames, in a window around the object's initial position. A disadvantage of this method is that it requires that the new object region overlaps the previous one. An advantage is that since it uses the mean-shift procedure, there is no need to perform a brute-force search;

- The Kanade-Lucas-Tomasi tracker, in its final version, proposed by Shi and Tomasi [ST94] computes the optical flow of an image region centered on an interest point, in order to locate the interest point's new position.

- TLD [KMM12] performs template matching before moving onto more computionally expensive steps to find the object on the scene.

For multiple objects: In order to track multiple objects, the interactions between them, as well as between them and the background must be modeled [YJS06].

An approach to the problem, by Isard and MacCormick [IM01], requires that the ground plane be known, and represents objects – in this case, people – by a generalized cylinder, defined by discs with different diameters around the feet, waist, shoulders and head. The background and foreground are modeled by different mixtures of Gaussians. The tracker needs to be initialized with the probabilities for object entry and exit. Tracking is done by using particle filters. It can handle object occlusion.

Another approach, by Tao et al. [TSK02], models the image as a set of layers, one for the background, and one for each object in the scene. In this way, they explicitly model the interactions between objects and background. The method is, therefore, resistant to even full object occlusion.

2.3.4 Silhouette or contour tracking

Silhouettes (sometimes also called blobs) and contours can precisely describe objects with complex shapes, such as a human or a hand, unlike simple geometric shapes, which merely approximate it.

Before tracking, one needs to concern oneself with how to store silhouettes and contours in memory. Silhouettes are usually stored in memory by having a boolean variable for every pixel in the image, with a value of one if that pixel is inside the silhouette and a value of zero if it's not [YJS06]. It is trivial to extend this kind of memory representation to multiple object silhouettes. Contours are stored either as a set of control points – the so-called explicit representation – or as a function, defined on a grid, such as a level set for each contour – an implicit representation. In the explicit the representation, if the control points include normal vectors, it is possible to distinguish the inside from the outside of the contour. Therefore, if the contour is closed, a silhouette may also be represented in this way.

Tracking is done by either finding the silhouette on the current frame, or evolving a contour from its position in frame $t-1$ to the object's position in frame t. Following, each of these approaches shall be discussed.

2.3.4.1 Shape matching

Shape matching is simply a search for the object's shape in a new frame. Since this requires that the object shape is fixed, typically only object translations can be handled by this approach. Shape matching is similar to template matching[7]. Searching for the new silhouette position is done by establishing an hypothesis for it, and comparing its appearance with the known object model. When the error between the new model's appearance and the known model is minimized, the object has been found.

Huttenlocher et al. [HKR93] propose such a scheme, by means of minimizing the Hausdorff distance between a portion of the image and the object model. A Hausdorff distance of d implies that every point of A is within a distance d of every point of B and vice-versa. Tracking is performed by minimizing the Hausdorff distance between the hypothesized location of the object – which is a translation of the previously known silhouette, in its vicinity – and the object's model.

[7] See the discussion on template matching in Section 2.3.3.

Li et al. [LCZD01] calculate the average optical flow inside the silhouette to approximate the object's new position. They then use a variation of Hausdorff distance metric, in order to verify the object's position. The modifications to the metric were done in order to compensate for the Hausdorff distance's sensitivity to image noise.

Another possible approach is to detect the silhouettes in multiple frames and then corresponding silhouettes between frames, similarly to how it's done for point correspondence, but exploiting the knowledge of the whole object's appearance. This approach is taken by Kang et al. [KCM04], for instance.

Haritaoglu et al. [HHD00] have proposed a silhouette tracker to detect people, in a surveillance application. The silhouettes are detected, in each frame, through background subtraction. They are, then, matched by calculating the correlation between the edges, in consecutive frames. In addition, since their approach is limited to tracking people, it can exploit knowledge about a person's anatomy, in order to cope with changing silhouette shapes and silhouette overlap.

2.3.4.2 Contour evolution

Another way to perform tracking is to iteratively evolve the object's known contour, in frame $t-1$, until it surrounds the object, in frame t. This kind of approach is limited by the requirement that the intersection between object silhouettes, in consecutive frames, is not empty [YJS06]. Tracking can be done by modeling the contour's state and motion using state space models, or by minimizing some function, denoting the contour's energy.

Using state space models: The contour's shape and motion parameters define the object's state. Various motion parameters can be used to represent the contour. Examples include a contour represented by several control points, each being "pushed" by springs of varying stiffness, and a model of an ellipse with several control points, where the contour sits on the lines normal to each control point. At each frame, the state, which can be, to a is updated in a way that minimizes its *a posteriori* probability.

Using energy function minimization: An energy function associated with the contour is minimized, and the contour's new position is calculated, based on that. This energy function can be in the form of the optical flow in the contour's region [BSR00], or an appearance model for both the object and the background [YLS04].

2.4 People tracking

The literature associated with people tracking is vast. An initial overview of the subject has been prepared by Poppe [Pop07]. Furthermore, Moeslund et al. have also published more in-depth literature surveys [MG01, MHK06]. People tracking is but a subproblem of object tracking. In fact, several of the previously analyzed object trackers have been proposed for tracking people or faces. Examples include [HHD00, VJS05, YLS04, SFC+11].

Generally, an object tracker can, then, track people, as long as it has a suitable shape and appearance model for them. In addition, since people trackers aren't generic object trackers, information about a person's structure can be exploited, in order to generate better results. For instance, a person has a torso, and a person's head is "above" it, in the person's dominant axis, and a person's limbs are also connected to the torso. Furthermore, a person has a limited number of postures they can take, and these can be modeled. Haritaoglu et al. [HHD00] take advantage of this. They model several human postures and views, and exploit information about a person's structure, in order to find multiple people in regions where their silhouettes overlap and handle occlusion. As another example, Jesus et al. [JAM02] create an articulated model of the human body. In each frame, the new position of the model is predicted according to several estimators. A match score is attributed to each predicted model and the best match is extracted as the person's model in the new frame.

Since we have a particular interest in tracking people for our work, we have used the Microsoft Kinect sensor in order to track people, due to its accuracy.

2.5 Current object tracking libraries

Generic, high-level, object tracking libraries are all but inexistent. The one closest to what this work is trying to achieve is OpenNI[8]. Using it, however, requires specialized hardware, with appropriate drivers. It is also focused on natural interaction and, in consequence, only able to process an online video/depth image feed.

The proposed object tracking library is able to process offline video, and also has a more relaxed programming model, allowing developers more freedom in regards to where they call its code.

2.6 Object tracking discussion

As has been shown in this section, object trackers have many and varied expectations regarding their input, and process it in wildly different manners. Besides requiring frame by frame image data, trackers may or may not require a set of pre-initialized training data, or a starting object position. They can also use a lone camera, or many. When processing those data, the trackers can also choose to extract a myriad of different features from it, in order to estimate the object's frame by frame position.

Developing a library with a common programming interface for these tasks requires, then, that we find what these trackers have in common. In Chapter 4 we discuss how we have solved these problems.

[8] OpenNI has been developed by PrimeSense, and is available at `http://www.openni.org`.

For our purpose of annotation of live dance performances, the object tracking algorithms we use need to be able to be used in real-time and have a decent precision. To this end, after some testing with other trackers, as described in Section 4.2, the trackers we actually used for our purpose of real-time annotation are the TLD Tracker [KMM12] and the Kinect [SFC+11].

3

Video Annotation Literature Review

This chapter concerns itself with video annotation, its associated problems and proposed solutions. In particular, the problems associated with creating user interfaces for video annotation are discussed. The problems and formerly proposed solutions for performing annotation on tracked objects in real-time are especially interesting for our work.

3.1 Background

Annotation is the process of marking up a document, so as to facilitate its comprehension [GCGI+04]. Its objectives vary in every instance. It can serve to remember a certain passage, reason about it, or reiterating it in other words, in order to make the document clearer to one or more of its readers or collaborators, serving therefore, as an active reading effort.

Video annotation poses particular challenges, since unlike traditional document annotation, any effort to convert the active reading principle to video (active watching) must have not only a spatial dimension, as paper annotations do, but also a temporal one [CC99]. This poses special user interface challenges, especially when combined with attaching annotations to tracked objects. Prior approaches to these challenges are discussed in Section 3.4 and our approach to attaching annotations to tracked objects is exposed in Chapter 5. Before that, though, we must look at other issues that arise when annotating video.

The problem of video annotation has been looked at from data models for annotation storage to, methods for automatic annotation, as well as interfaces to support manual annotation. Let us examine each of these in turn.

3.2 Annotation storage

Concerning annotation storage, there are several data models, although, in general, it is preferable to store the annotation data separately from the video stream, so that the annotations can be distributed, regardless of copyright issues with the video itself [AP05]. This separation between video data and annotations is present in all the analysed data models, regardless of whether the annotations are stored in a format devised by the author(s) [AP05, HHK08, Val11], or using the MPEG-7 standard's [ISO02] Description Definition Language to that effect, as in [NTM07] and [RSK02]. In Advene's [AP05] case, the data model says nothing on how the data should be visualized, leaving a lot of room for the application developer to make that decision, based on his or her needs.

For our work, annotations and object anchors are saved in a custom-made XML format, building on what was previously done in the Creation Tool [Val11].

3.3 Automatic video annotation

Automatic video annotation is focused on automatically generating metadata on the video, generally in order to facilitate its archival and retrieval.

Regarding automatic video annotation, Neuschmied *et al.* have presented one of the more sophisticated approaches, in which they not only perform shot detection on the video, but they also extract features on each frame, in order to enable object tracking [NTM07]. However, this tool requires the video to be preprocessed, and is limited in the kinds of objects it recognizes. It is also focused on annotations as video metadata, not as augmentations of the video's content. In contrast, our work includes real-time, user-customizable annotation over tracked objects.

Diakopoulos *et al.* have equally demonstrated shot detection but, in their work, they also perform face detection, as well as a semi-automatic audio transcription, which is then analysed, in order to extract references to people, money and comparisons [DGE09].

The *MPEG-7 Metadata Authoring Tool*, by Ryu, Sohn and Kim, also has a content analysis module which detects shots, allowing the users to group them in scenes [RSK02].

These tools are also capable of enabling users to manually annotate the video. However, that annotation is focused on merely adding metadata to it.

Any method for automatic object detection and tracking can also be used for automatic video annotation, by following one or more desired classes of objects throughout the video. Reliable shot and scene detection should be used in this case, so as not to mistakenly and continuously detect the same object, in two completely different scenes. Since our work is focused on annotating live or unedited video from dance performances, we make the assumption that each video we are annotating is made in a single shot, so this problem does not apply.

In this work, we create anchors to tracked objects, onto which further annotations can be attached. This concept has come largely from the work that has been done on text document annotation [BBGC01, BM03]. The process we use to create and attach annotations to these anchors is described in Chapter 5.

Our work focuses more on manual annotations, since we aim to enable users to comment the video, and to take and share notes about the video's content. There are interface design challenges involved in enabling users to achieve this. A study of these and other challenges follows.

3.4 Manual video annotators

One of the earliest efforts at digital video annotation was the Experimental Video Annotator (EVA) [Mac89]. It allowed users to create video annotations consisting of text, or links to other images and/or videos. Note that the video being annotated was analog, but the images were digitized to display them for annotation, which required special hardware.

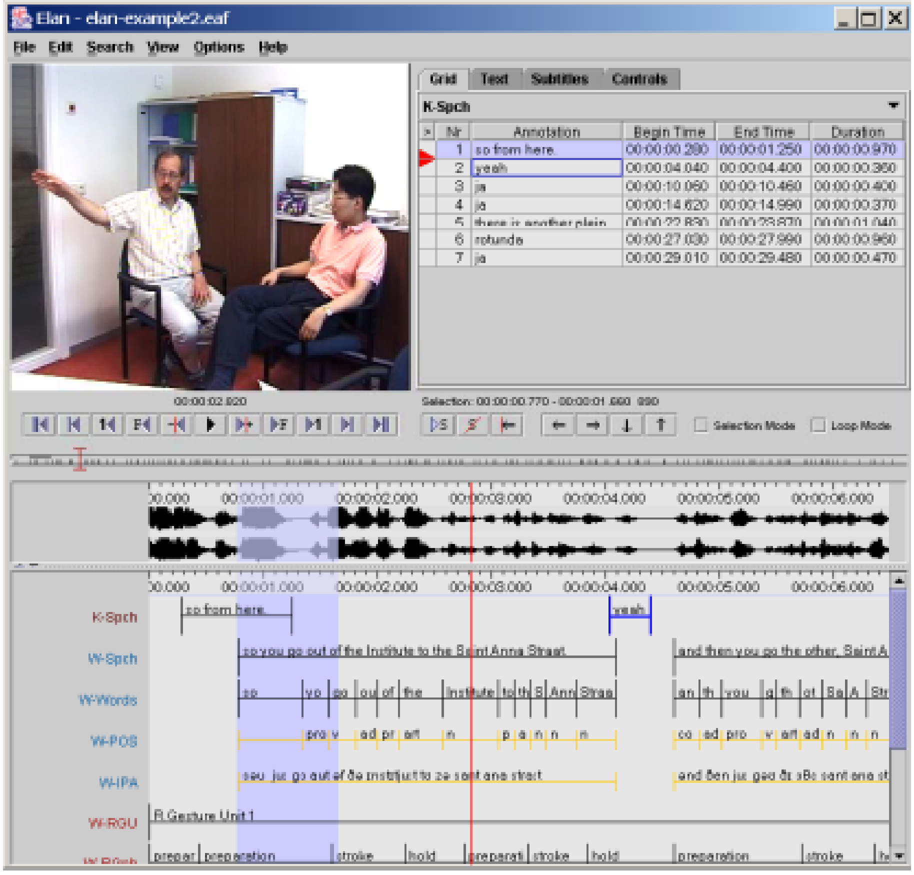
AnTV [CC99] could also be used for creating video annotations, which consisted of text, images and video clips. This annotator allowed the users to decide the spatial and temporal position of these annotations.

ELAN (EUDICO Annotator Linguistic) [BRN04], pictured in Figure 3.1, was designed for performing linguistic analysis, but it is capable of generic video annotation. It can annotate single- or multi-camera videos with text, video and images, which are associated with a video segment on the timeline. This annotator allows its users to organize their annotations in hierarchical *tiers*.

Along the years, several insights have been gained in user interface design, in order to support video annotation. Tools such as *VideoTraces* [CFS03], *VCode* [HHK08], the *MPEG-7 Annotation Tool* [NTM07], *Videolyzer* [DGE09] and the *Creation Tool* [Val11] teach us valuable lessons in user interface design. Apart from [NTM07], these were designed based on input from professionals in their target user group.

VideoTraces [CFS03] (see Figure 3.2) aims to enable users both to evaluate recorded dance performances and to share those evaluations for discussion. Users can annotate the video with gestures and voice annotations. Changes in video playback are also shared as part of the annotation process, so that collaborators can see exactly what the original annotator was looking at when he made the original comments. The combination of the changes in video playback and the annotations is called a *trace*.

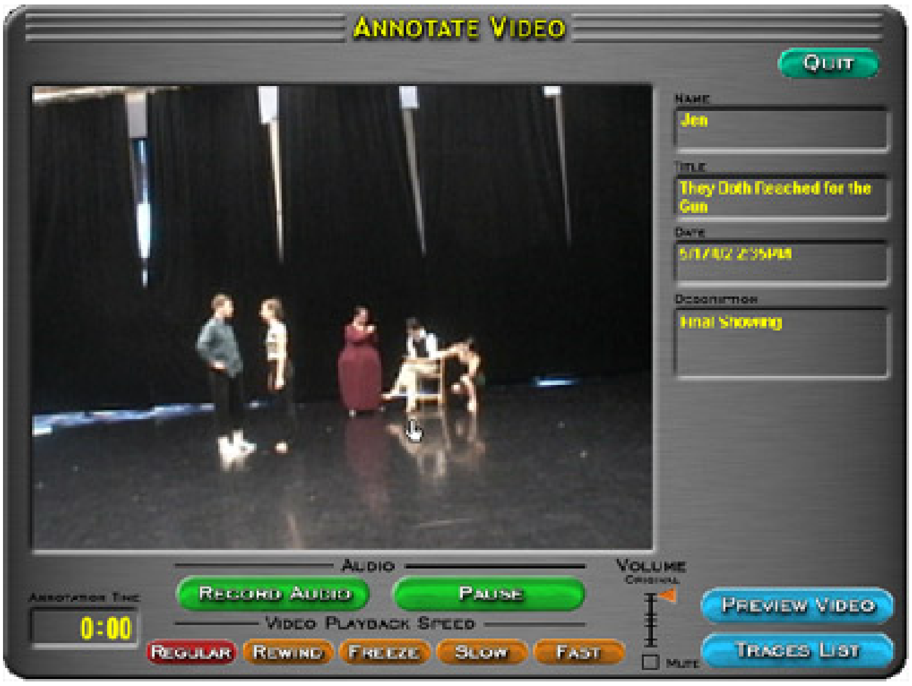
In VCode [HHK08], it is possible to create timeline "tracks" that represent different event types. Events can be momentary – referring only to a specific instant in time – or ranged – referring to a time interval. In the interface, every event is represented in a timeline, where the momentary event tracks are stacked, in order to take less screen real-estate and the ranged tracks are under them, in a scrollable area of the timeline. It is also possible to show external data, from any given source, beneath the timeline. This external data is synchronized with the timeline, so the annotators can be sure that they are not just imagining some detail. Videos related to the one currently playing are shown


Elan - elan-example2.eaf
File Edit Search View Options Help
Grid Text Subtitles Controls
K-Spch
00:00:02.920
Selection Mode
Loop Mode
K-Spch
W-Spch
W-Words
W-POS
W-IPA
W-RGU
so from here.
yeah
so you go out of the Institute to the Saint Anna Street
and then you go the other, Saint A
R.Gesture Unit 1
prepar preparation stroke hold preparati stroke hold preparation stroke

Nr	Annotation	Begin Time	End Time	Duration
1	so from here.	00:00:00.280	00:00:01.250	00:00:00.970
2	yeah	00:00:04.040	00:00:04.400	00:00:00.360
3	ja	00:00:10.060	00:00:10.460	00:00:00.400
4	ja	00:00:14.620	00:00:14.990	00:00:00.370
5	there is another plain	00:00:22.830	00:00:23.870	00:00:01.040
6	rotunda	00:00:27.030	00:00:27.990	00:00:00.960
7	ja	00:00:29.010	00:00:29.480	00:00:00.470

Selection: 00:00:00.770 - 00:00:01.660 890

Figure 3.1: ELAN with a document open [BRN04].



NAME
Jen

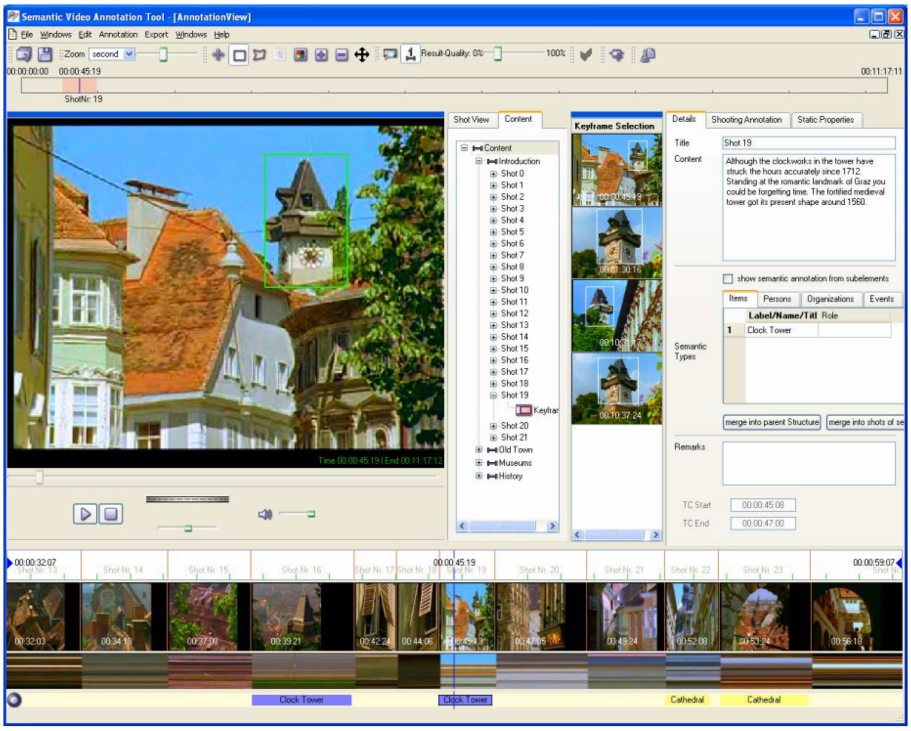
TITLE
They Both Reached for the Gun

DATE
5/17/02 2:35PM

DESCRIPTION
Final Showing

ANNOTATION TIME
0:00

Figure 3.2: VideoTraces' main window [CFS03].



Title: Shot 19

Content: Although the clockworks in the tower have struck the hours accurately since 1712. Standing at the romantic landmark of Graz you could be forgetting time. The fortified medieval tower got its present shape around 1560.

☐ show semantic annotation from subelements

Semantic Types

	Label/Name/Titl	Role
1	Clock Tower	

Remarks

TC Start 00:00:45:08

TC End 00:00:47:00

Time 00:00:45:19 | End 00:11:17:12

Figure 3.3: The MPEG-7 Annotation Tool's user interface [NTM07].

in a dock, next to the video playback window. When one of them is clicked, it is played in the main window, and the video that was previously playing takes its place in the related videos. The videos in the dock are chosen by the user, who can specify the instant they start playing, relative to the main video.

There are two timelines in the MPEG-7 Annotation tool [NTM07] (Figure 3.3). The one in the top encompasses the whole length of the video, while the one bottom one displays the shot boundaries and dissolves for only a section of the video. Unlike VCode, this tool also supports binding annotations to a spatial region in the video, thus allowing the automatic creation of annotations for tracked objects.

The WaCTool [CTGP08] has a different kind of audience. This tool was proposed to annotate TV streams, so its target users are not professionals. Though a bit limited, on account of only allowing the annotation of individual frames, it has some interesting features, such as using "digital ink", allowing the users to create free-form annotations. Furthermore, its interface allows for discussion between users, be it through text, or voice chat.

Costa *et al.* propose an interface centered on the concept that annotations are merely another way of looking at a video. They have called each such perspective a video lens [CCGa02].

Videolyzer's interface [DGE09] has a different objective than the systems analyzed thus far. It aims to be a platform for discussion on the video and, as such, it dedicates a great deal of screen real-estate to the display of comments and replies to them. A video transcript can also be seen beside the video, so the users can select what they're commenting on with greater precision. A stacked timeline is also present, showing a regular navigation bar, vertical strips of pixels taken from certain frames, markers showing where there are automatic annotations and bars indicating user comments or annotations. The coloring of these bars indicates the degree of support for each particular annotation and their brightness indicates the number of annotations in that point in time. Detected faces are drawn in the video, as a rectangle.

Bargeron and Moscovich [BM03] present a way to automatically determine to what part of a text a given annotation should be anchored to. In video, whenever we are not using an automatic tracker, such as the Kinect, we are not aware of the scene's structure and we do not restrict the available kinds of annotations, so users must manually create anchors for the objects they wish to track.

In the Ambulant Player [Bul04] project, pen-based annotations are dynamic, but have their spatial path explicitly defined by the user. In Goldman's work [GGC$^+$08] graphical video annotations are combined with a particle object tracking algorithm, but this work does not consider the particular case of pen-based annotations and the tracking feature needs a long time of video pre-processing.

There have also been approaches to video annotation display in the context of augmented reality [GRD05], with the objective of placing an object's labels where they are least visually obtrusive.

Dragicevic et al. [Pie08] also make use of object tracking, in order to allow navigation in the video by clicking and dragging along an object's tracks.

Both object tracking and video annotation have been extensively explored in the literature. However, rarely have they been studied together. Most of the work in these subjects was concentrated in the form of using some kind of automatic object tracking in order to generate video metadata, with the goal of improving video search and browsing (for instance, [NTM07]).

As such, work on associating annotations with tracked objects is less common, even if it allows annotations to retain their context over time. Examples of such work include [CC09], which is quite limited since its tracking is done by frame differencing, making it only follow image motion, without regard as to why that motion occurred. For instance, it could be that some other object has occluded the tracked object. Another interesting exploration of associating annotations with tracked objects comes from the MediaDiver [MFAH$^+$11] tool, shown in Figure 3.4 which can track hockey players across multiple cameras, and allows users to create annotations about a specific player, or to change cameras when that player leaves the current camera's frame. By using several instances of an object's position over time, users can also train the system to follow that object. However, following arbitrary objects requires the user to work on a pre-recorded video, which he then must pause several times in order to have the system following that object. Furthermore, the system does not allow them to associate visual notes with the players on the video window. Its tracking capabilities also need a pre-calibrated, multi-camera setup

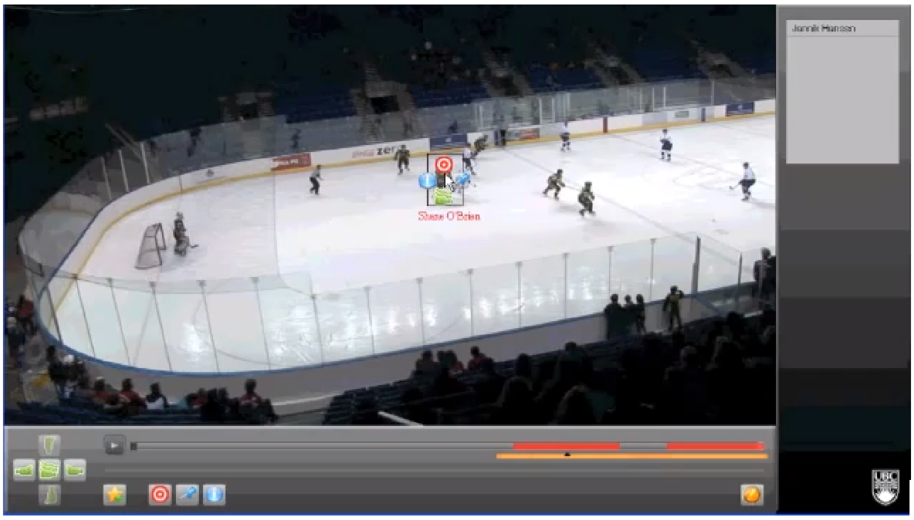


Figure 3.4: MediaDiver's main window [HHK08].

in order to work. In contrast, our work can track and be instructed to follow arbitrary objects live and in real-time.

On Tablets, most of the approaches to media annotation have been focused on text [BM03, GCD11]. The ones focused on video have not used object tracking, thus far [DE06, RB03].

A more recent integration of several modes of annotation can be seen in [Val11], where the author details the development of Creation Tool, pictured in Figure 3.5. Since our work presents the integration of object tracking functionality into this tool, we shall present a more in-depth look into it in the next section.

3.5 The Creation Tool

Our work uses a video annotator for contemporary dance [Val11] as a case study. The usage of such an application allows the study of real-time annotation of video objects, using Tablets both in outdoor and indoor environments.

The Creation Tool, pictured in Figure 3.5, enables its users to annotate video with text, hyperlinks, audio notes, pen-based notes, and special user-configurable marks, all running in a Tablet, as previously detailed in [CVS+11, CVAa+12]. This tool was developed for annotating dance performances or rehearsals, and it enables users to create digital ink, audio, text and hyperlink annotations. It is also possible to freeze a single frame and annotate on it. The tool also presents two modes of video visualization: real-time mode, which is synchronized with the live event, and delayed mode, which displays the video with a small time delay [CVAa+12].

In order to enable this tool to annotate video objects in real-time, we have developed object tracking functionalities, so the annotations can actually move, as well as support for moving object selection. The object tracking support came in the form of a generic,

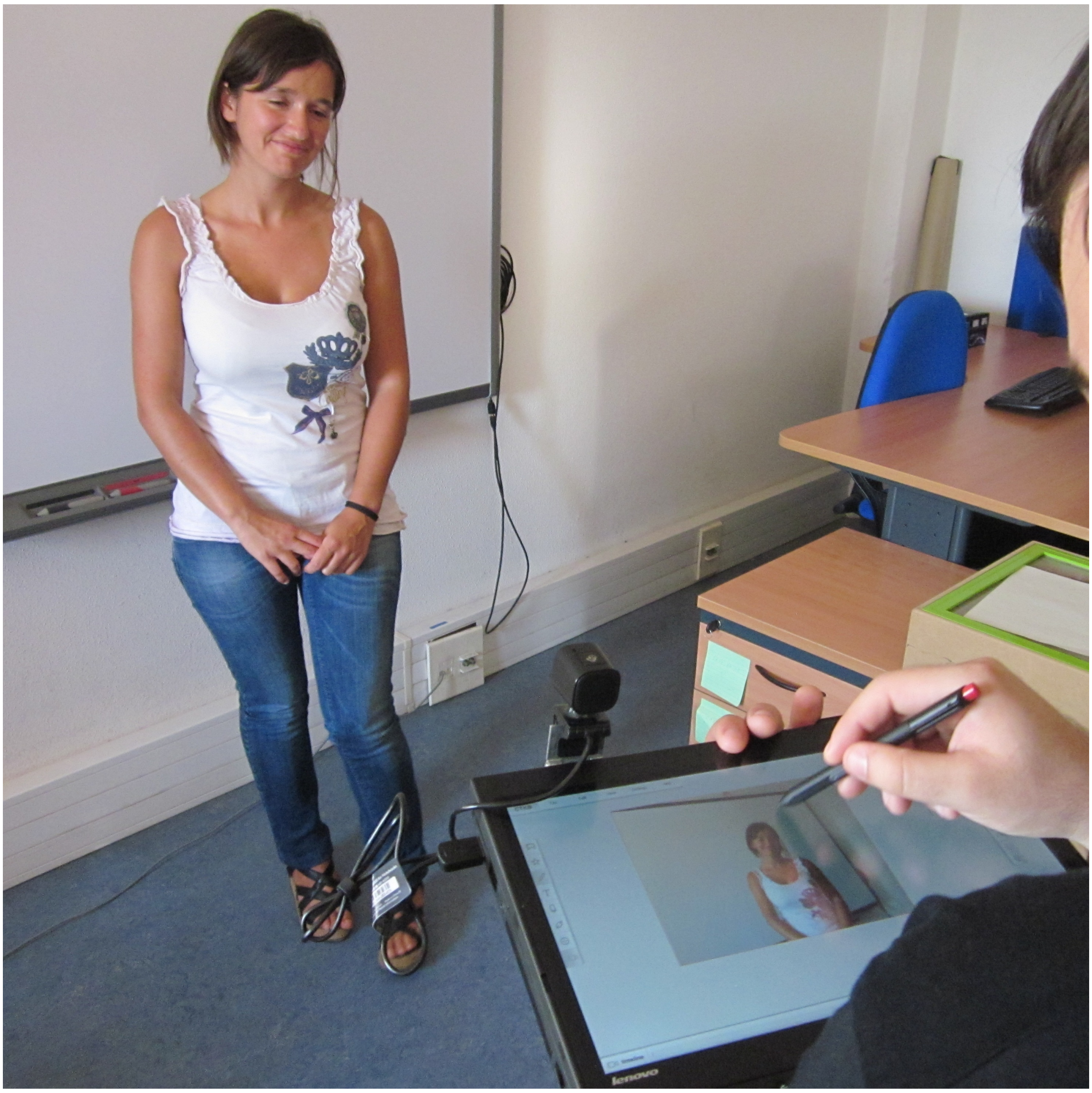


Figure 3.5: Tablet running the Creation Tool.

high-level object tracking framework. Our goal in developing this framework was to be able to use several object trackers at the same time, with a common API to use them. This development process is discussed in the next chapter.

3.6 Video annotation discussion

As we have seen in this chapter, digital video annotation has been an extensively studied subject since the late 1980s, with EVA [Mac89]. Models for annotation storage, methods for automatic annotation and interaction techniques for manual video annotation have been discussed.

Since our work is already based on the Creation Tool [Val11], the model for annotation storage and most of the user interface was already defined. We chose to build on this system, since it already combines a great variety of methods and modalities for annotation. Our work in it consists in adding motion tracking functionality, in order to maintain the annotations' context. To this end, we use the Kinect to automatically create anchors to annotate on people in the video, much like in [NTM07], but available in real-time, and with the option of associating multi-modal annotations with the tracked objects.

We also offer the possibility of tracking arbitrary objects. MediaDiver [MFAH+11] also does this, but it requires the user to select the object's position in multiple frames, and only on previously recorded video, and can only associate a more limited set of annotations with them. Our work, in contrast, allows users to attach multi-modal annotations to tracked objects in real-time, by learning the object's appearance over time, based on a single frame, using TLD [KMM12]. In order to facilitate the selection of the object to track in real-time, we have developed two variants of the Hold method [AHFMI11]. The process of integrating motion tracking functionality is described in more detail in Chapter 5.

For now, we shall discuss our approach to object tracking and the framework we have created to support that functionality.

4

An Object Tracking Framework

In order to annotate video objects in a video where no ground truth data is available, one must, first of all, actually track said objects. In this chapter, we will focus on our approach to generalize tracking an object in a scene.

In the case of the Creation Tool, we are mostly interested in tracking the performers' position in the video. The choreographers may, however, be interested in tracking and annotating a prop, or images projected to the scene's background, a task that the tracking algorithm used for following the performers may not be suited for. How could that be accomplished if, as we have seen in Chapter 2, object trackers have massive differences in data needs and object representations? Our solution was to create a high-level object tracking framework, codenamed *libobtrack*[1], presenting a common interface for tracking algorithms that:

- Can use varied object representations;
- Can work with algorithms that need prior training or not;
- Can work with algorithms that need or do not need a hint to the object's initial position.

This can be a challenge, since besides requiring frame by frame image data, trackers may require a set of pre-initialized training data, or a starting object position. When processing those data, the trackers can also choose to extract a myriad of different features from it, in order to estimate the object's position in every frame.

[1] The source code for *libobtrack* is available at `https://github.com/jmfs/libobtrack`.

The development of a framework with a common programming interface for these tasks demands, then, that we find what these trackers have in common. In essence, they all:

1. Perform some initial training step. For those that do not need training, this step is a no-op;

2. Perform an initial object detection, with or without the help of a hint for the object's starting position;

3. Receive the next frame of data and process it in some fashion;

4. Return the object's estimated position, and may be able to return its shape and/or appearance.

These were, then, the requirements for the design of the framework's programming interface.

For the purpose of annotating video, merely knowing the object's estimated position may be useful, but knowing its shape enables application developers to depict it more accurately. For these reasons, the framework also provides abstractions for an object's shape, enabling developers to extract it from any tracker. This capability comes at the cost of increased complexity for making a tracker conform to the framework's programming interface, although we believe that its usefulness outweighs this cost. Abstracting the object's appearance proved too difficult for us to do, since we'd have to abstract things with almost nothing in common. For example, the information that interests us in a color (pixel RGB components) has nothing to do with what we are interested in a texture (roughness, periodicity, tilt, size). The texture is also spread out over several pixels, while each pixel has independent color information. Creating a common interface able to obtain all of these features would require obtaining all the pieces of this information. Even if we were able to, it would only work until someone devised some other kind of image feature to use in their tracker. Fortunately, we have found that in practice and at least for our application, the object's appearance is less of a concern than its position or shape. The appearance is then kept internally in each tracker's implementation.

Pseudocode of a typical usage scenario for the framework is presented in Listing 4.1. In it, `initParameters` is tracker-dependent, and `getTrainingData`, `getNextFrame`, and `getHint` are the responsibility of the application developer. As can be seen, it is only on initialization that there may be a need for tracker-specific information. The rest of the process is abstracted away by the framework, once bindings for a tracker are implemented in it.

Let us now look at how the framework's design goals were achieved.

Listing 4.1: An application's interaction with the object tracking framework.

```
Tracker tracker = new MyTracker(initParameters);
if(tracker.needsTraining()) {
  Pair<Image, Shape[]>[] trainingData = getTrainingData();
  tracker.train(trainingData);
}
Image frame;
while(getNextFrame(frame)) {
  if(!tracker.isStarted()) {
    Shape hint = null;
    if(tracker.needsHint())
      hint = getHint();

    tracker.start(frame, hint);
  }
  tracker.feed(frame);

  Shape[] shapes = tracker.objectShapes();
  /* Do whatever processing is required */
}
```

4.1 Framework implementation and discussion

The design of the object tracking framework has to take into account the needs of both application developers and tracker developers. Tracker developers have a need to easily integrate their tracker into the framework. Application developers who want to use more than one kind of tracker need to have a consistent interface between them.

Since there are so many kinds of trackers, as exposed in Chapter 2, an initial idea was to have a base `Tracker` class, and having several other base classes inheriting from it, such as `PointTracker`, `MultiCameraTracker` or `SkeletonTracker`. This idea was discarded, however, since if a tracker was a multi-camera point tracker, the design would have two options: either have `Tracker` be a purely abstract class, or to have multiple inheritance from it and keep both instances of the class synchronized. Both these approaches encounter some classical problems with multiple inheritance: the first one would need to have either tracker developers overriding every method of these methods, or to have tracker users explicitly deciding which one to call, while the second one would have two, most likely not synchronized, instances of the `Tracker` class in memory, due to the way multiple inheritance works in C++.

In the end, we have opted to include the entire interface in a single `Tracker` class, which tracker developers can derive from in order to integrate their tracker in the framework, thus avoiding problems with multiple inheritance. The `Tracker` class is at the heart of the framework. The class' interface is shown in Listing 4.2.

Listing 4.2: The `Tracker` class.

```cpp
class Tracker {
public:
  //! Error codes for the start and feed methods. All error codes are negative.
  enum Errors {
    //! Returned for a tracker which needs a hint, but doesn't receive one
    NO_HINT = INT_MIN,
    //! Returned in case the data passed is invalid
    INVALID_DATA,
    /*! Returned if further initialization is needed
      Having a need for it should be avoided whenever possible,
      in individual trackers.
     */
    INIT_NEEDED,
  };
  Tracker(bool needsTraining, bool needsHint);
  virtual int init();

  /*! Performs an initial object detection, with or without hints,
    or updates one or more object's current shape and/or appearance models.

    If needsHint is true, ti should contain data, otherwise,
    NO_HINT is returned.

    If needsHint is false, what to do with any data in ti is up
    to each individual tracker's implementation.

    \param ti Hints about the objects in the image.
      If ti is NULL, or if it doesn't contain data, and needsHint == true,
      NO_HINT is returned.
    \param idx If non-negative, indicates the first object index to update.
      If negative, new objects are tracked.

    \return The number of detected objects in the image, or a negative error code.
   */
  virtual int start(const TrainingInfo* ti = NULL, int idx = -1) = 0;

  /*! Feeds a new image to the object tracker. If no initial object detection
    has been done yet, start() is called.

    \param img A new image.
    \return The number of detected objects in the image.

    \sa start
   */
  virtual int feed(const cv::Mat& img) = 0;
  bool needsTraining() const;
  bool needsHint() const;
  bool isTrained() const;
  bool isStarted() const;
  virtual bool train(const std::vector<TrainingInfo>& ti);
  virtual bool train(const TrainingInfo& ti);
  virtual void stopTrackingSingleObject(size_t idx);
  virtual void stopTracking();
  /*! Appends the shapes found to a vector.
    The contents are only guaranteed to be valid pointers until the next call to feed().
    \param shapes Output. The found shapes will be appended to this vector.
  */
  virtual void objectShapes(std::vector<const Shape*>& shapes) const = 0;
  virtual void objectShapes2D(
      std::vector<const Shape*>& shapes, int forImage = 0) const;

protected:
  /*! If true, this tracker has already been trained and
    it is ready to start tracking objects.
  */
  bool trained;
  bool started; //! If true, initial object detection has been done

private:
  bool _needsTraining; //! Specifies if this tracker needs to be trained by the train() function
  /*! Whether this tracker needs an initial hint to the object's position.
  */
  bool _needsHint;
};
```

Unsurprisingly, this programming interface closely mirrors the previously enumerated set of goals for the framework.

Tracker developers derive from `Tracker` in order to provide their own trackers in the framework. Since some of the methods would not apply, or would be the same from tracker to tracker, `Tracker` provides default implementations for these. For instance, trackers that do not need training do not need to override the `train` method. Also, overriding the `stopTrackingSingleObject` method isn't needed for trackers able to detect objects automatically, since the object would just be re-detected in the next frame.

Some trackers require initialization of additional libraries or special data structures in order to be able to be used. Since some of these libraries' C++ interfaces do not follow the Resource Acquisition Is Initialization (RAII) idiom [Str94], we made the decision not to use C++ exceptions[2], since reliably freeing only the memory and resources acquired inside the try blocks is all but impossible for non-trivial code. Due to this and the fact that constructors cannot return values, the `init` method was introduced to deal with possible failures of this nature.

Training the tracker with object shapes is handled either by the `train` method or

[2] In the case of the trackers already included in the framework, OpenNI is the main culprit.

Listing 4.3: The `Shape` class.

```
/*! Base class for representing abstract shapes.
  Each shape must derive from this.
*/
class Shape {
public:
  /*! Returns the Shape's centroid.
    If the Shape is 2D, the centroid's Z coordinate will be 0.0f.
  */
  virtual cv::Point3f centroid() const = 0;
  /*! Returns an axis-aligned bounding rectangle for the shape.
  */
  virtual cv::Rect boundingRect() const = 0;
  /*! Returns the smallest possible bounding rectangle for this shape
     (may not be axis-aligned).
  */
  virtual cv::RotatedRect boundingRotatedRect() const = 0;

  virtual bool isInvalid() const;

  virtual bool isComposite() const;
  virtual bool isAlternative() const ;
  virtual bool isSkeleton() const;
  virtual bool is3D() const;

  /*! Puts all the pixels contained by this Shape into result.
    Does not clear result before the operation.
  */
  virtual void getPixels(std::vector<cv::Point>& result) const = 0;
};
```

some previous training is loaded internally in each tracker's implementation. The training data consists of images associated with a set of objects found in them. This training should not be confused with a hint to an object's initial position in the scene, which can be passed in the `start` method. After this process, the tracker can be updated with new images by calls to the `feed` method.

After this process, a `Tracker` can be queried for the objects that it has detected through the `objectShapes` methods, which return one `Shape` per detected object. The `Shape` class is the base class for all object shapes. When returned from the tracker, there is a `Shape` class for each object, so that users of the framework can associate each object with a particular shape. Disappearing objects are handled by having the returned `Shape` as invalid. For instance, for rectangles, this is made by having the top left corner at (INT_MIN, INT_MIN) and having zero width and height. The user can check for invalid `Shapes` with the `isInvalid` method. This is done so users of the framework can keep accessing the same object by the same index in the returned vector. Let us, then analyze, the `Shape` class, an abstract class shown in Listing 4.3.

From this interface we can see that every 2D shape can return its centroid, axis-aligned bounding box, smallest bounding rectangle, and a pixel blob. If the user only wishes to use a single kind of tracker, he/she may choose to sacrifice abstraction in order to access more detailed information on the shape.

The currently supported shapes are pixel blobs, 2D and 3D skeletons, and rectangles, whether they are axis-aligned or not. Since there is a `1 : 1` correspondence between objects and `Shape`s, complex object shapes are handled by having helper classes to support combining simpler ones. These classes are based on the set operations of union (`ShapeUnion`), intersection (`ShapeIntersection`) and difference (`ShapeDifference`). A shape with alternative representations can be represented by creating a subclass of `ShapeAlternatives`.

4.2 Available Tracking Algorithms

As detailed in [CVS+11], this framework's design was first tested and refined by adapting OpenCV[3]'s implementation of the CAMSHIFT [Bra98] tracker to work with it, as well as by developing an experimental point tracker. We have devised a simple custom tracker, which worked by extracting interest points from each image, and matching descriptors between frames. This process was tested with SIFT [Low04] and SURF [BETVG08] point detectors, but since they were slow, what was done in the end was to find FAST [RD06] features from each image, build BRIEF [CLSF10] descriptors for them, and remove the features which had irregular movement, compared to the others. Although these proved to be worthwhile tests to ensure that designing such a framework was viable and to address some of that design's early shortcomings, the results that these approaches yielded were not accurate enough for our purposes.

In the meantime, the Microsoft Kinect Sensor became usable in a personal computer, making it the following candidate for adaptation to the framework, since it corresponded to the state of the art in tracking people, while also having a simple programming interface, in OpenNI[4]. It therefore became the next logical candidate for inclusion in the framework. In order to have a state-of-the-art tracker without these limitations, and since the TLD [KMM12] tracker, and later Nebehay's improvements to it [Neb12], were open-sourced, we have proceeded to integrate it in the framework, completing the current set of available trackers.

We believe this framework constitutes an important tool for integrating tracking functionality into applications, by providing a common interface for developers to use the tracker most appropriate for their purposes, without needing to care for implementation details, or what kind of object representation it uses.

Let us now look at the challenges involved in integrating the framework in the existing video annotator and creating interface support for annotating tracked objects in real-time.

[3] A computer vision library, available at `http://opencv.willowgarage.com`.

[4] The offical Kinect SDK (available at `http://www.kinectforwindows.com`) was not yet available at the time we developed support for the Kinect.

5

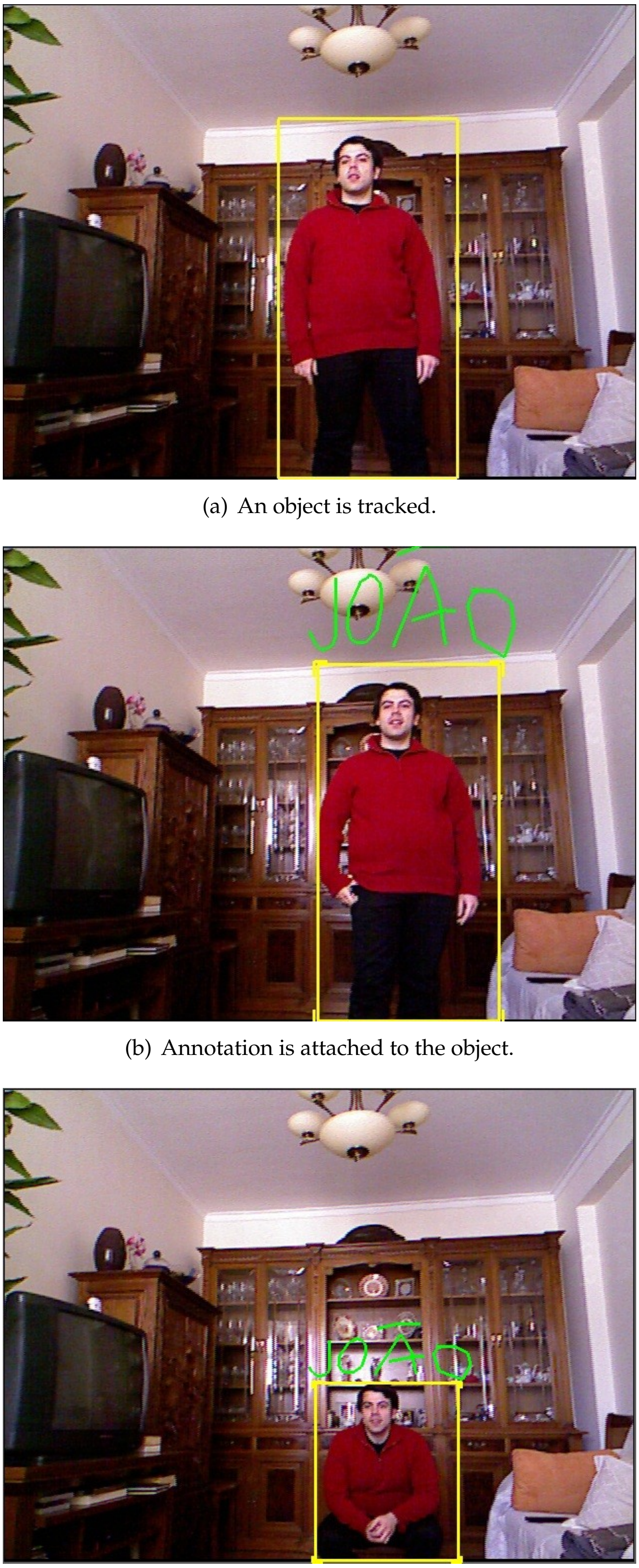
Real-Time Annotation of Video Objects

Having developed the object tracking framework, there were many challenges involved in integrating object tracking with video annotation, so the annotations maintain their association with an object on the scene. In this chapter we describe them, and our proposed solutions, starting with the task of integrating the tracking framework into the Creation Tool.

Before starting the integration process, however, a proof of concept to explore how to attach annotations to tracked objects was developed.

In order to test the feasibility and performance of attaching ink annotations to tracked objects, a proof of concept example was created. In it, users either drew an ink annotation to become an anchor for a tracked object, or anchors for automatically detected objects (with the Kinect) were created. Object anchors could then be selected, and annotations could be drawn and be attached to them, as can be seen in Figure 5.1. The user then knows a given note is attached to a given anchor, since the note follows the anchor's movement.

In this application, the object tracks were not saved, so all we had access to in each frame, was the object's current position and the object's starting position. In order to achieve the annotation following behavior, the position of the annotation currently being drawn, relative to the anchor was noted. The points comprising the new annotation were then translated and scaled so as to be in the same relative position to the object's *starting* position. This way, both the object anchor and its associated annotations could



(a) An object is tracked.

(b) Annotation is attached to the object.

(c) The annotation moves around with the object.

Figure 5.1: Proof of concept for attaching annotations to tracked objects.

be multiplied by the same transformation matrix, in order to appear in their correct positions in every frame. We have maintained this approach when integrating this into the Creation Tool, since our data structures make it easier for us to transform the annotations relative to the object's starting position, than relative to the position in the frame where the annotation was attached to it.

In this proof of concept, we did not concern ourselves with the annotations' lifetime. Once an annotation is attached to an object, that annotations follows that object until the object leaves the scene. This situation has changed in the Creation Tool. In the next sections, we shall describe the process of integrating these concepts into that annotator.

5.1 Integrating the Framework Into the Creation Tool

The Creation Tool offered users the possibility of creating ink, mark, text, link and voice annotations for their videos [Val11]. Integration of the framework into the Creation Tool proved difficult, for a few reasons, namely:

- Some of the annotation types had a lot of common functionality, without a lot of common code. At first, this generated the decision of tying tracked objects to digital ink annotations, such that an object anchor (see Section 5.3) was an ink annotation, and only ink annotations could be attached to it. This created a lot of complexity in the ink annotation class and generated many bugs. This plan was eventually abandoned, and the code was refactored to bring common functionality into a single class and allow other types of annotations to be attached to objects.

- The annotations were (understandably) designed to be stationary once created. Having moving and long-lasting annotations went against some assumptions made in the Tool's original design. For instance, annotation selection wasn't possible during recording. This made sense for annotations that were on-screen for around two seconds, but object anchors can remain there much longer, and need to be selected so that other annotations can be attached to them.

- Some of the used trackers are computationally expensive, which makes looking for a large number of objects infeasible. Worse, if those objects have permanently left the scene, computational power is being wasted on a useless endeavour.

The way the integration ended up being done was by almost completely separating the tracking functionality from the rest of the tool's functionality. A new type of annotation, the object anchor, was created. An object anchor is a rectangular annotation denoting the position of a tracked object, which follows that object around the scene. This separation allows most of the complexity of tracking to be isolated in one place, as well as removing some user confusion[1]. This confusion stemmed from not knowing *how* an object to track was going to be selected. Should the user draw an arrow pointing at it? Should he place a mark over it? Having it as a separate step eliminates this confusion.

Automatically selecting the last created object anchor worked even better. From our observation, users would assume that after creating an object, new notes would be attached to it.

To relate the objects currently being tracked to the stored anchors, we use a vector of integers, which has the same size as the vector returned by objectShapes. The integers denote the index of that tracked object in the object anchors vector, so we know which anchor is associated with which tracked object.

A `parent` element was inserted in ink, text, link and mark annotations, so they know which anchor they are attached to. Moving them is, then, a matter of translating and scaling them so they maintain their position relative to their parent object anchor. See Section 5.2 for details.

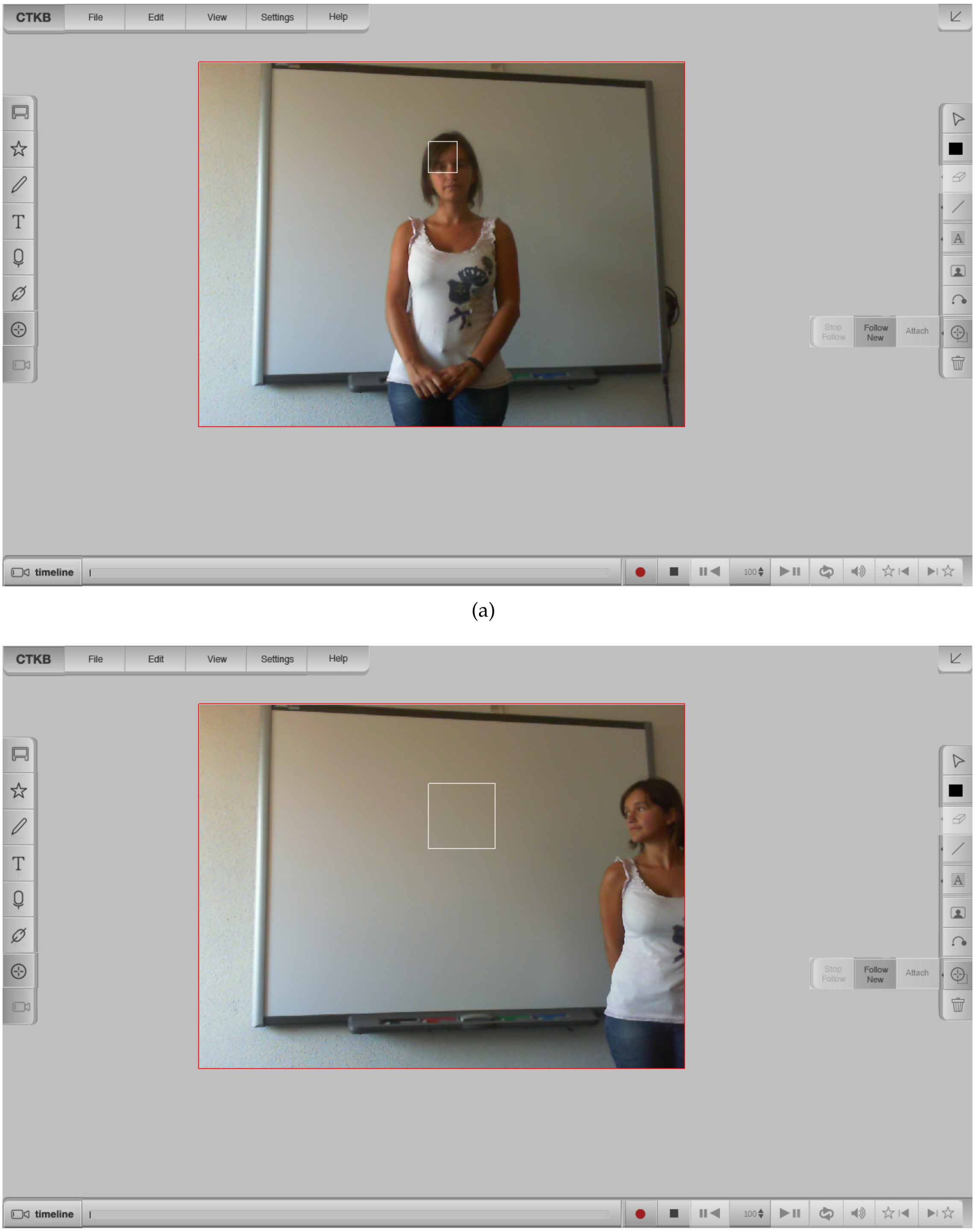
Since object tracking is computationally expensive[2], there was a need to stop looking for objects that have left the scene. To this end, we have implemented a system where an object ceases being tracked if it hasn't been seen for more than a set amount of time. This amount of time has been experimentally set at fifteen seconds, to ensure that there's a reasonable chance the object is re-detected before it is forgotten.

An important task of video annotations is serving as an anchor between a point of interest of any media document and additional information. For text documents, Brush et al. even stress that "without a good anchor, a system can't position annotations correctly in a document for display to users" [BBGC01]. In our users' case, manually creating good anchors means creating an accurate selection of the object we mean to track. One of the challenges involved in this is the creation of an interface to both create and support the association of additional annotations to that anchor. This involves the traditionally difficult task of selecting and annotating moving or editable objects [BM03, AHFMI11]. This difficulty is increased by the fact that on live or currently-playing video streams, users have a very limited time to react to what they are seeing and to decide whether or how they want to annotate it. If the scene is busy, or the features of interest in the video are moving fast, this time can be in the tens of milliseconds and whatever feature the user was annotating may have moved away while he/she was trying to select it.

Figure 5.2 illustrates the previously discussed problem. In 5.2(a), the user starts selection, in 5.2(b) the selection's starting point is no longer useful, since the object of interest has since moved away from its prior position. This means that by the time a user has finished drawing his note, its relevance to the object being annotated has been lost. Our approach to this problem was to use object tracking in order to enable the user to create anchors on the moving objects, so he/she can associate annotations to them. These annotations can then continue to be relevant in the face of changes in the scene. Let us then see how to accomplish this.

[1] For more details, user testing is discussed on Chapter 6.

[2] See Table 6.1 for details in our particular case.



(a)

(b)

Figure 5.2: Problems with selecting moving objects.

5.2 Using Object Tracking to Annotate Objects in Real-Time

As was previously mentioned, we need anchors to the actual objects of interest in the video, so we can annotate those objects at a later time. Ideally, these anchors should be created automatically. However, doing so has some disadvantages, among which:

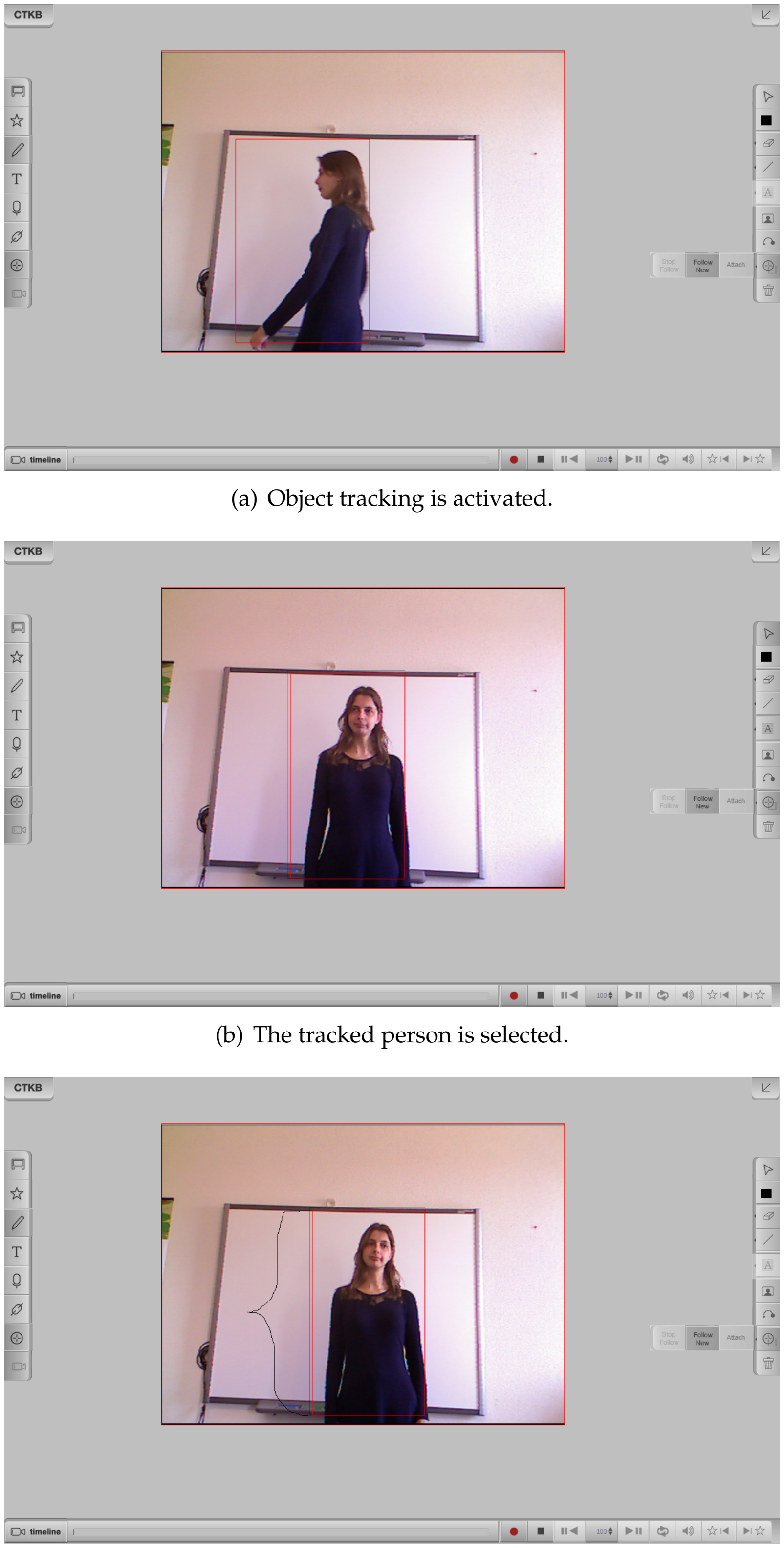
- Any automatic object tracker will only recognize a limited class of objects.
- Automatic object trackers may require specialized cameras or tracking dots on the object being tracked.

In the Creation Tool, it is possible to have anchors automatically created for people in the scene, when using the Microsoft Kinect to capture it. That solution does not work well when mobility is required, so users are also offered the possibility of creating their own anchors for later use, by using the TLD tracker. This also gives them the possibility of tracking arbitrary objects, not just humans.

In the case of the Creation Tool, tracking can be commenced by clicking the "motion tracking tool" button. The Creation Tool can perform object tracking with both the Kinect and TLD at the same time. Controlling when the Kinect starts tracking is simply a matter of pushing the "motion tracking tool", since it automatically detects people on the scene. The performer's current positions on the scene are then drawn as rectangles with different colors, so as to easily distinguish their tracks from one another. These rectangles shall then serve as anchors for further ink, text, link, and mark annotations.

Associating these kinds of annotations with the anchor can be done by selecting the anchor, and then creating the annotation as usual. This annotation then follows the tracked object around the scene, and is now associated with that tracked object. This process is pictured in Figure 5.3. Previously, the process involved an extra step: pressing an "Attach" button so the note would be attached to the anchor. This extra step proved confusing for users in our lightweight user testing, so it was removed in favor of having new notes following the currently selected anchor. In addition, achors for new objects the user has manually instructed to follow (via the "Follow New" button) become automatically selected.

All object tracks are saved, so they can be available both when using the Tool's delayed annotation mode -– where the user annotates the video of what happened a few seconds ago -– and at a later time, when playing back the video. Object tracks are saved as axis-aligned bounding boxes for each frame the object is in. The annotation following behavior is achieved by querying the parent object for its position in the frame where associated the annotation was drawn. When comparing this starting position with the object's current position, an appropriate transformation matrix can be computed. The annotation is then multiplied by this matrix in order to translate and scale the annotation to its correct place in every frame. Given the tracking framework's design, rotating the associated annotations along with the tracked object is trivial to implement. However, we have chosen against it, due to performance considerations, since that would require



(a) Object tracking is activated.

(b) The tracked person is selected.

(c) A digital ink annotation is associated with the tracked person and accompanies her movement throughout the scene.

Figure 5.3: Attaching annotations to a moving object.

calculating the smallest bounding rectangle of every tracked object in every frame, which can be expensive proposition when using complex object shapes, such as people's silhouettes, and on cluttered scenes. This is problematic in the Creation Tool's running environment, a Tablet which is already recording HD video, handling user interaction, and performing object tracking, in the worst case.

Despite its great effectiveness at tracking people, using the Kinect on the Creation Tool's intended mobile setting would be impractical at best. As previously mentioned, anchors for moving objects can also be created with TLD. This tracker requires a different interface, since it needs a hint to the object's initial position. In our annotator's case, this hint is merely a rectangle drawn with the pen. Selecting an object's position from a moving video stream is difficult, since between the time when a user started the selection and the time he ended it, the object may have changed its position. In the next section, we present our approaches to solving this problem, which we also use to select the moving anchors, and to associate new ink, link and mark annotations to them.

5.3 Creating and Using Anchors to Moving Objects

Selecting moving objects is a challenging proposition, for which Hajri et al. [AHFMI11] present a sensible and effective solution, the Hold method. The authors demonstrate that selecting moving objects in a game they created is easier when the objects' motion is stopped when the mouse button is down. Releasing the mouse button while the cursor is over an object then selects that object. While we also do freeze the video stream to facilitate moving object selection, our application differs from theirs in a few regards: a) in our application, when users want to create an anchor for an object, the application does not know where the object the user wishes to select is actually located, b) when a user is creating an anchor for an object, the task he is doing is actually drawing a rectangle around the feature of interest, not performing a selection, and c) using the Hold method on live video can make the user miss a few frames of the video that is being recorded. When taken together, a) and b) require that we either create an additional, specific step to have the user block the current frame, or that we hold the frame at the moment the user starts to draw the rectangle's first vertex. Since TLD can learn the object's shape and distinguish it from the background, it doesn't require the initial object hint to be perfect. We have therefore opted for the latter approach, which has the advantage of emphasizing the temporary nature of the pause, due to the user's motor system being temporarily engaged while the selection is lasting [MSB91]. A disadvantage of this approach is that, unlike in Hold, the position where the selection started matters, even if it does not need to be perfect. This discussion of problems a) and b) also applies to the selection of the anchors, and to the creation of new annotations associated wth them. Finally, to deal with c), we have devised two different approaches, which we have called "Hold and Overlay" and "Hold and Speed Up".

"Hold and Overlay", pictured in Figure 5.4, consists in drawing the frame where the

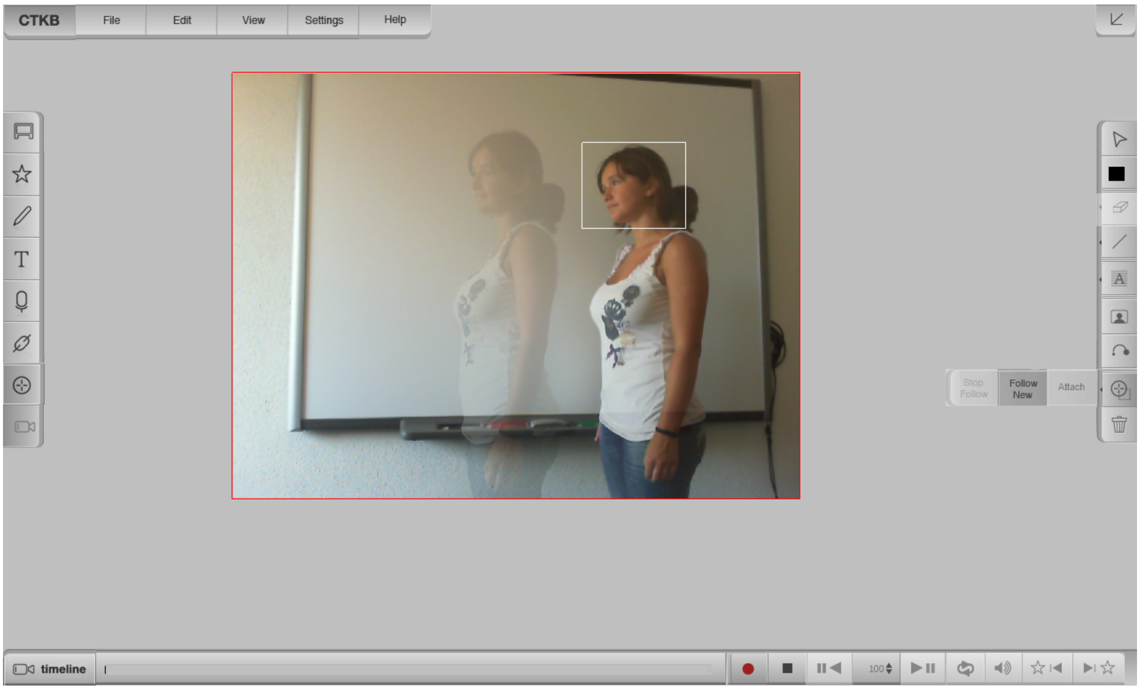


Figure 5.4: Hold and overlay.

user started the selection and then overlaying the translucent live video feed onto it, until the user finishes his selection. This enables the accurate selection of whatever the user wishes to track, while providing a cue that the video is still recording and showing what's currently on the scene. The live video's opacity has experimentally been set at 20%, and users have expressed that this selection mode will pose problems when it is used in busy scenes i.e. scenes with a lot of movement or noise. To alleviate this problem, users suggested to make the overlay's transparency customizable. We agree.

"Hold and Speed Up" works by freezing the frame while the user is selecting the object and, when that task is done, speed up the video, displaying the missed frames until it returns to the live video feed. Video recording is still happening in the background, both while the user is selecting the object he/she wants to track, and while the speed-up transition is occurring, as illustrated in Figure 5.5. Selection time, or t_s, is the amount of time the user takes to select the object. The current frame is held on screen during this time. Transition time, or t_t is the amount of time the speed-up transition is displayed. The sum of both these times is the select and speed up time, t_{ssu}.

Using this approach involves determining how long the speed up takes and in what fashion it is processed. The speed-up takes a set amount of time (t), which is proportional to the elapsed time since the user started holding the frame (t_e), following the following formula:

$$t_t = \frac{M_t \times t_e}{C + t_e} \tag{5.1}$$

Where M_t is the maximum amount of time we want the speed-up to take and C is a constant. The lower C is, the faster the climb toward M_t. We have experimentally set M_t

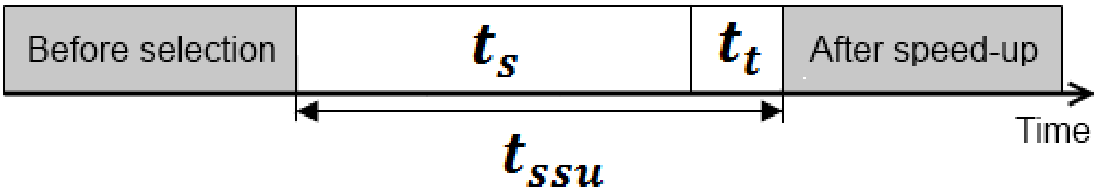


Figure 5.5: Select and speed-up timeline.

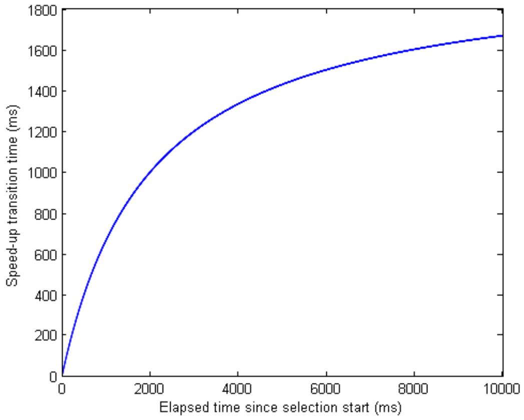


Figure 5.6: Transition time as a function of selection time.

at $2000ms$. We believe two seconds is a reasonable maximum amount of time to let the user know what is going on, while not taking too long to reach the live video again. We have also set C to 2000, since that ensures that the speed-up time is never greater than the selection time. Due to the fact that we are still recording frames while the speed-up is occurring, that equation is insufficient. We must compensate for those frames. This is currently done by doing an initial calculation of t_t, and then multiplying it by the video's frame rate, to see how many frames, at maximum, will be recorded during the speed-up. We then add that number to f_e and plug it into (5.1), and recalculate t_t. We keep iterating in this process until the change in t_t is less than a set precision, which we have set at half the time of a video frame, according to the video's frame rate. If the application is running slower than the video's frame rate, we can just return to live video sooner, so the consequences are not serious. It should be noted that this process started with $t_e = t_s$ and ended with our goal of $t_e = t_{ssu}$. The relationship between the elapsed time and the transition time is illustrated by the graph in Figure 5.6, which shows the graph of $g(x) = 2000 + \frac{x}{2000+x}$. It relates the amount of time elapsed since the user started holding the frame (in the x axis) to the amount of time the speed-up transition will take (in the y axis).

A way to find out the frame we should be currently displaying for the transition is needed. In order to achieve this, every frame since the user started the selection is recorded with a timestamp, relative to the instant the selection started. To retrieve the

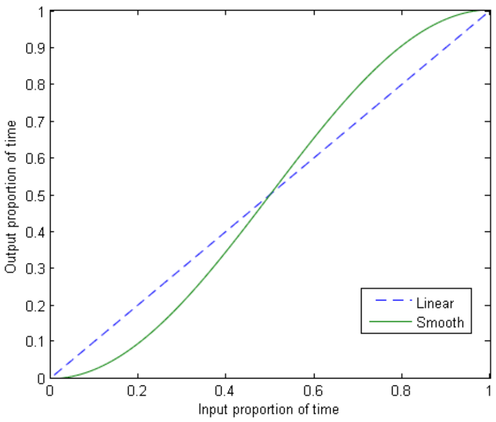


Figure 5.7: Linear speed-up vs smooth speed-up.

frame we want at a given point of the transition, we could linearly map the current instant in the transition (t_c) from transition time to the total select and speed up time, thus getting the current frame's time (f), like so:

$$f = t_{ssu} \times \frac{t_c}{t_t} \tag{5.2}$$

We would then display the first frame with a timestamp greater than f. However, rather than making the transition occur at a constant speed, we have decided to smooth it by gradually speeding it up at the beginning, continuing at a nearly constant speed and then slowing it down until we reach the live video, so the user can understand what is going on better. To do this, we calculate the current frame on the speed-up (f), according to the following formula:

$$f = t_{ssu} \times \frac{1 - cos(\frac{t_c}{t_t} \times \pi)}{2} \tag{5.3}$$

We can see how the speed-up occurs in Figure 5.7, which displays a graph of $h(x) = \frac{1-cos(x \times \pi)}{2}$, for $x \in [0, 1]$, which is the interval we are interested in. Alongside it, the linear speed-up is displayed, for comparison.

These approaches also work well when selecting previously created object anchors, since we also have the user draw a rectangle to perform selections in it. Let us now look at how we store these annotations for future reference.

5.4 Anchor and Annotation Storage

In keeping with how things were previously done in the Creation Tool [Val11], annotations are saved in an XML format. We have extended the previously used system, so that it can save anchors and the note's association to them. In order to do this, a new `parent`

Listing 5.1: An ink annotation element with a parent set.

```
<annotation_set>
    <annotation an:type="ink">
        <id>4</id>
        <text_recognized>Hello</text_recognized>
        <begin>245</begin>
        <end>678</end>
        <camera>1</camera>
        <parent>3</parent>
        <formatting>
            <color>
                <red>255</red>
                <blue>255</blue>
                <green>255</green>
            </color>
            <thickness>5.0</thickness>
        </formatting>
        <path>
            <x>23</x>
            <y>53</y>
            <x>35</x>
            <y>50</y>
        </path>
    </annotation>
</annotation_set>
```

element has been added to the ink, text, link and mark annotations, as seen on the example in Listing 5.1. The content of this element is the ID of a new type of element, called `<anchor>`.

The `anchor` element stores the details of that object anchor, such as its color and thickness, whether it was automatically created, and its frame by frame position, represented as a bounding rectangle. If the object was not found in a set of frames, a special invalid position marker is added to the file, thus making sure the anchor is not drawn for those frames. Otherwise, due to how the annotation lookup is implemented, the annotation would be drawn at anchor's next known position. A sample `anchor` element is shown in Listing 5.2.

Listing 5.2: An anchor annotation element.

```xml
<annotation_set>
    <annotation an:type="anchor">
        <id>5</id>
        <begin>123</begin>
        <end>287</end>
        <camera>1</camera>
        <is_auto>0</is_auto>
        <formatting>
            <color>
                <red>255</red>
                <blue>255</blue>
                <green>255</green>
            </color>
            <thickness>2.0</thickness>
        </formatting>
        <path>
            <frame num="123" x="412" y="353" w="201" h="304" />
            <frame num="124" x="409" y="345" w="203" h="300" />
            <frame num="125" x="400" y="335" w="198" h="294" />

            (...)

            <invalid_frames start="185" end="202" />

            (...)

            <frame num="287" x="197" y="453" w="175" h="270" />
        </path>
    </annotation>
</annotation_set>
```

6

Testing and Performance

The Creation Tool currently runs in a Lenovo X220 Tablet, with a 64-bit, 2.5 GHz Intel Core i5-2520M processor, 4 GB of RAM and an Intel HD Graphics 3000 GPU. Since this tablet does not have a backward-facing camera, our current solution for mobility is to use a 720p HD camera with 360-degree rotation, which we have attached to the Tablet. This camera is a Microsoft LifeCam HD-6000. For our stationary, indoor tests, we have connected a first generation Microsoft Kinect sensor to the Tablet.

Preliminary tests were carried out on a workshop with choreographers and dancers. Our informal discussions with them revealed that the trackers have worked satisfactorily, and that they were willing to use object tracking in their annotation workflow, mainly wishing to track persons and body parts. However, they had shown difficulty in selecting the objects they wished to track, as well as the annotations that were attached to them. While researching ways to deal with this problem, we have decided on adapting Hajri's approaches to moving object selection [AHFMI11] to our work, a process we have described in Section 5.3.

Table 6.1 shows how the Tool's performance is related to the number of tracked objects.

For our performance test, we've had up to two people in front of the Kinect's camera

Table 6.1: Performance measurements as a function of the number of tracked objects, in frames per second.

Kinect / TLD	0	1	2
0	30.21	10.61	5.49
1	30.08	9.68	5.85
2	28.88	9.88	5.91

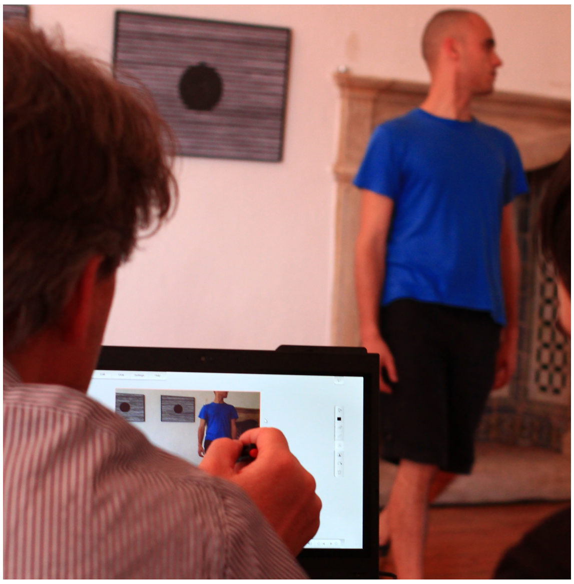


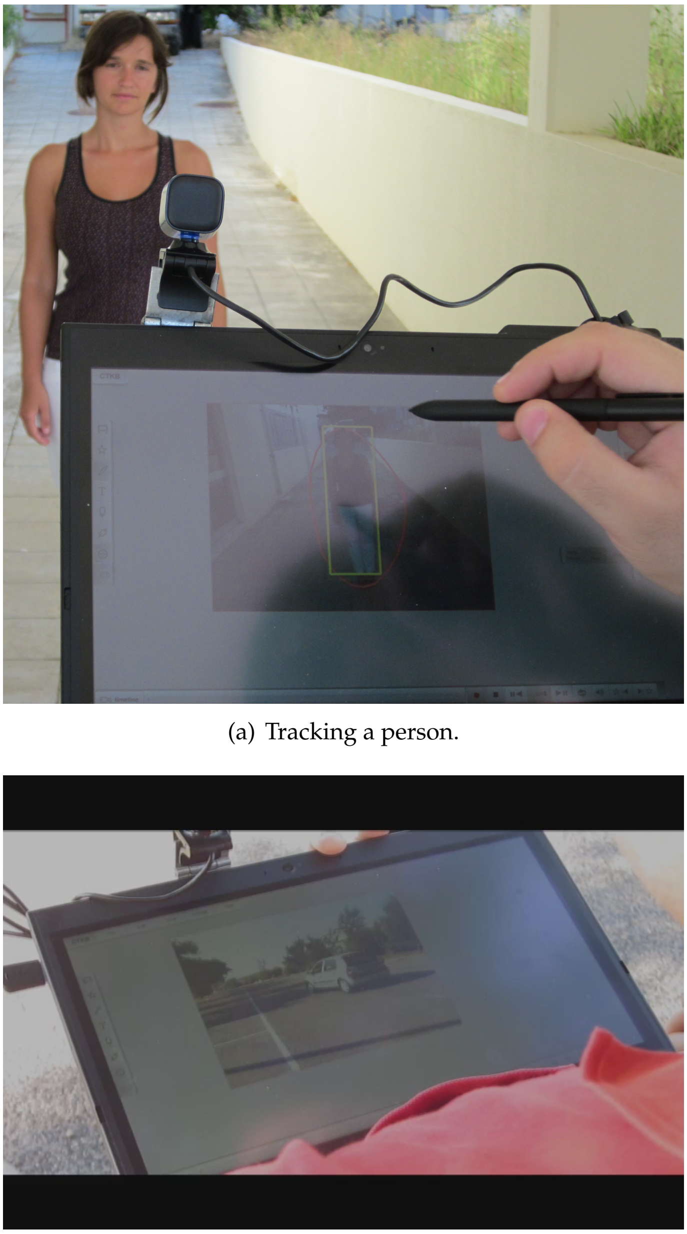
Figure 6.1: Preliminary indoor tests with choreographers and dancers.

and taken several samples of the amount of frames processed in a five second periods. The average of these frame rates is shown on the table. TLD trackers are tracking the people's faces. In the cases where there are two objects tracked by TLD, and less than that by the Kinect, TLD tracks a person's face, and the palm of his right hand, facing the camera. Predictably, tracking objects with the Kinect doesn't impact the performance as much as with TLD, since tracking with it is a per-pixel classification problem, and the number of pixels in the image remains the same. The slight increases in frame rate when tracking with TLD and increasing the number of tracked objects with the Kinect are small enough to be attributed to variations in TLD's initial rectangle, and statistical noise.

While a user selects an object to track, the frame where the selection started is being held by the application to help him with that action. Since we cannot start tracking until the user has finished the selection, we must miss tracking the object in several frames. To do otherwise would require immediately calculating the object's position for every frame recorded while the first frame was being held, which is not feasible. To account for this problem, videos are processed post-recording, to find the object's position in the missing frames.

We have also tested the Tool's outdoor performance and found out that the Kinect cannot be reliably used outdoors, even if a power outlet is available. We believe the cause of this to be that the Kinect's infrared camera cannot find the projected infrared dots in the presence of that much ambient light. TLD's outdoor use was satisfactory, even if somewhat sensitive to sudden illumination changes. The algorithm's learning component can deal with gradual changes and still remain following the object.

Further tests have shown, however, that when using one of the Hold methods for starting to track a new object, the object will not be found if it performed non-planar motion, since TLD is looking for a template which no longer exists in the image. When



(a) Tracking a person.

(b) Tracking a car.

Figure 6.2: Outdoor tests.

tracking in recordings, this is not a problem. In real-time, these methods remain useful both for selecting the moving anchors and annotations, and for attaching a new annotation to a selected anchor.

Additional lightweight user tests were done with researchers, PhD students and alumni, in order to ascertain the usability of tracking with Kinect and TLD, and of our two approaches to moving object selection.

6.1 Lightweight user testing

6.1.1 Users

We have questioned 9 non-expert users, 6 males (66.67%) and 3 females (33.33%), who were on average 31.7 years old, with a standard deviation of 6.4 years. Concerning the users' education, most of the users had a master's degree (66.67%), 22.22% of them had a bachelor's degree and 11.11% had a PhD. 88.89% of users were already familiar with with pen-based technology before. An overwhelming majority of the inquired users habitually take annotate their work (88.89%). From them, most used a paper notebook to do so (87.5%), with a significant portion also using a mobile phone (75%) or a laptop (50%).

6.1.2 Testing scenario

The users were given a brief explanation of the Creation Tool's purpose, followed by some experimentation with drawing regular ink annotations, while object tracking was deactivated, with a person moving in front of the camera. Object tracking was then activated and users were briefed on the existence of two trackers, A (the Kinect) and B (TLD). We then turned off our methods for moving object selection and asked the users to select the anchor produced by the Kinect, and to attach a circle drawn around that person to that anchor. Then, we asked the users to use TLD to follow the person and, again, to attach a circle drawn around that person. At this point, users were asked to rate their perceived tracking accuracy of both trackers when tracking people. Users were then asked to repeat this process using both "Hold and Overlay" and "Hold and Speed-Up", then proceeding to answer the rest of the questionnaire, which can be consulted, along with the raw results, in Appendix A.

6.1.3 Results analysis

Users have found the Kinect to be more accurate than TLD at tracking people. In a 1 to 5 scale, users gave the Kinect's accuracy an average score of 4.56, while TLD scored 3.33, with 1 being "Poor" and 5 being "Excellent". To determine the statistical significance of the difference between these two values, we have done a t-test, based on the null hypothesis that the averages would be equal, with an alpha level of 0.05, meaning our confidence in the results was 95%. This determined that the difference between them

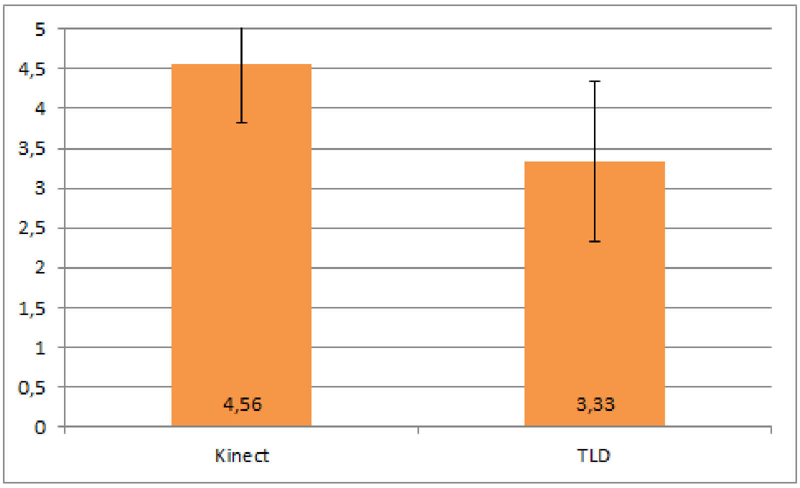


Figure 6.3: Users' evaluation of the trackers' accuracy.

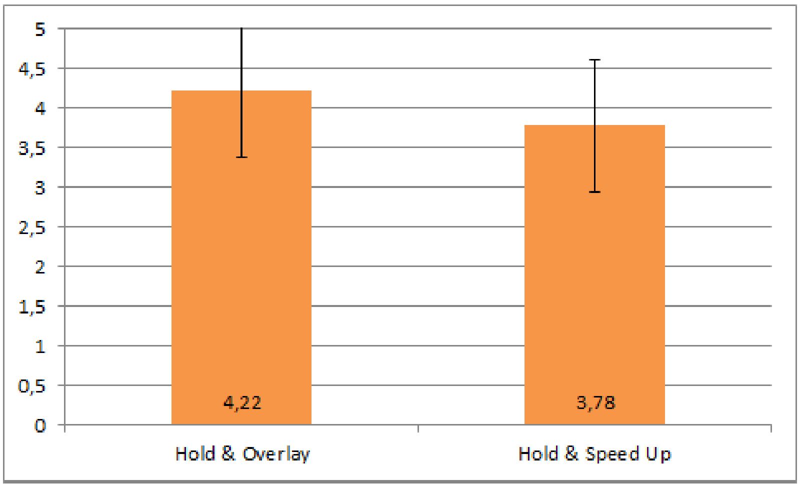


Figure 6.4: Users' evaluation of how frequently they would use each annotation method.

is statistically significant, since its p-value (0.0158) is significantly lower than 0.05. The means and standard deviations are presented in Figure 6.3.

Users were asked how frequently they would use each annotation method, and reported that they would use both "Hold and Overlay" and "Hold and Speed Up", with mean scores of 4.22 and 3.78, respectively, where 1 means "Rarely" and 5 means "Frequently". The p-value for the difference between annotation methods was 0.375 so we must conclude that while both methods are helpful, none of the two is significantly better than the other in these tests. Figure 6.4 illustrates the means and standard deviations of user's scores.

The next question asked the users to evaluate the ease of use of either tracker with either annotation method. The results are illustrated in Figure 6.5, where 1 means "Difficult" and 5 means "Easy". Both annotation modes exhibit similar scores, but with significant differences between trackers, with p-values of 0.00395 for "Hold and Overlay" and

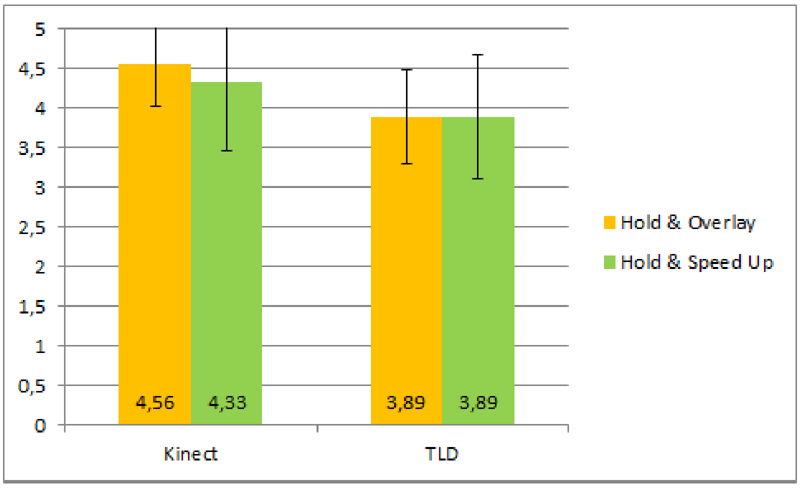


Figure 6.5: Users' opinion on how frequently they would use each annotation method.

0.035265 for "Hold and Speed-Up". We believe this to happen because we were comparing the tracking of people, and people are automatically tracked by the Kinect, without requiring the user to select the person he wishes to track.

The ease of attaching an annotation with a tracked object had a mean score of 4.33 out of 5, and the overall experience of using the Creation Tool with motion tracking was also rated 4.33 out of 5.

When characterizing their emotional experience, the most significant negative comment the users had was that the experience was "Complex" (22.22%). However, the experience has been overwhelmingly positive, with 77.78% of users describing it as as "Useful" and 55.56% describing it as "Attractive". The full results can be seen in Figure 6.6.

Based on users' open question responses, we have also gained some valuable feedback, namely that:

- Some users expressed confusion when seeing the overlay on "Hold and Overlay". It has been suggested that the overlay's opacity should be customizable. We agree.
- There were users who found that the transitions between the selection tool and the ink annotation tool should be streamlined. To meet this suggestion, we are going to make the selection tool detect that an anchor has been selected, and automatically revert to the annotation tool that was previously used.

In the end, though, we can conclude that these users' experience with using motion tracking on the Creation Tool was largely positive.

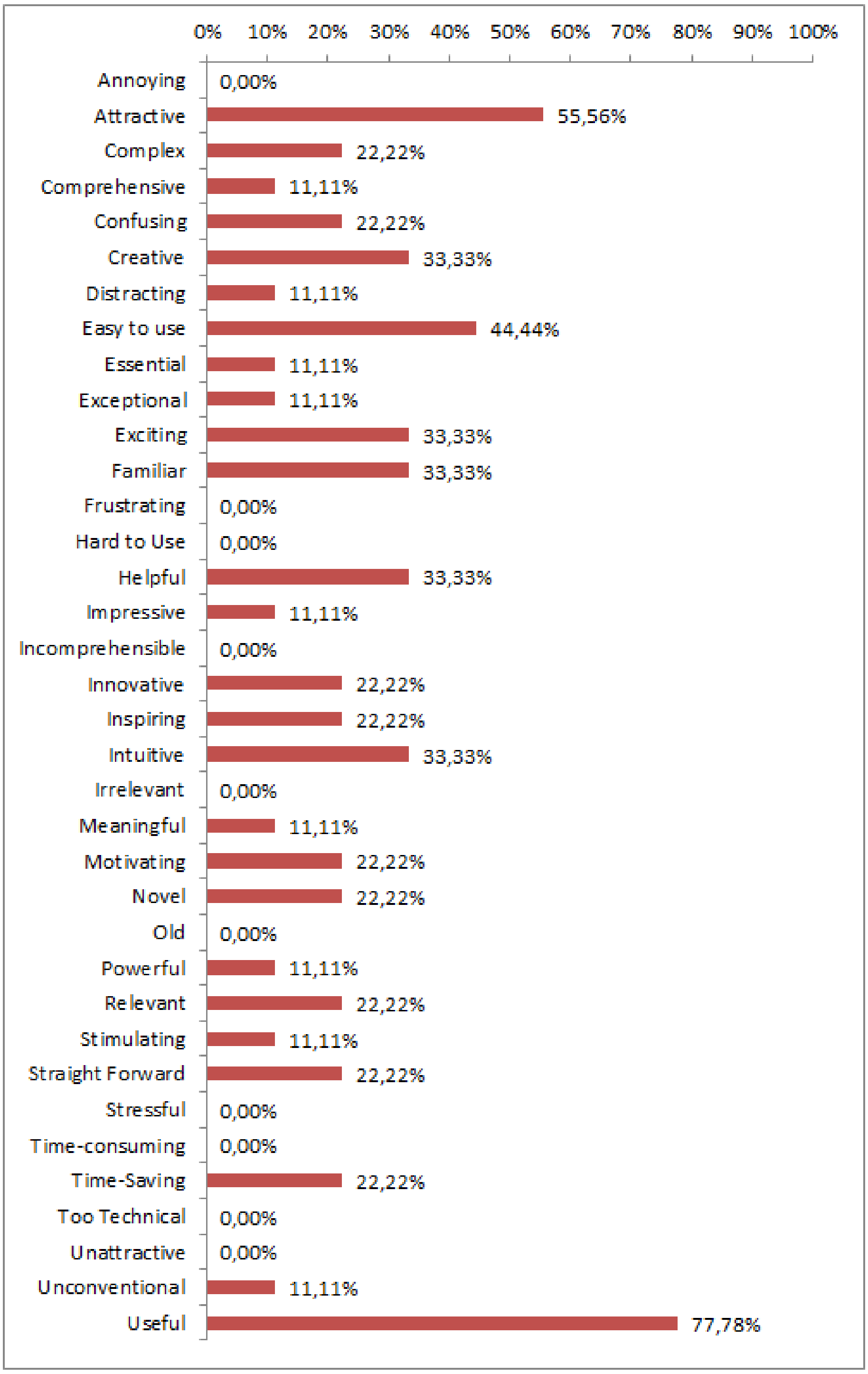


Figure 6.6: Users' emotional experiences in the tests.

7

Conclusion and Future Work

We have presented and described an approach to annotating video objects on a live video stream in real-time, using Tablets. This approach included the development of a generic, high-level object tracking framework and its integration into an existing video annotator, the Creation Tool. The framework currently includes support for the Kinect and for TLD, which are state-of-the-art trackers. For the future, the recent Compressive Tracking algorithm [ZZY12] is a prime candidate for inclusion in the framework.

The integration of the object tracking framework into the Creation Tool enables users to create anchors for objects in the scene, facilitating the use of multiple trackers in it. Using object tracking in the Tool allows annotations to be associated with objects on the scene over time.

We have also presented the problems involved in creating, selecting, and attaching annotations to moving objects. In order to solve them we have introduced a particular category of annotations, called “anchors”, which follow a moving object, and developed two real-time annotation methods, called “Hold and Overlay” and “Hold and Speed Up”. The first method holds a frame of the video, and overlays the live video feed onto it. The second one also holds a frame of the video but gradually speeds it up until the live feed is reached. Regular annotations can be attached onto anchors, so they retain a visual, moving association with the object they pertain to. We have found these methods to work well for their purpose. For the future, users suggested that we improve “Hold and Overlay” by making the overlay’s opacity customizable.

In order to reduce the need for buttons on the task of linking an annotation to a moving object, we have made it so that new annotations are attached to currently selected anchors. In our observation, this reduced user confusion compared to when there was a separate attaching step, which involved clicking an “Attach” button on the interface.

Users still demonstrated some confusion when having to switch between selection and annotation tools. To address this problem, in the future we will make the selection tool detect that an anchor was selected and immediately revert to the last used annotation tool.

We have performed informal, preliminary tests and found the tracking functionality satisfactory in indoor environments. In outdoor environments, TLD continues to work as expected, but it proved impossible to track with the Kinect. While we have performed additional lightweight user testing, it would be interest to perform tests with a larger sample size and with expert users in the future. For now, the lightweight user testing validated our approaches to moving object selection, although it did not suffice to determine which one is the better approach. It also showed positive evaluations for the concept of using motion tracking to aid video annotation, with the tested users considering the process easy.

We believe to have a strong contribution to the field of video annotation, by demonstrating the ability to annotate tracked objects in real time, thus allowing those annotations to maintain their context when the underlying scene changes. This contribution has been validated by the acceptance of three papers to international conferences [CCS+11, CVS+11, SCFC12], with [SCFC12] having the same main subject as this work. We hope to have improved the ability of annotators, such as choreographers, to do their work quickly and efficiently.

Bibliography

[AHFMI11] Abir Al Hajri, Sidney Fels, Gregor Miller, and Michael Ilich. Moving target selection in 2d graphical user interfaces. In *Proceedings of the 13th IFIP TC 13 international conference on Human-computer interaction - Volume Part II*, INTERACT'11, pages 141–161, Berlin, Heidelberg, 2011. Springer-Verlag.

[AP05] Olivier Aubert and Yannick Prié. Advene: active reading through hypervideo. In *HYPERTEXT '05: Proceedings of the sixteenth ACM conference on Hypertext and hypermedia*, pages 235–244, New York, NY, USA, 2005. ACM.

[BBGC01] A. J. Bernheim Brush, David Bargeron, Anoop Gupta, and J. J. Cadiz. Robust annotation positioning in digital documents. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, CHI '01, pages 285–292, New York, NY, USA, 2001. ACM.

[BETVG08] Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. Speeded-Up Robust Features (SURF). *Comput. Vis. Image Underst.*, 110(3):346–359, 2008.

[BFB94] J. L. Barron, D. J. Fleet, and S. S. Beauchemin. Performance of optical flow techniques. *Int. J. Comput. Vision*, 12(1):43–77, 1994.

[BJ98] Michael J. Black and Allan D. Jepson. EigenTracking: Robust Matching and Tracking of Articulated Objects Using a View-Based Representation. *Int. J. Comput. Vision*, 26(1):63–84, 1998.

[BK99] David Beymer and Kurt Konolige. Real-Time Tracking of Multiple People Using Continuous Detection. In *IEEE International Conference on Computer Vision (ICCV) Frame-Rate Workshop*, 1999.

[BM03] David Bargeron and Tomer Moscovich. Reflowing digital ink annotations. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, CHI '03, pages 385–393, New York, NY, USA, 2003. ACM.

[Bra98] G.R. Bradski. Real time face and object tracking as a component of a perceptual user interface. In *Proceedings of the Fourth IEEE Workshop on Applications of Computer Vision*, WACV '98, pages 214–219, Washington, DC, USA, oct 1998. IEEE Computer Society.

[BRN04] Hennie Brugman, Albert Russel, and Xd Nijmegen. Annotating multimedia / multimodal resources with elan. In *LREC 2004: Proceedings of Fourth International Conference on Language Resources and Evaluation*, pages 2065–2068, 2004.

[BSR00] Marcelo Bertalmío, Guillermo Sapiro, and Gregory Randall. Morphing Active Contours. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22:733–737, 2000.

[Bul04] Dick Bulterman. Animating peer-level annotations within web-based multimedia. In *EuroGraphics Multimedia Workshop*, pages 49–57. Eurographics Association, 2004.

[Can87] J. F. Canny. A computational approach to edge detection. *Readings in computer vision: issues, problems, principles, and paradigms*, pages 184–203, 1987.

[CC99] Nuno Correia and Teresa Chambel. Active video watching using annotation. In *Proceedings of the seventh ACM international conference on Multimedia (Part 2)*, MULTIMEDIA '99, pages 151–154, New York, NY, USA, 1999. ACM.

[CC09] Diogo Cabral and Nuno Correia. Pen-based video annotations: A proposal and a prototype for tablet pcs. In *Proceedings of the 12th IFIP TC 13 International Conference on Human-Computer Interaction: Part II*, INTERACT '09, pages 17–20, Berlin, Heidelberg, 2009. Springer-Verlag.

[CCGa02] Miguel Costa, Nuno Correia, and Nuno Guimarães. Annotations as multiple perspectives of video content. In *MULTIMEDIA '02: Proceedings of the tenth ACM international conference on Multimedia*, pages 283–286, New York, NY, USA, 2002. ACM.

[CCS$^+$11] Diogo Cabral, Urândia Carvalho, João Silva, João Valente, Carla Fernandes, and Nuno Correia. Multimodal video annotation for contemporary dance creation. In *Proceedings of the 2011 annual conference extended abstracts on Human factors in computing systems*, CHI EA '11, pages 2293–2298, New York, NY, USA, 2011. ACM.

[CET98] Timothy F. Cootes, Gareth J. Edwards, and Christopher J. Taylor. Active Appearance Models. In *Proceedings of the 5th European Conference on Computer Vision-Volume II*, ECCV '98, pages 484–498, London, UK, 1998. Springer-Verlag.

[CFS03] Gina Cherry, Janice Fournier, and Reed Stevens. Using a digital video annotation tool to teach dance composition. *Interactive Multimedia Electronic Journal of Computer-Enhanced Learning*, 2003.

[CLSF10] Michael Calonder, Vincent Lepetit, Christoph Strecha, and Pascal Fua. Brief: binary robust independent elementary features. In *Proceedings of the 11th European conference on Computer vision: Part IV*, ECCV'10, pages 778–792, Berlin, Heidelberg, 2010. Springer-Verlag.

[CM02] Dorin Comaniciu and Peter Meer. Mean Shift: A Robust Approach Toward Feature Space Analysis. *IEEE Trans. Pattern Anal. Mach. Intell.*, 24(5):603–619, 2002.

[CRM03] Dorin Comaniciu, Visvanathan Ramesh, and Peter Meer. Kernel-Based Object Tracking. *IEEE Trans. Pattern Anal. Mach. Intell.*, 25(5):564–575, 2003.

[CTGP08] Renan G. Cattelan, Cesar Teixeira, Rudinei Goularte, and Maria Da Graça C. Pimentel. Watch-and-comment as a paradigm toward ubiquitous interactive video editing. *ACM Trans. Multimedia Comput. Commun. Appl.*, 4(4):1–24, 2008.

[CVAa$^+$12] Diogo Cabral, João G. Valente, Urândia Aragão, Carla Fernandes, and Nuno Correia. Evaluation of a multimodal video annotator for contemporary dance. In *Proceedings of the International Working Conference on Advanced Visual Interfaces*, AVI '12, pages 572–579, New York, NY, USA, 2012. ACM.

[CVS$^+$11] Diogo Cabral, João Valente, João Silva, Urândia Aragão, Carla Fernandes, and Nuno Correia. A creation-tool for contemporary dance using multimodal video annotation. In *Proceedings of the 19th ACM international conference on Multimedia*, MM '11, pages 905–908, New York, NY, USA, 2011. ACM.

[DE06] Nicholas Diakopoulos and Irfan Essa. Videotater: an approach for pen-based digital video segmentation and tagging. In *Proceedings of the 19th annual ACM symposium on User interface software and technology*, UIST '06, pages 221–224, New York, NY, USA, 2006. ACM.

[DGE09] Nicholas Diakopoulos, Sergio Goldenberg, and Irfan Essa. Videolyzer: quality analysis of online informational video for bloggers and journalists. In *CHI '09: Proceedings of the 27th international conference on Human factors in computing systems*, pages 799–808, New York, NY, USA, 2009. ACM.

[FJ90] David J. Fleet and A. D. Jepson. Computation of component image velocity from local phase information. *Int. J. Comput. Vision*, 5(1):77–104, 1990.

[FT97] Paul Fieguth and Demetri Terzopoulos. Color-Based Tracking of Heads and Other Mobile Objects at Video Frame Rates. In *CVPR '97: Proceedings of the 1997 Conference on Computer Vision and Pattern Recognition (CVPR '97)*, page 21, Washington, DC, USA, 1997. IEEE Computer Society.

[GCD11] Gene Golovchinksy, Scott Carter, and Anthony Dunnigan. Ara: the active reading application. In *Proceedings of the 19th ACM international conference on Multimedia*, MM '11, pages 799–800, New York, NY, USA, 2011. ACM.

[GCGI$^+$04] R. Goularte, J.A. Camacho-Guerrero, Jr. Inacio, V.R., R.G. Cattelan, and M.G.C. Pimentel. M4note: a multimodal tool for multimedia annotations. In *WebMedia and LA-Web, 2004. Proceedings*, pages 142 – 149, oct. 2004.

[GGC$^+$08] Dan B. Goldman, Chris Gonterman, Brian Curless, David Salesin, and Steven M. Seitz. Video object annotation, navigation, and composition. In *UIST '08: Proceedings of the 21st annual ACM symposium on User interface software and technology*, pages 3–12, New York, NY, USA, 2008. ACM.

[GRD05] Edward Rosten Gerhard, Gerhard Reitmayr, and Tom Drummond. Real-time Video Annotations for Augmented Reality. In *In: International Symposium on Visual Computing*, pages 294–302, 2005.

[HHD00] Ismail Haritaoglu, Davis Harwood, and Larry S. David. W^4: Real-Time Surveillance of People and Their Activities. *IEEE Trans. Pattern Anal. Mach. Intell.*, 22(8):809–830, 2000.

[HHK08] Joey Hagedorn, Joshua Hailpern, and Karrie G. Karahalios. VCode and VData: illustrating a new framework for supporting the video annotation workflow. In *AVI '08: Proceedings of the working conference on Advanced visual interfaces*, pages 317–321, New York, NY, USA, 2008. ACM.

[HKR93] D.P. Huttenlocher, G.A. Klanderman, and W.A. Rucklidge. Comparing Images Using the Hausdorff Distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15:850–863, 1993.

[IM01] M. Isard and J. MacCormick. BraMBLe: A Bayesian Multiple-Blob Tracker. In *Proceedings of the IEEE International Conference on Computer Vision*, volume 2, page 34, Los Alamitos, CA, USA, 2001. IEEE Computer Society.

[ISO02] ISO. *ISO/IEC 15938:2002 - MPEG-7: Multimedia content description interface*. ISO, 2002.

[JAM02] Rui M. Jesus, Arnaldo J. Abrantes, and Jorge S. Marques. Tracking the human body using multiple predictors. In *AMDO '02: Proceedings of the Second International Workshop on Articulated Motion and Deformable Objects*, pages 155–164, London, UK, 2002. Springer-Verlag.

[KCM04] Jinman Kang, Isaac Cohen, and Gerard Medioni. Tracking people in crowded scenes across multiple cameras. In *Asian Conference on Computer Vision 2004*, 2004.

[KMM12] Zdenek Kalal, Krystian Mikolajczyk, and Jiri Matas. Tracking-learning-detection. *IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE*, 34(7):1409–1422, July 2012.

[KZL03] Wai Kin Kong, David Zhang, and Wenxin Li. Palmprint feature extraction using 2-d gabor filters. *Pattern Recognition*, 36(10):2339 – 2347, 2003.

[LCZD01] Baoxin Li, R. Chellappa, Qinfen Zheng, and S.Z. Der. Model-based temporal object verification using video. *IEEE Transactions on Image Processing*, 10(6):897 –908, jun 2001.

[LK81] Bruce D. Lucas and Takeo Kanade. An iterative image registration technique with an application to stereo vision. In *IJCAI'81: Proceedings of the 7th international joint conference on Artificial intelligence*, pages 674–679, San Francisco, CA, USA, 1981. Morgan Kaufmann Publishers Inc.

[Low99] D.G. Lowe. Object recognition from local scale-invariant features. In *Proceedings of the Seventh IEEE International Conference on Computer Vision, 1999*, volume 2, pages 1150–1157, Washington, DC, USA, 1999. IEEE Computer Society.

[Low04] David G. Lowe. Distinctive image features from scale-invariant keypoints. *Int. J. Comput. Vision*, 60(2):91–110, 2004.

[Mac89] W. E. Mackay. Eva: an experimental video annotator for symbolic analysis of video data. *SIGCHI Bulletin*, 21(2):68–71, 1989.

[Mar98] Catherine C. Marshall. Toward an ecology of hypertext annotation. In *Proceedings of the ninth ACM conference on Hypertext and hypermedia : links, objects, time and space—structure in hypermedia systems: links, objects, time and space—structure in hypermedia systems*, HYPERTEXT '98, pages 40–49, New York, NY, USA, 1998. ACM.

[MFAH$^+$11] Gregor Miller, Sidney Fels, Abir Al Hajri, Michael Ilich, Zoltan Foley-Fisher, Manuel Fernandez, and Daesik Jang. Mediadiver: viewing and annotating multi-view video. In *Proceedings of the 2011 annual conference extended abstracts on Human factors in computing systems*, CHI EA '11, pages 1141–1146, New York, NY, USA, 2011. ACM.

[MG01] Thomas B. Moeslund and Erik Granum. A survey of computer vision-based human motion capture. *Comput. Vis. Image Underst.*, 81(3):231–268, 2001.

[MHK06] Thomas B. Moeslund, Adrian Hilton, and Volker Krüger. A survey of advances in vision-based human motion capture and analysis. *Comput. Vis. Image Underst.*, 104(2):90–126, 2006.

[Mor81] Hans P. Moravec. Rover visual obstacle avoidance. In *IJCAI'81: Proceedings of the 7th international joint conference on Artificial intelligence*, pages 785–790, San Francisco, CA, USA, 1981. Morgan Kaufmann Publishers Inc.

[MS05] Krystian Mikolajczyk and Cordelia Schmid. A Performance Evaluation of Local Descriptors. *IEEE Trans. Pattern Anal. Mach. Intell.*, 27(10):1615–1630, 2005.

[MSB91] I. Scott MacKenzie, Abigail Sellen, and William A. S. Buxton. A comparison of input devices in element pointing and dragging tasks. In *Proceedings of the SIGCHI conference on Human factors in computing systems: Reaching through technology*, CHI '91, pages 161–166, New York, NY, USA, 1991. ACM.

[Neb12] Georg Nebehay. Robust object tracking based on tracking-learning-detection. Master's thesis, Faculty of Informatics, TU Vienna, 2012.

[NTM07] Helmut Neuschmied, Remi Trichet, and Berard Merialdo. Fast annotation of video objects for interactive TV. In *MULTIMEDIA '07: Proceedings of the 15th international conference on Multimedia*, pages 158–159, New York, NY, USA, 2007. ACM.

[PEP98] Constantine Papageorgiou, Theodoros Evgeniou, and Tomaso Poggio. A Trainable Pedestrian Detection System. In *Proceedings of the 1998 IEEE International Conference on Intelligent Vehicles*, 1998.

[Pie08] Pierre Dragicevic and Gonzalo Ramos and Jacobo Bibliowitcz and Derek Nowrouzezahrai and Ravin Balakrishnan and Karan Singh. Video Browsing by Direct Manipulation. In *CHI '08: Proceeding of the twenty-sixth annual SIGCHI conference on Human factors in computing systems*, page 237–246, New York, NY, USA, April 2008. ACM.

[Pop07] Ronald Poppe. Vision-based human motion analysis: An overview. *Comput. Vis. Image Underst.*, 108(1-2):4–18, 2007.

[RB03] Gonzalo Ramos and Ravin Balakrishnan. Fluid interaction techniques for the control and annotation of digital video. In *Proceedings of the 16th annual ACM symposium on User interface software and technology*, UIST '03, pages 105–114, New York, NY, USA, 2003. ACM.

[RD06] Edward Rosten and Tom Drummond. Machine learning for high-speed corner detection. In *Proceedings of the 9th European conference on Computer Vision*

- *Volume Part I*, ECCV'06, pages 430–443, Berlin, Heidelberg, 2006. Springer-Verlag.

[RPD10] Edward Rosten, Reid Porter, and Tom Drummond. Faster and better: A machine learning approach to corner detection. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 32:105–119, 2010.

[RSK02] Jeewoong Ryu, Yumi Sohn, and Munchuri Kim. MPEG-7 metadata authoring tool. In *MULTIMEDIA '02: Proceedings of the tenth ACM international conference on Multimedia*, pages 267–270, New York, NY, USA, 2002. ACM.

[SBW02] Haim Schweitzer, J. W. Bell, and F. Wu. Very Fast Template Matching. In *ECCV '02: Proceedings of the 7th European Conference on Computer Vision-Part IV*, pages 358–372, London, UK, 2002. Springer-Verlag.

[SCFC12] João Silva, Diogo Cabral, Carla Fernandes, and Nuno Correia. Real-time annotation of video objects on tablet computers. In *Proceedings of the 11th International Conference on Mobile and Ubiquitous Multimedia*, MUM '12, pages 19:1–19:9, New York, NY, USA, 2012. ACM.

[SFC$^+$11] J. Shotton, A. Fitzgibbon, M. Cook, T. Sharp, M. Finocchio, R. Moore, A. Kipman, and A. Blake. Real-time human pose recognition in parts from single depth images. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 1297 –1304, june 2011.

[SM00] Jianbo Shi and Jitendra Malik. Normalized Cuts and Image Segmentation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 22(8):888–905, 2000.

[SS05] Khurram Shafique and Mubarak Shah. A Noniterative Greedy Algorithm for Multiframe Point Correspondence. *IEEE Trans. Pattern Anal. Mach. Intell.*, 27(1):51–65, 2005.

[ST94] Jianbo Shi and Carlo Tomasi. Good Features to Track. In *1994 IEEE Conference on Computer Vision and Pattern Recognition (CVPR'94)*, pages 593 – 600, 1994.

[Str94] Bjarne Stroustrup. *The Design and Evolution of* C++. Addison-Wesley Professional, 1994.

[TSK02] Hai Tao, Harpreet S. Sawhney, and Rakesh Kumar. Object Tracking with Bayesian Estimation of Dynamic Layer Representations. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24:75–89, 2002.

[Val11] João Gaspar Valente. Sistema multimodal para captura e anotação de vídeo. Master's thesis, Universidade Nova de Lisboa, 2011.

[VJS05] Paul Viola, Michael J. Jones, and Daniel Snow. Detecting pedestrians using patterns of motion and appearance. *Int. J. Comput. Vision*, 63(2):153–161, 2005.

[VRB01] Cor J. Veenman, Marcel J. T. Reinders, and Eric Backer. Resolving motion correspondence for densely moving points. *IEEE Trans. Pattern Anal. Mach. Intell.*, 23(1):54–72, 2001.

[WADP97] Christopher Richard Wren, Ali Azarbayejani, Trevor Darrell, and Alex Paul Pentland. Pfinder: Real-time tracking of the human body. *IEEE Trans. Pattern Anal. Mach. Intell.*, 19(7):780–785, 1997.

[WL93] Z. Wu and R. Leahy. An optimal graph theoretic approach to data clustering: Theory and its application to image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15:1101–1113, 1993.

[YJS06] Alper Yilmaz, Omar Javed, and Mubarak Shah. Object tracking: A survey. *ACM Comput. Surv.*, 38(4):13, Dec 2006.

[YLS04] Alper Yilmaz, Xin Li, and Mubarak Shah. Contour-Based Object Tracking with Occlusion Handling in Video Acquired Using Mobile Cameras. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26:1531–1536, 2004.

[ZvdH06] Zoran Zivkovic and Ferdinand van der Heijden. Efficient adaptive density estimation per image pixel for the task of background subtraction. *Pattern Recogn. Lett.*, 27:773–780, May 2006.

[ZZY12] Kaihua Zhang, Lei Zhang, and Ming-Hsuan Yang. Real-time compressive tracking. In *Proceedings of the 2012 European Conference on Computer Vision, part III*, ECCV 2012, pages 864–877, Berlin, Germany, 2012. Springer-Verlag.

A

User testing questionnaire and results

A.1 Questionnaire

CTKB

FCSH FACULDADE DE CIÊNCIAS SOCIAIS E HUMANAS UNIVERSIDADE NOVA DE LISBOA

FCT FACULDADE DE CIÊNCIAS E TECNOLOGIA UNIVERSIDADE NOVA DE LISBOA

Questionnaire – Creation-Tool & Motion Tracking

1. **Considering the two trackers, A (Kinect) and B (TLD), and their accuracy on tracking people. Please classify the accuracy of each tracker:**

	Poor				Excellent
A	1	2	3	4	5
B	1	2	3	4	5

2. **Consider the two techniques of video annotation with motion tracking, Hold & Overlay and Hold and Speed Up, recording a live event:**

   Hold & Overlay – the video is paused during the task of the annotation and a half-transparent video window is overlaid, showing the live event and until the task ends.

   Hold & Speed Up – the video is paused during the task of the annotation and played it after, in fastfoward, until the video is synchronized with the live event.

   Please, classify how often are you willing to use each type.

	Rarely				Frequently
Hold & Overlay	1	2	3	4	5
Hold & Speed Up	1	2	3	4	5

3. **Consider the two techniques of video annotation with motion tracking, Hold & Overlay and Hold and Speed Up, as well as the tracking methods A (Kinect) and B (TLD). Please, classify them.**

Annot. Method	Tracker	Difficult				Easy
Hold & Overlay	A	1	2	3	4	5
	B	1	2	3	4	5
Hold & Speed Up	A	1	2	3	4	5
	B	1	2	3	4	5

4. Consider the task of associating an annotation with a tracked object or person. Please, classify it.

Difficult Easy

1 2 3 4 5

5. Consider the annotation with motion tracking. Please, classify it.

Difficult Easy

1 2 3 4 5

6. Choose the expression(s) that better classifies your experience with the Creation-Tool & Motion Tracking.

□ Annoying
□ Attractive
□ Complex
□ Comprehensive
□ Confusing
□ Creative
□ Distracting
□ Easy to use
□ Essential
□ Exceptional
□ Exciting
□ Familiar
□ Frustrating
□ Hard to Use
□ Helpful
□ Impressive
□ Incomprehensible
□ Innovative
□ Inspiring
□ Intuitive
□ Irrelevant
□ Meaningful
□ Motivating
□ Novel
□ Old
□ Powerful
□ Relevant
□ Stimulating
□ Straight Forward
□ Stressful
□ Time-consuming
□ Time-Saving
□ Too Technical
□ Unattractive
□ Unconventional
□ Useful

7. Comments and Suggestions:

____

____

____

____

____

____

____

8. **Do you usually annotate during your work process?** □ Yes □ No

If <u>yes</u>, in which technology?

□ Paper Notebook
□ Desktop Computer
□ Laptop Computer
□ Mobile Phone
□ PDA
□ Tablet
□ Other: _______________

9. **Were you familiar with pen-based technology (Tablets or PDAs) before:** □ Yes □ No

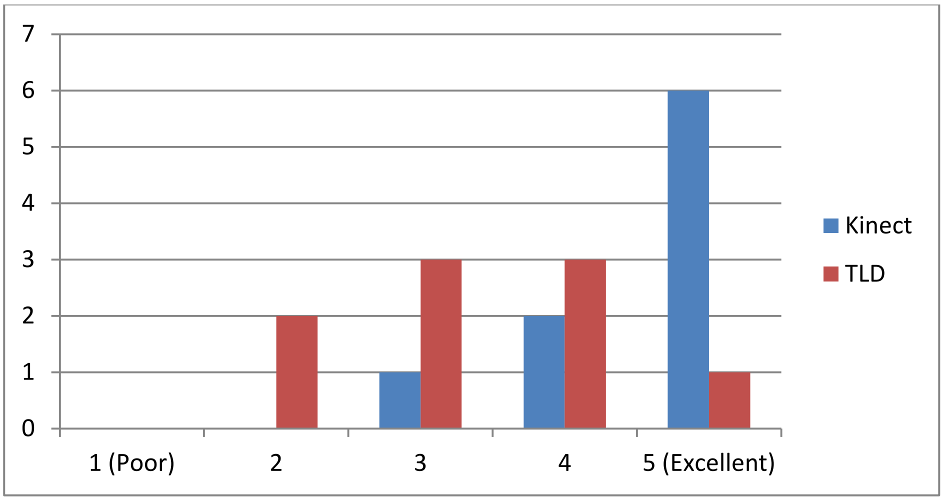
10. **Age:_____**

11. **Gender:** □ Male □ Female

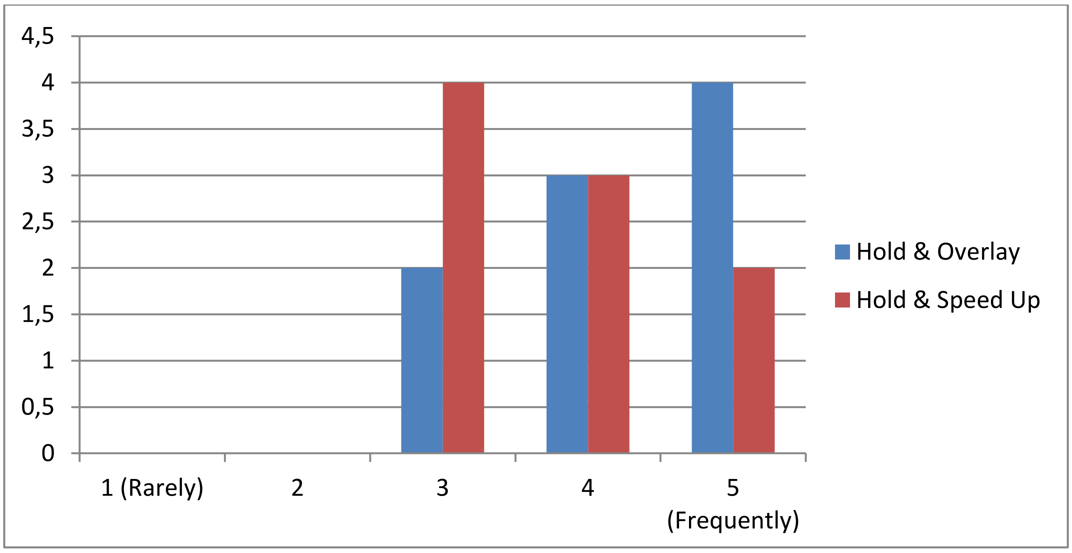
12. **Education:** □ Elementary School □ High School □ Bachelor □ Master □ PhD

A.2 Results

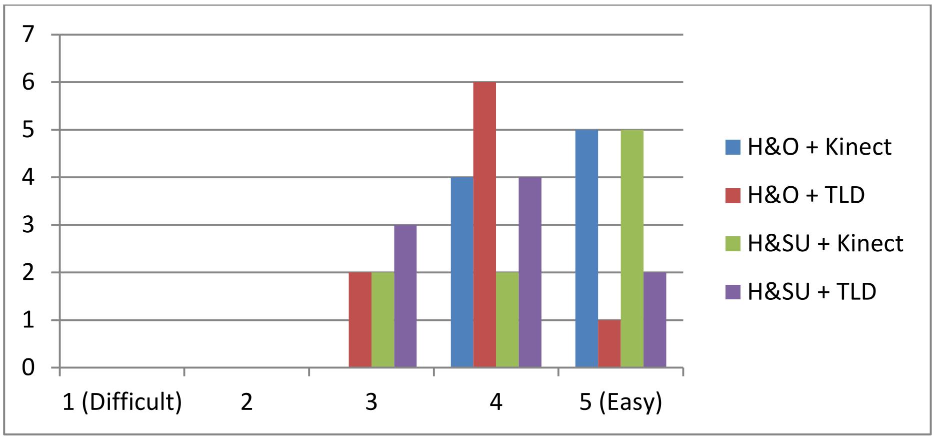
1. **Considering the two trackers, A (Kinect) and B (TLD), and their accuracy on tracking people. Please classify the accuracy of each tracker:**



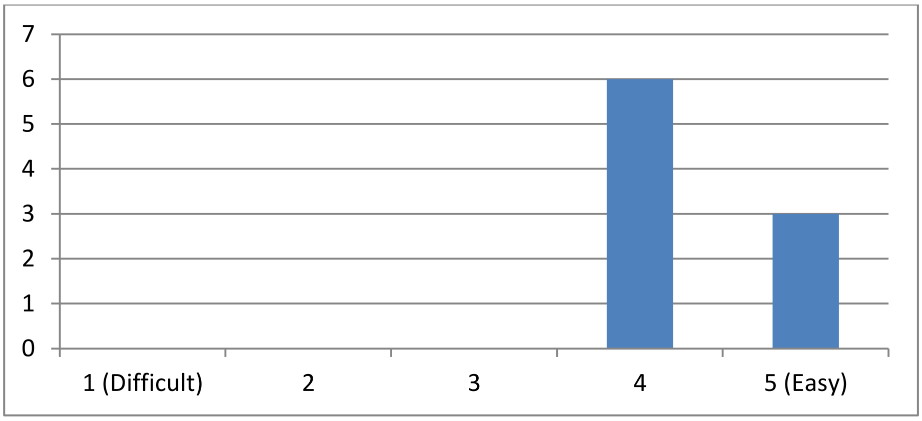
2. **Consider the two techniques of video annotation with motion tracking, Hold & Overlay and Hold and Speed Up, recording a live event:**
Hold & Overlay – the video is paused during the task of the annotation and a half-transparent video window is overlaid, showing the live event and until the task ends.
Hold & Speed Up – the video is paused during the task of the annotation and played it after, in fastfoward, until the video is synchronized with the live event.
Please, classify how often are you willing to use each type.



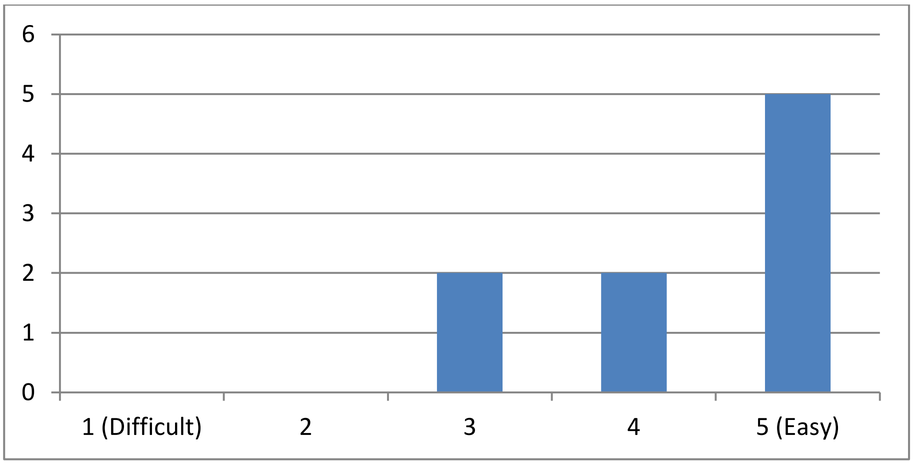
3. **Consider the two techniques of video annotation with motion tracking, Hold & Overlay and Hold and Speed Up, as well as the tracking methods A (Kinect) and B (TLD). Please, classify them.**



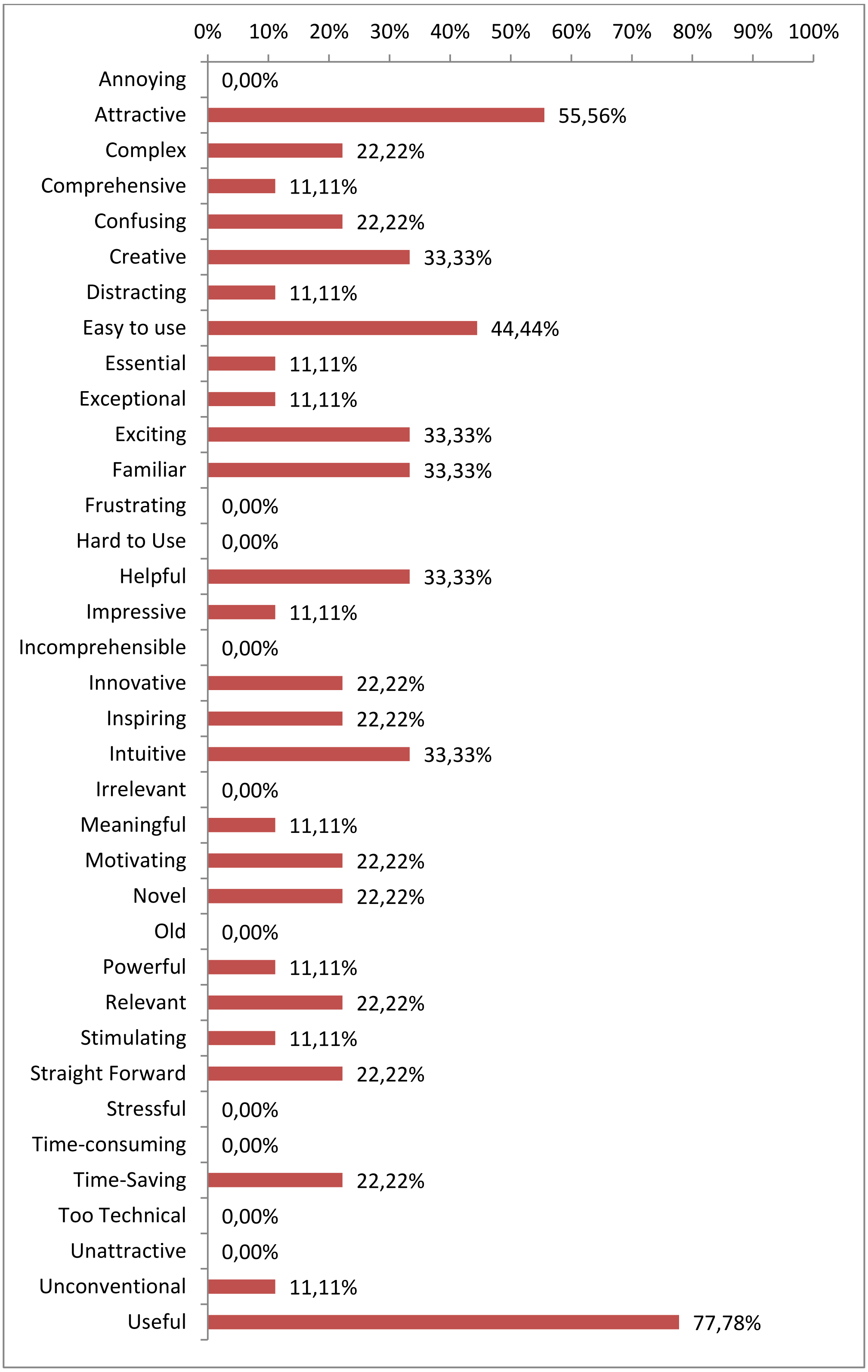
4. **Consider the task of associating an annotation with a tracked object or person. Please, classify it.**



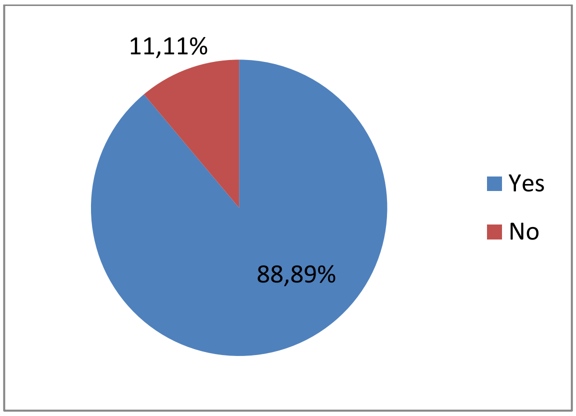
5. **Consider the annotation with motion tracking. Please, classify it.**



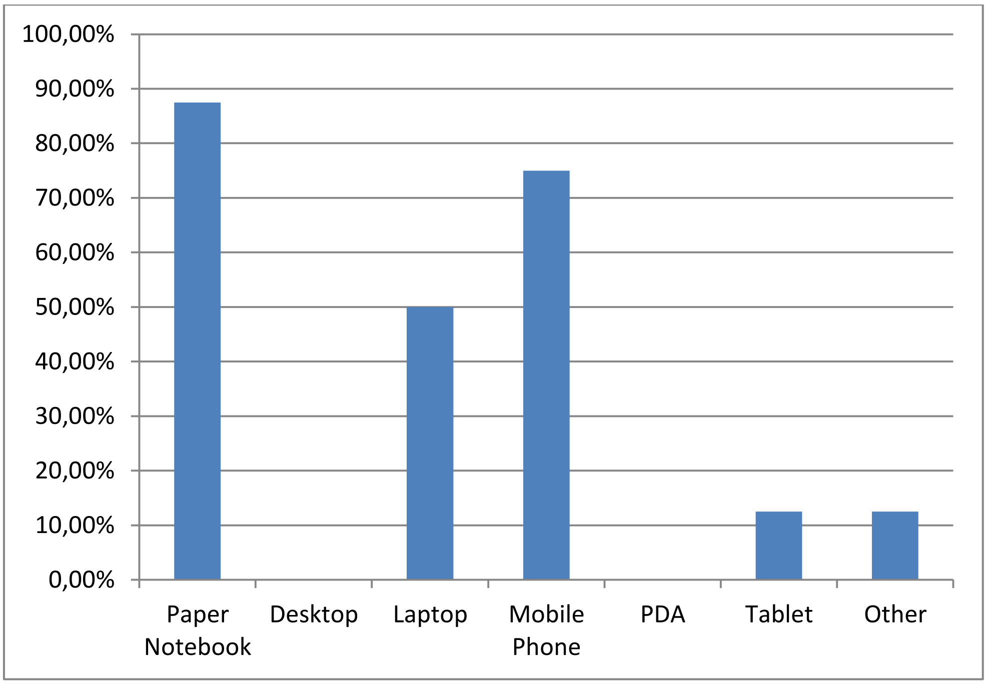
6. **Choose the expression(s) that better classifies your experience with the Creation-Tool & Motion Tracking.**



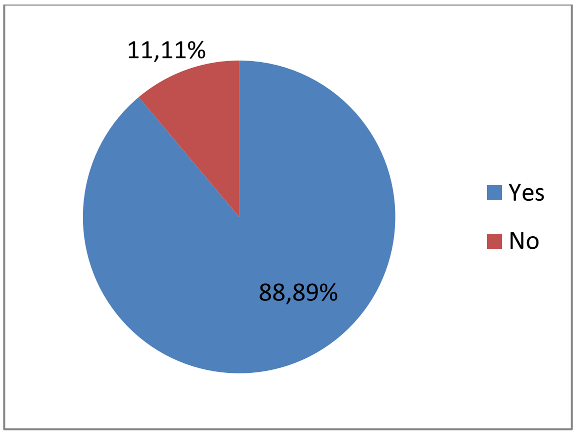
8. **Do you usually annotate during your work process?**



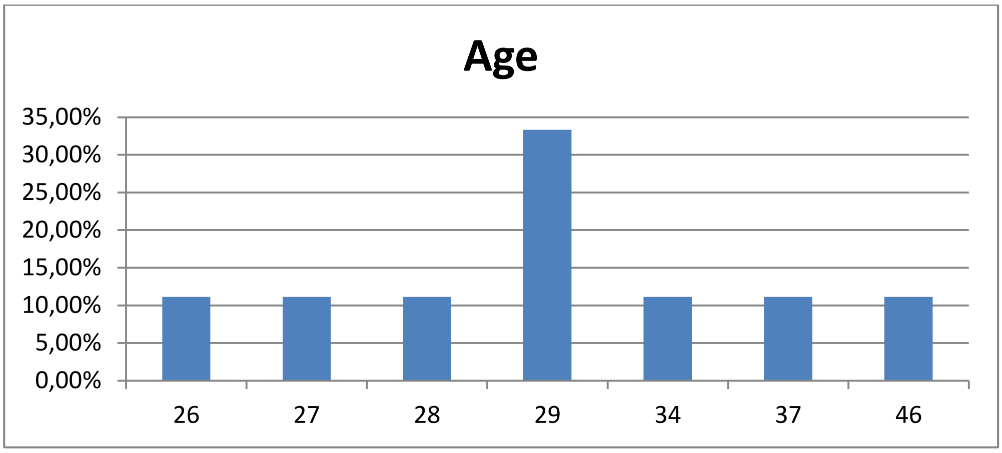
If <u>yes</u>, in which technology?



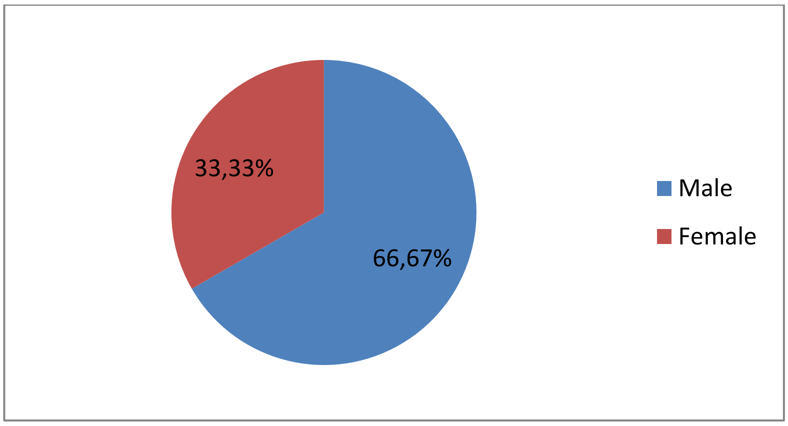
9. **Were you familiar with pen-based technology (Tablets or PDAs) before:**



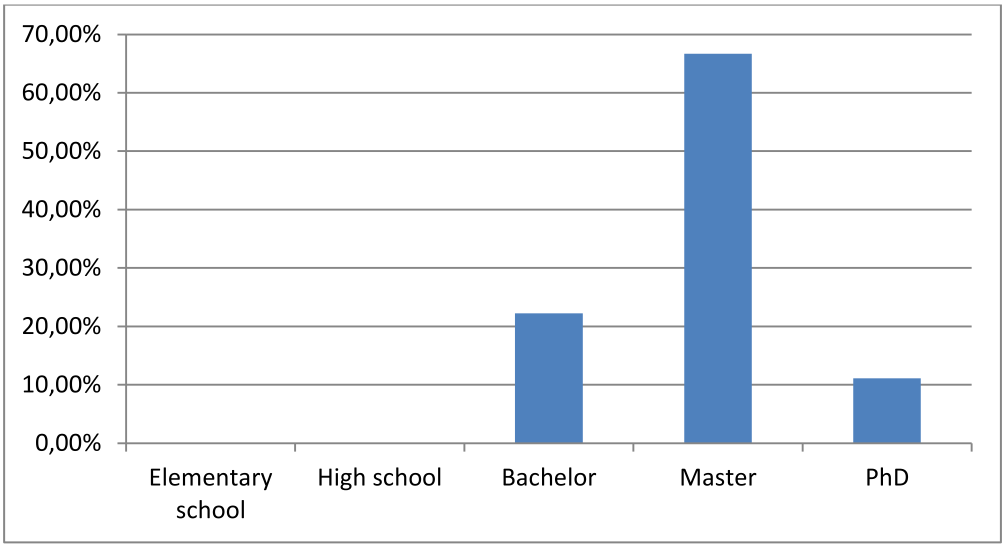
10. Age:



11. Gender:



12. Education:


Elementary school
High school
Bachelor
Master
PhD