An Abstraction for Correspondence Search using Task-Based Controls

Gregor Miller and Sidney Fels

Human Communication Technologies Laboratory
University of British Columbia
Vancouver B.C., Canada, V6T 1Z4

Abstract. The correspondence problem (finding matching regions in images) is a fundamental task in computer vision. While the concept is simple, the complexity of feature detectors and descriptors has increased as they provide more efficient and higher quality correspondences. This complexity is a barrier to developers or system designers who wish to use computer vision correspondence techniques within their applications. We have designed a novel abstraction layer which uses a task-based description (covering the conditions of the problem) to allow a user to communicate their requirements for the correspondence search. This is mainly based on the idea of *variances* which describe how sets of images vary in blur, intensity, angle, etc. Our framework interprets the description and chooses from a set of algorithms those that satisfy the description. Our proof-of-concept implementation demonstrates the link between the description set by the user and the result returned. The abstraction is also at a high enough level to hide implementation and device details, allowing the simple use of hardware acceleration.

1 Introduction

Computer vision has recently seen a rise in production of real-world applications, such as on mobile device's cameras (for stitching and face detection) and real-time pose estimation for games and other gesture-based interfaces. However most of these are developed by experts in computer vision, in dedicated teams designing reliable units of software to perform these tasks. The application of these techniques could be increased if they were easier to use and deploy, however there has not been much work published on how to provide access to these sophisticated techniques to developers; effective use of these methods requires extensive knowledge of how the algorithms work and how their parameters affect the results, expertise beyond the scope of mainstream developers or system designers (termed *users* for the rest of the paper).

The contribution of this paper is a task-based description applied as an abstraction to the correspondence problem in computer vision, to hide the details of specific methods and their configuration. The abstraction may be employed by users to describe the type of vision problem they are trying to solve, and our novel interpreter uses the description to select an appropriate algorithm (with parameters derived from the user's description).

There are numerous benefits of a higher-level abstraction for computer vision: users can focus on their application/system's main task, without focussing on algorithms and parameter tuning; subsequent improvements in techniques for particular problems can be incorporated later with re-implementation; various back-ends can be supported, allowing specific methods to be employed for different requirements e.g. low power for mobile or high speed and accuracy using servers; hardware-acceleration can be used seamlessly, optionally in coordination with CPU; and finally, computer vision expertise can be more readily adopted by researchers in other disciplines, such as HCI and graphics.

If any abstraction is used to access vision methods, hardware and software developers of the underlying mechanisms are free to continually optimise and add new algorithms. This idea has been applied successfully in many other fields, notably graphics with OpenGL, and is the main goal of OpenVL[1] for computer vision. OpenVL is an abstraction framework which hides algorithmic detail and provides developers with access to sophisticated vision methods, such as segmentation[2], human body pose estimation[3] and face detection[4]. The work presented here applies a similar methodology to construct a task-based description of correspondence search at a low enough level to maintain flexibility but high enough for mainstream developers to apply successfully, within the OpenVL framework.

Various technology companies have also recognised the need for a solution to this problem, although most have focussed on hardware acceleration and not higher-level abstraction. A working group at Khronos (a standards body) are developing a low-level hardware abstraction layer called OpenVX to accelerate vision methods[1]; this layer would sit beneath libraries such as OpenCV [5] in order to accelerate existing library calls (much like projects such as OpenVIDIA[2]). Unfortunately OpenCV presents algorithms directly to the developer which requires that they have significant expertise in computer vision - otherwise they are not able to take full advantage of the library. Our proposed abstraction would act as an additional higher-level layer to hide the details of correspondence algorithms and hardware acceleration from developers and allow them to focus on developing applications.

The correspondence problem is a fundamental challenge in computer vision with many robust solutions for a given set of narrow conditions. It is important in many real-world applications such as image stitching, super-resolution, image stabilisation, camera calibration, object detection and 3D reconstruction. To provide flexible approaches to these applications to non-expert users, we must first abstract the complexity to a higher, more intuitive level.

[1] http://www.khronos.org/vision
[2] http://openvidia.sourceforge.net

2 Related Work

Previous attempts on simpler access have generally been in the development of vision or image processing frameworks which present lists of algorithms; the contributions in general have been how the algorithms are presented. Developments in artificial intelligence were used in an attempt to automate the vision pipeline [6–8]. Others provided higher-level access to vision algorithms through object-oriented methodologies for accessibility and reusability reasons, such as in the Image Understanding Environment project (IUE) [9]. Some have attempted to solve specific problems, such as the OpenTL framework [10] which tries to unify efforts on tracking in real-world scenarios In general, the algorithm categorisation and direct access of these approaches requires users to have expert knowledge of vision methods.

Vision applications can be created by using a data flow structure to connect components, using visual programming interfaces such as the Khoros software development environment [11] and Apple's Quartz Composer[3]. These contain components such as colour conversion, feature extraction, spatial filtering, statistics and signal generation, among others. Declarative programming languages have also been used to provide vision functionality in small, usable units, e.g. ShapeLogic[4] or FVision [12], although they are limited in scope due to the difficulty of combining logic systems with computer vision. While these methods provide a simpler method to access and apply methods, there is no abstraction above the algorithmic level, and so users of these frameworks must have a sophisticated knowledge of vision to apply them effectively. As a more graphically intuitive method to overcome the computer vision usability problems, the RADIUS project [13] employed user-manipulated geometric scene models to help guide the choice of algorithm. The level of abstraction provides good usability although power, breadth and flexibility are reduced.

There are many open vision libraries that provide common vision functionality, such as OpenCV [5], Mathworks Vision Toolbox[5] and Gandalf[6]. These libraries often provide utilities such as camera capture or image conversion as well as suites of algorithms, which has previously been shown to lessen the effectiveness on application [14]. All of these software frameworks and libraries provide vision components and algorithms without any context of how and when they should be applied, and so often require expert vision knowledge for effective use. For example, many feature detectors/descriptors are provided by OpenCV but with no indication of under what conditions each works most effectively. Our goal with this paper is to outline a higher-level abstraction for access to these methods through an intuitive task-based interface.

[3] https://developer.apple.com/technologies/mac/graphics-and-animation.html
[4] http://www.shapelogic.org
[5] http://www.mathworks.com/products/computer-vision
[6] http://gandalf-library.sourceforge.net

3 Task-Based Correspondence

The primary aim of our contribution is to provide non-experts in computer vision with intuitive access to sophisticated feature matching techniques. We define a description model for users to specify what the problem is they wish to solve, instead of the current method of defining *how* to solve it. The description is used by our framework to select appropriate feature descriptors, configure their parameters and execute them to return the required result.

3.1 Abstraction through Description

We are using a simple definition of the correspondence problem: among a set of images, find regions whose structure is similar, based on a required strength threshold. The central idea of the abstraction for correspondence is *variances*. Assuming two regions $R_1 \in I_1, R_2 \in I_2$ have identical structure (where I_1, I_2 are images, not necessarily different), the variances describe how the two regions differ. There are many possible variances which generally indicate a difference in appearance (such as intensity) or a distortion of the structure (such as blur or tilt).

Note that our definition covers regions, and not points, despite the result often being a central point in the region. The result from our framework is in the form of regions or central points, as decided by the requirements of the user. Note also that our definition does not define variances as between *images*, but regions. This is due to the fact that there may be matches occurring within a single image, or the set of images may have variations in the property e.g. selective focus. In the future, we will be introducing variances also as requirements, to allow users to specify that matches which have a particular variance (e.g. blur) are not needed.

Each variance is associated with an expected quantity of how much it varies. Currently it is challenging to be highly specific with these, as the descriptors have not been evaluated to this level of detail. Therefore we employ a simple scale from *None* to *High* to allow the user to indicate the level of variance expected.

Each variance is intended to be orthogonal to the others within their description space, to avoid overlap in the description and to encourage completeness. Our eventual goal is to create a unified space for vision descriptions, to apply to all problems, which can be interpreted into algorithms and parameters to provide the user with a solution. The descriptions should be kept as small as possible while maintaining the largest possible coverage of the description space, to help minimise the complexity as the description language is extended.

The other main component of the description from the variances is the *constraints*. These essentially control the execution of the methods and the search space for correspondences. The search space can be defined as *Set* or as *Image*. In *Set*, the search for matches occurs over 'the set' of all regions taken from all images, and each region gets N matches from the required quantity. For *Image*, the search occurs across images only i.e. a region in one image can only be matched to regions in other images, and N matches will be returned per image.

The search operation can be augmented with a qualifier of where to search: *All*, the default, uses all regions or images; *Source* overrides the default search and instructs the framework to only return matches from regions in the same image as the to-be-matched region; *Exact* allows the images to be used to be specified, or a number to be specified to constrain the number of best matches to a subset of images. The final constraints for the correspondence problem are *Quantity* (how many matches to return per region, either within the *Set* or from the *Images*) and *Strength*, the main (simple) threshold to control the reliability of the returned match.

3.2 Algorithm Selection

The interpreter is the second layer in our framework, and receives the description from the user passed through the interface (e.g. API). It is responsible for choosing the appropriate feature matching algorithms based on the described variances, defining the parameters of each algorithm based on the input description, and post-processing constraints which can't be defined as a parameter (e.g. finding the top N matches in K images).

The addition of an algorithm to the framework is accomplished through a 'plug-in' system, defined using an internal interface. Each algorithm must implement this interface; the interpreter then uses it to provide the algorithm with the input images and the full user-defined description. The algorithm returns matches in the interface-defined representation, so that all algorithms return the same type to the user.

The process of adding a new algorithm to the framework is as follows:

– The problem conditions (for correspondence, this is the variances) under which the algorithm is designed to return reliable results, defined using expected tolerance to variances (i.e. how invariant the method is). The set of conditions defines a larges dimensional volume in which algorithms occupy sub-volumes.
– The input must be converted to the format used by the algorithm.
– After execution the results must be checked for compliance with the constraints, and flagged for processing in the interpreter if needed.
– The match results must be converted into the global framework's format for presentation to the user (as regions or points).

It is extremely challenging to define the conditions for the algorithm based on a higher-level description: there may be many ranges under which it works well; it may perform best under certain optimal circumstances, but perform well enough under other conditions; it may not work as well as other algorithms when condition volumes intersect, and so a ranking system may be needed.

There is also the problem of how to define the operating conditions: the creator of the algorithm could define them (or an expert in the area), or we could use a standardised dataset within an evaluation framework to generate them. We could also use machine learning techniques to automatically determine the conditions based on the description. In our framework, we have defined the conditions (as ‘experts’) using evaluations of feature descriptors from the literature; our resources for the condition definitions are presented in Section 4.2.

The correspondence interpreter in our framework will select all algorithms which match the user-defined description and constraints, to provide as many matches as possible. The interface allows the user to specify the efficiency required, which can tune the number of methods chosen down. This is also based on the level of parallel processing available on the host machine and allowed by the user.

4 Evaluation of the Task Description

Our framework is implemented in C++, with three separate layers to allow for simple replacement of any one layer. The description layer sits on top and acts as the interface between the user and the lower layers. It is through this that the user provides a description of the correspondence problem. The description is passed to the second layer, the interpreter, a thin layer which chooses which algorithms are appropriate given the description, and then configures each algorithm's parameters. The interpreter can also define any necessary pre- or post-processing operations (such as noise removal or image scaling). The lowest layer of the three is where the algorithms sit. For this work we have defined a condition set for five feature descriptor algorithms for matching image regions. Each algorithm is registered with the interpretation layer along with its optimal operating problem conditions (defined in Table 1). The conditions are defined using the variance components of the description. Given a lack of available specific evidence for the performance of each feature descriptor, their capabilities are approximated using a four-scale *None*, *Low*, *Medium* and *High*.

4.1 Parameters

The correspondence description provides the following set of variances: position, size, orientation, blur, intensity and tilt distortion. All are measured on the scale mentioned previously. Unfortunately the use of such a scale is not consistent across all variances, and so each must be documented individually.

- Position: Indicates the expected level of variation in the region's position in image coordinates. This can encode problem conditions such as the quantity of expected overlap among images. This is usually used purely to reduce the search space, and is not used in algorithm selection.
- Size: *None* indicates identical size, *High* means changes of 100% or more. Sizes are measured in units of image width (based on source region).
- Orientation: 2D rotation up to 180° (either direction)
- Blur: It is challenging to determine a scale of blur which is easily understood, and to avoid measurements based on pixels (since they are not directly relevant to the vision/matching problem). In future we will need a more sophisticated description for blur, taking into account multiple blur kernels

and parameters. For now, since this type of variation is not accounted for within feature descriptors, a simpler description will do. The scale we use starts from zero blur (*None*, i.e. circle of confusion has a radius lower than the current sampling density); *Low* blur is defined as a gaussian blur with $\sigma = 1.0$ and a kernel size of 0.5% image width; *High* blur uses the same definition but with a kernel size of 2% image width.

– Intensity: This is also challenging - we treat it as an absolute measurement, so *Low* is approximately 10% difference, *Medium* is 30% and *High* is above 50%.
– Tilt Distortion: This represents the central viewing angle difference between views (and could also account partly for lens distortions). The variance is measured on a scale from $0°$ to $45°$ (*None* to *High*).

Table 1. The mapping from description to algorithm within our interpreter, using the variances between images. The methods satisfy any permutation of their conditions.

Algorithm	Size	Orientation	Blur	Intensity	Tilt Distortion
SIFT [15]	High	High	High	Medium	Medium
SURF [16]	Medium	Low	High	High	Low
ORB [17]	Low	High	Low	Low	None
MSER [18]	None	None	None	Low	Medium
FREAK [19]	Medium	None	High	High	Medium

4.2 Feature Matching Algorithms

The algorithms chosen to sit in the lowest layer of our framework are shown in Table 1. Each algorithm registers itself with the interpreter using the variance capabilities indicated in the table. The set of mappings from description to algorithm (defined in the Table) are quite expansive, and could be more specific with a more direct evaluation of capabilities. In the cases where a user supplies a description which is not represented by our interpreter's algorithms, we can perform a 'best effort' and provide the closest match, or provide an informative error.

The mapping from variances to algorithms were inferred using multiple sources: size, orientation, blur and intensity for SIFT, SURF and ORB[7]; ORB comparison to SIFT and SURF for orientation [17]; size, orientation, intensity and tilt distortion for SIFT and SURF [20]; orientation, blur, intensity and affine transforms for SIFT and SURF [21]; size and tilt distortion for SIFT and MSER [22];

[7] `http://computer-vision-talks.com/articles/2011-08-19-feature-descriptor-comparison-report/`

orientation, blur and tilt distortion for SIFT and MSER [23]; size, blur, intensity and tilt distortion for MSER [24]; and finally, size, blur, intensity and tilt distortion for SIFT, SURF and FREAK [19].

The conditions chosen for each algorithm are examples, and not meant to be definitive. Choosing the conditions is a difficult problem, and perhaps the best solution is an evaluation of each algorithm under all known conditions on a ground-truth dataset. We can then assign the conditions from this evaluation or use a learning-based approach to bypass the literal conditions completely - although this would require a differently designed interpreter.

By default, the interpreter executes as many of the algorithms as it can, based on how well they match the user's description. This will provide the highest number of matches, but is also slower or uses more resources. The user can override this behaviour by prioritising efficiency over quantity. The user may also choose to switch implementations from the CPU-based to the GPU-based algorithms (which use SIFT and SURF only): this is seamless and the actual algorithms included are not exposed to the user. This allows the user to take advantage of the parallel hardware and provide a faster computation. The interface does not change for multiple implementations, ensuring that the code written for one device type will work on any other device type.

The strength constraint does not use the variances, even though some algorithms may have a higher strength than others. Instead it is used to modify the parameters of the algorithms, such as the number of nearest neighbours to require for a match or the minimum distance between feature vectors.

4.3 Results

In Figures 1 and 2 we illustrate four different descriptions (using variances) which describe their pair of images. For each example three results are provided, at High, Medium and Low Strengths. Figure 3 shows results for the four previous examples but without providing any description. The images used for this demonstration were taken from a feature detection evaluation site[8] and a feature evaluation paper [21].

The first example in Figure 1 is of two images containing a lot of detail (church exterior). The variances defined lead to SIFT being chosen by the interpreter to find the correspondences. The results are generally good at all strengths (likely due to the high detail), with some errors apparent at Low strength. The second example in Figure 1 has images with a significant level of structure, with the two images showing the same building at two focal lengths. The description supplied here invoked SIFT, SURF and FREAK and provided excellent correspondences, with some errors at Low strength.

The first example in Figure 2(a) contains two images with a large size variation and low-to-medium in all other variances. The interpreter executed SIFT to accommodate the size and tilt variation. At a High strength the results are sparse but accurate, and as the strength is lowered more matches are provided

[8] `http://lear.inrialpes.fr/people/mikolajczyk/Database/det_eval.html`

but many noticeable errors occur. This is due to the more extreme difference in images. The second example illustrates a single image matched with a blurred version of itself. The description translated to the execution of SIFT, SURF and FREAK, providing a few correspondences. The results seem to be as good when the strength is lowered, which may indicate we should provide more control over these cases to the user (to allow for a variation of strength or method when blur is a significant variation).

To see how well the framework performs in the absence of any information, we tested on all previous examples and gave the interpreter a description with all variances set to None. All methods are executed in this case. The results are generally acceptable for High Strengths, with only a few mistakes on the more extreme examples. There are significantly more errors in the medium case, along with a higher number of returned correspondences. The time taken was much longer in all cases due to executing all algorithms.

5 Conclusion

We have presented a novel abstraction for the correspondence problem which provides users with the ability to target specific conditions for the correspondence problem. This allows them to find the most efficient and accurate algorithm(s) without requiring a high level of expertise in feature detectors and descriptors. We demonstrated the results produced when providing our framework with a description of variances, and illustrated how the matches were of higher quality than not providing any information.

We intend to expand the variances, possibly to multiple levels of description satisfaction: for example, SURF can do well enough on tilt distortion but might need to be controlled via strength by the user; alternatively ORB is generally faster than the others, but not as accurate in certain conditions - this could be an override control for the user.

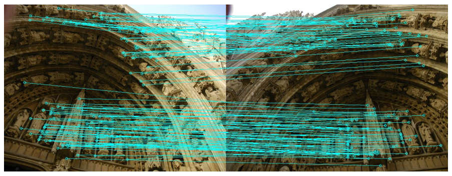
Acknowledgements. We would like to gratefully acknowledge the support of the *Natural Sciences and Engineering Research Council of Canada (NSERC)* and the *Canadian Graphics, Animation and New Media Network of Centres of Excellence (GRAND NCE)*.

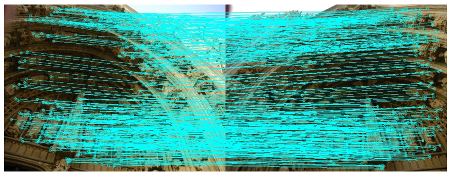
References

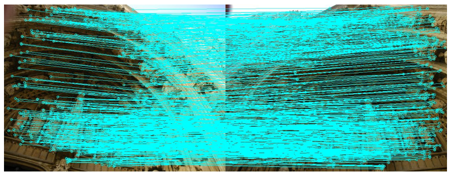
1. Miller, G., Fels, S.: OpenVL: A task-based abstraction for developer-friendly computer vision. In: Proceedings of the 13th IEEE Workshop on the Applications of Computer Vision (WACV). WVM'13, IEEE (2013) 288–295
2. Miller, G., Jang, D., Fels, S.: Developer-friendly segmentation using OpenVL, a high-level task-based abstraction. In: Proceedings of the 1st IEEE Workshop on User-Centred Computer Vision (UCCV). WVM'13, New York City, New York, U.S.A., IEEE (2013) 31–36

Example (a) - Variances:
Position(Medium); Size(None); Orientation(Low);
Blur(None); Intensity(Low); Tilt Distortion(Medium).

Original images

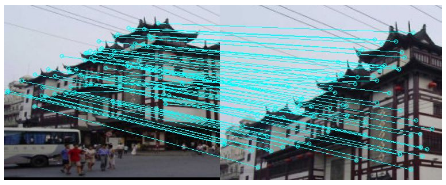

Strength: High

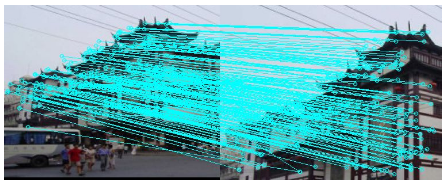

Strength: Medium


Strength: Low

Example (b) - Variances:
Position(Medium); Size(Medium); Orientation(None);
Blur(None); Intensity(None); Tilt Distortion(Low).

Original images


Strength: High


Strength: Medium

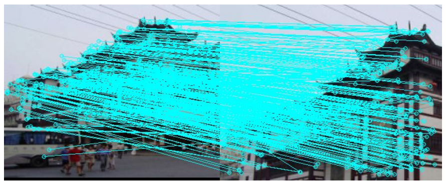
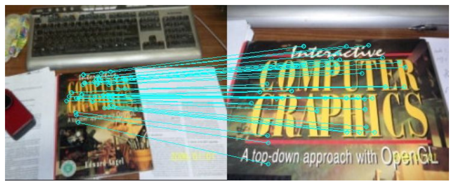
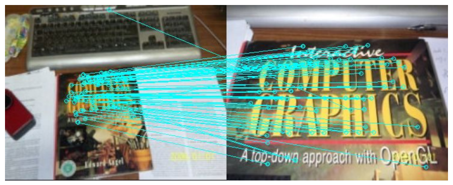

Strength: Low

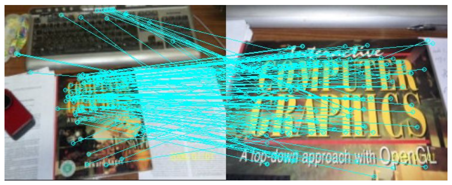
Fig. 1. The original images for each example are shown along with the results at three strength levels, with the stated variances above each set of images. Example (a) illustrates when almost all variances are set, and the returned matches at all three strengths with good results. Example (b) also demonstrate good results, but with noticeable mismatches at lower strengths.

Example (a) - Variances:
Position(Medium); Size(High); Orientation(Low);
Blur(Low); Intensity(Low); Tilt Distortion(Low).

Original images

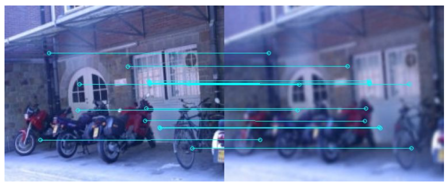

Strength: High

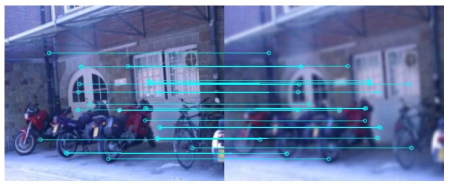

Strength: Medium


Strength: Low

Example (b) - Variances:
Position(None); Size(None); Orientation(None);
Blur(High); Intensity(None); Tilt Distortion(None).

Original images


Strength: High


Strength: Medium

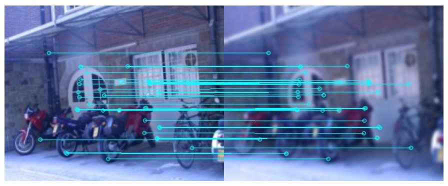
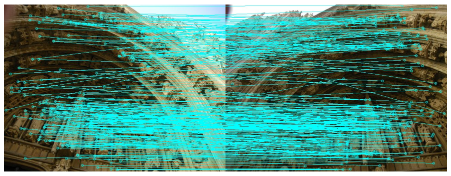
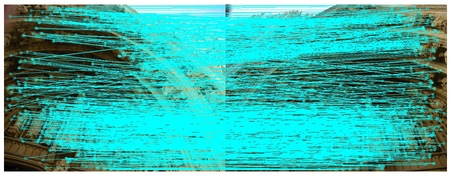
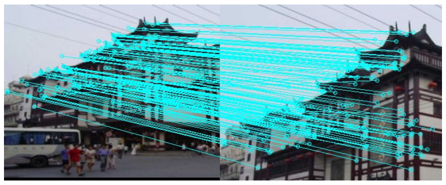

Strength: Low

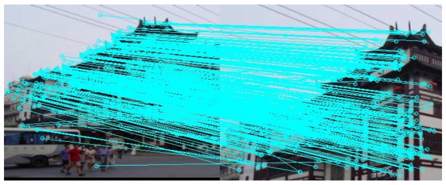
Fig. 2. The original images for each example are shown along with the results at three strength levels, with the stated variances above each set of images. It is noticeable in Example (a) that in more extreme cases like this, the strength can have a noticeable effect on quality of returned match. This is also seen in Figure 1(a), indicating that perhaps strength plays a more important role in size differentials. In Example (b), the level of blur is so high that it inhibits feature extraction, even at the lowest strength (although the results obtained at a low strength are quite accurate). This may indicate that we should provide a finer level of control over the strength for the user.

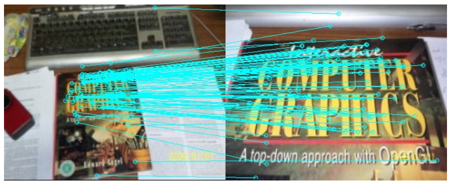
Examples with all variances set to *None*

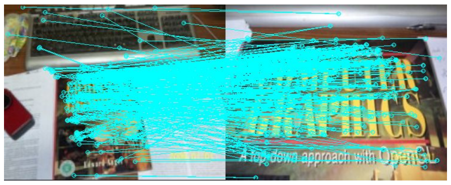

Strength: High

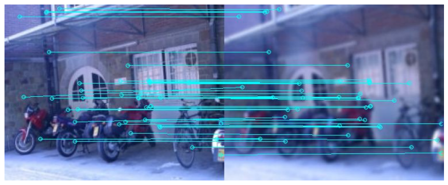

Strength: Medium


Strength: High


Strength: Medium


Strength: High


Strength: Medium


Strength: High

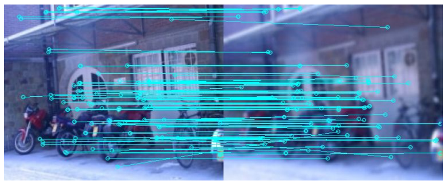

Strength: Medium

Fig. 3. The examples here demonstrate the result when you set all variances to *None*, and so all feature algorithms are executed. At a high strength, this provides reasonable results, with only the occasional error (such as in the book example). At medium strength it is clear that there are many more errors (and many more matches) than in the case where the specific problem was described. In all cases, this took significantly longer to compute, since all algorithms were used. Given the higher quality of correspondence and the decrease in time taken, this demonstrates the improvement which can be made with knowledge of the problem conditions in addition to the reduced level of expertise required to use our framework.

3. Oleinikov, G., Miller, G., Little, J.J., Fels, S.: Task-based control of articulated human pose detection for openvl. In: Proceedings of the 14th IEEE Winter Conference on Applications of Computer Vision. WACV'14, New York City, U.S.A., IEEE (2014) 682–689
4. Jang, D., Miller, G., Fels, S., Oldridge, S.: User oriented language model for face detection. In: Proceedings of the 1st Workshop on Person-Oriented Vision (POV). WVM'11, New York City, New York, U.S.A., IEEE (2011) 21–26
5. Bradski, G., Kaehler, A.: Learning OpenCV: Computer Vision with the OpenCV Library. 1st edn. O'Reilly Media, Inc. (2008)
6. Matsuyama, T., Hwang, V.: SIGMA: a framework for image understanding integration of bottom-up and top-down analyses. In: Proceedings of the 9th international joint conference on Artificial intelligence. Volume 2., Morgan Kaufmann Publishers Inc. (1985) 908–915
7. Kohl, C., Mundy, J.: The development of the image understanding environment. In: Proceedings of the Conference on Computer Vision and Pattern Recognition. CVPR'94, Los Alamitos, California, U.S.A., IEEE Computer Society Press (1994) 443–447
8. Clouard, R., Elmoataz, A., Porquet, C., Revenu, M.: Borg: A knowledge-based system for automatic generation of image processing programs. Transactions on Pattern Analysis and Machine Intelligence **21** (1999) 128–144
9. Mundy, J.: The image understanding environment program. IEEE Expert: Intelligent Systems and Their Applications **10** (1995) 64–73
10. Panin, G.: Model-based Visual Tracking: the OpenTL Framework. 1st edn. John Wiley and Sons (2011)
11. Konstantinides, K., Rasure, J.R.: The Khoros software development environment for image and signal processing. IEEE Trans. on Image Processing **3** (1994) 243–252
12. Peterson, J., Hudak, P., Reid, A., Hager, G.D.: Fvision: A declarative language for visual tracking. In: Proceedings of the Third International Symposium on Practical Aspects of Declarative Languages. PADL '01, London, UK, Springer-Verlag (2001) 304–321
13. Firschein, O., Strat, T.M.: RADIUS: Image Understanding For Imagery Intelligence. 1st edn. Morgan Kaufmann (1997)
14. Makarenko, A., Brooks, A., Kaupp, T.: On the benefits of making robotic software frameworks thin. In: Proceedings of the Workshop on Measures and Procedures for the Evaluation of Robot Architectures and Middleware. IROS'07, New York City, New York, U.S.A., IEEE (2007) Invited Presentation.
15. Lowe, D.G.: Distinctive image features from scale-invariant keypoints. International Journal of Computer Vision **60** (2004) 91
16. Bay, H., Ess, A., Tuytelaars, T., Van Gool, L.: Speeded-up robust features (surf). Comput. Vis. Image Underst. **110** (2008) 346–359
17. Rublee, E., Rabaud, V., Konolige, K., Bradski, G.: Orb: An efficient alternative to sift or surf. In: Proceedings of the 2011 International Conference on Computer Vision. ICCV '11, Washington, DC, USA, IEEE Computer Society (2011) 2564–2571
18. Forssén, P.E., Lowe, D.: Shape descriptors for maximally stable extremal regions. In: IEEE International Conference on Computer Vision. Volume CFP07198-CDR., Rio de Janeiro, Brazil, IEEE Computer Society (2007)
19. Ortiz, R.: Freak: Fast retina keypoint. In: Proceedings of the 2012 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). CVPR '12, Washington, DC, USA, IEEE Computer Society (2012) 510–517

20. Bauer, J., Sünderhauf, N., Protzel, P.: Comparing several implementations of two recently published feature detectors. Intelligent Autonomous Vehicles **6** (2007) 143–148
21. Juan, L., Gwon, O.: A comparison of sift, pca-sift and surf. International Journal of Image Processing **3** (2009) 143–152
22. Yu, G., Morel, J.M.: A fully affine invariant image comparison method. In: Acoustics, Speech and Signal Processing, 2009. ICASSP 2009. IEEE International Conference on. (2009) 1597–1600
23. Morel, J.M., Yu, G.: Asift: A new framework for fully affine invariant image comparison. SIAM J. Img. Sci. **2** (2009) 438–469
24. Mikolajczyk, K., Tuytelaars, T., Schmid, C., Zisserman, A., Matas, J., Schaffalitzky, F., Kadir, T., Gool, L.V.: A comparison of affine region detectors. Int. J. Comput. Vision **65** (2005) 43–72