VISION PRIMITIVES FOR THE INTUITIVE PROGRAMMING OF FLEXIBLE MICROASSEMBLY CELLS

Yuri L. De Meneses[1], Cyrille Lebossé[2], Cédric Wulliens[1], J. Jacot[1], S. Koelemeijer[1]

[1]Laboratoire de Production Microtechnique (LPM-IPR)
Ecole Polytechnique Fédérale de Lausanne (EPFL)
1015 Lausanne, Switzerland
yuri@ieee.org

[2]Ecole Nationale Supérieur de Physique de Strasbourg
Bd. Sébastien Brant
F-67412 Illkirch, France
Cyrille.lebosse@voila.fr

ABSTRACT

This paper describes the geometric image-analysis library being developed by LPM to guide a flexible microassembly cell in vision-guided assembly tasks. The library is based on computer vision primitives that have a mechanical interpretation (area, center of gravity), affording the intuitive programming of the assembly cell. For more complex geometric relationships, *templates* are provided. These are described, evaluated and shown in example assembly tasks.

INTRODUCTION

In spite of the increasing degree of monolithic integration in microsystem technologies (MEMS, MOEMS, CMOS, etc), a microassembly step, whereby individual components are positioned, attached, and finally packaged, is usually required. This step is necessary due to manufacturing costs that may preclude a full integration, because of incompatible processes or substrates or simply due to geometric constraints, since most processes are planar [1].

Currently, most microsystems that fall in the categories above are assembled in small batches, either because they are in early-development stages or because market size does not justify mass production [2]. Manual assembly is not possible for anything but the smallest batches, because humans cannot maintain the required precision and yield over time. Therefore microsystem manufacturers need flexible microassembly workstations capable of accommodating to frequent product changes. To achieve this with low setup costs, flexible workstations should be easily reprogrammed.

The Laboratoire de Production Microtechnique (LPM) of the Ecole Polytechnique Fédérale de Lausanne (EPFL), Switzerland, is actively developing a flexible microassembly workstation that can be easily, "intuitively", programmed (configured) by somebody who does not need to be a programming specialist, allowing lab technicians or operators to set the production up. The workstation, described in [3], is based on a high-precision robot (0.5 or 5 µm resolution) equipped with a computer-vision system and a graphic user interface for setup and configuration. The operator can perform a step-by-step assembly by combining robot actions and image-analysis primitives that identify and select the different assembly parts. The sequence of operations is stored in a script and afterwards the workstation can automatically go through it, in what is called automatic mode.

A key element in this "intuitive programming" paradigm are the image-analysis primitives which are available in the assembly station. This paper describes the geometry-based, image analysis library developed at the LPM, which provides these primitives. It then discusses their performance (precision, speed) and shows how they can be combined to solve an assembly task.

THE GEOMETRIC IMAGE-ANALYSIS LIBRARY

The main requirements of the geometric image-analysis library libgeom, as with most computer-vision libraries, is that it should provide primitives that are invariant to perturbations found in the vision system. In the case of robotic-assisted assembly, where mostly rigid parts are manipulated, these perturbations are translation, rotation and scaling (change in zoom). Furthermore, other perturbations such as blur, illumination changes or quantization noise may appear, and they can be modeled as additive noise.

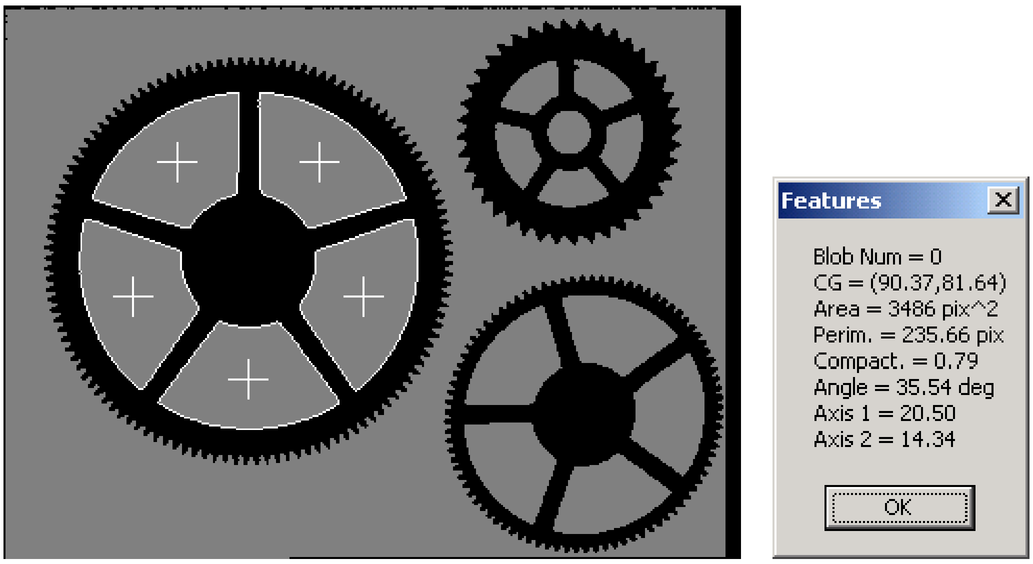
An additional requirement stemming from the "intuitive programming" philosophy is that the vision primitives should be easily understandable by a non-computer-vision specialist. Since operators will most likely have some knowledge of mechanics, ideally the primitives should have an easy physical or mechanical interpretation.

To meet the above requirements libgeom exploits geometric features of the blobs present in the images. Blobs, mathematically known as connected components, are connected sets of pixels with the same properties, most typically intensity or color. Under the right conditions, blobs can correspond to the actual objects or parts seen in the image. But it should be borne in mind that virtual objects such as shadows may also appear as blobs. In the case of libgeom, blobs are defined by the intensity on a binary image. To facilitate the generation of such an image, libgeom provides an algorithm for optimal image-binarization according to geometric criteria.

For each blob in the image, the operator can obtain a set of features, shown in table 1 together with their degree of invariance. Because these features are of integral nature, being based on a set of pixels, they are also robust in the face of additive noise that might affect the position of the pixels.

Blob feature	**Invariance**		
	Translation	Rotation	Scale
Area	X	X	
Perimeter	X	X	
Center of Gravity		X	X
Compactness ("roundness")	X	X	X
Main axis of inertia (orientation)	X		X
Eccentricity	X	X	X
Minimum enclosing circle			
Bounding Box (rectangle)			

Table 1: Blob features available in libgeom and their invariance.


Features

Blob Num = 0
CG = (90.37,81.64)
Area = 3486 pix^2
Perim. = 235.66 pix
Compact. = 0.79
Angle = 35.54 deg
Axis 1 = 20.50
Axis 2 = 14.34

Figure 1: Example of blobs detected during a watch-assembly task. Blobs do not necessarily correspond to actual objects (gears); they might be only iconic objects (the holes in the parts). Geometric criteria (size, roundness,...) help the operator to select only pertinent blobs.

Additionally, a series of SelectBy...() functions allow the user to select certain blobs based on their features. Thus, for example, if the operator needs to pick a circular part, say a gear, he can select the blob with the highest compactness ("roundness") and then determine its center of gravity. Or else, if the operator knows the approximate size of a part, taken from a previous measurement, SelectByArea(target) will identify the blobs or parts that most closely matches it. These selection operations can be chained so that at the end the operator obtains the one desired part.

TEMPLATES

The centers of gravity, although robust, do not provide sufficient positioning information for all assembly tasks. The operator might need to place some glue somewhere else than in the center of part, say at one of its ends. In other cases a new part should be placed at a precise position that does not correspond to any blob, and this position should be given to the robot. For these situations libgeom provides the operator with geometric models or templates that determine a target point in the image relative to a set of points, called reference points, which can stem from other blobs or geometric relationships.

The operator creates a template by 'teaching' the system on a reference part or by manual operation (i.e. by mouse-clicking on the desired position.) To this end the operator provides a target point and a list of reference points. This list is typically built with the centers of gravity of other blobs or parts in the image, carefully sieved through a series of selection operations. The template then stores the mathematical relationship between the target point and the ordered set of points [4]. This relationship renders templates invariant to translation, rotation and scaling. Moreover, because templates use several points, they are robust against noise.

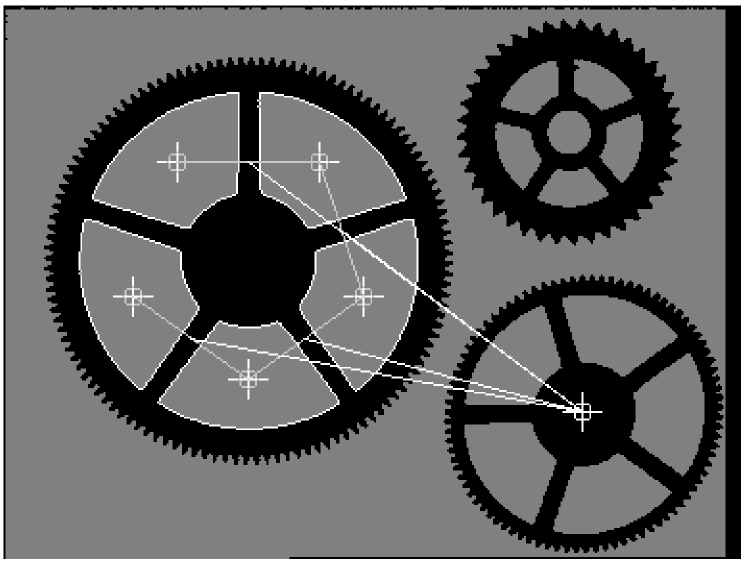


Figure 2: Example of a template defining the position of a gear (target point) relative to the blobs in a neighboring gear (reference points).

An interesting feature of templates for "intuitive programming" is that, based on the stability of the mathematical relationship of the points, the system can give the operator an immediate feedback on the a priori attainable precision when using that particular template (before it is even applied!), since some configurations (e.g. collinear points) are more error-prone than others. When the operator is satisfied with the expected quality of the template, he can store it for later.

To apply a template, in programming or automatic mode, the operator or the script provide a set of points and indicate the template to be applied. The system then calculates the resulting position and gives a feedback on the actual measurement error due to perturbations in template points.

Implementation

Templates define a linear relationship between reference points and a target point. This relationship can be implemented in several ways. The implementation currently available in `libgeom` is based on N-1 lines (where N is the number of reference points) that stem from the midpoint of the segments connecting the reference points. The target point is determined by the intersection of these lines. For $N > 3$ this is an overdetermined system that is solved by minimizing the least-square error.

Some templates are more robust than others, depending on the configuration of reference-points. For instance, almost-collinear reference points will produce almost-parallel lines, with very unstable solutions. The stability of the least-squares solution can be evaluated a priori, that is, before actually solving the system, through the condition number of the associated equation matrix [5]. The condition number provides a measure of the template quality. The worst-case error is proportional to the square of the condition number. So a good rule of thumb is to keep this number smaller than 10.

Templates also provide an a posteriori measure of quality. The residual mean square error of the least-square algorithm gives the precision or confidence with which the solution, the target point, has been determined.

Performance evaluation

The performance of the templates has been evaluated by studying the impact of noise in the desired solution. To this end, the position of each reference points is modified with gaussian noise with 0 mean and standard deviation σ. To provide a scale-invariant measure, the standard deviation is defined as a percentage of the norm of the main inertial axis of the reference pattern.

The error introduced by noise in the determination of the target point will be higher for targets that are farther away from the reference points. To provide a measure of performance that can be compared across all targets, the resulting positional error is decomposed into translation error and rotation error. Thus, for a target point at a distance R from the center of gravity of the reference points, the total error can be estimated as $E_{trans} + R \cdot E_{rot}$. Translation error is due to the resultant translation of the reference points and it has, as expected, zero mean and standard deviation equal to $\sigma / \sqrt{N}$ where N is the number of reference points in the template. Rotational error is due to the difference in angle with respect to reference pattern. It also has zero mean and its variance is proportional to the input noise variance, except for seriously unstable templates (high condition number).

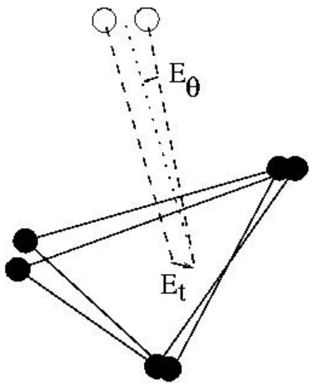


Figure 3 Translation and rotation error in determining the target position (white ball) from a triangular reference (black balls)

Figure 4 shows rotational and translation errors averaged over randomly-chosen patterns.

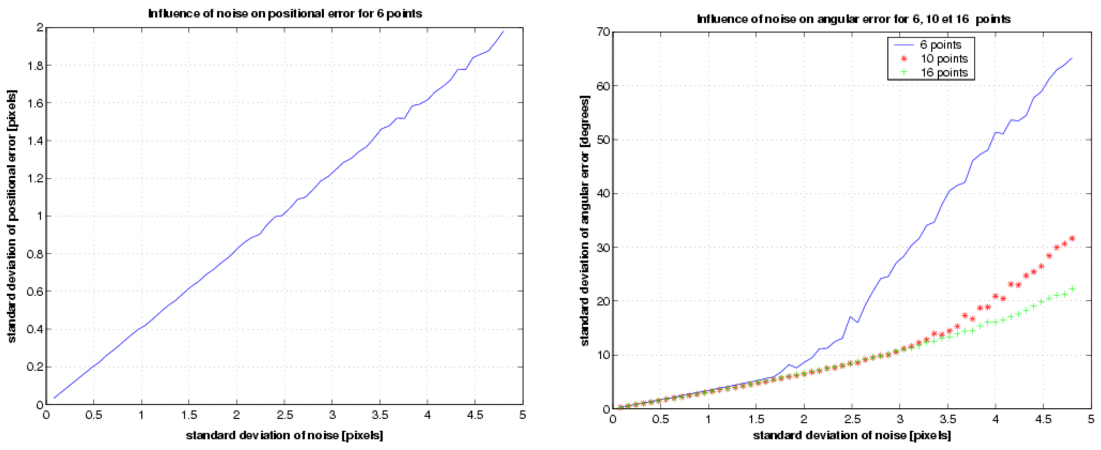
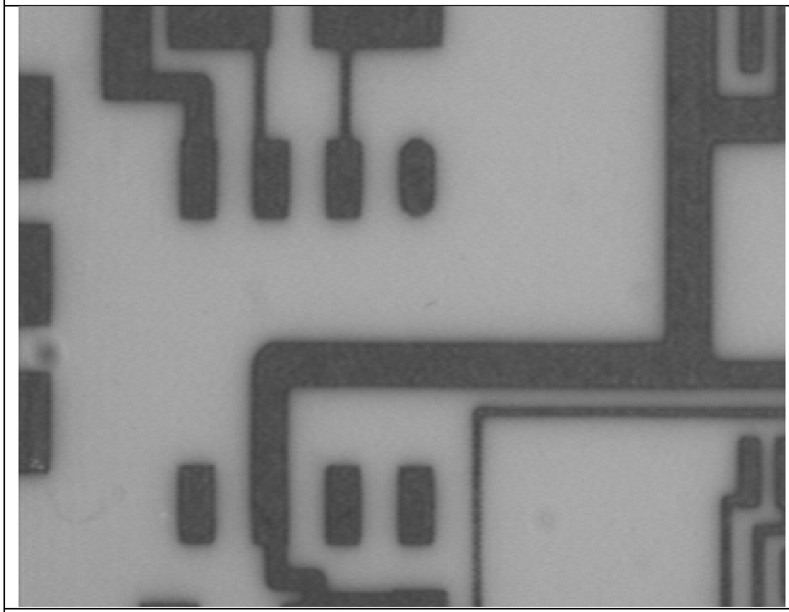
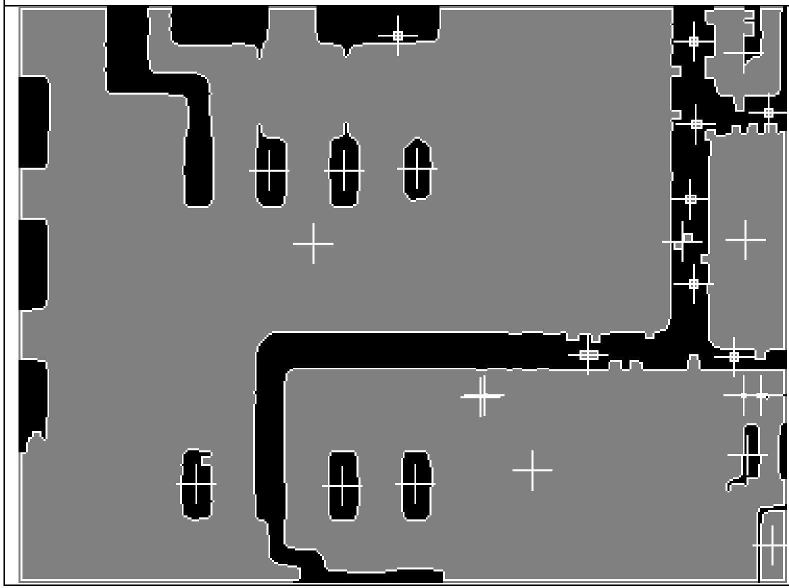


Figure 4: Translation and rotation errors averaged over difference configurations of reference points. The slope provides a measure of the robustness of the implemented templates.

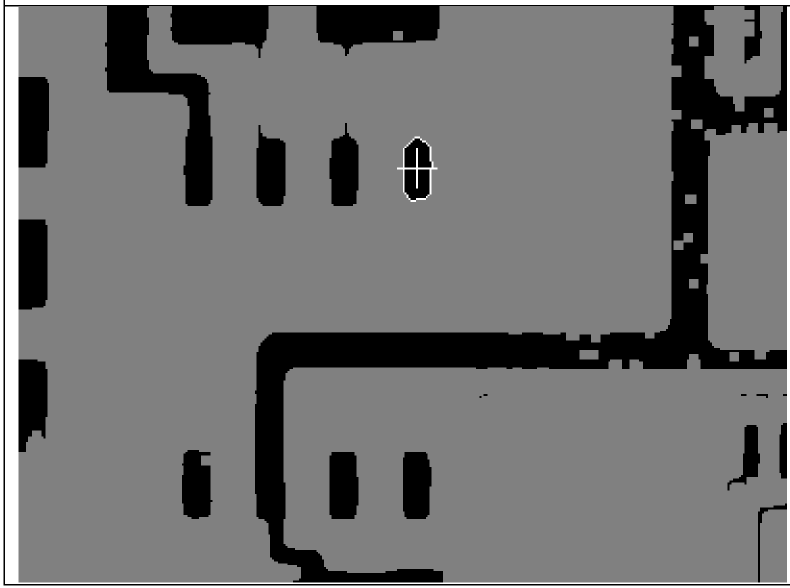
USE CASES IN ASSEMBLY TASKS

Task 1: To place and solder an SMD component on a PCB, the position of pin no. 1 has to be verified. The pad corresponding to pin 1 can be recognized by its rounder shape.

Result	Command
	
	Original image

	`Threshold(90)`
	`Dilate`
	`Dilate`
	`FindBlobs`

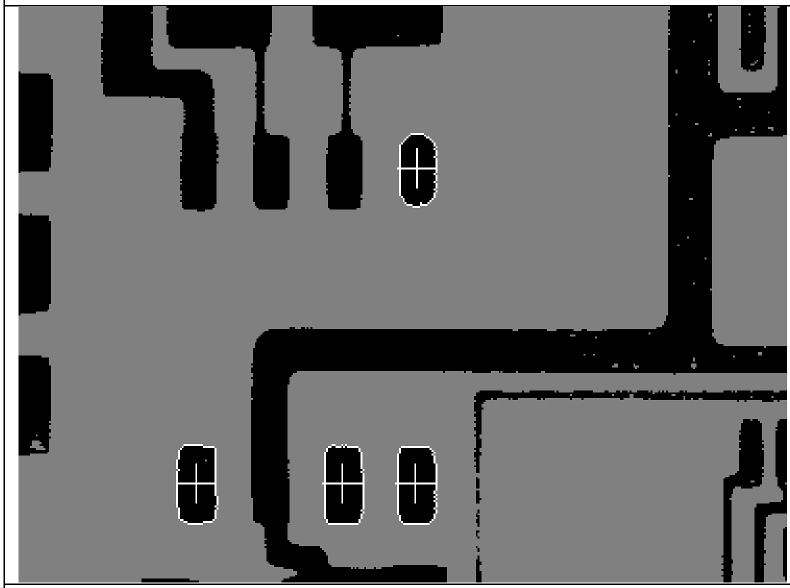
	Set to black all pixels below 90. Apply morphological dilation (erosion of back objects) to eliminate copper wires and isolate the pads.

	`SelectByCompact(0.8,1.0)`
	`SortByArea( );`
	`SelectFirstN(0);`
	`P=GetCenter(0);`

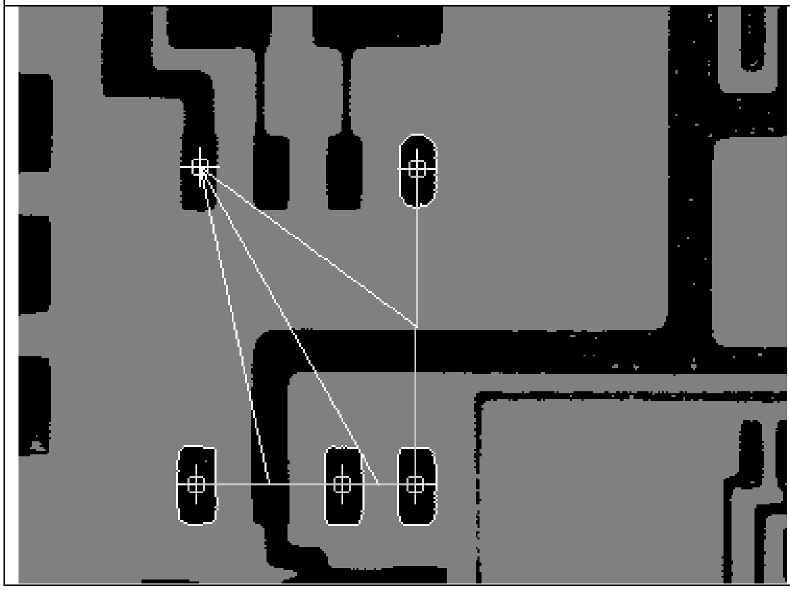
	Select only objects with roundness above 0.8. This leaves some small blobs in. Sort in descending order of surface. Compute the center of the biggest blob. The position of pin 1 is thus recovered.

Task 2: Suppose we want to depose some glue or soldering paste on pad no 4, which is hidden by a thick wire (Vcc). To determine its expected position we can use a template relating the hidden pad with the isolated --and hence easier to determine-- pads.

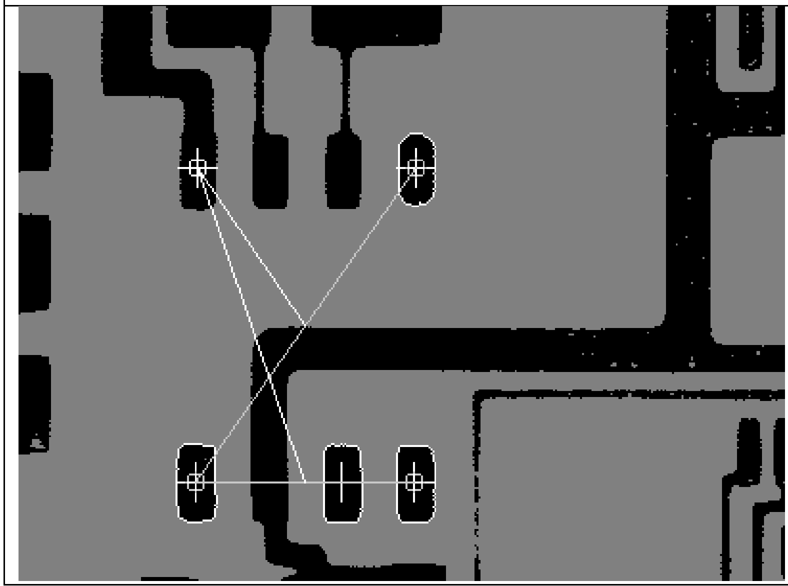
Result	**Command**

	`Threshold(90)`
	`FindBlobs`
	`SelectByCompact(0.7,1.0)`
	`SelectByArea(500,1500)`

	Binarize with a threshold of 90, select blobs with a roundness higher than 0.7 and an area between 500 and 1500.

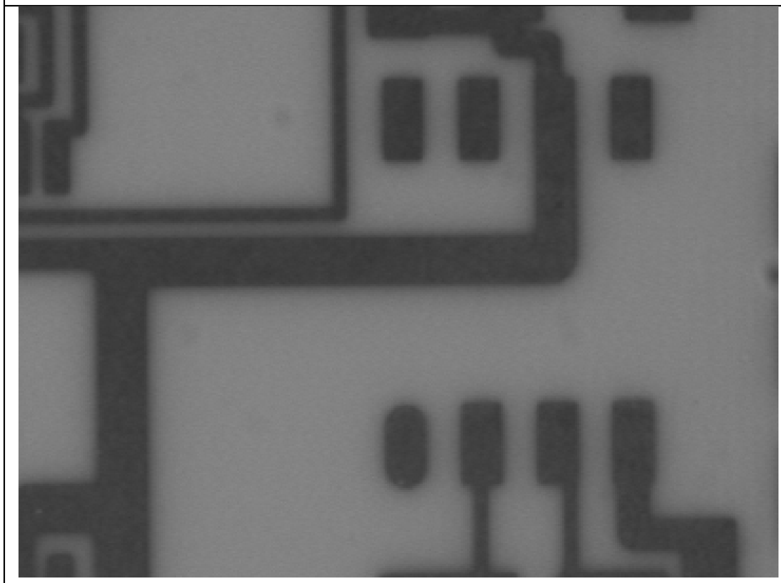
	`SetReference(4)`
	`P = GetUserInput()`
	`SetTarget(p)`
	`C=ComputeModel`

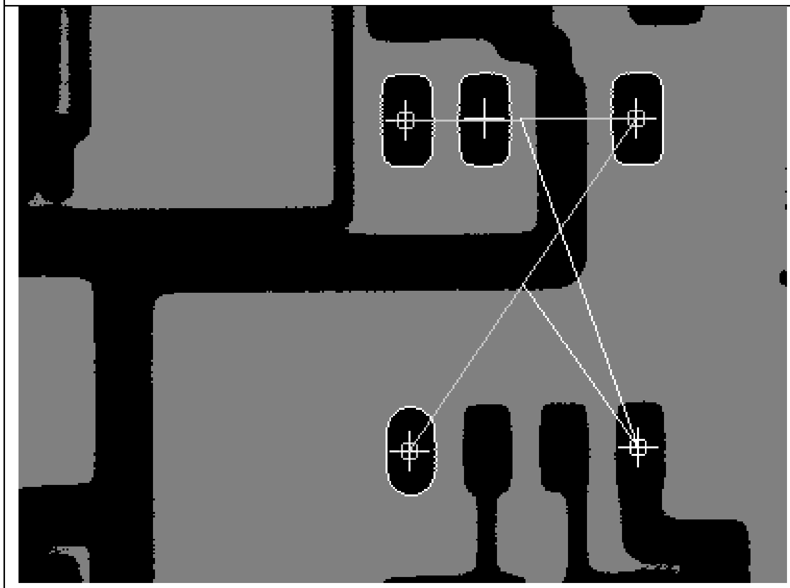
	Add all 4 blobs as reference points. Set the target point to the position clicked by the user (91,84). Compute template or model. The template quality is given in C = 15.99 Too high. The template is abandoned.

	ShuffleReference
	P = GetUserInput()
	SetTarget(p)
	C=ComputeModel
	SaveModel("SO8-7_33.mod")

	Reorder the reference points. Set the target point to the position clicked by the user (91,84). Compute template or model. Template quality is now C = 7.33 Save template to a file for later use.

Task 3: Application of the saved template on another circuit board (e.g. in automatic mode)

Result	Command
	
	Load a new image

	Threshold(90)
	FindBlobs
	SelectByCompactness(0.7,1.0)
	SelectByArea(500,1500)
	SetReference(4)
	LoadModel("SO8-7_33.mod")
	P,Q,E = GetTarget(model)

	Binarize and detect blobs. Select blobs with roundness above 0.7 Save the 4 blobs as reference points. Load the saved model and compute the corresponding target. Quality (Q) is *7.27* and error (E) *0.00*

CONCLUSION

The vision primitives of the geometric image-analysis library (libgeom) being developed at the Laboratoire de Production Microtechnique (LPM) provide functionalities that can be exploited to guide a flexible microassembly cell in vision-guided tasks. To allow non-specialist operators to program the microassembly cell the vision primitives are based on geometric properties of image objects or blobs, such as centers of gravity, area, axis of inertia, etc. These properties have a clear mechanical interpretation.

To build more complicated operations, templates have been introduced, whereby the geometric relationship between a target point and a set of points (reference points) can be defined, learned and stored for reuse. Other functionalities, not discussed in this paper, are the automatic segmentation (binarization) based on the optimization of geometric criteria such as roundness or orientation.

Future work

To implement more complicated tasks (such as recognition of a given part) it is necessary to build higher-level operations by combining vision primitives. This can be accomplished by a scripting language, allowing the creation of libraries of routines for common tasks (e.g. gear insertion, chip positioning, etc). This will add further flexibility to the assembly cell, since it can be updated by installing additional routines.

Other developments concern the automated construction of templates and the detection of reference points by their relative positions (in fact, an extended template.).

REFERENCES

[1] K.B. Yesin, G. Yang, J. Gaines, and B.J. Nelson, "Microassembly of silicon-based hybrid microsystems," in Proceedings of International Precision Assembly Seminar (IPAS'03), 2003, pp. 199-204.

[2] S. Koelemeijer Chollet, F. Bourgeois, and J. Jacot, "Economical justification of flexible microassembly cells," in *Proceedings of IEEE International Symposium on Assembly and Task Planning (ISATP03),* Juillet 2003.

[3] S. Koelemeijer Chollet, F. Bourgeois, L. Benmayor, B. Moll, C. Wulliens, and J. Jacot, "A Flexible Microassembly Cell for Small and Medium Sized Batches," in *Proceedings of 33rd International Symposium on Robotics*, 2002.

[4] T. Zimmerman, *Capturing a-priori knowledge by training two-dimensional deformable templates*, Ph.D. thesis, EPFL, Switzerland, September 2002.

[5] Gill, P.E. and Murray, W. and Wright, M.G., *Numerical linear algebra and optimization*, Addison-Wesley,