Automatic Generation of Image Analysis Programs[1,2]

M. Herrmann, C. Mayer, and B. Radig

Image Understanding and Knowledge-Based Systems, Technische Universität München, Boltzmannstrasse 3, Garching, D-85748, Germany

e-mail: {herrmmic, mayerc, radig}@in.tum.de

Abstract—In this paper, we introduce a system that generates computer vision programs for a given task, which is specified by regions of interest in a collection of example images. The system relies on a database of operators, which are combined by an automated planning approach in order to create executable programs. We present an early proof–of–concept implementation that relies on a limited database to solve simple tasks, such as finding players in a soccer video or cups on a table. Our experimental evaluation shows that the basic approach is working on relative simple scenarios. Future work will focus on integrating more complex problem descriptions, which require more sophisticated planning strategies in order to compensate for rapidly increasing search spaces.

Keywords: automatic programming, inductive programming, generate–and–search, machine learning, computer vision, image analysis, object detection

DOI: 10.1134/S1054661814030079

1. INTRODUCTION

Traditionally, program code is manually designed and written by programmers, who select operators and parameter values and combine them to create an executable program. For that purpose, the programmers rely on their experience and knowledge and on the technical documentation of operators. After the program is composed, the programmer executes it on sample data to inspect its behavior and to ensure that its output matches the task specification. Finding an appropriate parameterization often requires many iterations. Especially for image analysis programs, alteration of parameters may change the complete behavior of a program, and determining the optimal parameter values is a crucial and time-consuming task.

In this paper, we propose a different approach, which creates the program code automatically given a set of annotated images with positive examples. This takes into account that providing the desired output is much more intuitive and less error prone than writing the complete program code and specifying parameter values. Our systematic approach is based on a database containing technical information about all available operators and how these operators may be combined. The system uses this database to create a large number of executable programs by exploring suitable combinations of operators and parameter values. The output of these programs is compared to the given specification. Currently, an early version of our system exists as a proof-of-concept implementation that includes a limited number of operators and therefore solves only basic computer vision tasks. So far, two sample applications have been inspected: the detection of players in a soccer match and the detection of cups on a table. However, these experiments show that the basic approach is successful.

2. RELATED WORK

Regarding automatic program synthesis, we distinguish between semi-automatic approaches and fully automatic approaches. Semi-automatic approaches create programs based on the user's specification of the program structure. Mostly easy-to-use programming interfaces are proposed, which assist the user in composing a complex system using primitive modules. The user interface provides the user with graphical representations of modules, which can be combined in the user interface to form a program. Afterwards, real program code in a programming language is created from these specifications. The advantage of this approach is that its method of programming is much more intuitive than the traditional one and that the transformation of the graphical program representation to the textual representation is usually fast.

In contrast, fully automatic approaches do not rely on any specification of the program structure, but work on specifications of desired program properties, such as given input-output pairs or sample data. This is also referred to as *inductive programming*. The

[1] The article is published in the original.

[2] This paper uses the materials of the report submitted at the 11th International Conference "Pattern Recognition and Image Analysis: New Information Technologies," Samara, The Russian Federation, September 23–28, 2013.

Received February 28, 2014

advantage of these approaches is that the person providing the test data doesn't have to be a programmer. On the other hand, creating the program is often computationally expensive. Kitzelmann distinguishes two major approaches for automated program construction [12]: Firstly, *analytical* approaches construct programs directly from the given input-output examples according to a set of fixed rules. Secondly, *search-based* approaches conduct a search in the program space and derive a fitness function from the examples. The latter methods rely on finding the program with the best fitness value. The advantage of search-based approaches is their flexibility, but they do not guarantee determining the optimal program. Kitzelmann provides a broad, yet very theoretical review of fully automatic program synthesis [12]. A program is abstracted as a function that delivers specified output to specified input. The task of inductive program synthesis is to find a program that generalizes from some example input-output pairs. Another survey is provided by Hoffmann et al. [7]. They evaluate seven systems for inductive programming using eleven different challenges. They demonstrate that the execution time varies greatly depending on the system and the task chosen and that it is, all in all, not possible to nominate the fastest approach. They conclude that still much remains to do in the area of automatic programming.

2.1. Semi-Automatic Approaches

A well-known example for this type of code generation is Simulink [23], an extension of MATLAB [22]. It provides the user with a graphical user interface to construct programs and creates MATLAB code after a compilation step. Another example is Reo, a system that creates a web service by combining other web services [10]. The developers are presented with a graphical user interface, in which they compose various web-based services and model the information flow. Afterwards, executable code is generated from these specifications. Rodrigues et al. present a similar system: a UML diagram is transformed in multiple steps and finally executable code is created [20]. The software HALCON, distributed by the company MVTec Software GmbH, provides helpful tools for image acquisition, camera calibration, and other computer vision tasks, which create code snippets to use in larger programs [6]. Although these systems provide automatic code generation on a certain level, they still require a lot of human expertise. Specifically, a human programmer still has to declare the program structure by "writing" the program in the graphical user interface. However, a graphical user interface is not always used to define the program structure. Reyes et al. present a code–to–code compiler that is used to generate highly parallelized code [19]. The user writes code in a C–like language and their system creates code to be executed on GPUs.

2.2. Analytical Approaches

An example for direct-derivative program synthesis is presented by Hofmann with his IGOR II system [8]. It is implemented in Haskell and derives a program from a list of functions partially specifying input-output examples. Remarkable about IGOR II is, that it is able to create recursive programs, as well. Crossley et al. propose to combine analytical and search-based strategies [2]. They utilize IGOR II to create seeds for a search with the ADATE system (see section 2.3). An evaluation using 5 sample programs provides promising results. In this area only a very few approaches exist, and they are mainly evaluated using rather theoretical challenges, such as developing a sorting algorithm or reversing a list of elements. According to the best of our knowledge, no algorithm exists in this area that has been applied to computer vision tasks.

2.3. Search-Based Approaches

This category includes not only search-based approaches, but also evolutionary programming. Evolutionary programming is inspired by the principles of Darwinism, where the population consists of program candidates and a fitness function defines which individuals are selected for reproduction to form the basis for the next generation. In an iterative procedure, a new generation of programs is created by modifying or merging the most suitable programs of the former generation, which are evaluated by a fitness function. An exhaustive overview is presented by Koza et al. [13].

An example for an evolutionary approach is presented by Vu et al. with their system ADATE, which was developed several years ago and has been applied to various challenges [24]. Recently, they demonstrated an application to graph-based image segmentation, a well-known image analysis procedure. Although, their work improved the basic approach dramatically, the computational cost needed to find this improved algorithm was also very high: the computation took 24 hours on 192 cores. Another example is presented by Möhrmann et al. [14]. Their system generates image analysis software by training classifiers with manually annotated training data. Demmel et al. consider linear algebra applications, like multiplying two matrices, and create the algorithms by searching in the program space for the most suitable program with respect to computational efficiency [3]. The reason for this time-consuming approach is that the optimal implementation depends on many factors, such as the available hardware. Olague et al. present an evolutionary programming-based approach for interest point detection [16]. They use approximately 20 operators and model programs as trees. Modification of the program therefore becomes modification of a subtree. Furthermore, they provide a broad introduction to evolutionary programming. Katayama proposes the MagicHaskeller system that conducts an exhaustive

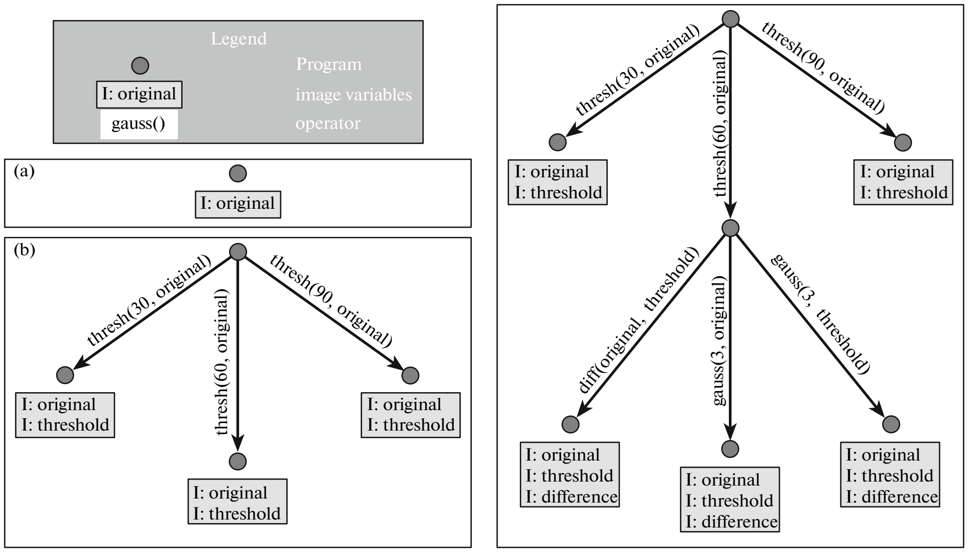

Fig. 1. The planner creates programs by combining single operators and instantiating the input parameters.

search in the space of Haskell programs and determines the fitness of a program using given input-output pairs [11]. A library of a Haskell-implementation of the system is publicly available and has been tested by other scientists doing research in the same field, as well. An early version of the system conducts an exhaustive search without any search heuristics, but recently heuristics have been added to speed-up the process.

Our approach also falls into this category. It has many similarities to automatic planning, which considers the creation of plans as a combination of actions with preconditions and postconditions. Preconditions have to be true before the action can be executed and postconditions model the influence of the actions on the world model [18]. In relation to our system, actions correspond to operators, preconditions correspond to the input variables for operators, and postconditions correspond to the output variables of operators. Often a heuristic is included, which controls the search tree expansion and prevents unsuitable combinations from being inspected [21].

3. SYSTEM DESCRIPTION

The basic idea of our approach is to create combinations of parameterized operators with the help of a heuristic search strategy and to apply them to manually annotated images. A fitness function determines to what degree their results match the annotations. Therefore, the system consists of several components:

—A database of available operators. For each operator, the database contains the types of variables that the input operator requires for execution and also the types of output variables that are created by the operator.

—A planner, which creates a search tree with programs in its nodes.

—A system, which checks a single program against the manually specified test data.

—A set of manually annotated sample images.

The core component of this approach is the planner. Therefore, it will be presented in greater detail.

3.1. The Planning of Programs

The planner recursively creates a search tree whose nodes are programs. Programs contain global variables (either specified as the input of the program or created by operators) and a sequence of the applied, parameterized operators. The planning process starts with the empty program, which contains only the original input image in the global variable list and an empty operator list. In a recursive procedure, the planner inspects all available operators and checks whether they are applicable, in other words, whether all input parameters can be assigned by already existent variables of the same type in the variable list of the current recursion path. Here, different types of operator parameters are handled in two ways. Iconic parameters (images or image regions) have to be present in the global variable list, whereas control parameters (numerical values, strings, etc.) are instantiated on–the–fly, dependent on some sampling strategy and the specification of the operator. For each combination of operator and parameterization, a new branch in the search tree is created.

An example is given in Fig. 1. In the first step (a), only the original image is in the global variable list and only operators that work on a single image are applicable. For instance, a threshold operator is applicable, since it requires only a single image and a numerical

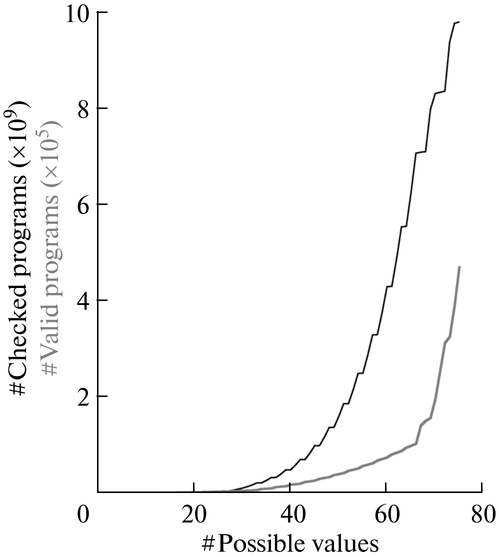

Fig. 2. The number of syntactic correct programs (black) and the number of valid programs (according to section 3.2, gray) are plotted against the number of possible control values in our cup examples (see Tables 2 and 1) with a maximum program length of four operators. Note the reduction of the search space by a factor of 2×10^4 due to the heuristic rules of section 3.2.

value, which is created on-the-fly. In this example the numerical parameter is sampled by three values, namely 30, 60 and 90. The planner then creates a tree node for each of these operators and attaches them to the father node (in this example the root). The search tree in step (b) consists of several nodes, each with the original image and a threshold image in the image list. As displayed in step (c), these nodes may now be expanded further with operators that use two images (the original image and the threshold image) or again with a single image operator, which now may choose between the original image and the threshold image. This procedure continues until a predefined depth of the tree is reached. Now each node and each leaf contains a program candidate that will be evaluated with the help of a fitness function (see section 3.3). Furthermore, this requires the programs to be compiled and executed with the manually annotated image data as input.

3.2. Heuristics for Search-Space Reduction

The programs created by the planning algorithm in section 3.1 are syntactically correct. That is, every assignment to an input parameter has the appropriate data type. However, each program will be evaluated with the help of a fitness function (see section 3.3) and there is an enormous amount of syntactically correct programs. To reduce the computational cost during the assessment of the programs, we apply a set of rules to identify programs that are not considered for evaluation. Although it is a simple heuristic method, the search space is reduced by several orders of magnitude as depicted in Fig. 2.

We apply the following four rules (in the order of verification during program creation):

—An operator whose inputs are outputs of the same operator at a previous position in the program, is considered redundant. Normally, the result of a repeated execution of the same operator can be achieved by a single execution with appropriate parameterization. For example, instead of applying a smoothing filter two times, one can apply the filter once with an increased mask size. (SELFCALL, H_{SC})

—An operator is considered redundant if the same operator with exactly the same parameter assignment is already present at a previous position in the program. (DUPLICATE, H_{DUP})

—A program whose last operator does not have the same output type as the desired program output (e.g. image region(s)) is not considered in the evaluation step. (VALID_OUTPUTS, H_{VO})

—An operator (except the last operator in the program) is considered redundant if none of its outputs is assigned to an input of an operator at a subsequent position in the program. (NO_DEPENDENCIES, H_{NODEP})

3.3. Assessment of Program Candidates

After generation and selection of appropriate program candidates (as illustrated in sections 3.1 and 3.2), the programs are assessed according to the task description and a corresponding fitness function. The program with the highest fitness is the final outcome of the system.

In our example, the task is to detect objects of a given class in images. The task description is given in terms of a set of annotated sample images $I_1, \ldots, I_n$. For each sample image I_i a set B_i^a of k_i axis-aligned rectangular bounding boxes $b_i^1, \ldots, b_i^{k_i}$ is specified. Each bounding box describes the visible area of an object (e.g., a cup) in the image. Annotation is performed in accordance with the guidelines of the PASCAL Visual Object Class Challenge [25].

The foundation of our fitness function is the overlap ratio $r(b_1, b_2)$ of two bounding boxes b_1 and b_2 (see [5]):

$$r(b_1, b_2) := \frac{\mathrm{area}(b_1 \cap b_2)}{\mathrm{area}(b_1 \cup b_2)} \quad (1)$$

The cover ratio $c(b_1, b_2)$ is defined as

$$c(b_1, b_2) := \frac{\mathrm{area}(b_1 \cap b_2)}{\mathrm{area}(b_1)}. \quad (2)$$

Note that the cover ratio is not symmetric and that $r(b_1, b_2) \in [0; 1]$ and $c(b_1, b_2) \in [0; 1]$ hold. Given a

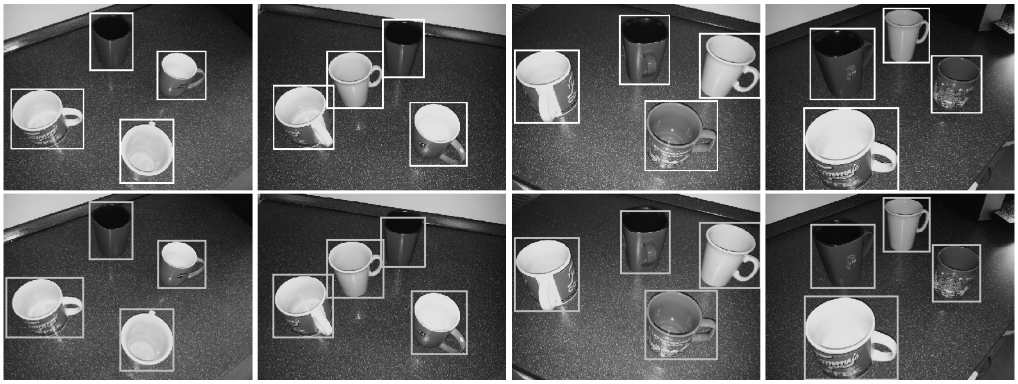

Fig. 3. Annotated images (top) and results of the best program (bottom) for the cups example.

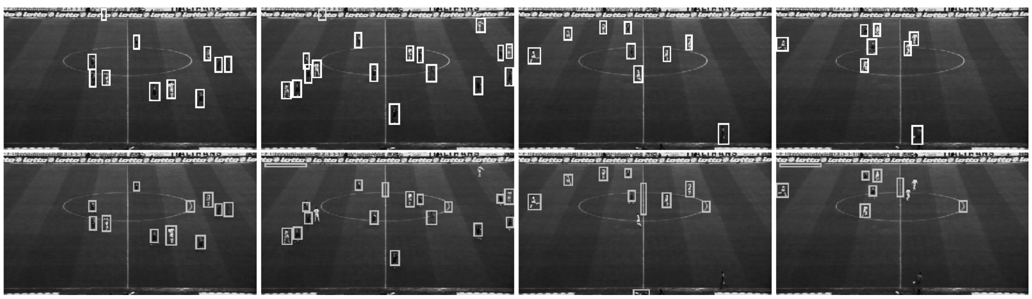

Fig. 4. Annotated images (top) and results of the best program (bottom) for the soccer example (image source: [4]).

set of predicted bounding boxes B_p and a set of annotated bounding boxes B_a, we calculate the F1 score $f_1(B_p, B_a)$ (which is the harmonic mean of precision and recall, i.e., $f_1(B_p, B_a) \in [0; 1]$) with respect to the overlap of bounding boxes. In spite of some drawbacks (see [17]), we use the F1 score because of its simplicity. To calculate overlaps, an assignment between predicted bounding boxes and annotated bounding boxes has to be established. This is usually done using Munkres's algorithm [15]. For reasons of lower computational cost, we perform a greedy assignment procedure similar to [1]. Algorithm 1 offers a detailed description. We choose $t = 0.4$ (instead of $t = 0.5$ as suggested by [5]). A lower threshold favors a higher detection rate at the cost of a lower positional precision.

Each program P in the set of program candidates is executed n times with each sample image I_i as input, respectively. It has a set of possible results R_P, which consist of all outputs of its last operator that match the data type of the annotation (in our example: image region(s)). Every possible result $R_P^k \in R_P$ provides a set of image regions $R_{P_i}^k$ for each input image I_i. In order to determine the F1 score, the axis-aligned minimum bounding box rectangle is calculated for each region in $R_{P_i}^k$, resulting in the set of bounding boxes $B_{P_i}^k$. The designated output of a program is the possible result with the highest geometric mean of the F1 score taken over all sample images. The program's fitness $w(P)$ is set to this score, i.e.:

$$w(P) := \max_{R_P^k \in R_P} \sqrt[n]{\prod_{i=1}^{n} f_1(B_{P_i}^k, B_i^a)}. \qquad (3)$$

4. EXPERIMENTAL EVALUATION

We demonstrate our proof–of–concept with the help of two simple examples: the detection of cups on a table and the detection of humans in soccer videos. Each image dataset comprises four images scaled to a size of 640 × 480 pixel (cups) and 640 × 360 pixel (soccer). The annotated images of both examples can be found in the top row of Fig. 3 and Fig. 4, respectively.

Our operator database contains a selection of six basic image processing operators (see Table 1) and the program length is limited to a maximum of four operators. The operators provided are two smoothing operators (`gauss_image` and `median_sqr`), a color space conversion from RGB to HSV (`trans_rgb`), a

Table 1. List of image processing operators. Legend: (I)—Input, (O)—Output

Operator	Description	Input/Output	Data type
gauss_image	Smooth image using discrete Gauss functions.	(I) image (I) size (O) image_gauss	color_image integer color_image
median_sqr	Compute a median filter with a square mask.	(I) image (I) size (O) img_median	color_image integer color_image
trans_rgb	Transformation of an image from RGB color space to HSV color space. Resulting channels are returned as single gray images.	(I) image (O) image_H (O) image_S (O) image_V	color_image gray_image gray_image gray_image
dev_image	Calculate for each color channel the standard deviation of gray values within a square window and take the maximum over all channels.	(I) image (I) size (O) image_dev	color_image integer gray_image
threshold	Segment an image using global threshold.	(I) image (I) min_gray (I) max_gray (O) region	gray_image integer integer region
holes_max_comp	Calculate connected components, take the holes of the biggest component (i.e. difference of convex hull and the region itself) and filter holes according to the size of their axis-aligned minimum surrounding rectangles.	(I) region (I) min_area (I) max_area (O) holes	region integer integer region_array

Table 2. Control parameters with their domains for the cup example

Operator	Input/Output	Domain
gauss_image	(I) size	{3, 7, 11}
median_sqr	(I) size	{3, 7, 11}
dev_image	(I) size	{3, 7, 11}
threshold	(I) min_gray	{0, 5, 15, 25, ..., 245}
	(I) max_gray	{10, 20, ..., 250, 255}
holes_max_comp	(I) min_area	{0, 1500, 3000, ..., 18000}
	(I) max_area	{70000}

deviation filter (dev_image), a threshold operator (threshold), and a operator for region filtering (holes_max_comp).

The best configuration of the operators is determined automatically by our system without any further user interaction. There are $\sum_{i=0}^{n} k^i = \frac{k^{n+1}-1}{k-1}$ possibilities to compose a program of maximum length n with k operators (in our case with n = 4 and k = 6 there are 1555 possibilities). However, the search space explosion (see Fig. 2) is primarily induced by the assignment of the parameters. First of all, for every operator arrangement all possible assignments of iconic input parameters (with respect to the correct data type) are taken into consideration. Furthermore the control parameters are instantiated on–the–fly, as mentioned in section 3.1. We use a straightforward approach, which is similar to grid search for hyperparameter optimization in machine learning applications (see e.g., [9]). The system considers every possible combination of input parameter instantiations. The possible values for each input control parameter are defined in the operator database and are depicted in Table 2 and Table 3, respectively.

Note that implicit background knowledge is modeled by the domains of the parameters. For example, the possible values for the min_area and max_area parameters of the holes_max_comp operator reflect some assumptions about the object size in the images.

In both examples, the planner created about 9.8×10^9 programs. After the application of the rules in section 3.2 only approximately 500.000 valid programs remained for evaluation. The detailed effects of the different heuristic rules are depicted in Table 4. The most useful filter by far is the rejection of programs with unused operators (NO_DEPENDENCIES, H_{NODEP}).

Table 3. Control parameters with their domains for the soccer example

Operator	Input/Output	Domain
`gauss_image`	(I) `size`	{3, 7, 11}
`median_sqr`	(I) `size`	{3, 7, 11}
`dev_image`	(I) `size`	{3, 7, 11}
`threshold`	(I) `min_gray`	{0, 5, 15, 25, ..., 245}
	(I) `max_gray`	{10, 20, ..., 250, 255}
`holes_max_comp`	(I) `min_area`	{0, 25, 50, ..., 300}
	(I) `max_area`	{1250}

Table 4. Statistics of the cup experiment for different maximum program lengths: the number of syntactically correct programs, the number of valid programs and, the number of programs rejected by the different rules of section 3.2. All program counts have the magnitude of 10^6

Max. Length	Programs	Valid	H_{SC}	H_{DUP}	H_{VO}	H_{NODEP}
3	5.691	0.081	0.001	0.017	0.039	5.553
4	9798.3	0.5	1.5	56.3	51.6	9688.4
5	>1600000	>1.6				

Table 5. The best program for the cups example (fitness: 0.967)

Input: `image_in`
1: `gauss_image (image_in, 3)` ⟶ `(image_gauss)`
2: `trans_rgb (image_gauss)` ⟶ `(image_H, image_S, image_V)`
3: `threshold (image_H, 95, 190)` ⟶ `(region)`
3: `holes_max_comp (region, 13500, 70000)` ⟶ `(holes)`
Output: `holes`

Table 6. The best program for the soccer example (fitness: 0.605)

Input: `image_in`
1: `median_sqr (image_in, 7)` ⟶ `(image_median)`
2: `dev_image (image_median, 11)` ⟶ `(image_dev)`
3: `threshold (image_dev, 0, 20)` ⟶ `(region)`
3: `holes_max_comp (region, 250, 1250)` ⟶ `(holes)`
Output: `holes`

The generation and evaluation of the programs took about 11 hours on a standard 3.0 GHz CPU with our unoptimized implementation. The programs with the best fitness can be seen in Table 5 and Table 6 and their results in the bottom row of Fig. 3 and Fig. 4, respectively.

Even in the more challenging real-world soccer example, the result of our simple example is quite satisfactory, with the fitness of the best program being $w = 0.605$. As can be seen in Table 7, the majority of valid programs have zero evaluation fitness ($w = 0$). Further investigation of these programs could lead to appropriate semantic rules and an even more restricted search space.

5. FUTURE WORK

We have demonstrated a simple proof-of-concept that worked on simple problem statements. In the future, a more powerful operator database is necessary to solve complex problems. As already mentioned, the sampling strategy and the heuristics for the tree expansion heavily influence the size of the search space and thus the runtime. The high ratio between created programs and valid programs in the area of several decimal powers shows that there is still much room for improvement of the search function. Not enumerating invalid programs would speed up the search process dramatically. A further improvement of the search strategy could be gained by including a semantical search heuristic, which would require including background knowledge. This could be used to favor suitable

Table 7. The distribution of valid programs according to the fitness w for the cups example and the soccer example, respectively.

Example	$w = 0$	$0 < w \leq 0.5$	$0.5 < w \leq 1$
Cups	390.608	58.362	21.526
Soccer	432.558	37.704	234

Algorithm 1: Calculation of F1 score with greedy data assignment

Input: Set B_p of predicted bounding boxes
Set B_a of annotated bounding boxes
Overlap threshold $t \in [0; 1]$;
Output: F1 score $f_1(B_\mathrm{p}, B_\mathrm{a})$
$f_1 \longleftarrow 0$; $tp \longleftarrow 0$; $fp \longleftarrow 0$; $fn \longleftarrow 0$;
foreach $b_a \in B_a$ **do**
  // Get bounding box with best overlap.
  $b_{p_{best}} \longleftarrow \arg\max_{b_p \in B_p} r(b_p, b_a)$
  if $r(b_{p_{best}}, b_a) > t$ **then**
    // Increment true positives, false positives
    // and false negatives according to overlap.
    $tp \longleftarrow tp + c(b_a, b_{p_{best}})$
    $fp \longleftarrow fp + (1 - c(b_{p_{best}}, b_a))$
    $fn \longleftarrow fn + (1 - c(b_a, b_{p_{best}}))$
    // Remove assigned predicted bounding box.
    $B_p \longleftarrow B_p \backslash b_{p_{best}}$
  else
    $fn \longleftarrow fn + 1$ *// Increment false negatives.*
// Increment false positives for every
// non-assigned predicted bounding box.
foreach $b_p \in B_p$ **do** $fp \longleftarrow fp + 1$
// Calculate F1 measure.
$d \longleftarrow 2tp + fp + fn$
if $d \neq 0$ **then** $f_1 \longleftarrow 2tp/d$
return f_1

operators or to choose more suitable parameter values. Furthermore, programs that have already been created could be included in the search strategy, either as single operators or as the starting point for a search. This way, by inspecting a similar, formerly solved problem, if it exists, the solution may be determined faster. Furthermore, our current approach does not consider control structures like if-clauses or loops. Finally, simple regions of interest are not capable of representing the targets of many real-world vision applications. Thus, a long-term goal is to incorporate more complex target specifications. This is the most challenging research opportunity, because it not only requires adapting the task specification procedure and the search heuristics, but also requires taking knowledge about the task context into consideration.

CONCLUSION

We have presented a system that automatically creates and evaluates computer vision programs for a given task. The combination of computer vision and automatic programming is a relatively new approach. Most fully automatic programming approaches are evaluated on rather theoretical tasks. In our current, early version, the task is specified by rectangular regions of interest in sample images. Our system utilizes six operators with a maximum program length of four instructions and a total of seven control parameters. Our evaluation shows that the basic approach is working, but capability and performance have to be improved. Further improvements will focus on defining a more efficient search strategy and on reducing the search space.

REFERENCES

1. M. D. Breitenstein, F. Reichlin, B. Leibe, E. Koller-Meier, and L. Van Gool, "Online multiperson tracking-by-detection from a single, uncalibrated camera," IEEE Trans. Pattern Anal. Mach. Intellig. **33** (9), 1820–1833 (2011).
2. N. Crossley, E. Kitzelmann, M. Hofmann, and U. Schmid, "Combining analytical and evolutionary inductive programming," in *Proc. 2nd Conf. on Artificial General Intelligence, AGI 2009* (Atlantis Press, Amsterdam-Paris, 2009), pp. 19–24.
3. J. Demmel, J. Dongarra, V. Eijkhout, E. Fuentes, A. Petitet, R. Vuduc, R. C. Whaley, and K. Yelick, "Self-adapting linear algebra algorithms and software," Proc. IEEE **93** (2), 293–312 (2005).
4. T. D'Orazio, M. Leo, N. Mosca, P. Spagnolo, and P. L. Mazzeo, "A semiautomatic system for ground truth generation of soccer video sequences," in *Proc. 6th IEEE Int. Conf. on Advanced Video and Signal Based Surveillance, AVSS '09* (IEEE Computer Soc., Los Alamitos, 2009), pp. 559–564.
5. M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes (VOC) challenge," Int. J. Comput. Vision **88** (2), 303–338 (2010).
6. MVTec Software GmbH. HALCON 11 (Munich, 2012). http://www.halcon.com (Accessed Feb. 27, 2014).
7. M. Hoffmann, E. Kitzelmann, and U. Schmid, "A unifiying framework for analysis and evaluation of inductive programming systems," in *Proc. 2nd Conf. on Artificial General Intelligence, AGI 2009* (Atlantis Press, Amsterdam-Paris, 2009), pp. 55–60.
8. M. Hofmann, "IGOR2 – an analytical inductive functional programming system: tool demo," in *Proc. 2010 ACM SIGPLAN Workshop on Partial Evaluation and Program Manipulation, PEPM'10* (ACM, New York, 2010), pp. 29–31.
9. C.-W. Hsu, C.-C. Chang, and C.-J. Lin, "A practical guide to support vector classification," Tech. Report (Department of Computer Science, National Taiwan

Univ., July 2003). http://www.csie.ntu.edu.tw/cjlin/papers/guide/guide.pdf (Accessed Feb. 27, 2014).

10. S.-S. T. Q. Jongmans, F. Santini, M. Sargolzaei, F. Arbab, and H. Afsarmanesh, "Automatic code generation for the orchestration of web services with Reo," in *Service-Oriented and Cloud Computing* (Springer, Berlin-Heidelberg, 2012), pp. 1–16.
11. S. Katayama, "Recent improvements of MagicHaskeller," in *Approaches and Applications of Inductive Programming* (Springer, Berlin-Heidelberg, 2010), pp. 174–193.
12. E. Kitzelmann, "Inductive programming: a survey of program synthesis techniques," in *Approaches and Applications of Inductive Programming* (Springer, Berlin-Heidelberg, 2010), pp. 50–73.
13. J. R. Koza, *Genetic Programming: on the Programming of Computers by Means of Natural Selection* (MIT Press, Cambridge, 1992).
14. J. Möhrmann, G. Heidemann, O. Siemoneit, C. Hubig, U.-P. Kaeppeler, and P. Levi. "Context generation with image based sensors: an interdisciplinary enquiry on technical and social issues and their implications for system design," Proc. World Acad. Sci., Eng. Technol. **74**, 1191–1197 (2010).
15. J. Munkres, "Algorithms for the assignment and transportation problems," J. Soc. Industr. Appl. Math. **5** (1), 32–38 (1957).
16. G. Olague and L. Trujillo, "Evolutionary-computer-assisted design of image operators that detect interest points using genetic programming," Image Vision Comput. **29** (7), 484–498 (2011).
17. D. M. W. Powers, "Evaluation: from precision, recall and f-measure to ROC, informedness, markedness & correlation," J. Mach. Learn. Technol. **2** (1), 37–63 (2011).
18. J. Rao and X. Su, "A survey of automated web service composition methods," in *Semantic Web Services and Web Process Composition* (Springer, Berlin-Heidelberg, 2005), pp. 43–54.
19. R. Reyes and F. de Sande, "Automatic code generation for GPUs in llc," J. Supercomput. **58** (3), 349–356 (2011).
20. A. W. O. Rodrigues, F. Guyomarc'h, and J.-L. Dekeyser, "An MDE approach for automatic code generation from UML/MARTE to OpenCL," Comput. Sci. Eng. **15** (1), 46–55 (2013).
21. F. Rossi, P. van Beek, and T. Walsh, *Handbook of Constraint Programming* (Elsevier, Amsterdam, 2006), Vol. 2.
22. The MathWorks, Inc. MATLAB R2013b (Natick, 2013). http://www.mathworks.de/products/matlab (Accessed Feb. 27, 2014).
23. The MathWorks, Inc. Simulink R2013b (Natick, 2013). http://www.mathworks.de/products/simulink/ (Accessed Feb. 27, 2014).
24. H. Vu and R. Olsson, "Automatic improvement of graph based image segmentation," in *Advances in Visual Computing* (Springer, Berlin-Heidelberg, 2012), pp. 578–587.
25. J. Winn and M. Everingham, The PASCAL visual object classes challenge 2007 (VOC2007) annotation guidelines (2007). http://pascallin.ecs.soton.ac.uk/challenges/VOC/voc2007/guidelines.html (Accessed Feb. 27, 2014).

Michael Herrmann, born 1980, received his diploma degree in computer science from the Julius–Maximilians–Universität Würzburg in 2007. From 2007 to 2012 he was a software engineer for industrial computer vision systems in private industry. Additionally, he was a research assistant at the Hochschule Rosenheim from 2010 to 2012. Since 2012 he is part of the research group Image Understanding and Knowledge-Based System at the Technische Universität München headed by Bernd Radig. His research interests include computer vision, machine learning, and especially the detection and tracking of objects in videos.

Christoph Mayer studied Computer Science at the Technische Universität München from 2000 to 2007 and received his doctoral degree in 2012. While he was working on his Ph.D, he has been working in the German Cluster of Excellence "Cognition for Technical Systems" in the Intelligent Autonomous Systems Group. His research interests were in the field of face model fitting, facial expression recognition and emotion recognition. He has been first author of the paper "Adjusted Pixel Features for Facial Component Classification" that appeared in the Vision and Image Computing Journal in 2009 and has been awarded with the best paper award in 2009 for the paper "Facial Expression Recognition with 3D Deformable Models" that has been presented at the conference "Advances in Computer-Human Interaction". His current research interest is in the automatic analysis of soccer games from optical camera data.

Bernd Radig received his diploma degree in Physics in 1972 from the University of Bonn and the doctor degree in Computer Science in 1978 from the University of Hamburg. There he got his venia legendi and finished his habilitation dissertation in 1982. He was Assistant and Associate Professor in Hamburg (1982–1986) and full professor, chair of Image Understanding and Knowledge Based Systems, Fakultät für Informatik, Technische Universität München (1986-2009). He is a member of the *Emeriti of Excellence* programme. He was chairman and founder of the Association of Bavarian Research Cooperations (1993–2007), a unique network of scientists, specialising in challenging disciplines in accordance with Bavarian enterprises. 1988 he founded the Bavarian Research Centre for Knowledge Based Systems (FOR–WISS), an institute common to the three universities TU München, Erlangen and Passau. He was general chairman of the annual symposium of the German Association for Pattern Recognition in 1981, 1991, 2001 as well as of the European Conference on Artificial Intelligence (ECAI), 1988. He is active as organizer and programme committee member of the German-Russian Workshop on Pattern Recognition. He holds the German Order of Merit (1992) and the award *Pro Meritis Scientiae et Litterarum* of the State of Bavaria for outstanding contributions to science and art (2002). His current research activities are in real-time image sequence understanding for applications in robotics, sports or driver assistance systems.