THE FLY ALGORITHM REVISITED: ADAPTATION TO CMOS IMAGE SENSORS

Emmanuel Sapin[1], Jean Louchet[1,2], Evelyne Lutton[1]
[1] *INRIA-Saclay, 4 rue Jacques Monod, F-91893 Orsay Cedex France*
[2]*Artenia, 24 rue Gay-Lussac, F-92320 Châtillon, France*
Emmanuel.Sapin@inria.fr, jean.louchet@gmail.com, evelyne.lutton@inria.fr

Keywords: Evolutionary Algorithm, Cooperative Coevolution, Computer Vision, Fly Algorithm, Image Sensor.

Abstract: Cooperative coevolution algorithms (CCEAs) usually represent a searched solution as an aggregation of several individuals (or even as a whole population). In other terms, each individual only bears a part of the searched solution. This scheme allows to use the artificial Darwinism principles in a more economic way, and the gain in terms of robustness and efficiency is important. In the computer vision domain, this scheme has been applied to stereovision, to produce an algorithm (the fly algorithm) with asynchronism property. However, this property has not yet been fully exploited, in particular at the sensor level, where CMOS technology opens perpectives to faster reactions. We describe in this paper a new coevolution engine that allow the Fly Algorithm to better exploit the properties of CMOS image sensors.

1 INTRODUCTION

Image processing and Computer vision are now an important source of problems for EC community, and various successful applications have been advertised up to now (Cagnoni et al., 2008). There are many reasons for this success, mainly due to the fact that stochastic and adaptive methods are convenient to address some ill-defined, complex and computationally expensive computer vision tasks (Horn, 1986). The great majority of EC image and vision applications is actually dealing with computationnally expensive aspect. There exists however less known issues related to real-time processing where EC techniques have been proven useful.

In stereovision, a cooperative coevolution algorithm[1], *the fly algorithm* (Louchet, 2000; Louchet, 2001; Louchet and Sapin, 2009), has been designed for a rapid identification of 3D positions of objects in a scene. This algorithm evolves a population of 3-D points, *the flies*, so that the population matches the shapes of the objects on the scene. It is a cooperative coevolution in the sense that the searched solution is represented by the whole population rather than by the single best individual. The *anytime* property of this algorithm has been discussed in (Boumaza and Louchet, 2001). It has been exploited in particular through the development of ad-hoc asynchronous robot controllers (Boumaza and Louchet, 2003). However, the advantage of being an asynchronous algorithm has not yet been fully exploited, due to the rigid sequential delivery of images by conventional sensors. This is the point we are examining in this paper. The paper is organised as follows: section 2 is an overview of the original fly algorithm, then section 3 presents the characteristics of CMOS image capture devices that can be exploited in the core of the fly algorithm (section 4). A computational analysis is developed in section 5 and a conclusion is given in section 6.

[1]These cooperative-coevolution algorithms are also called "Parisian approach."

2 CCEAS AND FLIES

2.1 Cooperative Coevolution

Cooperative coevolution strategies actually rely on a formulation of the problem to be solved as a cooper-

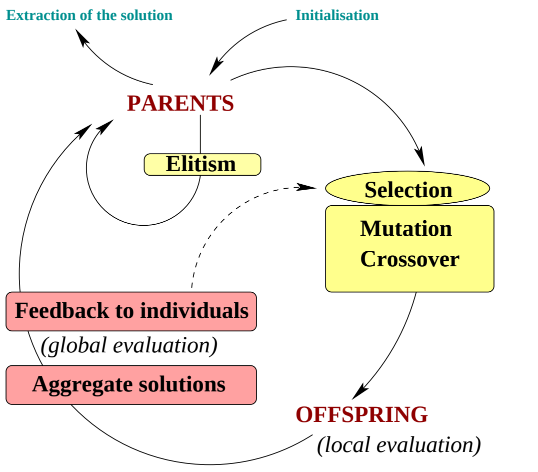

Figure 1: A Parisian EA: a monopopulation cooperative-coevolution

ative task, where individuals collaborate or compete in order to build a solution. They mimic the ability of natural populations to build solutions via a collective process. Nowadays, these techniques have been used with success on various problems (Jong et al., 2007; Wiegand and Potter, 2006), including learning problems (Bongard and Lipson, 2005). A large majority of such approaches deals with a coevolution process that happens between a fixed number of separated populations (Panait et al., 2006; Bucci and Pollack, 2005).

We study here a different implementation of cooperative coevolution principles, the so-called Parisian approach (Collet et al., 2000; Ochoa et al., 2007) described on figure 1, that uses cooperation mechanisms within a *single* population. It is based on a two-level representation of an optimization problem, where an individual of a Parisian population represents only a part of the solution to the problem. An aggregation of multiple individuals must be built in order to obtain a solution to the problem. In this way, the coevolution of the whole population (or a major part of it) is favoured instead of the emergence of a single best individual, as in classical evolutionary schemes.

The motivation is to make a more efficient use of the genetic search process, and reduce computational expense. Successful applications of such a scheme usually rely on a lower cost evaluation of the partial solutions (i.e. the individuals of the population), while computing the full evaluation only once at each generation.

The fly algorithm is a direct application of this principle to stereovision (see section 2.2). It is actually an extreme case, as it is so well conditionned for CCEA that there is no need to compute a global fitness evaluation for feedback to individuals. A local

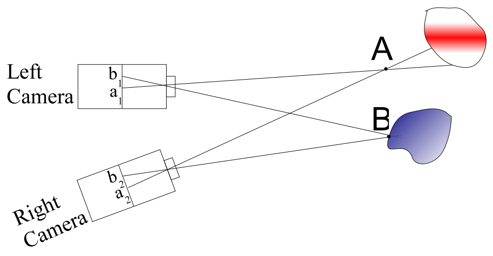

Figure 2: Pixels b_1 and b_2, projections of fly B, get identical grey levels, while pixels a_1 and a_2, projections of fly A, which receive their illumination from two different physical points on the objectifs surface, get different grey levels.

(and computationnally efficient) evaluation is enough to run the loop of figure 1. With appropriate parameter tuning it is then possible to obtain "real-time" evolution for video sequences.

2.2 Principle of the Fly Algorithm

An individual of the population, i.e. a *fly*, is defined as a 3-D point with coordinates (x,y,z). As presented in (Louchet and Sapin, 2009), if the fly is on the surface of an opaque object, then the corresponding pixels in the two images will normally have highly similar neighbourhoods as shown in figure 2. Conversely, if the fly is not on the surface of an object, their close neighbourhoods will usually be poorly correlated. The fitness function exploits this property and evaluates the degree of similarity of the pixel neighbourhoods of the projections of the fly, giving higher fitness values to those probably lying on objects surfaces.

2.3 Original Algorithm

The first version of the algorithm was generational. At each generation, flies are created thanks to genetic operators then evaluated using the fitness function. The number of flies in the population is called $popu$ and at each generation the rate of fly created by mutation and crossover are called *mut* and *cross*. A generation of the fly algorithm can be described by algorithm 1 where fitness(f_1) is the fitness of the fly f_1, mutation(f_1) is the result of a mutation on the fly f_1 and crossover(f_2, f_1) is the fly resulting from the cross-over of the flies f_2 and f_1.

After a generation g the image is refreshed so the computation of the fitness function depends on the last image sent by the image sensor at the end of generation g-1.

Algorithm 1 Fly algortihm

```
for $i = 0$ to $popu \times mut$ do
    flies $f_1$ and $f_2$ randomly chosen
    if fitness($f_1$)<fitness($f_2$) then
        $f_1 \leftarrow mutation(f_2)$, computation of fitness($f_1$)
    else
        $f_2 \leftarrow mutation(f_1)$, computation of fitness($f_2$)
    end if
end for
for $i = 0$ to $popu \times cross$ do
    flies $f_1$ and $f_2$ randomly chosen
    if fitness($f_1$)<fitness($f_2$) then
        $f_1 \leftarrow crossover(f_1, f_2)$, computation of the fitness($f_1$)
    else
        $f_2 \leftarrow crossover(f_2, f_1)$, computation of the fitness($f_2$)
    end if
end for
```

2.4 Steady-State Version

The first step to adapt the fly algortihm to a CMOS image sensor is to create and evaluate flies dynamically. The notion of generation disappears and each time a fly is created, algorithm 2 is applied.

Algorithm 2 Fly algorithm

```
$i$ = a random number between 0 and $mut + cross$
if $i < mut$ then
    flies $f_1$ and $f_2$ randomly chosen
    if fitness($f_1$)<fitness($f_2$) then
        $f_1 \leftarrow mutation(f_2)$, computation of fitness($f_1$)
    else
        $f_2 \leftarrow mutation(f_1)$, computation of fitness($f_2$)
    end if
else
    flies $f_1$ and $f_2$ randomly chosen
    if fitness($f_1$)<fitness($f_2$) then
        $f_1 \leftarrow crossover(f_1, f_2)$, computation of fitness($f_1$)
    else
        $f_2 \leftarrow crossover(f_2, f_1)$, computation of fitness($f_2$)
    end if
end if
```

Fresh image data are now available at each evaluation rather than at each generation as in the previous version of the algorithm. The advantage is this allows to better exploit the fact image data are exploited quasi-continuously : each fly is evaluated with reference to more recently updated pixel values, enabling faster reactions to new events or new objects in the scene.

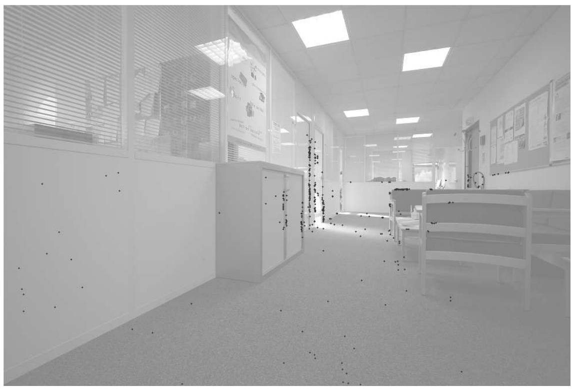
Figure 3: Corridor scene and flies resulting of one particular run of the Fly algorithm, projected on the corridor scene (printout contrast has been reduced in order to enhance the visibility of flies).

In order to compare the two versions of the Fly Algorithm, one hundred runs have been performed with the corridor scene shown on figure 3 with flies resulting of one particular run of the Fly algorithm. The original version of the Fly Algorithm runs until 200 generations and the steady-state version is running until having the same number of evaluations of the fitness function. The computation time required by the steady-state version is 9 percent less than for the original version of the Fly Algorithm.

3 CAPTURING IMAGES DIGITALLY

The delay for capturing an image is critical in the fly algorithm. There exists two main different technologies for capturing images digitally (Dal,) in which light is converted into electric charge and then into electronic signals:Charge coupled device (CCD) and Complementary metal oxide semiconductor (CMOS). For the former, every pixels charge is transferred through a very limited number of output nodes to be converted into voltage, buffered, and sent off-chip as an analog signal. All the pixels are devoted to light capture, and the output's uniformity is high allowing a good image quality. For the latter, each pixel has its own charge-to-voltage conversion, and the sensor often also includes amplifiers, noise-correction, and digitization circuits, so that the chip outputs digital bits. It results that a pixel could be checked in an image without checking the whole image.

CMOS image sensors have already been used with artificial vision algorithms. Chalimbaud and Berry (Chalimbaud and Berry, 2004) have implemented a template tracking in which this possibility allows to improve the perception and to focalise the system on areas of interest. Tajima et al. (Tajima et al., 2004) developed a prototype vision system maintaining conventional data transfer speeds using a CMOS image sensor. El Gamal (Gamal, 2002) presented developments which take advantage of the modifications of deep submicron CMOS processes. Larnaudie et al. (Larnaudie et al., 2004) have developed a CMOS imaging sensor for tracking applications. Our goal is to adapt the fly algorithm to optimize the use of CMOS image sensors.

4 THE FLY ALGORITHM FOR CMOS IMAGE SENSORS

4.1 Characteristics of the CMOS Image Sensors

The characteristics of the CMOS image sensors depend on the manufacturer and the model (Chalimbaud and Berry, 2004; Tajima et al., 2004; Gamal, 2002; Larnaudie et al., 2004). With some common CMOS image sensors, it is possible to send requests for the values of a single line of pixels instead of the whole image.

The time required to respond for a line L_n of pixels will be called t_0. The response time for a line close to line L_n is shorter than the time needed to respond to a random line. The response time to lines L_{n-1} and L_{n+1} is t_1 and the response time to lines L_{n-2} and L_{n+2} is t_2. The numbers t_0, t_1 and t_2 are such that $t_1 < t_2 < t_0$. The values of t_0, t_1 and t_2 depends on each CMOS image sensor.

The goal is to try to exploit this property in order to optimize the fly algorithm thanks to a new evolutionary engine.

4.2 Algorithm

In the fly algorithm, the fitness function of a fly is normally evaluated right after the creation of the fly. The main idea is to wait before evaluating a fly until there is a sufficient number of flies whose projections are in the same line. The flies which are waiting to be evaluated cannot be chosen by the evolutionary operators.

The new fly algorithm is based on a table T in which a dimension is the number of lines of the image called *line*. In this table, all the flies are stored before being evaluated. When enough flies waiting for evaluation are in the same line, all these flies are computed.

In order to determine when the fitness function of a fly has to be evaluated, thresholds T_0, T_1 and T_2 are used by algorithm 3 in which $time$ is a counter to know how much time the program has spent.

Algorithm 3 Creation of the fly F

```
$TimeStart \leftarrow CurrentTime$
$L \leftarrow$ Projection along x-axis of the fly $F$
**if** number of flies in $L < T_0$ **then**
    Storage of fly $F$ into line $L$
**else**
    $Time \leftarrow CurrentTime - TimeStart$
    computation of the fitness function of fly $F$
    $TimeStart \leftarrow CurrentTime$
    Processing line $L$
**end if**
$Time \leftarrow CurrentTime - TimeStart$
```

At each creation of a fly, the projection on left image of the fly F along the x-axis is computed. This projection is the same on both images if the cameras are parallel. If the number of flies in table T at the corresponding line L is lower than threshold T_0, then the fly F is stored into line L otherwise the fitness function of the fly F is computed and line L is processed. The processing of a line is described in algorithm 4.

Algorithm 4 Processing line L

```
$Time \leftarrow CurrentTime - TimeStart$
computation of the fitness functions of flies at line $L$
$TimeStart \leftarrow CurrentTime$
Dumping of the flies at line $L$ from table $T$
**if** number of flies in $L+1 > T_1$ **then**
    $L \leftarrow L+1$, Processing line $L$
**else**
    **if** number of flies in $L-1 > T_1$ **then**
        $L \leftarrow L-1$, Processing line $L$
    **else**
        **if** number of flies in $L+2 > T_2$ **then**
            $L \leftarrow L+2$, Processing line $L$
        **else**
            **if** number of flies in $L-2 > T_2$ **then**
                $L \leftarrow L-2$, Processing line $L$
            **end if**
        **end if**
    **end if**
**end if**
```

The processing of a line L is a recursive procedure which begins with the computation of the fitness functions of all the flies at line L and the dumping of these flies from table T. Then if the number of flies in table T at line L+1 is higher than threshold T_1, then line $L+1$ is processed; otherwise the number of flies in table T at line L-1 is compared to threshold T_1.

The numbers of flies in table T at lines $L+1$, $L-1$, $L+2$, $L-2$ are successively compared to thresholds T_1 and T_2. If a number of flies is higher than the threshold then the corresponding line is processed. This recursive process is the key to the success in the use of the property of CMOS image sensor.

5 ANALYSIS AND COMPARISON

5.1 Analysis

The values of thresholds T_0, T_1 and T_2 are a key point of the algorithm. If these thresholds are too high, flies in some parts of the scene could have to wait too long to be evaluated, and the algorithm would not react fast enough to new events in the scene. If these thresholds are too low, the characteristic of CMOS sensors are not exploited well enough. The thresholds T_0, T_1 and T_2 depend on the delays, t_0, t_1 and t_2 required to respond for different lines. t_0 is the response time for a line L_n, t_1 is the response time to lines L_{n-1} and L_{n+1} and t_2 is the response time to lines L_{n-2} and L_{n+2}.

The flies will be evaluated if there are more than T_0 flies for which the projection is on the same line. The response time will be t_0 so the average response time per fly will be $\frac{t_0}{T_0}$. For the same reason, if there are enough flies the projection of which is on lines L_{n-1} and L_{n+1}, the response time per fly will be $\frac{t_1}{T_1}$. If there are T_2 flies which the projection is on lines L_{n-2} and L_{n+2}, the response time per flies will be $\frac{t_2}{T_2}$.

The numbers t_0, t_1 and t_2 depend on each CMOS sensor and in order to analyse the fly algorithm adapted to CMOS sensor, T_1 and T_2 are chosen equal to $\frac{T_0}{3}$ and $\frac{2\times T_0}{3}$.

The next step is to study the number N_{wait} of flies which are waiting to be evaluated. Figure 4 shows the average variation of the number N_{wait} for different thresholds for one hundred runs for 200 generations of the fly algorithm on the corridor scene shown on figure 3. One can see the number of flies N_{wait} depending on threshold T_0.

The number of flies N_{wait} which are waiting to be evaluated are constant for given thresholds. One can see on the graphics that the higher the threshold, the higher the number of flies N_{wait}. These flies are not used by the algorithm because they cannot be chosen by the evolutionary operators.

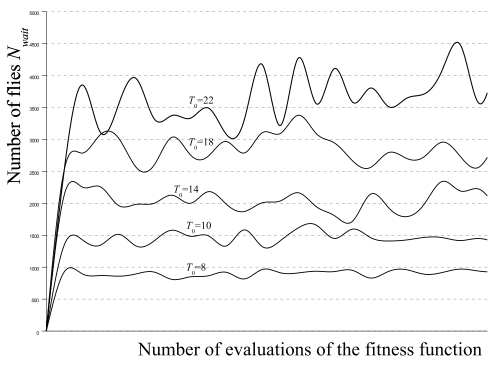

Figure 4: Variation of the number of flies which are waiting to be evaluated for different thresholds for 100 runs for 500000 evaluations of the fitness function of the fly algorithm on the corridor scene shown on 3.

5.2 Comparison

The results of the different versions of the algorithm are compared. The number of evaluations by a fitness function is counted and the time the program spends in algorithms 4 and 3 is known as the counter $time$.

For these two algorithms, let N_0 be the number of evaluations of a random line and N_1 and N_2 be the numbers of evaluations of lines spaced by respectively 1 and 2 from the line of the fly previously evaluated. The time taken by all the requests to the sensor is given by $N_0 \times t_0 + N_1 \times t_1 + N_2 \times t_2$. For algorithm 2, the time taken by all the requests to the sensor is given by $(N_0 + N_1 + N_2) \times t_0$.

The time the algorithm spends for the cross-over, the mutation and the evaluation of the fitness function is the same for both versions of the fly algorithm.

The two versions differ in the time of the requests to the sensor and the time spent in the two algorithms 4 and 3. Then, algorithms 4 and 3 are faster than algorithm 2 if $N_0 \times t_0 + N_1 \times t_1 + N_2 \times t_2 + time < (N_0 + N_1 + N_2) \times t_0$, So $time < N_1 \times (t_0 - t_1) + N_2 \times (t_0 - t_1)$. Up to our knowledge, the possible numeric values for t_0, t_1 and t_2 allow to verify this equation. The variable time depends on thresholds T_0, T_1 and T_2 as shown on figure 5. The integers N_1 and N_2 depends on thresholds T_0, T_1 and T_2 as shown on figure 6.

6 CONCLUSION

CCD and CMOS sensors are the two main types of sensors. CMOS sensors allow random access to a part of an image. We presented how the Fly Algorithm can be modified in order to exploit this property. As the internal delays in a CMOS camera are depending

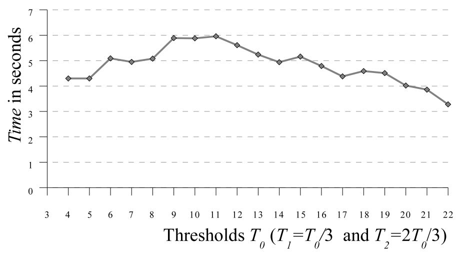

Figure 5: Variation of the variable *time* for different thresholds for 100 runs for 500000 evaluations of the fitness function of the fly algorithm on the corridor scene shown on figure 3.

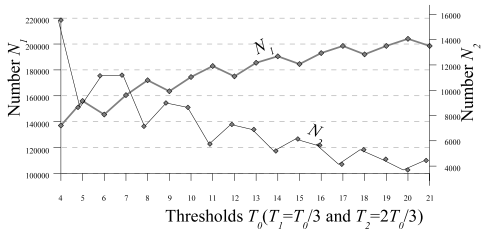

Figure 6: Variation of the numbers of evaluations N_1 and N_2 for different thresholds for 100 runs for 500000 evaluations of the fitness function of the fly algorithm on the corridor scene shown on figure 3.

on the order of pixel requests, we described a new evolutionary engine based a strategy to determine in which order the flies have to be evaluated to reduce the average reaction time of the algorithm.

The next step is to fix the parameters depending of the caracteristic of a given CMOS sensor. Future works could include study of using the CMOS image sensor to refresh the image in most relevant regions, depending on the scene. The improvement presented here could also be used to increase the quality of the fly algorithm to solve the problem of SLAM shown in (Louchet and Sapin, 2009).

REFERENCES

Bongard, J. and Lipson, H. (2005). Active coevolutionary learning of deterministic finite automata. *Journal of Machine Learning Research 6*, pages 1651–1678.

Boumaza, A. and Louchet, J. (2001). Using real-time parisian evolution in robotics. *EVOIASP2001. Lecture Notes in Computer Science*, 2037:288–297.

Boumaza, A. and Louchet, J. (2003). Mobile robot sensor fusion using flies. *S. Cagnoni et al. (Eds.), Evoworkshops 2003. Lecture Notes in Computer Science*, 2611:357–367.

Bucci, A. and Pollack, J. B. (2005). On identifying global optima in cooperative coevolution. *GECCO '05: Proceedings of the 2005 conference on Genetic and evolutionary,.*

Cagnoni, S., Lutton, E., and Olague, G. (2008). Genetic and evolutionary computation for image processing and analysis. *Genetic and Evolutionary Computation for Image Processing and Analysis*.

Chalimbaud, P. and Berry, F. (2004). Use of a cmos imager to design an active vision sensor. In *14me Congrs Francophone AFRIF-AFIA de Reconnaissance des Formes et Intelligence Artificielle*.

Collet, P., Lutton, E., Raynal, F., and Schoenauer, M. (2000). Polar ifs + parisian genetic programming = efficient ifs inverse problem solving. In *In Genetic Programming and Evolvable Machines Journal*, pages 339–361.

Gamal, A. E. (2002). Trends in cmos image sensor technology and design. *Minternational electron devices meeting*, pages 805–808.

Horn, B. H. (1986). Robot vision. *McGraw Hill*.

Jong, E. D., Stanley, K., and Wiegand., R. (2007). Genetic and evolutionary computation for image processing and analysis. *Introductory tutorial on coevolution, ECCO '07*.

Larnaudie, F., Guardiola, N., Saint-P, O., Vignon, B., Tulet, M., and Davancens, R. (2004). Development of a 750 × 750 pixels cmos imager sensor for tracking applications. *Proceedings of the 5th International Conference on Space Optics (ICSO 2004)*, pages 809–816.

Louchet, J. (2000). From hough to darwin : an individual evolutionary strategy applied to artificial vision. *Artificial Evolution 99. Lecture Notes in Computer Science*, 1829:145–161.

Louchet, J. (2001). Using an individual evolution strategy for stereovision. *Genetic Programming and Evolvable Machines*, 2(2).

Louchet, J. and Sapin, E. (2009). Flyes open a door to slam. *In EvoWorkshops. Lecture Notes in Computer Science*, 5484:385–394.

Ochoa, G., Lutton, E., and Burke, E. (2007). Cooperative royal road functions. In Evolution Artificielle, *Tours, France, October 29-31, 2007*.

Panait, L., Luke, S., and Harrison, J. F. (2006). Archive-based cooperative coevolutionary algorithms. *GECCO '06: Proceedings of the 8th annual conference on Genetic and evolutionary computation*.

Tajima, K., Numata, A., and Ishii, I. (2004). Development of a high-resolution, high-speed vision system using cmos image sensor technology enhanced by intelligent pixel selection technique. *Machine vision and its optomechatronic applications.*, 5603:215–224.

Wiegand, R. and Potter, M. (2006). Robustness in cooperative coevolution. *GECCO '06: Proceedings of the 8th annual conference on Genetic and evolutionary computation*, pages 215–224.