A biomimetic reactive navigation system using the optical flow for a rotary-wing UAV in urban environment.

Laurent Muratet , Stéphane Doncieux , Jean-Arcady Meyer

AnimatLab, LIP 6, University of Paris 6 - CNRS, France

Abstract— **Flying insects, such as houseflies or bees, have sensory-motor abilities that still outperform those of UAV. Beyond their flying manoeuvrability, they benefit from a reactive navigation system that enables them to wander in cluttered environments thanks to visual information. Their nervous system, relatively simple compared to that of other animals, exploits this information to efficiently extract the optical flow, which informs them about the distances of perceived objects. A navigation strategy that exploits the optical flow to avoid collisions with both lateral and frontal obstacles has been used here to control a realistic simulated rotary-wing UAV in a 3D urban textured environment.**

I. Introduction

Flying robots are specific platforms whose control raises new problems with respect to ground robots.
For instance, they can hardly call upon usual sensors to navigate in their environment because infra-red sensors are sensitive to external light and can detect nearby obstacles only; because sonar sensors are too slow or too heavy for small platforms; and because lasers are too dangerous to be used in urban environments. Not surprisingly in these conditions, there is currently no obvious solution to the problem of deciding with which sensor to equip a small UAV capable of flying in an urban environment without hitting obstacles.
In this paper, we draw inspiration from biology to assess the capacity of vision to tackle this problem. Indeed, visual images provided by a moving camera inform about the structure of the environment and notably about the distance of surrounding obstacles. This information can be extracted from the analysis of pixel movements in the image, i.e., by what is called the *optical flow*. Because an equivalent strategy has been exploited for millions of years by flying insects or birds, we have implemented it on a simulated helicopter moving in a realistic 3D environment to assess its adaptive value in these conditions.

Results to be shown here are based on a realistic physical model of a rotary-wing UAV, which is combined with a 3D-engine that generates images that are used by the obstacle-avoidance system. This system interacts with a low-level controller in charge of keeping the helicopter as stable as possible. The 3D environment is generated and monitored with Crystal Space[1], an open source software. A real-time correlation-based algorithm is used to compute the optical flow.

The paper successively describes the physical model of the UAV and its low-level controller, the optical flow algorithm, the navigation strategy and its biological background, the high-level controller for obstacle-avoidance. The results of several simulations realized in three different urban environments are then presented and discussed. Finally, possible improvements of the system are suggested.

II. The simulated UAV platform

❑ The physical model

The simulated robot is a rotary-wing UAV inspired from the Concept 60 SR II (Figure 2), a remote-controlled helicopter produced by the Kyosho Company. It has six degrees of freedom: three coordinates (x, y, z) and three attitude angles, i.e., the yaw (ψ), pitch (θ) and roll (φ). It weights 4.5 kg, with a length of 140 cm, a width of 15 cm and a height of 47 cm. Its main rotor has a diameter of 150 cm while that of the tail rotor is 26 cm. It is assumed to be able to carry the visual system described below.

Figure 1. The Concept 60 SR II.

[1] http://crystal.sourceforge.net

To simulate this engine, we used Autopilot[2], a physical simulator that has already been efficiently applied to such purpose[3].

❑ The robot's sensors and effectors

The navigation strategy implemented in this work called upon seven sensors: a video camera, two accelerometers, two gyros, a compass and an altimeter.
Data acquisition by the video camera is simulated using the realistic 3D environment generated by Crystal Space. This software outputs 25 images per second that make it possible to compute an optical flow that can be used by the high-level controller to avoid obstacles.
Inertial information is obtained from the attitude angles and from the speed of the UAV that are provided by the physical model. These values are corrupted by a Gaussian noise that simulates measurement errors.

❑ The robot's low-level controller

A helicopter is an unstable platform that must be controlled permanently. Therefore, besides using the optical flow to monitor horizontal displacements that would serve to avoid obstacles, the six degrees of freedom of the robot must be controlled in parallel. To this end, a low-level controller has been designed that makes possible a flight in straight lines, with a small lateral component (V_y) and at a constant altitude. This low-level controller capitalizes upon six PID blocks (Figure 2) whose inner coefficients were empirically determined during the experiments. More specifically, the main rotor collective block is used to keep the helicopter at a target altitude. The tail rotor collective block controls the yaw motion during turns and compensates the anti-torque generated by the main rotor in order to keep a target heading Ψ_{target}. The longitudinal attitude block regulates the pitch and, consequently, the forward motion V_x. The lateral attitude block controls the roll and the lateral motion V_y that the strategy described below needs to keep as low as possible.
θ, φ, V_x and V_y are obtained trough integration of gyro and accelerometer values. Z is obtained from the altimeter and Ψ from the compass.
This low-level controller efficiently keeps each of the DOF, but can't avoid small pitch, roll and heading oscillations. As they corrupt the computation of the optical flow, a simple linear filter that simulates a mechanical gyro stabilization system[4] has been used to eliminate these oscillations.

[2] http://autopilot.sourceforce.net
[3] http://autopilot.sourceforge.net/gallery.html
[4] http://www.ken-lab.com/stabilizers.html#KS2

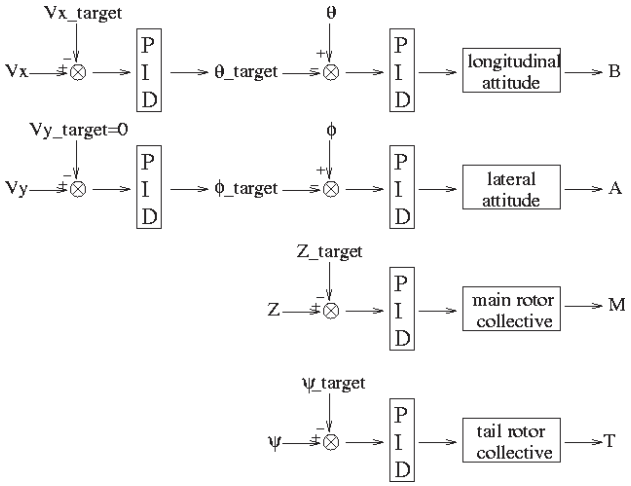

Figure 2. The low-level controller calling upon six PID modules. $Z_{-target}$ remains constant and $V_{y\text{-}target}$ remains null during an experiment. Ψ_{target} and $V_{x\text{-}target}$ are determined by the high-level obstacle-avoidance controller. The dynamics of the helicopter depend upon four output signals: B, A, M and T.

III. The optical flow

❑ Computational principle

The optical flow perceived by a camera is the velocity field created by the projection of the scene onto the image plan. Two main categories of algorithms may be used to compute this flow **[1]**. Those that are based on differential methods focus on accuracy. Those that are based on correlations are targeted at robustness and have been preferred in the present application.

Standard correlation-based methods try to determine, for each pixel (x,y), the v*v pixel-wide patch Pv centered on (x+u,y+w) in frame t that best matches the patch Pv centered on (x,y) in frame t-1. Matching patches are sought in a square region delimited by points of coordinates (x-n,y-n) and (x+n,y+n). For each possible match, a matching strength value M is computed from equation **(1)** that compares the pixel intensities I of the two patches. We use f, the absolute difference between their intensities, to compare two pixels.

For each pixel (x,y):

$$\forall (u,w) \in \{-n,\ldots,n\} \times \{-n,\ldots,n\} : M(x,y;u,w) = \sum_{(i,j)\in p_v} f\left(I_{t-1}(x+i,y+j) - I_t(x+u+i,y+w+j)\right) \quad \textbf{(1)}$$

Under these conditions, the actual motion of pixel (x,y) is (u,w), which corresponds to the displacement between

(x,y) and the centre of the patch that minimizes the matching function M (Figure 3). However, to determine this motion is computationally expensive and hardly applicable to real-time applications because of the quadratic dependency of the search upon n, the maximum motion detected: to find the best one, the algorithm must compute (2n+1)*(2n+1) matches. This is why we used a variant of this algorithm – the so-called *real-time quantized* version **[3]** - that was more compatible with our needs.

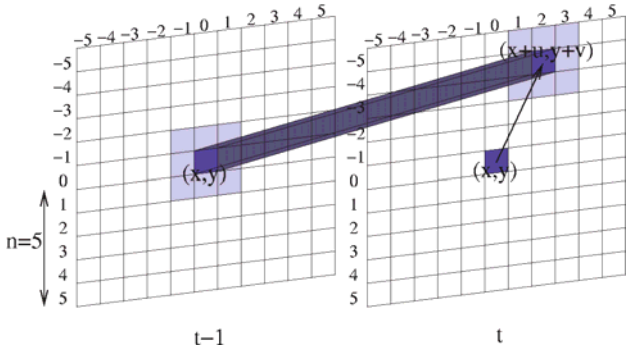

*Figure 3: Best match between two Pv patches of pixels in two consecutive frames. Accordingly, pixel (0,0) is assumed to have moved to position (2,-4) between frames t-1 and t. In this example, 3*3 pixel-wide patches, and a target region defined by a n value of 5, were used.*

- **Real-time application**

The real-time quantized algorithm is a correlation-based algorithm in which the search over space is replaced by a search over time. Using equation **(1)**, the matching strength is computed in a n=1 neighbourhood. Thus, the possible values for u and w are restricted to the range {-1,0,1}. For each pixel, a motion between the image at time t-1 and the image at time t is computed, as well as motions between images at t-2 and t, t-3 and t, and so on until t-S, S being an empirically-determined depth value.

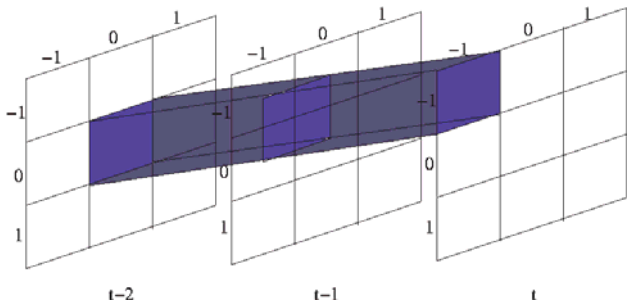

Figure 4: A motion of (-1,-1) between t-2 and t is equivalent to a motion of (-1/2,-1/2) between t-1 and t if we make the assumption of a constant motion.

As demonstrated on figure 4, with the assumption of constant motion, the actual motion of pixel (x,y) is (u/i,w/i), which corresponds to the displacement (u,w) between (x,y) in frame t-i and the centre of the patch in frame t that minimizes the matching function M. As a consequence, this variant evaluates pixel motions in the range 1/S to 1, while the original algorithm operates in the range 1 to n. Moreover, with n set to 1, the variant computes 3*3 matches per frame, i.e., 9*S matches for S frames. Therefore, this method is a trade-off between a quadratic search over space and a linear search over time. There is however a slight drawback associated with this computational gain, because the method doesn't afford detecting motions greater than one pixel per frame. Nevertheless, this limitation may be overcome by considering blocks of pixels instead of isolated ones. Pooling pixels by blocks of p*p multiplies the amplitude of detectable motions by p, and reduces the number of pixels considered by p*p. Hence, it improves both the range and the speed of the algorithm. Such improvement has not been used here.

- **Obstacle-avoidance**

Biological inspiration

In 1865, Von Helmholtz explained how some animals are able to evaluate the distances of lateral objects by using motion parallax. Since that time, numerous studies have investigated the corresponding mechanisms. In particular, Franceschini **[6]** described how the organization of the compound eye of the housefly, and how the neural processing of visual information obtained during the flight, allow this insect to compute its distances to lateral obstacles and to avoid them. This biological knowledge was exploited to implement a real opto-electronic device inspired by the visual system of the housefly on several terrestrial and aerial robots, thus allowing them to wander in their environments **[6]**, **[7]** and **[8]**. However, the corresponding control laws concerned one degree of freedom only, and the flying robots that were used were not entirely free of their movements. Moreover, the real-time computation of the optical flow calculation was made easy thanks to the specific environments that were used, in which dedicated white and black stripes helped to detect motion.

Likewise, other scientists have understood how other animals use the frontal optical flow to estimate the so-called *time-to-contact*, i.e., the time before a frontal collision is likely to occur. The gannet is supposed to use this information when it quickly dives to catch a fish and determines the exact moment when to fold its wings before entering into the water **[9]**.

The high-level obstacle-avoidance controller

The component of the optical flow that is required to avoid obstacles is generated by a forward translation of the observer. In this case, the horizontal velocity of an object is proportional to the inverse of its distance to the observer.
From figure 5, one may derive the following relationship: $\frac{x}{z} = \frac{X}{Z}$ which, after differentiating with respect to time leads to: $\frac{\dot{x}}{x} = -\frac{\dot{Z}}{Z}$.

Replacing $\dot{x}$ by Vx, $\dot{Z}$ by –Vo and Z by d*cos(β), it finally appears that the pixel velocity Vx is given by equation:

$$Vx = \frac{V_0 \times x}{d \times \cos(\beta)} \quad \textbf{(2)}$$

From this equation, we can deduce that a strategy that equalizes the perceived pixel velocities will tend to maintain equal the distances to obstacles on both sides of the helicopter. This strategy is called a *balance strategy* **[5]**.

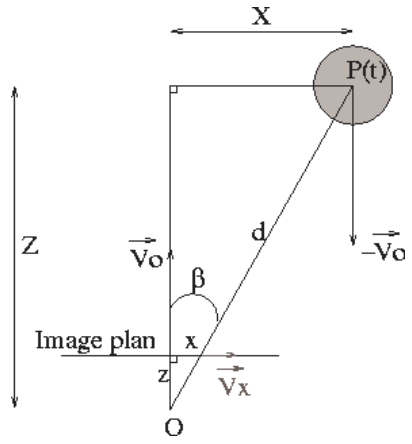

Figure 5. The moving UAV in point O perceives the circular object P(t) with a relative angle β. d is the distance between the observer and the object. Vo is the speed of the UAV.

To be efficient, this strategy must rely on the optical flow that is generated by forward translations only. Accordingly, the high-level controller that implements it must make decisions about the platform's trajectory only in situations where this platform is neither turning ($|\Psi - \Psi_{target}| < 1°$) nor skidding ($|V_y| < V_0 * 0.05$, $V_0 = \sqrt{V_x^2 + V_y^2}$). When the high-level controller has triggered a turn by determining an appropriate Ψ_{target} value in order to avoid an obstacle, skids occur for some time, even after the platform gets aligned on the right course. To minimize such skids and to limit the risk of collisions, a target forward speed of 0 is transmitted to the low-level controller whenever the helicopter needs to turn. An equivalent behaviour has been observed in the housefly whose flight trajectories are a combination of straight lines and tight turns, during which any decision based on the optical flow is inhibited **[6].**
Under these conditions, the control law that served to balance the optical flow on both sides of the helicopter was:

$$\Delta\psi\_\text{target} = k\left(Vx_{right}^{mean} - Vx_{left}^{mean}\right) \quad \textbf{(3)}$$

In this equation, Vx_{right}^{mean} and Vx_{left}^{mean} represent the means of horizontal motions of pixels on the left and on the right, k being a proportionality factor.

Moreover, as the optical flow algorithm can detect motions in a limited range only (from 1/S to 1 pixel), care must be taken to avoid pixel motions getting past these bounds. As the horizontal pixel velocity is proportional to the ratio V_0/d, the longitudinal speed V_0 must be adapted to the average distance to obstacles. The more cluttered this environment, the slower the robot must fly, and vice versa. This behaviour has been observed in bees, which adapt their velocity to the distance of perceived obstacles **[10].**
For these reasons, we compute a target speed V_{x_target} according to the following equation:

$$\Delta Vx\_\text{target} = OF_D - w \times \max\left(Vx_{right}^{mean}, Vx_{left}^{mean}\right) \quad \textbf{(4)}$$

where OF_D corresponds to the desired average optical flow (we empirically set OF_D to 0.06 in these experiments). In this equation, the maximum of the optical flows on both sides of the platform was used instead of an average value, thus avoiding premature accelerations in cases when near obstacles could be detected on one side only.

Finally, the high-level controller was endowed with a second reflex that served to avoid hitting a front wall, a situation in which the lateral optical flows are equal on both sides of the robot. This reflex called upon a rough estimate of the time-to-contact τ, according to which a U-turn order ($\Delta\Psi_{target}$ = 180°) was generated when the quantity of movement on both sides of the image was above a given threshold ε.

Thus, the high-level controller that was used in this work implemented a *subsumption architecture* **[2]** according to which the U-turn reflex had a higher priority level than the balance strategy.

IV. Experimental results

The controller just described has been put at work in three different environments. They are 100m-wide cubes, while the main rotor of the simulated helicopter has a length of 150 cm. They are closed by detectable walls thus making it possible for the helicopter to wander inside of them for a long time, if no crash occurs. Twenty experiments, lasting five minutes each, were performed in each environment. The starting point of each such run was a randomly-chosen position S, with no near frontal obstacle.

We used a simulated camera with a field of view of 90 degrees, with an extended-in-time optical flow algorithm using a S value of 10 steps and a 3x3 patch Pv. An input visual flow of 25 simulated images of 256x256 pixels per second was thus generated that served to control the helicopter (Figure 6).

Figure 6. The left image represents the simulated environment. The central image shows horizontal motions detected by the optical flow algorithm: black pixels are not moving, light-grey pixels are moving towards the left, dark-grey pixels are moving towards the right. The right image indicates the velocity: clearer pixels move faster.

Table I shows the results that have been obtained in each environment. Figures 7 to 9 show specific trajectories.

	Env 1	Env 2	Env 3	Env 3 with constant ambient light
Survival rate (%)	100	95	70	100
Average survival time (sec)	300	299.82	258.7	300
Standard deviation of survival time (sec)	0	0.79	79	0
Average velocity (ms)	0.64	0.56	0.47	0.45
Standard deviation of velocity (m/s)	0.06	0.03	0.04	0.04

Table I. Experimental results obtained in three different environments. 20 experiments, lasting 300 seconds each, were performed in each environment.

It thus appears that the helicopter never hit an obstacle in the first environment, but that its survival rate dropped to, respectively, 95% and 70% in the two others, more cluttered, environments.

To avoid the corresponding crashes, lowering the ε threshold implied in the τ reflex wouldn't be wise because it would prevent the helicopter from entering into narrow corridors and, thus, from exploring its environment.

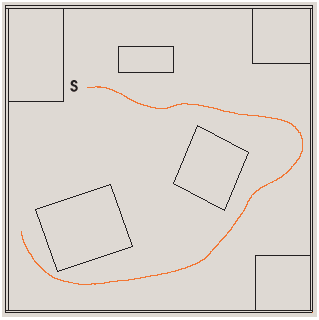

Figure 7: first environment including six obstacles which cover around 20% of the surface. S is the starting point of a specific trajectory among 20.

A closer inspection of all the situations in which crashes occurred revealed that these episodes all concerned dark-textured walls. They were due to the sensitivity of the algorithm to light conditions: not enough moving pixels being detected, the total optical flow was underestimated and the avoidance reflex was not triggered. This is why we performed a fourth series of 20 runs in Environment 3, with the same initial conditions that were previously used in this environment, but using a constant ambient light to eliminate dark places (Table I column 4). The corresponding survival rate raised to 100 %.

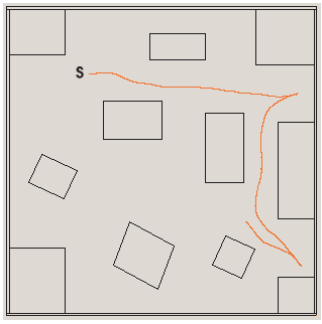

Figure 8: second environment including 11 obstacles which cover around 30% of the surface. S is the starting point of a specific trajectory among 20. The helicopter has made two U-turns to escape from dead-ends.

Concerning velocity control, the average speed of the helicopter decreased with the number of obstacles from 0.64 m/s in Environment 1 to 0.47 m/s in Environment 3. Although additional information would be necessary to confirm this conclusion, it seems that the low-level

controller succeeds to adapt the helicopter's velocity to its local environment, confining it to a range (from 1 to 1/S) where the optical flow extraction algorithm works correctly.

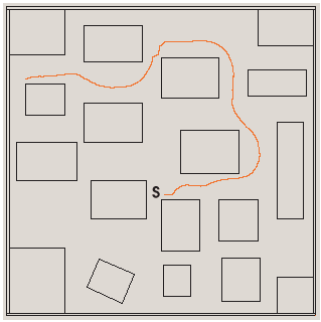

Figure 9: third environment including 18 obstacles which cover around 40% of the surface. S is the starting point of a specific trajectory among 20.

V. Discussion

Several improvements on the current simulations are conceivable at this stage of development. A first possibility would be to call upon evolutionary algorithms, like those we already applied to other flying platforms **[4]**, to generate low-level controllers likely to reduce blind periods when the UAV is turning or skidding. Still another possibility would be to improve the optical flow extraction algorithm, notably by considering blocks of pixels instead of isolated ones, so as to make the computation less time-consuming and to detect motions of greater amplitude. The impact of light variations on the algorithm should also be considered, as this effect seemed to be responsible for the observed collisions in the experiments described above.

The ultimate goal of this research effort being to implement the navigation strategy on a real helicopter, three major improvements seem necessary to bridge gaps between simulation and reality. In particular, a wind model must be added to the present system. This is an ambitious challenge because the dynamics of a wind blowing between buildings is very hard to simulate. Additionally, we have to assess whether the current simulation of the UAV's sensors is realistic enough. Likewise, the optical flow computation algorithm that has been used here must be checked on real images.

VI. Conclusion

The preliminary simulation results presented here are encouraging. The rotary-wing UAV is able to explore a cluttered environment without hitting obstacles, thanks to a two-level controller based on optical flow calculations that has been tested in unknown urban-like environments. This controller generates a cautious behaviour: the corresponding helicopter is preferentially following wide corridors and it tends to stay in their middle. At the same time, its longitudinal translation velocity is automatically adapted to obstacle density: the helicopter moves faster in free space and slows down in presence of obstacles. Moreover, when facing a frontal obstacle, the controller is able to generate a tight U-turn that ensures the UAV's survival.
Several improvements to the current simulated system are considered in the text that should help implementing it on a real platform.

REFERENCES

[1] Barron, J. L., Fleet, D. J., Beauchemin, S. S., (1994). System and experiment performance of optical flow techniques. *International journal of computer vision.* 12, 43-77.
[2] Brooks, R. A., (1991b). New approaches to robotics. *Science*. 253, 227-1232.
[3] Camus, T. (1997). Real-time quantized optical flow. *The journal of real-time imaging* (special issue on real-time motion analysis). 3, 71-86.
[4] Doncieux, S. and Meyer, J.-A. (2003). Evolving neural networks for the control of a lenticular blimp. In Raidl and al. (Eds). *Application of evolutionary computing.* Springer Verlag.
[5] Duchon, A. P. (1996). Maze navigation using optical flow. In Maes, Mataric, Meyer, Pollack and Wilson (Eds). *From animals to animats 4: Proceedings of the fourth international conference on simulation of adaptive behaviour*. The MIT Press.
[6] Franceschini, N., Pichon, J. M., and Blanes, C. (1992). From insect vision to robot vision. *Philosophical Transactions of the Royal Society of London. Series B*, 283-294.
[7] Franceschini, N. and Mura, F. (1994). Visual control of altitude and speed in a flying agent. In Cliff, Husbands, Meyer and Wilson (Eds). *From animals to animats 3*: *Proceedings of the third international conference on simulation of adaptive behaviour*. The MIT Press.
[8] Franceschini, N. and Netter, T. (1999). Neuromorphic optical flow sensing for nap-of-the-earth flight. *Mobile robots XIV*, SPIE Vol. 3838. Bellingham.
[9] Lee, D.N. (1980). The optic flow field: the foundation of vision. *Philosophical Transactions of The Royal Society of London*. Series B. 290, 169-179.
[10] Srinivasan, M.V. and al. (1996). Honeybee navigation en-route to the goal: visual flight control and odometry, *Journal of experimental biology, Special issue on navigation,* Vol. 199(1), pp. 237-244.