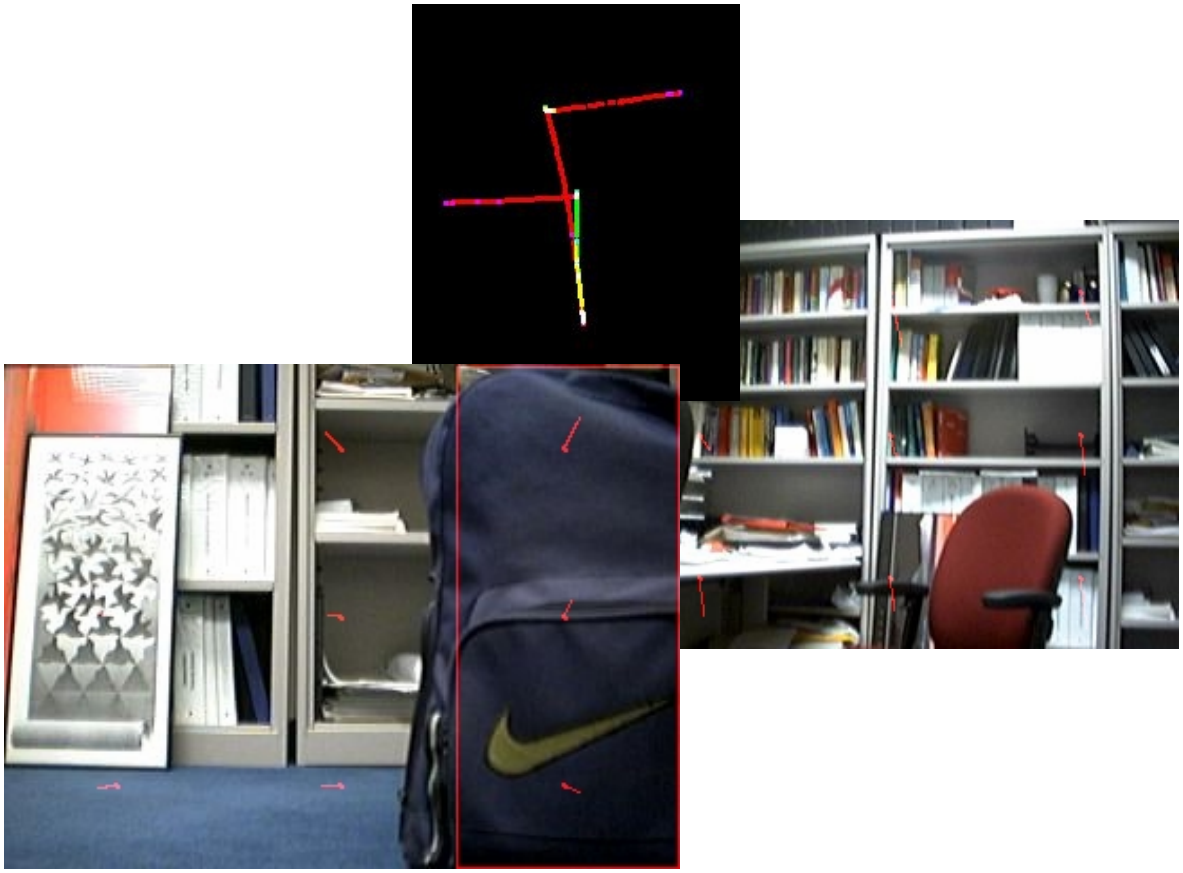
ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Carnegie Mellon
THE ROBOTICS INSTITUTE

Diploma work 2002-2003

Low overhead optical flow based robot navigation



Candidate: Dey Sebastien (MT-EPFL)
Professors: Prof. Illah Nourbakhsh (RI-CMU)
Prof. Raymond Clavel (IPR-LSRO-EPFL)
Assistants: Terrence Fong (IPR-LSRO-EPFL)
Sebastien Grange (IPR-LSRO-EPFL)
Charles Baur (IPR-LSRO-EPFL)

Index

This work has been realized in Pittsburgh at the Robotics Institute of the Carnegie Mellon University under supervision of Pr. Illah Nourbakhsh.
I would like to apologize for all misspellings and grammar errors in this paper.

1 Introduction

Vision is becoming more and more important in robotics. As the computation power grows and size reduces, the use of vision for robot navigation tends to generalize, even for hobbyists. Currently, there are three main topics in mobile robotics where vision is most used:

- robot navigation, 'open loop' navigation with only a visual feedback for avoiding obstacle and/or path planning/finding
- robot localization, finding the position of the robot in a metric or topologic map only on the base of visual feedback
- gesture/face/pattern recognition for giving orders to a mobile robot

This project belongs to the first category. Based on the fact that a lot of living things (in any case the high evolved ones) on earth mainly base themselves on their vision for their displacement in the real world (even insects do that), it is surely possible to develop a vision based system that would allow a mobile robot to navigate in real world using a visual feedback from a color camera. The current work takes place in a wider project that deals with the development of a new vision based robotics' sensor, whose aim would not be to replace all others sensors, but to provide a vision-based alternative for some of them, with the principal advantage to be able to combine multiple sources of information in an unique sensor: a camera. The long term ideas we would like to achieve are:

- vision based odometry
- vision based obstacle avoidance
- recognition of objects, places, landmarks,…

We also want to impose restrictions in terms of costs and computational power (in fact both are linked). We want to be able to use cheap cameras and low-cost boards so that the eventual good results we have could be used by any robotic enthusiast. The work described in this paper deals with the premises of this long term development. We choose to first focus on the use of optical flow to achieve movement recognition, guidance (e.g. use of only visual feedback to guide the robot on a straight course) and obstacle avoidance.

Optical flow is not a modern invention, it is one of the multiple methods used by some insects on the earth to gather information about their environment. Indeed, bees flying in a corridor (for example) equate the optical flow magnitudes on each side of their heading to stay approximately in the corridor's center ([1],[[2]), insects use optical flow to control their speed and movement in obstacle avoidance and human seem to use it too [3].

Most of the recent works on optical flow concentrate themselves on having the most accurate movement representation with a high resolution to be able to reconstruct geometric details of the real world. These works, when they succeed, have the disadvantage to need a lot of processing power and/or to be unusable in real-time application like robot navigation, thus, we need to find a method allowing us to use optical flow in a real time manner and to extract from it useful information for robot navigation. We don't want to focus on quantitative distance measurements and geometrical representations of the world, but simply take the qualitative measurements from optical flow and use this to determine a robot next action. Moreover, contrary to most of the existing works which are using intensity images, we are going to use all 3 rgb channels separately.

By definition, optical flow is the apparent motion of luminance patterns in the images. It is in fact a vector field giving us the presumed movement of the pixels in a scene by assuming that their characteristics stay constant over

time. Under not too restrictive assumptions, optical flow can be assimilated to the motion of objects in a scene or to the movement of a camera itself.

The first step of our current project will be, given our material restrictions, to find a method to extract in a qualitative way the movement of a camera in space on the only base of optical flow. Then we will use these results and the information collected to implement optical flow based guidance and motion recognition on a robot. To achieve these ideas, the material we use is:

- Phillips ToUcam USB camera and Pentium 3 450 MHz windows based laptop for the 'camera movement experiments'
- Phillips ToUcam USB camera, Advantech PCM3350 PC104 board (main cpu) [4], Custom-made board (motor control), and custom made robot chassis for the 'robot based experiments'.

This list complies with our need to have a low cost system, as the Philips ToUCam is a low-cost webcam, the pc104 board is one of the cheapest in its category, the motor control board is comparable to the boards which can be found on the hobbyist market (Cerebellum, handyboard,…) and the robot chassis contains no special mechanical parts (motors are modified servos).

For the continuation of this paper, section 2 will describe the existing work on the optical flow based robot navigation, section 3 will discuss the existing methods for calculating optical flow and present the one we use, section 4 will describe our workstation-based experiments (retrieving camera movement in space), their results and drawbacks, section 5 the use and implementation of these results on our robot and section 6 will give a conclusion and an exhaustive list of future applications.

2 Existing work

On the topic of optical flow computation, as this is a rather old subject, a lot of work has been done on finding methods for computing high definition and high reliability optical flow. Standard test sequences have been set for testing optical flow computing methods. However, our aim is not to invent a new method for computing optical flow, but rather to find out among all existing methods the one that could be usable in our project. Barron, Fleet and Beauchemin [5] have made a lot of work on evaluating and describing all existing techniques. P. Gollard has presented in her thesis [6] a technique using colors to estimate optical flow. The interesting point in this paper is not the technique itself, but the theoretical presentation of color perception and existing optical flow computation methods.

On the topic of optical flow based robot navigation, the existing works have almost always the same subjects as ours, but with the main difference to be done without having limits in terms of computational power. For example, S. Telizer from MIT [7] try to "develop algorithms that will be used for robust visual navigation of mobile autonomous agents. The approach is to efficiently compute and use optical flow fields to extract the features of the environment that are important for our purpose and to use this information as our guide for motion.". They currently have two programs available that have been designed to use optical flow information to 1) avoid obstacles 2) calculating time to contact with a surface. The main specificity of their work which are totally different form ours is the component types they are currently using (see fig. 2.1). Their robot is an industrial indoor robot, and the camera is a digital CCD one with integrated digital signal processing capabilities.

Antony A. Argyros proposed in [8] an approach for visual robot navigation based on the bees model (trying to have the same amplitude of optical flow on both sides of the heading) to achieve navigation in an indoor environment. His work describe a "new method for visual robot navigation based on the principles of purposive vision. Thus, the aim of this research is not what vision can offer towards building a general purpose world representation, but how the visual system of a robot can be designed in order to assist the robot in exhibiting particular behaviors", which is again dealing with the same idea as ours. The main idea is to use peripheral cameras to compute separately left and right side optical flow and then controlling the robot in a way that both values are identical. Their results are encouraging, their robot is able to navigate in a laboratory without hitting obstacles, but, as for the MIT project, they have no limits in terms of material, they use a multi-camera system (Fig. 2.1) and their robot is not fully autonomous.

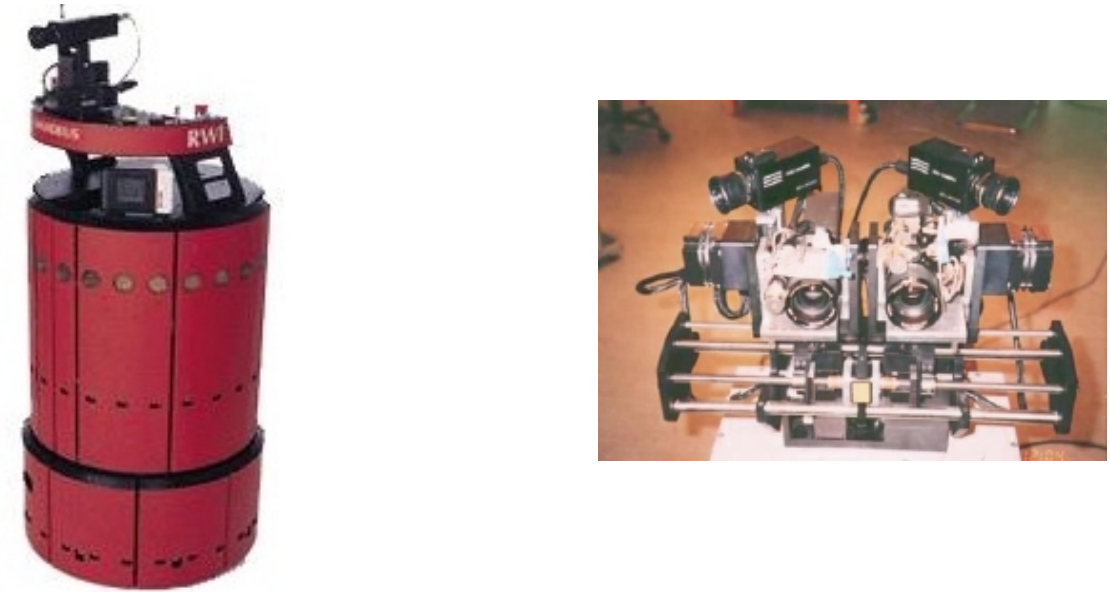


Fig.2.1: To the left, the robot used in the S. Telizer MIT project, to the right the camera head in Argyros approach

Andrew P. Duchon, in [9], made maze navigation using optical flow, again on the basis of the bees model, but used a simulated robot traveling in a virtual maze. The optical flow was not computed on the base of the images as could be seen by the robot, but by calculating it on the base of the velocity of the robot.

Our work, even if the subject is the same as the presented ones, has the aim to be low-cost and to need low computational power. We do have material limitations, so we have to find a special method to achieve the same sort of results as these works do but with our limitations. Moreover, contrary to the presented works, we want to use raw rgb images.

It is worth to mention that there exists another topic widely used in vision based navigation, the use of looming as a range sensor [10],[11]. This is an interesting, reliable and straightforward method allowing the ranging of obstacles by looking on the changes of their apparent size as the robot moves towards them. The problem is that it only works with high contrasted obstacles vs. background and need precise odometric information in order to work.

3 Optical flow

As explained in [6], for a correct use of optical flow, one need to understand some basic concepts such as motion field, image flow and optical flow. Motion field is a 3D field which describes the 3D motion of object in a sequence. The image flow is the 2D visible portion of motion field projected onto the image plane and the optical flow is an estimate of the image flow which gives us a 2D field of velocities associated to the same scene according to its variation of brightness patterns. More simply, it is the measure of the speed in which a point in one image moves to another point in a subsequent image.

Some of the work using optical flow try to reconstruct the motion field on the base of the optical flow as an approximation of the image flow, as it seems impossible to do the reconstruction directly from a measurement of the image flow without a priori knowledge on the motion field itself. Of course, these works are of no use for our current application as we aren't dealing with motion field reconstruction, but some of the observations made can be useful, for example to understand the drawbacks of optical flow.

For example, if we take a scene consisting of an uniformly painted ball rotating around its center, the image flow will be non-zero for every point of the ball while the optical flow will be uniformly zero as the brightness of the scene does not change. But if we take the same scene with a stationary ball but a moving light source, the results will be the exact opposite, non-zero for the optical flow and zero for the image flow. So, first point, optical flow is greatly dependant on the lightning conditions. Second point, optical flow need to be used on highly contrasted scenes to avoid seeing false zero movements.

In the continuation of this section, we are going to present the basic methods existing for computing optical flow, then we will present the one we chose.

3.1 Computing optical flow

All equations presented in the following chapters (3.1 – 3.2.2) come mainly from [5], [6] and [12]. Some shortcuts have been made as the mathematical part of the optical flow computation is not my first concern, but the reader interested in further details should read the given references

There are three main families for computing the optical flow, the first deals with **pattern matching**, the second are **frequency-based** methods which use energy/phase information in the output of velocity tuned filters and the third family is **gradient based** methods which compute image velocity from spatio-temporal intensity derivatives.

The pattern matching method is straight forward to understand. The frequency-based is far too much complex to be used in a small embedded system because we would have to deal with Fourier transform of our image. We would need to compute the FFT which would exceed our computational power restrictions. The gradient-based method is the most widely used and is based on the following assumption: for short period of time, a scene information stays constant in time. It means that, for example, we could follow a pixel in a scene just by looking at his brightness (constant-brightness assumption). This assumption needs some explanations, mathematically it is written as:

$$\forall \mathbf{x} \in E(t), \exists \mathbf{u}\ tq.\ E(\mathbf{x},t) = E(\mathbf{x} + \mathbf{u}\,\delta t, t + \delta t)$$

Where E(t) is a NxN intensity image taken at time t, E(**x**,t) is the brightness of pixel **x** at time t, and **u** its supposed velocity. By using a Taylor expansion and assuming an infinitesimal δt time interval [6], we come up with:

$$\frac{\partial E}{\partial x}u + \frac{\partial E}{\partial y}v + \frac{\partial E}{\partial t} = 0$$

where $\frac{\partial E}{\partial x}$, $\frac{\partial E}{\partial y}$ and $\frac{\partial E}{\partial t}$ are the partial derivatives of the brightness in space and time (will be referred as E_x, E_y and E_t in the continuation of this work) , and u,v the components of the velocity vector of the pixel. This equation is under-constrained. So, to be able to solve it, we need more constraints, which could be given by one of the following methods (taken from [5] and [6]):

- Smoothness constraint: This constraint is based on the assumption that the scene being imaged is relatively smooth and therefore that the optical flow itself is relatively smooth. Horn & Schunck, who are the first to make this assumption [5], used the square magnitude of the velocity field gradient to measure the field smoothness. So, they transformed the optical flow computation into an optimization problem of two criteria, the error in image brightness changes and the smoothness of the field:

$$\begin{cases} \theta_a = E_x u + E_y v + E_t \\ \theta_\beta^2 = \left(\frac{\partial v}{\partial x}\right)^2 + \left(\frac{\partial v}{\partial y}\right)^2 + \left(\frac{\partial v}{\partial x}\right)^2 + \left(\frac{\partial v}{\partial y}\right)^2 \end{cases}$$

  leading to the problem of minimizing the weighted sum of these quantities:

$$\theta^2 = \iint \left(\alpha^2 \theta_\beta^2 + \theta_\alpha^2\right) dx dy$$

  but without entering into too much details (interested reader should read [6]), this method needs a lot of computation power because it cannot be solved in a traditional way, but with an iterative method. Moreover, the implementation of this method requires a spatio-temporal Gaussian pre-filtering of the images [5].

- High order derivatives: This method add constraints by requiring that not only intensity values at (**x**,t) and at (**x**+**u**δt,t+δt) must be equal, but also their first orders' derivatives. So, we get three additional equations:

$$\begin{cases} E_{xx} u + E_{xy} v + E_{xt} = 0 \\ E_{yx} u + E_{yy} v + E_{yt} = 0 \\ E_{tx} u + E_{ty} v + E_{tt} = 0 \end{cases}$$

  These three equations plus the basic one form an overconstrained system which can be solved by the pseudo-inverse method (see [6]), but it has the drawback to need two times more calculation than the basic constant brightness assumption and to need pre-filtering of the images (temporal and spatial).

- Neighborhood-sampling approach: This is the most easy and straight-forward method to add constraints to our system, we simply make the assumption that the optical flow can be considered constant at all points of a small neighborhood. This way, if we take a neighborhood of nxn points around a given pixel, we will have n^2 equation for solving the optical flow of this neighborhood. Again, this over-constrained system will be solved by using a pseudo-inverse method.
  Of course, some people used a different model for the local behavior of the optical flow, and don't assume that the optical flow is constant over the small neighborhood, but take a linear variation model. But, as always, the more complex the model is, the more computation power you need.

- Multiple constraints approach: Here, the aim is to find *k* other functions than the brightness of an image which could be associated with the object surface and which do not change when the object move. With such *k* additional equations, one would be able to get a system of *k* linear equations equivalent to the major optical flow constraint. The high-order derivative is a kind of such approach, but uses the brightness of the image to add constraints, here, we want to find other functions.
  The most successful experiment (Mitchie et al, see [6]) using this approach used very simple functions like average (in 3x3 window), variance (idem), median (in 5x1 horizontal window),…

3.2 The current method

Actually, we found that the gradient based approach was the most suitable for our case, where we want to be able to deal with raw rgb images. Using a pattern matching technique would have been a solution, but it seems from [5] that matching techniques are less reliable than gradient based ones. In the gradient based methods, we need to choose a way to add constraints to our system and, at first, we found that two approaches where suitable: the neighborhood-sampling approach and the multiple constraint approach.

The neighborhood-sampling approach seems ideal because it is the method which consumes the less computation power. As we will see later in more details, given a neighborhood we only need to compute the first partial derivatives of the pixel values belonging to this neighborhood, and solve our optical flow equations by applying simple mathematical operations (no algebra, no matrixes, only simple operators) on sums of these derivatives. We do not really need any pre-filtering. So we can say that the number of operations required for computing optical flow using this method will be linearly dependent of the neighborhood size. However, this approach has the drawback that the assumption made about the constancy of the field is rarely true.

On the other hand, the multiple constraint approach would have allowed us to use more advanced color information directly, for example by setting the constraint equations as the hue and saturation values of the image (as one knows that the hue and saturation values of a pixel do not depend on the lightning conditions, or at least to the intensity variations in lightning), but it needs more computation power because we would have to compute the specific color values (HSI, Normalized RGB,…) of given pixels before applying the optical flow equations to them and also to pre-filter the images.

The choice we made was to first go on by using the neighborhood-sampling approach, and find a way to allow it to be used not only on the intensity (gray-level) image, but on all of the channels (r, g or b) of an image. In fact, we focus on trying to find an interest operator which could tell us on which pixel of a selected area of the image we should center our neighborhood, and in which channel of the image the optical flow should be computed, and then use the neighborhood-sampling approach on this channel and centered on the selected pixel. That way we mix the information contained in raw color images with a brightness-only based optical flow based method.

3.2.1 Neighborhood-sampling approach

The neighborhood-sampling approach makes the assumption that the optical flow is constant inside a nxn neighborhood of a scene, but we know that if an object is moving in a scene, this assumption will be totally wrong at its boarders, so why do we use this approach ? In fact, in most cases of a mobile robot navigating in a real world, the assumption is right. If we move a robot toward a corridor or in a room, it is true to say that the optical flow is piecewise constant, as in fact we will be looking at a sort of background which, from the point of view of the robot, is having a piecewise constant moving speed. The assumption fails only when the robot is navigating near an obstacle

in a large room. In such a case, the apparent speed of the obstacle will be different from the speed of the background, but as we will se it later, it can be useful for obstacle detection.

Concerning computing power, we saw that the computation of optical flow for a lot of points is far away our resources (450MHz PC processor), so we needed to find a solution to reduce the number of points where the optical flow would be computed while keeping in mind that our first step is to have a system being able to give us movement information of a camera moving in space. Thus, we decided to divide our basis images in 9 sub-images, each one of them giving us an optical flow vector. This choice is purely arbitrary but proved to be good. Within these 9 sub-

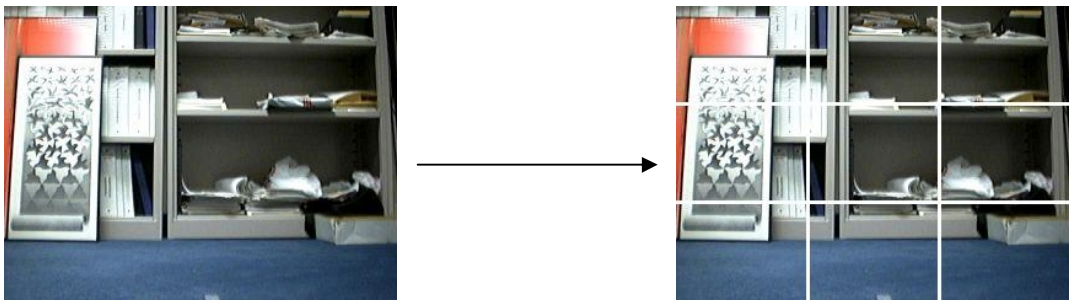


Fig. 3.1: To the right, we can see the nine sub-images where the nine optical flow values will be computed

images, we will search for our interest point and channel and, centered on this point, we will put our nxn neighborhood on which the neighborhood-sampling approach based optical flow will be computed

Now, concerning the computation of the optical flow itself, we know from §3.1 that with the neighborhood-sampling approach, we have an overconstrained equation system. In fact, for every pixel $\mathbf{X}_i$ within our *nxn* neighborhood N, we can write:

$$(\nabla E)^T \mathbf{v} + E_t = 0 \qquad \text{(a)}$$

where ∇E and E_t are computed at $\mathbf{X}_1$, $\mathbf{X}_2$,..., $\mathbf{X}_{n^2}$.Therefore, the optical flow can be estimated within N as the constant vector **v** that minimizes the functional $\Psi[\mathbf{v}]$ given by:

$$\Psi[\mathbf{v}] = \sum_{x_i \in N} \left[(\nabla E)^T \mathbf{v} + E_t \right]^2$$

This problem can be solved directly using a pseudo inverse method [6]. We first need to put together the basis equations (a) for all $\mathbf{X}_i$ within our *nxn* neighborhood N in a linear system form:

$$A\mathbf{v} = \mathbf{b},$$

where A is the n^2x2 matrix whose i-th row is $\nabla E(\mathbf{X}_i)$ and **b** the n^2 vector whose i-th row is $-E_t(\mathbf{X}_i)$. To solve this equation by using pseudo-inverse method (or least square solution), we first multiply each side of the equation by A^T:

$$A^T A\mathbf{v} = A^T\mathbf{b},$$

then, we can write:

$$\mathbf{v} = (A^T A)^{-1} A^T \mathbf{b}$$

which is the solution of our equation. By looking more carefully at the components of this final equation, we can see a few interesting points:

- A^TA is a 2x2 matrix, so its inverse could be computed easily.
- a closer look to the components of $(A^TA)^{-1}A^T\mathbf{b}$ shows us that $\mathbf{v}(u,v)$ can also be written as:

$$\begin{cases} u = \dfrac{\sum E_x E_y \sum E_y E_t - \sum E_x E_t \sum E_y^2}{\sum E_x^2 \sum E_y^2 - \sum E_x E_y \sum E_x E_y} \\ v = \dfrac{\sum E_x E_y \sum E_x E_t - \sum E_y E_t \sum E_x^2}{\sum E_x^2 \sum E_y^2 - \sum E_x E_y \sum E_x E_y} \end{cases} \quad (1)$$

which means that in fact we do not even need any matrix operation to solve the optical flow equations but only common mathematical operations on the partial derivatives, which is encouraging for the computing power needs.

The calculation of these derivatives is a crucial problem too, as the method used can make a large difference in the results. Horn [12] suggested to calculate the derivatives for points halfway between pixels in x, y and t directions, which lead to the three following formulas:

$$\begin{aligned} E_x(x,y,t) = {} & E(x+1,y+1,t+1) - E(x,y+1,t+1) + E(x+1,y,t+1) - E(x,y,t+1) + \\ & E(x+1,y+1,t) - E(x,y+1,t) + E(x+1,y,t) - E(x,y,t) \\ E_y(x,y,t) = {} & E(x+1,y+1,t+1) - E(x+1,y,t+1) + E(x,y+1,t+1) - E(x,y,t+1) + \\ & E(x+1,y+1,t) - E(x+1,y,t) + E(x,y+1,t) - E(x,y,t) \\ E_t(x,y,t) = {} & E(x+1,y+1,t+1) - E(x+1,y+1,t) + E(x,y+1,t+1) - E(x,y+1,t) + \\ & E(x+1,y,t+1) - E(x+1,y,t) + E(x,y,t+1) - E(x,y,t) \end{aligned} \quad (2)$$

3.2.2 Computing the optical flow vectors

We want to have very low computing power need for our optical flow algorithm, this way we have chosen to compute only 9 optical flow values and to use a gradient based approach with neighborhood-sampling hypothesis on raw rgb images. In the last section, we saw that our problem is reduced to common mathematical operations on the partial derivatives of the pixels belonging to the neighborhood patch. So, if we can reduce the size of the patch, we can improve the computation time.

Moreover, we need these 9 optical flow values to be as reliable as possible, so we need to find an interest operator that will, for every one of the 9 sub-images where optical flow will be computed, give us the center pixel of our *nxn* neighborhood (called the focus) and the channel (red, green or blue) in which the gradient-based optical flow computation would be done with the highest possible confidence. The reason for not using intensity (gray-level) images but raw rgb data, is that taking intensity image would bring a loss of information and less reliable results, as can be seen on fig. 3.2.

We are now going to describe the method used to find our focus and channel. We will give the example of what happens for one sub-image, knowing that the same process is repeated for every sub-images (see fig. 3.3):

1. we will apply our interest operators on all possible candidate positions of our focus within the sub-image
2. at all those positions, we will compute the operator score for all 3 channels separately
3. we will select our focus as the position among all candidate positions where the score was the best, which will also give us the band in which this best score has been computed

After some tests, we found that the best results were obtained by combining two interest operators, a gradient based one and a so-called weighted mask one. Using two operators will allow us to select among 3 classes of candidates (2 operators x 3 channels x all candidate positions = 6 possibilities per candidate position) in order to further improve the reliability of the optical flow vector. To reduce the computation time of our algorithm, we set the candidate focus positions as being ¼ of all possible positions on a so called search area (see fig. 3.3).

a. b. c.

Fig. 3.2: Here are the results of a 2D movement recognition used to compare the operators. We take the integrated mean optical flow of the 9 sub-images to compute a mean displacement which is reported as a path on output images (a. and c. above). The test sequences (b., from left to right and form top to bottom, shows every ten frames of our sequence) are first recorded, and then the algorithm is applied to the saved data. The test movement was made by hand and was composed of a vertical up-down movement followed by a left-right horizontal one and finally a down-up translation of approximately same amplitude as the first move. The a. image is the resulting path when using the basic intensity (gray-level) gradient based method, without any interest operator and the c. image is the resulting path of our actual algorithm, using color based interest operators. We immediately see the improvement when using the interest operators.

Let's describe the two operators:

- Gradient-based interest operator: This operator realize the following function:

$$\frac{E(x-1,y-1,t)-E(x+1,y-1,t)+E(x-1,y+1,t)-E(x+1,y+1,t)}{2}+ \\ \frac{E(x-1,y-1,t)-E(x-1,y+1,t)+E(x+1,y-1,t)-E(x+1,y+1,t)}{2} \tag{3}$$

  where E(x,y,t) is the scanned channel pixel value at x,y of image taken at time t. This operator is a 2D edge detector, it seeks for the biggest edge in the selected channel of the sub-image, and returns the edge value (the score of this operator), pixel position and channel. Taking the notation which is introduced in fig. 3.3, it could be translated in:

  ((A-B+C-D)+(A-C+B-D))/2.

- Weighted mask operator: This operator realize the following function:

$$\overline{E(t)}- \\ \left(\frac{E(x-1,y-1,t)+E(x+1,y-1,t)+E(x+1,y+1,t)+E(x-1,y+1,t)}{8}+ \right. \\ \left. \frac{E(x-1,y-1,t+1)+E(x+1,y-1,t+1)+E(x+1,y+1,t+1)+E(x-1,y+1,t+1)}{8}\right) \tag{4}$$

  where $\overline{E(t)}$ is the mean value of the selected channel over the search-area's pixels used as center of our interest operator. This operator is a sort of contrast detector and it seeks for a patch of bright contrast among the search area pixels of the selected channel and, as the first operator, returns the contrast value (again, its score), pixel position and channel. Taking the notation of fig. 3.3, it could be translated in:

  $\overline{E(t)}$ - ((A+B+C+D)+(A'+C'+B'+D'))/8, where the ' indicates pixels from image at time t+1.

As explained in fig 3.3 these operators are applied at every possible focus position within the search area of the selected sub-image. The pixel among the whole search area where the biggest score has been realized among all 6 possibilities (2 interest operators x 3 channels) will give us the focus position the channel used to compute our gradient-based/neighborhood-based optical flow. This process is repeated for every 9 sub-images.

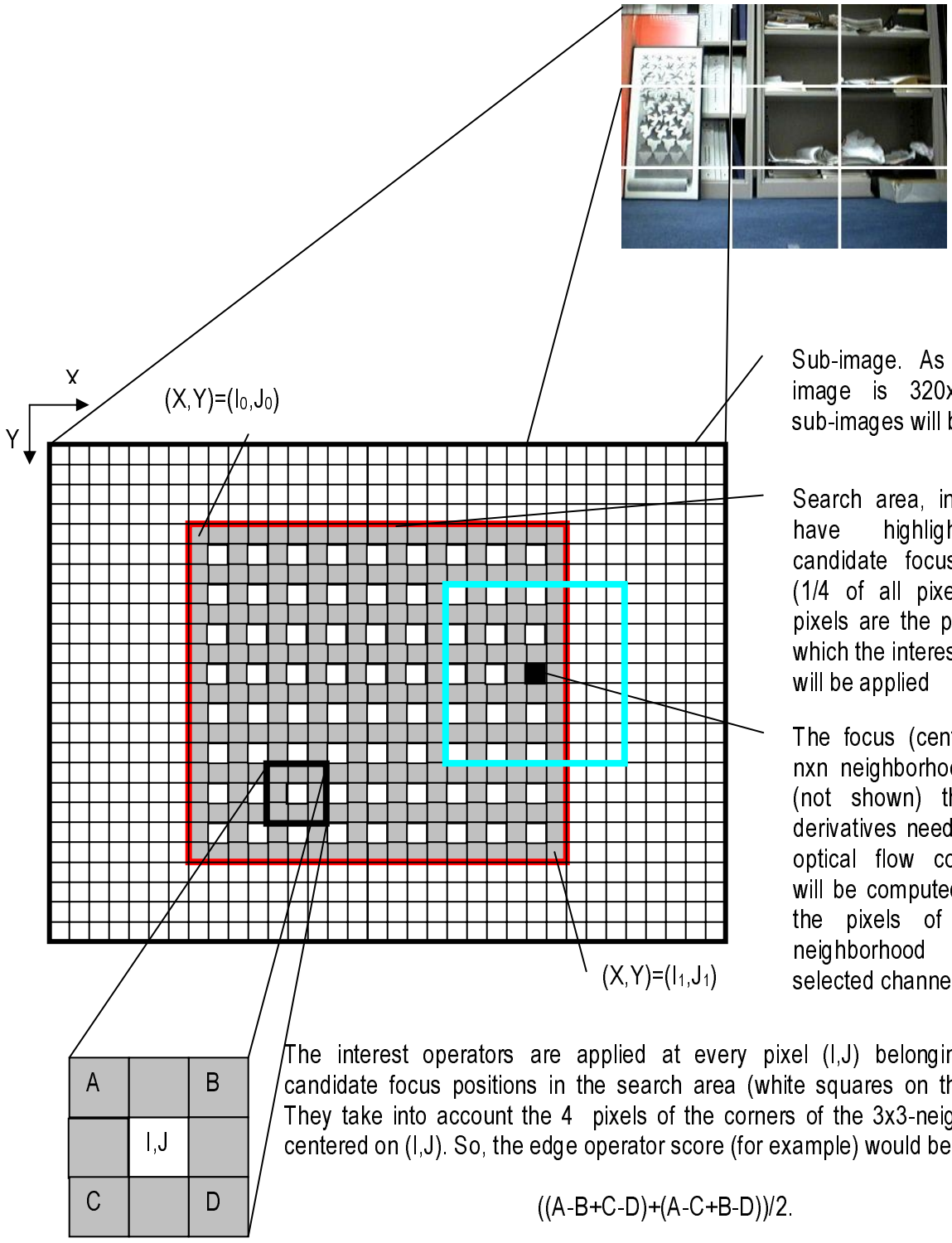


Fig. 3.3: Schematics of the whole interest-operator/optical flow computation process, given for one of the 9 sub-images

A pseudo-code example of what would be the research of the interest point over a search area of one of the 9 sub-images would be (see fig. 3.3 for notations):

(to simplify, let's assume that we know the mean value of each channel over the search area)

```
v = I0+1, w = J0+1, z = I1-1, t = J1-1

for(x=v ;x≤z ;x=x+2)
{
    for(y=w ;y≤t ;y=y+2)
    {
            Evaluate weighted mask operator and gradient-based operator at pixel (x,y) for all 3 channels,
            formulas (3) and (4)

            Compare with last highest value for each channel and each operator and keep current position and
            score if it's better than the last saved one

            \\ Which means that at the end of this loop over the search area, we will have 6 possible positions to
            \\ center our neighborhood ( 2 interest operators x 3 channels )
    }
}
return the pixel position (I,J) and channel whose overall score among the two operators was best
```

And, knowing the pixel position (x=I,y=J) which gives us the center of our neighborhood and the channel on which it should be computed, the pseudo code for the optical flow computation would be (reminder: the neighborhood size is nxn):

for(x= $\left(I-\left(\frac{n}{2}-1\right)\right)$;x≤ $\left(I+\left(\frac{n}{2}-1\right)\right)$;x=x+2)

{

    for(y= $\left(J-\left(\frac{n}{2}-1\right)\right)$;y≤ $\left(J+\left(\frac{n}{2}-1\right)\right)$;y=y+2)

    {

            Evaluate all partial derivatives at position (x,y) on the given channel, formula (2)

            Update the various sums needed for the computation of the optical flow, formula (1)

    }

}

return the flow vector, formula (1) (Pay attention to the possibility of 0 division)

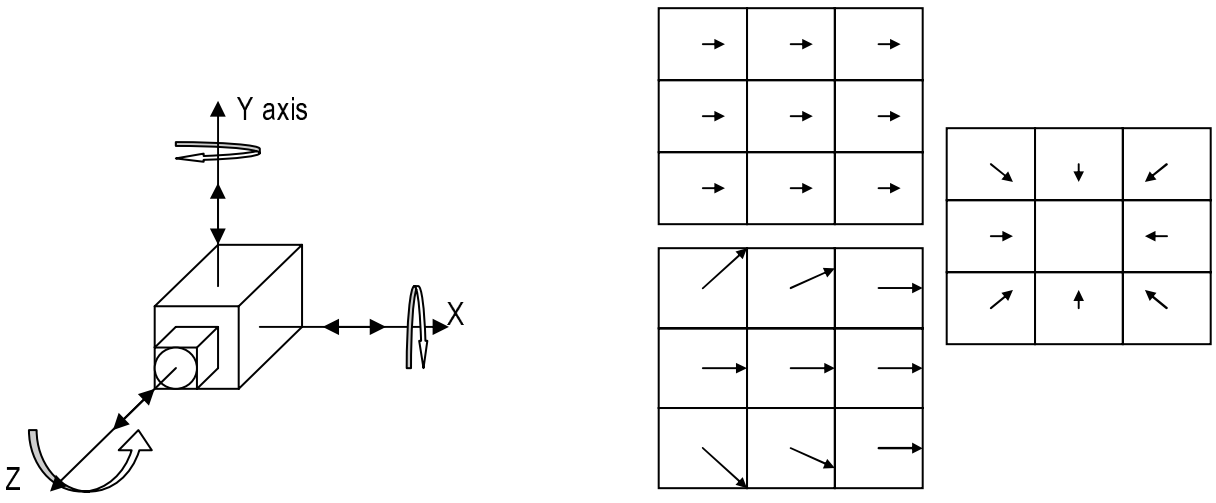
4 Wokstation based test and experiments

As a first step to an optical flow based mobile robot, we conducted some experiments on a workstation to see what kind of information could be extracted from our current optical configuration (reminder: optical flow is only calculated for 9 points over the whole image). Moreover, our first goal, was to use optical flow information to recover the movement of a camera in space

4.1 3D motion recovery

First of all, it seems that it is almost impossible to differentiate a translation among X or Y axis from a rotation around the complementary axis (see Fig.4.1a), but in the case of our mobile robot application, where we are going to use a differential drive ones, it is not a too big problem as such a robot should not have translation along X or Y axis, except as noise. Thus, there's no real need to be able to make the distinction between rotation and translation. So, our system will be limited to 4DOF motion extraction:

- detection and discrimination between Z translation and rotation
- detection of X,Y rotation or translation (which we will generically call X or Y translation)



a. b.

Fig. 4.1: a. The camera space of coordinates
b. Examples of ideal vector fields for basic movements, from up to down we have: X axis translation (left movement), Z axis translation (backward movement) and combination of X and Z translation (left and forward).

Secondly, we need to find a way to extract these movements from the 9 optical flow values given by our system. Figure 4.1b shows ideal cases of what our 9 optical flows should be for some typical movements. As, for the moment, we only want to be able to produce qualitative data, we can discretize the 4DOF in 20 basics movements:

- X, Y and Z tranlations, 6 movements
- Any combination of two translations, 12 movements
- Z rotation, 2 movements

Now the problem is to find a method to match our actual measures of optical flow with these basic movements. We will present here our solution which is based on the two following techniques:

- Bounding box method
- Idealized vector field comparison

4.1.1 The motion recovery algorithm

Movement recovery of a camera in space on the base of the optical information has to deal with the physical problems that displacements parallel to the optical axis (Z translation) gives optical flow vectors of a much lower magnitude than the same displacement within the same scene but along an axis perpendicular to the optical axis (X or Y translation). This problem is particularly important when dealing with a combination of parallel and perpendicular movement where it becomes difficult to retrieve the parallel part of the movement.

The first thing to do is to find some sort of filter that, before that the classification of movements begins, will be able to tell us if the current movement has a component parallel to the optical axis or if it is mainly a movement perpendicular to the optical axis, thus minimizing the set of possible movement we should be looking for. The idea is very simple, given a set of optical flow vectors, we take their bounding box and bounding box center of mass and, depending on the position (center of mass) and size of this bounding box (see fig. 4.2), we are able to make a first

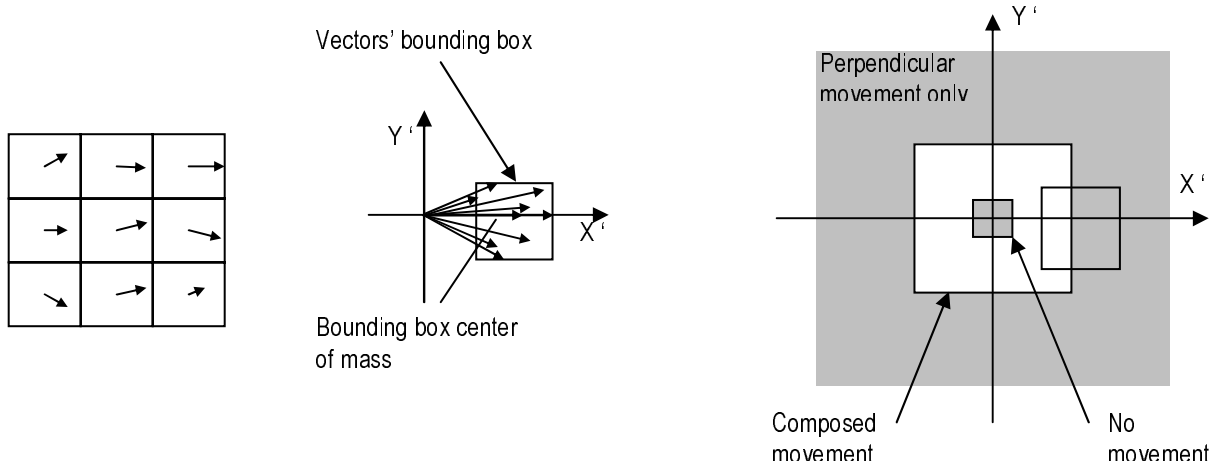


Fig. 4.2: The bounding box method for pre-filtering.

a. Typical real vector field for a left movement.

b. We take all vectors from a. whose variance is below the variance of the whole field (that way we exclude unreal vectors) and put them at the origin of a new 2D space (bounding box space, scale =2x). We take the bounding box of the vectors whose and its center of mass.

c. We now try to find to which area of the new 2D space these vectors belongs to (in the current example, they belong to the composed movements area) by looking at the size and position (center of mass) of this bounding box

classification between no-movement, combination of X/Y-Z movements or X/Y movement only. Our current pre-filter uses the following hypothesis:

1. If the bounding-box is small AND its center of gravity belongs to the 'no movement' area (Fig. 4.1 c.) of the X'Y' space → no movement
2. If the bounding-box is bigger than a certain threshold AND its center of gravity belongs to the 'composed movement' area (Fig. 4.1 c.) of the X'Y' space → high probability of Z movement BUT X-Y movement are possible too.
3. If the bounding-box center of gravity is away from the origin of the X'Y' space (belongs to the 'perpendicular movement only' area (Fig. 4.1 c.)) → X-Y movement only

Then, having idealized vector fields for each ones of the 20 possible 4DOF movements (see § 4.1), we try to match our current real vector field with the idealized ones. The comparison is made on the basis of angular error between actual vectors and the corresponding ideal ones. Moreover, the information given by the bounding box method allow us to put more weight on some movements' confidence (Zfactor), or to avoid computing some comparison. Thus, if we are in case 2., we can set a reduction factor (Zfactor) on the error values computed for Z movements so that Z movements will be more probable than X-Y ones, and if we are in case 3., we can avoid computing the comparison between the actual vector field and any of the ideal ones dealing with Z motion.

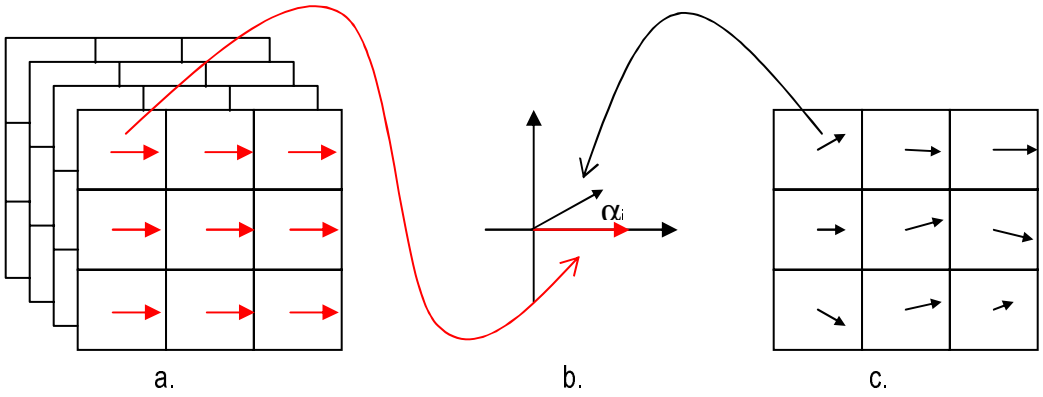


Fig. 4.3: Database comparison

a. The current ideal vector field being used as a basis of comparison.

b. The error calculation, the total confidence for a comparison is given by:

$$100 - \left(\frac{100}{\pi} \cdot \frac{\sum_{i<9}^{i=0} \alpha_i}{9} \right) \quad [\%]$$

c. The actual computed vector field.

4.2 Movement classification

Now, we have a method which is able, for each frame of a sequence (knowing image at time t and image at time t+1), to give us all 20 basic movements probabilities for the motion of the camera. We want now to extract in a reliable way the real movement, and we know by some tests that taking the basic movement which matched the best with the actual movement is not always the best choice. So, in order to avoid as much as possible false matches, we have the following strategy:

1. Instead of taken the current computed optical flow value for the 9 sub-images of the current frame, we take the average of the last K optical flow values of each of the 9 sub-images, knowing that the higher the K value is, the more 'inertia' we will have in movement transition
2. We filter the current averaged optical flow to make a pre-classification and detect an eventual 'no – movement', or reduce the number of necessary comparisons
3. After having computed all the matching confidences of our current optical flow vector field with the ideal fields, we divide them in eight 1D movement categories, we call this part the cross-confidence computation. The eight 1D categories are: Forward, backward, up, down, left, right, left or right rotation.
   (It means that, for example, a movement composed of X and Y translation (i.e. left and up) will add its confidence to the left category and to the up category)
   Then, we select the two categories which have the best overall score and compute the mean score of the remaining categories. After that, we eliminate from within the two best scores, the ones which are lower than the previously calculated mean plus a certain threshold (It allows to avoid the classification of random vectors fields where the comparison scores would be almost the same). We also eliminate the second best score if its value is too much lower than the first best score. The remaining result(s) are the matched movement(s).

To summarize the complete algorithm of 4DOF motion recognition, a diagram is given in fig. 4.4. Results are surprisingly good, overall >90% correct classification on real world tests.

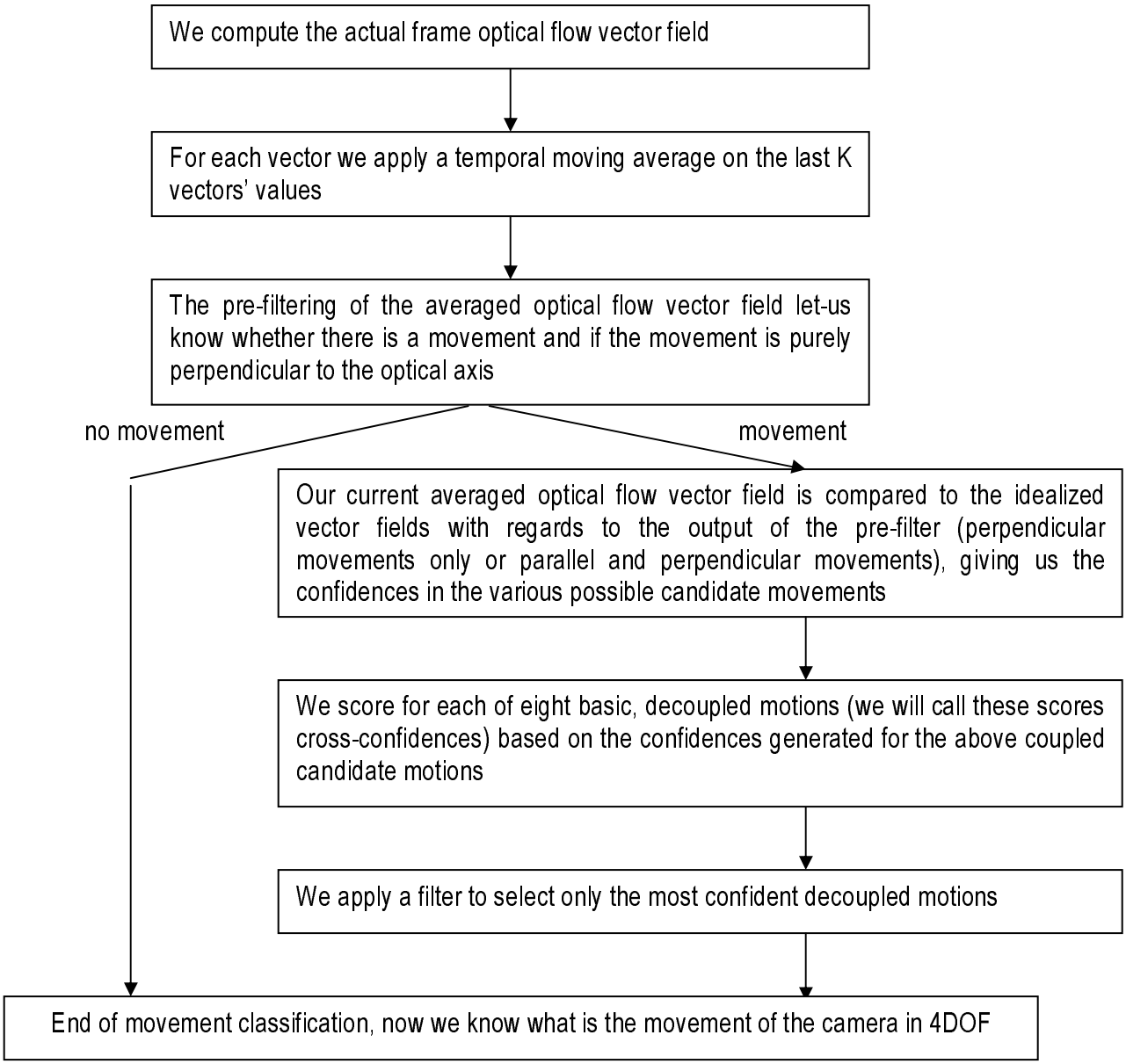


Fig. 4.4: Simplified block diagram of the movement classification algorithm

4.3 Obstacle detection

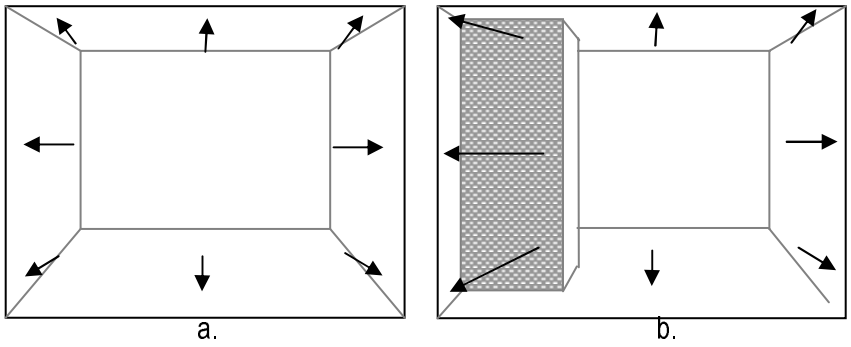


Fig. 4.5:Ideal cases of:
a. Moving forward without obstacles
b. Moving forward with an obstacle on the left side of the scene.

Having a reliable measure for the nine optical flows, we should be able to detect obstacles when the camera is moving along its Z axis. In fact, having an obstacle in the scene when going forward to it should theoretically lead in a bigger optical flow among the nine sub-images of the screen in which the obstacle appears (see Fig. 4.5).

So, by detecting in an appropriate way when it occurs that three side-most optical flow value of a frame have a magnitude really greater than the three opposite side ones, we should be able to detect obstacles.

4.4 Experiments

In this final section, we will present the experiment we have done with the presented algorithm. The aim was to move the camera either in space or as if it was fixed on a mobile robot, and see if it recorded the right movements and/or detected real obstacles. As for the interest operators comparison, the sequences are first recorded, then the algorithm is applied on the saved data. However, as the complete cycle frequency is ~20Hz, we can admit that the experiments have been run in real-time, the only reason for us to record the data before applying the algorithm is a practical reason for facilitating data extraction. The tests we realized are as follows:

1. **Handmade** test, moving the camera in office, X-Y plane. The theoretical movement is down, right, up, stop for a while and then left until near the starting point:
2. The camera being on a mobile platform, we move it **manually** forward into an office environment. We put an 'obstacle' on the right side of the path and try to detect it. The resulting movement should be almost straight forward.
3. The camera is on a mobile robot platform, and the robot is **manually** moved along offices hallways. The trajectory consist of a forward move followed by a left edge turn and a second forward move.

The movement speed for test 1 & 2 is ~ 5-8 in/sec. (10-20 cm/sec) and ~ 12 in/sec (30 cm/sec) for test 3, the average frame rate was ~20 Hz on the Pentium3 laptop (see Table 4.6)

Task	*Time [ms]*	*%*
Image loading	57.6	89.6
Algorithm *	6.45	10
Mov. Recognition	0.16	0.2
Others events	0.12	0.2
Total	64.3	100

**Complete Interest operator / Optical flow computation / Pre-filtering algorithm*

Table 4.6: Average timing values for our complete algorithm running on the 450MHz Pentium 3, after speed optimization (necessary for the pc104 porting, see §5).

Following are figures representatives of our tests. For figures 4.7 & 4.8, we have from top to bottom and left to right: some frames of the complete sequence, a graph showing us for a certain frame all 20 basics movements probabilities and to its right the mean integrated displacement computed on the base of the optical flow values. Fig. 4.9 shows the environment used in the office hallways test and the path followed. The following rules have been used for calculating the results:

- we do not take into account the movement classification errors when they occur in a transition of movement, that means that we wait for the first frame returning a correct movement classification for a given movement to start counting false classifications.
- for a Z movement to be classified correctly, it only needs that one of the two selected categories of movement is the Z one.

The overall results are the following :

- Test 1: 100% correct movement classification, for 87 frames within a total of 98 (9 starting frames used to have a correct mean value for our optical flow, 2 transitions errors).
- Test 2: ~91 % correct movement classification, for 84 frames within a total of 98 (13 starting frames). Concerning obstacle detection, we correctly detected the obstacle during 45% of the frames (counting frames from the moment when the obstacle is first detected to the moment when the obstacle isn't more visible). False detections occurred during 41% of the others frames. The percentage of correct obstacle detection is very low, but considering that it's enough to see once an obstacle to avoid it, it is usable. On the other hand, having 41% of the time a false obstacle detection is a problem. However, by carefully looking the frame by frame data, one can see that a false detection has a smaller duration (in term of consecutives frames) than a real one.
- Test 3: Although the obstacle detection is enabled in this test, the main experiment was to see the capability of the algorithm to correctly classify movement in a real robot case. The classification results are as follows: ~91% correct classification for the first part of movement (112 selected frames), ~96% for the left turn (44 selected frames) and ~92% for the second forward move (73 selected frames). The mean score is: ~92% (229 selected frames over 299 existing ones, the 70 unselected frames are either no movement frames when the robot was stopped between the moves or transition errors).

The following parameters were used:

- K, number of frames on which the optical flow vectors are averaged, equals 9
- Zfactor, reduction factor on the error values of Z movement confidence computation, equals 0.9

The overall correct classification of movements is better than 90%. These first workstation based results are really good considering the low computation power we are using. The obstacle detection capability is not so easily measurable and is more a practical result to be tested in real robot applications, but results obtained in test 3 are encouraging. Of course, the environment we chose for these experiments fulfill the requirements needed for optical flow, which means that the camera was always looking at contrasted enough images, and that light sources where stable (indoor lightning). If we make the same tests with a non-contrasted background, or with changing lightning conditions, they will completely fail…

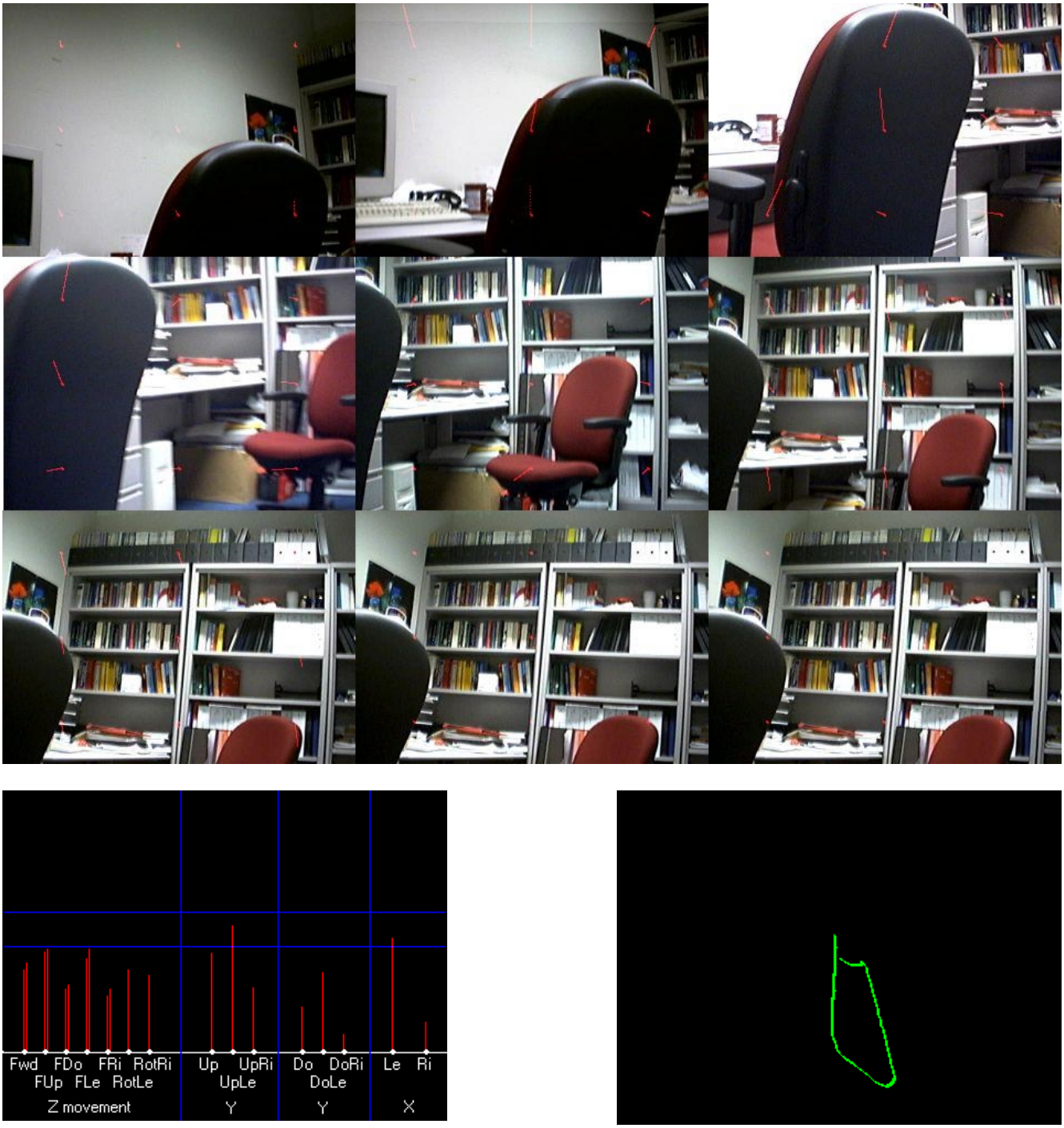


Fig.4.7: Test 1, X-Y movement in office environment.

***Upper images**: Every 10 frames of the sequence. Be attentive to the optical flow vectors on the frames. We can see that there are some errors in the field (e.g. frame 40, upper left corner, optical flow goes down instead of right) which are errors due to trying to compute optical flow on a flat painted surface.*

***Bottom left image**: Example of every 20 basic movements probability based on the last frame of the sequence. The reader immediately sees that the most probable movement should be a composition of up and left movement*

***Bottom right image**: Resulting path of the 2D motion, actually what we were looking for.*

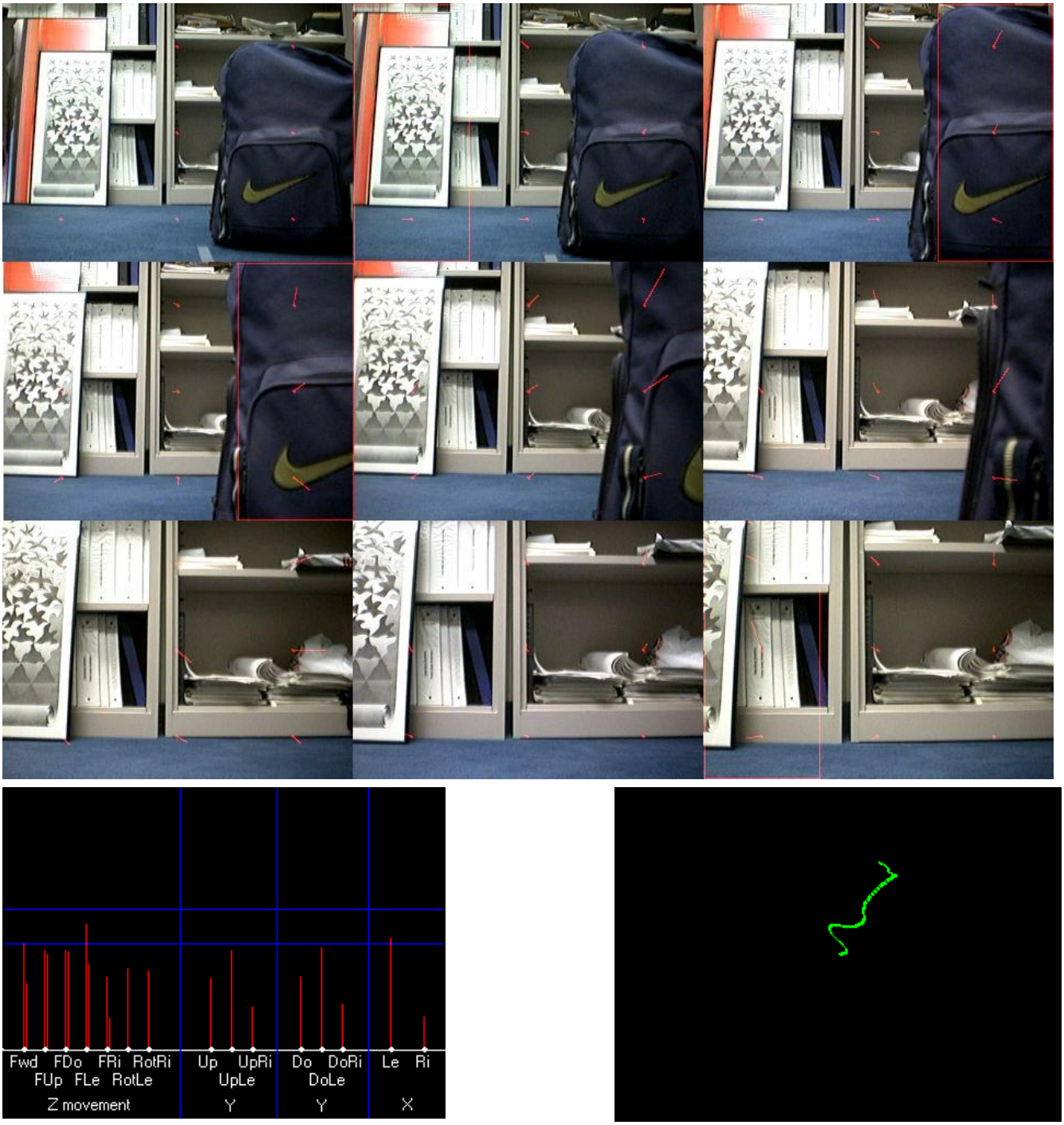


Fig.4.8: Test 1, forward movement in office environment .with obstacle

***Upper images**: Every 10 frames of the sequence. On frames 20, 30, 40 and 90 we can see a big square on one of the side of the screen, it is the signal that an obstacle has been detected in this frame. Detections in frames 20 an 90 are false ones whereas those of frames 30 and 40 are correct ones.*

***Bottom left image**: Example of every 20 basic movements probability based on the last frame of the sequence. The reader immediately sees that the most probable movement should be a composition of left and forward*

***Bottom right image**: Resulting path of the 2D motion, actually what we were looking for. Forward movement is seen as a n up movement and we have a little drift to the left*

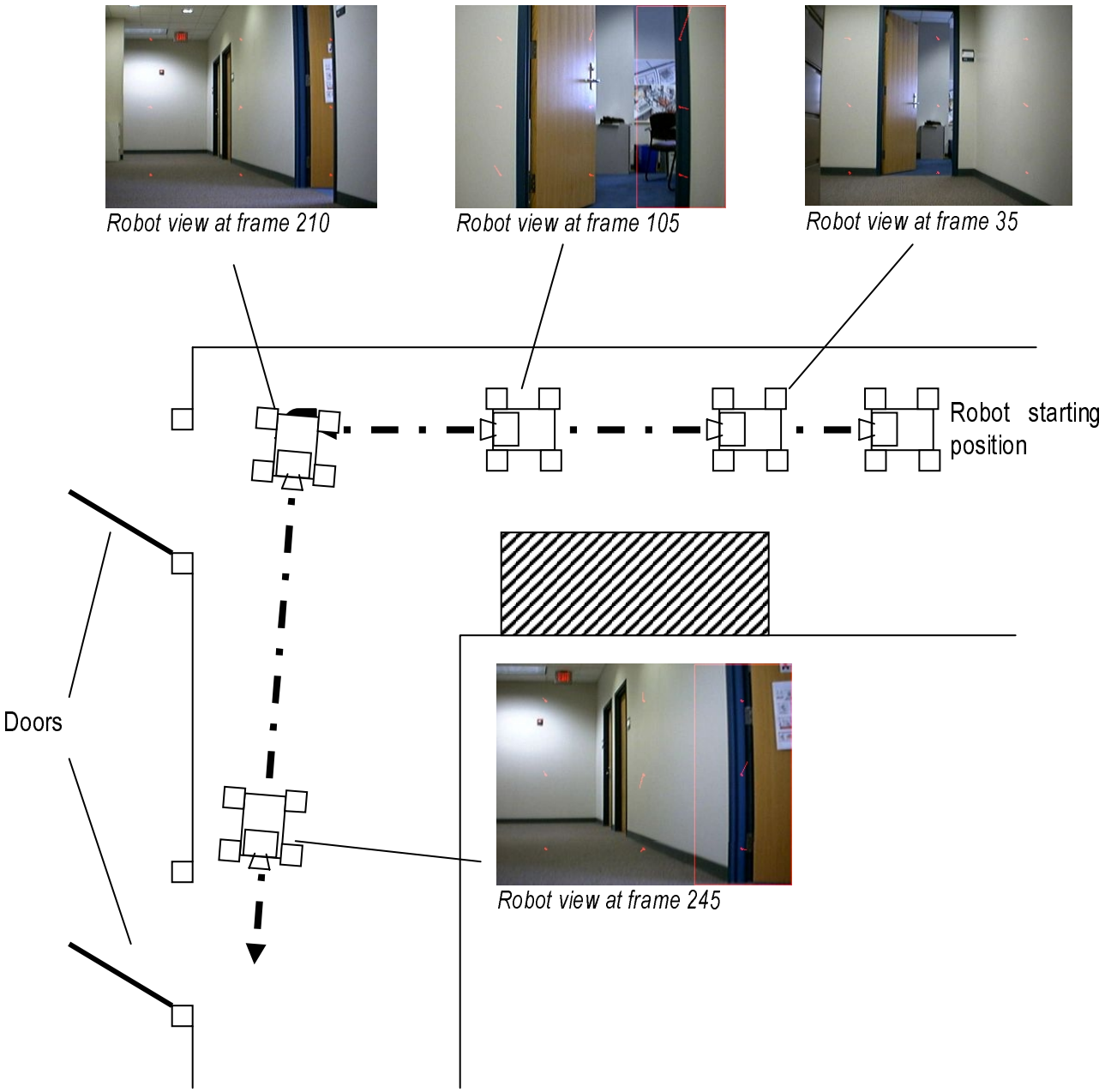


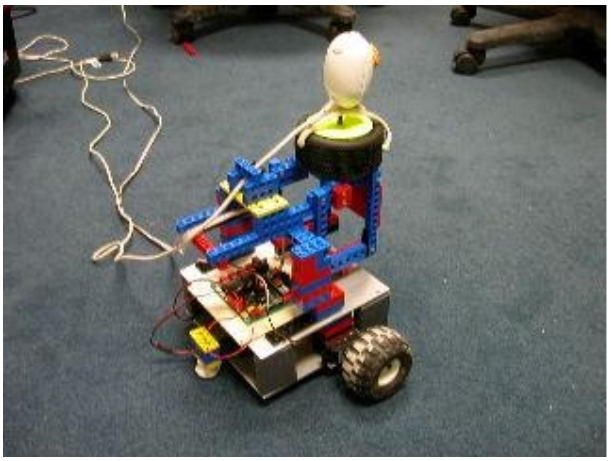
Fig.4.9: Test 3, corridor hallway in office environment. We have a schematic representation of the path followed by the robot with at some positions the image taken by the camera and corresponding frame number. At point A, the robot stops, turns left, stops again and go forward.
It is interesting to see, by reading the frame by frame report, that obstacle detection could have been useful during this move. During the first forward movement, the robot detects obstacle on his right side (wall) until it's near the door where it detects the left part of the door as an obstacle. Then, on the second forward move, the robot almost always detect an obstacle on its right side (wall).

5 Robot based experiments

In the following chapters, we will first describe the robot, then summarize the whole process of porting our laptop-application to a pc-104 based one and finally present the various experiments and results used to validate our algorithm.

5.1 The robot

The robot is a custom-made differential drive one. The motors are modified servos, the motor controller is based on a home-made Pic card which communicates to the robots-brain (laptop or workstation) via a RS-232 serial link. Any differential drive robot could have been used, as long as it's big and powerful enough to carry the pc-104 board, the webcam and the power source needed.



a.

b.

Fig.5.1 : The home-made differential-drive robot (a), and the Nomad Scout used in the 'master and slave' experiment (b). On both is mounted the ToUCam webcam used for our experiments

The motor command board used is a simple multi-purpose pic-based board which has been reprogrammed in order to communicate with a PC to receive commands for the motors. Due to the small motors drivers (capable of driving only 12V @ 1A), we had to use small motors, so we chose to use the most simplistic way of having a motor with gearbox by using modified servos.

5.2 Porting

As it has already been said, the principal aim of the big project in which this works is included is to achieve a low-cost vision based sensor. But we also want our system as small and lightweight as possible, which means that we don't want to use a laptop or any workstation as the 'brain' of our robot, but rather a card-size one. As a first approach, we chose a low-cost pc-104 board, the advantech PCM 3350 (Fig. 5.2), which is based on a fan-less 300MHz Geode processor. Further specifications can be found in [4].

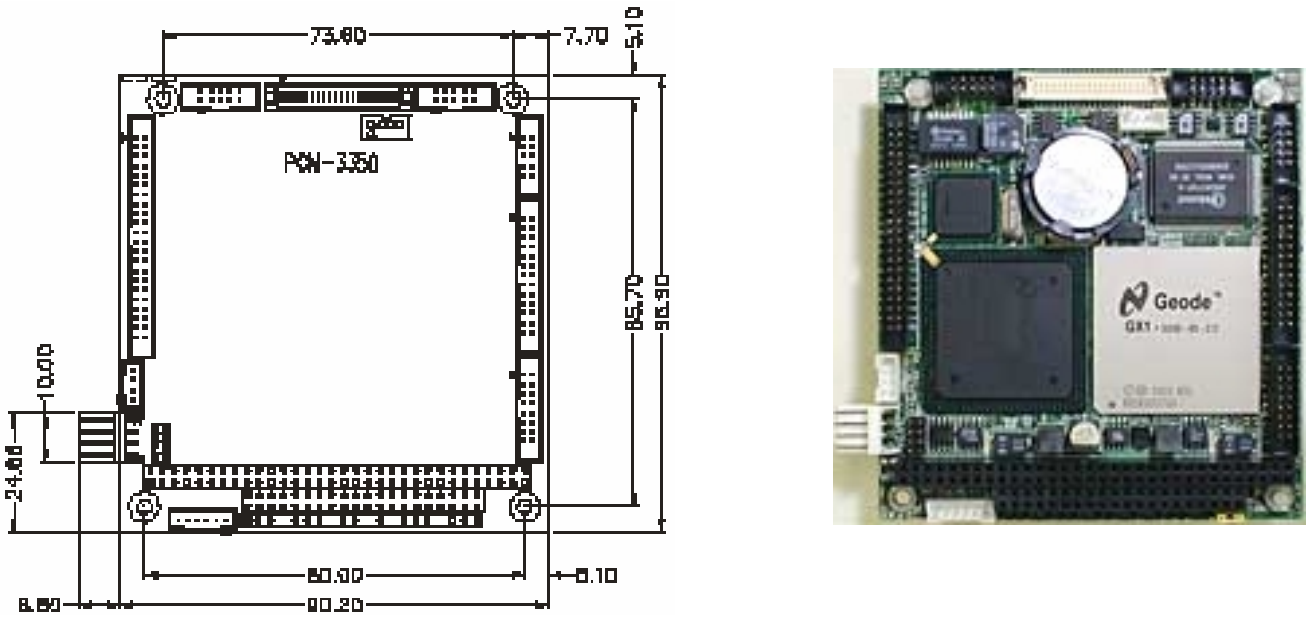


Fig.5.2: The currently used PCM 3350 pc104 board.

The main step in the porting of the application is to optimize the code so that the application can be launched in real-time on the pc-104. In fact, it appeared that after optimization our complete algorithm takes ~106 ms to run when loading previously saved images sequences from the HD. But, when trying to get real-time images from the camera, the whole program slows-down to a cycle time of ~240 ms. (see Table. 5.3). For the moment, this is an

Task	*PC-104 timings [ms]*			*Laptop timings [ms]*	
	(A)	*(B)*	*(C)*	*(B)*	*(C)*
Image loading/grabbing	92.9	92.4	176	57.6	29.8
Algorithm *	47.9	8.9	28	6.45	10.4
Mov. Recognition	1	1.1	1.2	0.16	0.63
Others events	0.03	0.025	33.6	0.12	14.9
Total	141.8	102.4	238.8	64.3	55.7

*Complete interest-operator/optical-flow/pre-filter algorithm

Table 5.3: Average cycle time measurements [ms]. We used the following settings:

(A) Offline, images loaded from HD, 150 frames long, before optimization
(B) Same as (A), after optimization
(C) Online, no-movement, 100 images grabbed by the camera

unresolved problem as normally grabbing images from the camera should take less time than loading them from the HD since the camera is able to run at more than 20HZ when using the camera-proprietary programs. Moreover, the increase in time on the pc-104 for the algorithm itself when grabbing real-time images should not be so big (3 times vs 1.5 times for the laptop). It appears that it is in fact a hardware problem coming from the implementation of USB on pc-104. Thus, for the moment, real-time robot based experiments have only been done with the laptop as 'brain', although theoretically same results would have been achieved on the pc-104 if the image grabbing time was faster.

5.3 Experiments

We have seen in chapter 4 that our algorithm is capable to output good qualitative information when dealing with handmade 4 DOF movements. Now, we want to validate these results in the real robot-world, where there are vibrations due to motors or wheels, shocks and bumps due to the terrain and/or the robot movements.

The first important change from the previous experiments is the reduction of the number of DOF we should be able to retrieve. Indeed, the possible movement of a mobile robot will be restricted to a plane, moreover, with a differential drive robot, we will only have 2 DOF (1 translation, 1 rotation). However, one must consider all possible orientations for the camera. Thus, the first choice to make concerns the camera itself:

Should it be pointing forward or to the ground or ceiling ? Would it be better to use several cameras or 'special' cameras (panoramic camera, wide angle,…) ?

Answering the second question is easy. As we said that we will use cheap cameras, we will use our webcam 'as is', without any modifications. The question of using one or more cameras can be discussed, but we chose to try with only one camera for the moment. For the question of the orientation, in ideal cases, we would not point forward as we know that it is difficult to correctly retrieve movements parallel to the optical axis. However, after some tests, it appeared that pointing to the ground was not reliable at all as the ground was either non-textured or its texture pattern was too small to be correctly used by our algorithm. Pointing to the ceiling sometimes brings the same problems but the major drawback is that the lights bring overexposed and unusable images. So, we conducted all the tests with the camera pointing forward.

The following experiments are not intended to bring a lot more data about our optical flow qualitative output, but rather to confirm the results we've had in chapter 4 but with a real robot, and restricted DOF. The first experiment is more a demonstration of the quality of the qualitative information that we are able to receive from our algorithm, while the second one deals with an eventual direct application of our algorithm. We try to guide a robot on a straight line only on the base of visual feedback, without any odometric information. These experiments will also show that in the current configuration, see § 6 for more details, our algorithm is not very useful for quantitative or geometric robot navigation, but opens a lot of interesting applications based on qualitative navigation which could be based on our algorithm.

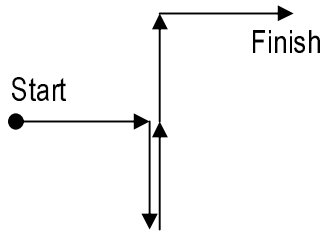
5.3.1 Master and slave

In this experiment the camera is mounted on a master robot (in our case a Nomad Scout, fig. 5.1b). The master robot is then moved around and our robot, called 'slave' robot, has to make the same movements as the master robot, only on the base of the visual information given by the camera mounted on it. We make the assumption that the master robot only makes decoupled movements, i.e. only fwd/bkwd translation or left/right rotation. However, on the 'slave' robot, we allow coupled movements to be processed, which means that we take into account the drifts that could occur when trying to move the master robot in straight motion (or in the opposite way, the false drifts seen by our algorithm).

The following parameters were used:

- K, number of frames on which the optical flow vectors are averaged, equals 7
- Zfactor, reduction factor on the error values of Z movement confidence computation, equals 0.65

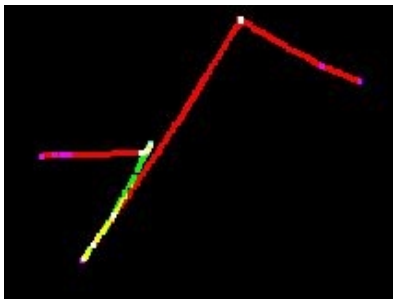
Of course, as we are not able to get reliable information about the magnitude of the displacements, the movements of our 'slave' robot are often different from those of the master robot (different speed and/or magnitude), but the important thing to see is that the algorithm (as we saw in § 4.4) is capable to output reliable qualitative information about the movement of the camera.

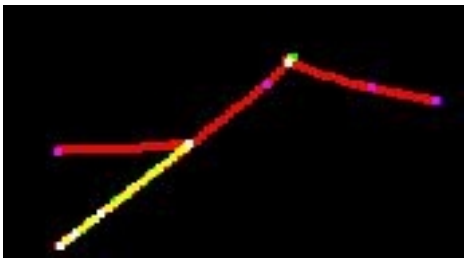


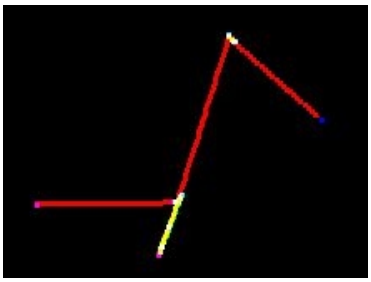
a.

Environement	*Number of frames of the master's movement*	*Noisy frames (unable to decide movement)*	*Transition errors*	*False movement frames*	*Correct frames*
(c.)	*306(100%)*	*12 (4%)*	*12 (4%)*	*25 (8%)*	*257 (84%)*
(d.)	*195 (100%)*	*1 (0.5%)*	*12 (6%)*	*10 (5%)*	*172 (88.5%)*
(e.)	*290 (100%)*	*22 (8%)*	*6 (2%)*	*14 (5%)*	*248 (85%)*
Average [%]	*/*	*4.2%*	*4%*	*6%*	*85.8%*

b.







c. *d.* *e.*

Fig. 5.4: The master & slave experiment:

a. The test movement is made by: fwd mvt / left turn / bkwd mvt / double fwd mvt / right turn / fwd mvt (all turns are 90°, all single fwd/bkwd movement are ~80 cm long)

b. Table reporting the movement classification results

c.d.e. The approximate real path followed by the slave robot (red= fwd translation, green = bkwd translation)

We set a test movement (see fig. 5.4 a.) to be done in a few environments and we just looked at the output movement of the slave robot. The path drawn on fig 5.4 c&d are an approximation of what was the path of the slave robot during the experiment. We base ourselves on the commands sent to the slave's motors to have an idea of what the movement of this robot should be. These paths have no other values than showing that the slave movement shape is approximately the same as the master one and moreover to show that efficient quantitative measurement cannot be achieved with the current algorithm.

The first and second test (Fig. 5.4 c. and d.) have been made in the same environment and on the same course (environment comparable to the one of the straight motion experiment Table 5.6 (B) & (C)), but in the first one the movement speed was 20 cm/s for translations and ~45°/s for rotations whereas the speed was almost doubled for the second test. The third test has been conducted in a completely different office-like environment (same as Table 5.6 (A)) and the speed of the master robot was approximately the same as in the first experiment. The main difference between the first experiments and the third one was that the environment was much more 'small'. In experiments 1 & 2 the mean distance from the camera to an obstacle was always > 1 meter, whereas in the third experiment this distance was always < 1 meter. The main problem with such a small distance between the camera and the obstacles (background) is that sometimes the algorithm is not able to find enough interesting points to compute optical flow vectors, leading in noisy outputs (see error description below).

To have a measurement of how good our slave robot follows the master's movements, we take into account the number of false movements the slave does, which can be divided in three categories:

1. The false movement classification, where the algorithm outputs the wrong movement. We don't care when, in forward or backward translation, a left or right component is coupled with a forward or backward one but, when a forward or backward component is present during a supposed left or right rotation, this is counted as an error.
2. The transition errors, when the algorithm needs some frames to pass from a movement to another.
3. The classifications problems. When the optical flow vector field is 'random', the algorithm is not confident enough in any of the possibilities of movement, and its output is 'noisy movement'.

We also graphically compare the path of the slave to the real path (see Fig. 5.4 c. d. e.).

From the graphical comparison, we can say that the slave robot movement shape is similar to the master's, but it is not surprising as we knew from §4.4 that qualitative measurement were reliable. From the study of the classification errors, we can see that:

1. False movement classification seems to occur randomly
2. Transition errors mainly occur when rotating. The reason is that the moving average used to smooth movements brings a certain inertia to pass from a 'no movement' to a 'left' or 'right movement' only. Indeed there are always some frames where the algorithm outputs ' left and forward' or 'backward and left' instead of the real 'left' only movement. These false classifications come mainly from the fact that we are making a moving temporal averaging of the optical flow vectors. Thus, when the robot stops (for example), the vectors don't fall immediately to 0 amplitude but rather diminish in size during a number of frame corresponding to the moving average size. And during this time interval, all movement classification will be false (in the best case, they are in continuation with the movement preceding the stop so we don't realize that there's false classification).This problem is mainly present with left or right rotational movements as their magnitude, and inertia, is bigger.
3. Classification problems are some time transitions problems, but are mainly randomly distributed, except in the third experiment where we had more noisy outputs due to difficulties for the algorithm to find interest points.

The speed difference between experiments 1 & 2 doesn't bring any more errors, it is rather the opposite. But, by carefully looking at the step-by-step movement classification, we made the following observations:

- for translation parallel to the optical axis, we see that the quicker the movement is, the more reliable the movement classification is as the magnitude of the optical flow vectors will be bigger.
- when the robot rotates too quickly, it can occur that the optical flow vectors are too big and not any more reliable (the pixel displacement exceeds the size of the neighborhood patch).

Now, if we try to characterize the quantitative measurement quality of our current algorithm (even if it is not the first aim of this experiment), we will first discuss about the first two experiments:

- problems mainly occur when rotating where it is difficult to have a correct estimation of the magnitude of the movement. We can see in Fig. 5.4 c. and d. that the first rotation is always underestimated whereas the second one appears to be quite well reproduced. This is mainly due to the poor repeatability of our algorithm which is directly related to the background on which the camera is pointing. Between the two edge turns of our test movement, the background differs and during the first rotation, the optical flow computation is less better computed leading to errors in estimating its magnitude.
- on the other hand, translations seem to be rather accurate in term of relative distance reproduction. The distances traveled by the robot between the different tests are not the same, but inside a test, they are quite comparable(every single translation should have the same magnitude).

About the third experiment, we can take it as a proof that our algorithm, in its current configuration, is not able to output quantitative information which could really be used alone as odometry measurements. But, if we were trying to place the movement on a known map of the robot environment, knowing only qualitative data would be enough to have a general positioning of the robot.

5.3.2 Visual feedback for straight motion

In this experiment, we wanted to test the possibility of using optical flow based visual feedback to guide a robot. The goal is to have our robot running in a straight line only on the basis of this visual feedback. Given an initial command of supposed straight motion (if the motors, gears, wheels,... were perfect, giving the same commands to both motors would lead in a perfect straight motion), the robot should detect any drift to the left or the right by looking at the optical flow and correct it by an appropriate opposed steering. The important issue in such a visual feedback device is that it is completely insensible to wheel slippage and other environment dependent odometry problems. To characterize the movements, measurements of drift and heading errors are made (see Fig 5.5).

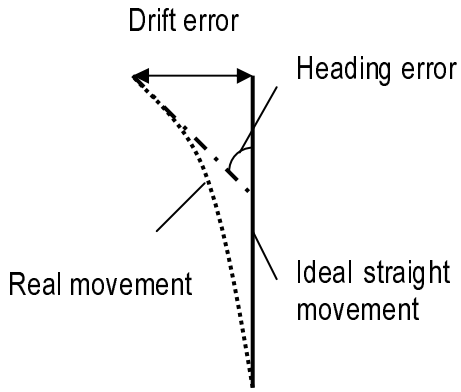


Fig.5.5:Drift and heading error measurement used to characterize the reliability of using visual feedback to guide a robot

The experiment has been conducted in diverse environments, we always start by running the robot in open loop mode (i.e we give him the initial straight motion command, but it runs open loop) and then we make the same run multiple times with visual feedback and different program settings. Each time we measure drift and heading error. As one can see (Table 5.6), visual feedback allows to achieve better results than open loop control, and coupled with obstacle avoidance, would be sufficient to guide a robot, as long as precision is not wanted. The lack of precision is due to the detection and measurement of drift. The repeatability of the measures being very low, we are not able to create a good controller for the robot. The actual controller is only a sort of integrative controller: as long as there's a detected drift in any direction, the controller keeps on slowing the speed of the opposite wheel (when the robot drifts to the left, we slow down the speed of the right wheel to correct it) without taking into account the magnitude of the drift. Thus, our robot is never going straight in a stable way, but has a zigzag sort of motion, with the amplitude (duration) of the zigzag being directly related to the size of the moving average used to smooth the optical flow vectors. Decreasing the size of the moving average has no real overall effect on the magnitude of the errors, but visually led to a more 'nervous' motion.

Environement description	*Type of measure*	*Drift error [cm] and direction*	*Heading error [degrees]*
Office environement, camera pointing at a bookshelf 2 meters away (A)	Open loop	21 (to the left)	17.5
	Mean values for corrected motion	4.25 (absolute)	4.55
Indoor environment, camera pointing at chairs and tables, background flat (B)	Open loop	6 (to the right)	5.7
	Mean value for corrected motion	3.3 (absolute)	4.6
Indoor environment, camera pointing at a distant background (C)	Open loop	7.5 (to the left)	7.2
	Mean value for corrected motion	3.3 (absolute)	3

(A)

(B)

(C)

Table 5.6 Environment and measurement for the guidance experiment

In order to improve the measurement of drifts and, in a more general way, of displacements, we would need a more efficient method to compute optical flow to have more reliable quantitative measurements. On the other hand, our current algorithm could directly be used as a backup device for the 'mechanical' odometry. It could detect slippage, or contradictions between the reel movement of the robot and the theoretical odometric based one, and eventually automatically correct the motors commands, overriding the odometry feedback.

6 Conclusion, future work

The first thing to understand about this work is that its aim was to be the starting point of a low-cost, multi-purpose, vision based sensor for mobile robotics. We chose to focus ourselves on the use of the optical flow, so we had to develop an algorithm capable of computing optical flow in a real time manner with computation power limitations. In its actual configuration (placement of the sub-images, number of optical flow vectors,…) our algorithm has correctly realized the aims we fixed at the start of this project (qualitative 4DOF movement classification and robot movement classification), but is not directly usable on a real robot. However, here are some ideas of what the next step of this project should be in order to really create an usable sensor:

- lateral obstacle detection can surely be implemented using this algorithm but with a different positioning of the sub-images. For example by looking only for the optical flow vectors of the sides of the images (see Fig. 6.1), and comparing the difference between the left and right vectors' magnitude. The same configuration could be used for navigating into a hallway and staying at its center by equating left and right vectors magnitudes. And even our algorithm with its poor quality of quantitative information would be enough.

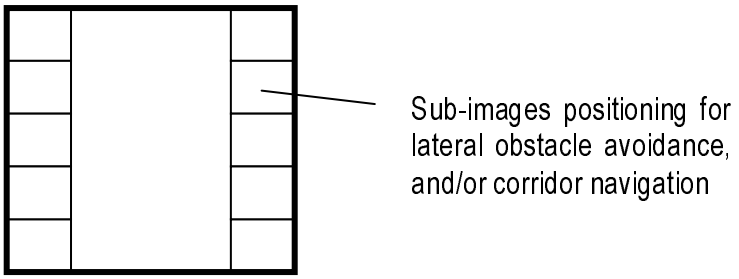


Fig.6.1: The sub-images placing in a corridor following or lateral obstacle avoidance mode

- frontal obstacle detection could be done the same way, using a sort of looming, we could just use the actual sub-image configuration and, when moving the robot forward at an approximate known speed, decide whether there's an obstacle in front or not by looking at the magnitude of the optical flow vectors (as we know that the nearer from an object we are, the bigger the divergence of the vectors will be when going toward it at a constant speed). Again, to do obstacle avoidance in such a case, we don't really need reliable quantitative information from our vector field, but only a qualitative

kind of tendance which will not tell us 'you are at X mm from an obstacle', but rather 'I think that we should turn because there could be an obstacle in front of you'.

- using our algorithm as a complement for odometry (§ 5.3.1)

- using our algorithm for robot localization. One can imagine to have a robot with a map of its environment in memory, but no sensor to position itself on the map. Knowing the start point of the robot we could move the robot towards its environment while integrating its movement as seen by the camera using our algorithm. At the end of the movement, we would have an approximate path of the robot motion (like those of Fig. 5.4 c. d. & e.), but this approximate path could be warped on the previously known map and leading us to a well defined path.

A lot of different possibilities are opened by this algorithm, leading us to the previously fixed goal of having a cheap vision based sensor.

7 Acknowledgements

I would like to thanks Pr. Illah Nourbakhsh, Sebastien Grange and Terry Fong who much helped me during this project. I would also like to thank the Robotics Institue of CMU and the VRAI-group of EPFL for having made my stay in Pittsburgh possible.

Pittsburgh, February 19, 2003

Sebastien Dey

8 References

[1] Insect vision, navigation and cognition laboratory web-page, http://cvs.anu.edu.au/insect/insect.html

[2] MV Srinivasan, M Lehrer, WH Kirchner and SW Zhang, "Range Perception through Apparent Image Speed in Freely Flying Honeybees", *Visual Neuroscience*, 6:519-535, 1991

[3] F. Bremmer, M. Lappe, " The use of optical velocities for distance discrimination and reproduction during visually simulated self motion", Exp Brain Res, 127:33–42,1999

[4] Advantech PCM3350 webpage, http://www.advantech.com/products/Model_Detail.asp?model_id=1-9B200&bu=

[5] J.L. Barron, D.J. Fleet and S.S. Beauchemin, " Performance of Optical Flow Techniques", *IJVC 12:1,* pp43-77, 1994

[6] P. Golland, "Use of color for optical flow estimation", Research Thesis, Israel Institute of Technology, 1995

[7] S.Telizer, " Optical Flow Based Robot Navigation", http://www.ai.mit.edu/people/lpk/mars/temizer_2001/Optical_Flow/#References, MIT, 2001

[8] A.A. Argyros, "Reactive robot navigation, a purposive approach", *Institute of Computer Science (Greece) & Computer Vision and Active Perception Laboratory (Sweden)*

[9] A. P. Duchon, "Maze Navigation Using Optical flow", *Department of Cognitive and Linguistic Sciences,* Brown University

[10] E. Sahin, P. Gaudiano, "Visual Looming as a range sensor for mobile robots", *Boston University Neurobotics Lab*

[11] E. Sahin, P. Gaudiano, "Development of a Visual Object Localization Module for Mobile Robots", *Boston University Neurobotics Lab*

[12] J.M. Walton, "Terrain Mapping from a Mobile Platform Through Optical Flow Techniques", *Department of Electrical Engineering and Computer Science, MIT, 1995*