Robot navigation using stereo-vision

Sylvain Lecorné, ENSEIRB, France

Alfredo Weitzenfeld, ITAM, Mexico

***Abstract*-- This article aims to present a stereo vision based algorithm developed for robot navigation. This algorithm has to analyze the minimum information to find obstacles and goals.**

The algorithm works in an indoor environment with walls, doors and obstacles. It first calculates the displacement of the relevant pixels of the pictures, calculate the distances of those points and then applies a filter to find the doors of the room in which the robot is. Another algorithm permits the robot to navigate with that information.

In a robotic context the algorithm has to be fast but efficient. The algorithm steps are: get pictures from 2 cameras; calculate contrasts of those pictures; find relevant points where to calculate displacements and 3-dimensional information; calculate displacement for those points; convert displacements into a top view of the environment; find angles where there probably is a door; determine the angle of the goal and goe toward.

***Index terms*—Robot navigation, stereo vision, optic flow, indoor environment**

I. Introduction

The goal of the project is to create an autonomous robot able to navigate in an office-type environment using stereo-vision. The article describes basic stereo vision algorithms; it then explains the algorithm developed as part of this work.

II. Stereo vision

A. Stereo vision concept

In Stereo vision with two parallel cameras intersection lines from the cameras to an object at an infinite distance will appear on the same corresponding pixel in the two cameras while an object close to the cameras will appear at two different pixel locations. As shown in figure 1, the nearer the object, the bigger its displacement between pictures.

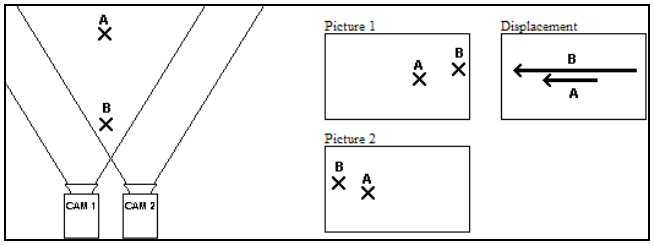

Fig. 1: Stereo vision with two parallel cameras

Distance computation can be done resolving using the following equation based on the diagram in figure 2:

$$d = \tan(\alpha 1) * a$$

$$d = \tan(\alpha 2) * b$$

$$a + b = cameraDistance$$

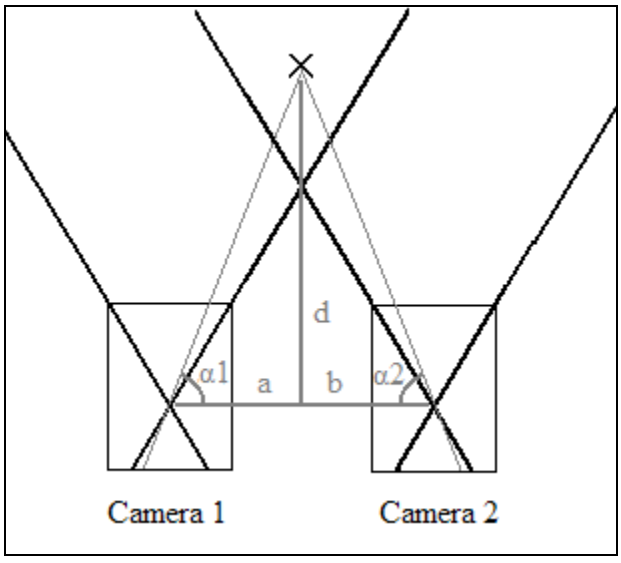

Fig. 2: Stereo vision object distance computation

To calculate the angle *α1* or *α2* we compute a proportional relation with the pixel abscise, i.e. the pixel displacement between the two cameras.

A good approximation is an invert-relation between this displacement and the distance of the object. In the algorithm we use that relation:

$$d = const / displacement$$

Many animals like mammals use stereo-vision to know distances to obstacles, food, danger, etc.

B. Optic flow

Some animals interpret visual information in different ways. Optic flow is the comparison between two pictures from the same eye (or camera) at two different times, when the animal or robot is moving [1][2][4]. For example insects like bees analyze moving pictures in each eye to calculate distance to obstacles (fig. 3). Another example is chicken that move their head rapidly back and forward to generate optic flow to produce three-dimensional information (fig. 4).

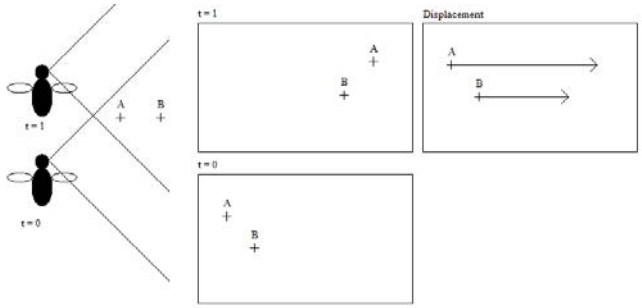

Fig. 3: Bees optic flow generation

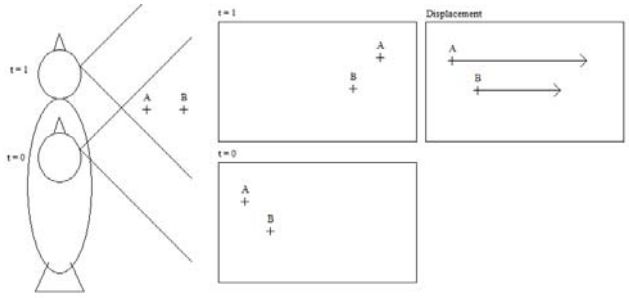

Fig. 4: Chicken optic flow generation

C. Algorithm comparison

To compute three-dimensional information with optic flow we compare two images from one camera at two different times. Typically the camera movement is perpendicular to the camera view direction but the movement can also be parallel to the camera view direction. The advantage is that we can compute three-dimensional information with only one camera. In an animal context it is possible to compute three-dimensional information from eyes where vision does not cross (one eye sees the left, and the other one sees the right). In a robotic context we have to know how the robot moves between the two images to compute the optic flow.

For example if we think the robot moved right but it did a little rotation it can disturb all the calculations. Furthermore if an object is moving in the scene it can be wrongly interpreted as three-dimensional information. If all the optic flows between two pictures are parallel it does not mean that they will be horizontal. Indeed if the robot is in an irregular surface, camera altitude or angle could have changed easily between continuous pictures as the robot moves. That forces us to look for optic flows in all directions.

With stereo vision we get two pictures at the same time from two cameras. With that method we do not have any problem with moving objects in the scene. Knowing exactly the distance of the two cameras we can easily convert the displacement generated into distance information. Another advantage is that displacements are always parallel and in the same direction because cameras do not move with respect to each other. We only have to look for displacements in one direction.

III. EXPLANATION OF IMAGE PROCESS AND NAVIGATION

The whole process cycle is made of 6 steps:

- Contrast calculation for left and right pictures: generates 2 contrasts images.
- Relevant points calculation in the left picture: generates a Point's array where we will calculate displacements.
- Displacement calculation between relevant points of left picture and best corresponding point in right picture: generates a Point's array representing for each relevant point in the displacement vector.
- Top-view conversion of displacements (converted in distances from the robot): generates an array containing for each relevant point its distance from the robot.
- Finding doors: generates a curve representing distance function derivative.
- Navigation: defines angle and distance the robot has to move to.

A. Contrast calculation

1) Why to calculate contrasts?

Different materials reflect light in different ways. If we get two images of the same object from two different angles we can get different colors and different intensities of the same portion of an object. Furthermore if we use cameras with automatic regulation of light intensity we can get different settings for the two angles. In that case it is very difficult to compare effectively the two images to find the displacements between each other.

If we take pictures of a single object (without moving the camera) changing light position and intensity, the contrasts pictures won't vary much. That's why we use contrasts to attenuate those light problems.

Furthermore in a monochrome area of a picture it is impossible to compute displacements. We can only calculate it where some contrast exists. That's why we also

use contrasts to determine where we can efficiently calculate displacements.

2) Algorithm

The algorithm we use to calculate contrast in a point computes the sum of the difference in red, green and blue between that point and the surrounding points. The result is the intensity of a grey point. Points with a strong contrast will appear light while others dark (in our paper we will invert black and white to make it more visible). We can note that this algorithm does not normalize the contrast because it will not affect the process. That is why we simply do the sum of differences and not the average of differences.

Contrasts are computed for pictures of the 2 cameras. In each point we use that formula:

$$Contrast(x,y) = \sum_{i,j} Diference(Pixel(x,y), Pixel(x+i, y+j))$$

$$Diference\ (p1, p2) = (abs\,(p1.red\ -\ p2.red\,) + abs(p1.blue - p2.blue) + abs(p1.green - p2.green))/3$$

The resulting image after this computation is shown in figure 5.

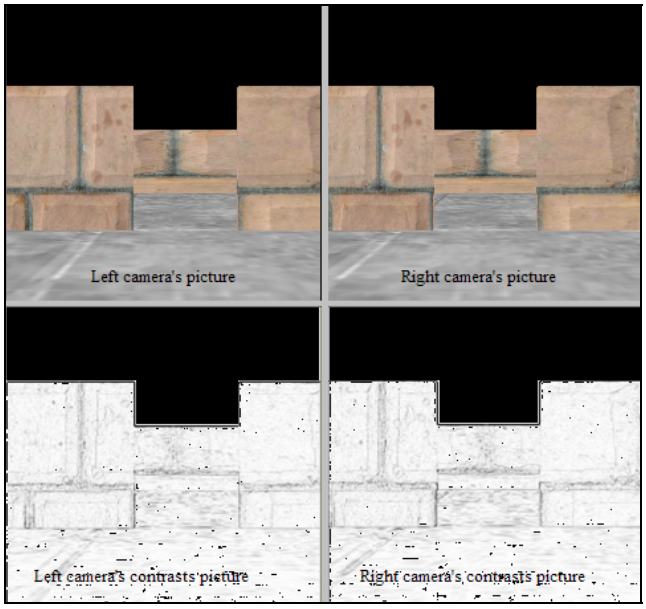

Fig. 5: Contrast calculation

B. Relevant points where to calculate displacements

The relevant points to calculate displacements are defined as the points that contain more disparity (in the left contrasts picture). To do this we calculate the standard deviation between the point intensity around the one we are interested in. A constant minimum value defines if a point is relevant or not.

These points are relevant for two reasons. The first one is that if there are not many changes in the picture, the displacement calculation will not be accurate. The second reason is that the robot has to know where obstacles and doors are. An object or a door will appear with high contrasts making it interesting to analyze those parts of the picture. In another way a partition of the picture without much contrast can be a wall (without enough texture) and it is not interesting to calculate the displacement in each point of the wall.

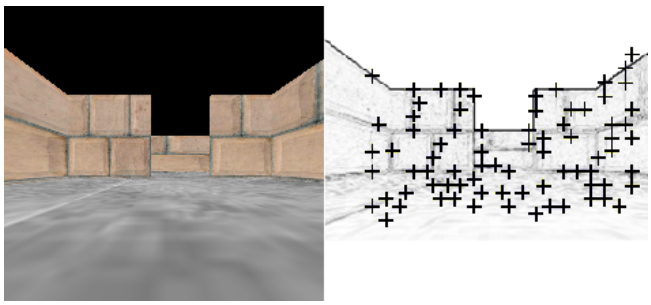

Fig. 6: Relevant points calculation (from the left contrasts picture)

We can see in figure 6 that there are relevant points (crosses in the right side of the figure) in every part of the picture where there are some changes. We can note that there are no relevant points near the borders (and more at the left) of the picture because we can not calculate the displacement in those areas. Indeed the corresponding area in the other picture could be away from the visual angle because from the right camera perspective, we see objects more towards the right.

C. Displacement calculation

Once we know where to calculate the displacements we try for each point to find the area in the right image that best corresponds to that area in the left image. For each point $(X+x, Y+y)$ of the right image around the point (X, Y) we are interested in calculating the difference between the left and right image and consider like displacement the vector (x, y) that minimize that difference (one vector per relevant point).

$$AreaDiference(X,Y,x,y) = \sum_{i,j} Diference(Pixel(Picture1, X+i, Y+j), Pixel(Picture2, X+x+i, Y+y+j))$$

In the particular case of stereo-vision (and not optic flow), we know that the two cameras are in the same horizontal line. For that reason we know that the displacements are horizontal, and we let y=0 in the formula. Furthermore an object appears always more towards the left in the right camera picture than in the left one. That is why we only have to look for displacements in one direction: x will move from the negative value of some parameter to 0.

Another parameter defines the size of the window that permits us to compares areas in the two pictures (in the formula, i and j will move between the negative and positive parameter window). This is shown in figure 7.

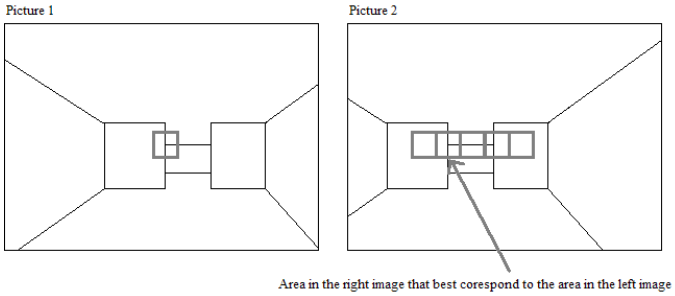

Fig. 7: Displacement calculation

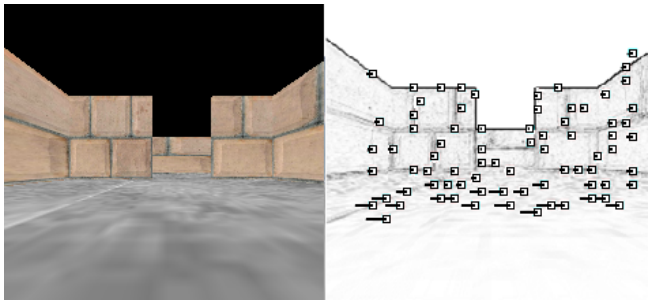

Fig. 8: Results of displacement calculation (in the left contrasts picture)

We can see in figure 8 that our algorithm gives us good results. In the picture we see very clearly that the closest points generate the biggest displacements. An important problem with the algorithm is that it is not able to calculate displacements in areas where there is not much contrast.

D. Top view transformation

Once we have the displacements we convert each one into a distance from the robot (an inverse relation). If we calculate optic flows in a 360° (with 10 steps of 36 degrees in our example) we can compute the complete room where the robot is. Figure 9 shows what the robot can see with those 10 steps:

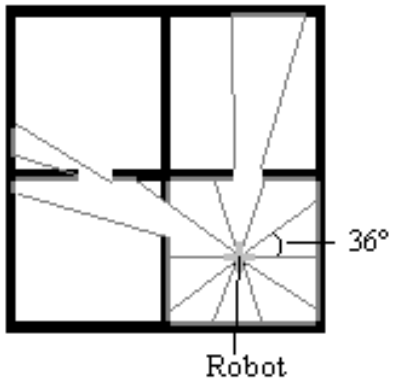

Fig. 9: What the robot at best can see

We have seen before that the algorithm permits to calculate displacements for each relevant point. We then convert each relevant point (position in X) into an angle to be able to remember information of 10 displacement calculations and have distances all around the robot. Figure 10 shows the result of the algorithm.

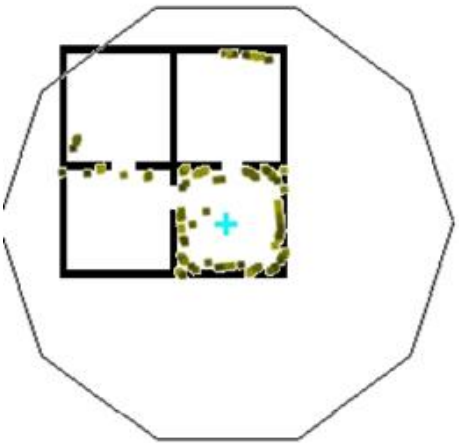

Fig. 10: Image computed from the two cameras pictures with 10 steps of 36 degrees

In those pictures we see that our algorithm permits to reconstitute the environment room, even if it generates some wrong points (for example the point closest to the robot do not correspond to a wall). The picture generated gives us good enough map information to be able to navigate.

E. Finding doors

In the algorithm we consider there is a door where the displacements have important changes. We generate a curve representing the changes in displacements calculated as the difference between the moving averages of two consecutive points. For each point x of the displacements curve we calculate the point *Changes(x)* of the changes curve as:

$$Changes(x) = Abs(MovingAverage(x) - MovingAverage(x-1))$$

$$MovingAverage(x) = (Displacement(x-1) + Displacement(x) + Displacement(x+1))/3$$

If we execute the algorithm in a room with one door the simulation gives the result shown in figure 11 (the center curve is a polar representation of the changes curve). We can see that the two sides of the door generate important values in the changes curve.

We have seen in figure 10 that the distances computation can give some undesired points. We use moving averages to identify these undesired points. These points are then taken out from further calculation. Indeed an undesired point will probably be far away from the others points, tending to generate high values in the resulting curves.

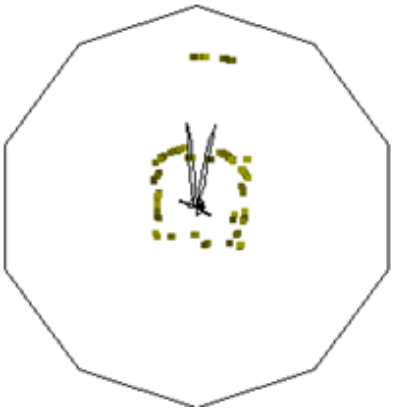

Fig. 11: Changes in the optic flow curve

F. Navigation

A very simple navigation algorithm was developed to test the previous algorithm. It first computes three-dimensional information in 360 degrees to find a door. When it finds a door it only computes three dimensional information with angles that permit to see the door but not in 360 degrees (see figure 12).
To find a door, the algorithm looks for the two highest values of the changes curve and considers the door direction as the average of the angles corresponding to those two highest values.
It then goes towards the door (a constant value defines how much it advances). When the robot passes a door it sees another one and does the same.

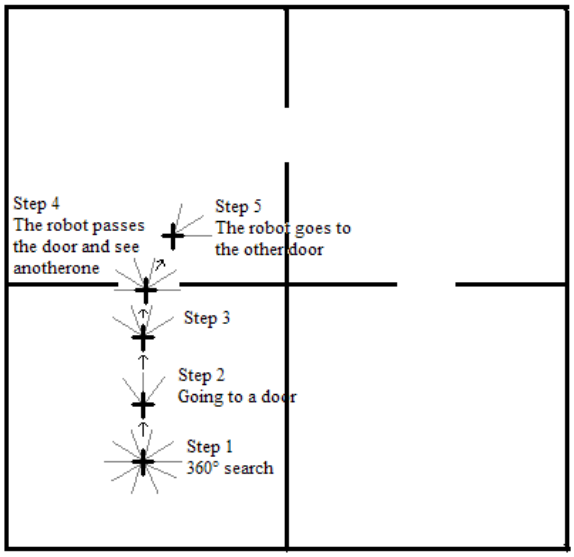

Fig. 12: The diagram shows intermediate steps as the robot goes from a room to another one

In our example with 4 rooms and 3 doors the robot goes successfully from the first to the last room and then goes back to the first one as shown in figures 13 and 14.

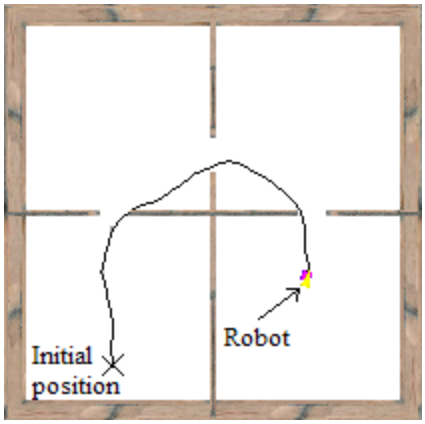

Fig. 13: Navigation till the last room

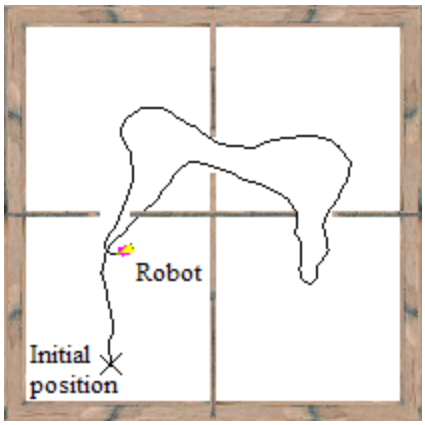

Fig. 14: The robot goes back to the first room

IV. Conclusion

The stereo vision algorithm developed is interesting due to its simplicity and performance. The time it needs to compute is compatible with real time robotic context. We have developed it in a simulator with Java 3D. We have to test it in a real environment with true images and will probably have to adjust it. In the current model, obstacles in the room are not considered. This work is still in progress and many additional issues need to be addressed such as how to identify hard to recognize doors and distinguish them from other objects in the environment.
In the future we would have to change door recognition in order not to interpret an obstacle as a door (an obstacle generates strong changes in displacements curve). For example, we can consider that a door is between two areas with strong changes in optic flows but with distances further apart. Furthermore the robot should avoid those obstacles, something that can be done by using repulsion vector fields. We also need to determine how to interpret three-dimensional information as obstacles.
The navigation algorithm is still very simple. It just looks for doors and goes where it finds one. Although it works well in our simplified simulated environment, we would need to test it and extend it in more complex scenarios. For

example if the robot enters a room and there is another door on the same wall we can not know if it will go to each room.
Some more complex algorithms exist. Many of them use landmarks to do an abstract map of the environment [5]. In our example we could do a landmark each time the robot passes a door and memorize them with angle information between the doors of a room.
Many algorithms also use neural networks to navigate. We could try to reproduce biological comportments to do an algorithm that learns the map to navigate efficiently [3].

V. References

[1] S.Sudhir Mohith, 1998, Real Time Interactive Object Tracking (RIOT), Master´s Thesis, Dept of Computation, University of Manchester Institute of Science and Technology, Manchester, U.K.
[2] Selim Temizer, 2003, A Dynamical Model of Visually-Guided Steering, Obstacle Avoidance, and Route Selection, International Journal of Computer Vision, vol 54, issues 1-3, pp 13 – 34, August-September, Kluwer.
[3] F.J. Corbacho and A. Weitzenfeld, 2002, Learning To Detour, in The Neural Simulation Language: A System for Brain Modeling, Eds A. Weitzenfeld, M.A. Arbib, A. Alexander, pp 319-341, MIT Press.
[4] J.L. Barron, D.J. Fleet, and S.S Beauchemin, 1994, Performance of optical flow techniques, International Journal of Computer Vision, vol 12, issue 1, pp 43 - 77, February, Kluwer.
[5] Don Murray and Jim Little, 2000, Using real-time stereo vision for mobile robot navigation, Autonomous Robots, vol 8, no 2, pp 161-171.