Ball Detection and Predictive Ball Following Based on a Stereoscopic Vision System

Davide Scaramuzza, Stefano Pagnottelli and Paolo Valigi
Dipartimento di Ingegneria Elettronica e dell'Informazione
Università di Perugia
Via G. Duranti, 93 - 06125 Perugia – Italy
davsca@inwind.it [pagnottelli,valigi]@diei.unipg.it

***Abstract*—In this paper we describe an efficient software architecture for object-tracking, based on a stereoscopic vision system, that has been applied to a mobile robot controlled by a PC. After analyzing the epipolar rectification required to correct the original stereo-images, it is described a new valid and efficient algorithm for ball recognition (indeed circle detection) which is able to work in different lighting conditions and in a manner faster than some modified versions of Circle Hough Transform. Then, we show that stereo vision, besides giving an optimum estimation of the 3D position of the object, is useful to remove lots of the false identifications of the ball, thanks to the advantages of epipolar constraint.**

Finally, we describe a new strategy for ball following, by a mobile robot, which is able to ”look for” the object whenever it comes out of the cameras view, by taking advantage of a ”block matching” method similar to that of MPEG Video.

***Index Terms*—ball detection, ball tracking, following, predictive, stereoscopic vision**

I. Introduction

Stereoscopic vision is a technique for inferring the 3D position of objects from two (or more) simultaneous views of the scene. Its advantages are that it offers a cheap solution for 3D reconstruction of an environment, it is a passive sensor and thus it does not introduce interferences with other sensor devices; finally, it can be integrated with other vision routines (such as object recognition and tracking) that we will describe.

In view of this application, we have realized a software architecture for recognizing a ball moving in front of two cameras and inferring its 3D position with respect to the cameras reference system. By using the information provided by the binocular vision, the algorithms have been applied to a mobile robot endowed with the above stereo rig and have allowed it to follow a ball rolling on the floor and to look for the object whenever it falls out of the cameras view.

A new technique for ball detection and an experimented strategy for looking for the ball are the objectives of this research.

The hardware is composed of a Pentium IV, 1,7 GHz, that manages the video acquisition of stereo pair, the processing of the two digital images and the digital signals to control the robot. Because of the computational burden required to process and rectify the original images via PC, the stereo images are captured at a frame rate of 6 Hz. So it has been developed a new and efficient algorithm for ball recognition that considers only those points of the edge maps of the images which are effectively candidate to belong to an arc and therefore to the contour of a possible ball.

This paper is organized as follows: first the software architecture for inferring the 3D position of the ball will be introduced, then we will describe how the new circle detection algorithm works. Finally, we will show the strategies for the ball following and the ball searching tasks.

II. 3D Reconstruction Architecture

In general the reconstruction of the world seen through a stereo camera can be divided in two stages:

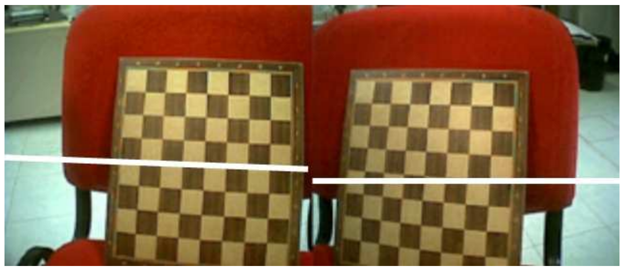
1) *Correspondence problem:* for every point in one image find out the correspondent point, on the other image, that is the projection of the same 3D point.
2) *Stereo-Triangulation:* derived the intrinsic and extrinsic parameters of the cameras and given the corresponding points, compute the (X, Y, Z) coordinates of the 3D point they refer to.

Both of the above tasks are easier to realize when cameras are in a standard setting, that is with parallel optical axes and coplanar image planes. In this setting epipolar lines correspond to the same horizontal rows in the two images and point correspondences are searched over these rows. Obviously the *standard setting* cannot be obtained with real cameras but, if we know the cameras calibration parameters (e.g. the focal distance of the two cameras, the entity of lens distortion and the relative position and orientation of the cameras) this problem may be overcome through the so called Epipolar Rectification.

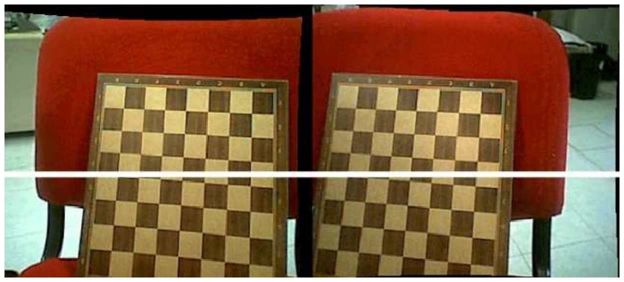
Rectification determines a transformation of each image plane so that pairs of conjugate epipolar lines become collinear and parallel to one of the image axes (usually the horizontal one) [7].

Figures 1(a),1(b) show two epipolar lines before and after the images have been rectified. The rectification task precedes 3D reconstruction and may be divided into three main stages: *perspective correction* (which considers the relative position and orientation of the cameras), *lens distortion compensation* and *bilinear interpolation* (needed to map the two final images with respect to the original ones, whose indexes of the texture map are stored in a Look-Up-Table).

The second stage in view of 3D reconstruction is to find the correspondent points relative to the object we want to localize. This is possible by applying the circle detection



(a)



(b)

Fig. 1. a) A stereo pair before being rectified. b) A stereo pair after rectification.

algorithm to each single image. When the algorithm stops, the locations of the circle centers in the images are obtained.

At this point it is to be noted that, if a ball is really present in front of the cameras, the centers coordinates must belong to the same horizontal line in order to satisfy the epipolar constraint, that is they must have the same y coordinate. So any violation of this law may be used to infer that the ball is absent. In practice, as the centers coordinates are real, due to interpolation, we have set the algorithm to reject those locations whose difference between the two ordinates is greater than 2 pixels. In this case the average value of the ordinates is assumed as a new y coordinate.

Derived the intrinsic and extrinsic parameters of the cameras and the correspondent points, it is possible, through triangulation, to compute the 3D position of the object in the real world. Thanks to an excellent calibration of the cameras we have obtained an accuracy lying between 1/1000 and 2%, respectively for 1 and 2 meter distances.

III. CIRCLE DETECTION

A. *Overview of existing versions*

The detection of the circles in digital images is one of the most important problems in visual industrial applications as circular objects frequently occur in many natural and manmade scenes. So far many circle-extraction methods have been developed [2],[3]. The Circle Hough Transform (CHT) [4] is one of the best known algorithms which aims at finding circular shapes with a given radius r within an image. Usually edge map of the image is calculated, then each edge point contributes a circle of radius r to an output accumulator space. For unknown circle radiuses, the algorithm should be run for all possible radiuses to form a 3-dimensional parameter space, where two dimensions represent the position of the center and the third one represents the radius. The output accumulator space has a peak where these contributed circles overlap at the center of the original circle.

In spite of its popularity, the large amount of storage and computing power required by the CHT and the inaccuracy in case of excessively noisy images are the major disadvantages of using it in real-time applications. So a number of modifications have been widely implemented in the last decade in order to reduce the computational burden and the number of false positives typical of the CHT. Use of the edge direction was first suggested by Kimme *et al*. [5], who noted that the edge direction, on the boundary of a circle, points towards or away from the circle center. This modification reduced the computational requirements as only an arc needed to be plotted perpendicular to the edge orientation at a distance r from the edge point. Subsequently, Minor and Sklansky [6], and recently Faez *et al*. [1], extended the use of edge orientation, by plotting a line in the edge direction. This has the added advantage of using a two rather than a three-dimensional parameter space. In this case the output accumulator space has a peak where these contributed lines overlap at the center of the original circle.

B. *Limits of the last version*

The limits of this last version are that:

1) Each edge point contributes a line to the parameter space independently of its neighbours. This results in an useful increase of computational burden (see fig. 3).
2) Edge orientation is taken equal to gradient direction that is calculated by applying a spatial differentiation operator to the original image. But this is true only for one color circles. Consequently gradient direction rarely coincides with the arc direction as depends on lighting conditions, color changes and shadows (see fig. 4).

Figure 2 shows the original color image used to test the algorithm and its relative edge map. As an accumulator space we have defined a 2D-matrix the size of the edge map. Drawing a line in the accumulator matrix means to increase by one the current value of each interested cell.

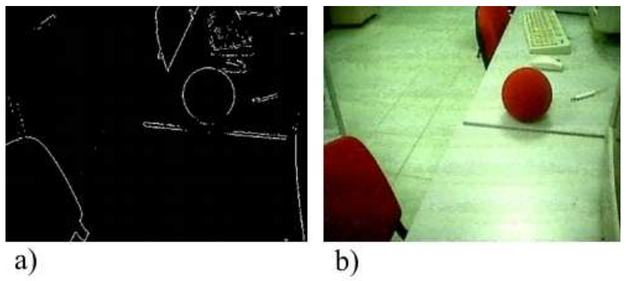


Fig. 2. a) Edge map of test image , b) Original test image.

The result of this process, iterated for every edge point, is shown in fig. 3, where hot colors correspond to greater values and vice-versa.

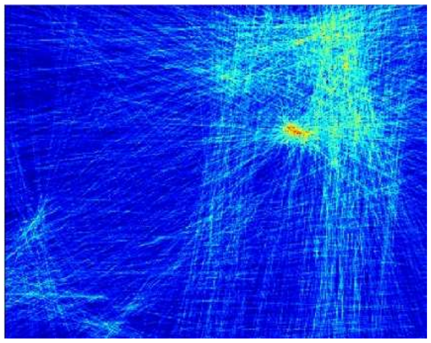


Fig. 3. Accumulator or parameter space.

The darker point in fig. 3 corresponds to the center location. However, as the line direction coincides with the gradient direction, these lines do not intersect all around an average point but a wider region (see fig. 4 where only some of radial lines are superimposed on the edge map).

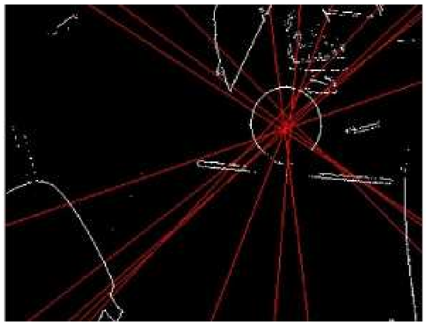


Fig. 4. Some lines plotted on the edge map.

C. The new algorithm

We have developed an innovative and efficient algorithm that computes the edge direction related to each edge point taking into account spatial distribution of its neighbours. Thanks to the major precision in estimating the edge direction, if a circle is present lines intersect in a smaller region and the peak in the accumulator space has a higher value. In addition, our algorithm also considers only those points in the edge map which are effectively candidate to belong to an arc and therefore to the contour of a possible circle. In particular the algorithm is able to (see fig. 5): reject angular points (1 in the figure), ignore isolated points (2 and 3), reject straight segments (4 and 5), plot lines in the direction of arc concavity.

D. "Pixel-to-Pixel" algorithm

Our algorithm, called "Pixel-to-Pixel", acts directly on the binary edge map, where each pixel values 1 if it is an edge point, or 0 otherwise. In order to identify line direction we explore the distribution of the pixels located in a window of dimensions $(2k+1)\cdot(2k+1)$ centred on each edge point (see fig. 6 for the axes direction).

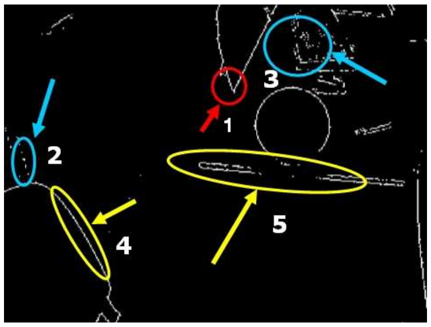


Fig. 5. The regions rejected by the new algorithm.

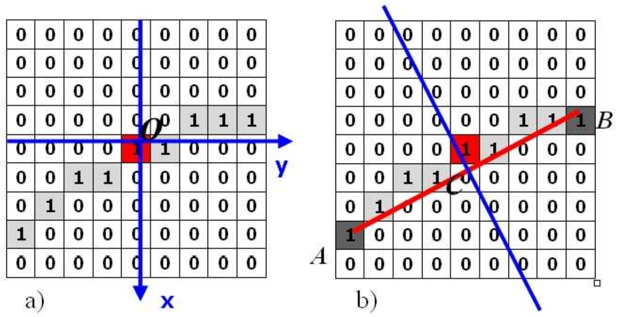


Fig. 6. A squared window centred on an edge point. a) axes origin at the central point b) chord and radial line of the arc.

Suppose the set of "1" in fig. 6a) to belong to a circular arc. Then the best estimation of radial line coincides with the perpendicular one to the chord $\overline{AB}$ passing for the mid point C of coordinates:

$$C = ((x_A + x_B)/2;\ (y_A + y_B)/2) \tag{1}$$

To determine the two chord extremes A,B, the algorithm examines all edge points in the current window and chooses those verify both:

1) Each of two must have the greatest Euclidean distance from axes origin.
2) One at least of the homologue coordinates must have opposite sign. This is needed to reject some spurious points or angular points.

In order to reduce the computational burden required to compute quadratic Euclidean distance with respect to origin $x^2 + y^2$, we have defined a new distance with the same results: $d(x, y) = |x| + |y|$. So only absolute values need to be calculated. After A,B have been determined, and so the equation of perpendicular line, we have used the location of mid point C to infer the arc concavity. In this way lines in the parameter space may be plotted only in the direction of concavity and consequently computational burden decreases.

To detect (and so reject) straight segments, the algorithm must verify that points A,B and O are aligned, that is when chord $\overline{AB}$ exactly intercepts x, y axes at origin O (see fig. 7).

It may be easily shown that this implies:

$$|y_B(x_B - x_A) - x_B(y_B - y_A)| = 0 \tag{2}$$

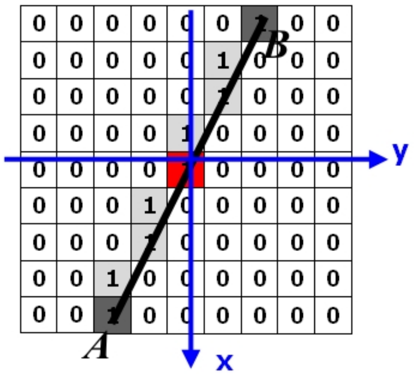


Fig. 7. Edge points distribution of a stright segment.

In practice, being parameter space discreet, x, y intercepts may lie in the range $[-1,+1]$ and the above condition becomes:

$$|y_B(x_B-x_A)-x_B(y_B-y_A)| \leq min(|x_B-x_A|,|y_B-y_A|) \quad (3)$$

Up to now we have only taken into account the three points A,B,O to get information about edge direction or to see if edge belongs to a straight segment. This occurs because we have supposed only a chain of connected pixels being present in the current window. However it may happen that some isolated edge points also appear in the window giving wrong results. Of course we could look for those points belonging to the same connected arc and discard the others, but this search process would increase computational time.

To overcome this limit we have exploited the fact that any edge in the squared window is 1 pixel thick at most because of use of an edge thinning process after edge detection. So if one only arc is present, then it must be $2k+1$ pixels long.

To conclude, "Pixel to Pixel" algorithm counts the edge points in the current window. If this count equals $2k+1$ then a potential arc may be present; then the algorithm looks for A,B and goes on as previously mentioned.

Figures 8 a),b) show some edge points which are discarded because they are less or more than $2k+1$. Note also that c) may represent the end of an arc but it is rejected too. In fact it would not probably contribute to correctly determine edge orientation. On the contrary d) might be correct but this is not the only case. Nevertheless the condition $n = 2k+1$ significantly reduces the number of cases which need to be examined.

E. Algorithm steps

Here we resume how "Pixel to Pixel" algorithm works:

1) Obtain the binary edge map of the raw image.
2) Create an accumulator space the dimension of the edge map and set zero each value.
3) Find next edge point (say O) and consider those points within a $(2k+1)\cdot(2k+1)$ window centred on O.

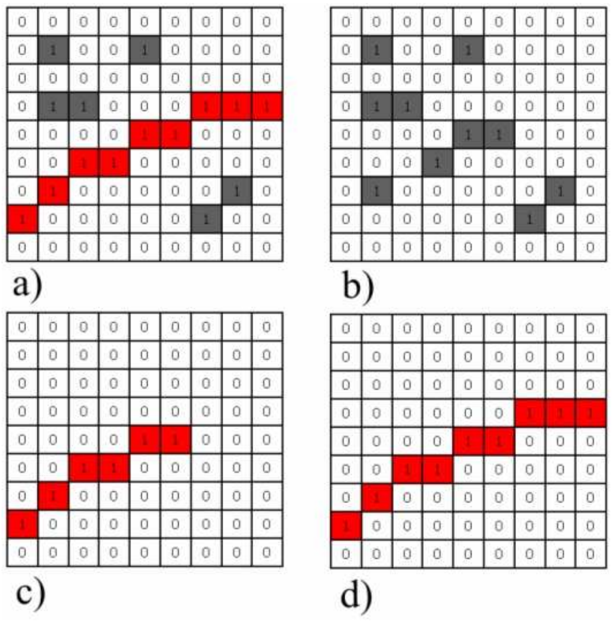


Fig. 8. a) b) c) are rejected because counts are different from $2k+1$ d) is correct.

4) Count edge points within the window. If this count equals $2k+1$ then go on, else repeat from step 3.
5) Find extremes A,B in the window.
6) Verify that A,B and O are not aligned. If it is not so then repeat from step 3, else compute the equation of line perpendicular to $\overline{AB}$ and passing for the mid point C.
7) Compute the direction of concavity and plot the above line in the accumulator space only towards this direction.
8) Come back to step 3.
9) When all edge points are examined, apply a smoothing filter to the parameter space and find the global maximum, whose coordinates give the potential circle location.

In order to improve the capability to identify the correct maximum, we have introduced a smoothing filter (e.g. an average filter) as a last step. In fact it is to be noted that all radial lines of a circle do not intersect in one single point due to the discreet nature of parameter space. The purpose of the average filter as a smoother (a squared convolution mask whose elements are all 1) is to emphasize these intersections.

F. Results

Here we present the circle detection results obtained by using different values of k. As an accumulator space we have used a 2D-matrix of size 352x288 pixels.

Figure 9 shows all radial lines computed by applying "Pixel to Pixel" algorithm to the test image of figure 2; lines are superimposed on the edge map for major clarity. Comparing it with figure 4 it is evident the better precision of this algorithm to detect circle center. Note also that the number of lines is smaller than the one in figure 3. In fact, straight segments, angular points and isolated points

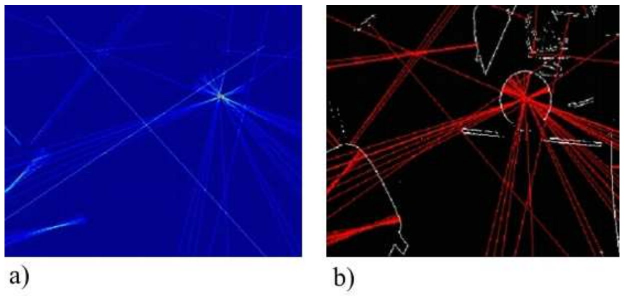


Fig. 9. a) Radial lines on parameter space. b) The same lines superimposed on the edge map.

are now discarded, so a less number of lines needs to be plotted. In the example above we have used k=5. To explain the effect of k we have used a second test image, where four circles of different radiuses are shown (see fig. 10).

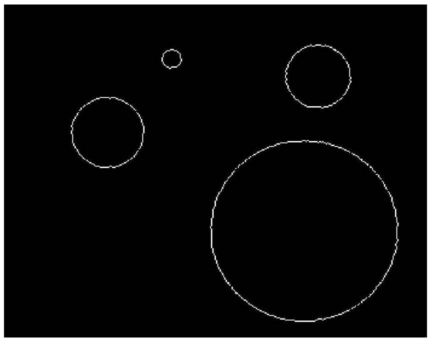


Fig. 10. Second test image.

Choosing $k = 3, k = 5, k = 8, k = 15$ each single circle (from smallest to biggest) may be isolated (see fig. 11).

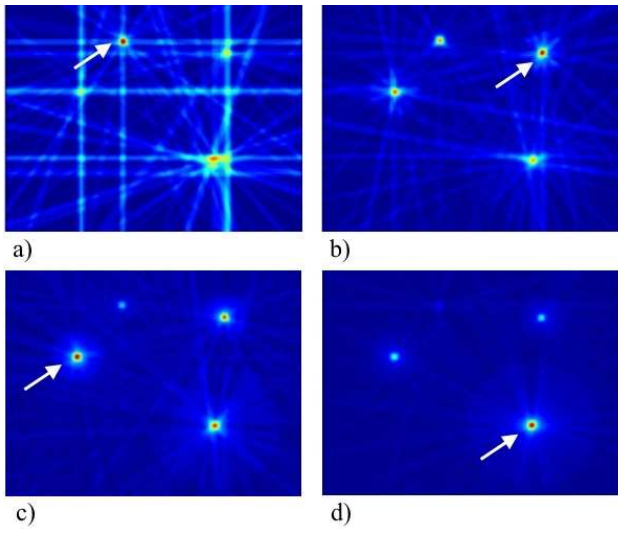


Fig. 11. a) k=3 b) k=5 c) k=8 d) k=15.

By increasing k, the number of votes of the bigger circles rises, while votes of the smaller ones reduce; vice-versa when k decreases. In the experimented strategy of ball following we have set k=5 because this guarantees the best results for different distances from the ball.

IV. BALL SEARCHING

The circle detection algorithm described in section III works very well when a ball is present in front of the cameras. But when the ball comes out of the cameras view it always returns a false detection because there is always a peak in the parameter space. When ball is absent, distribution of centers locations is random due to the presence of noise and so robot motion is chaotic. As mentioned in section II we have used epipolar constraint in order to avoid these false positives but a little percentage of mistakes occurs anyway.

Here we describe a new algorithm that removes up to 90% of false detections and guarantees to look for the ball when it is out of the robot view. To do this we have taken advantage of temporal correlation between consecutive frames, used in MPEG Block Based Motion Estimation[8]. If a ball is present in the current frame it will appear with similar color intensity in the next frame.

One way to express the concept of similarity between two blocks of pixels of RGB images is the so called SSD (Sum of Squared Differences) [8]; let I_1, I_2 be two color image blocks of size $(2k+1)\cdot(2k+1)$pixels, so:

$$SSD = \sum_{i=1}^{2k+1}\sum_{j=1}^{2k+1}[I_{1R}(i,j) - I_{2R}(i,j)]^2 + \tag{4}$$

$$+[I_{1G}(i,j) - I_{2G}(i,j)]^2 + [I_{1B}(i,j) - I_{2B}(i,j)]^2$$

SSD values zero only if I_1, I_2 are equal, but in case of a good matching between blocks its value is anyway small. In practice we have chosen $k = 10$and fixed the SSD threshold to 200,000. So when $SSD < 200,000$ two Blocks of size 21x21 may be considered to be similar.

Our strategy adopts both circle detection and SSD to remove any false detection. Suppose a ball is moving within cameras view. At first it uses circle detection algorithm to identify center location in the current frame. Once this is determined, a block of pixels centred on this point is compared via SSD with a block already stored in memory at previous frames (see fig. 12). If matching is suitable then the stored block is replaced by the present one, otherwise it is compared with blocks of next frames until this condition is satisfied. The Block we have considered is relative to the left camera.

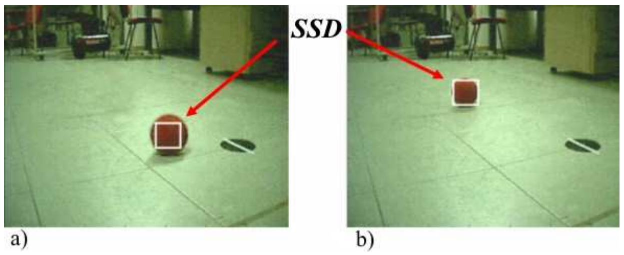


Fig. 12. a) The block considered in a previous frame. b) The new block in the current frame which needs to be matched to the previous one.

This Block updating runs continuously as long as the object is visible. But when ball falls out of cameras view, even if circle detection may return something, SSD gives a too large value so that the ball searching task starts. This means that the mobile robot begins to turn around until ball is visible again. In order to speed up the search, the robot chooses sense of rotation in the direction where the ball has been seen the last time.

Now it is to be noted that when the object appears again, lighting conditions or ball size may be different, so block matching may give a negative result. On the contrary circle detection algorithm returns a valid location.

In order to avoid conflicts between the two algorithms, we have implemented a statistical approach which consists in storing in a shift register the last five 3D locations returned by ball detection and in computing mean and variance of distribution.

In fact, when there is no relative motion between ball and robot variance is theoretically null; in practice a dense distribution of the position values is always present due to image noise. On the other hand, when a relative motion exists consecutive 3D positions of the ball may be generally thought as lying on a straight line corrupted by noise (see fig. 13). In this case the new ball location will be considered if close to the line that best fits the last acquired points.

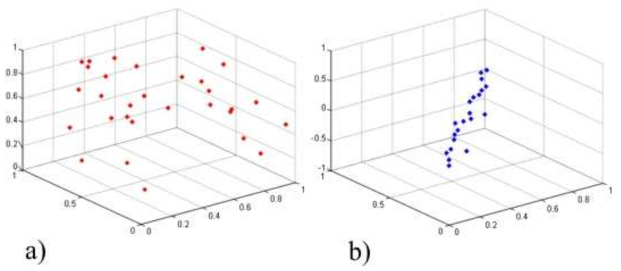


Fig. 13. a) Random distribution when the ball is absent. b) Estimated positions when the ball is moving.

Therefore, to overcome the mentioned conflicts, the algorithm works as follows:

1) Store in a shift register the last five potential 3D positions returned by ball detection.
2) Compute mean and variance of the five-point distribution.
3) Apply ball detection to the current stereo-frame (sixth acquisition).
4) If ball location is less than the square root of variance or is close to the best fitting line, then accept it as correct. Whatever is the result, shift the register and store this position as last.

Observe that statistical verification continuously occurs, together with ball detection and ”block matching”, either when ball is present or absent.

Now we summarize how the predictive algorithm for ball following and searching works. At the beginning, when program starts, robot is still and ”block matching” task does not run yet. However statistical distribution of first five location returned by ball detection is examined. If the current ball position (the sixth one) is considered valid then robot starts to follow it and current Block is stored as a reference.

Subsequently, ball detection, block matching and statistical verification always run together. While ball is rolling within cameras view, statistical analysis has no influence because is block matching that verifies whether a value is suitable. On the contrary when object is out of robot sight this is performed by statistical approach.

V. CONCLUSIONS

In this paper we have presented a software architecture for ball detection and following based on a stereoscopic vision system. In particular we have described a different approach for circle detection and a predictive method to look for the object whenever it comes out of cameras view.

Our circle detection algorithm is able to identify potential arcs in the edge image due to a different way to infer edge orientation, which takes into account spatial distribution of near edge points. The presented algorithm may also discard straight segments, isolated points, angular points and may distinguish the direction of an arc, resulting in a low computational burden. In order to reject any false detection of the ball we have used epipolar constraint that forces two correspondent circle locations to belong to the same row.

Finally we have described a predictive algorithm to look for the object, based on a block matching method among frames and on a statistical analysis of 3D positions of the ball in consecutive frames.

REFERENCES

[1] Ali Ajdari Rad, Karim Faez, Navid Qaragozlou, ”Fast Circle Detection Using Gradient Pair Vectors”, Proc. VIIth Digital Image Computing: Techniques and Applications, December 2003.
[2] T. D’Orazio, N Ancona, G. Cicirelli, M. Nitti, ”A Ball Detection Algorithm for Real Soccer Image Sequences”, IEEE, 1051-4651, 2002.
[3] Coath, P. Musumeci, ”Adaptive Arc Fitting for Ball Detection in RoboCup”, 2002.
[4] R.O.Duda and P.E.Hart, ”Use of the Hough Transform to Detect Lines and Curves in Pictures”, Communications of the ACM 15, pp: 11-15, 1975.
[5] C.Kimme, D.Ballard, and J.Sklansky, ”Finding circles by an array of accumulators”, Proc. ACM 18, pp: 120-122, 1975.
[6] L.Minor and J.Sklansky, ”Detection and segmentation of blobs in infrared images”, IEEE Trans. SMC 11, pp: 194-201, 1981.
[7] A. Fusiello, E. Trucco, A. Verri, ”A compact algorithm for rectification of stereo pairs”, Machine Vision and Applications 12, 16-22, 2000.
[8] I.E.G. Richardson, ”H264 and MPEG-4, Video Compression, Video coding for next generation multimedia”, Wiley, 2003.