Humanoid Gait Stabilization based on Omnidirectional Visual Gyroscope

Matteo Finotto, Emanuele Menegatti
Department of Information Engineering (DEI), Faculty of Engineering,
University of Padua, Via Gradenigo 6/a, I-35131 Padova, Italy

***Abstract*— Omnidirectional vision has already been used for attitude and heading estimation, thanks to its wide field of view. We adapt some existing methods in order to obtain a simple vision system capable to estimate pitch, roll and yaw angles for a real time application, using the parallel lines of the environment. We test this visual gyroscope and visual compass in some common environments to show that the proposed sensor continues to work even if the number of detected lines is small. We mount the catadioptric sensor in place of the head of a humanoid robot and use the measured angles to stabilize the walk, allowing the robot to keep a vertical position also if the terrain changes its slope. In this way we also stabilize the robot's perception, reducing camera oscillations.**

I. Introduction

It is very important for a humanoid robot to have some methods of control in order to stabilize the gait in case of loss of balance.

In general, the problem of estimate the camera motion relative to the environment is fundamental to many vision-based mobile robot applications, such as autonomous navigation and localization. Common sensors, like gyroscopes and accelerometers, are extremely sensitive to measurement errors since the estimate of rotation angles is made through double integration of sensed accelerations. This implies that possible small errors integrated over time lead to a large localization error over long paths. Standard GPSs have other limitations, for example they are not able to work in many indoor and urban environments. All these devices can be affected by structural and mechanical fatigue, vibrations, temperature change and electric interferences, that can cause erroneous data readings [18].

In the last few years omnidirectional vision has started to be used as a tool to support or replace the devices mentioned before. Several methods have been proposed. In [9] they use the image projection of 3D parallel lines to estimate the Z-axis camera rotation angle. It is also possible to extract the XY-plane translation from the optical flow, working with omnidirectional [8] [15] or panoramic images [16]. In these and other papers, they assume to work with wheeled robot moving only along XY-plane and so the principal movements are rotation and translation. Working with a humanoid robot, we are instead interested in a visual device able to measure pitch and roll inclination, as well as yaw angles.

Most of the works dealing with pitch and roll estimation using omnidirectional sensors are related to small helicopters and autonomous planes, belonging to the UAV (Unmanned Aerial Vehicle) category. For cost and weight reasons, they do not mount gyroscopes and accelerometers. The omnidirectional sensor is used both to capture environment images and to estimate the aircraft position. These vision-based attitude computation methods generally use the skyline as reference [12] [6]. However, the horizon line becomes an inadequate feature in indoor environment because it does not completely appear in the image, due to the presence of furniture and other obstacles in the room.

Our approach, adapted from [5], consists in finding the vanishing points of the lines in the image and then calculate camera inclination using the technique presented in [4]. With such a visual sensor, we are able to produce a method to stabilize the walk of our robot, a Kondo KHR-1HV [20], without using any other inertial sensor.

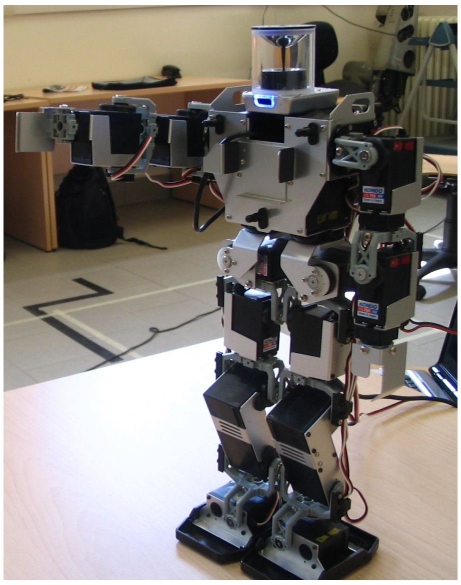


Fig. 1. The robot with the catadioptric sensor.

II. Equivalent sphere

We use a central catadioptric sensor consisting in a coupling between a hyperbolic mirror and a webcam.

Geyer and Daniilidis [7] have demonstrated how the projection of a real point P on the image plane through the omnidirectional mirror is equivalent with a two-step projection via a unitary sphere centred on the focus F_1 of the mirror (the single viewpoint): the 3D point is projected to the sphere surface and then to the image plane from a point placed on the optical axis.

Proceedings of the 4th Workshop on Humanoid Soccer Robots
A workshop of the 2009 IEEE-RAS Intl. Conf. On Humanoid Robots (Humanoids 2009),
Paris(France), 2009, December 7-10
ISBN 978-88-95872-03-2
pp. 52-59

For a hyperbolic mirror this second projection centre S is between the north pole and the sphere's centre (Fig. 2). Its distance from the circle's centre is $\frac{d}{\sqrt{d^2+4p^2}}$, where d is the distance between the foci and $4p$ is the latus rectum (the chord through the focus F_1 parallel to the directrix d) [2].

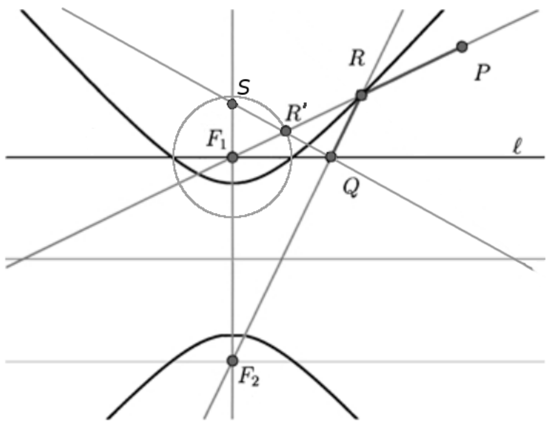


Fig. 2. Two equivalent projections for a hyperbolic mirror (Adapted from [7]). A ray of light passing through P and incident with F_1 is reflected by the mirror in R to a ray of light incident with F_2. Equivalently, the same ray first intersects the circle in R', then R' is projected from S intersecting ℓ in Q. Lines RF_2 and QF_2 are coincident.

In this way, since the two-step mapping via the sphere is a one-to-one correspondence, we can reproject a 2D point back to sphere.

There are several methods to calculate this projection. We choose to first calibrate the sensor with *OcamCalib Toolbox for Matlab* [21] written by Davide Scaramuzza and then to use a function of this tool to calculate the 3D transformation on the sphere. The results are saved in a text file, so we do the all procedure only once.

A. Vanishing points detection

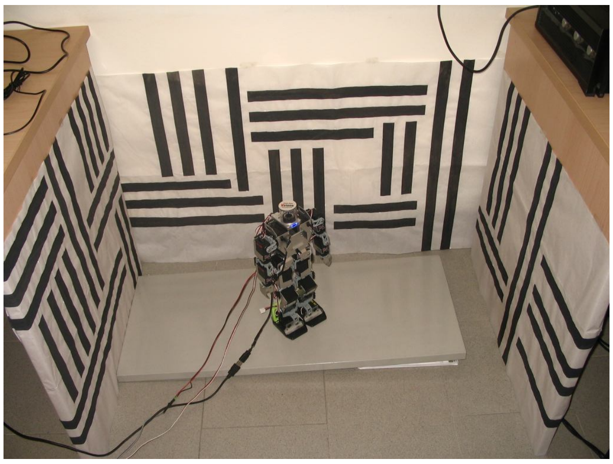
Vanishing points are the points to which parallel lines converge. The wide field of view of an omnidirectional image permits to observe a high number of lines and the associated vanishing points lie on the image plane, so they can be tracked continuously from frame to frame.

The main idea is to extract three orthogonal bundles of parallel lines from the environment, so we can build an absolute reference frame that we will use to calculate pitch, roll and yaw angles.

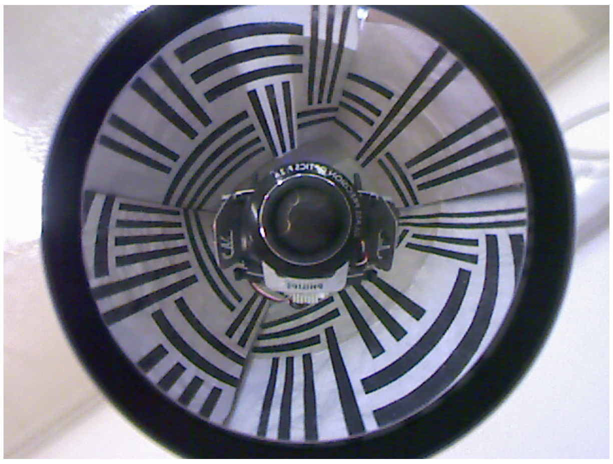
We have built a test environment consisting of four white panels with random black vertical and horizontal lines (Fig. 3).

However, the proposed algorithm works in every environment if it is possible to extract at least two orthogonal bundles of lines, each composed of at least three parallel lines. The more parallel lines we can extract the more the angles estimate will be robust.

Detecting lines in a catadioptric image is not a trivial problem. Indeed, the projection on the image plane of a 3D real line can be any kind of conic like a straight line,



(a) Frontal view



(b) Omnidirectional image

Fig. 3. The test environment.

a circle, a hyperbola, a parabola and it is very hard to detect a conic without knowing its shape. Also occlusions have to be considered. In [17] e [19] a solution using an adaptation of the Hough transform has been proposed. These methods permit to correctly detect the lines, but with high computational cost, too high for a real-time application.

In our work, we adapt the method proposed in [3], able to extract the real world lines from the omnidirectional image working with the equivalent sphere model. The following process is valid for any mirror model once the sensor is calibrated and the 3D transformation involving the sphere is calculated.

The first step consists in detecting the edges in the image using a Canny edge detector. The result is a binary image where the edges are white (the value of each pixel is 255 working with a depth of 8 bit) and the remaining pixels are black (value 0). We then chain the connected edge pixels. For doing this we scan the image row by row: when we find the first pixel different from 0 we put it in a new chain, we set it to 0 so we do not consider it in the future no more and we control if one of the eight contiguous pixels is different from 0, starting from the pixel at its right and turning clockwise.

If yes, this new pixel is inserted in the chain and set to 0, we control the eight contiguous pixels and so on, until some adjacent edgel exists. When the neighbours finish, we come back to the first pixel of the chain and we repeat the same operation a second time, so we consider a possible fork. Further bifurcations will enter in a different chain. See Fig. 4 for an example.

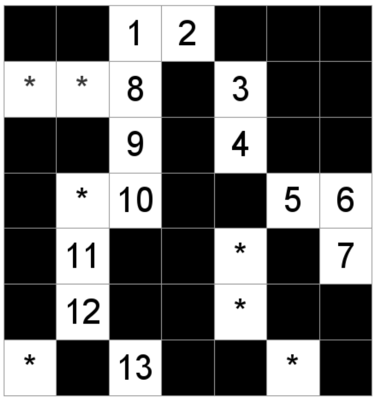


Fig. 4. An example of the chaining of the connected edgels. The numbers show the insertion order. Pixels marked with an asterisk are not inserted in the current chain, even though they are connected to the others.

We delete all the chains with a length shorter than $minPixels$. For any valid chain we store the two extremes, that can be the first pixel inserted in the chain (pixel 1 in Fig. 4), the last pixel inserted (pixel 13), or the last pixel inserted before the second propagation (pixel 7). In the previous example the extremes are pixel 7 and 13. It has been proved in [7] that the projection of a real world line onto the sphere is a great circle, has in Fig. 5.

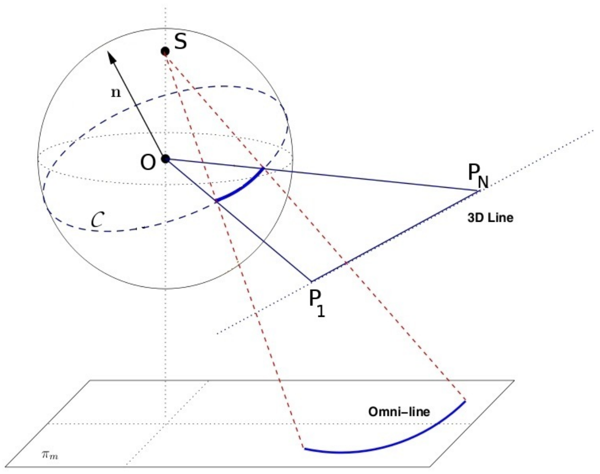


Fig. 5. The projection of a line on the sphere is a great circle (from [11].

Exploiting this property we can infer whether a chain corresponds to the projection of a world line. We project each chain on the sphere using the file that we have saved after the sensor calibration. Let $P_1 = (X_1, Y_1, Z_1)$ and $P_N = (X_N, Y_N, Z_N)$ be the two endpoints of a chain composed of N pixels. These two points define a unique great circle $\mathcal{C}$ that lie on the plane $\mathcal{P}$ passing through the sphere centre O, whose normal is $\overrightarrow{n} = \overrightarrow{OP_1} \times \overrightarrow{OP_N}$. For each point P_i of the circle we calculate its distance from the plane $\mathcal{P}$ with the formula $d = \frac{\overrightarrow{OP_i} \cdot \overrightarrow{n}}{|\overrightarrow{n}|}$. If the average of the distances of the chain points from $\mathcal{P}$ is smaller than $minDist$, the whole chain corresponds to the projection of a line in the world and we identify it with its normal. Otherwise, we split the chain into two sub-chains at the point of maximum distance from the plane and we perform the same control for each of the two sub-chains. The splitting step ends when the considered chain corresponds to a line or the chain length is smaller than $minPixel$. In [3], they suggest to use relative high values for both thresholds because the shorter the chain is, the more fragile the estimation of the great circle normal is.

As proved in [7], the great circles corresponding to a bundle of parallel lines intersect into two antipodal points on the sphere. Thus, in order to detect bundles of parallel lines we use the following algorithm. Let $\overrightarrow{n_1}$ and $\overrightarrow{n_2}$ the normals of two great circles on the sphere. Vector $\overrightarrow{u}$ passing through the intersections I_1 and I_2 of the two circumferences can be calculated with the cross product $\overrightarrow{u} = \overrightarrow{n_1} \times \overrightarrow{n_2}$. We compute the same operation for each pair of circle. If at least three lines have the same intersection points, we consider them as a bundle.

It has been proved in [5] that the direction of the vector $\overrightarrow{u}$ is the same of that of the bundle of parallel lines associated to it (Fig. 6). So all the lines in the same bundle are characterized to have the same vector $\overrightarrow{u}$.

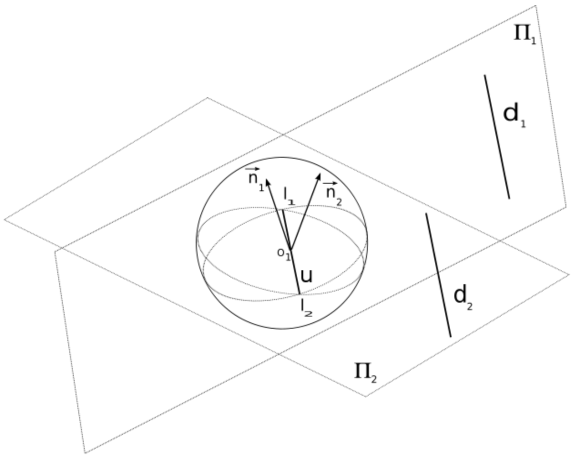


Fig. 6. The direction of two parallel lines is the same of that of the line passing through the intersection points of the corresponding circumferences (from [3]).

In our implementation, once we calculate the vanishing points of each pair of circumferences, we group them according to their position using special lists. Every list contains a reference point that is the average point of the elements inserted in that list. For each vanishing point, we calculate the distance from the reference point of each list. When we find a distance lower than a certain threshold, we put it in the corresponding list. If no suitable list exists, we create a new one.

The average point of the list with the greatest number of elements represents the vector corresponding to the principal direction. The second direction is determined by the vector orthogonal to it, belonging to the list with the greater number of elements. For efficiency reasons, the third direction is achieved making the cross product of the other two ones. So, we must detect at least two perpendicular bundles of parallel lines in the environment. It is possible in this way to define an orthogonal reference system made up of these three axes. An example of the whole procedure is shown in Fig. 7.

III. Visual gyroscope

The reference frame is not yet completed because we have not set the axes orientation. To distinguish the vertical axis from the horizontal ones, in [5] the authors propose to detect the sky supposing that it represents the brightest part close to the omnidirectional image border. We work in an indoor environment so we cannot use this approach. We avoid the problem assuming that the sensor starts to work with the robot in vertical position. Thus, in the first captured frame the software decides that the vertical axis is that, among the three calculated, at the minimum distance from the vertical vector $Z = [0, 0, 1]$.

To calculate pitch and roll angles it is sufficient to refer to this axis (labeled with V), while the two horizontal axes will be used in the next paragraph to estimate the rotation. Once we fix the initial reference frame, the visual gyroscope starts to calculate the camera slope analyzing each new captured frame. From every image we extract the three principal directions. Each new axis will replace the corresponding one calculated in the previous frame, so we can track their movements. We do not replace the old axis if the corresponding new one diverge more than a certain angle. We assume that the rotation between two consecutive frames is always lower than $45°$ for each of the three angles, that is a valid assumption with the high frame rate that we use.

The angles estimation is not made calculating the axis movement between consecutive frames, because this can lead to a sum of errors, but it is made in an absolute way comparing the last V vector with the Z axis.

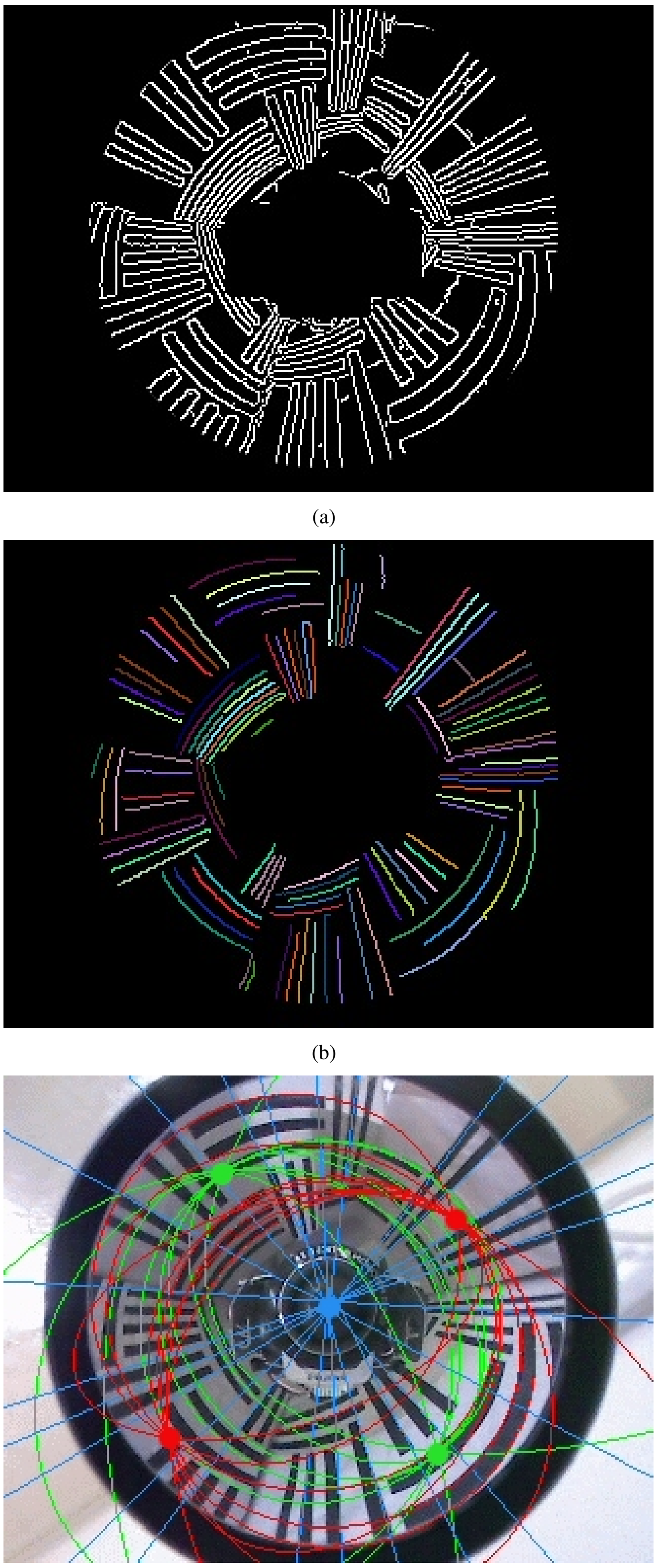
We deduce pitch ψ and roll ρ angles as follows

$$\psi = \frac{y_V}{|y_V|} \arccos\left(\frac{|z_V|}{\sqrt{y_V^2 + z_V^2}}\right), \tag{1}$$

$$\rho = \frac{-x_V}{|x_V|} \arccos\left(\frac{|z_V|}{\sqrt{x_V^2 + z_V^2}}\right). \tag{2}$$

Each new measurement is averaged with the four previous ones, so we can obtain more accurate values without high fluctuation from one frame to another.

The angles obtained with the catadioptric sensor have been compared with those provided by an inertial accelerometer. We can see in the graphic in Fig. 8 that there are no


(a)

(b)

(c)

Fig. 7. Steps for the vanishing points extraction from Fig. 3 (b): (a) detection of the edges and concatenation of the connected pixels, (b) detection of the lines that correspond to straight lines in the real world, (c) computation of the vanishing points corresponding to the three principal directions of the reference frame.

substantial differences between the two measurements. This proves the correctness of the designed visual gyroscope.

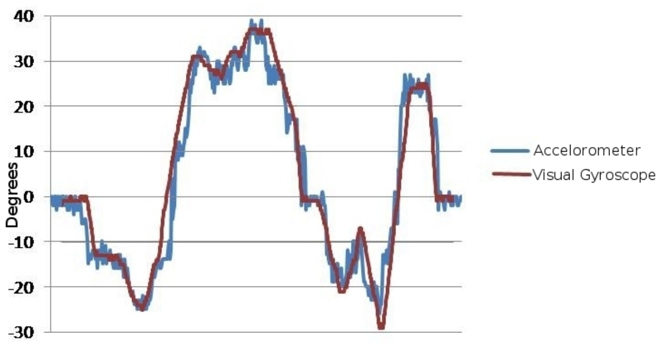


Fig. 8. Comparison between the pitch angle calculated by an inertial accelerometer and that calculated by the visual gyroscope.

A. Visual compass

The most used technique to calculate the rotation consists in tracking some particular points with RANSAC so that we can estimate the rotation angle between consecutive frames, working with omnidirectional [9] or unwarped images [12] [16], or using the optical flow like in [8] and [15].

In our work instead, we try to see if the simple vanishing points method is also valid in this scope. To build the visual compass we use the two horizontal axes of the reference frame, labeled with R_1 e R_2. It is not important which of these two axes is the first and which is the second, we are only interested to distinguish them so we can track their movements. The robot rotation will be calculated relatively to their start position. As seen before, we extract the new axes R'_1 and R'_2 from each frame. They will replace the previous ones only if the difference angle is less than a certain quantity. The rotation angle is always estimated between the initial and the last axis. In this case, the provided information is duplicated because we only need one horizontal axis to calculate the yaw angle. Knowing two horizontal axes we can apply a control method. Indeed what one can expect is that the angle given by R_1 is the same given by R_2. So, if the difference between the two angles is less than a certain angle we compute their average, otherwise one of the two vanishing points is wrong and we consider only the angle nearer to the corresponding one calculated in the previous frame. The results are good enough if the movements of the robot are on the XY plane.

B. Results in real environments

The proposed algorithm has been tested also in environments different from the test one of Fig. 3. It continues to work if it is possible to extract some parallel lines. As we can see in the following two examples, we are able to extract all the possible lines from the image. In Fig. 9 we place the robot in a RoboCup environment. All the field lines and the goalposts have been extracted and they can be used to define a three axes reference frame. In Fig. 10 instead, the robot is set on a desk in our laboratory. As we can see, this kind of environment contains several objects that define a lot of lines. Unfortunately, most of these lines are too short and represent only noise for the visual gyroscope. So we discard all the lines shorter than a certain length and preserve only the longest lines. Even if the number of detected lines is small, it is sufficient to estimate attitude and heading angles correctly, but we have to consider that the noise increases if the lines extracted are shorter.

The computation time is suitable for a real-time application since the algorithm in our C++ implementation can process about 20 frame per second.

IV. Body stabilization

We have seen how to extract the vanishing points of parallel lines from the environment and use them to estimate the slope and the rotation of the catadioptric sensor. We mount the sensor in place of the head of our robot. Since the camera is stuck to the robot, the calculated angles correspond of the slope of the robot trunk. In this way we can use this information to stabilize the robot during the walk without using any inertial sensor. It is interesting to notice that we do not stabilize only the robot's posture but also the perception. Indeed, since the robot always tries to keep its trunk in vertical position, camera oscillations are reduced and this is a great benefit for all the vision algorithms that we use during the robot walk, such as tracking and blob recognition.

A. Stabilization of the vertical position

In the first test the robot had to keep the vertical position correcting its pitch and roll angles when standing still. To do this, we correct the four servomotors of the ankles, compensating them with a quantity equal to the opposite of the angle measured in the two directions. We performed two experiments. In the first one, when the robot is pushed forward or backward, it puts up resistance rising on the tip toes or on its heels (in Fig. 11 (a) the robot is reacting to a backward push) to maintain its vertical posture. The same if the push is lateral. In the second experiment the robot stands on a platform which changes its inclination and it is able to modify its position in real time keeping it upright. In Fig. 11 (b) the robot is on an inclined plane in the position corresponding to the upright posture as it was on a horizontal surface. In Fig. 11 (c) instead we can notice how the compensation applied to the ankles brings the robot back to the vertical position.

B. Walking with stabilization

Many articles show how to stabilize a humanoid robot gait modifying the motor positions in order to maintain the Zero Moment Point inside the convex hull of the foot-support area [14]. Most of the time, walking gaits are learnt in simulation simplifying the robot model as one or two masses inverted pendulum model [13]. To pass to the real robot it is necessary an exhaustive study to establish the weight distribution and the movements of each servomotor. The simple model used in simulation must be converted to

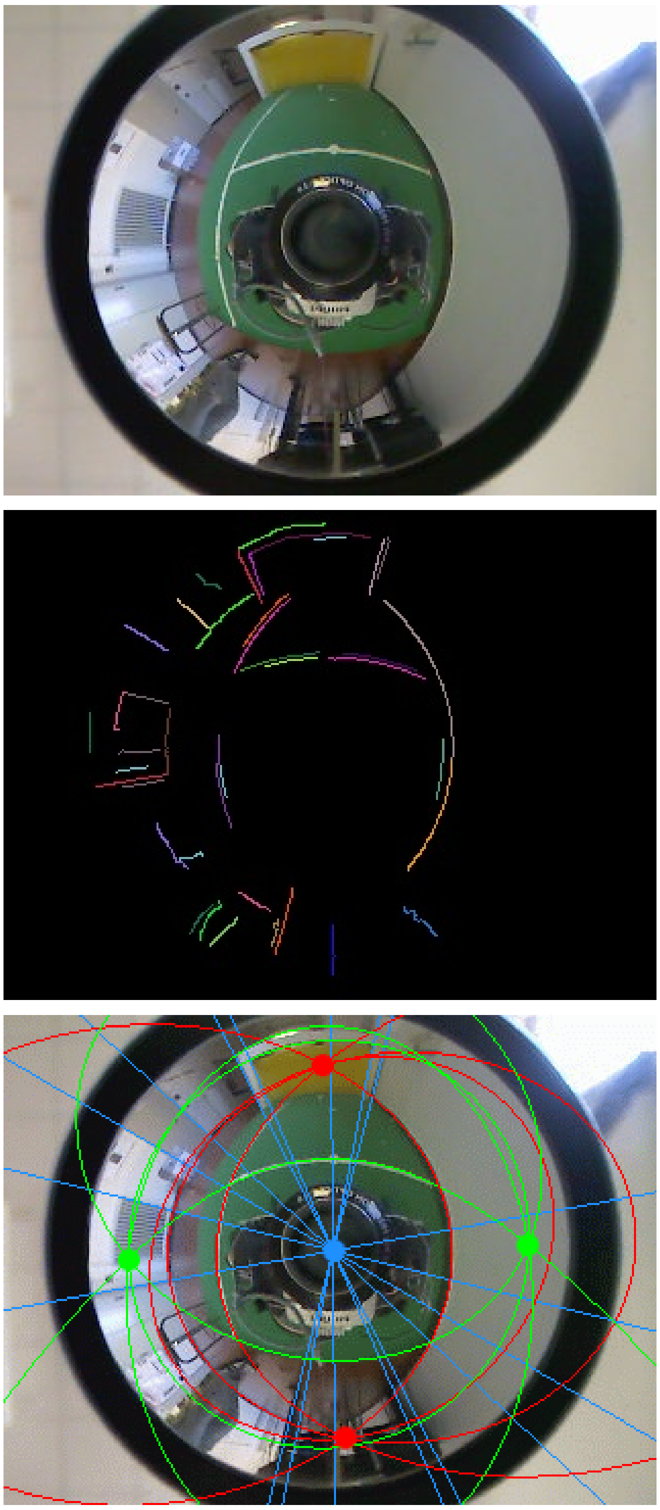


Fig. 9. Vanishing points detection in a RoboCup environment.

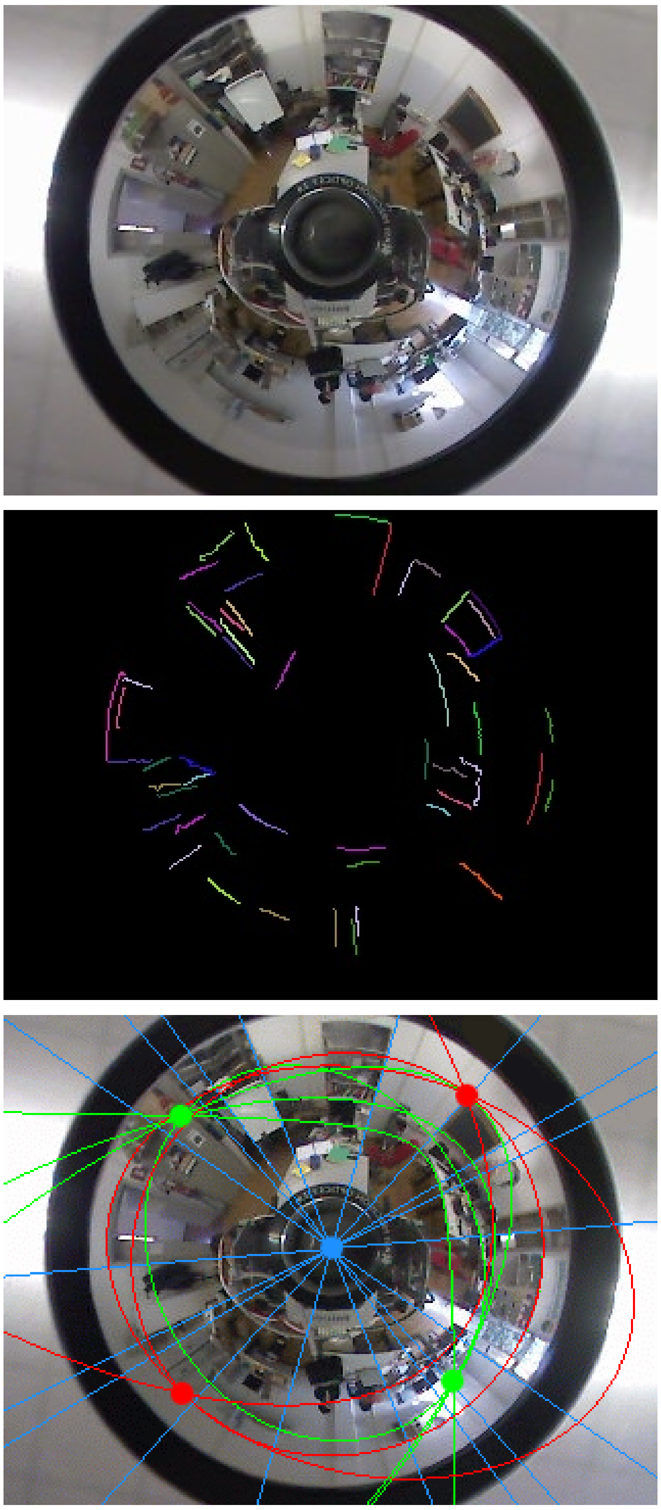
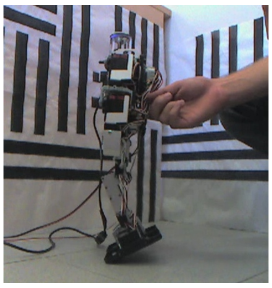
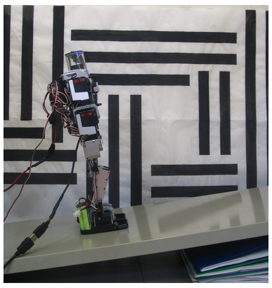


Fig. 10. Vanishing points detection in a lab environment.


(a) The robot reacts to a backward push rising on the tip toes


(b) The robot on the inclined plane without any correction

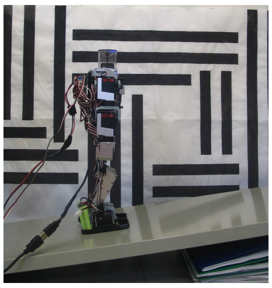
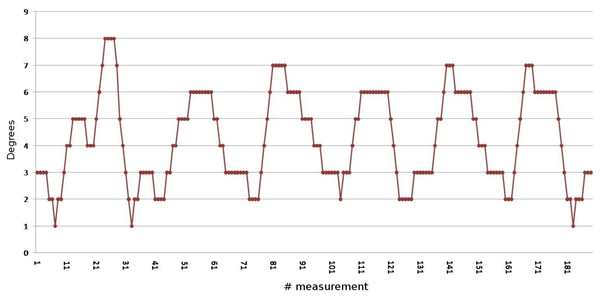
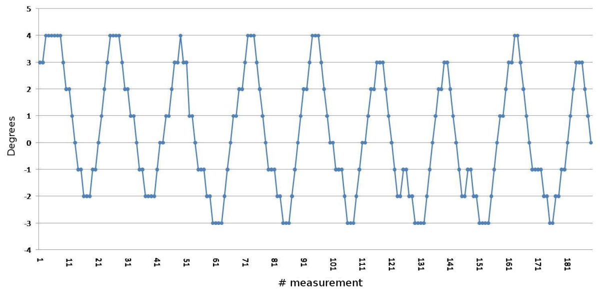

(c) The robot on the inclined plane with ankle correction

Fig. 11. Posture correction using the visual gyroscope.



(a) Pitch angle



(b) Roll angle

Fig. 12. Pitch and roll angles measured during the walk.

adapt it to the real robot, considering that complexity grows when the degrees of freedom increase.

We chose a different approach, as suggested in [1] and [10]. The idea is to create an offline walk pattern, fix minimum and maximum thresholds on the inclination that the robot can achieve during the walk and when the visual gyroscope detects an out of range value modify the robot walking gait to recover balance. One can choose to straighten up the robot, to lower its centre of mass, to slow down its movements or to stop them until it recovers its balance, or to put it in a safe position that minimize the damages in case of fall.

We have designed a static walk pattern made up of four elementary movements. We have made several experiments to measure the range of pitch and roll angles for each movement, using our visual gyroscope.

It is interesting to note the periodicity of the graphs of the two angles measured during the walk (Fig. 12). Analyzing these graphs, it is possible to realize if there is any phase of the walk in which there are anomalous oscillations and in this way correct the position of the motors.

At the moment we do not consider the roll angle because modifying the robot gait with a lateral stabilization requires more deep studies. Indeed in this situation we need to modify all the servos of both legs to ensure the right support in every phase of the walk. What we do is to use the pitch angle to control frontal inclination.

During the walk, for each of the four elementary movements we control if the pitch angle is inside the fixed range of values. If not, we change the two ankle motors responsible for the frontal slope, compensating them with an angle equal to the opposite of the difference between the current value and the nearest threshold. The same compensation is applied to the other three elementary movements. Applying a stabilization to a static walk allow to obtain a more fluid and balanced gait. The most evident results can be noticed on an inclined plane, on which the robot walks with a vertical position and it is able to modify its posture online when the terrain changes its slope (Fig. 13).

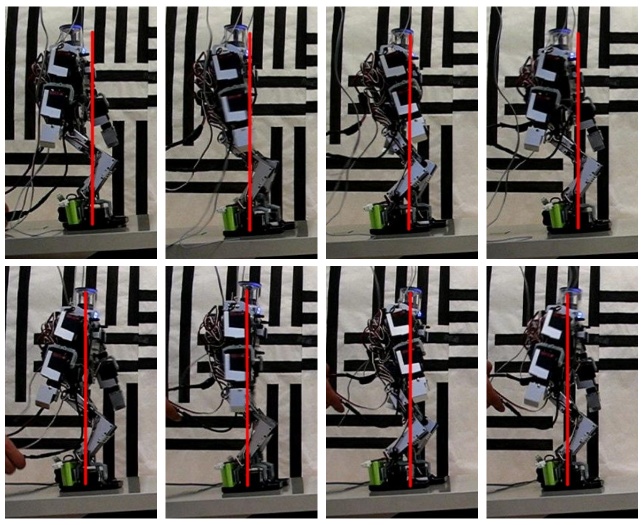


Fig. 13. Four steps of the walk. The gait above is without stabilization, while we apply stabilization in the gait below.

Two videos are available at the following links: *http://www.youtube.com/watch?v=PfuQ0VHrz1o* *http://www.youtube.com/watch?v=H2YCr9quJuk*

V. Conclusions and future works

We have presented a simplify method to use the catadioptric sensor as a visual gyroscope and a visual compass.

We have performed experiments in three different environments to show that we can extract a suitable number of parallel lines to determine the vanishing points of the three principal orthogonal directions.

The designed stabilization system, even though it is very simple, allows to observe some improvements, consisting in a better balance and in the capacity to maintain the vertical position independently of the terrain slope.

The next step is to improve the system allowing a more complex stabilization to take account of also the lateral inclination.

VI. Acknowledgments

The authors would like to thank Stefano Ghidoni for his advice and his review work.

References

[1] J. Baltes, S. McGrath, J. Anderson, *Active Balancing Using Gyroscopes for a Small Humanoid Robot*. Second International Conference on Autonomous Robots and Agents (ICARA), pag. 470-475, December 13-15, 2004, Palmerston North, New Zealand.

[2] J. P. Barreto, H. Araujo, *Geometric Properties of Central Catadioptric Line Images and Their Application in Calibration*. In "IEEE Transactions on Pattern Analysis and Machine Intelligence", Volume 27, Issue 8, pag. 1327-1333, August 2005.

[3] J. C. Bazin, I. Kweon, C. Demonceaux, P. Vasseur, *Rectangle Extraction in Catadioptric Images*. IEEE 11th International Conference on Computer Vision, pag. 1-7, 2007.

[4] J. C. Bazin, I. Kweon, C. Demonceaux, P. Vasseur, *UAV Attitude Estimation by Vanishing Points in Catadioptric Images*. IEEE International Conference on Robotics and Automation, Pasadena, CA, USA, May 19-23, 2008.

[5] C. Demonceaux, P. Vasseur, C. Pégard, *UAV Attitude Computation by Omnidirectional Vision in Urban Environment*. IEEE International Conference on Robotics and Automation, Roma, Italy, 10-14 April 2007.

[6] C. Demonceaux, P. Vasseur, C. Pégard, *Robust Attitude Estimation with Catadioptric Vision*. Proceedings of the 2006 IEEE/RSJ International Conference on Intelligent Robots and Systems, pag. 3448-3453, October 9-15, 2006, Beijing, China.

[7] C. Geyer, K. Daniilidis, *Catadioptric Projective Geometry*. International Journal of Computer Vision, Volume 45, Issue 3, pag. 223-243, 2001.

[8] J. Gluckman, S. K. Nayar, *Ego-Motion and Omnidirectional Cameras*. In Proceedings of the Sixth International Conference on Computer Vision, pag. 999-1005, 1998.

[9] G. L. Mariottini, S. Scheggi, F. Morbidi, D. Prattichizzo, *A Robust Uncalibrated Visual Compass Algorithm from Paracatadioptric Line Images*. In First Workshop on Omnidirectional Robot Vision, Lecture Notes in Computer Science, Springer-Verlag, 2009.

[10] S. McGrath, J. Baltes, J. Anderson, *Active Balancing Reflexes for Small Humanoid Robots*. In Proceedings of the 17th IFAC World Congress, Volume 17, Part 1, July 6-11, 2008, Seoul, Korea.

[11] C. Mei, E. Malis, *Fast Central Catadioptric Line Extraction, Estimation, Tracking and Structure from Motion*. In Proceedings of the 2006 IEEE/RSJ International Conference on Intelligent Robots and Systems, pag. 4774-4779, October 9-15, 2006, Beijing, China.

[12] I. F. Mondragón, P. Campoy, C. Martinez, M. Olivares, *Omnidirectional Vision applied to Unmanned Aerial Vehicles UAVs Attitude and Heading Estimation*. Preprint submitted to Robotics and Autonomous Systems, February 27, 2009.

[13] Napoleon, S. Nakaura, M. Sampei, *Balance Control Analysis of Humanoid Robot based on ZMP Feedback Control*. In Proceedings of the 2002 IEEE/RSJ International Conference on Intelligent Robots and Systems, pag. 2437- 2442, Volume 3, October 2002, Lausanne, Switzerland.

[14] V. Prahlad, G. Dip, C. Meng-Hwee, *Disturbance Rejection by Online ZMP Compensation*. In Robotica (2008), Volume 26, pag. 9-17. Cambridge University Press, United Kingdom.

[15] O. Shakernia, R. Vidal, S. Sastry, *Omnidirectional Egomotion Estimation From Back-projection Flow*. In "CVPRW Computer Vision and Pattern Recognition Workshop", volume 7, pag. 82. June 2003.

[16] I. Stratmann, *Omnidirectional Optical Flow and Visual Motion Detection for Autonomous Robot Navigation*. PhD Thesis, Universität Osnabrück. October 2007.

[17] P. Vasseur, E. M. Mouaddib, *Central Catadioptric Line Detection*. In Fifteenth British Machine Vision Conference (BMVC), Kingston University, London, September 7-9, 2004.

[18] J. A. Walraven, *Failure Mechanisms in MEMS*, Internantional Test Conference, pag. 828-833, 2003.

[19] X. Ying, Z. Hu, *Catadioptric Line Features Detection using Hough Transform*. In Proc. of International Conference on Pattern Recognition (ICPR04), Cambridge, United Kingdom, pp. 839-842, 2004.

[20] Kondo Kagaku Co. URL: *http://www.kondo-robot.com/*

[21] oCamCalib, tutorial e download.
http://asl.epfl.ch/∼scaramuz/research/Davide_Scaramuzza_files/Research/OcamCalib_Tutorial.htm