Detection of Aerial Balls Using a Kinect Sensor

Paulo Dias, João Silva, Rafael Castro, and António J.R. Neves(✉)

IRIS Group, DETI / IEETA, University of Aveiro,
3810–193 Aveiro, Portugal
{paulo.dias,joao.m.silva,rafaelcastro,an}@ua.pt

Abstract. Detection of objects in the air is a difficult problem to tackle given the dynamics and speed of a flying object. The problem is even more difficult when considering a non-controlled environment where the predominance of a given color is not guaranteed, and/or when the vision system is located on a moving platform. As an example, most of the Middle Size League teams in RoboCup competition detect the objects in the environment using an omni directional camera that only detects the ball when in the ground, and losing any precise information of the ball position when in the air. In this paper we present a first approach towards the detection of a ball flying using a Kinect camera as sensor. The approach only uses 3D data and does not consider, at this time, any additional intensity information. The objective at this stage is to evaluate how useful is the use of 3D information in the Middle Size League context. A simple algorithm to detect a flying ball and evaluate its trajectory was implemented and preliminary results are presented.

1 Introduction

To our knowledge, most of the RoboCup teams of the Middle Size League (MSL) have limited vision systems regarding the detection of the ball when in it is in flight. Most teams use an omni directional camera on top of the robots that only detects correctly ball position when on the ground since they use projective geometry and a single camera.

Given that most of the robots shoot the ball through the air, the possibility to detect the ball when in flight is very relevant. Obvious solutions using more than a single camera (either using two additional cameras to provide stereo vision, or combining the information from the omni directional camera with additional cameras) can be considered as in [1–3]. However they present some limitations: first these additional cameras may point outside the field and cannot use background or color information to simplify ball segmentation (since a flying ball might be in the air with any possible backgrounds - tribunes, chairs, spectators, etc.) or might have limited field of view (most omni directional camera point downwards, meaning a maximum height of around 60 cm).

In a previous work [4], we developed algorithms based on color and shape detection using a single perspective camera. In this work, the above mentioned problems were detected. The work presented in this paper uses a different vision approach based on a depth sensor instead of an intensity sensor. This work

R.A.C. Bianchi et al. (Eds.): RoboCup 2014, LNAI 8992, pp. 537–548, 2015.
DOI: 10.1007/978-3-319-18615-3_44

presents several similarities with the work of Khandelwal et al. [5] that uses a Kinect sensors as a low cost ground truth detection system. As 3D sensor, a Kinect was chosen given its low price (making it possible to use in an aggressive environment such as RoboCup), its ability to directly provide 3D depth information and its refresh rate of 30 fps similar to the RGB camera used in the onmi direction vision system of the robots [6].

2 3D Data

Kinect is a motion sensing input device developed by Microsoft and launched on November 2010. The sensor includes an RGB camera, an infrared laser projector, a monochrome CMOS sensor, and other components less relevant for our application. The field of view is 57 degrees horizontally, and around 43 degrees vertically. Acquisition of the 3D data from the Kinect is done using a C++ wrapper for libfreenect that transforms depth images into an OpenCV matrix. For an initial stage of the application, ROS was used to access directly the 3D cloud of points. However, the time needed for processing and transmitting the 3D data was large and it was verified that direct access to the 2D depth data was faster. Besides, the CAMBADA team code structure is not ROS based, which would ultimately become an issue. The transformation from raw data to metric is done using a formula found in an online manual [7].

In its original configuration, Kinect normal working range is from 0.8 m to 4 m. However, the sensor provides distance measurements for longer distances while suffering from additional errors and loss of precision which causes a discretization effect to appears with distance increase. Figure 1 presents views of a typical 3D cloud of points with texture as provided by a Kinect. Two of the main limitations of the Kinect for this kind of application are clearly visible in this figure. For larger distances a discretization of space occurs turning the cloud of points into several parallel planes. Also, given the speed of the ball a trail of points appears around the flying objects as shadow points that do not exist and results from averaging between points in the object and in the rear wall. This effect is more significant for higher speeds of the objects and is well known in 3D data when jump edges occur resulting in a mixed distance between the foreground and the background object.

3 Ball Detection Algorithm

Despite the limitations inherent from the discretization of the space, more visible at higher distances, the use of 3D information still seems to be a good option.

3.1 Flying Objects

A first approach to ball detection using the Kinect cloud of points as a source would be to use geometry, for example fitting half spheres to the data and

Fig. 1. Views of colored clouds of points acquired with a kinect (Color figure online).

trying to find areas of interest. Preliminary trials have been performed with this approach but, given the discretization effect, this fitting only appeared to provide reliable data when close to the sensor (within the typical Kinect working range, less than 4 m). For longer distance the spherical shape of the ball becomes difficult to detect.

The approach used in this paper is to detect flying objects within the Kinect field of view. To achieve this, given the properties of a flying ball in the MSL environment, we decided to voxelize the space to work in an occupancy voxel space rather than considering the while cloud of points.

Figure 2 shows the results of the voxelization of a cloud of points for a grid size of 0.05 m, value obtained experimentally in order to maximize the detection performance. Empty voxels are not presented and adjacent voxels are presented with different colors and transparency to ease visualization.

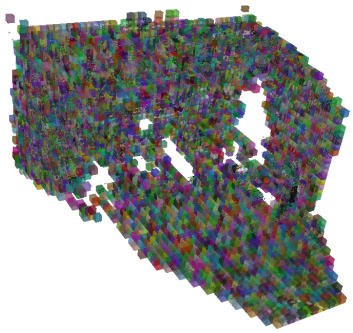

Fig. 2. Voxelization of a cloud of points for a grid size of 0.05 m.

This step, besides allowing an increase of the process speed by reducing computation, also allows us to define a flying object as an object that occupies a given number of voxels with a minimum number of points and whose surrounding voxels are empty. It can be seen as a 3D mask inside voxels non empty (containing a minimum number of points) and the outside voxels empty. Figure 3 presents an example of such a mask. Inside voxels (occupied) are represented with one color and border voxels (empty ones) are represented with another color.

The values used for the grid and the mask obviously depend on the size of the ball to detect. However, they have to be defined taking into consideration two main aspects: *(1)* the grid size must be large enough to allow that a real flying ball, when voxelized, does not become smaller than the space between any two planes. This issue becomes more hazardous at farther distances; *(2)* the grid mask must be large enough to accommodate a volume larger than the ball, since some blurring is inevitable due to the high speeds achieved by a ball.

With this mask approach, we expect to rule out false positives from any other object on the field of play, since all other artifacts during a game can only be a robot or a human. Since all of them have a clear "connection" with the ground, the mask will not allow a valid detection.

Also, any object that could effectively be identified by the mask as a ball, at this point, could be outside the field of play. Since this is a complementary vision system for our robots and since they know their position inside the field of play, further integration steps will be responsible for handling these possible false positives.

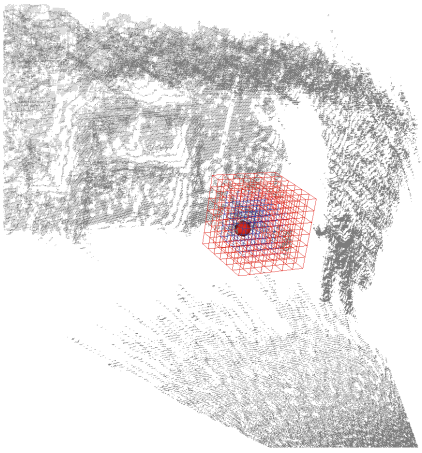

Fig. 3. Example of 3D mask used for flying object identification (Color figure online).

In our preliminary tests in a field with limited range and a wall at 7 m, we empirically set the following values for the grid and mask sizes: the grid size of the occupancy grid is 0.27 m, and the mask size (corresponding to the outside of the mask) is 5. As it is, and with our test scenario, the resulting mask size is around 1.35 m meaning that a flying object must be at least 0.27 m away from any other structure to be detected. These values were used to avoid wrong detection related to objects close to the wall at 7 m, where the discretization effect of the Kinect is significant. In Fig. 4 we present the ball correctly detected in three consecutive points corresponding to a kick away from the sensor.

3.2 Ground Objects

The algorithm presented in the previous section is only suitable to detect flying balls, but it can be easily adapted to detect ground balls by ignoring the bottom part of the 3D mask for balls lying on the ground. To apply this idea it is necessary to know where the ground is and apply a different 3D mask for voxels lying above the ground.

To allow for this additional detection, we align the grid with the ground by introducing an additional step in the first processed image. For the first acquired image, the ground plane is detected using Random Sample Consensus Algorithm (RANSAC) already available in the PCL package [8]. Given the transformation between the plane and the original coordinate reference, we compute the Rigid Body Transform between the original coordinate frame and the ground plane and apply this transformation to every point cloud ensuring that the XZ axis will be aligned with the ground plane. Computing the voxelization in this new coordinate frame ensure that the grid cells corresponding to $y = 0$ are containing the ground plane (Fig. 5).

For optimizing the process and since usually the height of the sensor will not change during acquisition, the rigid body transform between the original

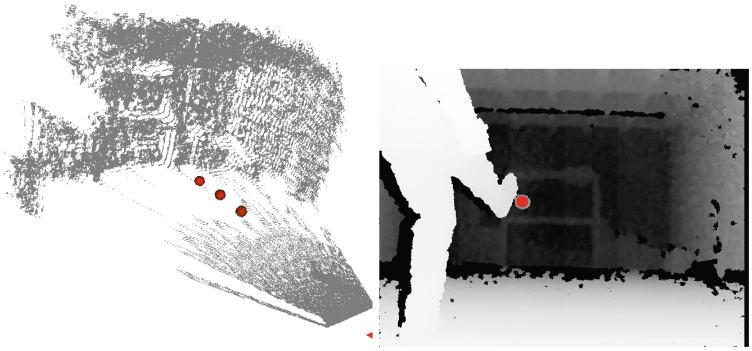

Fig. 4. The ball correctly detected in three different moments during a kick away from the sensor.

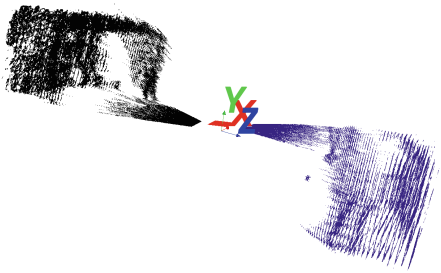

Fig. 5. 3D cloud of points in original Kinect coordinates (light blue), detected ground plane (red square), 3D cloud in new coordinate system, with ground plane aligned with xOz plane (Dark) (Color figure online).

coordinate frame and the ground is computed only once in the first image (since the RANSAC plane fitting is quite heavy) but any following point cloud is transformed to align the ground plane with the XY coordinate plane. The final algorithm will be exactly the same but will use a given mask for flying objects when $y > 1$ and the ground mask when $y = 1$.

Figure 6 shows the results of ground ball detection in 4 consecutive images corresponding to a kick away from the user with the ball on the field.

The general algorithm to detect ground and flying ball is the following:

Compute Ground Plane from first image
Compute transform to align ground with XZ plane
for *Each point cloud* **do**
    Align point cloud with XZ plane
    Voxelization of grid
    Detection of flying ball $y > 1$ *(use flying object mask)*
    if *no flying ball* **then**

Detect ground ball y=1 (use ground mask mask that ignores bottom part of flying mask)
end if
end for

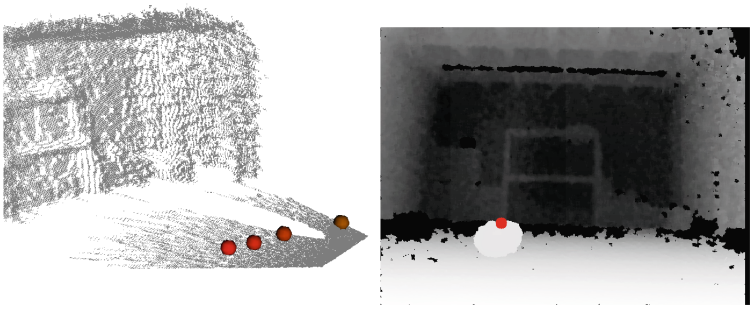

Fig. 6. Example of detection of ground ball in 4 consecutive 3D images.

4 Trajectory Estimation

Besides detecting the ball in the environment, it is important for the robot to estimate the ball trajectory in order to predict the best action to take. In robotic soccer, the algorithm for trajectory estimation presented in this section is useful so that the goalkeeper can move to a position in order to prevent a goal.

For flying objects, and considering that air resistance is negligible, the trajectory can be approximate by a simple ballistic trajectory. To perform this evaluation, we keep trace of the last ball positions. Currently, 10 previous position are kept since it is enough for most of the flying movements detected.

The trajectory estimation algorithm computes the 2D vector (x,y) between the actual ball and the initial point of the trajectory (the first ball detected that supports the actual trajectory). It then used this vector norms to estimate the point on the trajectory (according to the current estimated ballistic) at the same distance. If the Euclidean between the original ball position and the point on the trajectory is below a threshold (empirically set to 0.25 m in the current experiments) the position is considered as supporting the current trajectory.

Given a number of points supporting the trajectory, Singular Value Decomposition (SVD) is used to compute the parabolic equation that best fits the whole supporting coordinates. The algorithm used is the one implemented in the Eigen Library [9].

The general algorithm for trajectory estimation is as follow.

Update history
if *ball detected* **then**
if *new position support previous trajectory* **then**
Compute trajectory with all points

```
    else
      Compute new trajectory with last 3 points
    end if
    if If trajectory error below threshold then
      Update new trajectory
    end if
end if
if NOT(last 2 positions exist and support trajectory) then
    reset trajectory
end if
```

Figure 7 shows the results of the trajectory estimation for a flying ball. The history of balls used for the computation is presented as large red spheres and the parabolic trajectory estimated is represented by the small blue spheres.

With the trajectory estimated, the agent can use the current projection of the ball position on the ground and even the predicted touchdown point of the ball.

5 Kinect Position Calibration on the Robot

Since the objective is to use the 3D position detection and trajectory algorithm on board of a robotic soccer robot, calibration of the position of the Kinect relative to its position on a robot must be performed. The coordinates of the detected ball on the field coordinate frame can then be obtained easily by applying the transformation from Kinect to the robot and then from robot to world position (this transformation comes from robot positioning routines). To allow flexibility and avoid any calibration based on measurement, a calibration program was developed. The calibration program (see Fig. 8) acquires on demand an image from Kinect. It then allows to automatically detect a ball on the scene using the ground detection algorithm. Alternatively, the user can also pick the center of the ball directly in the 3D cloud of points. The process must be repeated for at least three ball positions.

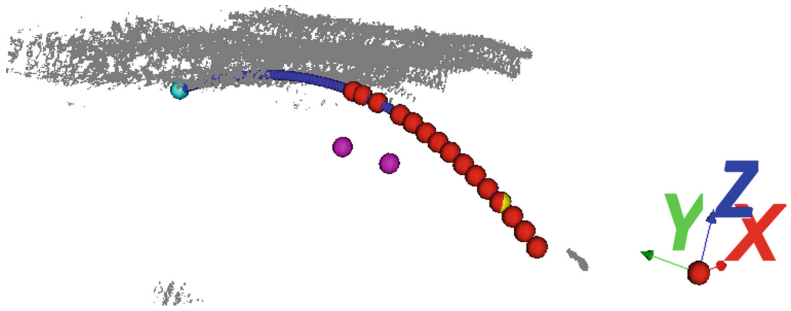

Fig. 7. Example of trajectory estimation: small blue spheres represent the trajectory computed from the balls in the history (large red spheres) (Color figure online).

In the left table, the coordinate of the detected/selected points in the robot coordinates must be provided. Then the software evaluate the rigid body transform between the 2 coordinates system corresponding to the Kinect position and orientation relatively to the robot origin. This matrix is saved and must be read by the 3D detection program for a given configuration of the Kinect. A simple calibration process consist in positioning the robot in the center/origin of the field, and then acquire 3D images with the ball on well-defined landmarks of the field. A typical configuration of the calibration procedure is presented in Fig. 9.

6 Experimental Results

The work presented in this paper is still in an early stage of development. However, the first results obtained seems very promissing. At this stage, the validation of

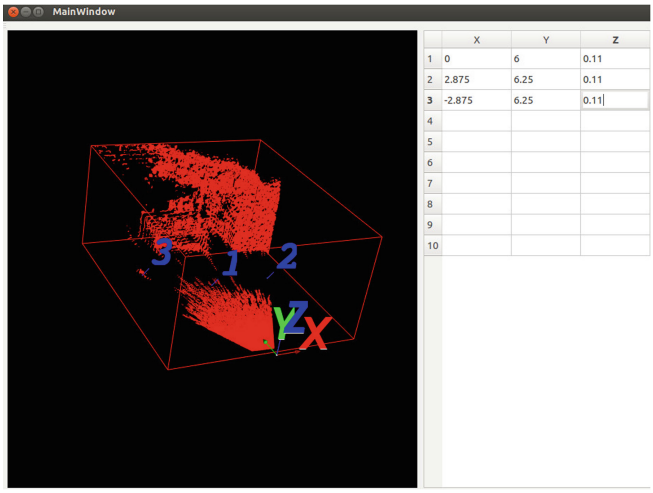

Fig. 8. Application for kinect position calibration with 3 reference points on the kinect cloud and the corresponding coordinate in robot coordinates.

Fig. 9. Example of kinect calibration set-up with a ball on a well known position.

the algorithm was performed by observation of the results using visual tools (see Figs. 4, 6 and 7). Based on these visual observations, the algorithm seems quite robust in open field (with no obstacles) and processing times are below 30 ms meaning that all the Kinect 30 images per second are processed. Some tests were performed with the Kinect on the top of a real robot. Figure 10 shows an example of the ball detection from the point of the view of the robot.

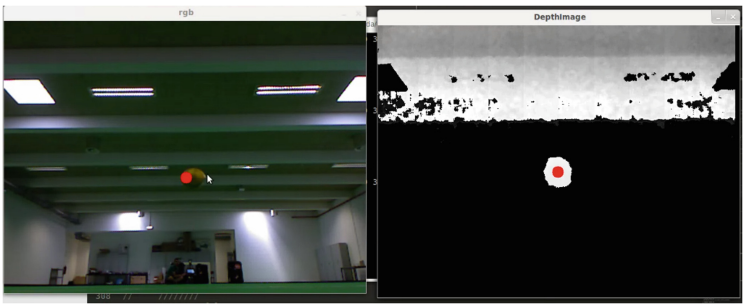

Fig. 10. Example of a ball detection from the point of view from a robot.

In this configuration the robot evaluated the ball position on several flying ball tests and provide feedback on the base station of the estimated intersection point (projecting the 3D position on the ground and computing the closest point to the line). Visual observation on the base station was according to the expected as the robot movement was toward the correct direction.

Table 1 and Fig. 11 presents experimental results regarding the position of the ball detected using the proposed algorithm. In this experiment, the ball was fixed by a thin wire at 1 m high in the front of the robot. Several distances were tested, from 2 to 6 m. As we can see, the error in position is small enough to allow the use of the proposed algorithm by the robots during a game.

Table 1. Ball position evaluation regarding the proposed vision system. The ball was fixed by a thin wire at 1 m high in the front of the robot ($x = 0$) at several distances (y).

Distance x/y (cm)	Average x/y (cm)	Std deviation x/y (cm)
0/200	-0.08/2.01	0.004/0.002
0/300	-0.11/2.96	0.007/0.029
0/400	-0.07/4.01	0.005/0.001
0/500	-0.06/5.00	0.169/0.718
0/600	0.01/5.97	0.002/0.003

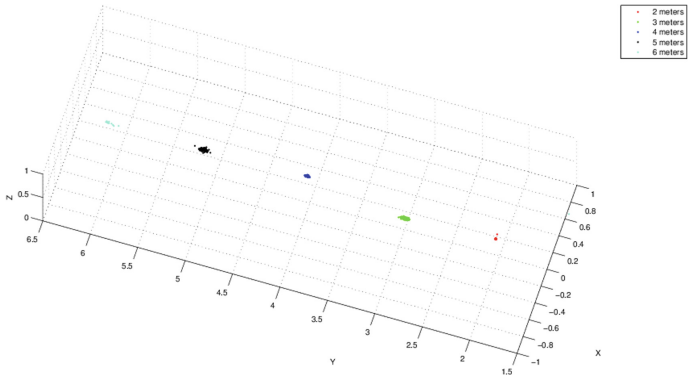

Fig. 11. Ball position detected by the proposed vision system with the kinect on the robot. The ball was fixed by a thin wire at 1 m high in the front of the robot ($x = 0$) at several distances (y).

7 Conclusion and Future Work

In this paper we present preliminary work toward the detection of 3D flying balls using only the depth information provided by a Kinect. The final objective is to use the algorithms presented here to extend the current vision of our MSL robotic agent to cope with 3D object positions. An algorithm based on an occupancy grid was developed. Preliminary results are encouraging since the algorithm can process the Kinect data in real time (processing at 30 fps) and the visual inspection of the detection is quite convincing.

Regarding future work, the first step is the fully integration of the Kinect on the platform to allow real experiments during a game, in the presence of other objects than the ball. Ball validation using the RGB image is also under development, in order to validate any false positive that can occur.

Aknowledgements. This work was developed in the Institute of Electronic and Informatic Engineering of University of Aveiro and was partially supported by FEDER through the Operational Program Competitiveness Factors - COMPETE and by National Funds through FCT - Foundation for Science and Technology in a context of a project FCOMP-01-0124-FEDER-022682 (FCT reference PEst-C/EEI/UI0127/2011).

References

1. Ahmad, A., Lima, P.: Multi-robot cooperative spherical-object tracking in 3D space based on particle filters. Robot. Auton. Syst. **61**(10), 1084–1093 (2013)
2. Voigtlnder, A., Lange, S., Lauer, M., Riedmiller, M.: Real-time 3D ball recognition using perspective and catadioptric cameras (2007)
3. Scaramuzza, D., Pagnottelli, S., Valigi, P.: Ball detection and predictive ball following based on a stereoscopic vision system. In: IEEE International Conference on Robotics and Automation, pp. 1561–1566 (2005)

4. Silva, J., Antunes, M., Lau, N., Neves, A.J.R., Lopes, L.S.: Aerial ball perception based on the use of a single perspective camera. In: Reis, L.P., Correia, L., Cascalho, J. (eds.) EPIA 2013. LNCS, vol. 8154, pp. 235–246. Springer, Heidelberg (2013)
5. Khandelwal, P., Stone, P.: A low cost ground truth detection system for RoboCup using the kinect. In: Röfer, T., Mayer, N.M., Savage, J., Saranlı, U. (eds.) RoboCup 2011. LNCS, vol. 7416, pp. 515–527. Springer, Heidelberg (2012)
6. Neves, A.J.R., Pinho, A.J., Martins, D.A., Cunha, B.: An efficient omnidirectional vision system for soccer robots: from calibration to object detection. Mechatronics **21**(2), 399–410 (2011)
7. Burrus, N.: Kinect calibration, consulted in 2013/2014
8. Rusu, R.B., Cousins, S.: 3D is here: point cloud library (PCL). In: IEEE International Conference on Robotics and Automation (ICRA), Shanghai, 9–13 May 2011
9. Guennebaud, G., Jacob, B., et al.: Eigen v3 (2010). http://eigen.tuxfamily.org