Calibration of Inverse Perspective Mapping for a Humanoid Robot*

Francisco Bruno Dias Ribeiro da Silva[1][0009−0008−2410−0045], Marcos Ricardo Omena de Albuquerque Máximo[1][0000−0001−5375−1076], Takashi Yoneyama[2][0000−0003−2944−4476], Davi Herculano Vasconcelos Barroso[1], and Rodrigo Tanaka Aki[1]

[1] Autonomous Computational Systems Lab (LAB-SCA), Computer Science Division, Aeronautics Institute of Technology, 12.228-900, São José dos Campos, SP, Brazil.
[2] Electronic Engineering Division, Aeronautics Institute of Technology, 12228-900, São José dos Campos, SP, Brazil.

Abstract. This paper proposes a method to calibrate the model used for inverse perspective mapping of humanoid robots. It aims at providing a reliable way to determine the robot's position given the known objects around it. The position of the objects can be calculated using coordinate transforms applied to the data from the robot's vision device. Those transforms are dependent on the robot's joint angles (such as knee, hip) and the length of some components (e.g. torso, thighs, calves). In practice, because of the sensitivity of the transforms with respect to the inaccuracies of the mechanical data, this calculation may yield errors that make it inadequate for the purpose of determining the objects' positions. The proposed method reduces those errors using an optimization algorithm that can find offsets that can compensate those mechanical inaccuracies. Using this method, a kid-sized humanoid robot was able to determine the position of objects up to 2 meters away from the itself with an average of 3.4 cm of error.

Keywords: robotics · humanoid robot · inverse perspective mapping · computer vision · calibration.

1 Introduction

ITAndroids is a team of students from the *Aeronautics Institute of Technology* (ITA) that participates in national and international robotics competitions. One of the leagues in which ITAndroids participates is Humanoid KidSize, which is a league of autonomous robots with a human-like body structure and senses that play soccer against each other [1]. This task involves some complex challenges, such as building and controlling the robot, as well as programming its decision making abilities.

* Supported by CNPQ (Conselho Nacional de Desenvolvimento Científico e Tecnológico).

One of these challenges is the robot's autonomous localization on the field with their human-like sensors. This is especially important, as knowing their position on the field is necessary to take a variety of actions, such as determining whether they are getting out of bounds of the field or attacking the correct goal (a field is symmetric, so a robot without its localization cannot discern between his and the opponent's goal). In a higher level, the knowledge of their position is also of paramount importance in planning and adoption of complex attack and defence tactics.

On the Humanoid League, with exception of some match commands (like start the match, foul and end the match), the robot is not allowed to receive information from an external computer. Therefore, the robots need to process all the information they need to play is a manner analogous to human players, including finding their own position.

Notice that solving the problem of finding the robot's position cannot be done only with the image of their camera (the field they play is, theoretically, symmetrical). However, given an algorithm that estimates well the position of the robot relative to some field marks it sees on camera, we can use a Monte Carlo Localization technique to determine the robot's pose in the field, as described in [2] (Here, field marks refer to the intersection of field lines, as can be seen in Fig. 1).

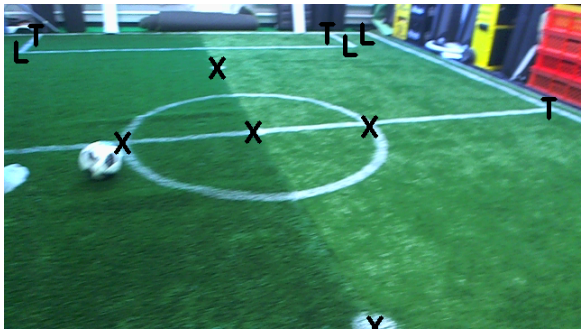


Fig. 1: Image with annotated field markers. The field marks are the intersections between field lines. This image was generated thanks to [3].

The algorithm that is used to perform the estimation of the robot position relative to the field marks it sees on camera is the inverse perspective mapping algorithm. Section 2 describes how this algorithm is used to obtain the position of objects in a plane (in our case, the plane is the playing field and the objects are the field marks) (for futher reading see [4]). Having a good estimation of the robot's position relative to some field marks we can use [2] to solve the localization problem. This is the standard approach used by teams in Robocup Humanoid KidSize competition.

However, this process is not easy to do in practice, as there is error involved in the inverse perspective mapping calculations, which can make the objects' positions estimation inadequate for the Monte Carlo Localization. Particularly, the ITAndroids humanoid robot uses the inverse perspective mapping algorithm to calculate its position relative to objects seen on camera. This process achieves good results on simulations, yet, does not perform well in real life, as the vector transformations involved in the calculations are sensitive to minor mechanical inaccuracies (for instance, a joint with a minor angle offset can output a major position difference). So, we need a calibration process capable of eliminating those mechanical errors in the calculations.

This paper contributes by developing a method to calibrate a humanoid robot inverse perspective mapping algorithm. The ideas here can be applied to other domains in which a robot performs a inverse perspective mapping to determine the position of objects relative to itself. The method developed here is based on *guessing* offsets on some of the robot's joints angles until the calculated positions of known objects are in agreement with their actual positions.

The remaining of this paper is organized as follows. Section 2 explains inverse perspective mapping (IPM). Section 3 presents detailed description of the calibration method and why it is done the way it is, so the reader can adapt our solution to their needs. Section 4 presents numerical and visual data of the results of our calibration method in a humanoid kid sized robot. Finally, Section 5 concludes and shares our ideas for future work.

2 Inverse Perspective Mapping

Inverse perspective mapping (IPM) is concerned with determining the 3D world position of an object seen in an image. In this context, the well-known pinhole camera model dictates that the perspective projection of a point $[x\ y\ z]^T$ represented in the world coordinate system yields an image point $[u\ v]^T$ [9] given by

$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = K \begin{bmatrix} R\ t \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}, \tag{1}$$

where K and $[R\ t]$ are the intrinsic and extrinsic matrices, respectively. These matrices are defined as

$$K = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}, \tag{2}$$

$$\begin{bmatrix} R\ t \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix}, \tag{3}$$

where f_x and f_y are the focal lengths expressed in pixel units, and (c_x, c_y) is the principal point. The parameters $r_{ij}, \forall i, j \in \{1, 2, 3\}$ and $t_k, \forall k \in \{1, 2, 3\}$ encode the camera's pose w.r.t. the world coordinate system.

The intrinsic matrix depends solely on the camera and the lenses, but not on the camera's pose, therefore its parameters can be determined once by using information provided in the camera's data sheet or by a calibration process usually provided by most computer vision libraries, such as OpenCV [5]. For a mobile robot, the extrinsic matrix is always changing with the robot's motion. In our case of a humanoid robot, the extrinsic matrix may be computed using the joint positions and kinematic chain information.

There are many issues that prevent a precise kinematic model, such as manufacturing imprecision, and joint flexibility and backlash. Moreover, due to communication bandwidth limits, we are unable to read all the servo positions within one walking control cycle, so we need to use expected joint positions for the leg's joints.

These errors propagate through the kinematic chain, making the extrinsic matrix estimate very imprecise. To mitigate these effects, we compute the camera's translation using only the joint positions, but use the torso's orientation estimate together with the positions of the neck joints measured by the servos' encoders for the camera's rotation. The torso's orientation is estimated by an Extended Kalman Filter (EKF) running on IMU data [6]. Unfortunately, this is not enough for an adequate estimate, since even small angular errors can be very harmful to distances estimated by IPM. Furthermore, there is also error in IMU alignment.

Notice that (1) shows how to project a world point to the image. IPM deals with the inverse problem, but there are infinite world points which are mapped to the same image point during perspective mapping. In humanoid robot soccer, objects seldom leave the ground, so to avoid this ambiguity, a common approach is to consider that the seen object is on the ground. Therefore, assuming that the object is at a constant height $z = h$ above the ground, we may write

$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = K \begin{bmatrix} R \; t \end{bmatrix} \begin{bmatrix} x \\ y \\ h \\ 1 \end{bmatrix}, \tag{4}$$

which is a system of linear equations with x and y as variables. Then, Equation (4) may be solved analytically to yield the world position $[x \; y \; h]^T$ of a given pixel $[u \; v]^T$. In our system, we use $h = 0$ for field features and goalposts (we use the intersection of the goalpost with the ground as feature) and $h = r$ for the ball, where r is the radius of the ball.

3 Calibration Methodology

The robot IPM model calibration can be divided in two parts. In the first, it uses its camera to get images with ArUco [7] markers and saves the position of each detected ArUco in a file. In the second part, it passes that data to an algorithm that estimates the offsets in some of the robot's coordinate transforms.

3.1 Data Collection

In the first part of the calibration process, the robot collects data from objects in known positions. To do that, we used ArUco markers placed in known positions. An ArUco marker is a synthetic square marker composed by a wide black border and an inner binary matrix which determines its identifier (id). The black border facilitates its fast detection in the image and the binary codification allows its identification and the application of error detection and correction techniques [7]. Examples of such markers can be seen in Fig. 2.

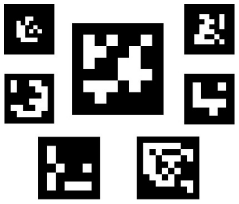


Fig. 2: Example of 7 distinct ArUco markers [7].

The operation principle of an ArUco marker is similar to that of a QR-Code. The choice for ArUco instead of the most familiar QR-Code is based on its ease of detection [7], allowing the robot to see them further away, yielding better results.

For the calibration with ArUco markers in known positions, we developed a carpet (see Fig. 3) with 120 markers in known positions. This carpet has different size markers, according to the robot's distance detection capability. Its size (3 m by 1.5 m) allows for a good area of calibration without making it difficult to be transported, stored or set up. This approach was inspired by the solutions other teams have developed, such as Rhoban [13] and MRL [14].

During calibration, the robot is placed in the bottom center of the carpet, with its feet's center aligned with the carpet side, as Fig. 4. Then, the robot obtains images of the carpet with its neck in different angles, in such a way to map the carpet entirely. Those images are processed by an ArUco detection algorithm. Every time the robot detects some ArUco, the information necessary to perform the IPM is saved, together with the position of every detection on the image and the real positions of the ArUcos. This information is saved in a file which is later read by the calibration algorithm.

3.2 Calibration algorithm

Comparing the calculated positions of each ArUco markers and their actual positions we notice substantial errors (see Fig. 5). That happens due to the robot's mechanical inaccuracies. To fix those inaccuracies we set offsets for the angles of some of the robot's coordinate transforms. We choose the angles that

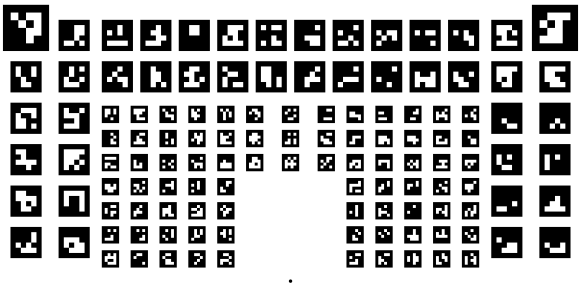


Fig. 3: Carpet with ArUcos that is used during a calibration. The real carpet is 3 m by 1.5 m.

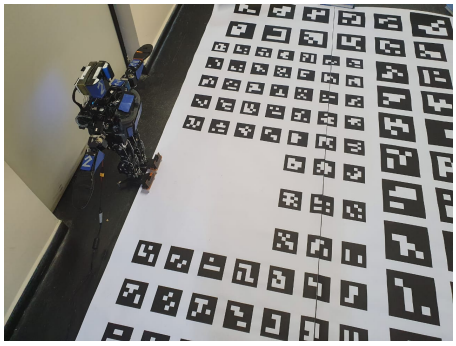


Fig. 4: Robot collecting data for calibration.

affect the calculation of the ArUco's positions, and they are: the three rotational degrees of freedom of the neck and the roll and pitch of the torso. The yaw of the torso was not taken into account, as the way the legs are built does not allow significant displacement in that direction. Also, length offsets were ignored. The calibration considering length offsets in the torso, legs and neck was also done, with small optimization difference. This calibration happens just before a match and must be performed as fast as possible. Adding length offsets doesn't increase the cost of computing (5), but takes longer for the optimizer to converge due to additional dimensions. Thus we conclude that adding those offsets is not worth the longer runtime.

The purpose of the calibration is to find the offsets that make the calculated positions of the ArUcos as close as possible to their actual positions. Considering that those offsets make the calculated positions sufficiently close to the real positions, we can assure that any object within a carpet of distance to the robot (up to 1.5 m in front and 1.5 m to the side) will have its distance calculated precisely, as it is equivalent to the robot detecting an ArUco marker in the same position.

Notice that, even though an area of 3 m by 1.5 m seems tiny, in comparison to a person, the Humanoid Kidsize League has robots between 40 cm and 90 cm and a field with length 9 m by 6 m of width, according to the RoboCup rules [8]. Thus, given a successful calibration, we can assure precise distance calculation in, at least, an area that corresponds to $(3 \cdot 1.5)/(9 \cdot 6) = 8.33\%$ of the area of the entire field. The calibration also generalizes well for positions out of this area because of how the IPM algorithm works. So, in practice, we have precise distance estimation for objects even out of the training area (here training area refers to the carpet area).

To find the best offsets to calculate the position of an object, we need an expression to evaluate the quality of each set of offsets. Let x be a vector containing the chosen offsets, $\mathcal{D}$ all the data collected in the first part of the calibration, we need a function $J(x, \mathcal{D})$ that outputs the cost related to the vector x given the collected data $\mathcal{D}$. In this problem, a vector x_1 is said to be better than other vector x_2 if $J(x_1, \mathcal{D}) < J(x_2, \mathcal{D})$. We have a problem of optimization of the cost function J. Let us construct J in the following way: for every ArUco detection $i \in \mathcal{D}$ we infer its position (using the IPM algorithm) and then get an error $\varepsilon_i = \hat{p}_i - p_i$. Then J is the sum of the square of the modulus of the errors:

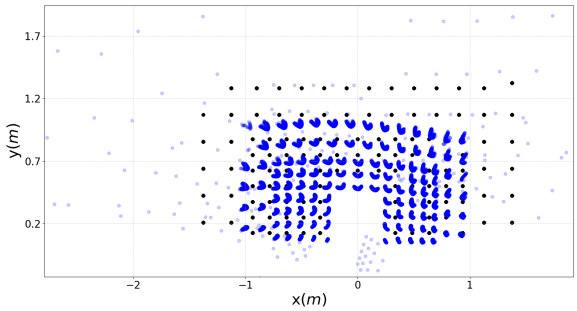
$$J(x, \mathcal{D}) = \sum_{i \in \mathcal{D}} \|\varepsilon_i\|^2 = \sum_{i \in \mathcal{D}} \|\hat{p}_i - p_i\|^2 , \tag{5}$$

where p_i is the ground truth position of the i-th detected ArUco in $\mathcal{D}$ and $\hat{p}_i$ is the calculated position of the i-th detected ArUco in $\mathcal{D}$. Remember $\mathcal{D}$ is all the collected data. In the example used in this paper, $\mathcal{D}$ is a text file with all the detected ArUcos' positions in their images, their real 3D positions and the robot joints' positions which are necessary to perform the IPM algorithm. p_i and $\hat{p}_i$ are 3 dimensional vectors, but in our case all ArUcos are in the same plane (they are on the ground, so $z = 0$), so p_i and $\hat{p}_i$ are reduced to two dimensional vectors.

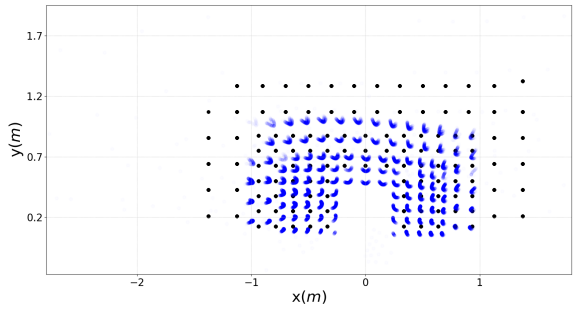
Calculating $\hat{p}_i$ is done using the information of the joints' angles saved in $\mathcal{D}$, the position of the detection in the image and the IPM algorithm, as described in section 2.

Knowing how to calculate $\hat{p}_i$ from the collected data and the offsets x, and the actual position of each ArUco p_i, the cost function $J(x, \mathcal{D})$ is defined. However, in practice, those definitions are not enough to optimize x, due to errors that occur during the data collecting part of the calibration. So, we need, firstly, to remove outliers from the data.

Removing outliers from data Plotting $\hat{p}_i$ for each detected ArUco $i \in d$ from some benchmark collected data (this data was collected during one of the calibration tests and is used throughout this entire paper) we obtain Figure 5. In Figure 5 (a), blue dots have alpha = 0.2 (image opacity level). Hence, dark points are actually more than one detection in the same region. Decreasing the value of alpha to 0.01 it is still possible to note some dark regions, as Figure 5 (b) shows.



(a) alpha = 0.2 for blue dots



(b) alpha = 0.01 for blue dots

Fig. 5: ArUcos' calculated positions before calibration. Blue dots are the calculated positions while black dots are the actual positions.

The difference between Fig. 5 (a) and (b) suggests that all the light blue dots found on Fig. 5 (a) are false positives. Those false positives are detrimental to the calibration, since some of them were detected really far away, creating a harmful bias in the optimization of x.

To remove the outliers in $\mathcal{D}$, we need to adopt a point selection criteria. The criteria we chose is described in the following steps:

- For each ArUco on the carpet, a list containing all the detections of this ArUco is made.
- For each list, we calculate the mean position and the standard deviation.
- We delete all the points in each list that are above 2 standard deviations from the mean. This value was chosen knowing that, for a gaussian distribution, approximately 95% of the data correctly detected will not be deleted. This allows us to exclude false positives without compromising the data.

The results of such selection can be seen in Figures 6 and 7. Red points are points that were excluded in the process. Blue points are the calculated points that were not excluded. The brown dot is the mean of the calculated positions (blue and red points) and the black point is the actual ArUco position.

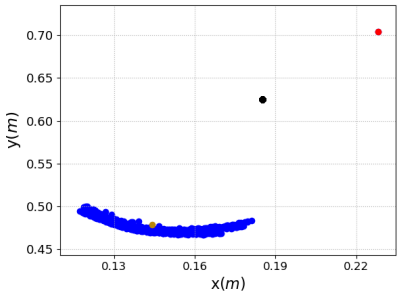


Fig. 6: Outlier removal example for an ArUco with 1 false positive.

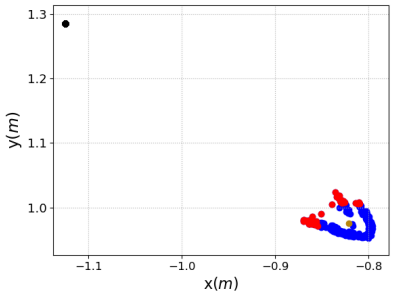


Fig. 7: Outlier removal example for an ArUco without false positives.

Even though this process may lose some true positives, such as Fig. 7, it is efficient to remove outliers, as shown in Fig. 6. The remaining data after this process is plotted in Fig. 8.
In fact, analysing Fig. 8 we can already associate each black dot with a cluster of blue points. This indicates that the outlier removal process worked correctly.

Optimizing offsets With the outliers removed from $\mathcal{D}$, we may optimize x through $J(x, \mathcal{D})$. Now, we need to choose an algorithm that estimates better offsets until we find the minimum of $J(x, \mathcal{D})$. This is a non-linear optimization problem. It is also hard to do some kind of gradient descent to solve this problem. Because of that, during the development of this calibration method, we used the following two optimization algorithms: Covariance Matrix Adaptation Evolution Strategy (CMA-ES) [11] and Nelder-Mead [12]. Both algorithms choose a starting value for x and, through a cost function (in this case J), they get a better x in the next iteration. The algorithms return a vector x when they reach a stop criterion, usually when $J(x, \mathcal{D})$ can not be reduced anymore by the algorithm.

It is important to highlight that neither of these algorithms guarantee convergence of $J(x, \mathcal{D})$ to its global minimum. Indeed, both algorithms can return

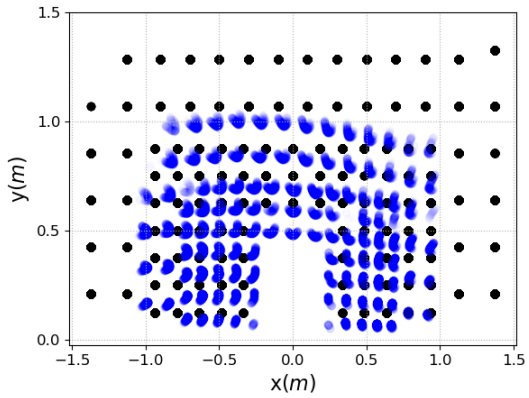


Fig. 8: Calculated positions after outlier removal (alpha = 0.1 for blue points).

a value of x found on a local minimum. For the purpose of this calibration, it is heuristically assumed that the values returned by these algorithms will, at least, suffice for our goals of calculating positions precisely, even if they are not the global minimum.

Due to time restrictions in the first tests, which were done using MATLAB, the Nelder-Mead algorithm was chosen instead of the CMA-ES algorithm. This algorithm is used as default in the final code of the ITAndroids team.

To suit our time constraints, the Nelder-Mead algorithm was rewritten in C++. Using the compilation flags "-Ofast -fext-numeric-literals -fPIC" for the g++ compiler, it was possible to achieve an execution time under 3 s on average. The results of the calibration using the Nelder-Mead algorithm (implemented in C++) are shown in Fig. 9.

4 Results and Discussions

Comparing Figs. 8 and 9, we notice that the offsets found by the algorithm improved the robot's position calculation, as the blue clusters are now much closer to their respective black points. In fact, the cost before optimizing x was $J(0, \mathcal{D}) = 3{,}782.52$. After optimizing, $J(x, \mathcal{D}) = 65.69$. That is a 98.3% cost reduction. Furthermore, the average cost of each detected ArUco was 0.00118 after the calibration. As all the positions are calculated in meters and the error is the squared distance, the quadratic mean of the distance between the calculated position of an Aruco and its actual position is $\sqrt{0.00118\ m^2} = 0.034\ m = 3.4\ cm$. So the average distance between an ArUco and its calculated position is less than 3.4 cm (remember that the arithmetic mean is less than or equal to the quadratic mean).

Finally, we need to check whether the solution found by the algorithm suits the problem conditions (even though the calibration results were good, if the

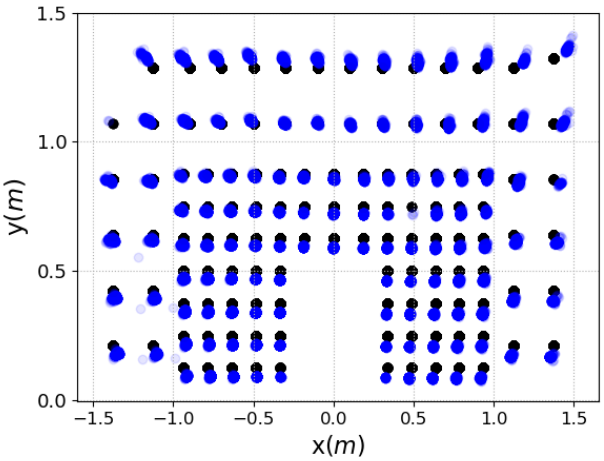


Fig. 9: Calculated positions after calibration ($alpha = 0.1$).

solution x has a big offset, say, a 38° offset, the solution should be disregarded, as this kind of error is beyond an usual mechanical error). In the example presented during this paper, the bigger offset found was the torso roll, with 7°, followed by the camera yaw with 1.4°. These values match the problem intuition of small offsets. Also, the solution must be obtained in a short time, as the robots are calibrated just before going into a match, so an algorithm that takes a long time to execute is impractical. As said in Section 3.2, it was possible to achieve an execution time, on average, under 3 s. This is enough for our purposes.

5 Conclusion

This paper presents a method for the calibration of the inverse perspective mapping model of a humanoid robot. The use of this calibration allows the IPM algorithm to precisely determine the robot distance to known objects in the field. This information can then be used with a Monte Carlo Localization technique, such as the one described in [2], to allow the robot to locate itself on the field.

In the example presented in this paper, the calibration reduced the costs of approximately 60 thousand object detections by 98.3%, making the average distance between estimated and real positions inferior to 3.4 cm. All those objects were placed in an area of 3 m by 1.5 m, which is around 8.3% of the field area of a RoboCup match.

For future research, we suggest:

- Use more complex models for the calibration (instead of the joint's angles).
- Use a deep neural network to optimize the offsets.

- Use the own field features to calibrate.
- Generalize this method so it can be applicable to other scenarios other than the humanoid league.

6 Acknowledgment

Francisco da Silva thanks CNPq (Conselho Nacional de Desenvolvimento Científico e Tecnológico) for his undergraduate research scholarship. Marcos Maximo is partially funded by CNPq through the grant 307525/2022-8. Takashi Yoneyama is partially funded by CNPq through the grant 304134/2-18-0. We thank the entire ITAndroids team, especially Lucas Steuernagel, for helping during data collection tests. We thank Robocup's sponsors MathWorks, FESTO, SoftBank Robotics and United Robotics Group for making all this development in robotics possible.

References

1. RoboCup, RoboCup Humanoid League, https://humanoid.robocup.org/ October 2020.
2. Muzio, A., Aguiar, L., Máximo, M. R. O. A., Pinto, S. C.: Monte Carlo Localization with Field Lines Observations for Simulated Humanoid Robotic Soccer, 2018.
3. Bestmann, M., Engelke, T., Fiedler, N.: TORSO-21 Dataset: Typical Objects in RoboCup Soccer 2021, https://github.com/bit-bots/TORSO_21_dataset
4. Tanveer, M. H., Sgorbissa, A.: An inverse perspective mapping approach using monocular camera of Pepper humanoid robot to determine the position of other moving robot in plane, 2018.
5. OpenCV, Camera Calibration. https://docs.opencv.org/4.x/dc/dbb/tutorial_py_calibration.html
6. Maximo, M. R. O. A.: Automatic walking step duration through model predictive control. Diss. PhD thesis, Aeronautics Institute of Technology, 2017.
7. OpenCV, Detection of ArUco Markers, October 2020.
8. RoboCup, Humanoid League Laws of the Game, http://www.robocuphumanoid.org/wp-content/uploads/RCHL-2019-Rules-changesMarked.pdf October 2020.
9. Szeliski, R., Computer Vision Algorithms and Applications, Springer, 2011, pp. 42–49.
10. Kajita, S., Hirukawa, H., Yokoi, K., Harada, K.: Introduction to Humanoid Robotics, Springer, 2014, pp. 19–42.
11. École Polytechnique, The CMA Evolution Strategy, http://www.cmap.polytechnique.fr/ nikolaus.hansen/cmaesintro.html October 2020.
12. Lagarias, Reeds, J., Wright, M., Wright, P.: Convergence Properties of the Nelder-Mead Simplex Method in Low Dimensions, SIAM Journal of Optimization, 1998, pp. 112–147.
13. Gondry, L., Hofer L., Laborde-Zubieta, P., et al.: Rhoban Football Club: RoboCup Humanoid KidSize 2019 Champion Team Paper, 2019
14. Mahmudi, H., Gholami, A., Delaravan, M. H., et al.: MRL Champion Team Paper in Humanoid TeenSize League of RoboCup 2019, 2019