ITAndroids Soccer3D
Team Description Paper 2018

Dicksiano Melo, Eric Endo Soares, Gustavo Nahum, Heládio Lopes, Juan Freire, Lucas Maia, Luckeciano Melo, and Marcos Maximo

Autonomous Computational Systems Lab (LAB-SCA)
Aeronautics Institute of Technology (ITA)
São José dos Campos, São Paulo, Brazil
{dicksianomelo,eric.toshio.e.s,gustavo.nahum,heladio.lopes,juanfdantas, lucasmaiamorais,luckeciano}@gmail.com
mmaximo@ita.br
itandroids-soccer3d@googlegroups.com
http://www.itandroids.com.br

Abstract. ITAndroids was reestablished in mid-2011 by undergraduate students at Aeronautics Institute of Technology. Since then, the team is growing fast and has already established itself as a strong robotics competition group, winning several competitions in Brazil and Latin America.

1 Introduction

ITAndroids is a robotics competition group associated to the Autonomous Computational Systems Lab (LAB-SCA) at Aeronautics Institute of Technology. As required by a complete endeavor in robotics, the group is multidisciplinary and contains about 50 students from different undergraduate engineering courses. In the last 5 years, we have achieved good results in competitions, especially in Latin America.

Our older Soccer 3D team had a Base made from Magma Offenburg, but since RoboCup 2016 we use our own Base using C++. Currently, ITAndroids is the only team of Latin America which uses its own Base code. RoboCup 2016 was the first run and we achieved the 6th place.

This paper describes our development efforts in the last years and points out some improvements we want to implement in a near future. Sec. 2 describes our team's code structure. In Sec. 3, we discuss our localization method. In Sec. 4, we show our motion control system. Sec. 5 explains our role assignment system for positioning. Sec. 6 shows our strategy and points our robot navigation method. Sec. 7 shows how we have started integrating Tensor Flow API into our code. Sec. 8 presents a tool developed for statistical tests. Finally, Sec. 9 concludes and shares our ideas for future work.

2 Code Structure

The code has been planned and divided in several modularized parts, so that each part can be separated from the others with ease. ITAndroids Soccer 3D used to work with a Base made from Magma Offenburg and ITAndroids Soccer 2D works with Agent2D. In this way, our current code structure was heavily influenced by those works.

2.1 Communication

It is the layer that directly connects with the server, in order to receive messages and send messages to it. This layer receives and sends a string as described in the server's website. It uses a socket to implement the communication, receiving a message that contains the information from the server.

2.2 Perception

This layer receives the string from Communication, and converts the string into a tree, parsing it. The layer then iterates over the created tree and creates new objects from it, so that the agent can have new information each new loop. The created objects come in the form of perceptors and each perceptor is as described in the server's website.

2.3 Modeling

Modeling basically models the world state. Not only that, it also models situations, e.g. in whether the agent has fallen. It uses the perceptors created in perception to update its models, getting the relative position of field landmarks to find it's positions, and acelerometer data to see if the agent has fallen.

Agent Model The Agent model is the part of the code that models specific agent modes. It calculates the transformation matrices, in order to change the coordinate system from the camera to a ground coordinate system, using robot joints and making simplifications on the system.

World Model The World Model is responsible for modeling parameters like game state, time, and position, so that these information can be used by Decision Making. It runs the Localization algorithm in order to estimate the robot's position.

2.4 Decision Making

Decision Making is a layer that divides each robot by decision makers. It consists Behavior and Decision Maker. One agent cannot change its decision maker, but, that decision maker must be able to integrate all the possible behavior the agent has available, e.g. a Soccer agent receives a SoccerDecisionMaker, and a goalie receives a GoalieDecisionMaker. Those decision makers dictate the movements the agent should take in order to successfully follow a determined strategy.

Behavior is a set of what the agent can to in order to change its own state. Behavior is a set of instructions that goes from high to low level of abstraction, in order to make the agent follow it's strategy. Each behavior can use other behaviors for a more abstract level of problem solving.

The layer has two parts, a part that is a data structure called BehaviorFactory, and a structure called Behavior. A BehaviorFactory stores all behaviors within itself, and each behavior has access to all other behaviors in the Behavior Factory.

2.5 Control

Control is the layer that gets the requests from behavior and changes it into more concrete things. For example, it takes a walk request created from one of the behaviors, and converts it into joints positions.

2.6 Action

Action is a layer that converts all the information the agent has created and wants to send to the server into a string, in a way the server can recognize.

3 Localization

Using complex strategies in robot soccer requires that the agent knows its global position in the soccer field. The problem of having a mobile robot estimates its pose with respect to a global coordinates system is termed Localization in the robotics community. To solve this problem, the standard approach involves using a Bayes filter, which iteratively incorporates sensors' measurements and the robot's actions to construct a probabilistic estimate of the robot position. Since implementing this technique directly is not feasible computationally, approximated techniques, such as the Kalman filter [12] or the particle filter [13], are often employed. We decided to use Monte Carlo localization (MCL) [6], which uses a particle filter to solve the Localization problem, because it is one the most efficient methods [11] and some teams in 3D Soccer Simulation have successfully used this technique [4, 5].

Our MCL implementation was greatly inspired by the work explained in [7]. Each particle mantains a pose estimate represented by a 3-dimensional vector $\mathbf{x} = [x, y, \psi]^T$, where x and y are global field coordinates and ψ is the horizontal angle the torso of the robot is heading. We use a bootstrap particle filter with resampling step [14].

In the sensing phase, we currently landmarks (flags and goalposts) observations and line observations. Our landmark observation model consider gaussian noises corrupting the horizontal distance and horizontal angle measurements with covariances σ_d^2 and σ_ψ^2, respectively. Furthermore, we consider that landmarks observations are independent of each other. Therefore, a suitable rule for updating the particles' weights is:

$$w_k^{(i)} = w_{k-1}^{(i)} \prod_j \left\{ \exp\left[-\left(\frac{d_j - \hat{d}_j^{(i)}}{\sigma_d} \right)^2 \right] \cdot \exp\left[-\left(\frac{\psi_j - \hat{\psi}_j^{(i)}}{\sigma_\theta} \right)^2 \right] \right\} \quad (1)$$

where $w_k^{(i)}$ is the weight of the i-th particle in the k-th sampling time, d_j is the measured horizontal distance between the robot and the j-th landmark, $d_j^{(i)}$ is the horizontal distance between the robot and the j-th landmark considering the current i-th particle's position, $\hat{\psi}_j^{(i)}$ is the measured horizontal angle between the j-th landmark and the robot's heading, $\psi_j^{(i)}$ is the expected horizontal angle between the j-th landmark and the robot given the i-th particle position. To determine adequate values for σ_d and σ_ψ, we started with measurement covariances presented in Simspark's documentation and finely tuned these values by hand. We do not execute a sensing phase when no vision update is present (note that Simspark sends vision updates only every 3 cycles).

For motion update, we use odometry information given by our walking engine, which gives the torso displacement vector $\Delta\mathbf{d}_k = [\Delta x_k, \Delta y_k, \Delta\psi_k]^T$ relative to the local torso coordinates frame of the previous time step. At first, slipping was making odometry and actual movement differ too much, especially at high walking speeds. Simply scaling each channel proved effectively in solving this:

$$\mathbf{d}'_k = \begin{bmatrix} \Delta x'_k \\ \Delta y'_k \\ \Delta\psi'_k \end{bmatrix} = \begin{bmatrix} \alpha\Delta x_k \\ \beta\Delta y_k \\ \gamma\Delta\psi_k \end{bmatrix} \quad (2)$$

Thus, $\mathbf{d}'$ was the value of displacement effectively used for motion update. The parameters α, β and γ were manually tuned by comparing the evolution of the robot's position and its estimate in Roboviz while the robot was walking. relative to the robot's coordinates frame) and manually tweaking α, β and γ. Then, we update the position of each particle i using:

$$\mathbf{x}_k = \begin{bmatrix} x_k^{(i)} \\ y_k^{(i)} \\ \psi_k^{(i)} \end{bmatrix} = \begin{bmatrix} x_{k-1}^{(i)} + \Delta x'_k \cos(\psi_{k-1}) - \Delta y'_k \sin(\psi_{k-1}) \\ y_{k-1}^{(i)} + \Delta x'_k \sin(\psi_{k-1}) - \Delta y'_k \cos(\psi_{k-1}) \\ \psi_{k-1}^{(i)} + \Delta\psi'_k \end{bmatrix} + \begin{bmatrix} \varepsilon_x \\ \varepsilon_y \\ \varepsilon_\psi \end{bmatrix} \quad (3)$$

where $\varepsilon_x \sim \mathcal{N}\left(0, \sigma_x^2\right)$, $\varepsilon_y \sim \mathcal{N}\left(0, \sigma_y^2\right)$ and $\varepsilon_\psi \sim \mathcal{N}\left(0, \sigma_\psi^2\right)$ incorporate process noise. These covariances were also manually tweaked. We execute a motion phase every cycle, then we naturally run more motion than sensing phases.

To reduce particle deprivation, we use a resampling step after sensing and motion phases [14]. Given that resampling algorithms may be computationally expensive, we use the $O(N)$ algorithm shown in [11], where N is the number of particles used.

To avoid the kidnapped robot problem, which happens in the 3D Soccer Simulation domain when the server teleports the agent, we used the strategy known as Adaptive-MCL [8, 11], which resets particles based on a heuristic estimate of how bad localized the agent is. Instead of distributing the resetted particles randomly in the soccer field [7], we use the current vision observations to better reset the particles [8].

Given that two landmarks observations, we may estimate where the robot is as shown in Figure 1. Note that we end with two hypotheses, which may be chosen at random. In our case, one of them will usually falls outside of the soccer field, thus may be discarded. If more than two landmarks are seen in the current cycle, two landmarks are chosen at random for each particle which is being resetted. Moreover, we add gaussian noises to the landmarks' observations before applying this resetting process to better spread the resetted particles.

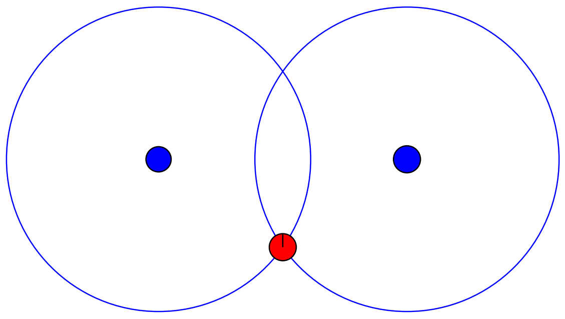

Fig. 1. Determining the agent's localization using two landmarks observation.

Finally, the agent's position estimate is determined by a weighted average of the particles' positions. For future work, we expect to determine the parameters using experiments or optimization techniques instead of relying on manual tweaking.

4 Motion Control

For walking, we use the ZMP-based omnidirectional walking engine described in [9]. In general terms, it follows the flux presented in Fig. 2. The input to the algorithm is the desired velocity $\mathbf{v} = [v_x, v_y, v_\psi]^T$ with respect to the robot's local coordinate system. At the beginning of a new step, poses for the torso and the swing foot are selected for achieving the expected displacement at the end of the step. So, a trajectory for the center of mass (CoM) to follow a reference Zero Moment Point (ZMP) trajectory is computed. The trajectory of the swing foot is obtained by interpolating between the initial and final poses of this foot. Finally, joints angles are calculated through Inverse Kinematics (IK) considering the poses of the support and swing feet. Note that the module "Next Torso and Swing Poses Selector" is called once for step, while the others are executed at the update rate of the joints.

Our step planner selects torso and feet poses to make the robot follows an omnidirectional model while respecting self-collision and leg reachability con-

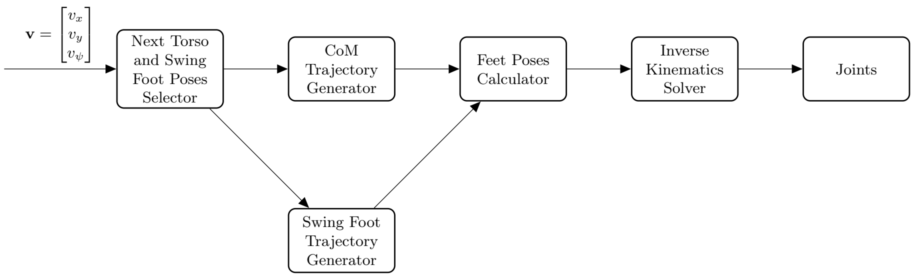
$\mathbf{v} = \begin{bmatrix} v_x \\ v_y \\ v_\psi \end{bmatrix}$
Next Torso and Swing Foot Poses Selector
CoM Trajectory Generator
Feet Poses Calculator
Inverse Kinematics Solver
Joints
Swing Foot Trajectory Generator

Fig. 2. Walking Engine overview.

straints. To reason about the robot dynamics, we approximate it using the 3D Linear Inverted Pendulum Model (3D-LIPM) [1]:

$$\mathbf{x}_{ZMP} = \mathbf{x}_{CoM} - \frac{z_{CoM}}{g}\ddot{\mathbf{x}}_{CoM} \tag{4}$$

where $\mathbf{x}_{ZMP} = [x_{ZMP}, y_{ZMP}]^T$ is the ZMP position, $\mathbf{x}_{CoM} = [x_{CoM}, y_{CoM}]^T$ is the CoM position, z_{CoM} is the CoM height, and g is the acceleration of gravity. The ZMP is kept at the center of the support foot during single support and moves from the current support foot to the next one during double support. We also use the torso's angular velocities measured by the gyrometer to stabilize the walk. This feedback strategy has proven quite effective, especially when walking on artificial grass.

For kicking, we first considered that it is a movement where the biped starts in a stand position, kicks the ball and returns to the same stand position. Given this description, we divided the motion into 4 phases: a) the ZMP is moved to the center of the support foot; b) the robot kicks the ball; c) the kicking foot is placed back on the ground; d) the ZMP goes back to the center of the torso. To balance the biped during kicking, we employ the same ZMP-based algorithm used for walking. For the getting up movements, we are using the original Robotis OP2 keyframes with slight modifications to make them work on artificial grass. For more information about our keyframe movements framework, please refer to [10].

5 Dynamic Role Assignment

5.1 Positioning Model

The Delaunay Triangulation algorithm calculate our player's positions and generate a formation set of agents for every single position of the ball. These positions serve as reference points for an agent to where they should be if there is not any enemy in a dangerous position just according to the ball‘s position in that moment. The Dynamic Role Assignment idea is that the agents can communicate themselves, decide the best lineup in that moment and assign the Delaunay Triangulation positions to the team, one for each agent. And do it all dynamically, sending and receiving messages in every communication cycle.

The lineup consists on eleven positions and each agent, using the data received in the World Model, is able to determine which agent should be assigned to each Delaunay Triangulation position based on the agents‘ position. Each position is assigned to a number 1 - 11, just as the agent‘s uniform number. So, the agent generates an array and assigns the number of the Delaunay Triangulation position to each player in its team.

5.2 Communication between Agents

The objective is send a vector containing each position assigned to each other agent in the team. Each agent can send and receive messages from the server in every cycle of 20 ms, with the message‘s size limited to 20 bytes (160 bits). The problem is that every integer occupies 4 bytes, and the team has 11 players. So, the vector would have occupy 11*4 = 44 bytes of memory, a lot more than the limit. The solution used was to use the base64 ascii encoder, a compression message encoder that creates a bijection between the vector and other structure which occupies less memory. Then, the agents would receive the encoded message and, using the same bijection, be able to decode and receive the message vector. The method proved itself to be very efficient, reducing the size of the role assignment vector form the original 44 bytes to only around 110 - 120 bits.

5.3 Voting System

When the agents have all sent their respective role assignment vectors to each other (They may differ from agent to since their perception is different), they acknowledge the position which was assigned more times to each respective agent. Then, the team's role assignment vector would be filled based on the assignments. Then, the final role assignment vector would be generated based on these votes, and each player verify the position he should go.

5.4 Marking System

The marking system is a sequential process used to mark opponents agents who are offensively dangerous during the match.

The System performs three steps: it decides the players that will be marked, defines roles to mark these players and, finally, uses the Role Assignment system to designate agents for those defined roles.

To decide which opponents are to be scored, a heuristic method is used based on the following conditions:

1. Opponent is close enough to take a shot on goal
2. Opponent is not the closest opponent to the ball.
3. Opponent is not too close to the ball

4. Opponent is not too far behind the ball

After that, a set of formation roles must be selected to mark the chosen opponents. For this, marking roles positions are calculated as the position 1.5 meters from a marked opponent along the line which connects that opponent to the center of our goal. The formation positions to be replaced are the closest to each marking position, and their selection is done by evaluating each one for every replacement, which is a suboptimal solution. In the future, we plan to use the Hungarian algorithm, which calculates the minimum sum of distances between the previous forming positions and the marking positions.

6 Strategy and Decision Making

6.1 Set Plays

As in a soccer game, during a simulated game there are situations where the ball is stopped. When this happens, the team with the ball has a certain amount of time to make a move, while the other team can't approach more than a certain distance from the ball. For some of these situations were formulated different Set Plays, they were: Own Goal Kick, Opponent Goal Kick and Own Corner Kick. We plan to implement new Set Plays for each Dead Ball situation in the game.

6.2 Robot Navigation

Planning consists of finding a sequence of actions that transforms some initial state into some desired goal state.

Potential Field This robot navigation method applies to each field's point a numerical value that corresponds to the potential caused by external agents. Therefore, is possible to know which points should be avoided and which point is the goal. We have several advantages of using Potential Fields to robot navigation [15]. First, it's easy to implement and visualize, and the resulting behavior of the robot is therefore easy to predict; furthermore, they support parallelism - each field is independent of the others and may be implemented as general software.

In our case, the function **calculateDirection()** receives our current position and a vector with all objects (players, the ball, the goal, etc) in the soccer field. Then, we calculate the potential that each object in this vector causes in a specific point of the field (our position). We can calculate this using the concept of potential like a constant divided by a power of the distance d between the object and position.

As we are interested in a direction, our function calculates the potential associated with each direction (horizontal and vertical). Potential is a scalar quantity, therefore we can add directly the potential caused by each object in the vector. So, our function calculates each potential and sum them all. After that, we can calculate the desired direction using this formula:

$$\theta = \arctan\left(\frac{V_y}{V_x}\right) \tag{5}$$

As you can see, our function **calculateDirection()** returns the angle (in radians) between the horizontal direction and desired direction, that points to the goal and avoids the obstacles.

6.3 Walk Optimization

ITAndroids Soccer 3D started to develop optimization tasks focusing on walking's movement. The purpose of this activity is improve stability and velocity of the movement, based in the fact that previous work on this area reported great results [2]. The team developed three tasks and optimized Omndirectional ZMP parameters and Navigation parameters. The three tasks are described below:

Sprint Task In this task, the robot runs straightforward. The fitness function takes in consideration the maximum distance traveled by the agent (as a "positive reinforcement") and if the agent has fallen down (as "negative"). The purpose of this task is just improve speed in the scenario of agent travelling to a directed target, without contact with another agent or the ball.

"GoToBall" Task This task was developed in [3] and basically the agent go to the ball and conduct it in the

The fitness function takes in consideration the distance between the ball and the goal and if the agent has fallen down (both as "negative reinforcement"). The purpose of this task is improve stability and speed in the scenario of agent conduction the ball towards the goal.

"GoToTarget" Task This task was based on the main task developed in [2] and is divided in three parts. In the first, a target is set 5 meters from the robot, in a random direction. After a specified time, the process is repeated. This loop of generate a random target and the agent run towards this targets occurs several times, with different distances from the agent (and different times). The fitness function takes in consideration the time that the agent used for reach the target and the distance used. Furthermore, the task also considered if the robot has fallen down in each loop. The purpose of this task is improve stability simulating the rapidly change of direction in the soccer game context.

6.4 Walk State Machine

With the optimization of walking parameters it became necessary to re-plan the movement. In this way, a new decision-making was implemented according to the walking situation. Are they: sprint, when the agent walks without the ball; conduct, when the agent walks with the ball; turn, when the agent cross curve paths; and circle, when the agent circulates the ball.

7 Tensor Flow

We have integrated the Tensorflow C++ API into our code. At this moment, it's possible to load a freezed graph (a graphdef from Keras, for example) and perform a forward propagation. We've tested with a simple model of particle filtering based on a denoising autoencoder (however, this model perfomed poorly on simspark enviroment). As future work, it's planned to develop the backward propagation, aiming to tasks that need online learning.

8 Statistic tool

A major issue beneath complex dynamics virtual environments such as in 3D soccer matches simulations is the obtainment of optimal parameters to be assigned to each player in several game events. For example, a player may have fallen to the ground in a certain position; as long as its physical constants differ deppending on whether its back was up or down, the parameters involved in the movement of getting up in the lowest time may differ. Another game scenario might be the one in which two opponent players rush towards the ball in order to kick it in the direction of one of its allies; while the running takes place, the employed parameters must provide an efficient speed in an attempt to arrive the earliest to the ball; however, as a player gets close enough to the ball, it should also focus on kicking with a reasonable strength and in the right position so that the ball moves the way it was intended: that requires even more caution with respect to balance and footstep planning.

Traditionally, what has been done in Soccer3D to generate optimal sets of such parameters is running optimization algorithms through scripts of trainers. These trainers simulate game-like conditions, e.g. running, kicking, getting up, etc., and serve the purpose of enhancing specific skills in players. However, there are certain weaknesses in trainers that, at least partially, compromise their performance: one of them is a consequence of their biased nature – once a trainer allots a sequence of tasks for a player to perform, the optimization algorithm will find out good parameters for the very tasks that were assigned, and there is a chance of obtaining parameters that are unstable for various other game conditions. Another critical difficulty refers to the greater complexity and multiplayer interaction that take place during actual matches. Given the relevance of these dynamics, their influence needs to be accounted for when calibrating player parameters, and the only way through that is effectively simulating actual matches.

However, the relatively slow motion of the soccer players results in a static game flow, and relevant game information happens very little often. Besides, matches' durations are lengthy, so that simulations activated manually by the team members require a considerable amount of time. Therefore, not only is this approach time consuming and represent a tedious experience to the members; it is also highly non-efficient, demanding several cycles of activation of the trainers and optimization servers, not to say all the following-along of matches led by the members for every set of parameters that are willing to be tested.

A better approach that was considered by the the Soccer3D team on this semester was the implementation of an automatic game analysis tool, that retrieves from the simulated matches all the relevant data required to have a deeper outlook of the set of parameters being tested. In order for this automatic analysis to be accomplished, two essential tools were conceived: a league manage tool, whose main purpose is to generate game logs containing all the match data available; and a statistical tool, which is in charge of processing and filtering the data generated by the league manager, and from that displaying information that is more easily understandable to the user.

With respect to the statistical tool, the first implementation yet ready to use is the total number of goals scored by each of the teams, as well as the time instants that every goal was scored. The next step still to be developed is the determination of the ball possession rate attributed to each opponent team. Further extensions of the statistical tool may include the number of fouls committed, the number of kicks performed and the number of times the players of a given team have fallen to the ground during the game.

9 Conclusions and Future Work

It's been saw that, even though we remade our code structure, there is much more to do. During our last competition, Robocup 2017, was clear that our team defense was acceptable. However, our attack was well below expected. Now that our movements have been optimized and we can actually advance much more through the field, we will focus on developing attack strategies to improve the goal balance, as well as the statistical tools necessary for a better knowledge about the effects of those changes.

Acknowledgement

We would like to acknowledge the RoboCup community for sharing their developments and ideas. Specially, we would like to acknowledge Magma Offenburg team for sharing their code and, with that base, helping us develop our current code. We thank our sponsors ITAEx, Micropress, Poliedro, Radix, Dassault Systems, Rapid, Mentinjo and Altium . We also acknowledge Mathworks (MATLAB), Atlassian (Bitbucket) and JetBrains (CLion) for providing access to high quality software through academic licenses.

References

1. S. Kajita, F. Kanehiro, K. Kaneko, K. Yokoi, and H. Hirukawa. The 3D Linear Inverted Pendulum Mode: A Simple Modeling for a Biped Walking Pattern Generation. In Proceedings of the 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems, October 2001.
2. P. MacAlpine, S. Barret, D. Urieli, V. Vu, P. Stone. Design and Optimization of an Omnidirectional Humanoid Walk: A Winning Approach at the RoboCup 2011 3D Simulation Competition. Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence (AAAI), July 2012.
3. D. Urieli, P. Stone. UT Austin Villa 2011: A Champion Agent in the RoboCup 3D Soccer Simulation Competition. Proc. of 11th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS), June 2012.
4. MacAlpine P., Urieli, D., Barrett, S., Kalyanakrishnan, S., Barrera, F., Lopez-Mobilia, A., Ştiurcă, N., Vu, V., Stone, P.: UT Austin Villa 2011 3D Simulation Team Report. Technical Report, The University of Texas at Austin, Department of Computer Science, AI Laboratory (2011).
5. Haider, S., W., M.-A., Raza, S., Johnston, B., Abidi, S., Sharif, U., Raza, A.: Karachi Koalas3D Simulation Soccer Team, Team Description Paper for World RoboCup 2012 (2012).
6. Dellaert, F., Fox, D., Burgard, W., Thrun, S.: Monte Carlo Localization for Mobile Robots. In: IEEE International Conference on Robotics and Automation (ICRA'99) (1999).
7. Hester, T., Stone, P.: Negative Information and Line Observations for Monte Carlo Localization. In: IEEE International Conference on Robotics and Automation (ICRA'08) (2008).
8. Coltin, B., Veloso, M.M.: Multi-Observation Sensor Resetting Localization with Ambiguous Landmarks. In: Proceedings of AAAI'11, the Twenty-Fifth Conference on Artificial Intelligence, San Francisco, CA (2011).
9. Marcos R. O. A. Maximo. Omnidirectional ZMP-Based Walking for a Humanoid Robot. Master's thesis, Aeronautics Institute of Technology, 2015.
10. Francisco Muniz, Marcos Maximo and Carlos Ribeiro. Keyframe Movement Optimization for Simulated Humanoid Robot using a Parallel Optimization Framework. In: Latin American Robotics Symposium. In Proceedings *of the 2016 Latin American Robotics Symposium (LARS)*, October 2016.
11. Thrun, S., Burgard, W., Fox, D.: Probabilistic Robotics. MIT Press (2005).
12. Kalman, R.E.: A New Approach to Linear Filtering and Prediction Problems. Transactions of the ASME–Journal of Basic Engineering 82, pp. 35–45 (1960).
13. Smith, A.F.M., Gelfand, A.E.: Bayesian Statistics Without Tears: A Sampling-Resampling Perspective. American Statistician 46, pp. 84–88 (1992).
14. Bruno, Marcelo G.S. Sequential Monte Carlo Methods for Nonlinear Discrete-Time Filtering. Synthesis Lectures on Signal Processing, January 2013, Vol. 6, No. 1, Pages 1-99.
15. Hellstrom, Thomas. Robot Navigation with Potential Fields. Department of Computer Science, Umea University, 2011.
16. Khatib, O., "Real-Time Obstacle Avoidance for Manipulators and Mobile Robots." 1985 IEEE International Conference on Robotics and Automation, March 25-28, 1985, St. Louis, pp. 500-505.