ITAndroids Soccer3D Team Description Paper 2016

Alexandre Muzio, Dicksiano Melo, Eduardo Henrique, Francisco Muniz, Ian Marzzo, José Lucas Saraiva, Luckeciano Melo, Luis Guilherme Aguiar, Marcos Maximo, and Matheus Bertolino

Aeronautics Institute of Technology,
São José dos Campos, São Paulo, Brazil
{ax.muzio,dicksianomelo,eduardo.hferreiras,
munizfco,iannlibos,jlucas.saraiva,luckeciano,
luisgspy,maximo.marcos,matheusbertolino1}@gmail.com
http://www.itandroids.com.br/pt-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. This technical paper describes our latest efforts regarding humanoid robots. Moreover, we discuss our plans for future development.

1 Introduction

ITAndroids is a robotics research group at Technological Institute of Aeronautics. The group was founded in 2006 by Jackson Matsuura, but it stayed inactive for many years until reestablished in mid-2011. As required by a complete endeavor in robotics, the group is multidisciplinary and contains about 30 students from different undergraduate engineering courses. In the last 4 years, we have achieved good results in competitions, especially in Latin America:

- 10th place in RoboCup 2D Soccer Simulation in RoboCup 2012;
- 1st place in RoboCup 2D Soccer Simulation in Latin American Robotics Competition (LARC) 2012;
- 2nd place in RoboCup 3D Soccer Simulation in LARC 2012;
- 3rd place in IEEE Humanoid Robot Racing in LARC 2012;
- 12th place in RoboCup 2D Soccer Simulation in RoboCup 2013;
- Top 12 in RoboCup 3D Soccer Simulation in RoboCup 2013;
- 1st place in RoboCup 2D Soccer Simulation in Brazilian Robotics Competition (CBR) 2013;
- 2nd place in RoboCup 3D Soccer Simulation in CBR 2013;
- 1st place in RoboCup 2D Soccer Simulation in LARC 2014;
- 2nd place in RoboCup 3D Soccer Simulation in LARC 2014;
- 3rd place in RoboCup Humanoid KidSize in LARC 2014;
- 1st place in RoboCup 2D Soccer Simulation in LARC 2015;

– 2nd place in RoboCup 3D Soccer Simulation in LARC 2015.

Our current Soccer 3D team had a Base made from Magma Offenburg, but we are developing our own Base using C++. We are ending this project, therefore we intend to use our new code in RoboCup 2016.

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 new team's code structure. In Sec. 3, we discuss our localization method. In Sec. 4, we show our motion control system. Sec. 5 shows our strategy and points out our positioning and our robot navigation method. Finally, Sec. 6 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 works with a Base made from Magma Offenburg and ITAndroids Soccer 2D works with Agent2D. In this way, our code structure was heavily influenced by those works. Besides that, this is the code structure of our new team. Our older team which will be used for RoboCup's qualification has a diferent structure.

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. First, the received message is parsed by a Parser, creating a tree that in each new "(" creates a new node, and each ")" ends a node. For example, the following message "(ACC (n torso) (a 0.00 0.00 9.81))".

2.3 Modeling

Modeling basically models the world state. It runs algorithms to discover where the robot is in the field, and where the other agents are. 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. Modeling is divided in two parts, a World Model and an Agent Model.

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 a layer that divides each robot by decision makers. One agent cannot change it's 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.

2.5 Behavior

Behavior is a part of Decision Making (it consists Behavior and Decision Maker), it is a set of what the agent can to in order to change it's 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. It stores things like NavigateToPosition and Attack behaviors. Each behavior can use other behaviors for a more abstract level of problem solving. For example, Attack behavior can call upon NavigateToPosition so that NavigateToPosition doesn't have to be reimplemented in Attack. Each behavior create a request, that is an communication interface with the Control layer, and it is how the agent knows which specific action to take.

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.

2.6 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. It is where the motion algorithms are implemented.

2.7 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. The messages are created in a way that the agent can be executed in sync mode with the server.

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 [9] or the particle filter [10], are often employed. We decided to use Monte Carlo localization (MCL) [5], which uses a particle filter to solve the Localization problem, because it is one the most efficient methods [8] and some teams in 3D Soccer Simulation have successfully used this technique [3, 4].

Our MCL implementation was greatly inspired by the work explained in [6]. 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 [11].

In the sensing phase, we currently use only landmarks (flags and goalposts) observations, however we expect to incorporate line observations in future developments. 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:

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

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:

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

where $\varepsilon_x \sim \mathcal{N}(0, \sigma_x^2)$, $\varepsilon_y \sim \mathcal{N}(0, \sigma_y^2)$ 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 [11]. Given that resampling algorithms may be computationally expensive, we use the $O(N)$ algorithm shown in [8], 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 [7, 8], 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 [6], we use the current vision observations to better reset the particles [7].

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.

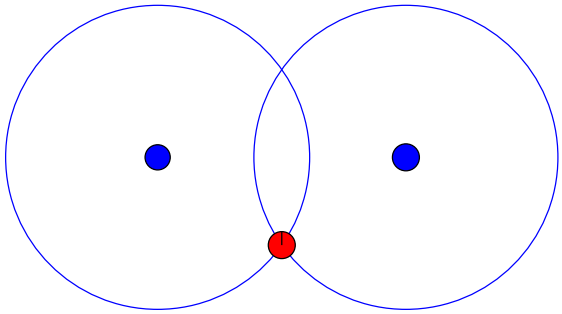

Fig. 1. Determining the agent's localization using two landmarks observation.

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

In 3D Soccer Simulation league, most actions of the robots are highly dependent on its ability to walk. Therefore, a great amount of our team efforts was focused on walking. In order to end a good walking method, several ideas were tested. Our latest walking models are described in this section.

4.1 Parametric Omnidirectional Walk

One of the walking methods tested by our team was based on [3]. The walking engine developed used a similar parametrization for the trajectory. However, our development was focused on being able to achieve a fast and stable walking without the need of using much computational resources during the optimization process.

On the optimization of the parameters of a walking trajectory, one of the biggest problems is the need of adapting the parameters to different goals. The two main goals in this task are walking as fast as possible and being as stable as possible. Since the combination of more than one goal on the same evaluation function commonly does not provide a good tradeoff between these goals, the problem was divided in two coupled optimization problems with different sets of parameters to be optimized and different goals for each problem.

The first problem was maximizing the stability of the movement keeping the speed constant. However, in order to take into account the influence of the size of the step in the movement stability, we kept the ratio between the size and the duration of the step constant and not both of these parameters. The second problem was to increase the speed of the robot as much as possible without making the movement too unstable.

The major advantage of this approach is that, using each parameter to optimize the feature of the walking that is more influenced by it, the influence of a change on a parameter is noticed earlier; thus, allowing a faster optimization process. For instance, the height to which the moving foot of the robot is lifted during a step influences the stability of the movement, but it does not influence the speed of the movement unless the feet of the robot are sliding on the floor. Therefore, a evaluation function that considers both the speed and the stability of the movement might fail to notice this change in the stability while it would be much easier to notice it if considering only the stability of the robot.

In the end, the optimization was composed of two alternating steps, one making the movement as stable as possible while keeping the speed constant and the other one increasing the speed as much as possible while keeping the other parameters constant. Using this strategy, it was possible to manually tweak the parameters and achieve a reasonable fast and stable motion. In the future, we intend to adapt this strategy to a method for automatically setting the parameters without the need of much computational effort.

4.2 ZMP Based Omnidirectional Walking Engine

In general terms, the walking engine follows the flux presented on Figure 2. The input to the algorithm is the desired velocity $\mathbf{v} = [v_x, v_y, v_\psi]^T$ with respect to the local coordinate system of the robot. Then, 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) that keeps the Zero Moment Point (ZMP) at the center of the support foot is computed by using an analytic solution of the 3D-LIPM equation. We approximate the CoM by a fixed position in the torso. 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.

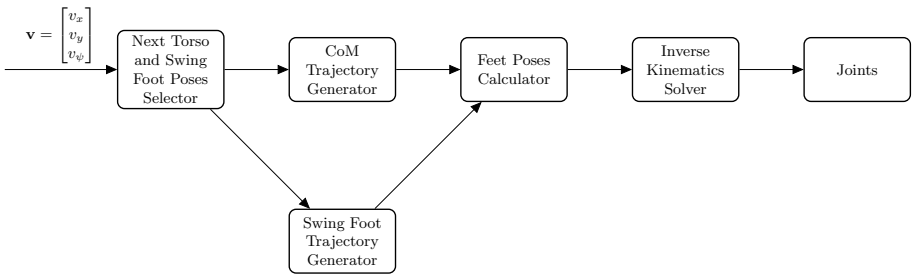

Fig. 2. Walking Engine overview.

A humanoid robot must satisfy dynamical constraints to remain stable. Moreover, the robot geometry imposes constraints: leg reachability is limited by leg physical dimensions and we also do not want movements where the legs collide. To achieve dynamic stability, our strategy is to generate a CoM trajectory that keeps the ZMP at the center of the support foot during single support. We may at least restrict the initial and final positions of this CoM trajectory, as will be explained below. Hence, our requirement is to have the robot matches the omnidirectional model only at the beginning and at the end of the step. Assuming constant v, we may compute the expected pose after a step duration T:

$$\begin{bmatrix} x[k+1] \\ y[k+1] \\ \psi[k+1] \end{bmatrix} = \begin{bmatrix} x[k] + 2\frac{v_x}{v_\psi}\sin\left(\frac{v_\psi T}{2}\right)\cos\left(\psi[k] + \frac{v_\psi T}{2}\right) - 2\frac{v_y}{v_\psi}\sin\left(\frac{v_\psi T}{2}\right)\sin\left(\psi[k] + \frac{v_\psi T}{2}\right) \\ y[k] + 2\frac{v_x}{v_\psi}\sin\left(\frac{v_\psi T}{2}\right)\sin\left(\psi[k] + \frac{v_\psi T}{2}\right) + 2\frac{v_y}{v_\psi}\sin\left(\frac{v_\psi T}{2}\right)\cos\left(\psi[k] + \frac{v_\psi T}{2}\right) \\ \psi[k] + v_\psi T \end{bmatrix} \tag{4}$$

For our multibody humanoid robot, it is convenient to select a body part as representative of the whole robot motion: we choose the torso for this. Thus, at the beginning of a new step, given the current torso and swing foot poses, an algorithm plans the torso and swing foot poses at the end of the step to make the torso arrive at the pose dictated by Equation (4) while trying to satisfy geometric constraints. This algorithm is based mainly on heuristics.

We still need to move the robot without losing balance. To reason about the robot dynamics, we approximate it using the 3D Linear Inverted Pendulum Model (3D-LIPM) [1]:

$$\mathbf{x}_{ZMP} = \mathbf{x}_{CoM} - \frac{z_{CoM}}{g}\ddot{\mathbf{x}}_{CoM} \tag{5}$$

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 it from the current support foot to the next one during double support.

However, the difference between the multibody humanoid robot dynamics and 3D-LIPM and perturbations, such as external forces and uneven terrain, will prevent the ZMP to match the reference. The robot is able to accommodate ZMP error up to the margins of the support polygon without tipping, which is often suffcient to allow open-loop walking if no strong perturbations are present. Nevertheless, closed-loop balancing strategies are useful to make the walking more robust. We have tried using the angular velocities measured from the gyrometer to stabilize the walk, which proved effective.

4.3 Kick

We consider that kicking is a motion where the biped starts in a stand position, kicks the ball and returns to the same stand position. Moreover, during kicking, one foot is taken off the ground, henceforth referred as kicking foot, while

the other one is kept on the ground as support foot. This description suggests breaking the motion in phases, thus we divided it in the following 5 phases:

- Phase A: the robot moves the ZMP to the center of the support foot to allow the kicking foot to be taken off the ground in the next phase without balance loss.
- Phase B: the robot takes the foot off the ground and position it to prepare for kicking the ball.
- Phase C: the robot kicks the ball.
- Phase D: the robot places the kicking foot on the ground.
- Phase E: the robot goes back to the stand position (ZMP is moved to the torso projection on the ground).

During phases B, C and D, the robot is in single support, so stability is of concern. Based on this perception, we use the same algorithm we used to balance walking for balancing kicking. Again, we constraint the CoM to maintain a constant height zCoM, so the dynamics becomes linear.

4.4 Keyframe Movement

A Keyframe movement is a movement where the positions of the joints are defined with joint angle and the time when that position must be achieved. When the time is between some of the predefine times, the joints positions are interpolated. The interpolation used for the joints is a spline interpolation, implemented in the Library.

Each Keyframe has a specific archive that has all the joints in discrete time steps.

Optimization To achieve better results with the keyframe movements, an optimization was used, optimizing the joints positions and the time between joints. The algorithm used for the Optimization was Covariance Matrix Adaptation Evolution Strategy [13], and the cost functions were adapted for each optimization problem.

Converter As it is known, the team had a base made from Magma Offenburg, and so, to avoid recreating the movements, a converter was created, and many movements were converted and heuristically improved. The converter considers that the same amount of time passes between two keyframes.

5 Strategy and Decision Making

5.1 Positioning

An algorithm was implemented to determine the positioning of our players when the ball is with the opposing team. The inspiration for this idea becomes from

Agent2D. We defined that the player who is closest to the ball will go to it, while the others will go to their positions defined by this algorithm. In this algorithm, the position of each player depends only on where the ball is on the field, i.e., the position is independent of the positions of the opponent players. This returns us a reasonably satisfactory field positioning. Initially, the positions of the 11 players for some ball positions in the field were defined, being strategically chosen so as to cover the different sectors of the field. With these ball positions, was created a Delaunay triangulation in order to obtain over the field a set of triangles, whose acute angles were as large as possible. Knowing that the vertices of these triangles represent ball positions whose players formation for them is already set, it was used a linear interpolation algorithm on the vertices of these triangles, called Gouraud's Algorithm [12], to determine the formation of the players to some ball position within that triangle that does not have a players positioning initially set. The same weighted average the algorithm realized with the vertices of the triangle for the ball position in its interior, is also used with the positions of each player to return an approximate position for each of them when the ball is in that position. Thus, it's calculated a position for each of our players when the ball is with the opposing team.

5.2 Robot Navigation

Motivation For the movement with the ball, we implemented a new method called Potential Field [15]. 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. Potential fields are a part of grid algorithms. They use the metaphor of electric field. So, imagine a charged particle navigating in an electric field. This particle will be attracted by particles with the opposite charge and repelled by another charged particles. Thus, in this case, the ball should represents an attractive charge and others players represents a repulsive charge. In such a way we can obtain the fastest path to get to the goal. In other words, to get to the safety range it does not mean the robot will implicitly strike on an obstacle but it is not safe. This range can be simulated by continuously increasing functions in the direction to obstacles. In such a way we can obtain a compromise between speed and safety, which is the most important criterion in real applications.

We have several advantages of using Potential Fields to robot navigation [14]. 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; they can be easily parametrized and configured during design phase or in real-time and the combination linear (the resultant force is the linear sum of each independent force) is flexible and can be tweaked with gains to reflect varying importance of sub-behaviors.

Implementation In our case, the function **calculateDirection()** receives our current position and a vector with all objects in the soccer field. For example: players (both from our team and from the adversary), the ball, the goal, etc. To represent each object, we defined a class called **PotentialFieldObjects** that has the position and a numerical parameter. This parameter describes its condition (if repulsive or attractive). With this value of position and this constant, we can 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 our point. We use a conventional sign for the value of potential. In our case, the negative sign indicates repulsion and the positive sign indicates attraction.

$$V_{rep} = \frac{k}{d^3}, k < 0 \tag{6}$$

$$V_{attrac} = \frac{k}{d}, k > 0 \tag{7}$$

However, as we are interested in a direction, our function calculates the potential associated with each direction (horizontal and vertical).

$$V_x = \left(\frac{x - x_0}{d}\right) V \tag{8}$$

$$V_y = \left(\frac{y - y_0}{d}\right) V \tag{9}$$

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:

$$\theta = \arctan\left(\frac{V_y}{V_x}\right) \tag{10}$$

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 Conclusions and Future Work

It's been saw that, even though we remade our code structure, we see that there is much more to do. We need to improve our movements so that the agent can be at a higher level, and the strategy has to worry less with movement restriction.

We intend to develop a Log Analyzer tool, so that it can be easier to get information from a match and not have to watch the entire match.

It's is also important that is established a high informative level of communication between the agents, so that each agent can have more information to decide on.

Acknowledgements

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 basis, helping us develop a new code. We also thank the company Radix for sponsoring the team.

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. M. Maximo, "Otimização de Caminhada de Robôs Humanóides," B.S. Thesis, Technological Institute of Aeronautics, 2012.
3. 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).
4. 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).
5. 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).
6. Hester, T., Stone, P.: Negative Information and Line Observations for Monte Carlo Localization. In: IEEE International Conference on Robotics and Automation (ICRA'08) (2008).
7. 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).
8. Thrun, S., Burgard, W., Fox, D.: Probabilistic Robotics. MIT Press (2005).
9. 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).
10. Smith, A.F.M., Gelfand, A.E.: Bayesian Statistics Without Tears: A Sampling-Resampling Perspective. American Statistician 46, pp. 84–88 (1992).
11. 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.
12. Gouraud, H. (1971). Computer display of curved surfaces. 1-80. UTEC-71-113; UTEC-CSc-71-113
13. N. Hansen and S. Kern. Evaluating the CMA Evolution Strategy on Multimodal Test Functions. In Eighth International Conference on Parallel Problem Solving from Nature PPSN VIII, Proceedings, pp. 282-291, Berlin: Springer, 2004.
14. Hellstrom, Thomas. Robot Navigation with Potential Fields. Department of Computer Science, Umea University, 2011.
15. 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.