NAO-Team Humboldt 2009

The RoboCup NAO Team of Humboldt-Universität zu Berlin

Alexander Borisov, Alireza Ferdowsizadeh, Christian Mohr, Heinrich Mellmann, Martin Martius, Thomas Krause, Tobias Hermann, Oliver Welter, and Yuan Xu

Institut für Informatik, LFG Künstliche Intelligenz, Humboldt-Universität zu Berlin, Rudower Chaussee 25, 12489 Berlin, Germany.

http://www.naoteamhumboldt.de
nao-team@informatik.hu-berlin.de

Fig. 1. The NaoTeamHumboldt at GermanOpen in Hannover, 2009. From left to right: (front) Yuan Xu, Thomas Krause, Heinrich Mellmann, Martin Martius; (rear) Prof. Dr. Hans-Dieter Burkhard, Tobias Hermann, Christian Mohr, Alexander Borisov, Prof. Dr. nat. Verena Hafner; Not in the image: Oliver Welter, Alireza Ferdowsizadeh;

1 Introduction

The NAO-Team Humboldt was founded at the end of 2007 and consists of students and researchers from the Humboldt-Universität zu Berlin. Some of the team members have had a long tradition within RoboCup by working for the

Four Legged league as a part of the GermanTeam in recent years. Though we used some concepts and ideas from the GT-platform, the software architecture was written totally new. Additionally, we developed several tools such as RobotControl and MotionEditor for testing, debugging or for creating new motion nets.

The first NAOs arrived in May 2008, so we had only 2 months for developing and testing algorithms before we participated in RoboCup 2008 in Suzhou. Despite this fact, we achieved the 4. place.

2 Architecture

Since the Humboldt-University has a long history in RoboCup there is a lot of experience and already existing code. We decided to implement a module based architecture which shares some concepts with the GermanTeam architecture from 2007.

2.1 Framework

Compared to GT2007 we tried to enhance simplicity and tried to avoid to have a too heavy framework. The programming language on the robot is C++.

We decided to divide our code into two parts: a part which is independent of the environment (Core) and a specific one. This enabled us to do a lot of development without having the robots and without having to re-implement everything on the real robot. Besides the real robot we use Webots and SimSpark for simulation and have also a special webcam-interface for experiments with cognition modules without running the real robot.

2.2 Debugging

We implemented a debugging over network possibility with the help of very simple but powerful debug concept.

It is possible to enable or disable certain debug code during runtime using debug requests. Debug drawings are used to visualize e.g. results of the image processing or the self localization. Our remote debugging tool RobotControl has been implemented in Java. We tried to avoid having a dialog for every possible debugging activity. Instead the console can be used if there is no graphical dialog needed.

3 Vision System

Our vision system is based on YUV444 images. This year we increased the resolution from 160x120 to 320x240 because of the limitations in object-recognition of the lower resolution.
Most of the algorithms are based on color classification. The classification is done

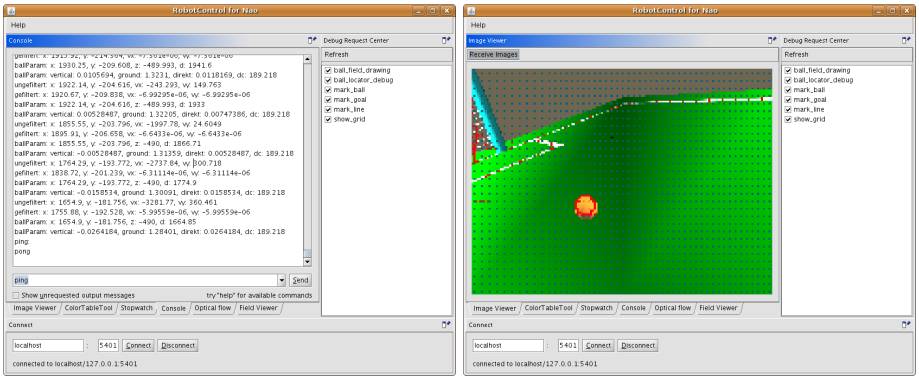

Fig. 2. RobotControl in action. On the left side you see the console, the right picture shows the image viewer with some enabled debug requests

using two different means - a look up table and especially for complementary colors, linear color space segmentation. Since it is, even in this resolution, inefficient to classify the whole picture we employ a grid for this task. The grid is laid over the picture, has a resolution of 80x60 and thus classifies only one in four pixels. It provides a list of all pixels from a given color class to the subsequent procedures which are as follows.

3.1 Goal Detection

The algorithm for goal detection relies mainly on the calculation of statistical measures for lists of equally colored pixels provided by the grid. For one class of goal colored pixels the algorithm first looks for the best 2 candidate pixels maximizing x^2+y^2 or x^2-y^2 (thus minimizing the distance to the lower left/right corner) while in the same iteration calculating all image moments up to 2nd degree of all pixels from this color class. Those moments are used to calculate the main axis for the distribution of pixels having goal color. The axis orientation gives a good hint wether both goal posts only the right or only the left one are on the image. Having gathered this information the algorithm knows how to interpret the 2 candidates found before and starts to explore via region growing from every pixel representing a goalposts base point thus further defining the candidates and finally filling the corresponding percept.

3.2 Line Detection

Line Detection is done without color classification. It instead scans the whole picture in horizontal an vertical direction looking for rises and subsequent falls in the images gray value function indicating the possible start and end of a line. Having found two of those points the edge angle is calculated by the Sobel

Operator and then as well as the two pixels positions averaged resulting in a new point and corresponding angle in the middle of these points. Points found this way are then first clustered by their angle and second by their probability of lying on one line. Clusters with a sufficient number of members are then accepted as lines.

3.3 Robot Detection

Robot Detection is done using red or blue color areas in the image. Those blobs representing distinct parts of a robot (i.e. head, shoulders, feet etc.) must have certain attributes such as certain area, center of mass, orientation or excentricity. Body part candidates identified this way are then tried to be grouped to form a robot. The position and orientation of the detected robot can easily be extracted from the geometric relations between the constituting color areas.

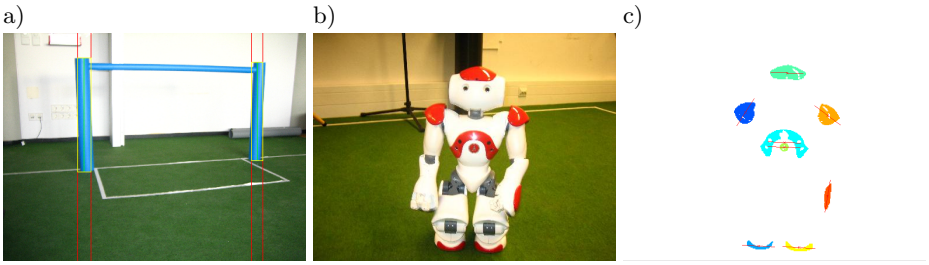

Fig. 3. a) A recognized goal, depicted by the two rectangles framing the goal posts; b) a seen robot; c) detected robot areas including their excentricity

3.4 Camera Matrix

For assigning the correct distances and angles to the recognized objects in the image we calculate the kinematic chain of the robot from the feet to the head. So bearing based distance measurements are possible for a variety of robot poses, e.g. perching or standing upright. The accelerations sensors, together with the foot sensors, help us to decide whether the robot is standing or has been falling down.

4 World Model

Our approach to represent the world state is to use different models for different objects. In having this separation we can have very effective and special models for each individual object type. We distinguish three modeling approaches - self localization, ball modeling, player modeling.

4.1 Ball Modeling

Tracking the ball is most important for the attacker and for the goal keeper. Since playing passes was hard with the Naos so far, we used a local model for each robot to track the ball. Still, by using self localization information we were able to communicate global ball positions to the other player. A Kalman filter was used to generate the local ball model.

4.2 Player Modeling

Having data over recognized field players we also want to have a player model to recognize friendly passing partners and to avoid kicking the ball into opponent players. Right now we want to find out which kind of model fits our needs best. Therefore we evaluate the advantages of a model which tracks all the different players seperately against those of a model which just takes occupied regions into account.

4.3 Localization

We now work on two different methods for self–localization: monte-carlo localization based on particle filter and constraint self–localization. We will explain these methods in the following sections.

4.4 Monte-Carlo Localization

Self-localization based on particle filters [1] works like this: We create a random array of particles within the field size. Each particle is a possible position of the robot in the field and therefore each particle consists of three components (x, y, θ), which represent the coordinates of the robot and rotation. Then for each particle i the likelihood ω_i is calculated. For this, we subtract the seen distance from the expected distance, and get d_{sim}. The same is done for the angle, resulting in θ_{sim}. The particle likelihood for a seen and identified goal post is then calculated as: $\omega_i = e^{\frac{d_{sim}^2}{\sigma^2}}$. After normalizing the likelihoods the particle are resampled.

Our next step is the calculation of the largest cluster of particles. For this purpose we draw a grid over the field and calculate the number of particles in each grid cell. The cell with the highest amount of particles is the best possible choice for a the location of the robot. We calculate the average of the coordinates of particles within this particular grid cell and input the date into the robot position:

$$\overline{X} = \frac{\sum_{j=0}^{n} x_i}{n}$$

$$\overline{Y} = \frac{\sum_{j=0}^{n} y_i}{n}$$

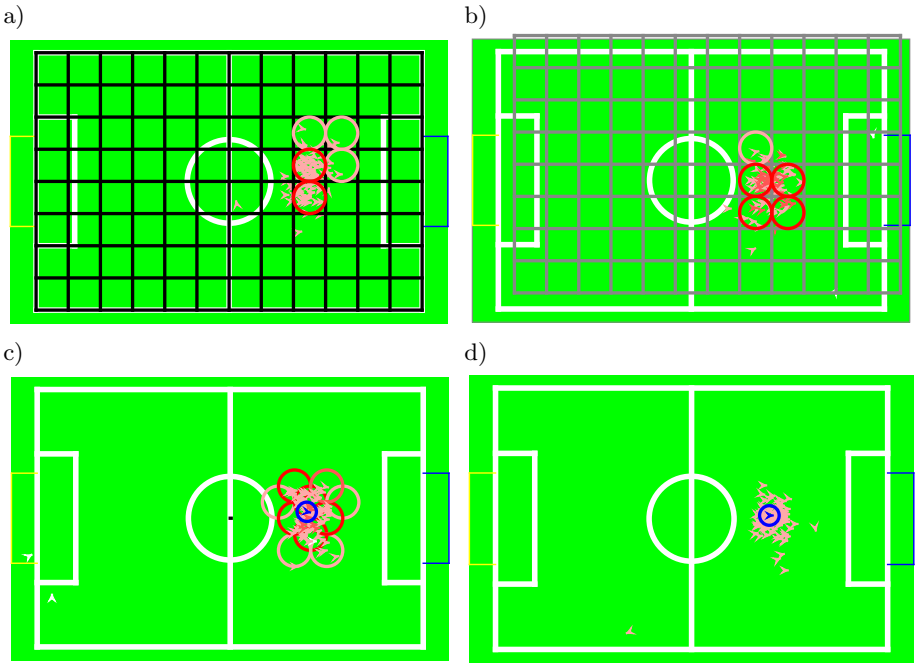

Fig. 4. a) The particles are clustered using a grid. Cells with the most particles are marked with red circles. b) Analogous, the particles are clustered with a shifted grid. c) Red circles mark the cells with the most particles (see a and b). The small blue circle with arrow inside it illustrates the estimated position of the robot as the center of gravity of the particles in the best grid cells. d) The particles and the estimated robot pose is illustrated.

$$\overline{\theta} = \arctan\left(\frac{\sum_{j=0}^{n} \sin\theta_j}{\sum_{j=0}^{n} \cos\theta_j}\right)$$

whereas n is the number of particles and is equal to 100 in our case.

After calculating the largest cluster we have noticed the largest cluster might jump from one grid cell to another grid cell. In the figure above, we see that there is only one largest cluster (the blue circle), but there are also two clusters of particles which are nearly as large as the largest one (two red circles). It happens if the cell border is exactly over the highest concentration of particles. So there is a large amount of particles in at least two neighboring grid cells. There is a high possibility that the largest cluster switches between these two or even more cells. We draw a second grid over the field to avoid this problem. This second grid is displaced by 50 percent of the grid cell largeness of the first grid. Then we locate the largest cluster in both grids and calculate the average of particles. We have found out that it dramatically reduces the possibility of the largest cluster jumping between the grid cells.

The picture illustrates the localization of NAO using the particle filter.

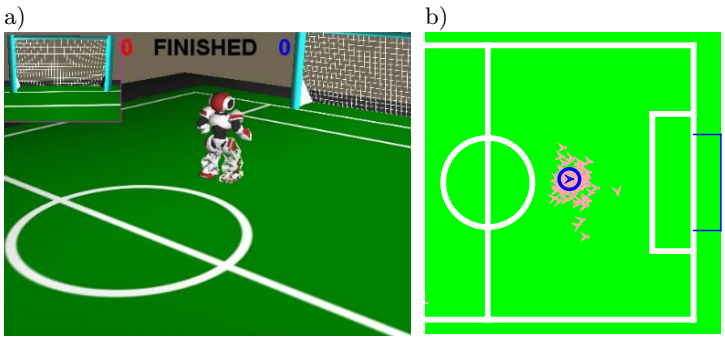

Fig. 5. Self-localization using the particle filter. a) The robot situated on the field. b) The estimated position of the robot calculated with the particle filter is marked as a blue circle.

4.5 Constraint Based Self–Localization

The reduction of flags during the last years combined with an increasing field size made the use of field lines for localization purposes more and more necessary. In earlier years particle filters proved to be an adequate choice for self localization. But it showed that the limited particle number can be problematic when using sensory data from field lines. For low particle numbers it is difficult to represent complex belief function. That is why we decided to try constraint based localization approaches, where the belief function is not being approximated by particles or Gaussians, but by constraint functions [2, 3].

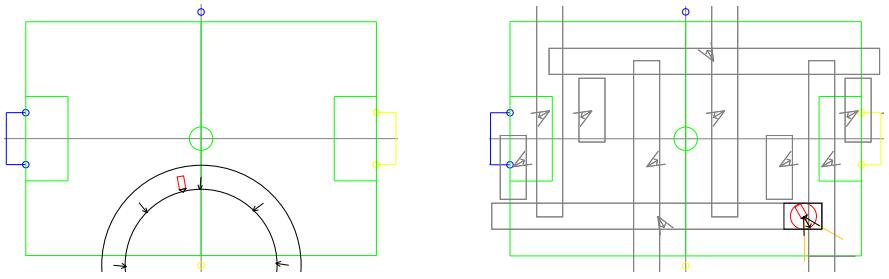

Fig. 6. Both figures show constraints, generated from percepts (pictures show the algorithm on an Aibo field). Left: a circular constraint as generated from flag percepts. Right: a line constraint, generated from line percepts.

For self localization we consider percepts from flags, goals and from lines. The shape of the constraint is determined by the kind of sensor data and expected sensor noise, which is dependent from the percept distance. After having generated all constraints, we propagate the constraints with each other as long as there are no more constraints or the resulting solution space becomes empty. The position belief of the robot is stored in form of constraints as well and propagated with the sensory constraints as well. If, for some steps, the belief doesn't fit to the sensory constraints, or even if no new sensory data are available, the belief constraint boarders are increased at first. If sensory data remain inconsistent, we reset the belief to the sensor data constraints.

5 Behavior

A general hierarchical behavior architecture has been used as a base for the behavior structure. For implementing our behavior we use XABSL (Extensible Agent Behavior Specification Language) [4]. This is a language to describe behaviors for autonomous agents. To allow effective behavior development we created a tool named *XABSL Editor*. In fact, this tool consists of an full featured editor with syntax highliting, a graph viewer for visualization of behavior state machines and an integrated compiler. Figure 7 illustrates the *XABSL Editor* which an open behavior file.

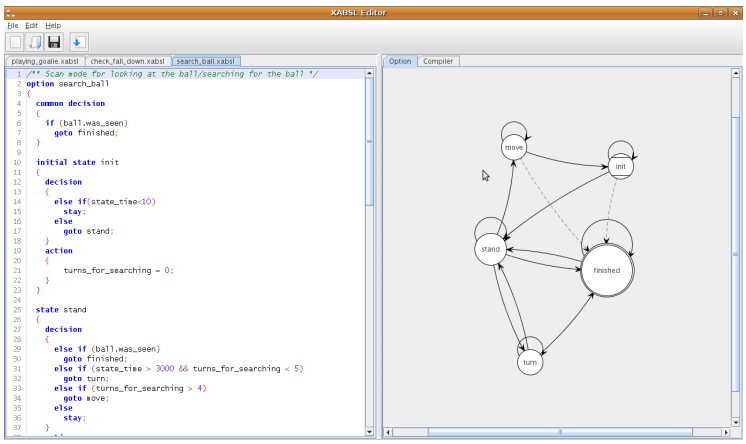

Fig. 7. XABSL Editor. On the left side you see the source code of an behavior option. On the right side the state machine of this option is visualized as a graph.

5.1 Behavior Architecture

This architecture simply allows layered use of different behaviors, from very low-level motion ones to high-level cooperative team skills or strategic decisions. Each option in behavior tree decides the running time of it's direct active sub-options in case of being activated itself, though each one takes care of it's own timing. Furthermore options have some internal state criteria, used as inter-option communication means, mainly utilized by sub-options and their super options. Each option can also have a possible transition condition, through which along with the option's state propagation, the active behavior-switching in any decision level can be safely accomplished.

5.2 Deterministic Fixed Behaviors

We have implemented a set of options based on the aforementioned structure. These consist of some basic options that trigger the low-level motions using our motion engine, and then more higher level behaviors. Basic options are for example `Move`, `Shoot` etc. On top of these basic options we have implemented a set of more complex options like `SearchFor`, `GoToPoint`, `GoToBall`, `TurnToAngle` and `TurnToShoot`. On one of higher decision levels, we have also implemented different role's behaviors such as `Defender`, `Attacker`, `Supporter` and `Goalie`. The roles are also allocated to the players on the field dynamically at run-time, and will be switched if necessary using inter-player communication based on game situations. By putting these options together to an option tree, a simple game-play behavior emerges. This set of behaviors in different levels are increasingly growing at the moment and we are also working on having more cooperative game skills, strategies and planning.

5.3 Dynamic Behavior Modeling

Here we are using a combination of different techniques. Besides hand-coded tuned behaviors, some dynamic behavior models, using improved fast reinforcement learning (RL) algorithms, are being developed as well; of course mainly with the help of Webots simulation environment. Particularly we are experimenting using a novel fuzzy reinforcement learning algorithm called Multi-Dimensional Group-wise Interpolation-based Fuzzy Actor Critic Learning, in order to acquire some dynamic behaviors. These methods are using actor-critic structure and (Self-Organized) Fuzzy Inference System (FIS) as the value function approximator in the usual RL, in order to increase the learning speed and performance. The experiments are currently planned for some rather intermediate-level to high-level options like kick, ball interception, pass or play options, since after evaluation in simulator they can be easily transferred on Nao with further small variations. But later of course some dynamic low-level skills and motions can also be developed, provided with our planned robot monitoring system for online learning on the real robot. The use of dynamic behaviors specially will grow as we move more to have cooperative multi-agent behaviors.

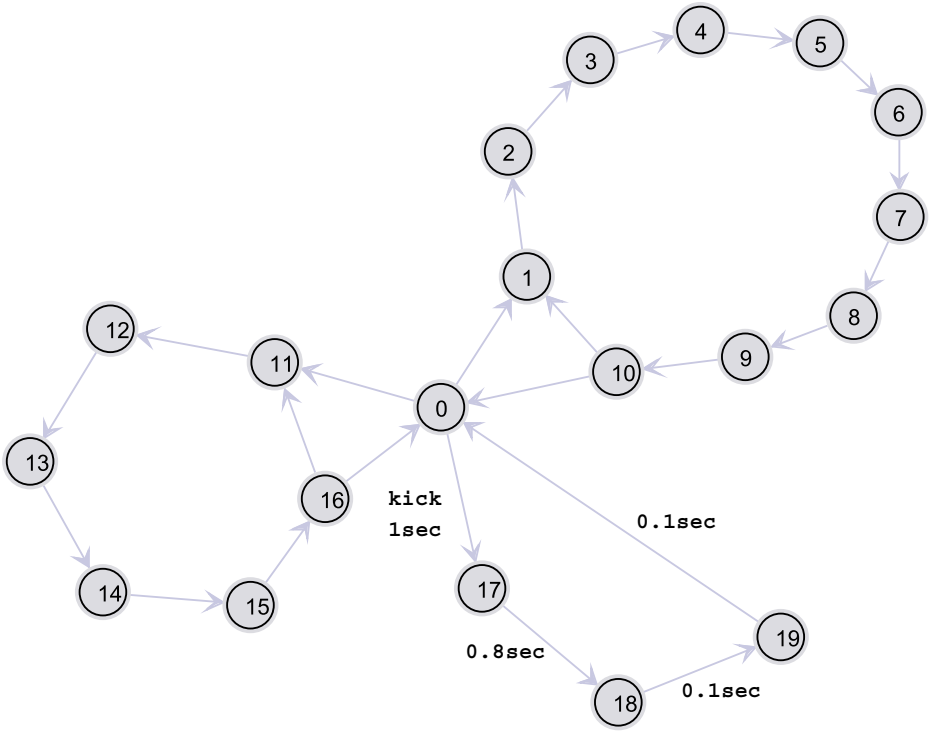

Fig. 8. Motion net; kick is specified with condition and duration

6 Motions

We use different methods for creating Motions. At the moment we use the Motions provided by Aldebaran for walking and turning. Special motions like standup are implemented with a key-frame based methode and inverse kinematics. (for detailed information see our Team Report 2008). We still describe a single motion net as a kind of DFSA (Deterministic Finite State Automaton). Each node describes the position of motors and each edge describes the transition between the two positions with a special condition and duration.

The advantage of the key–frame based method is an easy implementation, expandability and modifiability of the net. So it is relatively easy to manage and configure a given key–frame net. The disadvantage is the missing dynamics. That is why it is hardly used for complex motions like walking procedure or turning.

6.1 Inverse Kinematics

At the moment we use inverse kinematics for omnidirectional kicking and some parameterized motions. In the near future we will increase the functionality

of gait-generation together with inverse kinematics to provide a omnidirectional walk. In contrast to key–frame based motions we don't have to specify every angle of any joint which is involved within the process. We provide the information of distance we want to walk or the angle we want to rotate and the inverse kinematics provides all the values for the involved joints to perform the action. The kind of motion is based on several parameters which can be altered like step length, step height and velocity which gives us the opportunity to optimize each motion.

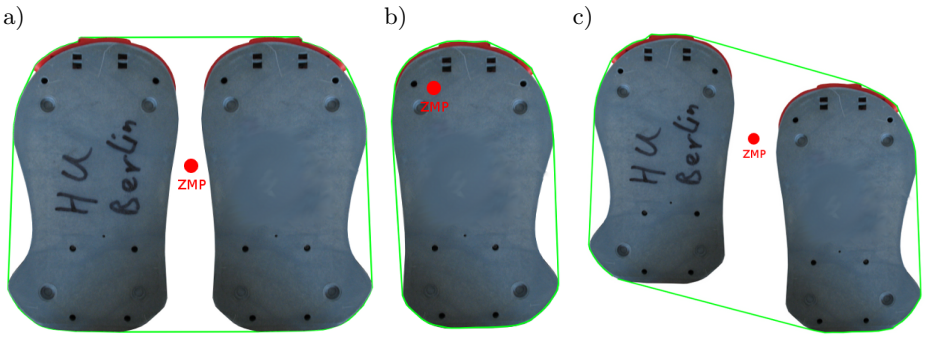

Fig. 9. Step forward (green line: support polygon): a) initial position with double support phase; b) One foot on the floor (single support phase); c) Both feet again on the floor (double support phase)

6.2 Zero-Moment-Point-Compensation

Since the key–frame engine and inverse kinematic are not able to deal with dynamic effects while moving, we are planning to compensate this with the help of a Zero-Moment-Point-compensator. The main idea for creating stable motions is based on analyzing the forces which occur inside and outside of the robot. The first step of the analysis consists of determining the current support polygon. The support polygon is the minimized polygon which connects all contact points of the robot with the floor. While performing motions this support polygon changes its position and shape. Since static stability constraints our motion a lot we are planning to use dynamic stability criterions to optimize our locomotion. We are going to use the concept of ZMP (Zero-Moment-Point) to check if the locomotion process is getting unstable and has to be adjusted. The ZMP is the point where all forces like acceleration and tilting are zero. It is necessary for stability criterion that this point is within the support polygon. The position of the ZMP can be determined by analyzing the CoM (Center of Mass) and with the knowledge of dynamic effects of the robot. A foot-trajectory will be given as input for the inverse kinematic or generated by the key–frame engine; and while the joints are controlled the ZMP-compensator will analyze the effects on the

CoM and calculate the position of the ZMP and will generate a hip-trajectory to guarantee that the ZMP is within the support polygon.

References

1. T. Röfer and M. Jüngel, "Vision-Based Fast and Reactive Monte-Carlo Localization," *IEEE International Conference on Robotics and Automation (ICRA-2003)*, pp. 856–861, 2003.
2. D. Goehring, H. Mellmann, and H.-D. Burkhard, "Constraint based belief modeling," in *RoboCup 2008: Robot Soccer World Cup XII* (L. Iocchi, H. Matsubara, A. Weitzenfeld, and C. Zhou, eds.), Lecture Notes in Artificial Intelligence, Springer. to appear.
3. D. Göhring, H. Mellmann, and H.-D. Burkhard, "Constraint based world modeling in mobile robotics," in *Proc. IEEE International Conference on Robotics and Automation ICRA 2009*, 2009. to appear.
4. M. Lötzsch, M. Risler, and M. Jüngel, "Xabsl - a pragmatic approach to behavior engineering," in *Proceedings of IEEE/RSJ International Conference of Intelligent Robots and Systems (IROS)*, (Beijing, China), pp. 5124–5129, October 9-15 2006.