Vision-Based Cognition of a Humanoid Robot in Standard Platform Robot Soccer

Somchaya Liemhetcharat [1], Brian Coltin [2], and Manuela Veloso [3]

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213, USA
[1]som@ri.cmu.edu
[2]bcoltin@cs.cmu.edu
[3]veloso@cs.cmu.edu

***Abstract*— In the Standard Platform League of RoboCup, teams compete using the same hardware platform — the NAO humanoid robot. The NAO has two cameras, but only one can be used at one time. The camera image is the primary input to the robot's cognition; the features detected by the vision module are used to self-localize and model objects in the world. In this paper, we focus on how the output of the vision module, the vision features, are used by the localization module to determine the robot's pose, and by the world model module to model objects in the world such as the ball and goal.**

I. Introduction

In the Standard Platform League of RoboCup [1], [2], all the teams use the same hardware platform, a NAO humanoid robot (Fig. 1) [3], where all the processing is done on the robots, i.e., there are no off-board computers or cameras. We are interested in teams of robots with onboard perception, control, actuation, and communication capabilities. In order for a team of robots to perform well playing soccer, each robot must be capable of cognition based on its sensory input. However, each robot's view of the world is limited, due to its spatial position and narrow field of view of its camera. As such, in addition to visual processing of the camera image, the robot has to model its position in the world, i.e., localization, and model other objects in the world, i.e., world modeling.

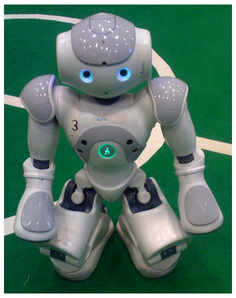

Fig. 1. NAO humanoid robot used in the RoboCup Standard Platform League.

The NAO humanoid robot has two cameras in its head, but it is only able to access one camera at a time, due to a shared hardware bus. The cameras are the main sensors of the robot, and the output of the vision module is the primary input for the other cognition modules, namely localization and world modeling. The objects in RoboCup are typically color-coded, e.g., balls are orange, goals are yellow and blue, and the vision module uses CMVision to segment the camera image into discrete colors before detecting objects [4]. When the color segmentation table is accurate, the objects in the camera image are detected accurately; when there are errors in color segmentation, e.g., due to a change in lighting conditions, objects may not be detected accurately or at all. Thus, the use of vision as the primary input to the other cognition modules is a double-edged sword — the performance of the other modules are strongly correlated with the accuracy of vision.

While vision is important to the robots, it is not their primary goal — the goal of the multi-humanoid team is to score goals playing soccer. However, scoring goals effectively requires accurate vision, good cognition of its position on the field, and the positions of other objects in the world.

In this paper, we formally define the robot cognition domain, which includes the sensor readings of the robot, the sets of vision features, robot poses, and object models. We then define the vision, localization and world model modules as functions that update the vision features, robot pose and object models respectively. These modules are then used to define the robot cognition problem, which we address in this paper.

Localization estimates the pose of the robot, and there is extensive previous related work. Monte Carlo methods are commonly used to perform localization on mobile robots [5], and other methods include grid-based Markov localization and Kalman filtering [6]. In order to perform localization, a sensor model must be developed [7], and we detail our method in this paper. In addition, due to the dynamic nature of RoboCup, where robots can be taken off the field due to a penalty, we use sensor resetting when new landmarks are detected [8].

World modeling involves creating models of objects in the world, which is used to generate the behaviors of the robot. The NAO humanoid robots have a limited field of view and are thus unable to perceive all objects in the world continuously. In order to perform its task of scoring goals, the robot has to look at different objects at different times. Thus, the world model module provides models of these objects when they are not currently in view. There has been extensive previous related work in world modeling, such as in tracking and modeling the actions of the robot and its teammates [9], and fusing information from multiple hypotheses and sources [10], [11]. A thorough explanation of our world modeling is provided in [12]. In this paper, we focus on the world modeling with regards to its dependence on vision, and how we process visual information in the world model.

II. Problem Statement

Definition 1: The **robot cognition domain** is a tuple $D = \{O, S, \mathcal{V}, \mathcal{P}, \mathcal{M}\}$, where:

- O is the set of labels of objects in the world;
- S is the set of possible sensor readings
- $\mathcal{V}$ is the set of possible vision features;
- $\mathcal{P}$ is the set of possible poses of the robot;
- $\mathcal{M}$ is the set of models of objects.

A sensor reading $s \in S$ contains all the inputs to the robot, e.g., the joint angles, camera image, odometry update, messages from other robots, and the state of the game.

The raw camera image in s is used by the vision module to generate vision features. Vision features $v \in \mathcal{V}$ provide relative coordinates to objects detected in the current camera image, i.e., $v : O \to (x_r, y_r)$, where (x_r, y_r) are the relative coordinates of the object from the robot.

Definition 2: The **vision** module processes sensor readings and generates vision features, i.e., $\texttt{vision} : S \to \mathcal{V}$.

A pose of the robot $p \in \mathcal{P}$ provides global coordinates of objects, in particular the robot itself, i.e., $p : O \to (x_g, y_g)$, where (x_g, y_g) are the global coordinates of the object.

Definition 3: The **localization** module processes the sensor readings and vision features and updates the pose, i.e., $\texttt{localize} : \mathcal{P} \times S \times \mathcal{V} \to \mathcal{P}$.

In other words, $\texttt{localize}(p, s, v) = p'$, where p' is an updated pose, using the previous pose p, sensor reading s and vision features v.

A model of the objects $m \in \mathcal{M}$ provides relative coordinates of the objects, i.e., $m : O \to (x_r, y_r)$, where (x_r, y_r) are the relative coordinates of the object to the robot. The main difference between a model m and vision feature v is that v returns coordinates of objects that are currently visible, while m models objects in the world, regardless of whether or not they are currently visible.

Definition 4: The **world model** module updates the object models, by processing the sensor readings, vision features and current pose, i.e., $\texttt{worldmodel} : \mathcal{M} \times S \times \mathcal{V} \times \mathcal{P} \to \mathcal{M}$.

After defining the modules above, we can now formally define the robot cognition problem:

Definition 5: The **robot cognition problem** is a tuple $\{D, V, L, W\}$, with the goal of obtaining accurate poses $\mathcal{P} \in D$ and models $\mathcal{M} \in D$ using V, L, W, where:

- D is the robot cognition domain;
- V is the vision module;
- L is the localization module;
- W is the world model module.

In this paper, we do not elaborate on the vision module, and focus on how the vision features (the output of the vision module) is used by the localization and world model modules.

III. Localization

Each robot must localize to determine its own position on the field in global coordinates based on vision and other sensor readings. The robot's global position is needed to shoot at the opponent's goal, defend the team's goal, and to provide a common coordinate frame for sharing information in the world model, such as the ball's position. Recall that localization is a function $\texttt{localize} : \mathcal{P} \times S \times \mathcal{V} \to \mathcal{P}$. $\texttt{localize}(p, s, v)$ takes a current pose estimate p, sensor readings s, and vision features v, and returns an updated pose estimate.

To update the pose estimate, we use Monte-Carlo localization with sensor resetting [8]. The pose p is a tuple of the form (c, a, w), where $c \in \mathcal{C}$ is the estimated pose in the configuration space $\mathcal{C}$ of the robot, a is a list of n configurations $c_i \in \mathcal{C}$ representing the possible configurations, or particles, maintained by the particle filter, and w is a list of n real numbers $w_i \in \mathbb{R}$ representing the weights, or likelihoods, of the particles. The number of particles n varies over time. See Algorithm 1 for a high-level overview of the localization algorithm. First, each of the particles is updated based on information from the game controller (the referee). The position of each particle is then updated based on the predicted motion, and finally the weight of each particle is updated based on how well it matches sensor inputs. Lastly, if any landmarks were visible, new particles are generated based on the distribution of the particle weights. We will discuss each step of the localization process in detail.

Algorithm 1 $\texttt{localize}(p, s, v)$ - Update the robot's pose based on sensing.

```
$(c, a, w) \leftarrow p$
**for** $i = 1$ to $n$ **do**
    $(a_i, w_i) \leftarrow \texttt{game\_controller\_update}(s, a_i, w_i)$
    $a_i \leftarrow \texttt{predict}(s, a_i)$
    $w_i \leftarrow \texttt{update}(v, a_i, w_i)$
**end for**
**if** $v$ not empty **then**
    $(l, w) \leftarrow \texttt{resample}(v, l, w)$
**end if**
**return** $(\texttt{compute\_pose}(l, w), l, w)$
```

A. Special Cases — Game Controller Update

First, each of the particles is updated based on information from the game controller. The game controller sends wireless messages to the robots from the referee based on the state of the game — for example, when goals are scored or a half ends. The game_controller_update function modifies the particles in three situations.

1) **Side Switch**. The global coordinate system used is relative to which side the robot is playing on, so that the team's own goal is always in the negative x direction and the opponent's goal is always in the positive x direction. If the two teams switch colors, e.g., during half time, the

x and y coordinates of all the particles are negated and the angle θ is rotated by $180°$.

2) **Initial Positions**. At the beginning of a half or after a goal is scored, the robots must return to their side of the field for kickoff. If a robot is in an invalid position or the team requests it, the referee manually places the robots in a predefined initial position. If according to the game controller the robots are heading to their initial positions and the robot's feet both leave the ground, all the particles are moved to the vicinity of the predefined initial positions.
3) **Penalties**. When a robot is penalized, it is temporarily removed from the game and then returned to play at the midfield line on the side opposite the ball, or possibly at the penalty point for the goalie. The game controller informs the robot when it is penalized or unpenalized, and the particles are evenly distributed between the possible return positions.

B. Robot Motion — Predict Step

In the function `predict` $: \mathcal{S} \times \mathcal{C} \rightarrow \mathcal{C}$, the positions of the particles are updated based on the motion model. The sensor readings s passed as an input to `predict` include the odometry information since the last call to predict, namely the changes in translation dx and dy and the change in orientation $d\theta$. This motion is applied to each of the particles with the addition of noise (see Algorithm 2.)

Algorithm 2 `predict`(s, c) - Update the particle position estimates, adding noise.

```
$(x, y, \theta) \leftarrow c$
$(dx, dy, d\theta) \leftarrow s$
$dx \leftarrow \mathcal{N}(dx, dx * \sigma_{mx}^2)$
$dy \leftarrow \mathcal{N}(dy, dy * \sigma_{mx}^2)$
$d\theta \leftarrow \mathcal{N}(d\theta, d\theta * \sigma_{m\theta}^2))$
$d\theta \leftarrow \mathcal{N}(d\theta, dy * \sigma_{my\theta}^2)$
return $\begin{bmatrix} x \\ y \\ \theta \end{bmatrix} + \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} dx \\ dy \\ d\theta \end{bmatrix}$
```

Noise is added proportional to the reported distance travelled in the translational components. Rotational noise is added proportional to both the reported rotation and the reported motion sideways in the y direction since sidesteps are especially non-straight. We set the standard deviation of the translational noise to $\sigma_{mx} = 0.3$. For the rotational direction, we set $\sigma_{my\theta} = 0.003$ and $\sigma_{m\theta} = 0.4$. These values were chosen through trial and error.

With this motion model, the particles spread out over time as the robot moves, reducing certainty.

C. Vision — Update Step

In the update step, the likelihood of each of the particle's positions is computed based on the current landmarks detected from vision. The new particle weight is given by

$$w' = w \prod_{v_i \in v} p(v_i | c)$$

where w is the old weight of the particle and $p(v|c)$ represents the likelihood of making the observation of a landmark v given that the current configuration is c. We will discuss in detail the sensor model with which we compute $p(v|c)$ for each type of landmark.

Algorithm 3 `point_likelihood`$(v, obs\_pos, exp\_pos, \sigma_x, \sigma_\theta)$ - Compute the likelihood of observing an object at obs_pos, but expected to be at exp_pos.

```
$dx \leftarrow |\|obs\_pos\| - \|exp\_pos\||$
$d\theta \leftarrow$ angle_norm(angle($obs\_pos$) $-$ angle($exp\_pos$))
return $e^{-\frac{dx^2}{2\sigma_x^2}} e^{-\frac{d\theta^2}{2\sigma_\theta^2}}$
```

As part of each landmark's likelihood computation, we use the generic function `point_likelihood`, where `point_likelihood`$(v, obs\_pos, exp\_pos, \sigma_x, \sigma_\theta)$ gives the likelihood of having observed an object at the relative position obs_pos, given that it was expected at a relative position exp_pos (see Algorithm 3). To compute the likelihood, we compute the difference in magnitude and angle of the two position vectors, and then take the probability of observing these values given two Gaussian distributions with standard deviations σ_x and σ_θ, respectively. The Gaussian probability density function does not need to be normalized since we are interested only in the relative weights of the particles. σ_x and σ_θ are currently specified manually and fine-tuned through trial and error; however, it is possible to learn them given a system to measure the ground truth of the world.

1) Goal Post Sensor Model: The vision module will detect either a left goal post, a right goal post, or an "unknown" goal post. For left or right goal posts, we use the result of the `point_likelihood` function, passing the expected position relative to the left or right post as a parameter. For unknown post detections, we call the function with both the position of the left post and right post, and take the value which is more likely (a higher return value). The variances in the Guassians for the goal post sensor model, σ_x and σ_θ, increase with distance as the object moves further from the camera, pixels become less precise, and small errors in the robot's joint angles have increasingly large effects. For goal posts, σ_x has a minimum value of 30 cm, and increases linearly with distance at a rate of 0.175 cm / cm. σ_θ has a minimum value of $10°$ and increases at a rate of $1.25°$ / m.

2) Corner Sensor Model: Corners are all similar to unknown goal posts in that each of the corners is ambiguous, i.e., it may correspond to multiple landmarks on the field. There are three distinct types of corners: T, L and X. The vision module reports the relative position and angle of each corner, as well as the type. Rather than computing the likelihood of observing each possible corner from the map and choosing the corner of maximal likelihood, as for unknown goal posts, we use a decision tree to efficiently find the nearest corner on the map to the observed corner. This decision tree spatially partitions the field, and for each particle, we choose the corner nearest to the global position where the corner was observed. We then use the `point_likelihood` function to compute the likelihood of observing the corner at this position, and multiply the weight from this function by a Gaussian probability density representing the likelihood of the observed angle of the corner, with $\sigma = 10°$. σ_x for corners begins at 10 cm and increases linearly with distance at a rate of 0.25 cm / cm. $\sigma_\theta = 10°$ for all distances, since corners are only visible from a relatively short distance of at most one meter, and the angular variations are small throughout this window.

3) Line Sensor Model: To model the probability of observing lines, we use a similar method to Hester and Stone in [13]. First, lines shorter than 10 cm are discarded, as they are often segments of the center circle or other robots. For each observed line segment, we take the point p on the line segment nearest to the robot. Then, we use a decision tree, as with corners, to find the most likely observed line for the particle (the tree for lines first splits the search space based on orientation and then based on position). We then take the point $\hat{p}$ on the nearest line segment closest to the robot. The update weight is then computed by the function `point_likelihood` with p as the observation in relative coordinates and $\hat{p}$ as the expected observation. σ_x has a minimum value of 40 cm and increases at a rate of 0.5 cm / cm, while $\sigma_\theta = 30°$. The variance is quite high for the distance of the lines, because their sensing with vision is quite poor. We multiply the weight returned by the `point_likelihood` function by a Gaussian probability density representing the likelihood of measuring the expected line at the observed orientation, with a variance of 10° (the orientation of the line returned from vision is quite accurate). Line detection in vision also suffers from a large number of false positives, so 0.9 is added to the weight of all line observations. This prevents a loss of diversity based only on inaccurate line observations.

4) Center Circle Sensor Model: The center circle is detected in vision by combining shorter straight line segments into a circle with a common centroid, similar to an algorithm presented in [14]. The resulting circle is compared to a point landmark at the center of the field using the *point_likelihood* function. σ_x begins at 10 cm and increases at a rate of 0.1 cm / cm, and $\sigma_\theta = 10°$.

D. Resample Step

In the resample step, the distribution of the weights of the particles is used to probabilistically generate a new sample set. See Algorithm 4 for an overview of the algorithm. First, the cumulative distribution function is calculated for the particles; that is, we normalize the weights of the particles, and then for each particle, we compute the sum of the weights of all particles before it. This allows us to draw from the distribution of the particle weights in logarithmic time with a binary search.

Algorithm 4 `resample`(v, l, w) - Update the positions l and weights w of the particles based on observations from vision v.

```
$cdf \leftarrow$ compute_cdf$(w)$
$weight_{total} \leftarrow 0$
$i \leftarrow 1$
while $i < 300$ and $weight_{total} < \frac{\alpha}{cdf_n}$ do
  if rand() $< p_{reset}$ then
    $(l_i, w_i) \leftarrow$ generate_from_vision$(v)$
  else
    $(l_i, w_i) \leftarrow$ sample$(l_{old}, cdf)$
  end if
  $weight_{total} \leftarrow weight_{total} + w_i$
end while
return $(l, w)$
```

Then, we generate the new particles. For each particle, with a probability dependent on the history of the average weight of the particles, we generate a new particle using sensor resetting [8]. The particles are generated based on vision of goal posts, so this step is skipped if no goal posts are visible. If two goal posts are visible, the particles are generated at the points on the field where the two circles of the observed distances from the two goal posts intersect. If only one goal post is visible, the particles are generated at random in a circle surrounding that goal post. Noise is applied to the distance and angle measurements before generating the particles, using the same Gaussian distributions as the goal post sensing model. Sensor resetting allows the kidnapped robot problem to be solved more effectively by moving the hypotheses quickly to the desired area.

If the particle is not generated using sensor resetting, it is generated based on the previous distribution of the particles. A particle is chosen at random from the old distribution using binary search on the cumulative distribution function. Noise is applied to the newly generated particles to encourage diversity, with a variance of 8 cm in the x and y directions, and 10° in the angular direction.

We continue generating new particles until either the maximum limit of 300 particles is reached or until the total weight of the particles is greater than a number inversely proportional to the total weight of the previous particles. So as the confidence in our estimates increases, the number of particles decreases. This behavior allows more particles to be used when we are uncertain of the robot's position, and less when we are more certain, which reduces computation.

E. Final Pose Selection

The particle filter maintains a collection of particles representing the hypotheses about the robot's position. However, the behaviors act based on a single estimate of the robot's position. We must fuse the particles into a single position estimate.

First, we find the particle of greatest weight which is within a set distance and angle of our previous position estimate. This helps to reduce jumping in the position estimate, and the resulting jumps in the robot's behavior. We take all particles within 40 cm of the chosen particle, and choose the centroid of this cluster as the robot's position. Choosing the centroid helps to smooth the robot's trajectory. If the total weight of particles within this cluster is small, we allow the robot to jump to the particle of greatest weight.

Finally, localization reports its certainty in the position to behaviors. If the standard deviation of all the weighted particle's x and y coordinates relative to the selected pose is lower than a certain threshold, localization reports to behaviors that the robot is lost.

IV. World Model

While the vision module process the current camera image and returns visible objects in relative coordinates, the robot's behaviors should take into account objects that are not currently visible. To do so, the world model keeps track of objects relevant to the behaviors of the robot, and provides an estimate of their positions in relative coordinates. These coordinates can be converted to global coordinates through localization if required. Recall that the world model process is a function `worldmodel` $: \mathcal{M} \times S \times \mathcal{V} \times \mathcal{P} \rightarrow \mathcal{M}$. `worldmodel`$(m, s, v, p)$ takes a prior model m, sensor readings s, vision features v, and pose p, and returns an updated model. The updated model $m' : O \rightarrow (x_r, y_r)$ can then be queried for the relative coordinates of objects in the world.

The world model module models the ball, goal, obstacles, and teammates. In this paper, we focus on the ball and goal models, as they are essentially driven by vision. The obstacles are updated by ultrasonic readings, and the status of teammates are updated by their accelerometers and foot pressure sensors, which they report through wireless messages.

A. Updating the Ball

When the vision module detects a ball in the current camera image, the world model returns the same object, and overrides the current model of the ball. Vision thus provides a short-circuit to the behaviors when the ball is detected, which ensures that behaviors always react to the currently visible ball when one is present, and not a potentially outdated model.

When the ball is not currently visible, the model of the ball provides an estimate of where it should be, allowing behaviors to turn the robot's head and body in the direction if required. The ball is modeled using multiple hypotheses (each hypothesis is a possible location of the ball), and the highest ranked hypothesis is returned by the world model in each call. However, as more sensor readings are processed, the rankings of hypotheses change so as to allow the robot to switch to a more viable hypothesis. Each hypothesis of the ball's location has a confidence value ranging from 0 to 1, where 1 represents full belief, and 0 represents no belief. The confidence value of the ball decreases based on circumstance, and [12] provides a detailed description of this process.

When the hypothesis of the ball is derived from the vision module, i.e., the ball was visible a number of frames ago, the confidence of the ball will decay over time. The decay represents the uncertainty of the ball's position over time, since RoboCup is a dynamic environment and the ball will travel around the field. In addition, as the robot walks in the environment, the confidence of the ball decays further, as the odometry updates of the robot are noisy. Thus, after the robot walks a certain distance, the ball's confidence will drop to a level such that the hypothesis is not longer viable. Lastly, if the robot's gaze is looking in the direction but vision does not detect a ball, the confidence of the ball drops rapidly, which ensures that invalid hypotheses of the ball are removed quickly.

Ball hypotheses can also be generated from teammate messages. When a teammates detects a ball through vision, the ball's global coordinates (converted from relative coordinates using the pose from localization) is broadcast to the entire team. These messages are then used to generate ball hypotheses in the robot's world model. The confidence of the teammate ball hypotheses also decays over time, again because of the dynamic nature of the RoboCup environment. Similarly, negative vision detection, i.e., the ball is not detected when the robot is looking at its estimated position, causes the confidence to decay rapidly. However, the confidence of a teammate ball hypothesis does not decay as the robot walks in the field. Odometry does not affect teammate ball hypotheses because the location of the teammate ball is given in global coordinates, which does not change as the robot walks.

The confidence of the ball hypotheses are thresholded into 3 levels: valid and not suspicious, valid but suspicious, and invalid. These levels correspond to discrete actions that the behaviors can take on the ball hypothesis. When the hypothesis is valid and not suspicious, the confidence of the ball's position is high, and behaviors can take actions as if the ball was visible, e.g., the robot can look at the goal while circling around the ball. If the ball is valid but suspicious, this means that the world model has a hypothesis of the ball's location, but is uncertain. Thus, behaviors should look at the ball's estimated location, which will either cause the ball's confidence to increase if the ball is visible, or decrease rapidly if the ball is not where it should be. In either case, the ball hypothesis will leave the suspicious state. Thus, the suspicious state is an active-feedback request of the world model.

B. Updating the Goal

The location of the goal in global coordinates is known to the robot from the map of the field. Given that the localization module provides a pose of the robot, i.e., its global coordinates, it is possible to infer the relative position of the goal. However, pose estimates are noisy and can cause the robot to take a shot in the wrong direction if localization is in error. For example, if the pose estimate from localization has an error of 10cm, the robot may think that it is standing between both goal posts, when in fact it is slightly to the left of the goal. A straight shot taken in this position would cause the ball to go out, instead of scoring a goal. Thus, the world model module models the goal using vision feedback.

Before taking a shot on the goal, the robot scans its surrounding to locate the goal visually. However, goal posts can be ambiguous in the camera image (see Fig. 2). The world model keeps track of goal posts as they are detected by vision, and merges the information into a single goal after the scan is completed. The world model keeps track of the 2 goal post detections with highest confidence. The left and right goal posts are then identified based on their relative angles to the robot, e.g., the left goal post is the one on the left.

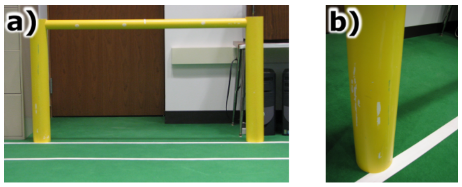

Fig. 2. a) A yellow goal. b) An ambiguous yellow goal post.

Using the relative angles and distances to the left and right poles, the world model module then estimates the location of the goal. In the field, the goal posts are 140mm apart. However, due to errors in vision detection, the distance estimates to the goal posts are usually poor (a detection of twice the actual distance is possible) since it uses heuristics such as the number of pixels, which is susceptible to errors in color segmentation. However, the angle estimates are accurate, since these are derived from the pixel location in the camera image. Thus, the world model mainly uses the angle estimates to derive the goal position. While the distance estimates given by vision are potentially poor, the ratio of distances provide a fair estimate of the relative difference in distances of the poles to the robot. For example, if one pole is detected at 1.5 times its actual distance, then the distance estimate to the other pole is likely to be also around 1.5 times the actual distance. From the accurate angle estimates, and the ratio of distances to the poles, the world model is able to reconstruct the true position and orientation of the goal relative to the robot.

The goal model is used for kicking accurately at the goal, when the robot is close to the goal. When the robot is far from the goal, the pose provided by the localization module is accurate enough for the robot, especially since the long distance kicks of the robots have a large variance in angles.

V. Conclusions

The NAO humanoid robots are used in the Standard Platform League of RoboCup, which is a highly dynamic and adversarial environment. The goal of the robot team is to score goals, and to do so effectively, the robots must be capable of perceiving the world accurately. The features detected by the vision module are used as the primary inputs to the other cognitive modules of the NAOs, namely localization, which determines the global position and orientation of the robot on the field, and world modeling, which models objects in the world that are not currently visible, such as the ball.

We formalized the robot cognition domain and problem, as well as the cognition modules. We then detailed the algorithms used in the localization module, such as how the particles are updated. We also described the sensor models we used in updating the weights of the particles when vision features, such as lines and corners, as detected, and how resampling and sensor resetting is performed. We then extract a single pose of the robot, by fusing information about the particles in our localization module.

We described how the world model module uses the output of vision to update models of the ball and goal. When a ball is visible to the robot, the world model uses the visible ball as its output; when the ball is not visible, the model of the ball is used to generate a hypothesis on the ball's location. The confidence of the hypothesis decays based on time and negative vision. If the hypothesis originated from vision and not from a teammate, then the confidence of the hypothesis also decays from the robot's motion.

The localization module provides a global pose of the robot, and it is possible to derive the relative position of the goal. However, to take accurate shots at the goal, the world model creates a model of the goal's position based on a visual scan performed by the behaviors of the robot. The angles to the goal posts, and the relative distances, are used to construct an accurate model of the goal's position and orientation relative to the robot, so that an accurate shot to goal can be performed.

References

[1] RoboCup, "RoboCup International Robot Soccer Competition," 2010, http://www.robocup.org.

[2] RoboCup SPL, "The RoboCup Standard Platform League," 2010, http://www.tzi.de/spl.

[3] Aldebaran, "Aldebaran Robotics - Nao Humanoid Robot," 2010, http://www.aldebaran-robotics.com/pageProjetsNao.php.

[4] J. Bruce, T. Balch, and M. Veloso, "Fast and inexpensive color image segmentation for interactive robots," in *Proceedings of IROS*, 2000.

[5] S. Thrun, D. Fox, W. Burgard, and F. Dellaert, "Robust monte carlo localization for mobile robots," *Artificial Intelligence*, vol. 128, no. 1-2, pp. 99 – 141, 2001. [Online]. Available: http://www.sciencedirect.com/science/article/B6TYF-42YM3PD-3/2/d1e9fd756696283bc6a8d9dc14d021ab

[6] J.-S. Gutmann, "Markov-kalman localization for mobile robots," vol. 2, 2002, pp. 601 – 604 vol.2.

[7] T. Laue, T. J. de Hass, A. Burchardt, C. Graf, T. Röfer, A. Härtl, and A. Rieskamp, "Efficient and reliable sensor models for humanoid soccer robot self-localization," in *Proceedings of 4th Workshop on Humanoid Soccer Robots*, 2009.

[8] S. Lenser and M. Veloso, "Sensor resetting localization for poorly modelled mobile robots," in *Proceedings of ICRA, the International Conference on Robotics and Automation*, April 2000.

[9] Y. Gu and M. Veloso, "Effective Multi-Model Motion Tracking using Action Models," *Int. Journal of Robotics Research*, vol. 28, pp. 3–19, 2009.

[10] P. Rybski and M. Veloso, "Prioritized Multi-hypothesis Tracking by a Robot with Limited Sensing," *EURASIP Journal on Advances in Signal Processing*, 2009.

[11] N. Mitsunaga, T. Izumi, and M. Asada, "Cooperative Behavior based on a Subjective Map with Shared Information in a Dynamic Environment," in *Proc. of IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 2003, pp. 291–296.

[12] B. Coltin, S. Liemhetcharat, Ç. Meriçli, J. Tay, and M. Veloso, "Multi-humanoid world modeling in standard platform robot soccer," in *Proceedings of 10th IEEE-RAS Int. Conf. on Humanoid Robots*, 2010.

[13] T. Hester and P. Stone, "Negative information and line observations for monte carlo localization," in *Proceedings of ICRA, the International Conference on Robotics and Automation*, May 2008.

[14] T. Röfer, T. Laue, J. Müller, A. Burchardt, E. Damrose, A. Fabisch, F. Feldpausch, K. Gillmann, C. Graf, T. J. de Haas, A. Härtl, D. Honsel, P. Kastner, T. Kastner, B. Markowsky, M. Mester, J. Peter, O. J. L. Riemann, M. Ring, W. Sauerland, A. Schreck, I. Sieverdingbeck, F. Wenk, and J.-H. Worch, "B-human team report and code release 2010," 2010.