

CMurfs: Carnegie Mellon United Robots for Soccer

Manuela Veloso, Somchaya Liemhetcharat,
Brian Coltin, Çetin Meriçli, and Junyun Tay

School of Computer Science, Carnegie Mellon University, USA

1 Introduction

In RoboCup'2009, Carnegie Mellon University participated as CMWrEagle, a joint team between the University of Science and Technology of China, led by Professor Xiaoping Chen, and Carnegie Mellon University, led by Professor Manuela Veloso.

For RoboCup'2010, Carnegie Mellon University will be participating as CMurfs - Carnegie Mellon United Robots for Soccer.

2 Architecture

Our architecture diagram for RoboCup'2009 is shown in Figure 1. The key idea of our architecture design is to separate the logic of the robots from the calls required by the Nao's SDK - NaoQi. This isolates NaoQi-related code, which allows us to upgrade between versions of NaoQi more easily. Furthermore, the non NaoQi-dependent components can be individually tested from a multitude of sources, so we can plug-and-play between modules that run through NaoQi, custom-made simulators, as well as running from logs that we create.

Another aspect of our architecture is the Agent component. Previously, we had a large vector of features that each of the components (such as Vision and World Model) would fill in. In addition, components would call functions on each other, making the components tightly-coupled. By creating the Agent component which acts as the proxy, each component is now isolated from the rest, and can be easily understood as a black-box that provides certain output given certain input. This removes the coupling effect between the components, so that we can test each component individually, as well as plug-and-play different versions of components if required.

Our experience in RoboCup'2009 showed that the decoupling of NaoQi from other components was a wise decision. We updated our version of NaoQi thrice during the development of our code for RoboCup'2009, and minimal changes were required in our codebase. In addition, we developed many tools, such as a Remote Control interface, a Graphical Debug Display, and a Log Reader, that were NaoQi-independent, and were able to take advantage of the new architecture. For RoboCup'2010, we will continue to improve our architecture design, and allow more fluid feedback between components.

3 Vision

The vision module maps from YUV camera images to the locations, relative to the robot, of any objects detected in the camera image. Vision also provides a heuristic

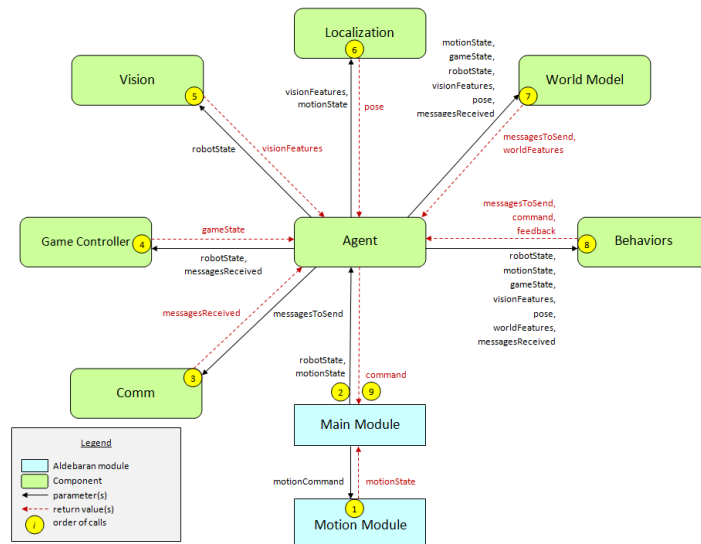


Fig. 1. Diagram of our architecture

confidence score for each of the detected objects that indicates how likely it is that the object is truly present in the image. We divide the vision pipeline into two stages. The first state, low level vision, uses the CMVision library [2, 1], to perform color segmentation on the image. CMVision uses a lookup table to map from YUV pixel intensities to symbolic colors, such as red, blue, or orange, as shown in Fig. 2. The library then builds up lists of the colored regions in the resulting image. These lists of regions, which specify the bounding box and centroid of each region, are what we used in the second stage of vision, high level vision, for object detection.

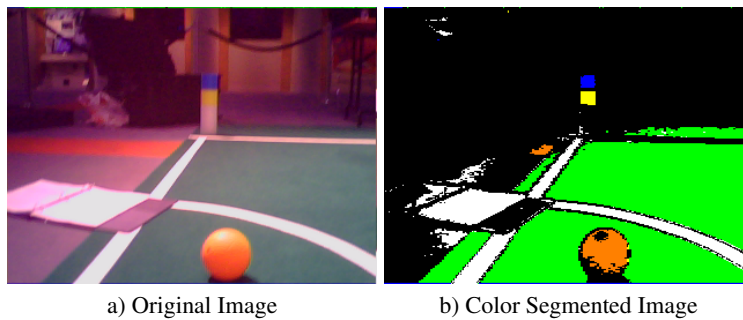


Fig. 2. An image from the robot before (a) and after color segmentation (b).

We implemented object detectors to extract the position and distance to several different classes of objects. The ball detector computes the distance and angle to the ball using several different methods including finding the area and center of the orange region, making use of the constraint that the center of the ball is a known height off the ground, and circle fitting. Additionally, we have an object detector for computing the distance and orientation to goal posts. If the crossbar on the top of the goal is visible and appears connected to the goal post then vision can provide the additional information of the side of the goal that this post belongs to. We also include a line detector, that samples the white pixels in the image and returns lines, corners and intersections in the image, which we can use for localization. Lastly, we have a robot detector that detects robots in the distance, allowing us to plan paths around them.

In addition to reporting the position and orientation of objects in the camera image, the object detectors also compute heuristic confidence values between 0 and 1, which indicate how confident we are that the object is truly present in the image. The behaviors respond differently to the reported information depending on the confidence values assigned to it.

4 World Model

Our team makes use of a World Model for updating and maintaining the locations of objects of importance. This module takes vision objects from high-level vision and merges them with previous information about objects. In particular, the world model contains ball location and the positions of each of the goal posts. The world model stores everything in local coordinates. Odometry information obtained from the Motion Module is used to update the locations of all of the objects each frame. The world model also decides whether objects are valid, suspicious, or invalid. Objects are valid initially, and after a certain amount of time since they were last seen, become suspicious and then invalid. The goal posts take longer than the ball to become invalid since they do not move, and so the only cause of error is the robot's odometry. When an object is suspicious, the behaviors should look towards to objects to either confirm or deny its presence. If an object is invalid, this means the world model does not know where it is.

Goal posts are somewhat trickier to model since there are two interdependent posts on each goal. Vision will report a left or right goal post if it can see the crossbar, but if it cannot it returns an unknown post. If we are given a left or right goal post we will overwrite the old value in the world model. If this new post is not the correct distance from the other post of the same color in the world model, the other post is marked as invalid. We also attempt to use the previous positions of left and right posts to match an unknown post to the correct side of the goal.

5 Communication

Given that there are 3 robots in each Nao team in RoboCup'2009, it was essential that the robots communicate. Thus, we implemented role-switching, which is dependent on communication between the robots.

The 2 field-players (i.e., not the goal-keeper) are attackers who attempt to score goals. However, when one robot is close to the ball, the other robot heads towards a “supporting” position. In order to achieve this, we implemented simple message passing through UDP, where the robots send a numeric value describing how confident they are about reaching the ball. The most confident robot then becomes the attacker, while the other becomes a supporter and heads to the supporting position on the field.

6 Behaviors

Our behaviors are built on a Finite State Machine (FSM) framework. Our Behaviors component takes input primarily from the World Model which holds the data we need to make decisions and generates Commands that are executed on the robot. The primary behavior we developed was that of the Attacker, Supporter and Goalie.

The attacker FSM cycles between 4 states: *lost*, *approach*, *orbit*, and *kick*. Each of these states calls another FSM.

The *lost* state is used when the ball’s position is invalid in the world model. In this state, the robot spins in a complete circle, searching for the ball. If the ball is not found, the robot heads half a meter backfield in the center of the field, as determined by localization. It then spins again and repeats the process.

The other states involve approaching and kicking the ball. The attacker is in the *approach* state when the ball’s position is valid but the robot is still far away. In this state, the attacker heads directly towards the ball. It then enters the *orbit* state, where the robot orbits around the ball until it is lined up for a kick with a goal. The attacker lines up for either a forward or side kick depending on which requires a shorter orbit. If the attacker does not know where the goal is, it aligns based on localization and searches for the goal as it orbits. The attacker will not kick the ball if the goal’s position is invalid. After it is aligned, the attacker transitions to the *kick* state. It lines up its foot precisely with the ball and then kicks towards the goal.

The supporter uses the same skills as the attacker, except that it does not approach the ball or kick it. Instead, the supporter heads to a position close to the ball, but out of the way of the attacker. This allows the robots to switch roles if the current attacker loses the ball, or if the ball rolls to a position closer to the supporter.

Conceptually, the goalie behavior has 4 FSM states: *lineUp*, *guard*, *search*, and *clear*. The goalie tries to line up between the ball and the center of the goal, when it is in the *lineUp* state. If it is in position, the goalie goes into the *guard* state and stretches its leg to maximize coverage of the goal. If the ball is lost, the goalie searches for it (*search* state). At any point in time, if the ball is found to be within the penalty box, the goalie enters the *clear* state, approaches the ball and kicks it away. Fig. 3 shows the FSM states and transitions of the goalie.

In order to move around the penalty box accurately, the goalie has to be well-localized. The Localization component was designed to work across the entire field, and was not precise enough for the goalie’s needs. As such, a separate localization function was written specifically for the goalie. It used visual features such as the lines and corners in the penalty box. The goalie’s localization function allowed the goalie to

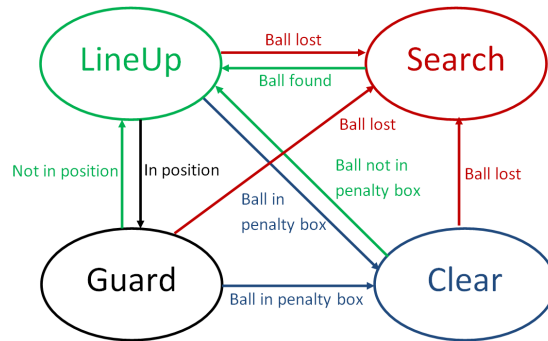


Fig. 3. FSM of the Goalie behavior

remain in the goal box throughout all games in RoboCup'2009, and line up in the right positions.

7 Motion

For RoboCup'2009, the motion engine developed by Wright Eagle was used in the competitions. The Wright Eagle motion engine models the robot's upper body as a linear inverted pendulum (Fig 4, 5).

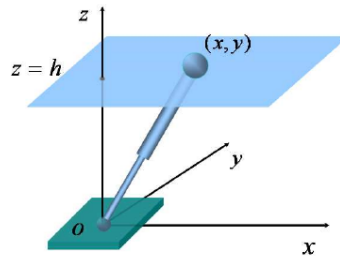


Fig. 4. 3D Linear Inverted Pendulum model

The walk planning is considered as selecting a sequence of ZMPs that will yield to stable walking by satisfying dynamical and mechanical constraints. An online sampling method is employed to tackle this problem and the walk definition is simplified by considering it a sequence of single-support phases instead of a more conventional single-support / double-support alternations [3,4].

Another motion engine is currently under development and we are planning to replace Wright Eagle motion engine with our in-house developed engine for RoboCup'2010.

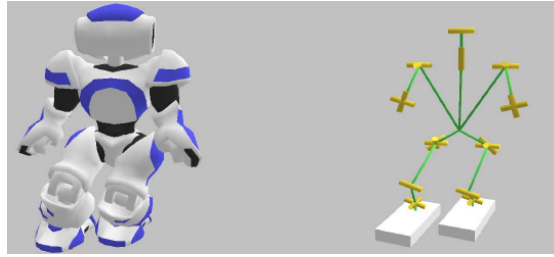


Fig. 5. Simulated Nao robot and its joints

References

1. J. Bruce, T. Balch, and M. Veloso. CMVision, www.cs.cmu.edu/~jbruce/cmvision/.
2. J. Bruce, T. Balch, and M. Veloso. Fast and inexpensive color image segmentation for interactive robots. In Proceedings of IROS-2000, 2000.
3. Jinsu Liu and Manuela Veloso. Online ZMP Sampling Search for Biped Walking Planning. In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'08), Nice, France, September 2008.
4. Jinsu Liu, XiaoPing Chen, and Manuela Veloso. Simplified Walking: A New Way to Generate Flexible Biped Patterns. In Proceedings of the 12th International Conference on Climbing and Walking Robots and the Support Technologies for Mobile Machines (CLAWAR), Istanbul, Turkey, September 2009.