

# CMurfs: Carnegie Mellon United Robots for Soccer

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

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. We first describe our CMurfs architecture, presenting the on-board computation on the robot in terms of a Cognitive Agent and the processes to connect to NaoQi. We then detail the components of the Cognitive Agent, followed by a brief description of Vision, Localization and World Model, and Behaviors. We finally briefly present the motions used by the behaviors.

## 2 Architecture

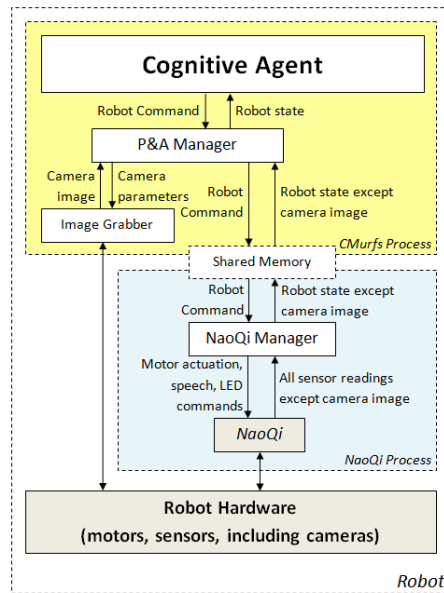
Our architecture diagram for CMurfs is shown in Fig. 1. Two processes run on the robot: the CMurfs process and the NaoQi process, boxed in yellow and blue respectively in Fig. 1. NaoQi is the Nao's SDK, which interfaces the robot's hardware and sensors.

A NaoQi Manager runs in the NaoQi process, allowing direct calls to NaoQi, e.g., to access the foot pressure sensors of the robot. The NaoQi Manager retrieves all sensors readings from the robot, except camera images, and sends motor actuation commands, speech commands, and LED commands to NaoQi in order to actuate the robot.

Within the CMurfs process, we have:

- an Image Grabber that interfaces the camera drivers, in order to set the cameras' parameters and retrieve images.
- a Perception and Actuation Manager (P&A Manager) that merges the partial Robot State (from the NaoQi Manager) and camera image (from the Image Grabber) into a complete Robot State, and sends the Robot Command to the NaoQi Manager.
- the Cognitive Agent, which processes the Robot State, generates Robot Commands, and handles network communications.

The NaoQi process and CMurfs process communicate via a shared memory system, where they read and write the Robot Command and Robot State (excluding camera images). Running 2 separate processes has multiple benefits: firstly, most of the system is independent from the NaoQi interface. When new versions of NaoQi are released, only the functions calls in the NaoQi Manager have to be updated, while the rest of the architecture remains unchanged. Secondly, changes made to the Cognitive Agent



**Fig. 1.** CMurfs architecture, which comprises the CMurfs process and the NaoQi process.

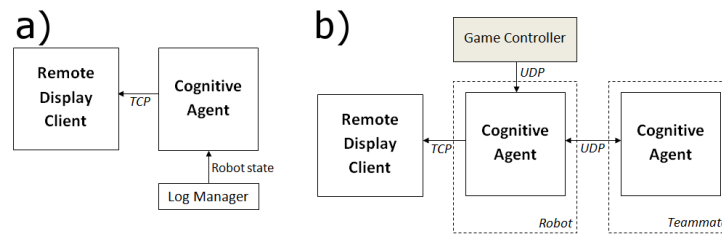
during development does not require restarting NaoQi, which takes approximately 1-2 minutes to restart. Furthermore, if errors occur in the CMurfs Process and it terminates, the NaoQi process remains unaffected and the robot will not fall over. Previously, our system resided entirely in the NaoQi process, and a termination caused the robot to remove stiffness from all its joints and fall down.

In addition, the Cognitive Agent is independent from the NaoQi SDK, and can be run without the robot. Fig. 2a shows the Cognitive Agent interfacing with a Log Manager, which reads logs files and sends Robot States to the Cognitive Agent for processing. The Cognitive Agent transmits data to a Remote Display Client via TCP, where information about the Cognitive Agent, such as the segmented image, detected visual objects, and global pose of the robot, can be displayed for debugging purposes.

When running on a robot, the Cognitive Agent similarly transmits display information to the Remote Display Client, and also communicates via UDP to its teammates (who also run their own Cognitive Agents). The Game Controller provides information about the state of the game, e.g., the time left in the half, to the Cognitive Agent via UDP as well. Fig. 2b shows the network communication of the Cognitive Agent. During an actual game, communication is only permitted between teammates, so the communication link to the Remote Display Client is not available then.

### 3 Cognitive Agent

The Cognitive Agent processes the Robot State, which contains the current camera image, transformation matrices and sensor values (e.g., ultrasound, body angle) and



**Fig. 2.** a) Running the Cognitive Agent from logs. b) Communication between the robot and teammates, the Game Controller, and the Remote Display Client.

generates a Robot Command, which comprises a Motion Command (e.g., walk, perform a keyframe), a Speech Command (to be used by the text-to-speech engine), and an LED command (to display colors on the robot).

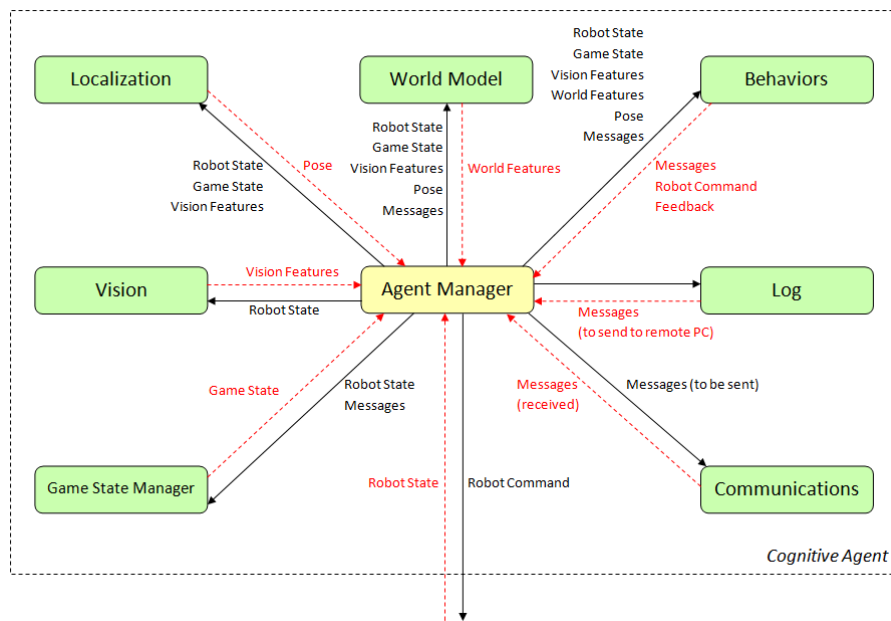
Fig. 3 shows the different components of the Cognitive Agent, namely:

- Agent Manager — manages the information flow between each of the other components.
- Game State Manager — updates the game state based on messages received from the Game Controller, as well as button presses.
- Vision — processes the camera image and produces vision features, e.g., balls, goal posts, lines and corners.
- Localization — estimates the global pose (position and orientation) of the robot using odometry and vision features.
- World Model — maintains hypotheses of the ball, using vision, and information shared by the teammates
- Behaviors — runs a Finite State Machine and generates robot commands and feedback, and messages to be sent to teammates
- Log — creates serialized information that can be sent across the network to a Remote Display Client
- Communications — sends and receives network packets

The input and output interface for each component is well-defined, which allows us to readily switch between versions of components. For example, we can toggle between running 2009’s version of Vision versus 2010’s by changing a line in a configuration file. The ability to switch between components enables easy comparisons, and ensure that no new bugs are introduced.

In addition to the input/output interface, we employ the use of a feedback mechanism, so that components can transfer information to each other readily. For example, the Behaviors component feeds back when a scan for goals is being performed, and the World Model component only models the goal during this period. The feedback mechanism ensures that the interface of components remains well-specified, so that different versions of components can continue to be swapped, without components making tightly-coupled calls to one another.

We now go into details on some of our components, namely Vision, Localization and World Model, and Behaviors.



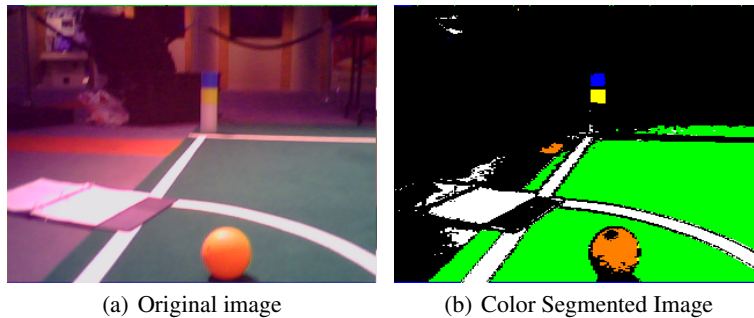
**Fig. 3.** Different components of the CMurfs Cognitive Agent. The Cognitive Agent receives a Robot State as input and generates a Robot Command as output.

## 4 Vision

The Vision component 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 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 [1, 2], 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. 4. 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.

We analyzed the run-time cost of vision in CMurfs, and noticed that the Vision component was taking up the majority of the processing time. Thus, we are improving the run-time efficiency of the thresholding and region-formation, as well as some of the object detectors. The ball and goal post detectors from 2009 worked well, and we continue to use them in CMurfs. However, the robot, line and corner detectors are being improved, so as to improve their efficiency.

The Vision component processes the camera images, and reports Vision Features, such as balls and goal posts, in a relative coordinate frame, with a heuristic confidence



**Fig. 4.** An image from the robot before (a) and after color segmentation (b).

value between 0 and 1, representing how confidence we are that the object is truly present in the image.

## 5 Localization and World Model

The Vision Features detected by the Vision component are used by the Localization component to generate a pose of the robot — its position and orientation in a global coordinate frame. We use Monte-Carlo localization with sensor resetting [3]. Each particle is a guess of the robot’s pose, and the position and weight of each particle is updated based on the motion of the robot (as reported by odometry), as well as Vision Features detected by Vision. Also, we update the particles of Localization based on feedback, such as a change in the game state (e.g., start of the game, return from penalty), and updates from the state of the robot (e.g., falling over, being picked up).

The pose generated by Localization is then used by the Behaviors component to determine the best action to perform. For example, the supporting robot, i.e., the robot not heading to the ball, can proceed to a good defensive position behind the ball.

While Localization estimates the global pose of the robot, the World Model component models hypotheses of the ball and the goal (when an active scan is being performed). We improved the World Model from 2009, where we only kept a single hypothesis of the ball and goal posts. In addition, in 2009, each goal post was modelled independently, and the width of the goal was allowed to grow and shrink, which is physically impossible. In CMurfs, we maintain multiple hypotheses of the ball, and create hypotheses from vision, actions and teammates’ information [4, 5]. For example, if the robot kicks a ball, it adds a hypothesis to where the ball should end up after the kick. The World Model then picks the most likely hypothesis to report to the Behaviors component. If a ball is not detected by Vision when the robot is looking at it, the World Model quickly degrades its confidence, and the next most likely hypothesis is chosen. This allows the robot to readily switch between hypotheses as it searches for the ball.

Goal posts as detected by Vision typically have poor distance estimates. As such, we incorporate a feedback mechanism, where the World Model only models the goal when Behaviors are actively searching for it with a scan. Then, the World Model maintains

hypotheses of the goal posts, and after the scan is completed (as reported by Behaviors), the information is fused together to generate a single hypothesis of the goal's position and orientation. This method enforces the physical constraint on the goal's width, and uses information about the relative angle estimates of the detected posts to generate the hypothesis. Our initial experiments have shown that the World Model creates an accurate estimate of the goal's position and orientation with this active scanning method. Using this estimate of the goal, Behaviors can then accurately shoot towards the goal when the robot is close to the goal, instead of relying on Localization, which tends to be poorer in those regions.

## 6 Communication

We used communication exclusively for role switching in 2009, where the robot closer to the ball would be the attacker, and the other field player would be the supporter. However, we were unable to share global ball information, as the Localization component provided poor pose estimates.

In CMurfs, we continue to use communication for role-switching, but will also incorporate global ball information, so that robots will be able to search for balls more effectively. The robots broadcast a UDP packet with a value on how confident they are about reaching the ball, as well as the ball's global position. The confidence value is used to determine the roles, while the global ball position is used to generate hypotheses in the World Model.

In addition, the Communication component listens to UDP packets from the Game Controller, and parses these packets, so that the Game State Manager can process them.

Besides the UDP communication framework, each robot has a TCP server, which allows the Remote Graphical client to connect to it, providing a view of the robot's internal state. During the actual competition, communication with other computers is disallowed, and the TCP server is deactivated. However, during the development and testing phase in the lab, the TCP server enables good debugging capabilities.

## 7 Behaviors

We use a Finite State Machine (FSM) framework for the Behaviors component, and we build upon the behaviors created in 2009. The Behaviors component takes input primarily from the World Model and Localization, and generates Robot Commands. We have 3 primary behaviors: Attacker, Supporter and Goalie.

Fig. 5(a) shows the 4 states of the Attacker: *search*, *approach*, *orbit*, *kick*. In the *search* state, the robot scans its surrounding areas and searches for the ball. If the ball cannot be found, the robot spins in a complete circle looking for the ball.

In the *approach* state, the Attacker heads directly for the ball, using an omnidirectional walk. The robot also actively avoid obstacles as detected by the ultrasound sensors. Upon reaching the ball, the robot switches to the *kick* state.

The *kick* state chooses an appropriate kick, and attempts to line the foot with the ball in order to perform the kick. If no kicks are available, e.g., if the robot wants to

kick the ball behind itself, then the FSM switches to the *orbit* state. The robot orbits around the ball at a fixed radius, until a kick becomes available, when it switches back to the *kick* state.

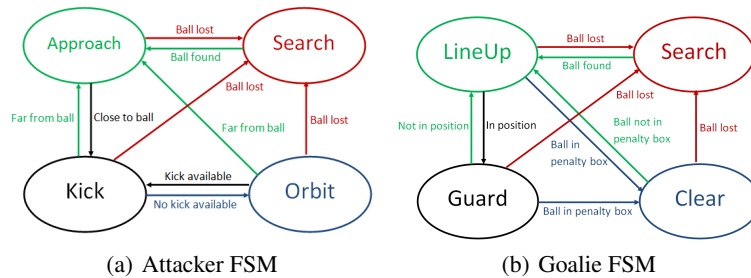


Fig. 5. FSMs of the Attacker and Goalie behaviors

We continue to use the Supporter and Goalie behaviors that we created in 2009. The Supporter behavior chooses a good position between the ball and the team’s goal, and heads there. In this way, direct shots taken by the opposing team can be blocked by the Supporter. In addition, the Supporter stays a fixed distance behind the ball, so that it can quickly switch roles to become an Attacker, if the current Attacker loses the ball or becomes penalized.

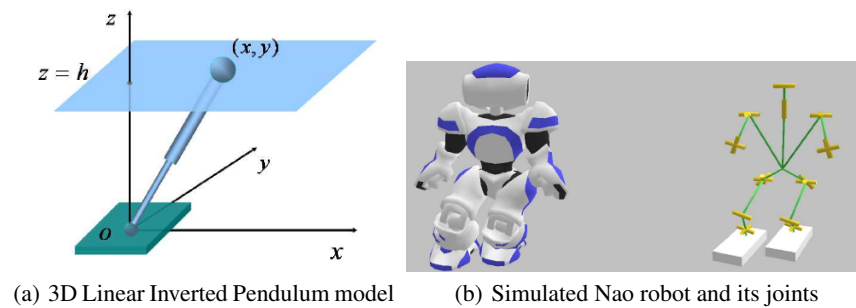
Conceptually, the Goalie behavior has 4 FSM states: *lineUp*, *guard*, *search*, and *clear*. The Goalie tries to line up between the ball and the center of the goal, when it is in the *lineUp* state. If it is in position, the Goalie goes into the *guard* state and stretches its leg to maximize coverage 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. 5(b) 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 is not precise enough for the Goalie’s needs. As such, a separate localization function was written specifically for the Goalie. It uses Vision Features such as the lines and corners in the penalty box. The Goalie’s localization function allowed the goalie to remain in the goal box throughout all games in RoboCup’2009, and line up in the right positions.

## 8 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 (see Fig. 6).

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 sam-



**Fig. 6.** Analytical model of the motion.

pling 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 [6, 7].

In CMurfs, we use the walk engine by Aldebaran, which provides a fast and stable omni-directional walk. The omni-directional walk simplifies the path-planning process for the robots, as many more actions are available (compared to turn, forwards and sidle in 2009).

We have also created new kicks for the Naos for CMurfs, which are fast and stable to execute. The kicks are pre-generated keyframe actions, which the robot performs open-loop. However, when generating the keyframes in the lab, we calculate the center of mass of the robot, to ensure that the robot is in a stable position.

## References

1. J. Bruce, T. Balch, and M. Veloso. CMVision, [www.cs.cmu.edu/~jbruce/cmvision/](http://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. S. Lenser, and M. Veloso. Sensor resetting localization for poorly modelled mobile robots. In Proceeding of ICRA-2000, 2000.
4. P. Rybski, and M. Veloso. Prioritized Multi-Hypothesis Tracking by a Robot with Limited Sensing. In EURASIP Journal on Advances in Signal Processing, 2009.
5. B. Coltin, S. Liemhetcharat, C. Meriçli, and M. Veloso. Challenges of Multi-Robot World Modelling in Dynamic and Adversarial Domains. In Practical Cognitive agents and Robots (PCAR), AAMAS 2010 Workshop, 2010.
6. 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.
7. 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.