

# CMWrEagle: Joint-Team of CMU and UTSC

Manuela Veloso<sup>†</sup>, Xiaoping Chen<sup>‡</sup>, Somchaya Liemhetcharat<sup>†</sup>, Jinsu Liu<sup>‡</sup>, Michael Phillips<sup>†</sup>, Brian Coltin<sup>†</sup>, Feng Xue<sup>‡</sup>, and Junyun Tay<sup>†</sup>

<sup>†</sup>Computer Science Department, Carnegie Mellon University, USA

<sup>‡</sup>Department of Computer Science and Technology, University of Science and Technology of China, China

**Abstract.** After performing very well in RoboCup’2008 SPL, Carnegie Mellon University has joined efforts with the University of Science and Technology of China to form CMWrEagle. Our goal is to leverage on both team’s experience in RoboCup, from last year’s SPL as well as the 4-legged league in previous years. We regard our joint-team, CMWrEagle, as an integration between researchers at Carnegie Mellon University as well as the University of Science and Technology of China. In this paper, we describe our architecture, followed by details of its components.

## 1 Introduction

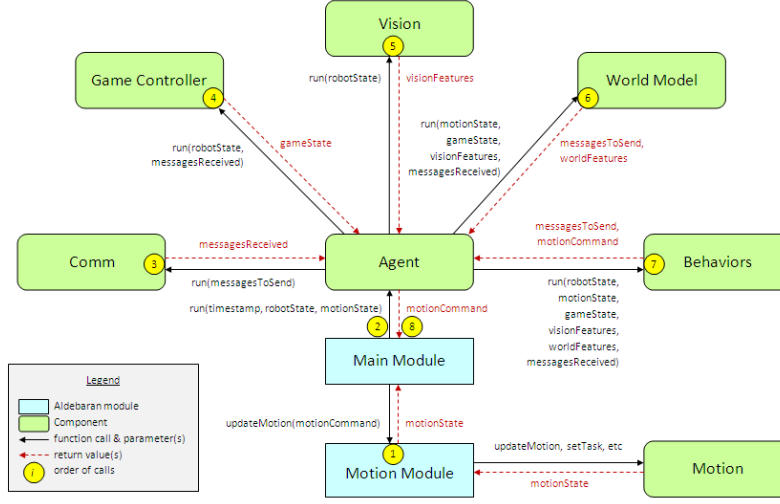
In RoboCup’2008 SPL, Carnegie Mellon University participated as GTCMUnited’08, a joint team between the Georgia Institute of Technology, led by Professor Tucker Balch, and Carnegie Mellon University, led by Professor Manuela Veloso. The University of Science and Technology of China participated as WrightEagle, a team lead by Professor Xiaoping Chen, that has been participating in RoboCup since 2000.

For RoboCup’2009, the Georgia Institute of Technology will not be able to participate and Carnegie Mellon has joined efforts with UTSC, the University of Technology and Science of China to form CMWrEagle.

For RoboCup’09, we are investigating vision, utility-based behaviors, and team coordination. Through CMWrEagle, the joint team with UTSC, we expect to continue our research on ”pick up teams” in which we try to connect robots developed at different institutions through ”pickup” protocols. The goal is to enable in the future more researchers to collaborate on teams from multiple institutions.

## 2 Architecture

Our architecture diagram 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.



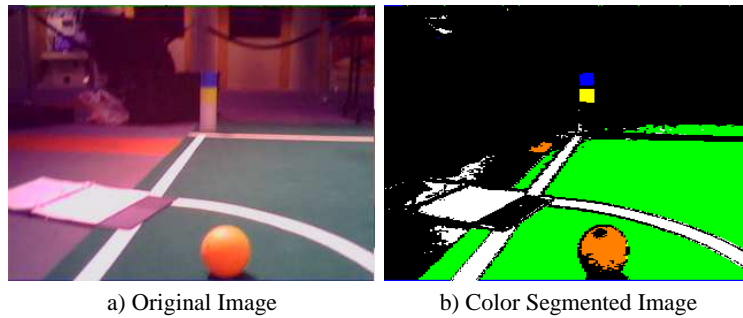
**Fig. 1.** Diagram of our architecture

Another aspect of our architecture is the Agent component. Previously, we had a large Features vector 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.

### 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 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 figure 1. 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 implemented object detectors to extract the position and distance to several different classes of objects. The most important object detector computes the distance and angle to the ball. We use several methods for determining the distance and angle of the ball including area and center of the orange region, making use of the constraint that



**Fig. 2.** An image from the RoboCup US Open before (a) and after color segmentation (b).

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 of 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 tactics in the behavior system then vary how they respond 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 visual goal information. The world model stores both of these bits of information in local coordinates due to our lack of a localization component. In order for this to work the world model uses odometry to update the locations of all of the objects it is maintaining. We also attach validity flags to our objects and declare them to be "suspicious" when a certain amount of time or a certain amount of movement has been accomplished since the last visual update. Once an object is suspicious, the behaviors should look towards to objects to either confirm or deny its existence.

Goal posts are somewhat trickier since we need both to have a well defined goal to kick toward. Vision will report a left or right goal post if it can see the crossbar but if it cannot then it returns an unknown post. If we are given a left or right goal post we will overwrite the previous and we can invalidate the other post if the distance constraint between the two posts is reasonably incorrect. We can also use the previous estimates of left and right posts to match an unknown post to its proper side of the goal.

## 5 Communication

Given that there are 3 robots in each Nao team in RoboCup'2009, it is essential that the robots communicate. At the very least, the robots should not get in each other's way when approaching the ball. 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 should back away, or head 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 does not approach the ball directly.

## 6 Behaviors

Our behaviors are built on a Finite State Machine (FSM) framework. Our behaviors module takes input primarily from the world model which holds the data we need to make decisions and tells the motion module to execute commands based on the state we are in. The primary behavior we developed was that of the Attacker. The attacker FSM cycled between 5 states: lost, alignFar, alignClose, approach, and kickAtGoal. Each of these are actually another FSM.

The lost FSM is used when the world model deems the ball invalid, and causes the robot to perform a search. The alignFar FSM is used when the attacker is far from the ball and drives the robot to approach a point near the ball from which the robot could kick the ball toward the goal. The attacker uses the approach FSM after the robot is too close to the ball to see it with the upper camera. It uses the lower camera to observe the ball and get it close to one of its feet. When the robot is close enough to the ball to kick it, it may invoke the alignClose FSM to execute an "orbit" around the ball to finish aligning with the goal. After the attacker is close enough to the ball to kick and is aligned with the goal it runs the kick FSM which runs our kick and then checks to make sure the ball actually moved.

For a supporter, it uses the same skills as the attacker, except that it does not approach the ball or kick it. Instead, once the supporter gets within a certain distance of the ball, it stops approaching and watches the ball intently. 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.

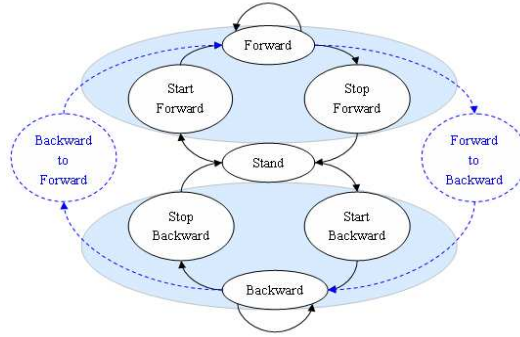
## 7 Motion

To make the robot gain more flexibility, we use a technique to connect the *periodic biped motions* together. The periodic biped motion is defined as the biped walking motions which include three types of steps: starting steps, walking steps and stopping steps. Each walking starts from starting steps and ends up with ending steps. A continuous walking can be generated by repeating the walking steps.



**Fig. 3.** Our attacker scoring a goal.

An example of our improved biped motion state machine is shown in Figure 4. The example demonstrates the state transitions between a forward walking motion and a backward walking motion. The part of state machine drawn by solid lines is the traditional method. The limitation of this method is that the robot has to stop when it wants to change the current walking task to the other one. In our approach, we directly calculate transitional steps which is shown as the part of Figure 4 drawn by dash lines.

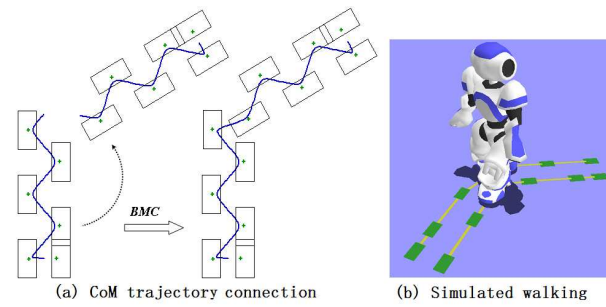


**Fig. 4.** Example of the improved biped motion state machine

We developed the *biped motion connection* technique to connect two biped motions smoothly by using two walking steps. Figure 5 is a demonstration of the connection. Figure 5(a) shows a smooth connection of two CoM trajectories. The generated trajectory can be used in the lower level to obtain the whole motion with inverse kinematics. The simulated robot achieved the connecting steps successfully as shown in Figure 5(b).

## References

1. J. Bruce, T. Balch, and M. Veloso. CMVision, [www.cs.cmu.edu/~jbruce/cmvision/](http://www.cs.cmu.edu/~jbruce/cmvision/).



**Fig. 5.** Demonstration of biped motion connection

2. J. Bruce, T. Balch, and M. Veloso. Fast and inexpensive color image segmentation for interactive robots. In Proceedings of IROS-2000, 2000.
3. Ashley W. Stroupe, Kevin Sikorski, and Tucker Balch. Constraint-based landmark localization. In RoboCup-2002: The Sixth RoboCup Competitions and Conferences, 2002.