TsinghuaHephaestus RoboCup 2005 Technical Report

Mingguo Zhao, Yixin Cai, Dizhi Zhou, Hao Dong, Zhibin Liu, Zongying Shi

mgzhao@tsinghua.edu.cn
{caiyixin99, zhoudz04, donghao00, liu-zb04}@mails.tsinghua.edu.cn

1 Introduction

Our team TsinghuaHephaestus was established in 2003 as a part of the Lab for Robot Intelligent Control, Department of Automation, Tsinghua University. With the research interests on robotics and artificial intelligence, our software for robot soccer was developed on Aibo ERS-210A, in which many basic skills were learned from other teams.

In November 2004, we bought Aibo ERS-7 and started to work on the new version of our software. The whole system was restructured and most of the modules were rewritten. We participated in RoboCup 2005 in Osaka, Japan. As a new team, we have got many things during our first entrance to the international arena, including state-of-the-art of various technical fields, management skills for a successful team and the understanding of the game itself.

The rest of this report describes the approaches we used in the 2005 competition in detail, which is organized as follows. Section 2 illustrates the structure of the system. Section 3, 4 and 5 focuses on the implementation of vision, localization and motion modules. Section 6 introduces the routine and basic behaviors used to make decisions. The last section gives a summary of our experiences in Osaka and discusses the future development of our team.

2 System Overview

This section describes the software architecture. Functionally, the system consists of six modules: vision, localization, decision, wireless, motion and a shared memory module. Each module implements one logical task:

- **Vision**: processes the incoming image captured by the camera of Aibo, and generates information needed by other modules, such as the relative position of a seen beacon or ball to the robot.
- **Localization**: implements the localization algorithm and builds up the World Model, as to be used in Decision.
- **Decision**: holds the gaming process, and generates the robot's behavior when playing.
- **Wireless**: implements communication among robots.
- **Motion**: manages all actuators of the robot, e.g. the motors and LED's, and controls locomotion or any other action of the robot according to requests by Decision.
- **Shared Memory**: maintains all data exchange among the other modules, and logs any necessary information for debugging.

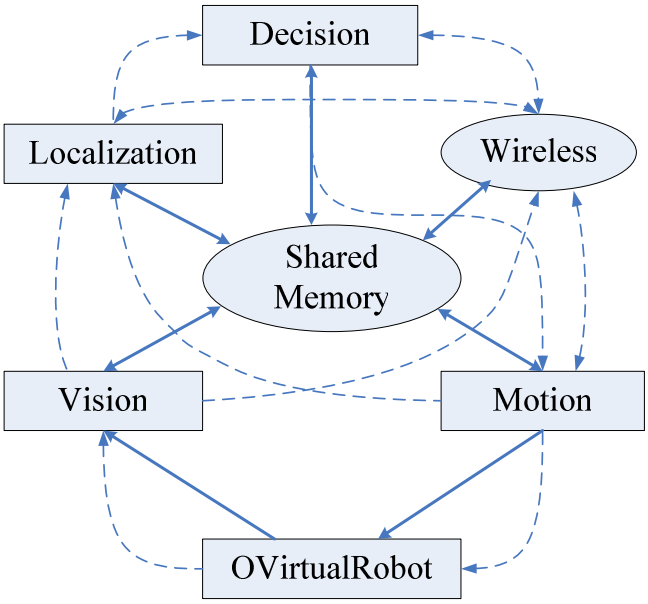


Fig. 1. System Architecture in Function Level.
Solid lines indicate the actual data flow while dashed lines the logical information flow.

The architecture above generally satisfies our application here in RoboCup. And the design of the Shared Memory module makes it easy for us to develop the other modules. For all other modules we only have to consider the data exchange between the module itself and the Shared Memory module. Although this kind of design may cause synchronizing problem, it rarely occurs during our experiments.

The six modules are organized into three objects: Cognition, Motion and Wireless, in which Motion and Wireless contain Motion Module and Wireless Module, while Cognition contains all the other modules. In this arrangement, necessary independence is achieved and

the traffic of data exchange between objects is reduced, thus cause fewer problems about OPEN-R messages and object synchronizing.

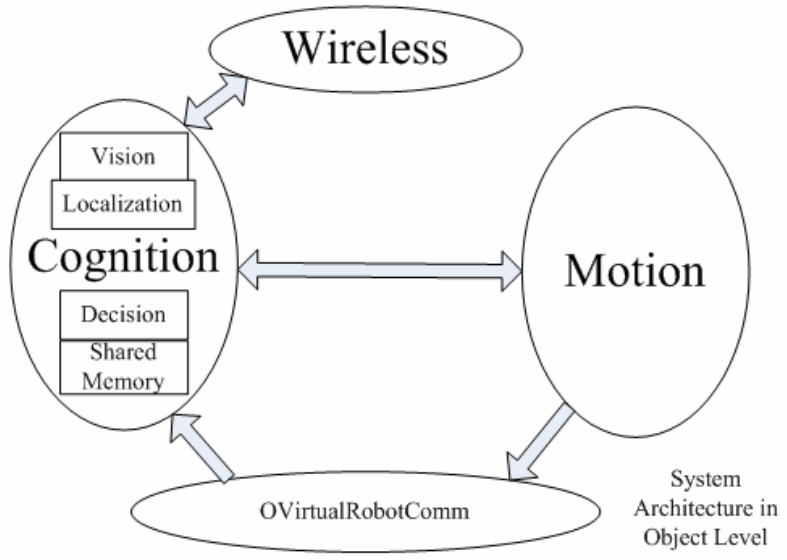


Fig. 2. System Architecture in Object Level.

3 Vision

The Vision Module deals with raw YUV images and outputs a list of all meaningful objects in the current scene. And this list contains necessary properties of the recognized objects. In other words, the main purpose of Vision Module is object recognition. In order to complete this, first we implement general and static color segmentation, and then object recognition based on blobbing from the segmented image.

3.1 Color Segmentation

We use a color lookup table to implement segmentation, which is generated from a C4.5 trained decision tree. The reason we employ decision tree, e.g. C4.5 algorithm, is that it has been proven to be robust against varying lighting condition, and that it is easy to get all its source code, thus saving our development work.

A large number of training data is needed for a decision tree to be trained to achieve good segmentation performance. And the training data set should cover the whole color space as possible. Normally, we capture about 30 frames of meaningful objects, such as goal, ball, robots, etc., and manually label all the pixels in the frames in order to be used as training data. Considering the varied lighting condition, those frames captured will also contain objects under shadow. The well prepared training data set is then fed to the C4.5 algorithm to generate the decision tree we need.

The performance of the decision tree results from the variety of the training data set and the quality of the manual labeling work. Sometimes it is necessary to adjust the structure of the training data set, e.g. excluding frames containing too much green field when there are too many green samples in the training data set. And in some cases we will try different labeling rules to get a better result.

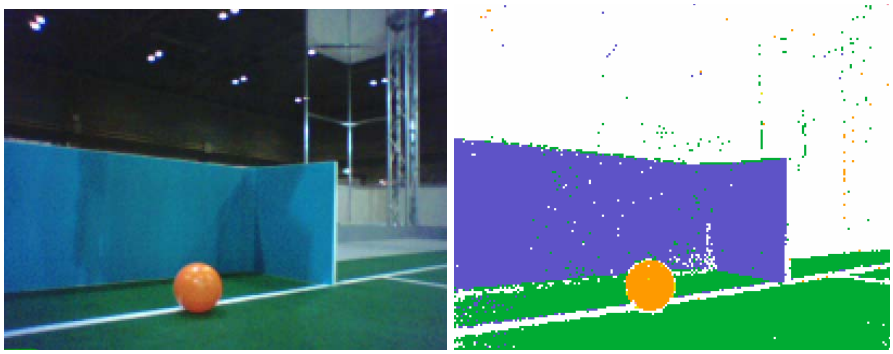


Fig. 3. Color Segmentation.
The left is the original picture randomly captured on the play ground, and the picture on the right is the color segmentation result of the left.

If the training samples are prepared carefully, the decision tree method can achieve a satisfactory segmentation performance. However, the process of preparing all those training

samples and the training of the decision tree is time consuming. Normally, about 2 hours is needed for us to build up a whole new lookup table. And it is also difficult to make adjustment for the lookup table, because operations on the samples are necessary and the decision tree need to be re-trained.

It is now one of our research interests seeking for more adaptive and robust methods of color segmentation, which need less calibration and manual adjustment, or which can handle the calibration process automatically.

3.2 Camera Model

As is known, in captured images distance from objects to the camera is lost. However, retrieving distance information is possible, for sizes of the meaningful objects, such as ball and beacons, are known.

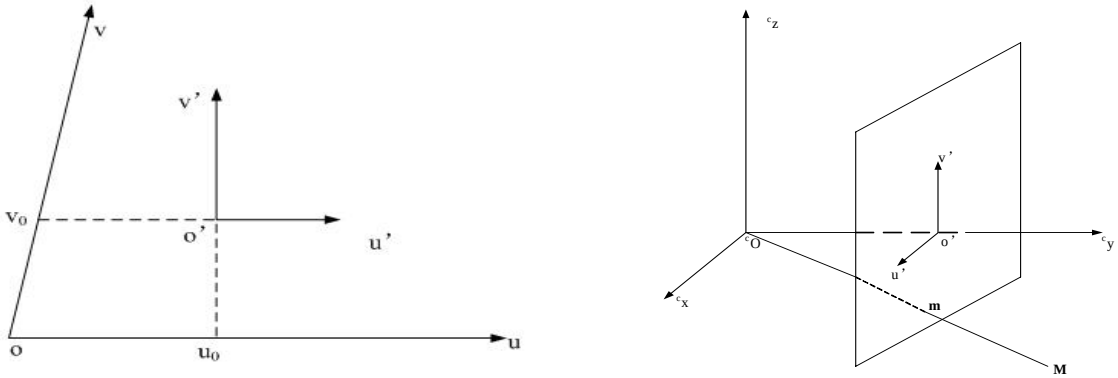


Fig. 4. Specifications for Coordinate Frames.

Fig. 4(a) specifies the image coordinate frame, and Fig. 4(b) the camera coordinate frame. The origin of the image coordinate frame is at the bottom left of the image. The origin of the camera coordinate frame is at the optical center of the camera's lens, and axis ${}^{c}y$ is along the axis of the lens. Plane $O'u'v'$ indicates the virtual image plane, which is symmetric with the image plane, thus the origin of the virtual image plane's coordinate frame is $\begin{pmatrix}0 & f & 0\end{pmatrix}^{T}$. The transformation from image coordinate frame to camera frame is as follows: (cf. Fig. 4)

$$\begin{bmatrix} {}^{c}x \\ {}^{c}y \\ {}^{c}z \end{bmatrix} = s \begin{bmatrix} \frac{1}{k_u f} & -\frac{\cos\theta}{k_v f} & \frac{-u_0 k_v f + v_0 k_u f \cos\theta}{k_u f\ k_v f} \\ 0 & 0 & 1 \\ 0 & \frac{\sin\theta}{k_v f} & -\frac{v_0 \sin\theta}{k_v f} \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \quad \textbf{(1)}$$

In the formula above, k_u and k_v are the unitage of axis u and v, θ is the angle between axis u and v (and normally $\theta \approx 90°$), f is the focus of the lens, and $[u_0, v_0]^T$ is the coordinate of O' in the image frame. s is the proportion between the distance from O to $\begin{pmatrix} {}^c x & {}^c y & {}^c z \end{pmatrix}^{\mathrm{T}}$ and that from O to $\begin{pmatrix} u' & v' & f \end{pmatrix}^{\mathrm{T}}$ in the camera frame (cf. Fig. 4).

In fact, note that ${}^c y = s\begin{bmatrix} 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} u & v & 1 \end{bmatrix}^{\mathrm{T}} = s$, as can be described as that all points on the image is projected to a plane perpendicular to the axis of the camera's lens, and the distance from the lens to this plane is s. Then for all objects, of which size is known, the distance from itself to the camera can be solved using triangle methods. First the image is projected to a plane with a certain s (any value as convenience), then the size of the projected object, the size of the actual object, and the assumed s are used to derive the actual s, finally the coordinate of the object is calculated using the derived s.

Parameters $\begin{pmatrix} k_u & k_v & u_0 & v_0 & f & s \end{pmatrix}$ need to be calibrated when implementing the method above. First we set up a plane perpendicular to the camera, thus the distance from the camera to the plane is measured as s. Then set up a grid of points in the plane, so that the coordinates of these points in the camera axis are known. Capture image from the camera and measure all those coordinates of the points in the image axis. The data set obtained is then used for calibration.

3.3 Object Recognition

The recognition of meaningful objects is based on the color segmentation result of an input raw image. For all segmented images, length encoding and region merging algorithms are performed, and a list of blobs for each color is generated as output. Elements in the blob list contain properties of each blob, e.g. size and position in the image, weight, center, etc. Object recognition is then implemented using information in the blob list. Take the recognition of the ball as example, first all orange blobs with proper size will be considered as candidate, and then a serial of rule checks are performed in order to exclude those unreasonable candidates, finally the blob that passes all the rule checks is recognized as the ball.

This method is simple, but effective. During the segmentation and the blobbing process the whole image is scanned twice, and no more scanning is performed, thus the computational expense is affordable. The performance of recognition depends on the quality of color segmentation and significantly the rules for checking. And all the rules are derived from common experience of developers, such as the rule that blob size of a goal candidate should be larger than a threshold, etc.

The method uses statistical information, that is to say, it takes all the pixels in the image into account. However, the detail of the image, e.g. the edges, is not employed. It is also very difficult to detect and recognize the field lines on the play ground using this method.

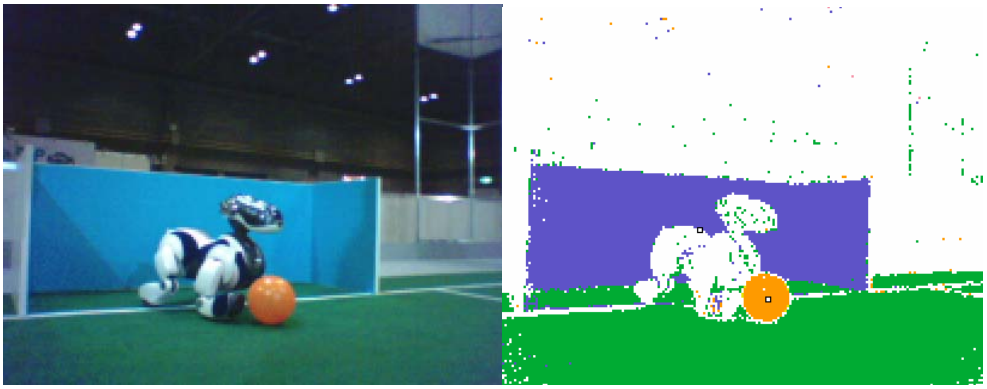


Fig. 5. Object Recognition.
The left scene is the original picture randomly captured in the play ground. On the right side is the segmented image, in which the two squares indicate the center of the recognized goal and ball.

4. Localization

Localization module in our system deals with locating the robots itself as well as the ball in the field coordinates. It reads the object list generated by vision module and creates world model for decision module.

World model contains state information about the field, which includes the posture of a robot $(x, y, \theta)^T$ and the position and velocity of the ball $(x, y, dx, dy)^T$. Each robot maintains two world models, called local world model (LWM) and global world model (GWM). LWM only contains information obtained from the robot itself.

4.1 Robot and Field Coordinate Frames

Two coordinate frames are used in the localization module, field coordinate frame and robot coordinate frame.

The robot coordinate frame is sometimes referred as local coordinate frame, whose origin is the point on the field just under the center of neck tilt joint. While the y-axis is on the field and along the heading of the robot and x-axis points to the right. (See Fig. 6)

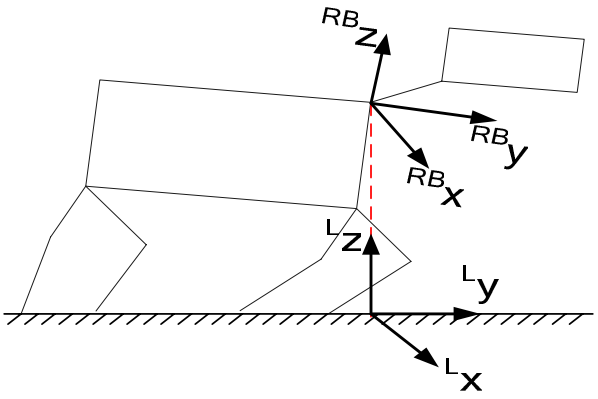


Fig. 6. Robot(Local) coordinates(L)

The field coordinate frame is sometimes referred as global coordinates, whose origin is located on the bottom left corner of own field, with y-axis pointing from own goal to opponent goal and x-axis pointing to right across the field. (See Fig. 7)

The relation between these two coordinate frames is shown in equation (2) and (3).

4.2 Global Self-localization

Knowing its own position in the field coordinate frame is essential for soccer robot to make proper decision and cooperate with other robots. In RoboCup Four-legged League, the field

has a fixed size with standard color labeled goals, beacons and lines, and the main sensor of the robot is a camera. So the localization problem here can be categorized as map-based vision localization under a structured environment.

$$\begin{bmatrix} {}^{G}x \\ {}^{G}y \\ {}^{G}\theta \\ 1 \end{bmatrix} = \begin{bmatrix} C({}^{G}\theta_R) & -S({}^{G}\theta_R) & 0 & {}^{G}x_R \\ S({}^{G}\theta_R) & C({}^{G}\theta_R) & 0 & {}^{G}y_R \\ 0 & 0 & 1 & {}^{G}\theta_R \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} {}^{RL}x \\ {}^{RL}y \\ {}^{RL}\theta \\ 1 \end{bmatrix} \quad \textbf{(2)}$$

$$\begin{bmatrix} {}^{RL}x \\ {}^{RL}y \\ {}^{RL}\theta \\ 1 \end{bmatrix} = \begin{bmatrix} C({}^{G}\theta_R) & S({}^{G}\theta_R) & 0 & -C({}^{G}\theta_R){}^{G}x_R - S({}^{G}\theta_R){}^{G}y_R \\ -S({}^{G}\theta_R) & C({}^{G}\theta_R) & 0 & S({}^{G}\theta_R){}^{G}x_R - C({}^{G}\theta_R){}^{G}y_R \\ 0 & 0 & 1 & -{}^{G}\theta_R \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} {}^{G}x \\ {}^{G}y \\ {}^{G}\theta \\ 1 \end{bmatrix} \quad \textbf{(3)}$$

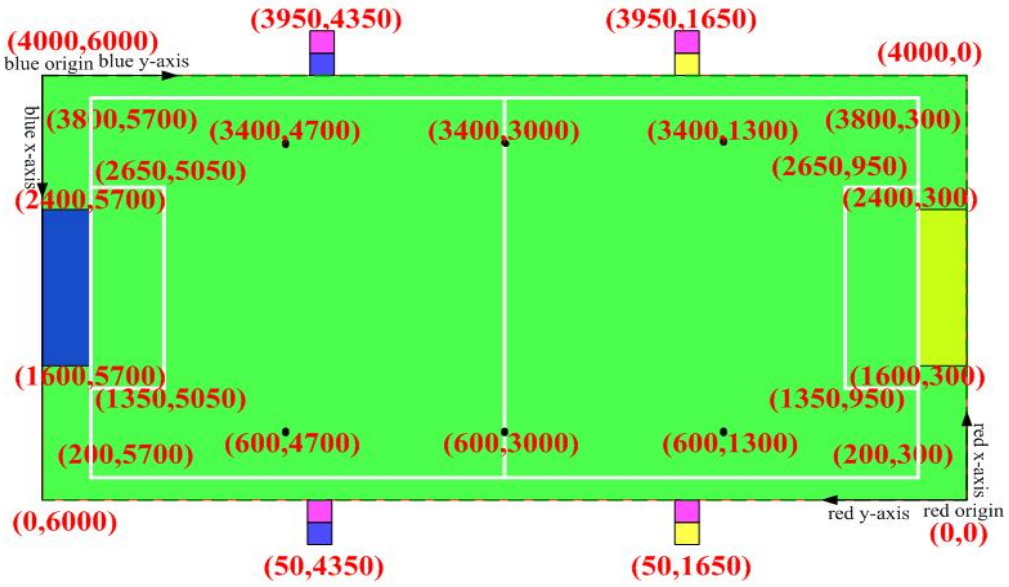


Fig. 7. Field coordinates and location of landmarks

There are still some special features in robot self localization problem in this league. First of all, the robot's vision sensor has a very limited field of view. In Small Sized League, some ceil cameras are used to provide global view. In Middle Sized League, omni-vision systems implemented by a properly designed mirror are used by most teams. While in this league, Aibo has a visual field of about 50 degree in both dimensions and a max distance for reliable recognition of meaningful objects around 3 meters. However, the cooperation of robots requires a robot's position in a global system in some sense. Relative distances and postures are not enough for robots to cooperate.

Secondly, the localization algorithm must be capable of providing position information continuously while the robot moves around. Although there are obvious artificial landmarks on the field and it is quite straightforward to calculate the robot's posture from its relative position to two different landmarks, this situation does not happen often during the game due to the limited view of the vision sensor and the locomotion which can hardly be modeled.

Finally, the localization algorithms must deal with collision and kidnapping which is usually undetectable. They are supposed to tolerant these kinds of noises or detect these situations and recover automatically.

The localization problem here can be modeled as state estimation for dynamic systems, which can be solved with Kalman filters. [2]

Kalman filter has been applied to various engineering problems since it was developed in 1960s. The basic form of Kalman filter is as follows.

Assume that a discrete linear system can be represented with following formulas

$$\mathbf{x}_{k+1} = \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k + \mathbf{w}_k \tag{4}$$

$$\mathbf{z}_{k+1} = \mathbf{H}\mathbf{x}_{k+1} + \mathbf{v}_{k+1} \tag{5}$$

Where $\mathbf{x}_k$ is the state of the system at time k, $\mathbf{w}_k$、 $\mathbf{v}_k$ are noises. Now the parameters of model, $\mathbf{A}$, $\mathbf{B}$, $\mathbf{H}$, input $\mathbf{u}_k$ and observation $\mathbf{z}_{k+1}$ are known, we expect the best estimation of the state $\mathbf{x}_k$. Kalman filter gives an iterative algorithm for this problem. [3]

In our implementation, the state vector of the dynamic system is $\begin{bmatrix} x_k & y_k & \theta_k \end{bmatrix}^T$, which represents the position and posture of the robot in field coordinates.

The input vector $\begin{bmatrix} \Delta f_k & \Delta l_k & \Delta\theta_k \end{bmatrix}^T$ comes from the motion module, which represents the estimated motion of the robot last time. This odometry information can be converted into field coordinates as:

$$\begin{bmatrix} x_{k+1} \\ y_{k+1} \\ \theta_{k+1} \end{bmatrix} = \begin{bmatrix} x_k \\ y_k \\ \theta_k \end{bmatrix} + \begin{bmatrix} \cos\theta_k & -\sin\theta_k & 0 \\ \sin\theta_k & \cos\theta_k & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} -\Delta l_k \\ \Delta f_k \\ \Delta\theta_k \end{bmatrix} = \begin{bmatrix} x_k - \Delta f_k \sin\theta_k - \Delta l_k \cos\theta_k \\ y_k + \Delta f_k \cos\theta_k - \Delta l_k \sin\theta_k \\ \theta_k + \Delta\theta_k \end{bmatrix} \tag{6}$$

Taking the noise of odometry into account, the formulas above are rewritten as:

$$\begin{bmatrix} x_{k+1} \\ y_{k+1} \\ \theta_{k+1} \end{bmatrix} = \begin{bmatrix} x_k - \Delta f_k \sin\theta_k - \Delta l_k \cos\theta_k \\ y_k + \Delta f_k \cos\theta_k - \Delta l_k \sin\theta_k \\ \theta_k + \Delta\theta_k \end{bmatrix} + \begin{bmatrix} \cos\theta_k & -\sin\theta_k & 0 \\ \sin\theta_k & \cos\theta_k & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} w_k^x \\ w_k^y \\ w_k^\theta \end{bmatrix} \tag{7}$$

The observation vector $\begin{bmatrix} {}^L x_k^b & {}^L y_k^b \end{bmatrix}^T$ comes from the vision module, which represents the position of landmarks in the robot coordinates (denoted with upper note L). For a specific landmark b, whose position in the field coordinates is $\begin{bmatrix} x^b & y^b \end{bmatrix}^T$, its position in L coordinates and field coordinates has a relationship as follows:

$$\begin{bmatrix} {}^{L}x_k^b \\ {}^{L}y_k^b \end{bmatrix} = \begin{bmatrix} x^b \cos\theta_k + y^b \sin\theta_k - x_k \cos\theta_k - y_k \sin\theta_k \\ -x^b \sin\theta_k + y^b \cos\theta_k + x_k \sin\theta_k - y_k \cos\theta_k \end{bmatrix} \quad \textbf{(8)}$$

The state of the system can be estimated using Extended Kalman Filter, which takes two steps. When the odometry information is available, state is predicted according to the following formulas:

$$\begin{bmatrix} \hat{x}_{k+1}^- \\ \hat{y}_{k+1}^- \\ \hat{\theta}_{k+1}^- \end{bmatrix} = \begin{bmatrix} \hat{x}_k - \Delta f_k \sin\hat{\theta}_k - \Delta l_k \cos\hat{\theta}_k \\ \hat{y}_k + \Delta f_k \cos\hat{\theta}_k - \Delta l_k \sin\hat{\theta}_k \\ \hat{\theta}_k + \Delta\theta_k \end{bmatrix} \quad \textbf{(9)}$$

$$A_{k+1} = \frac{\partial f}{\partial X} = \begin{bmatrix} 1 & 0 & -\Delta f_k \cos\hat{\theta}_k + \Delta l_k \sin\hat{\theta}_k \\ 0 & 1 & -\Delta f_k \sin\hat{\theta}_k - \Delta l_k \cos\hat{\theta}_k \\ 0 & 0 & 1 \end{bmatrix} \quad \textbf{(10)}$$

$$W_{k+1} = \frac{\partial f}{\partial w} = \begin{bmatrix} \cos\hat{\theta}_k & -\sin\hat{\theta}_k & 0 \\ \sin\hat{\theta}_k & \cos\hat{\theta}_k & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \textbf{(11)}$$

$$P_{k+1}^- = A_{k+1} P_k A_{k+1}^T + W_{k+1} Q_k W_{k+1}^T \quad \textbf{(12)}$$

When visual observation is available, the estimated state is corrected as follows:

$$K_{k+1} = P_{k+1}^- H_{k+1}^T (H_{k+1} P_{k+1}^- H_{k+1}^T + V_{k+1} R_{k+1} V_{k+1}^T)^{-1} = P_{k+1}^- H_{k+1}^T (H_{k+1} P_{k+1}^- H_{k+1}^T + R_{k+1})^{-1} \quad \textbf{(13)}$$

$$\begin{bmatrix} \hat{x}_{k+1} \\ \hat{y}_{k+1} \\ \hat{\theta}_{k+1} \end{bmatrix} = \begin{bmatrix} \hat{x}_{k+1}^- \\ \hat{y}_{k+1}^- \\ \hat{\theta}_{k+1}^- \end{bmatrix} + K_{k+1} \begin{bmatrix} {}^{L}x_{k+1}^b + (\hat{x}_{k+1}^- - x^b)\cos\hat{\theta}_{k+1}^- + (\hat{y}_{k+1}^- - y^b)\sin\hat{\theta}_{k+1}^- \\ {}^{L}y_{k+1}^b - (\hat{x}_{k+1}^- - x^b)\sin\hat{\theta}_{k+1}^- + (\hat{y}_{k+1}^- - y^b)\cos\hat{\theta}_{k+1}^- \end{bmatrix} \quad \textbf{(14)}$$

$$P_{k+1} = (I - K_{k+1} H_{k+1}) P_{k+1}^- \quad \textbf{(15)}$$

4.3 Ball Tracking

The ball tracking is also implemented using EKF. When the robot is at position $\begin{bmatrix} x_k^r & y_k^r & \theta_k^r \end{bmatrix}^T$ and observes the ball at position $\begin{bmatrix} {}^{L}x_k & {}^{L}y_k \end{bmatrix}^T$, the position of the ball in the field coordinates $\begin{bmatrix} x_k & y_k \end{bmatrix}^T$ can be calculated as:

$$\begin{bmatrix} {}^{L}x_k \\ {}^{L}y_k \end{bmatrix} = \begin{bmatrix} x_k \cos\theta_k^r + y_k \sin\theta_k^r - x_k^r \cos\theta_k^r - y_k^r \sin\theta_k^r \\ -x_k \sin\theta_k^r + y_k \cos\theta_k^r + x_k^r \sin\theta_k^r - y_k^r \cos\theta_k^r \end{bmatrix} + \begin{bmatrix} v_k^x \\ v_k^y \end{bmatrix} \quad \textbf{(16)}$$

A static ball model is used in EKF, which represents a ball standing still in its original position if there is no observation of its moving. The algorithm is implemented motion update phase and vision update phase similar to what is described in 4.2.

4.4 Creating Global World Model

To create a global world model, LWM from other robots are received and evaluated. The results are then combined with one's own LWM.

For the location of other teammates, information contained in the received LWM is inserted into GWM directly. If there is no LWM from another robot for a while, its location in GWM does not change while the corresponding confidence factor decreases at a constant speed.

However, it's a little complicated with the ball because it can be seen by more than one robot. So the state of the ball is fused with the information from others using a method similar to Kalman Filter.[4] Because it confuses a robot with a wrong ball location too often, this function is not used during the game.

4.5 Result and future work

This localization module works quite well in experiment. Our robot can move to the desired positions in Ready state automatically within a relatively short time and with certain accuracy. However, the robot often lost its position during a real game due to lack of enough visual cues, especially near the corners of the field.

In the future, field lines must be used as meaningful landmarks and an efficient behavior to locate the robot when necessary is also needed.

5. Motion

As we all know, locomotion is one of the most important skills in AIBO soccer game. Our locomotion module receives the instruction from behavior module and actualizes it on the hardware of AIBO-ERS7 through OPEN-R. Fig. 8 shows the architecture of the locomotion module.

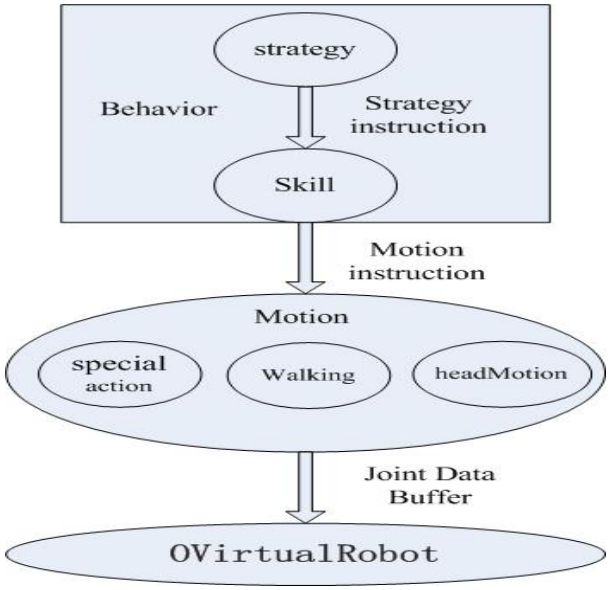


Fig. 8. Architecture of the Locomotion Module

There are three main foundational sub-modules in locomotion: Walking, Special-action and Head Motion.

Walking

Our inverse kinematics leg model of AIBO is based on Bernhard's omni-directional walk paper[5].Our WalkingEngine calculates the paw position according to the Motion instruction , and the angle of the joint from the paw position is then calculated using model information of AIBO-ERS7 which is provided by Sony corporation.

The parameters for walking omni-directional also from reference to the Bernhard's omni-directional walk paper are up to 13. We spend a lot of time to optimize these parameters to make the dog walk faster and more stable using hand-tuning. Many parameters and foot locus including rectangle and half ellipse are checked to make the dog walk faster and more stable. At last we find several parameters for different moving directions. We use foot locus of half ellipse for fast forward and turning, while rectangle for sideway and backward. The max forward speed is 30cm/s. The WalkingEngine will choose the appropriate parameters for different walking instructions. We will use the machine learning approach to optimize the walking parameters in the next year.

Special-action

We develop many special actions of ERS-7, but only little of them can work effectively in the competition. There are mainly three group of actions: Getup、kicking and blocking . The Getup action includes the *GetupFront*, *GetupRear* and *GetupSide* which make the dog stand up from front, rear and side when it fall over itself. The kicking action includes *ForwardKick*, *HeadKick*, *ChestPush* and others which are effective in shooting the ball by different part of the dog in different power. The blocking includes block right, block left, block middle which is used by goalie and defensive player to block the ball from any direction.

Although we have so many kicking actions, how to select each action during the game based on the robot and ball position for different aiming direction is the key point. So we create the kicking evaluation experiment table to solve this problem. Fig. 9 shows the experiment table.

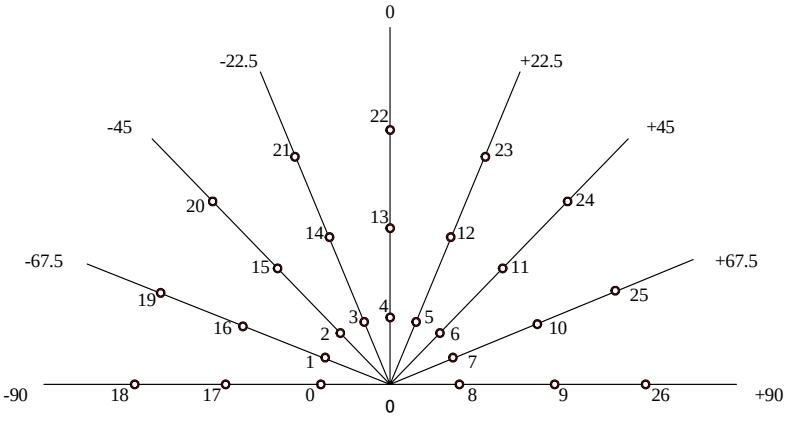


Fig. 9. K**icking evaluation experiment table**

As in the Fig. 9, the circle point is the ball position and the point O is the origin of the dog. Seven lines divides region of the dog ‘s forward direction into eight sectors with 22.5 degree in each step .Point 0 to 8 is the nearest kicking point, point 9 to 17 is the best kicking point, point 18 to 26 is the farthest kicking point. We put the ball in each point and make the dog do one kicking, then record the distance and direction of the ball’s final position for one evaluation. Finally each kicking action and its evaluation records are stored in a table which is used for behavior module to choose the effective action.

Head motion

The AIBO ERS7 head has three degrees of freedom: neck tilt, head pan, and head tilt. There are two kinds of motion: looking absolutely and looking relatively. The absolute type is used in pointing at beacon and goals, and the relative type is used in pointing at ball.

6. Decision

This module is responsible for making decisions and controlling behaviors. It uses the information coming from ***G**lobal **W**orld **M**odel* (GWM), game controller commands, and team messages from teammates.

The decision making routine has a layered structure, which is similar to a decision tree. However, the leaf nodes, which represent the basic behaviors, can be referred by different parent nodes.

We incorporated a finite state machine into the layered structure, and used three variables to determine transitions between different states. These variables are: ***S**elf **L**ocalization **C**onfidence* (SLC), ***G**lobal **B**all Location **C**onfidence* (GBC) and ***L**ocal **B**all Location **C**onfidence* (LBC). According to different combinations of them, different behaviors are chosen, e.g. when the combination of the three variables means current location of the robot itself is unreliable and there is little knowledge about the ball, the next behavior is most likely to locate the ball and the robot itself.

The following part presents a bottom to top description of our Decision module in details.

6.1 Localization Behaviors

There are two kinds of localization behavior: *Active Localization* and *Beacon Tracking*. When the *SLC* is very low, *Active Localization* is performed, otherwise *Beacon Tracking* is performed to improve the reliability and precision of the robot's localization result.

The whole procedure of Active Localization comes into three steps: 1) swaying head to search for landmarks without moving its body; 2) swaying head and turning the body without changing its position; 3) navigating on the field with head swaying.

What deserves a special mention is that, when the robot sways its head, the tilt angle of the head should have a proper value to make sure the robot has the maximum probability to see the upper parts of the beacon, because only these parts can provide distinguishing information for localization.

Additionally, there is another fact should be taken into account: if the robot keeps tracking only one landmark, there will not be enough evidence to achieve accurate estimation of the robot's location. Hence, we maintain a table of landmarks that are probably in front of the robot. The landmarks in the table are sorted in a distance (relative to the robot) ascending manner, and accordingly the robot will look to the direction of them in a sequential way so as to gain adequate information to improve localization result.

As we know, the nearer the landmark is, the less uncertainty and error of measurement comes about. Therefore, we give a priority to making use of the closest landmarks, and this is the exact reason why we sort the landmarks in a distance ascending manner as mentioned above.

6.2 Searching Ball

Because of the limited view of the robot and the unpredictable dynamic of the soccer game, it is hard to make sure that robot could keep the ball in its sight at every view. So we employ the following strategy for searching ball.

When the searching ball routine is called for the first time, there may be two different cases: if the robot lost track of the ball not long before, it will look to the direction where the ball was last seen; otherwise it will pay its attention to the area around the *known ball* which is obtained from team messages.

If the ball is not found after the first step, the robot will sway its head while standing still. However, due to the limited view of the robot, it can't make a complete search of the field in front of it by just swaying its head once. As a result, we make the robot sway its head from left to right at a given tilt angle, and scan back at another tilt angle (see Fig. 10). Additionally, the planed angle of the goalie is a little different to that of field players, because at most of the time the ball might be quite far from the goalie, so its view should cover the relatively further section of the field.

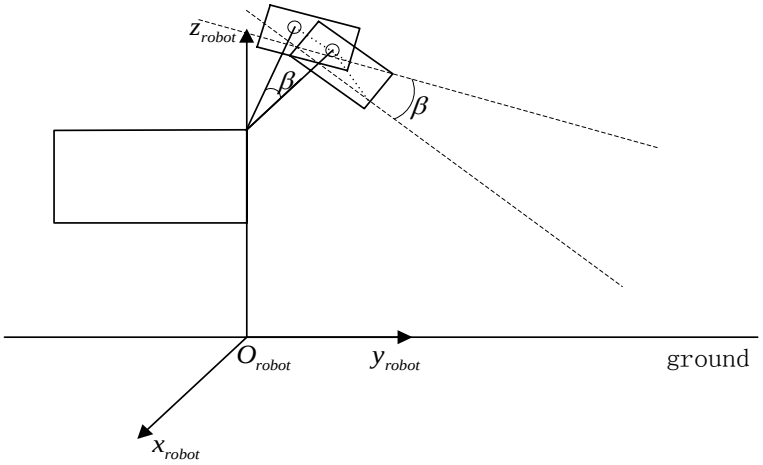


Fig. **10**. **Tilt angle in searching ball**
Sway head from left to right and sway back with different tilt angle. The discrimination angle is denoted as β .

The tilt angle of the head is calculated as follow: 1) for goalie, the greater tilt angle makes the optical axis of the camera 10° below the horizontal plane; 2) for field players, 15° below the horizontal plane. However, the discrimination angle for all robots is 40° .

If the ball is still not found after the second step of the searching ball routine, the robot will perform the third step, during which the robot cruises around the field with head swaying simultaneously. If the confidence of the *known ball* location is higher than a certain threshold, the robot would give a priority to cruise around the *known ball* location first.

For the goalie, its all-important responsibility is to block in front of the goal, so in the third step of the searching ball routine, it should not move away from current position, but only turn its body, searching for ball. As field players, if the *known ball* position is reliable, it will turn to that direction first.

6.3 Tracking Ball

Tracking ball is an essential behavior for soccer playing. There are mainly two kinds of information about the ball to be utilized to perform tracking ball behavior: one is the local ball information when the robot actually sees the ball, and the other is the global ball information from other sources.

Our strategy is to make the robot keep the ball in center of its sight. If the robot actually sees the ball, it should keep the ***real ball*** in center of its sight; otherwise, it will turn its head to certain direction to make the position of ***known ball*** in center of its sight. The reason for choosing such a strategy is: if you keep something in the central area of your sight, then no matter which direction it moves to, you will have the maximum probability to still have it in sight at your next glimpse.

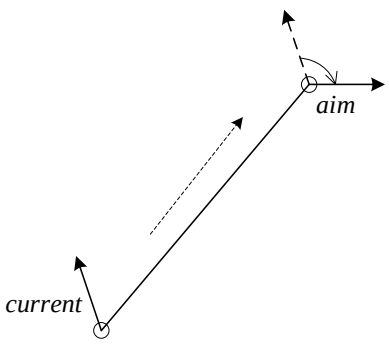
6.4 Moving to Point

This routine is responsible for generating a series of motion commands, given the current position and posture of the robot and the aim position and posture.

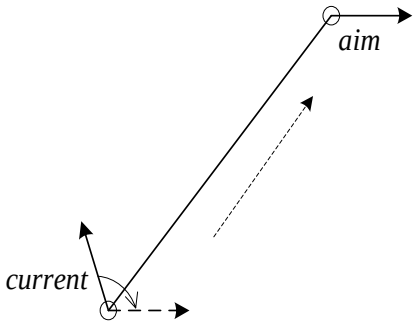
Our consideration about how to make the robot move to the aim position and posture is the time cost should be the lowest. For convenience, let's denote *position and posture* as ***PS*** (position state), and there are basically four different strategies for moving from one *PS* to another (see Fig. 11), they are: *1)* moving from the current position to the aim position along the line first, and when the aim position is reached, turning to the aim posture (Fig. 11 (a)); *2)* turning to the aim posture first, and then moves to the aim position along the line (Fig. 11 (b)); *3)* turning to directly head to the aim position first, and then moving forward to the aim point, when reached, turning again to the aim posture (Fig. 11 (c)); *4)* turning to face the opposite direction to the aim position first, then moving backwards to the aim position, and finally turning to the aim posture when the aim position is reached (Fig. 11 (d)).

Every time, the program calculates the over all time consumption of the four strategies, and picks out the most efficient one, then, accordingly, the next motion commands is generated.

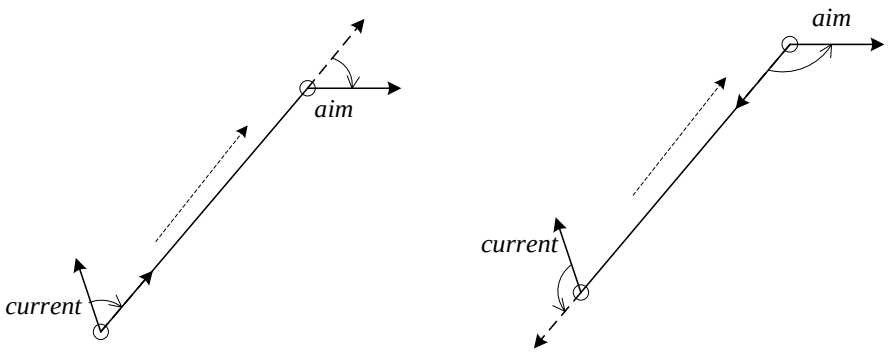
Though these four strategies seem too simple and may not cover all conditions, it can adapt to the fast changing situation and deal with various circumstances. It means that the chosen strategy can switch from one to another at any step during the whole moving process.



(a)



(b)



(c) (d)

Fig. 11. Four strategies for moving from one *PS* to another

7. Result and Future work

In RoboCup 2005, we have got two lost and one win. It is not a satisfying result indeed. The first match was lost against DogBots with score of 1 to 3, although we have some opportunities to make more goals in the first half. The second match we won with score of 1 to 0 against TeamChaos. And in the Intermediate Round match we lost with score of 0 to 1 against UChile1 in the last 3 seconds. It is a pity that we missed the chance to go further.

The most important reason for our lost is the limit of our dogs' speed and the capability to control the ball. Due to the lack of time and technical support, the gait of our dogs is tuned by hand, which made it so slow compared to others. We will do more work on machine learning in gait optimization in this year and hope to make the speed of our dog never be the choke point of our team.

Finishing matches in the competition, we watched other games seriously. From the other team's highlight in the match, we have got the point that controlling the ball is the key of winning. Since the field became larger and the walls around it were removed, the league seems to encourage everyone to control the ball and do some passing and accurate shooting which is more like human soccer. The rUNSWift team, Nubots and GermanTeam did very good jobs on these and achieved success in the competition. Controlling the ball is a complex task for behavior, vision and locomotion. So we will do more work in these subjects to get a more precise ball position and more accurate movement.

After participating in RoboCup 2005, we are getting more and more interested in this activity, and are eager to try our best to make some contributions to the league.

How to make robots do right things in a dynamic or even unknown environment is still an open problem. There are many theoretical and practical problems remained for further research. Among these problems, our recent research work will be focused on:

- Constructing a vision system which is robust to varying lighting condition
- Localization in a highly symmetrical environment with ambiguous features
- Machine learning and its application in legged gait optimization

References

[1] RoboCup official Website, http://www.robocup.org
[2] B. Hengst, C. Sammut, W. Uther and etc. “Team report of rUNSWift 2003”, Nov. 2003
[3] G. Welch, G. Bishop, “An Introduction to the Kalman Filter”, SIGGRAPH 2001, Course 8
[4] Ashley W. Stroupe, Martin C. Martin, and Tucker Balch, Distributed Sensor Fusion for Object Position Estimation by Multi-Robot Systems, ICRA2001
[5] Bernhard Hengst, Darren Ibbotson, Son Bao Pham, and Claude Sammut (2001). Omni-directional Locomotion for Quadruped Robots. Lecture Notes in Computer Science, RoboCup 2001: Robot Soccer World Cup V, pages 368–373. Springer, 2002.