CMDash'05: Team Report

Manuela Veloso, Paul E. Rybski, Sonia Chernova, Colin McMillen, Juan Fasola,
Felix vonHundelshausen, Douglas Vail, Alex Trevor, Sabine Hauert,
Raquel Ros Espinoza

School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213
http://www.cs.cmu.edu/˜coral

1 Introduction

The CMDASH'05 team follows our past teams CMPack'04, CMPack'03, CMPack'02, CMPack'01, CMPack'00, CMTrio'99, and CMTrio'98 [27, 19, 30, 29]. This is our second year of using the Sony ERS-7 robotic platform. We have continued our research into new ways of modeling the world and maintained our focus on robust behaviors and cooperation. In this paper, we provide an overview of the specific technical components in our team and focus in detail on several recent additions.

2 Vision

The vision module is responsible for converting raw YUV camera images into information about objects that the robot sees and are relevant to the task. This is done in two major stages. The low level vision processes the whole frame and identifies regions of the same symbolic color. As in previous years, we use CMVision [7, 6] for our low level vision processing. The high level vision module uses these regions to find objects of interest in the image. We have continued to improve both aspects of the robot's vision which allows our team to operate well in a variety of lighting conditions.

2.1 Line Vision

This year we extended our vision system with a line vision module that is able to recognize straight lines, corners and the center circle at a rate of 30 frames per second. For all features, both position and orientation are recovered with high accuracy. For the center circle, the orientation is obtained by determining the orientation of the center line that passes through the circle. Corners and the center circle can only be detected up to a distance of approximately 1.5 meters from the robot. These detected features are used by our localization framework.

The line vision module is fast enough to run in parallel with the CMVision color segmentation system. The overall processing is similar to the vision system of the FU-Fighter's MidSize [31]. We use the region tracker [32] to extract the boundary contours of all white regions that are below the horizon. Here we run the region tracking on top of our segmentation results, determining the boundary contours of the white areas in the segmented images. Then we process the contours similar as described in [31].

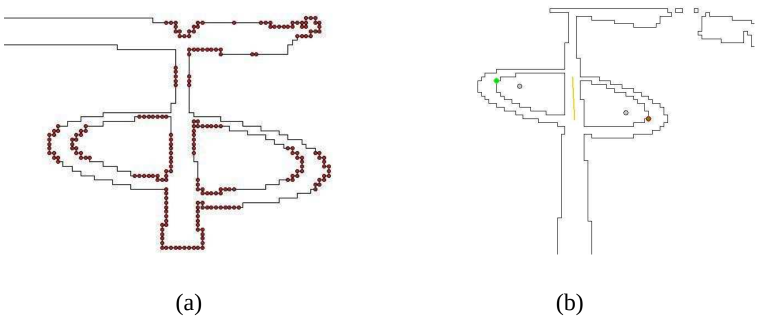


Fig. 1. (a) The small circles identify areas of high curvature found on the lines. (b) The center circle as identified by the AIBO vision system.

We split the contours at high curvature points and classify between straight and curved segments.

Parallel straight segments spaced at a small distance are identified to represent the two borders of a field line and are grouped together. Finally both sides of a hypothetical field line are checked for green. Corners are detected on the base of the straight lines. The center circle detection is different from [31]. Because of the larger perspective distortion we could not directly use the MidSize method. The modified method is based on detecting the two halves of the center circles. By using the region tracking algorithm, the half circles can be detected extremely efficiently because the rotation of the boundary contours of the half-circles is different from the other lines. In each hypothetical half of the center circle we match a triangle, two points being at the intersection of the center line and the circle and the third point located on the curved part of the half circle. On the basis of these triangles we perform several consistency checks: We have to find two triangles that have two sides that are parallel and close to each other and also we run different checks on the angles of both triangles. Finally, we check whether both halves of the center circle are green. In contrast to the FU-Fighters Middle Size feature detection, the center circle detection will fail in the presence of occlusion. Figure 1(b) shows the circle identified by an AIBO standing on the center line of the field.

2.2 Fast, Lookup Based Correction of Chromatic Distortion

The ability to quickly and accurately color segment an image is both the most basic and the most important task of the robot's vision system; color has semantic meaning in the RoboCup domain and incorrectly segmenting a scene can have serious consequences. For example, a robot that mistakes green carpet for a cyan goal at the corner of the camera image [perhaps due to chromatic distortion from the lens], might incorrectly update its localization estimate based on the spurious landmark. Or, even worse, it might kick the ball towards this phantom goal in the corner of the image.

The above scenario with the spurious goal is a very real issue; there is significant chromatic distortion in the images from the ERS-7's camera. The most noticeable trait is that pixels are biased towards blue the further they are from the center of the image. Rofer et al describe this effect and propose an approach for correcting chromatic distortion based around the idea of decomposing the distortion into two, independent components. These components are the intensity of the pixel and the distance of the pixel from a center point in the image. Each of the Y, U, and V channels are corrected independently and the correction is implemented through the use of lookup tables so that it can efficiently execute in real time on the robot. Our color correction code is an independent implementation of this approach that was described and implemented by the German Team in their champion 2004 RoboCup code [23].

2.3 Quantifying and Correcting Distortion

As pointed out in [23], the color distortion is best observed by gathering images of uniformly colored objects such as the cyan and yellow goals or a blank, white surface. We define the "true" color of each image to be the mean of all (Y,U,V) triplets in the image. In a camera with zero distortion and under perfect conditions, all pixels in the image would have exactly this value. The distortion of an individual pixel is taken as $YUV_{actual} - YUV_{image\ mean}$. Note that each component is treated independently; we will produce corrections for each channel without reference to the other two. The overall distortion of each color channel can be taken to be the total distortion of that component from pixels in the image, which we compute as the sum of the squared distortion of each value:

$$\sum_{P \in Image} (P_c - \mu_c)^2$$

P_c is the value for component c of each pixel and μ_c is the mean of that component over the entire image.

Following the decomposition proposed by the German Team, we assume that the distortion for each pixel is affected by two, independent factors. These are the radius of the pixel from the center of the image and the intensity value for that component at that location. This means that the distortion equation can be broken down into:

$$P_c - \mu_c = RadialDistortion_c(r(P)) \cdot IntensityDistortion_c(P_c)$$

As above, c indicates which color component is currently under consideration. $r(P)$ represents the distance of pixel P from the center of the image. Note that both the radial distortion and intensity based correction functions differ for each color component. The true form of the correction functions is unknown, so we chose to approximate them with polynomials. Empirical experimentation showed that second degree polynomials yielded the best performance.

To discover coefficients for the two polynomials for each color channel, we used the downhill simplex method as described in [22]. In practical terms, this leads to performing three independent optimizations, one for each channel, and with optimization performed in a six dimensional space since we used second degree polynomials. The objective function being optimized was the sum squared error over several training images from each of the yellow goal, the blue goal, and a white wall. The goal when gathering these training images was to photograph a region of uniform color and shadowing.

Implementation Notes Conceptually, it is easy to think of the correction as computing each of the two correction factors, multiplying them together, and then adding the result to the measured value for each pixel. However, this is not particularly efficient.

In practice, it is easier to precompute tables and lookup the new pixel value given the measured value and the radius of the pixel in question. This can be done via two lookups. First, a lookup is performed to determine the distance of the current pixel from the center of the image. The offset of the pixel in the raw bitmap from the camera is used as the index and the table has a size linear in the size of the image. The second lookup is into a two dimensional table that is indexed by the radius of the pixel as well as the intensity value for the channel in question. In total, there are three such tables, one for each channel, and the size of each table depends on the distance from the center of the image to a corner and the color depth of each individual channel. Using 8 bits for each channel, the resulting tables are roughly the same size as an image, so storage space is not a problem. This second table contains the new pixel component value itself rather than a correction to avoid extra computation. So, to correct each pixel, four lookups are necessary. The first is to find the radius of the pixel and the remaining three are to load new values for each of the Y, U, and V channels.

Both the radius lookup table and the color correction table are generated on bootup using stored values of the coefficients for each of the six polynomials.

2.4 Robot Detection

One way to address the problem of robot detection is to search for red or blue regions in the color segmented image, and if the regions conform to some defined constraints, a robot is then detected. This seems like a reasonable solution, however, in our experience the shade of blue of the robot uniforms is so dark that it shows up as black in the color segmented images. Since the background color is black and the shadows on the robot are also black, finding blue robots effectively with this method is difficult. In addition, trying to correct the segmentation to emphasize the recognition of the uniform color blue results in classifying a lot of the background and shadows as blue as well, so the problem still persists. Instead, we chose a solution that would first identify where the field was located in the image and subsequently detect obstacles lying on the field. Finally, the detected obstacles were assumed to be robots, as the only obstacles on the field during game play, besides the ball, are robots.

Pre-processing the segmented image The first pre-processing step is to reduce the image size of the segmented image. The original 208x160 segmented image is reduced by a factor of four, by essentially scanning across the image along grid lines that are parallel to the image horizon line and recording one out of every four pixels. By scanning along these grid lines instead of the image rows/columns, the algorithm is able to account for any rotation of the AIBOs camera. The image is reduced in order to simplify and speed up the task of searching for robots in the image. All further processing is performed on this reduced image. After size reduction, the image is scanned to find the field horizon. The field horizon is defined as the line that best separates the background from the soccer field in the original image, and that is parallel to the image horizon line. The field horizon is detected by scanning the columns of the reduced image looking for the start of a sequence of green pixels. The height of the highest point in the image that starts a sequence of green pixels denotes the location of the field horizon. After the field horizon has been detected, field lines are removed by scanning the image columns for small white regions surrounded by green. Field lines and the center circle are removed because they tend to interfere with the detection of obstacles on the field. Figure 2 shows an example of a reduced image and detected field horizon.

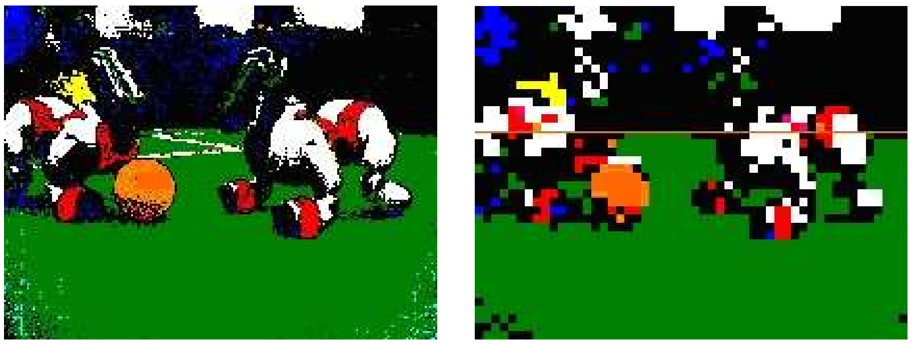

(a) The original segmented image (b) The reduced image with a detected field horizon

Fig. 2.

Detecting Obstacle Regions Using the assumption that robots are always on the field, we can search below the field horizon for pixels that are not green (not field) and not orange (not ball) and group them to form obstacles, which will eventually become our detected robots. Each column in the image is scanned below the field horizon searching for the start of a sequence of four green/orange pixels, at which point the scanned length along the column is recorded and scanning is initiated on the following column. This procedure essentially finds the length along each column of an obstacle that intersects the field horizon line. Columns with small lengths are thrown out. Neighboring columns with lengths a reasonable amount apart are grouped together. The lengths of all the columns within a group are averaged to find the length for that group. Using this length, along with the starting and ending column positions, a bounding box is created for each group; these bounding boxes represent the obstacles located on the field. Resulting bounding boxes that are too small or in some way inadequate are removed. The final bounding boxes are considered to encompass robots and represent the output of the robot detector. Example outputs are shown in Figure 3.

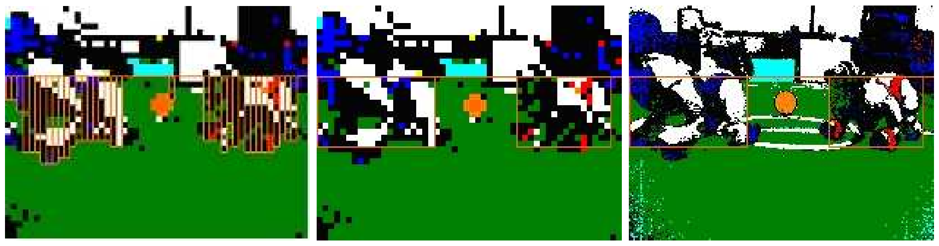

(a) Initial column scan results for obstacle colors (b) Final detected obstacles after grouping columns (c) Robot detection overlayed onto original image

Fig. 3.

Robot Color and Distance Estimates The team color is determined by searching for red regions within the bounding box of the detected robots. If convincing red regions are found within a robots bounding box, the robot is said to be on the red team; blue team otherwise. The distance to a detected robot is calculated by taking the center pixel of the bottom line of its bounding box and projecting a ray through it onto the ground plane. The distance to the intersection point is considered to be the distance to the robot. The robot detector is able to detect robots up to a maximum distance of approximately 1.2 m away. Results of the robot detector along with the predicted team colors are shown in Figure 4.

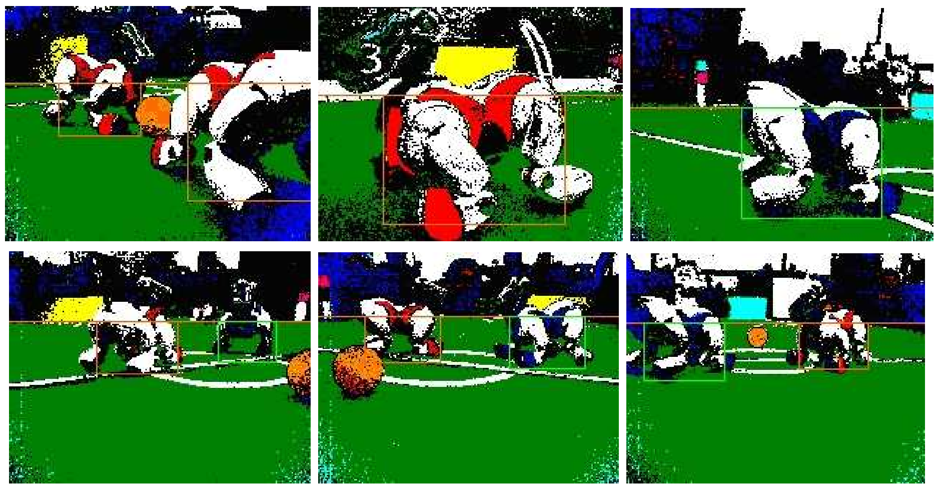


Fig. 4. A sequence of images showing robot detection results.

3 Localization

The CMDASH'05 team continues to use our Monte Carlo localization (MCL) system developed for use on the AIBOs to continually estimate their global location on the field. Our MCL algorithm uses a sensor-based resampling technique to re-localize the robots when they are lost called Sensor Resetting Localization [20]. This algorithm is robust to modelling errors including unmodelled movements, such as robots being pushed or removed for penalties, as well as for systematic errors. The field landmarks, goals, and field lines are used as features for the localization system.

4 World Modeling

To act intelligently in complex and dynamic environments, teams of mobile robots must estimate the position of objects by using information obtained from a wide variety of sources. Some examples of such sources include tracked object information from each robot's sensors, models of how each robot is able to affect the environment and manipulate those objects within it, and information obtained from teammates. For any problem of reasonable complexity, teams of mobile robots do not have the sensors necessary to perceive all aspects of a dynamic environment. As a result, the sources of information used in such a state estimator can be corrupted by noise that is extremely difficult if not impossible to fully model. In our research into robust teams of robots capable of operating within dynamic adversarial environments, we have found that not all sources of information about a single quantity of interest can be handled equally. To address this problem, we utilize a method for reasoning over a discontinuous hypothesis space where a strict ordering is imposed on the sources of information. By segmenting the information sources into different classes, a prioritized hierarchy of state estimates are inferred. Using this hierarchy, the decision process that governs each individual robot's actions can easily select the most informative state estimate to use as its input. A robot's actions can affect its perception of the environment as well as the environment. By reasoning about the expected utility of certain classes of estimates over others, the robot can select the "best" estimate from the set to act upon. This will provide more information to the robot that will in turn update the ordered hierarchy of possible estimates. This report describes our prioritized state estimation techniques as applied to a real-time adversarial multi-robot domain.

4.1 Related Work

Probabilistic state estimation techniques have been used very successfully for tracking applications for several decades [4]. In these approaches, the tracker possesses a model of the object so as to simulate its expected dynamics and predict its motion. Uncertainty and errors are modeled as noise generated by a random process. In the case of the Kalman filter [17], the state estimates and the noise are modeled as Gaussian distributions. Two limitations of this approach are the Gaussian assumption and the assumption of model linearity. This has been addressed in multiple ways, including changing the dimension of the state vector as the tracked target changes its perceived dynamics, such as with a variable state dimension (VSD) filter [3]. Our approach differs from the VSD filter in that while the number of hypotheses changes, the specific dimensions of those hypotheses estimates do not change. Additionally, each of the hypotheses is treated independently as a potential location for the object. Another multiple model technique is the interacting multiple models (IMM) filter [2] which uses a weighted mixture of different process models to handle complex system dynamics. Our approach uses independent classes of hypotheses that are not merged but are kept disjoint. A decision process selects the most relevant hypothesis based on specific external domain knowledge.

Another method of changing the dimensionality of the state vector based on sensor information is made use of by multiple hypothesis tracking, whereby multiple independent state estimators are used to estimate a multi-modal probability density. This approach has been used very successfully for challenging mobile robot localization problems [25] [16]. Hypotheses that represent possible robot poses are either created or destroyed dynamically as new sensor evidence becomes available which supports those operations. The hypotheses are updated and normalized to form a full probability distribution over all possible robot poses. Our approach extends the multi-hypothesis tracking paradigm by tagging the different hypotheses with the source of information that generated them segmenting them into different independent classes.

Recently, non-parametric distributions estimated through sampling techniques, such as the particle filter [1], have been used to great success. These approaches use stochastic resampling techniques to estimate completely arbitrary distributions at the cost of requiring greater computational resources. However, this too can be overcome by dynamically adjusting the number of particles as computational resources become available or are needed elsewhere [12]. Rao-Blackwellized particle filters [10] factor a joint state estimation problem into a more computationally efficient form. This approach has been used very successfully for tracking a target with a mobile robot [18] [13] where the actions of the robot change the process characteristics of the object. Our approach does not factor a joint state estimate, where one state variable, such as the location of the tracked object, is dependent upon another state variable, such as the location of the sensor. Rather, our approach is primarily concerned with explicitly reasoning about the sources of information that feed into a particular estimator. As such, our approach is complementary to the above approaches and could be integrated as part of a larger framework.

Simultaneously integrating multiple sources of information from potentially multiple sensors is a common sensor fusion problem [11]. Our approach can be considered an example of a competing fusion problem where multiple different sources of information provide equivalent information about the target, but the noise models from those sources of information can differ dramatically.

Additionally, a robot that can make use of information returned from its teammates has the potential to perform better than acting alone [14]. However, in the face of teammate information uncertainty caused by sensing, position, and sensor timestamp noise, explicitly merging teammate information can actually decrease the quality of the estimate. Explicit reasoning about the sources of information and factoring the space of estimates into ranked subgroups has proven to be an effective solution to this problem [24]. We explore and extend this line of research.

4.2 World Modeling Challenges in the RoboCup Domain

In the AIBO RoboCup legged league, two teams of four Sony AIBO robots autonomously play soccer against one another. While robots on the same team use 802.11b wireless Ethernet to communicate with each other, no additional off-board computation is allowed. The team thus forms a distributed sensor processing network. Several sources of information are available to each robot that allow it to generate a world model, including its sensors, communication, and kinematic models of their actuators. Additionally, each team possesses an *a priori* map of the field which gives the locations of the markers, goals, and field lines in a global reference frame. Visual observations of these environmental features are used to localize the robot on the field.

Using their wireless Ethernet, the robots can share local observations made about the environment with their teammates. Because the robots do not have a centralized compute server, they do not have a method of synchronizing their internal clocks. The lack of accurate timestamps on observations makes fusion of the sensor data much more challenging.

The four-legged chassis of the AIBO gives it a wide variety of motions that it can use to manipulate objects such as the ball. A large library of pre-defined kicking motions is available to each robot on the team. The robot is typically unable to visually track the ball during a kick because the ball is under the chin and behind the camera when the kick is initiated.

Reasoning about the world in the face of uncertainty is a challenging problem that is made more complex by the presence of another team on the field. The presence of opponents introduces a great deal of difficult or impossible to model noise that must still be acknowledged in algorithms for understanding the robot's sensor data. This paper describes our approach for modeling the different sources of information and our approaches for understanding how best to use them to generate a consistent and accurate model that allows the AIBO team to best complete their tasks. We will be focusing primarily on the estimation of tracked objects based upon multiple different sources of information.

4.3 Reasoning over Disjoint Hypotheses Classes

The challenge of understanding the real world in the face of uncertainty is very important in the field of mobile robotics. A mobile robot will have access to multiple different sources of environmental information; one for each different kind of sensor or modeling modality that they might have. Information from similar sensor modalities can easily be integrated as it has the same form. However, merging very different sensor modalities requires careful transformation and scaling. This becomes a challenge when some sources of information have noise models that are difficult or impossible to model properly. This is the problem that we address with this work.

A very powerful and general probabilistic algorithm to handling real-time sensor uncertainty is the Bayesian filter [15]. The Bayesian filter has been used very successfully in many robotic estimation algorithms [26]. A general description of a first-order feed-forward Bayesian filter can be shown as:

$$Bel_{-}(\hat{X}_t) = \int P(\hat{X}_t|u_{t-1}, \hat{X}_{t-1}) Bel(\hat{X}_{t-1}) d\hat{X}_{t-1} \tag{1}$$

$$Bel(\hat{X}_t) = \eta P(z_t|\hat{X}_t) Bel_(\hat{X}_t) \tag{2}$$

where $Bel_{-}(\hat{X}_t)$ is the belief of the robot's state vector at time t computed by integrating over the belief space at time $t-1$ after applying a model of system dynamics to the tracked object. When a sensor reading z_t is obtained, the belief state is updated converted and updated to $Bel(\hat{X}_t)$ with a likelihood distribution of the observation conditioned on the state vector. In this basic formulation, all sensor readings are uniformly merged into a single state density estimate where there is no history or record of how each sensor update may have contributed to the current estimate. Our approach to handling world modeling from multiple sensor inputs extends the basic Bayesian filter approach by tagging the contribution of each source of information to the state estimate. This allows additional information, such as the utility of the source of data on the estimate, to play a factor in the decision processes that the robot makes when solving its task. The space of disjoint hypothesis classes generated from the state estimate can be expressed as:

$$\hat{X}_t = \begin{bmatrix} [\hat{x}, \hat{y}, \hat{\dot{x}}, \hat{\dot{y}}]^{c1} \\ [\hat{x}, \hat{y}, \hat{\dot{x}}, \hat{\dot{y}}]^{c2} \\ \vdots \\ [\hat{x}, \hat{y}, \hat{\dot{x}}, \hat{\dot{y}}]^{cN} \end{bmatrix} \tag{3}$$

where each entry in this vector is an independent probabilistic state estimate that is tagged by the particular source of information that generated it. In this case, the superscripts on the individual hypotheses represent potentially distinct sensor classes. These disjoint estimates are merged or pruned by taking into account domain knowledge. A block diagram of this algorithm is illustrated in Figure 5.

World Modeling in a Multi-Agent Dynamic Adversarial Domain For efficiency purposes, the AIBOs use the Kalman filter to generate a probabilistic estimate for the position of the ball. As mentioned previously, a shortcoming of a basic Kalman filter algorithm is that it assumes that all of the noise models can be estimated using white Gaussian noise. The final state and uncertainty estimate are also represented as a single Gaussian distribution. An extension to this algorithm is the multiple hypothesis tracking Kalman filter by which a multi-modal probability density can be estimated from a bank of Kalman filters.

Our approach to multi-hypothesis state estimation using Kalman filters is to maintain a list L of Kalman filters which each independently represent a sensor reading about a tracked object. In this way, a multi-modal estimate generated by multiple potentially conflicting readings can be maintained until additional sensor information removes one

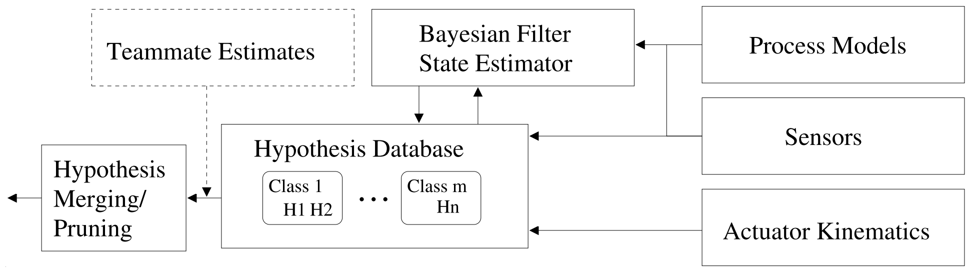


Fig. 5. Dataflow diagram for the multiple hypothesis space state estimator. Environment information from sensors, kinematic models of the robot's actuators and sensor poses, and a library of sensor models are used to support a database of probabilistic hypothesis about the state of the world. Different hypotheses can belong to different classes, depending on the source of information which created them. Each hypothesis is updated by a Bayesian filter and a post-processing decision process uses domain knowledge to merge relevant estimates and prune those that are of lesser value. Teammate information is also considered, but only at the post-processing phase.

Multi-hypothesis propagation algorithm

```
– Given:
    1. A list of Kalman filters $L$
    2. Sensor poses and expected fields of view
– Propagate
    • For each filter $l$ (with state estimate $\hat{x}_{t/t}$ and covariance matrix $P_{t/t}$) do
        ∗ Propagate the state and covariance matrix from time $t-1$ to time $t$
              $\hat{x}_{t+1/t} = F\hat{x}_{t/t} + Bu_t$
              $P_{t+1/t} = FP_{t/t}F^T + GQ_tG^T$
        ∗ Update the process noise matrix $Q$ of the filter based on the expected readings of the sensor
        ∗ If likelihood of $P_{t+1/t}$ is less than a threshold, delete filter $l$ from the list
```

Table 1. Multi-hypothesis state estimation propagation algorithm

or more hypotheses that are inconsistent with new sensor data. When new sensor data arrives, a gating function is used to determine which filter should be updated with the new information. If no hypothesis matches the data, a new hypothesis will be initialized. All hypotheses have an uncertainty model which is represented as a covariance matrix P. As per the propagation algorithm, the uncertainty of the covariance matrix will continuously grow if there is no sensor data. Eventually, a check is performed to determine whether the covariance of the estimate has grown too large to be practical. In

Multi-hypothesis sensor update algorithm

```
– Given:
    1. A list of Kalman filters $L$
    2. Reading from a sensor $z$
– **Update**
    • If Kalman filter list $L$ is empty, initialize a new filter
      based on the sensor reading and exit
    • else For each Kalman filter $l$ **do**
        ∗ Compute the Mahalanobis distance for the filter $l$
          and the sensor reading
```

$$\begin{aligned}\hat{z}_{t+1} &= H\hat{x}_{t+1/t}\\ r_{t+1} &= z_{t+1} - \hat{z}_{t+1}\\ S_{t+1} &= HP_{t+1/t}H^T + R\\ M &= r_{t+1}S_{t+1}^{-1}r_{t+1}^T\end{aligned}$$

```
    • Select the Kalman filter $l_i$ with the smallest Maha-
      lanobis distance $M_i$
    • If $M_i <= 3$, apply the sensor estimate to that filter
```

$$\begin{aligned}K_{t+1} &= P_{t+1/t}H^T S_{t+1}^{-1}\\ \hat{x}_{t+1/t+1} &= \hat{x}_{t+1/t} + K_{t+1}r_{t+1}\\ P_{t+1/t+1} &= P_{t+1/t} - P_{t+1/t}H_{t+1}^T S_{t+1}^{-1}H_{t+1}P_{t+1/t}\end{aligned}$$

```
    • else Initialize a new filter based on the sensor reading
      and add it to the list.
```

Table 2. Multi-hypothesis state estimation sensor update algorithm

this case, a particular filter is no longer informative (it is essentially a uniform density distribution) and is removed from consideration.

The sources of information that feed into this estimator can have distinctly process models. Our model allows the use of both positive and negative information returned from the sensors to adapt the process noise of the estimators. When the tracked object is observed by the sensors, the process noise is set to a model which describes the dynamics of that object. However, when the sensors are viewing an area where the tracked object is *expected*, but no readings are found, the process noise increases drastically to reflect the notion that the object has moved. This has the effect of rapidly growing the uncertainty of the hypothesis in question. The propagation algorithm for our disjoint multi-hypothesis tracker is shown Table 1, the sensor update algorithm in shown in Table 2, and the specific notation for these algorithms are described in Table 3.

$\hat{x}_{t/t}$	Estimated state at time t w/sensor readings at time t
F	State transition function
B	Control function
u_t	Control input at time t
$P_{t/t}$	Covariance matrix at time t w/sensor readings at time t
G	Noise input function
Q_t	Process noise covariance matrix at time t
z_t	Sensor reading at time t
$\hat{z}_t$	Estimated value of sensor reading at time t
H	Sensor function
r_t	Residual between expected and actual sensor readings
R	Sensor noise covariance
S_t	Computed covariance of sensor reading at time t
K_t	Kalman gain at time t

Table 3. Description of terms used in the multi-hypothesis Kalman filter algorithm

4.4 Action Selection Based on Multiple Hypotheses

In the AIBO RoboCup league, there are multiple sources of information that must be accounted for when tracking the ball. These include:

- **Game Manager :** When the ball goes out of bounds, it is immediately replaced in a fixed location depending on the offending team and the quadrant of the field where the ball went out.
- **Vision :** The robot's own camera is the most reliable source of information for the ball's position.
- **Actuation :** Kicks are performed blindly as the ball is usually under the robot's camera. Kinematic models are used to predict likely ball positions after manipulation.
- **Teammate :** Teammate ball information is typically the worst source because while their tracked local information is accurate with respect to their local reference frame, the global position can be erroneous if they are mis-localized.

The individual sources of information are used to generate a disjoint hypothesis space that is filtered for the most relevant information. The multi-hypothesis state estimator makes use of both positive and negative information when updating the process noise for the individual Kalman filters. Process noise is based on the following states and is ranked from lowest (1) to highest (5):

1. **Visible :** The estimate is being detected and tracked with the camera
2. **Possession :** The estimate is not seen, but the robot believes that the ball is under its chin and can be manipulated.
3. **Not in camera FOV :** The estimate is not updated with camera information because it is not within the expected field of view (FOV) of the camera.

4. **In camera FOV but occluded :** The estimate is expected to be visible in the calculated camera FOV but isn't. However, occluding objects (such as other robots) are visible in the image, so the object could be present but occluded.
5. **In camera FOV but not visible :** The estimate is expected to be visible in the calculated camera FOV but it isn't. No additional occluding objects are present.

The higher the process noise, the faster the region of uncertainty for the estimate will increase while there are no sensor updates. The greater the uncertainty in the estimate, the lower the likelihood that it will be chosen as the next hypothesis to explore by the robot's decision processes.

Hypothesis Ranking Algorithm The tracker makes use of multiple different sources of information. These are ranked in order of expected quality and used to guide the behaviors to search for the ball and track it when found. The decision process which merges/prunes the estimates encodes all of the relevant domain knowledge that is necessary for segmenting the hypothesis space into the relevant subsets. The algorithm for deciding which source of information to use in the hypotheses returned from the estimator is illustrated in Table 4.

At each timestep, a set of source-tagged hypotheses from the robot's own estimator as well as a set of source-tagged hypotheses from each of the teammate's estimators are examined. When the ball is actively visible to the robot, the estimate which best matches this data is used as the best estimate of the ball location and all others are ignored. If the ball is not actively visible, the individual hypotheses are filtered based on the sources of information used to generate them, how recent they are, and their relative confidences.

If the robot is not actively tracking visual information, but it believes that it is in possession of the ball, the active hypothesis class will be based on the belief that the ball moves with the robot. After a kick is completed, two hypotheses are created. The first is a prediction of where the kick would have placed the ball. The second is a failure case that places the ball near the robot's head. The first hypothesis has a lower uncertainty associated with it. The robot will focus on each hypothesis until it expires before going to the next one. If at any point the ball is seen, the search is complete.

If there are no ball hypotheses available from teammates, the robot will always use it's own ball hypothesis for as long as one exists. Similarly, if the robot does not have it's own ball hypothesis but has information from teammates then it will only rely on the teammate's information. If the robot has both it's own set of hypotheses and teammate hypotheses, and has to chose which estimate to use, the robot will trust it's own estimate up to a short period of time after losing visual contact with the ball due to the dynamic aspects of the environment. When multiple teammate hypotheses are present, the robot must additionally select which of them to use. This is dependent on the role that the robot is taking on the team. For example, the goalie robot chooses to rely on ball estimates reported by the defender over other players because of the greater importance associated with balls in the defense region of the field.

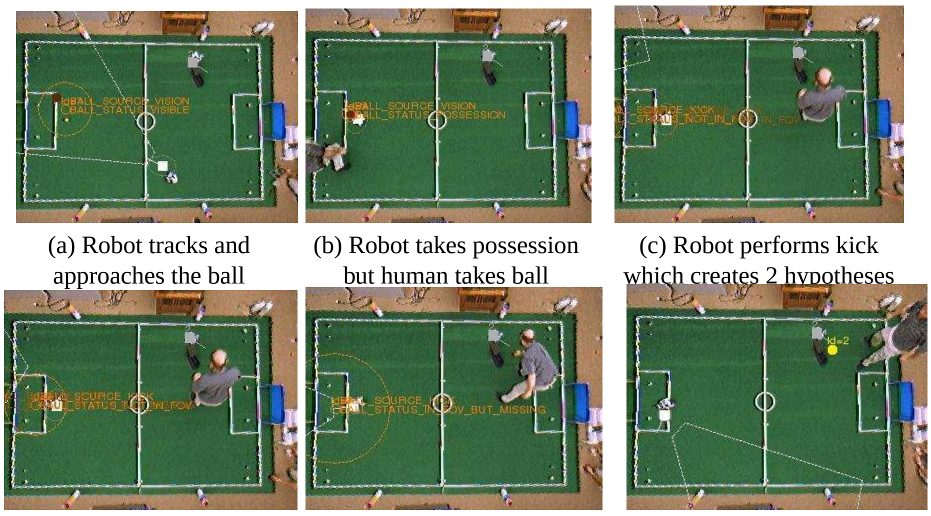
Hypothesis selection algorithm

```
– Given:
    1. Set of hypotheses based on the robot’s own sensors : $H_r$
    2. Set of hypotheses based on teammate sensors : $H_t$
– Select self or teammate information
    • If $H_r$ not empty and $H_t$ not empty
        ∗ If vision data actively supports a hypothesis in $H_r$ then select self-hypothesis
        ∗ else If time_since_ball_viewed $<$ threshold then select self-hypothesis
        ∗ else select teammate-hypothesis
    • else if $H_r$ not empty then select self-hypothesis
    • else if $H_t$ not empty then select teammate-hypothesis
    • else return
– Track hypothesis classes
    • If self-hypothesis
        ∗ If ball is actively in view of the camera, filter self estimates with source vision and actively track the most likely one
        ∗ else If game manager reports a throw-in, start a new tracker at the expected in-bounds point
        ∗ else If ball is in possession, track possession estimates
        ∗ else If ball was kicked, track the kick estimates
        ∗ else Track any estimates based on older vision information
    • else If teammate-hypothesis
        ∗ Rank the teammate estimates based on current self-role
        ∗ Track best ranked estimate based on teammate role and position
```

Table 4. Algorithm for filtering disjoint sets of hypotheses

4.5 Empirical Validation

Figure 6 illustrates how the multiple disjoint hypothesis tracking algorithm works with the AIBOs. A calibrated camera mounted over the soccer field is used to capture video of the AIBOs. The hypothesis state estimate generated by the robots is projected onto the video in these images to compare the estimates vs. ground truth. In Figure 6(a), an AIBO observes the ball on and moves towards it in an attempt to kick it into the goal. An anonymous human grabs the ball away from the AIBO just as it begins the kicking action in Figure 6(b). After the kicking motion, shown in Figure 6(c), two hypotheses



(a) Robot tracks and approaches the ball (b) Robot takes possession but human takes ball (c) Robot performs kick which creates 2 hypotheses

(d) Kick success hypothesis is removed (e) Ball is shown to stationary teammate (f) Kick failure hypothesis is removed. Teammate reports ball

Fig. 6. An illustrative example of the multi-hypothesis algorithm. These images show a top-down view of the soccer field superimposed with world model log information captured live from the robots. This demonstrates how the multiple hypothesis estimation algorithm directs the robot's actions. The robot attempts to kick the ball but the ball is moved by an anonymous human before the kick is completed. The robot tracks cycles through the available hypotheses before switching to listen to its teammate. The straight dashed lines show the robot's expected field of view of the field.

are generated. The first represents the predicted dynamics and location of the ball under the assumption that it was successfully kicked. The second represents the predicted dynamics and location of the ball under the assumption that the kick failed. In Figure 6(d), the AIBO's camera shows that the kick success hypothesis is not visible and so sets the process noise of that hypothesis much higher and that hypothesis rapidly decreases in likelihood and is removed. The AIBO switches to tracking the kick failure hypothesis at this point in Figure 6(e). This AIBO's camera also shows that this hypothesis has no ball associated with it and the covariance rapidly increases. Finally, in Figure 6(f), the hypothesis disappears as well and the robot switches to listening to its teammate's transmitted information.

4.6 Opponent Robot Tracking

Each visual observation of an opponent robot is stored for a maximum of 4 seconds after which time it is removed from consideration. Behaviors can query arbitrary rectangles in space for the existence of opponent robot observations. When a robot receives robot observations from its teammates, these observations are incorporated to the robot's own set of readings and are queried in the same fashion. Similarly, the observations are removed from consideration after four seconds.

5 Behaviors

The CMDASH'05 behavior subsystem has been completely replaced from previous years. In CMPack'04 and previous years, the behavior subsystem was written in C++. We make use of patches to Python 2.3.3 that were originally used by rUNSWift in the RoboCup 2004 competition. This year, the CMDASH'05 behavior system consists of the Python interpreter running directly on the robot and all of our behavior code has been completely rewritten in the Python language. Programming in python offers several important advantages.

- As an interpreted language, run-time errors do not crash the robot (as C++-based behaviors would), but rather provide informative debug information which can be captured in a log or read directly over the wireless Ethernet.
- Specifying what code is running on the dog at any point can easily be accomplished by FTPing the new Python files to the robot and restarting the interpreter remotely. This does not require rebooting the AIBO.
- The interpreter can be interacted with directly by using a remote python shell utility called *dogshell*. Parameters and variables can be changed dynamically without restarting the interpreter.

As in previous years, individual behaviors are written as finite state machines using a standard class. Each behavior is defined as a set of explicit states and state transitions and is able to monitor statistics such as the time spent in each state. This well formed framework leads to improved performance and debugging by providing methods to detect infinite loops, oscillation and other unwanted behaviors easily. Figure 7 shows an example state machine for a simple dribble behavior.

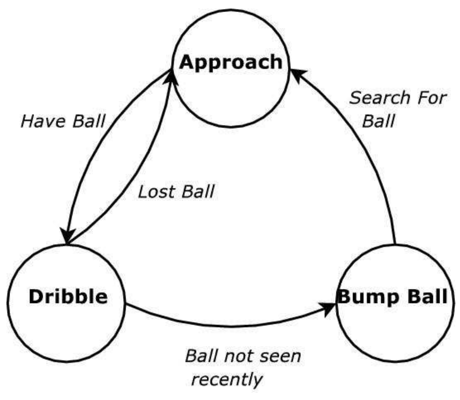


Fig. 7. Finite state machine for the dribble behavior.

5.1 Coordination and Teamwork

Over the past few years, teams have experimented with different methods of team coordination. Many of these strategies involve attempting to keep teammates from running into each other as well as placing teammates in good locations on the field so that they can be in good positions to receive passes or go after a free ball. While there have been many good approaches, no one strategy seems to have emerged as being clearly superior to all others. One reason for this is that several different coordination strategies are likely to be applicable in a single situation. Since some strategies may work better than others, a team that selects the superior strategy will be at an advantage. Thus, one of the most important problems to address when designing a multi-robot soccer team is in selecting the kind of coordination strategy that they will use during the game. Teams may choose to use a fixed coordination strategy defined *a priori*, but if chosen poorly, such a fixed strategy may not fare well against the strategy of the other team. Thus, an important extension to the research problem of coordination strategies is the ability for a team to dynamically change their strategy at runtime to adapt to their opponents' strengths and weaknesses.

Dynamically selecting a different strategy depending on the situation can be very powerful technique, but can be very challenging to implement well. Robots that use a dynamic coordination system must be able to perceive and properly evaluate the state of the world as well as the state of their own progress. This information is vital when making the decision to switch from a poorly performing strategy to one that could potentially work better.

We have identified several different levels for dynamic coordination that can be applied to a robotic team. These include:

- A "first-order" approach, where the robots use a fixed coordination strategy and each robot modifies the parameters of its behavior according to the world state.
- A "second-order" approach, where the robots have multiple ways of handling different situations. In order to utilize a second-order strategy, the robots must be able to evaluate the world state so that they can choose between the different behaviors they have at their disposal.
- A "third-order" approach, where the robots have several different team strategies, or "plays," which describe the coordinated actions of all of the robots together. Depending on the world state, different plays may apply; the team collectively decides upon the right behavior to apply in a given situation.

5.2 Changing Single Robot Parameters

We define the first-order coordination strategy as the ability for the robots to set their own behavior based on the state of the world. In this kind of system, each robot is programmed with a single behavior set which is used to control the robot's behavior in its environment.

We have tried two different methods for representing first-order coordination strategies. The first is a potential fields approach and the other is an approach that we call constraint-based positioning. In previous work [28], we give a detailed description of our implementation of potential field-based coordination. In this approach, we use potential fields both to determine the role that each robot plays (attacker, supporting attacker, and defender) and also to determine where the supporting robots position themselves on the field of play. On efficiency issue with potential fields occurs when they are used to coordinate the actions of a team of robots in a very dynamic world. In this situation, the fields may need to be recomputed for each every new sensor reading. This does not tend to be true for implementations of potential fields that are used for navigation in more static environments. In general, however, it's possible for minor disturbances in the positions or strengths of individual attraction and repulsion fields to cause fairly significant changes in the local gradient surrounding the robot.

Constraint-based positioning is an approach to robot positioning that we have developed in the CMPack'04 team for the 2004 RoboCup competition. Under this regime, robots are still assigned roles using a potential function, but the field positions chosen by the supporting robots are subject to a set of constraints. This approach was developed because there are several hard constraints that we would like to enforce on the robots' positions which are difficult to specify clearly with potential fields. For instance, defender robots need to avoid their own goalie's defense box, because entering the defense box is a violation which will cause the robot to be removed from play for 30 seconds. Other constraints that we would like to enforce include not crossing in front of a robot that is about to take a shot on goal, not coming within a certain minimum distance of a teammate, and so on. Consider a situation in which a robot is near the defense zone and a teammate is directly approaching it. Should the robot move toward the goal, violating the defense-zone constraint, or stand still, violating the teammate-distance constraint? Our implementation of constraint-based positioning allows us to prioritize the constraints, so that the robot knows that entering the defense zone is a more serious violation than coming near a teammate. In theory, the priorities of these constraints could be represented as a potential field, but we have found that debugging the complex potential fields that result can be difficult. If no constraints are in danger of being violated, the robot can choose to move to a specific point that is chosen based on the current state of the world. In this case, the robot can still use potential fields to choose an open area on the field or to choose a path to navigate around local obstacles.

Our experience with RoboCup has been that a single positioning function defined for a particular role tends to be too limiting. Trying to capture all of the possible actions that a robot might accomplish can cause the complexity of the positioning function to grow beyond what is manageable. A soccer-playing robot might have multiple ways of approaching the goal, each of which has advantages depending on the relative position of the goalie and/or his other players. In some situations, the robot may want to try one approach and if it fails, try a different approach. Behaviors like these may be mutually exclusive and as such could be very difficult for a single function to capture. Experimental validation of these methods can be found in [21].

6 Motion

The CMDASH'05 legged kinematics system makes use of a frame-based motion system and a parameterized walk engine [8]. This walk engine was originally developed for the ERS-210s and was enhanced and updated for the ERS-7s for the CMPack'04 team last year.

6.1 Kicking Motions

We have improved upon the kicking effectiveness of the CMPack'04 team with the addition of new kicks and dribbling motions. As with our team last year, the expected angle and distance for each kick is stored in a lookup table which is used by the behaviors to select an appropriate kick based on where the robot wants to place the ball.

6.2 Walk Learning

In order to obtain a fast walk, we have continued using the autonomous approach for optimizing fast forward gaits based on genetic algorithms [9] that was used in the CMPack'04 team. Our approach has proven to be very effective on the ERS-7 as it allows us to reach speeds of up to 40 cm/sec.

6.3 Odometry Calibration

New for this year is a simple but robust way for calibrating the robot's odometry. After learning a walk as described in section 6.2, its specific odometry parameters must be empirically determined in order to be used effectively by the behaviors. Our walk engine returns a theoretical maximum speed for the walk as well as odometric information. However, this assumes noise-free contact with the world which is impossible to achieve. Our odometry correction process consists of two steps: (1) computing the maximum speed of the walk, and (2) computing any correction factors for the surface on which the robot walks.

To calculate the maximum speed, the robot runs a simple warmup behavior which moves it in specific directions for a set amount of time at the maximum speed possible for that walk. The distance that the robot travels in that time is used to compute the true maximum speed. The same process is repeated for turning.

To calculate the odometric corrections, the robot is told to translate and specific distance as well as rotate about a specific angle. If the robot overshoots, then a correction factor is computed that allows it to better predict the distance that it travels or the angle that it turns. The computed parameters are used to calculate the commands that must be sent to the robot's motors in order to achieve the velocities requested by the behavior system.

Figure 8 presents an overview of this process. The behavior system outputs a vector (x, y, a) indicating the manner in which the robot's body should move. Acceleration bounding is then performed, and in the case of significant changes in velocities the values are bounded to the physical abilities of the robot. While the components of the vector do represent velocities, we have found that setting the motion velocities directly to these values often does not result in the desired trajectory. For example, when the motion command contains only a sideways component, we find that the robot tends to rotate significantly while strafing. This is due to the fact that the front legs are shorter

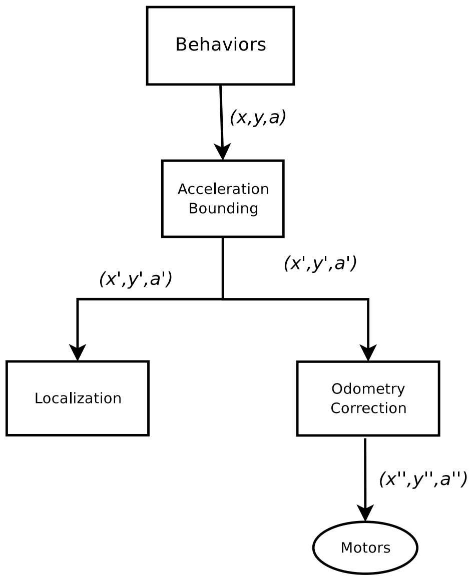


Fig. 8. Overview of the odometry pipeline.

than the rear when using the crouching walk. As a result, in order to generate pure sideways motion an angular velocity component must be added to counter the rotation. The odometry parameters are used to convert the velocity vector (x', y', a') into the motor commands that will result in the desired motion (x'', y'', a''). Note that the values used by the localization system for the odometry update are (x', y', a'), the motion requested by the behaviors, only checked for acceleration bounds.

7 Tools

7.1 Simulator

A new addition to our tools and utilities this year is a basic behavior simulator called *Puppysim*. Simulating the behavior of a robot, and especially a full team of robots, is a difficult and complicated task as a multitude of environmental factors have to be taken into account. Our simulator represents only the basic aspects of the environment and is used to test the robot's behavioral response under the ideal environmental conditions. Despite this limitation, we found the simulator to be extremely useful in the behavior development and debugging process, especially when dealing with algorithms for robot navigation or positioning. The Puppysim simulator was implemented using the Python programming language; it simulates only behaviors implemented in Python. The simulator's output is displayed via a GUI interface as shown in Figure 9.

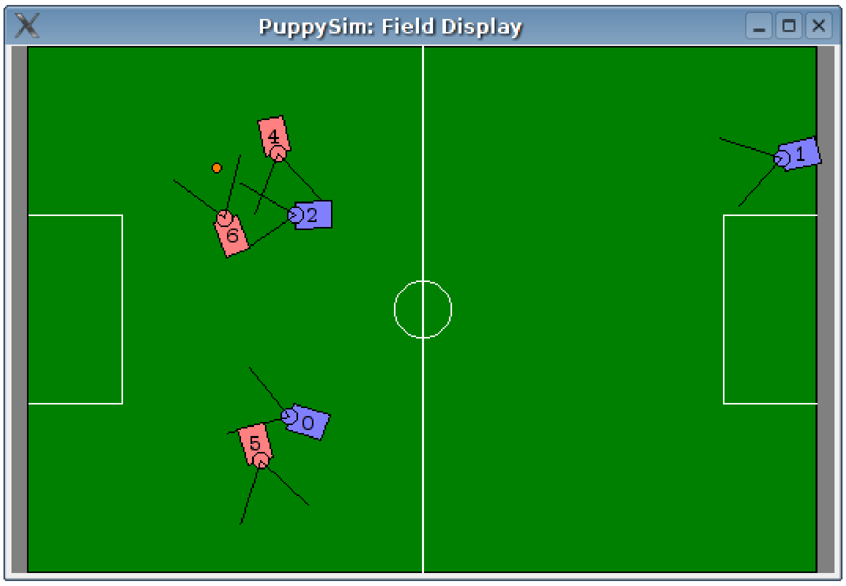


Fig. 9. Screenshot of the Puppysim behavior simulator running two teams of robots.

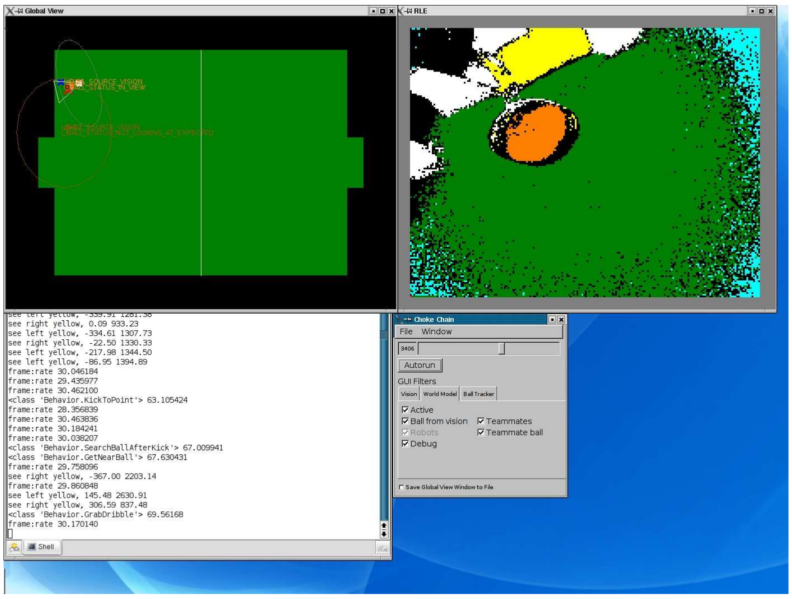
7.2 Logging and Debug GUI

Capturing sensor logs for analysis is an important part of the AIBO research process. The CMDASH'05 team is capable of logging a wide variety of sensor information and inferred state information. Some of the more important information includes:

- raw vision video frames
- color segmented video frames
- inferred positions and confidences of objects identified by high-level vision (ball, markers, goals, other robots)
- estimated positions of objects based on integrating multiple video frames (i.e. world model estimate of the ball position)
- robot's own position based on the MCL localization
- teammate positions and ball locations based on wireless communication
- general debug information

This information can be stored in a logfile that is saved to the memory stick after the robot's operation, as well as transmitted over the wireless Ethernet during operation. Our graphical visualization program, called *chokechain*, allows someone to view this data in real-time directly from the robot, or to step forward and backward through it if captured to a logfile. Figure 10 shows a sample view of the interface.

Calibrated Overhead Camera for Ground Truth Obtaining ground truth of the robot's position is extremely important for validating the accuracy of our algorithms.



```
see left yellow, -339.91 1281.38
see right yellow, 0.09 933.23
see left yellow, -334.61 1307.73
see right yellow, -22.50 1330.33
see left yellow, -217.98 1344.50
see left yellow, -86.95 1394.89
frame:rate 30.046184
frame:rate 29.435977
frame:rate 30.462100
<class 'Behavior.KickToPoint'> 63.105424
frame:rate 28.356839
frame:rate 30.463836
frame:rate 30.184241
frame:rate 30.038207
<class 'Behavior.SearchBallAfterKick'> 67.009941
<class 'Behavior.GetNearBall'> 67.630431
frame:rate 29.758096
see right yellow, -367.00 2203.14
frame:rate 29.860848
see left yellow, 145.48 2630.91
see right yellow, 306.59 837.48
<class 'Behavior.GrabDribble'> 69.56168
frame:rate 30.170140
```

[✓ Active]
[✓ Ball from vision]
[✓ Teammates]
[✓ Robots]
[✓ Teammate ball]
[✓ Debug]
☐ Save Global View Window to File

Fig. 10. The chokechain GUI illustrating a top-down world model view of the world in the upper left, the color-segmented image on the upper right, debug information shown in the terminal window in the lower left, and the window control panel with information filters in the lower right.

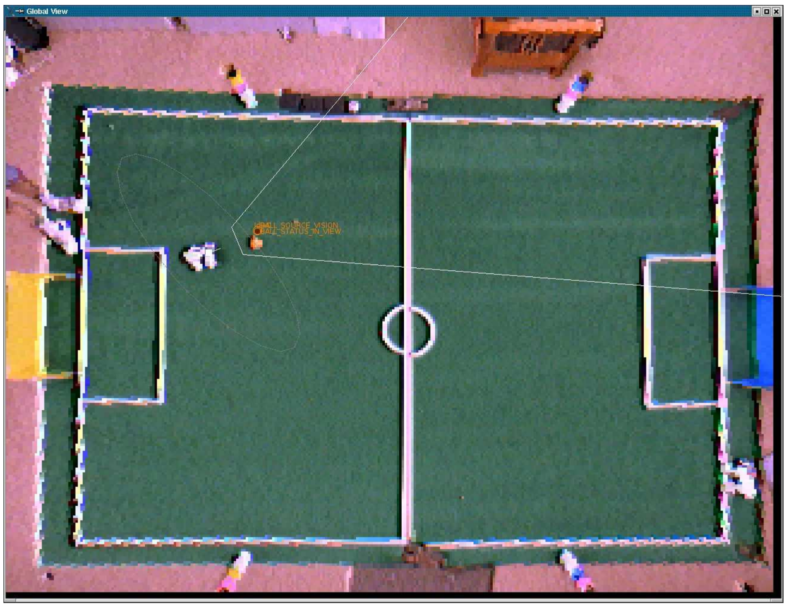


Fig. 11. Overhead camera view with world state information superimposed on it. This image shows the robot's position with uncertainty, the field of view of the camera, and an estimate of the ball's position in the image.

In particular, having the ability to compare the robot's estimated location on the field and its estimate of the ball to the real world is very useful. In order to facilitate this, we have mounted a camera over our field to obtain an overhead view of the robots. Radial distortion of the image caused by our camera's wide field of view is corrected using a Matlab camera calibration toolbox [5]. Once the intrinsic parameters of the camera are obtained, the position of the field in the camera images is obtained by having the user click on the screen to identify the outer boundaries of the field. Once aligned, any world model information generated by the robot can be overlayed on the corrected camera image in order to compare it against ground truth. Figure 11 shows the global view in *chokechain* but using the overlay on the live video feed.

8 Conclusion

With CMDASH'05 , we pursue our research on teams of intelligent robots. We continue to note that a team of robots needs to be built of skilled individual robots that can coordinate as a team in the presence of opponents. We have built upon our experiences from last year and enhanced our AIBO software with new visual features, a Python interpreter for behaviors with new behavior semantics, more extensive modeling and estimation of the world, better odometry calibration, and more advanced debugging and visualization tools.

Acknowledgements

This research has been partly supported by Grant No. DABT63-99-1-0013, by generous support from Sony, Inc., and by a National Physical Science Consortium Fellowship. The content of this publication does not necessarily reflect the position of the funding agencies and no official endorsement should be inferred.

References

1. Sanjeev Arulampalam, Simon Maskell, Neil Gordon, and Tim Clapp. A tutorial on particle filters for on-line non-linear / non-gaussian bayesian tracking. *IEEE Transactions on Signal Processing*, 50(2):174–188, 2002.
2. Y. Bar-Shalom, K. C. Chang, and H. A. P. Blom. Tracking a maneuvering target using input estimation vs. the interacting multiple model algorithm. *IEEE Transactions on Aerospace and Electronic Systems*, 25:296–300, March 1989.
3. Y. Bar-Shalom and K. Kirmiwal. Variable dimension filter for maneuvering target tracking. *IEEE Transactions on Aerospace and Electronic Systems*, 18(5):621–629, September 1982.
4. Y. Bar-Shalom, X.-R. Li, and Thiagalingam Kirubarajan. *Multitarget-Multisensor Tracking: Principles and Techniques*. Wiley-Interscience Publication, 2001.
5. J. Y. Bouguet. Camera calibration toolbox for matlab. Website: http://www.vision.caltech.edu/bouguetj/calib_doc/.
6. J. Bruce, T. Balch, and M. Veloso. CMVision, www.cs.cmu.edu/˜jbruce/cmvision/.
7. J. Bruce, T. Balch, and M. Veloso. Fast and inexpensive color image segmentation for interactive robots. In *Proceedings of IROS-2000*, 2000.

8. James Bruce, Scott Lenser, and Manuela Veloso. Fast parametric transitions for smooth quadrupedal motion. In A. Birk, S. Coradeschi, and S. Tadokoro, editors, *RoboCup-2001: The Fifth RoboCup Competitions and Conferences*. Springer Verlag, 2002.
9. Sonia Chernova and Manuela Veloso. An evolutionary approach to gait learning for four-legged robots. In *Proceedings of IROS 2004*, 2004.
10. A. Doucet, J.F.G. de Freitas, K. Murphy, and S. Russell. Rao-blackwellised particle filtering for dynamic bayesian networks. In *In Proceedings of the Conference on Uncertainty in Artificial Intelligence (UAI)*, 2000.
11. H. Durrant-Whyte. Sensor models and multisensor integration. *The International Journal of Robotics Research*, 7:97–113, December 1988.
12. Dieter Fox. Adapting the sample size in particle filters through kld-sampling. *International Journal of Robotics Research (IJRR)*, 22, 2003.
13. Yang Gu. Tactic-based motion modeling and multi-sensor tracking. In *In Proceedings of the American Association for Artificial Intelligence (AAAI)*, pages 1274–1279, Pittsburgh, PA, July 2005.
14. Jens-Steffen Gutmann, Wolfgang Hatzack, Immanuel Herrmann, Bernhard Nebel, Frank Rittinger, Augustinus Topor, Thilo Weigel, and Bruno Welsch. The CS freiburg robotic soccer team: Reliable self-localization, multirobot sensor integration, and basic soccer skills. *Lecture Notes in Computer Science*, 1604, 1999.
15. A.M. Jazwinsky. *Stochastic Processes and Filtering Theory*. Academic, New York, 1970.
16. Patric Jensfelt and Steen Kristensen. Active global localization for a mobile robot using multiple hypothesis tracking. *IEEE Transactions on Robotics and Automation*, 17(15):748–760, October 2001.
17. R. E. Kalman and R. Bucy. New results in linear filtering and prediction theory. *Transactions of ASME, Journal of Basic Engineering*, 83:95–108, 1961.
18. Cody Kwok and Dieter Fox. Map-based multiple model tracking of a moving object. In *In Proceedings of the RoboCup 2004 Symposium*, Lisbon, Portugal, July 2004.
19. S. Lenser, J. Bruce, and M. Veloso. CMPack: A complete software system for autonomous legged soccer robots. In *Autonomous Agents*, 2001.
20. S. Lenser and M. Veloso. Sensor resetting localization for poorly modelled mobile robots. In *Proceedings of ICRA-2000*, 2000.
21. Colin McMillen, Paul Rybski, and Manuela Veloso. Levels of multi-robot coordination for dynamic environments. In *Multi-Robot Systems Workshop*, 2005.
22. W. Press, B. Flannery, S. Teukolsky, and W. Vetterling. *Numerical Recipes in C, the Art of Scientific Programming*. Cambridge University Press, second edition, 1993.
23. T. Röfer, T. Laue, H-D. Burkhard, J. Hoffmann, M. Jüngel, D. Göhring, M. Lötzsch, U. Düffert, M. Spranger, . B. Altmeyer, V. Goetzke, O. v. Stryk, R. Brunn, M. Dassler, M. Kunz, M. Risler, M. Stelzer, D. Thomas, S. Uhrig, U. Schwiegelshohn, I. Dahm, M. Hebbel, W, Nisticó, C. Schumann, and M. Wacher. GermanTeam 2004. Technical report, 2004.
24. Maayan Roth, Douglas Vail, and Manuela Veloso. A real-time world model for multi-robot teams with high-latency communication. In *In Proceedings of the 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2494–2499, Las Vegas, NV, October 2004.
25. Stergios I. Roumeliotis and George Bekey. Bayesian estimation and kalman filtering: A unified framework for mobile robot localization. In *In Proceedings of the International Conference on Robotics and Automation (ICRA'00)*, San Francisco, CA, April 2000.
26. Sebastian Thrun. Robotic mapping: A survey. In G. Lakemeyer and B. Nebel, editors, *Exploring Artificial Intelligence in the New Millenium*. Morgan Kaufmann, 2002.

27. William Uther, Scott Lenser, James Bruce, Martin Hock, and Manuela Veloso. CM-Pack'01: Fast legged robot walking, robust localization, and team behaviors. In A. Birk, S. Coradeschi, and S. Tadokoro, editors, *RoboCup-2001: The Fifth RoboCup Competitions and Conferences*. Springer Verlag, Berlin, 2002.
28. Douglas Vail and Manuela Veloso. Dynamic multi-robot coordination. In *Multi-Robot Systems: From Swarms to Intelligent Automata, Volume II*, pages 87–100. Kluwer Academic Publishers, 2003.
29. M. Veloso and W. Uther. The CMTrio-98 Sony legged robot team. In M. Asada and H. Kitano, editors, *RoboCup-98: Robot Soccer World Cup II*, pages 491–497. Springer, 1999.
30. M. Veloso, E. Winner, S. Lenser, J. Bruce, and T. Balch. Vision-servoed localization and behavior-based planning for an autonomous quadruped legged robot. In *Proceeding of the 2000 International Conference on Artificial Intelligence Planning Systems (AIPS'00)*, 2000.
31. Felix von Hundelshausen. *Computer Vision for Autonomous Mobile Robots*. PhD thesis, Department of Computer Science, Free University of Berlin, Takustr. 9, 14195 Berlin, Germany, September 2004. http://www.diss.fu-berlin.de/2004/243/indexe.html.
32. Felix von Hundelshausen and R. Rojas. Tracking regions. In *Proceedings of the RoboCup 2003 International Symposium, Padova, Italy*, 2003.