Vision-Servoed Localization and Behaviors for an Autonomous Quadruped Legged Robot

Manuela M. Veloso, Elly Winner, Scott Lenser, James Bruce, and Tucker Balch
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213-3891
{mmv}@cs.cmu.edu

Abstract

Planning for real robots to act in dynamic and uncertain environments is a challenging problem. A complete model of the world is not viable and an integration of deliberation and behavior-based reactive planning is most appropriate for goal achievement and uncertainty handling. This paper reports on our successful development of the integration of perception, planning, and action for the Sony quadruped legged robots. We address the particular robotic soccer task, as Sony provided the robots to us specifically for the RoboCup robotic soccer competitions. The quadruped legged robots are fully autonomous, so must have onboard vision, localization and agent behavior. We briefly present our perception algorithm that does automated color classification and tracks colored blobs in real time. We then briefly introduce our localization algorithm called Sensor Resetting Localization (SRL) which is an extension of Monte Carlo Localization. SRL is robust to movement modelling errors and to limited computational power. Vision and localization provide the state input for action selection. The planning challenge we addressed resulted in a robust and sensible behavior scheme for the robot that effectively handles dynamic changes in the accuracy of the perceived information. We developed a two-constraint system for utility-based thresholded localization. One constraint specifies how much time the robot must spend acting, and the other constraint specifies how good its localization information must be before the robot uses it. Finally, we have devised several special built-in plans to deal with times when urgent action is needed and the robot cannot afford collecting accurate perception information. We present results using the real robots demonstrating the success of the algorithms. Our team of Sony quadruped legged robots, CMTrio-99, won all but one of its games in RoboCup-99, and was awarded third place in the competition.

Introduction

The robots used in this research were generously provided by Sony (?). The robots are an excellent platform for research and development and provide us exceptional hardware that is commercially viable. The robot consists of a quadraped designed to look like a small dog. The robot is approximately 20cm long plus another 10cm due to the head. The neck and four legs each have 3 degrees of freedom. The neck can pan almost 90° to each side, allowing the robot to scan around the field for markers. The robot stands 20cm high at the shoulder with the head extending another 10cm. Figure ?? shows a picture of the dog pushing a soccer ball.

Figure 1: The Sony quadruped robot dog with a ball.

1999 was the second year the Sony legged robot league was a part of RoboCup (?). Although the Sony legged robot league in RoboCup-98 used a different hardware platform (?), the challenges we faced in RoboCup-99 were the same. Our team benefitted greatly from the experience of the three previous teams (?). When we began work, we already knew about some of the most difficult problems we would face, and how previous teams had solved them.

All teams in the RoboCup-99 legged robot league used the same hardware platform: the Sony quadruped legged robots (?). The robots are autonomous, and have onboard cameras. The onboard processor provides image processing, localization and control. The robots are not remotely controlled in any way, and as of now, no communication is possible in this multi-robot system. The only information available for decision making comes from the robot's onboard camera and from sensors which report on the state of the robot's body.

The Sony AIBO robots were commercially sold in March of 1999. But the robots used for the RoboCup-99 competition are not the same as those commercially available. The robots we used are equipped with slightly different hardware, and, unlike the commercial product, are programmable.[1]

The soccer game consists of two ten-minute halves, each begun with a kickoff. In the kickoff, the ball begins in the center of the field, and each team may position its robots on its own side of the field. After each goal, play resumes with another kickoff.

Each team consists of three robots. Like our team last year, CMTrio-98, and most of the other 1999 teams, we subdivided our team into two attackers and one goaltender. In this paper, we will focus on the attackers, but will discuss the goaltender in Section ??.

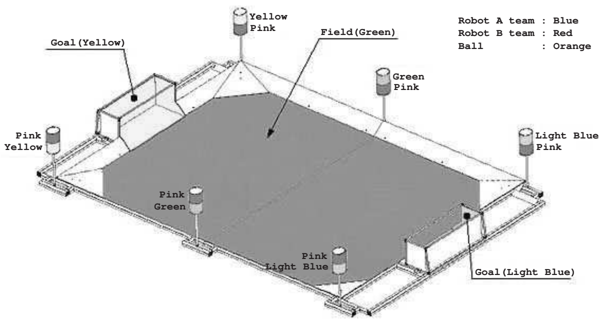

Figure 2: The playing field for the RoboCup-99 Sony legged robot league.

The field is 280cm in length and 180cm in width. The goals are centered on either end of the field, and are each 60cm wide and 30cm tall. Six unique landmarks are placed around the edges of the field (one at each corner, and one on each side of the halfway line) to help the robots localize themselves on the field. The landmarks are 20cm off the ground and 20cm high. Figure ?? shows a sketch of the playing field.

Vision

The vision system processes images captured by the robot's camera to report the locations of various objects of interest relative to the robot's current location. These include the ball, 6 unique location markers, two goals, teammates, and opponents.

1. **Image capture/classification:** images are captured in YUV color space, and each pixel is classified in hardware by predetermined color thresholds for up to 8 colors.

2. **Region segmenting:** pixels of each color are grouped together into connected regions.

3. **Region merging:** similarly colored regions are merged together based on satisfaction of a minimum density for the merged region set for each color.

4. **Object filtering:** false positives are filtered out via specific gemoetric object filters for each type of object, and a confidence value is calculated for each detected object.

5. **Distance and transformation:** the angle and distance to detected objects are calculated relative to the image plane, and then mapped into an ego-centric coordinate system relative to the robot's body.

[1] Our extensive videos of this event provide invaluable illustrative support to the work presented in this paper.

The onboard camera provides 88x60 images in the YUV color space at about 15Hz. These are passed through a hardware color classifier to perform color classification in real-time based on previously set thresholds.

When captured by the camera, each pixel's color is described as a 3-tuple in YUV space, where each component's value can vary from 0 to 255. The color classifier then determines which color classes the pixel is a member of, based on a rectangular threshold for each class in the two chromiance dimensions (U,V). These thresholds can be set independently for every 8 values of intesity (Y). For a particular pixel, the value of Y determines which set of rectangular thresholds to access, and each of the rectangles that the (U,V) point is within represents a color which that pixel is a member of. Thus for each color, a 3-dimensional space composed of 8 rectilinear volumes defines which pixel values are a member of that color class. An example of the results of this classification is provided in Figure ??.

The final result of the color classification is a new image indicating color class membership rather than the raw captured camera colors. The 88x60 image has bits set for whitch classes, if any, a particular pixel is a member of. This is the input for the next step in the system, in which the connected regions of a particular color are determined.

Multiple color region merging can be an expensive operation which severly impacts real-time performance if key optimiations are not made. The approach taken here involves two separate stages which address these concerns.

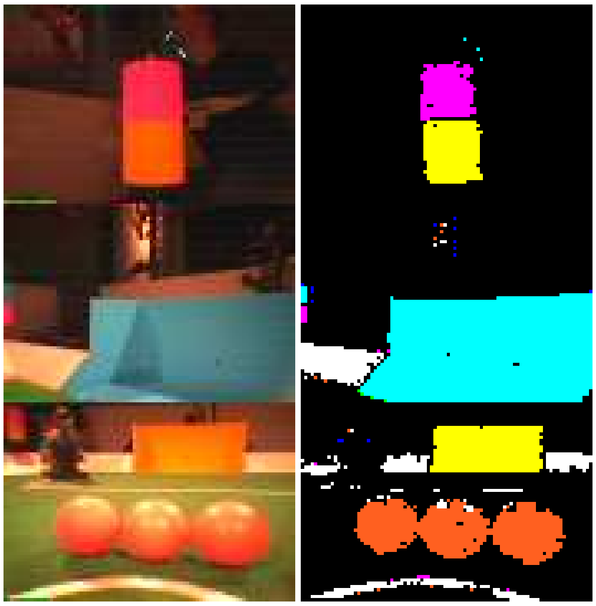

Figure 3: An example image classified using the approach presented in the paper. The image on the left is a composite of objects tracked by a soccer robot at RoboCup-99: a position marker (top), a goal area (middle) and three soccer balls (bottom). The classified image is on the right.

The first stage is to compute a run length encoded (RLE) version for the classified image. In machine vision applications, significant changes in adjacent image pixels are relatively infrequent, thus grouping similar values of the color classified pixels results in a substantial decrease in storage and processing requirements for subsequent steps. This is because the processing routines can operate on entire runs at a time, rather than individual pixels. There is also the practical benefit that region merging need now only look for vertical connectivity, because horizontally connected components have already been merged in the transformation to the RLE representation of the image.

The region merging method employs a tree-based *union find* with path compression. This offers performance that is not only good in practice but also provides a hard algorithmic bound that is for all practical purposes linear (?). The merging is run in place on the classified RLE image, thanks to a field included in each run indicating its parent in a union tree. Initially, each run labels itself as its parent, resulting in a completely disjoint forest. The merging procedure scans adjacent rows and merges runs which are of the same color class and are neighbors under four-connectedness. This results in a disjoint forest where the each run's parent pointer points upward toward the region's global parent, which in this implementation is the upper-leftmost member of the connected region. A final pass is made to compress all of the paths in the tree so that each run's parent field is pointing directly to the global parent. Now each set of runs pointing to a single parent uniquely identifies a connected region, so the parent field can be thought of as a label which is unique to each region. The process is illustrated in Figure ??).

In the next step we extract region information from the merged RLE map. The bounding box, centroid, and size of each region is calculated incrementally in a single pass over the forest data structure. During the pass, the region labels are renumbered so that they immeidately follow each other in a region table, which facilitates fast lookup for the calculation of the region statistics. This process could eaily be extended to extract additional statistics, such as a convex hull or edge points list. The information extracted, though relatively simple in this case, provided enough information for the higher level manipulations.

After the statistics have been calculated, the regions are separated by color into separate threaded linked lists in the region table. Finally, they are sorted by size so that later processing ateps can deal with the larger (and presumably more important) blobs, and ignore relatively smaller ones which are most often the result of noise.

The next step attempts to deal with one of the shortcommings of object detection via connected color regions. Due to partial occlusion, specular highlights, or shadows, it is often the case that a single object is broken into a few separate but nearby regions. A single row of pixels not in the same color class as the rest of the object is enough to break connectivity, even though the object may occupy many rows. In order to correct for cases when nearby regions are not connected but should be considered so, a density based merging scheme was employed. Density is represented here as the ratio of the number of pixels of the color class in the connected region to the overall area of its bounding box. By this measurement heuristic, two regions that have a small separation relative to their sizes will likely be merged, since they would tend to have relatively high density.

For each color class, a minimum densitry of merged regions is specified, as well as the expected maximum number of objects of that color. If the observed regions exceed the expected number of objects, pairs of regions are compared, starting with the largest two first. If a pair of regions could be merged while still meeting the minimum density, the two regions are combined.

Localization

Our localization algorithm is based upon a classical Bayesian approach which updates the location of the robot in two stages, one for incorporating robot movements and one for incorporating sensor readings. This approach represents the location of the robot as a probability density over possible positions of the robot. In the CMTrio-98 localization algorithm, this probability density is represented using a grid based division of the pose space (?). Our localization algorithm, called Sensor Resetting Localization (SRL), is based upon a popular approach called Monte Carlo Localization (MCL). Both SRL and MCL represent the probability density using a sampling approach.

Monte Carlo Localization(MCL) (?) represents the probability density for the location of the robot as a set of discrete samples. Since each sample is an exact point in locale space, updates of samples are easy to implement. The density of samples within an area is proportional to the probability that the robot is in that area. Since the points are not distributed evenly across the entire locale space, MCL focusses computational resources where they are most needed to increase the resolution near the believed location of the robot. The position of the robot is calculated from these samples by taking their mean or some variant of mode. The uncertainty in this estimate can be estimated by calculating the standard deviation of the samples. There are two kinds of update operations that are applied to the sample set, movement updates and sensor updates.

For a more complete description of Monte Carlo Localization, please see Fox et al. (?) and Dellaert et al. (?).

We encountered some problems implementing MCL for the robot dogs. MCL took too many samples to do global localization. This resulted in poor localization results when the robot initially did not know its location. With the number of samples we could actually run on the hardware, the samples were too spread out to localize the robot correctly. SRL gets around this by resampling from the sensor readings to focus the samples were they are needed during global localization.

MCL could not handle the large systematic errors in movement we experienced. Every sensor reading gives MCL a chance to correct a small amount of systematic error. The amount of systematic error that can be corrected for increases with larger movement deviations and larger numbers of samples. If the systematic error in movement gets too large, MCL will slowly accumulate more and more error. We need to handle systematic errors in movement because measuring the movement parameters for a robot is time consuming and difficult. Systematic errors in movement also occur when the environment changes in unmodelled ways. For example, if the robot moves from carpeting to a plastic surface such as the goal, the movement of the robot for the same motion commands is likely to change.

MCL does not handle unexpected/unmodelled robot movements very well. The time MCL takes to recover is proportional to the magnitude of the unexpected movement. During this time, MCL reports incorrect locations. Unexpected movements happen frequently in the robotic soccer domain we are working in. Collisions with other robots and the walls result in motions having unexpected results. Collisions are difficult to detect on our robots and thus cannot be modelled by the localization algorithm. Another unexpected movement we incur is teleportation due to application of the rules by the referee. MCL takes a long time to recover from this.

Some of the drawbacks of MCL can be alleviated by adding adaptive sample set sizing. Even with adaptive sample set sizing, MCL is still more sensitive to systematic errors in movement than SRL. MCL with adaptive sample set sizing requires different computational resources as the number of samples changes. Adaptive sample set sizing as described in Fox et al. (?) can take an extremely long time if the robot thinks it is one position and the sensor readings indicate a different position, especially if robot movements are very accurate. We were unable to apply MCL with adaptive sample set sizing since we are working in a real time domain and do not have any additional computational power available.

We used an extension of Monte Carlo Localization called Sensor Resetting Localization(SRL). SRL is motivated by the desire to use fewer samples, handle larger errors in modelling, and handle unmodelled movements. SRL adds a new step to the sensor update phase of the algorithm. If the probability of the locale designated by the samples we have is low given the sensor readings $P(L|s)$, we replace some samples with samples drawn from the probability density given by the sensors $P(l|s)$. We call this sensor based resampling. The logic behind this step is that the average probability of a locale sample is approximately proportional to the probability that the locale sample set covers the actual location of the robot, i.e. the probability that we are where we think we are. Please see the paper on Sensor Resetting Localization for more details (?).

Movement update.
$P(l^{j+1}|m, l^j) = P(l^j)$ **convolved** $P(l'|m, l)$
This stage is the same as Monte Carlo Local-

ization

1. **foreach sample** s **in** $P(l^j)$
   2. **draw sample** s' **from** $P(l'|m,s)$
   3. **replace** s **with** s'

Sensor update.
$P(l^{j+1}|s,l^j) = P(l^j) * P(l|s)/\alpha$ **where** α **is a constant.**
Steps 1-?? of this stage are the same as Monte Carlo Localization

1. **[optional step] replace some samples from** $P(l^j)$ **with random samples**
2. **foreach sample** s **in** $P(l^j)$
   3. **set weight of sample equal to probability of sensor reading,** $w = P(l|s)$
4. **foreach sample** s **in** $P(l^j)$
   5. **calculate and store the cumulative weight of all samples below the current sample** (cw(s))
6. **calculate total weight of all samples (tw)**
7. **foreach sample** s' **desired in** $P(l^{j+1})$
   8. **generate a random number(**r**) between 0 and tw**
   9. **using a binary search, find the sample with maximum** $\mathrm{cw(s)} < \mathrm{r}$
   10. **add the sample found to** $P(l^{j+1})$
11. **calculate number of new samples,** $\mathrm{ns} = (1 - \mathrm{avg_s ample_p rob/prob_t hreshold}) * \mathrm{num_s amples}$
12. **if**$(\mathrm{ns} > 0)$ **repeat ns times**
    13. **draw sample(**s'**) from** $P(l|s)$
    14. **replace sample from** $P(l^{j+1})$ **with** s'

We modelled all of our movements by 3 Gaussians, one for movement distance, one for movement direction, and one for heading change. We represented our sensor distributions as 2 Gaussians, one for distance from the landmark and one for egocentric angle to the landmark. We set deviation equal to a percentage of the mean for distance and a constant value for angle.

Sensor Resetting Localization is applicable in domains where it is possible to sample from the sensor readings $P(l|s)$. This is not a problem if landmarks are being used as the sensor readings as the sensor distributions are easy to sample from. If all possible locations of the robot are known, this sensor based sampling can be done by rejection sampling. However, rejection sampling increases the run time for resampling in proportion to the probability of having to reject a sample.

One of the advantages of SRL is that fewer samples can be used without sacrificing much accuracy. This is possible in part because it is more efficient when globally localizing. When the first marker is seen during global localization, the probability of almost all of the samples is very low. Thus the average probability of a sample is ridiculously small and SRL replaces almost all the locale samples with samples from the sensors. This results in all of the samples being distributed evenly around the circle determined by the marker. So, if we are using 400 samples, we have 400 samples instead of the 1-2 of MCL to represent the circle around the marker. Naturally, this reduces misleading errors during global localization. This also reduces the time required to converge to the correct localization since the correct answer has not been thrown out prematurely. After seeing another marker the circle collapses to a small area where the circles intersect. The average probability of the locale samples now is much higher than after seeing the first marker since more samples have been concentrated in the right place by the first sensor reading. Thus, if the threshold for sensor based resampling is set correctly, no new samples will be drawn due to the second sensor readings. As long as tracking is working, no new samples are generated from the sensors and the algorithm behaves exactly like MCL.

SRL can handle larger systematic errors in movement because once enough error has accumulated, SRL will replace the current estimate of the robot's location with one based on the sensor readings, effectively resetting the localization. Adaptive sample set sizing helps MCL, but MCL is still more sensitive to systematic errors in movement and unexpected/unmodelled robot movements. SRL is also easier to apply to real time domains since the running time per step is nearly constant and easy to bound.

SRL can handle larger unmodelled movements than MCL. The localization algorithm needs to handle extended collisions with other robots and the wall gracefully. SRL does this by resetting itself if its estimate of current robot position gets too far off from the sensor readings. SRL is able to handle large unmodelled movements such as movement by the referee easily. SRL does this by resetting itself the first time it gets a sensor reading that conflicts with its estimated position. MCL would take a long time to correct for long distance teleportation such as this since enough error in movement has to occur to move the mean of the samples to the new location.

Localization Capabilities

We tested Sensor Resetting Localization on the robots provided by Sony. We also did extensive tests simulation to compare Sensor Resetting Localization with Monte Carlo Localization with and without random noise samples added (?).

We tested SRL on the real robots using the parameters we used at RoboCup '99. We used 400 samples for all tests. In order to execute in real time, we were forced to ignore about 50% of the sensor readings. Due to inevitable changes in conditions between measuring model parameters and using them, the parameter for distance moved was off $\approx 25\%$, for angle of movement $\approx 10°$, and for amount of rotation $\approx .6°$/step. The deviations reported to the localization were 10% for movement and 15% for vision. We had the test robot run through a set trajectory of 156 steps while slowly turning it neck from side to side. The robot was instructed to stop after 7 different numbers of steps had been executed. The final position of the robot was measured by hand for each run. We did five runs at each of the 7 number of steps and averaged the results. We compared the actual location of the robot at each of the 35 data points with the location reported by the localization algorithm. We calculated the error in the mean position over time and the deviation the localization reported over time. We also calculated an interval in each dimension by taking the mean reported by the localization and adding/subtracting 2 standard deviations as reported by the localization. We then calculated the distance from this interval in each dimension which we refer to as interval error. We report both average interval error and root mean squared interval error. We feel that root mean squared interval is a more appropriate measure since it weights larger, more misleading errors more heavily. We also calculated the percentage of time that the actual location of the robot fell within the 3D box defined by the x,y, and θ intervals.

The table below shows the localization is accurate within about 10cm in x and y and 15° in θ despite the erroneous parameter values. The actual location of the robot is within the box most of the time and when it is outside the box, it is close to the box. The fact that the localization seldom gives misleading information is very important for making effective behaviors. The error in position and the deviation reported quickly converges to a steady level. The deviation tends to go up at the same time the error goes up which keeps the interval error low and avoids misleading output. In competition, we observed that the localization algorithm quickly resets itself when unmodelled errors such as being picked up occur. The actual performance of the localization during play tends to be worse since the robot spends much less time looking at the markers.

	x (mm)	y (mm)	theta (°)
average error	99.94	95.14	14.29
average interval error	15.18	4.91	2.07
rms interval error	34.92	13.94	3.82
in box percentage	74.29%	80.00%	57.14%

Behavior-Based Planning

The behavior of the robot is an especially difficult problem in this domain, in which the robot acts under uncertainty and must be able to quickly and gracefully improve and degrade its performance as the availability of localization information changes.

Because the localization system is reliant on visual identification of landmarks, in order to keep its localization information up-to-date, the robot must scan for landmarks. As the robot walks, the camera experiences pitch and roll, which causes the images it collects to change significantly from one frame to the next. Because of this, the vision system's identification of objects and the estimate of their distances and angles degrades. The localization system depends heavily on very accurate information about the landmarks, so our algorithm requires that the robot stop moving while looking for landmarks. The process of stopping and scanning for landmarks usually takes the robot between 15 and 20 seconds.

Because the vision system is reliable, we assumed, in designing our behavior algorithms, that the information it provides is correct.

Our approach provides the robot with the ability to control its knowledge of the world: in order to learn more about where it is, it can spend more time looking for landmarks. Although having more information helps the robot tremendously, soccer, like other dynamic domains, is time-critical, so every moment spent looking around is lost time. Opponents can use also the robot's inattention to their advantage.

Our strategy includes: i) a two-constraint system for utility-based thresholded localization, ii) an architecture that allows the robot to upgrade and degrade its performace quickly and gracefully, iii) behaviors that are reasonable even when the robot does not know where it is, and iv) several special localization-dependent behaviors which dramatically increase the robots' efficiency.

Control over State Knowledge

Robotic soccer is a time-critical game, so it is undesirable for the robot to stop and look around when it is not necessary to do so—not only because this wastes time, but also because when the robot isn't paying attention to the game, its opponents have a chance to act unimpeded.

One possible localization strategy, used by the by this year's team from from LRP University in France (?), involves localizing the robot very infrequently, if at all. However, the benefits of accurate localization are significant.

We present a scheme that balances the time required to get accurate localization information with the benefits this information provides.

Utility-Based Thresholded Localization We found that if we defined what a "useful" amount of localization information was and allowed the robot to stop and look around every time its localization information fell below that level, the robot spent most of its time looking for landmarks. However, if we changed the definition of "useful" information to the point that the robot was able to act for a reasonable amount of time before stopping to look for landmarks, then its localization information was almost never good enough to use; in our tests, the robot frequently scored own-goals, or tried to push the ball into the walls of the field.

We use a system of two constraints to force the robot to act for long enough to avoid disrupting its behavior while also requiring that its localization information is accurate enough to use.

Constraint 1—Enforcing Action: Our first constraint is that the robot spend a certain amount of time acting before it stops to look for landmarks. If we do not constrain the amount of time the robot spends acting, it will stop so frequently to look for localization information that its ability to continue its action will be disrupted. Each time the robot stops to look for landmarks, there is some chance that it will have trouble finding the ball when it finishes localizing and looks for it again. This happens because the robot accidentally nudges the ball away, or because it fails to stop moving before looking away from the ball. Therefore, stopping more frequently increases the chance that this will happen and the robot will have to begin searching for the ball, a time-consuming procedure.

However, if the robot does not look around for localization information, it loses the opportunity to get and use this valuable information which allows it to score much more quickly.

Our scheme uses a counter to require the robot to act for a specified amount of time before looking for landmarks. The amount of time the robot must act before looking could depend on the confidence the robot has in its current localization information and on its current goals. In our scheme, however, it is invariant.

We require that the robot act for the time it takes the image module to proccess 350 frames of data, or about 40 seconds. Recall that stopping to look for landmarks takes the robot between 15 and 20 seconds, not counting the time it takes it to recover the ball afterwards. So we demand that it spend about 2/3 of its time acting.

We chose to count the time in image frames processed by the vision module because a full system call is more expensive than using this information. The number of frames processed is relatively constant per unit time, and is immediately available to the control module since each processed frame invokes an update in the control module.

Constraint 2—Limited Localization: The second constraint is how accurate we demand the localization information to be. We measure accuracy with the standard deviations returned by the localization module. If the information is accurate enough, the robot should not stop to look for landmarks. But if it is not accurate enough, the robot should not use it.

It is not immediately obvious how good our localization information must be before it is usable. Clearly, if our demands are too high, the robot will rarely be able to use the information it has gathered. And if they are too low, it will use information that is so inaccurate as to be useless at best, and damaging at worst.

Our results show that a "good" localization estimate of θ, or the angle of the robot on the field, have a standard deviation of $30°$ or less. A "good" localization estimate of x and y, the coordinates of the robot's location on the field, must have a standard deviation of less than 600mm, or just under one quarter of the field length.

Behaviors with No Need for Localization One of the best ways of finding a balance between localization information and time spent to acquire it is simply to avoid localizing when it is not necessary. We have identified two times when it is unnecessary: when the robot has recently lost sight of the ball, and when it has no information at all about the location of the ball. Our scheme allows the robot to act intelligently but does not rely on localization information in either of these situations.

Recovering a Recently Lost Ball: The robot often loses sight of the ball while it is trying to manipulate it. This is usually because it has walked past the ball. It must search for the ball, but because it has just lost sight of the ball, more information is available for it to use.

Instead of incorporating a costly full-scale world view into the robot architecture, we implemented a very simple but extremely effective algorithm. When the robot loses sight of the ball, it first walks backwards for almost the entire width of the field. If it has walked past the ball, this usually allows it to spot it. However, sometimes walking backwards makes the robot turn its body, sometimes away from the ball. So, like our team last year, CMTrio-98 (?), this year's robots turn in the direction in which the ball was last seen. After this, the robot considers the ball lost and begins a random search for it.

Random Search: When the robot does not know where the ball is, it must wander the field to search for it. One way of searching for the ball is to build an "oriented" search, in which the robot uses localization information to systematically search each area of the field. This relies on very accurate localization information and on a complete search of the field, both of which take a lot of time. Instead, we use a very simple algorithm that is much faster. We wrote a random search algorithm that does not rely on localization information at all. Until the robot sees the ball, it alternates between walking forward a random amount and turning a random amount.

Since it is not reliant on localization information, our random search algorithm allows the robot to cover the field more quickly, on average, than does a comparison scheme we built in which the robot attempts to walk a circuit of the field.

Acting with Little Information

Frequently during games, the standard deviations of the robot's localization information are so high that the information should not be used. As explained in Section ??, the robot should not stop and look every time its localization information is inaccurate. Therefore, we must make sure that it can act reasonably even when its localization information is not good enough to use.

We wrote an algorithm which allows the robot to score goals without information about the robot's angle and x and y location on the field. The algorithm is as follows:

```
Until see the goal,
  walk sideways to the right around the ball;
When see the goal,
  stop walking sideways;
Walk forward into the ball.
```

This allows the robot to consistently score goals with no information from the landmark-based localization module at all. However, this takes the robot much longer than it does when the robot has such information.

Upgrading Performance

Not only must the robot be able to progress towards its goals when it does not have useful localization information, but it must be able to improve its performance as soon as it gets such information. We used several different strategies to make sure that the robot's performance would be able to improve quickly as well as degrade gracefully as the availability of good information changed.

Localization-Dependent Performance Enhancements We have developed three performance enhancements that rely on localization information and that are robust and reliable even with noisy information (?). The first helps the robot decide in which direction to circle around the ball, or whether to circle at all. The second allows the robot to skew its approach to the ball so that it doesn't have to spend time circling. The third allows the robot to score goals even if it is unable to see the goal itself.

Mode-Based Architecture We built a control architecture based on basic modes of behavior we identified as important. To switch between modes, the robot uses knowledge about basic features of its state, such as: i) whether it is paused, ii) into which goal it is trying to push the ball, iii) whether it is in possession of the ball, whether it is close to the ball, and iv) whether it knows where the goal is. Performance enhancements include switching to a different mode (for example, if the robot has good enough localization information, it knows where the goal is without seeing it, so it can appropriately change from circling the ball to pushing it towards the goal), and improving the performance of the robot within a mode.

The modes we have defined for the attacker are:

1. `Head Searching`: the robot is searching for the ball by turning its head;
2. `Searching`: the robot is searching for the ball by turning its head and its body;
3. `Approaching`: the robot is approaching the ball;
4. `Circling`: the robot is circling the ball;
5. `Scoring`: the robot is pushing the ball towards the goal;

The algorithm we use to switch among these modes is approximately as follows:

```
If robot sees ball and is not close to it,
  mode = Approaching;
If robot does not see ball and did recently,
  mode = Head Searching;
If robot does not see ball and has not recently,
  mode = Searching;
If robot does not know where goal is and
    is close to ball
  mode = Circling;
If robot knows where goal is and is close to ball,
  mode = Scoring.
```

We are able to optimize the performance of the low-level implementation of the modes by using localization information. By separating the high-level behavior from the low-level implementation, we ensure that the robot's high-level behaviors do not change frequently as the available information changes. Instead, the way the robot executes these behaviors changes in response to the lack or availability of localization information.

Special Cases—Urgent Action

In some cases, the balance between localization and action does not apply. Although each of these cases could perform much better with localization information, immediate action is necessary, so the cost of localizing thoroughly is prohibitively high. We have found shortcuts and compromises that allow the robot to perform as well as possible in these special cases, even without spending time looking around for landmarks.

Approaching the Ball Possession of the ball is a critical part of a soccer game. The team that is able more frequently to possess the ball has an incredible advantage over its opponents. Therefore, in our strategy, when the robot sees the ball, it rushes towards it, not waiting to localize.

This strategy has negatives, clearly. If the robot does not know where it is on the field, it will not know what to do with the ball when it gets to it. Nevertheless, it is better for a robot to look around when it is in possession of the ball than when it is farther from the ball. When the robot is standing near the ball, it is blocking one side of the ball from visibility and attacks, and is more able to respond quickly to an attack because it is already close by.

Kickoff We were surprised to discover, in the initial games of RoboCup-99, how much an advantage is gained by winning the kickoff. When the ball moves to one side of the field, it is very difficult for the robots to move it to the other side of the field. In the RoboCup-99 games, the team that won the kickoff usually scored a goal, simply because the ball never emerged from the side of the field to which it was initially pushed.

Because of this, we felt it was crucial to have an extremely aggressive kickoff. We took advantage of the fact that, at each kickoff, the robots were facing the opposite goal by having them run straight towards the ball, which is placed in the center of the field, between them and the goal. We allowed the robots to run with the ball without localizing at all for almost half the length of the field–nearly the whole distance between their starting positions and the goal. Even when the error-prone motion of the robots causes them to stray far from their projected path towards the goal, they are usually able to drive the ball onto the other robots' side of the field before stopping to localize.

This was a very important advantage in almost all of our games, since we were able to win most of the kickoffs. In the one game we lost, we had accidentally turned this feature off during the first half of the game, and our robots were no longer able to win the majority of the kickoffs. When we turned it back on in the second half, our robots again dominated the kickoffs, and were therefore able to score a goal and prevent the other team's robots from scoring any.

Goalie Another time when swift action is crucial is when a robot is playing the position of goaltender. However, this position also requires very accurate localization, since it is so necessary for the goalie to be inthe correct position in front of the goal. We found that if we allow the goal to rely heavily on the localization model, the robot spends a lot of time looking for landmarks instead of looking at the field, and so can miss saving goals.

Also, it seems that the localization algorithm doesn't provide sufficiently accurate information for the goaltender position. It often allows the goalies to be close to the appropriate position, but even with a standard deviation of 10cm, given that the goal is 60cm, this can easily leave large segments of the goal unblocked and unguarded. Not only that, but sometimes the localization information is wrong, since it relies on seeing the landmarks, and the goalie wanders out into the field before discovering that it is in the wrong place and trying to return.

Finally, by avoiding the landmark-based localization module altogether, we were able to find a way for the goaltender not only to avoid looking frequently at landmarks, but also to position itself more accurately in front of the goal. Our final algorithm is as follows:

```
Starting Position: Centered in front of the goal,
  facing the other side of the field.
Scan the horizon for the ball;
If the ball is seen, run straight after it;
If lose sight of the ball for more than 2 frames,
  turn until own goal is seen;
If see own goal, run towards largest area of goal seen
  until it fills visual field;
If own goal fills visual field
  turn until opposing goal is seen.
```

This final version of the goalie is extremely aggressive, and extremely successful. One of the main strengths of this algorithm is that it takes advantage of the special situation in which the goal tender finds itself—standing very close to one goal, and facing the other. Because the goals are the largest visual features on the field, it is easy to use them to localize this special position. Because the goalie pushes the ball far away from the goal, it usually has plenty of time to run back to the goal and turn around before the ball comes nearby.

Conclusion

In this paper, we reported on our work on the control system of soccer-playing Sony quadruped legged robots. We briefly described the components of the robots and the setup of the game. We then contributed an algorithm to control and balance the amount of localization information the robot has; to score goals without that information; to employ as much of that information as is available and to upgrade and degrade performance gracefully as availability changes; and to deal with special cases in which more urgent action is necessary. Finally, we presented the results of tests we ran to determine the best way to control the amount of localization information the robot has.

Results from our matches in RoboCup-99 at IJCAI-99, Stockholm show our algorithms to be effective. Our team won all but one of its games, and the one it lost was lost by only one goal. Our team was the only one in this year's league to score goals against opposing teams and never to score a goal against itself. Our goaltender was the only one in this year's league to score a goal.

Acknowledgements

We would like to thank Sony for providing us with wonderful robots and walking movements to work with.