Vision Calibration and Processing on a Humanoid Soccer Robot

Piyush Khandelwal, Matthew Hausknecht, Juhyun Lee, Aibo Tian and Peter Stone
Department of Computer Science
The University of Texas at Austin
Austin, TX 78712
{piyushk,mhauskn,impjdi,atian,pstone}@cs.utexas.edu

Abstract— **In RoboCup, the problem of quickly and accurately processing visual data continues to pose a significant challenge. The Aldebaran Nao, currently used by the Standard Platform League, has two cameras for visual input, of which only one has been typically used. The integration of both cameras presents a new opportunity but also a challenge. While it is possible to obtain better information using both cameras, more cameras require more work to calibrate. We propose a novel camera calibration algorithm which automatically tunes a camera such that its color perceptions match those of another camera. Additionally, recent vision challenges introduced in RoboCup have necessitated the use of higher resolution images. We build on existing work in color based segmentation and present novel extensions to facilitate the move to higher resolution images, including memory optimizations, fast line and curve detection, and differentiation via robot pose based transformations. All work presented in this paper was successfully used by the UT Austin Villa Robot Soccer team, which secured 3rd place overall and 2nd place in the technical challenges at RoboCup 2010.**

I. INTRODUCTION

RoboCup is an international research initiative designed to advance the fields of robotics and artificial intelligence, using the game of soccer as a challenge domain. The long-term goal of RoboCup is to build a team of 11 humanoid robot soccer players that can compete with a human soccer team by the year 2050 [1]. RoboCup is divided among several leagues, each presenting different research problems to the robotics community.

The standard platform league (SPL) in RoboCup uses the same hardware platform for all teams in order to focus research on software development, allowing the direct comparison of many different approaches. The SPL's current robot, the Aldebaran Nao, is a humanoid which stands 58cm high. It has two CMOS digital cameras as its primary visual sensors, both of which are located along the center of its face at different angular offsets. The Nao needs to split up the processing power of an x86 AMD GEODE 500 MHz processor amongst tasks such as vision, localization, motion control, and behavior, and at the same time maintain a sufficient frame rate to be reactive enough to play soccer.

Although significant advances have been made in recent years in object detection, the Nao's limited processing capability has prevented the reuse of many state-of-the-art vision systems. For instance, in our experiments, the frame rate for the Nao drops to approximately *14 fps* just by accessing the entire image without any additional processing. Such limitations motivate the creation of innovative solutions, such as fast image segmentation and blob formation [2]. In this work, raw color values in YUV space are mapped to segmented colors via axis-parallel thresholds, forming cubic regions in color space. Because actual color boundaries do not necessarily conform to these cubic regions, segmentation precision was often limited. Another approach is to create a table which maps values from YUV space to segmented colors [3], removing the limitation of cubic color spaces. This approach proved effective at segmentation, but required significant time and effort to create each color table. Since each camera has its own peculiarities, a unique color table is required for each camera. The use of both cameras on a single Nao doubles the total number of color tables to be created and hence doubles the total effort. Our calibration approach (section III) simplifies this task by automatically adjusting the hardware parameters for a given camera in order to match the color perceptions of another camera, allowing the re-use of the same color table.

Fig. 1: An image of the 6m by 4m soccer field on which the Naos play.

Secondly, the recent reduction in ball size necessitated the use of a higher resolution image to continue to detect the ball at long distances. We propose two memory optimizations that speed-up the image retrieval for such high resolution images.

Finally, we implemented a new rapid line and curve detection methodology capable of quickly and accurately identifying both straight lines and curves. Such an approach is necessary to identify the center circle, one of the few unique landmarks on the field. To differentiate between lines and curves, we employed a transformation which projects the detected line segments from the robot's perspective onto the ground plane (see Fig. 7). Our vision processing approach is described in Section IV of this paper.

II. PROBLEM OVERVIEW

In this section we formally present the vision problem that needs to be solved. The RoboCup soccer environment can be described in terms of a few discrete objects (Fig. 1). The *green field (ground)* is demarcated by *white lines*, a *white center circle* and *white penalty cross markers*. There is a *blue goal* and a *yellow goal* on opposite sides of the field. The robots play soccer with an *orange ball*. Finally, robots on each team have *blue* and *pink* team markers.

The vision problem has been previously described in [3]. We repeat the problem statement here for completeness, while incorporating the differences in the vision challenge since:

1) Inputs:
   - A stream of limited-field-of-view images with defects such as noise and distortion. These can be from either of the 2 cameras on the robot.
   - The robot's pose over time, particularly the tilt, pan, roll of the camera. These are typically used to calculate the relative position of the camera with respect to the ground.
2) Outputs:
   - Distances and angles to a fixed set of stationary objects (such as *lines*, *goals*). These objects have a known set of possible locations. This information is typically required for localization.
   - Distances and angles to a fixed set of mobile objects (such as *balls*, *robots*). This information is typically required for behavior control.

III. CALIBRATION

Seamlessly integrating both the Nao's cameras into a cohesive vision system remains a challenging problem. We take a step towards this goal by creating a system capable of auto-calibrating the hardware parameters for multiple cameras to achieve consistent color perceptions across all cameras.

The need for such an auto-calibration system arose when experiments indicated that the color perceptions of the Nao's top and bottom cameras were sufficiently different to preclude the Nao's color-based object identification with the same color table. The skewed color perceptions likely result from differences in the amount and the manner in which light enters each camera as a result of the different camera enclosures and mounting angles used for the top and bottom camera. Additionally, small manufacturing differences even between top or bottom cameras of two different robots often result in different color perceptions. In the past this problem has been addressed by creating and tuning separate color tables for each camera, but already this takes considerable effort, even for experienced tuners, typically one hour per table. Adding the top camera of each robot to the list of cameras needing to be tuned would double the total number of color tables needing to created and tuned, increasing the amount of effort and time required to play on a new field.

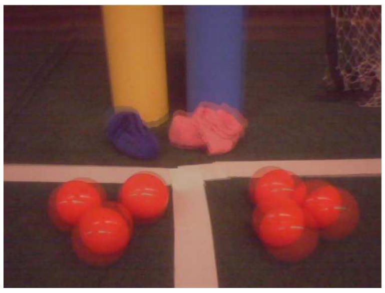


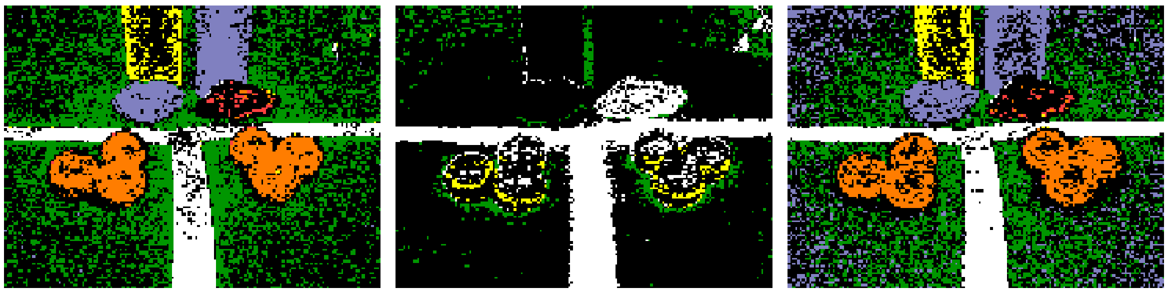
Fig. 2: Approximate dual camera overlay

Past work [4] has addressed a related problem of standardizing color perceptions by automatically creating color tables based on known objects in the robot's visual field. We take a different approach by automatically tuning the hardware parameters (brightness, contrast, hue, gain, etc.) of each camera until the perceived colors match color perceptions of a known good camera. After color perceptions are standardized, a single color table can theoretically be used for a full team of robots, saving the effort of manually creating one color table per camera or two per robot.

The process of standardizing color perceptions begins with a static image R being saved from a known good camera. R should be representative in the sense that it should contain non-negligible portions of each color defined in the color table. For the Naos this involved capturing an image containing orange balls, white lines, portions of the blue and yellow goal, and blue and pink team identifiers. Each color needs to be present in R because the color of other cameras will be adjusted to match this image and if only a few colors are present, these color will likely be well matched at the expense of the others.

After acquiring the static image R, another camera from the same or a different robot is selected and positioned such that the objects in it's image I overlap those in R. We have found that using a graphical overlay of I and R serves as an effective guide to a human attempting to correctly position the new camera. The fact that the Nao's top and bottom cameras are mounted at different angles inside of the robot's head complicated the process of trying to align the top camera to exactly overlap a representative image taken from the bottom camera as the physical limits of the Nao's neck joints prevent the head from obtaining certain angles which would be necessary to compensate for the angular mount differences between the cameras. However, combining body and neck tilt allowed a close approximation of the original camera position. Figure 2 shows the graphical overlay of images taken from a Nao's top and bottom camera. This overlay is impossible to perfectly align for this reason.

Having aligned the current camera's image I with the representative image R, we proceed to automatically tune the gain, exposure, blue chroma, red chroma, brightness contrast, saturation, and hue of the current camera until we maximize the color match between I and R. Tuning is performed by a Hill Climbing search through the space of possible camera settings. At each iteration of hill climbing, a new set of parameters N is created by perturbing the current parameters



(a) Representative Image (b) Before Calibration (c) After Calibration

Fig. 3: Segmented images before and after camera calibration.

C in a random direction. Next, some number of images (we used 3) are captured using the new settings. Finally, if the average color match of these images captured using N exceeds the average color match of the old parameters C, then N becomes the new baseline and another round of hill climbing begins.

The color match between each image I and R is computed by segmenting both I and R with some color table (Fig. 3b and Fig. 3a respectively), typically the one in use when R was saved. This allows us to filter out the irrelevant colors in both images and compute the image match over the relevant colors defined in the color table. To compute the image match, we examine each of the different colors in the representative image. For each pixel labeled with that color in R, we check if the corresponding pixel in I is also labeled with the same color. The average pixel match for that color is computed as the number of pixels in which R and I match divided by the total number of R's pixel of that color. Taking the average of these pixel matches provides the overall color match. Algorithm 1 shows the detailed computation. Computing the color match in this manner encourages the highest overall match between all colors present on the field. If certain colors were deemed more important than others, a weighted average could be used instead.

In practice, the hill climbing search was run until convergence (Fig. 3c). Approximately two frames were scored per second, resulting in one iteration (3 frames) per 1.5 seconds. Convergence was typically found within 100 iterations or 2.5 minutes of real time. However, the algorithm was often allowed to run for several hundred more iterations to ensure it had reached a local maximum. Moreover, the only human intervention required throughout the entire process is the initial camera alignment.

Auto-calibration was successfully applied to each of the four Naos on UT Austin Villa's 2010 RoboCup team in order to learn camera parameters for each robot's top camera. When top cameras were integrated into the existing code, auto-calibration saved the team from having to tune the hardware parameters on each of the four robots' top cameras, as well as having to create new color tables for these four cameras. In practice, although the color match was likely good enough to use without color table modification, small changes were made to the color tables of each robot to reach a very fine grained color space. Even so, these modifications required time on the order of 5 minutes per color table rather than the typical hour to create a new table.

Algorithm 1 Color Match Computation

```
1:  $R$ ← Representative Image
2:  $I$ ← Current captured image
3:  pixelCnt ← # pixels of each segmented color
4:  matchCnt ← # pixel matches for each segmented color
5:  avgMatch ← average pixel match for each color
6:  **for** each segmented pixel $p$ in image **do**
7:      truePixelColor ← $R[p]$
8:      givenPixelColor ← $I[p]$
9:      pixelCnt[truePixelColor]++
10:     **if** truePixelColor == givenPixelColor **then**
11:         matchCnt[truePixelColor]++
12:     **end if**
13: **end for**
14: **for** each $color$ present in image **do**
15:     avgMatch[$color$] ← matchCnt[$color$]/pixelCnt[$color$]
16: **end for**
17: overallMatch ← average(avgMatch)
18: return overallMatch
```

Figure 4 graphs the overall color match as the top camera of a Nao is auto-calibrated to match the color perceptions of the Nao's bottom camera. Figure 3 shows the representative image R from the bottom camera, and sample images from the top camera before and after calibration.

Several limitations of auto-calibration not present in UT's robotics lab became apparent after using it on the practice

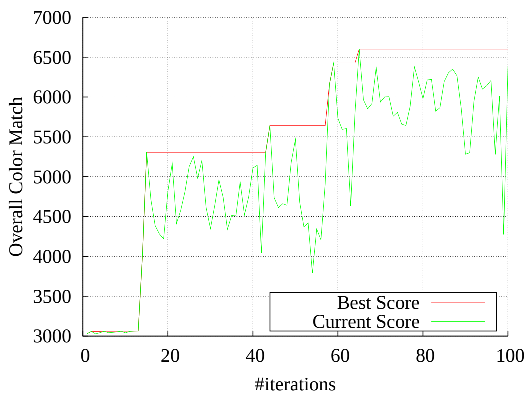


Fig. 4: Sample calibration run converging to a local maximum.

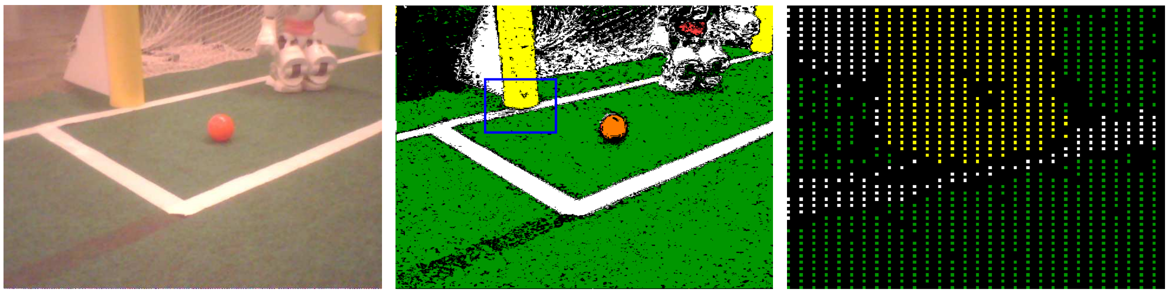

(a) Raw Image (b) Segmented Image (c) Pixel Sub-sampling

Fig. 5: The segmentation procedure is shown here. The raw image in Fig. 5a is converted to the segmented image using the color table. If we segment all the pixels, we get the image in Fig. 5b. In Fig. 5c, we show an enlarged portion of the blue box in Fig. 5b, which shows the true segmented image with pixel sub-sampling.

fields of RoboCup 2010. For instance, it was often challenging at RoboCup to create a representative image without having to re-arrange the goals on the practice field to fit into a single field of view, inconveniencing other people trying to simultaneously use the field. Additionally, auto-calibration works best with a static background image which was hard to maintain on a crowded practice field. However, temporary disturbances in background image pose little problem to the algorithm as they simply result in the current policy receiving low color match scores, which causes the hill climbing to stay at the same policy until the background disturbance is cleared.

We have presented an approach to the problem of standardizing color perceptions between different cameras. However, there is still much work to be done in coordinating the use of the top and bottom camera such as the choice of when to use each camera and how to identify the best times to switch between the two in order to maximize the amount of pertinent information acquired.

IV. VISION PROCESSING

In addition to maximizing information acquisition through the use of two cameras, we also improve object detection with the use of a higher resolution image as well as line and curve detection and differentiation via robot pose estimation.

The raw images provided by the Nao are in the YUV422 format and have a 640 by 480 size, giving the images a size of 600 KB. A huge amount of the processing time in vision is consumed by accessing this image, which amounts to 8.8 MBps at 15 fps. This is a significant difference from previous years where a 16 times smaller image (160 by 120) was used. Since image accesses are the bottleneck of the image processing system, we can increase frame rate by optimizing the memory accesses of the vision system. We present two independent techniques that should be applicable to any color based segmentation system.

The first technique happens at the lowest level of the hardware: cache, memory, and memory controller. The main idea is to reduce the number of memory accesses with type casting and to maximize the cache hit rate. When accessing two neighboring pixels, we burden the memory controller only once by retrieving a 32-bit value instead of querying four 8-bit pixel channel values independently. This retrieves information about two adjacent pixels together, containing the Y channel for both the pixels, and the shared U and V channels. Thus, the number of memory read requests can be reduced by a factor of 4. Once the pixels' YUV values are read, the YUV tuple values are looked up in the color table, which contains a one-to-one mapping between YUV tuple values and segmented colors [3]. The second technique reorganizes the color table to maximize the cache hit rate when querying a sequence of YUV tuples. Since two adjacent pixels share U and V channels, the order of the color channels in the color table is maintained as VUY instead of YUV. This reduces cache misses while segmenting a pair of adjacent pixels, due to their proximity in the color table. Results from memory optimization are presented in Table I.

TABLE I: The time taken to access and segment one image frame.

Method	Time (ms)
No optimizations	138.616
Color table reorganization	112.867
Color table reorganization & efficient retrieval	**96.023**

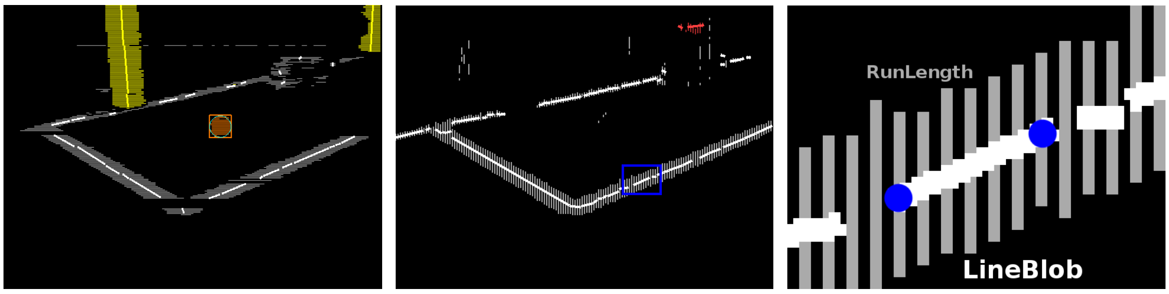
Our vision system divides the object detection task into 3 stages, which are listed below.

1) **Blob Formation** - The segmented image is scanned using horizontal and vertical scan lines, and blobs are formed for further processing.
2) **Object Detection** - The blobs are merged into different objects. In this paper, we shall primarily limit our discussion to line and curve detection.
3) **Transformation** - We use the information given by the pose of the robot to generate ground plane transformations of the line segments detected.

The novel contributions in the vision processing pipeline are the new method for rapid line and curve detection during the object formation phase, and using the transformations to make differentiating between lines and curves easier.

A. Blob Formation

The blob formation procedure is similar to previous approaches such as [2] and is outlined in Fig. 6. The segmented image is scanned using vertical and horizontal scan lines, effectively sub-sampling the image (Fig. 5c). A vertical scan line is placed every 4 columns and a horizontal scan line every other row, and only these sub-sampled pixels are segmented (Fig. 5). Our system also retrieves and segments additional pixels in an ad hoc manner when we are unable to



(a) *HorizontalRunLengths* & *VerticalBlobs* (b) *VerticalRunLengths* & *HorizontalBlobs* (c) Enlarged view

Fig. 6: A representative image for the blob formation procedure. An enlarged view of the blue box in Fig. 6b is shown in Fig. 6c for clarity. In Fig. 6c, the grey lines are the vertical run lengths and the white line is the corresponding horizontal blob formed.

detect the ball. From these scan lines, we form *RunLengths* of same colors. *RunLengths* are a set of contiguous pixels along a scan line having the same segmented color value. Fig. 6a shows the *HorizontalRunLengths* generated using horizontal scan lines, and Fig. 6b shows the *VerticalRunLengths* from the vertical scan lines. Fig. 6c gives an enlarged view into a portion of Fig. 6b, which shows the individual *VerticalRunLengths* in grey.

In the next stage we form blobs from these run lengths. Instead of using the Union-Find procedure as in [2], we simply merge adjacent run-lengths of the same color on the basis of overlap and width similarity. The main reason behind using this simpler approach is that it is more geared towards finding lines, and not a similar blob of color. Note that *VerticalRunLengths* combine to form a *HorizontalBlob*, as shown in Fig 6c. Similarly *HorizontalRunLengths* are combined to form *VerticalBlobs* (Fig. 6a).

For each blob formed, we calculate some information required for further processing. For a given horizontal blob, we have a start point and an end point, as shown by the blue circles in Fig. 6c. This gives us start coordinates (x_s, y_s) and end coordinates (x_e, y_e). We also calculate $\dot{y}_s$ and $\dot{y}_e$ as the start and finish slope respectively, by calculating slope across a few run lengths. Next we calculate $\ddot{y}$, which is the rate of change of slope across the blob. Some averaging is performed to account for noise. A similar set of values are also calculated for each vertical blob.

B. Object Detection

Our next step aims to join these disconnected blobs to form objects. Ball and robot detection are done using heuristics based on the output of the blob formation procedure, and are not discussed here. In this section, we will primarily talk about our approach to line segment detection, which is one of the novel contributions of this paper. Vertical blue and yellow line segments serve as candidates for goal posts. Horizontal and vertical white line segments serve as candidates for the lines and center circles on the field.

We explain our methodology for segment formation in terms of horizontal blobs, and an equivalent discussion should automatically apply for the vertical blobs. To construct candidate segments, we make the simplifying assumption that each line segment, whether it be a straight line or an ellipse (i.e. a projection of the center circle - Fig. 7a), can be represented by the following parabola for short segments:

Algorithm 2 Line Segment Formation

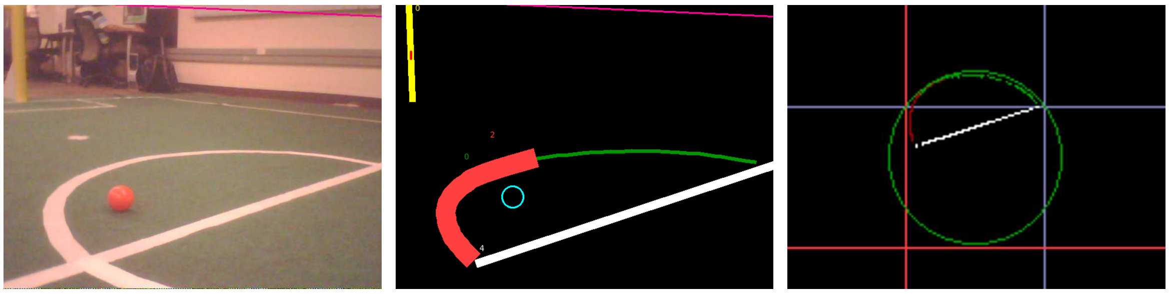
```
1: Input: $B \leftarrow$ Ordered set of $Blobs$
2: Output: $S \leftarrow$ A set of candidate $LineSegments$
3:
4: for each Blob $b_i$ in $B$ do
5:   $s$ = new $LineSegment$
6:   $s$.initialize($b_i$) {Initialize A,B,C. Put blob in segment}
7:   $bestSegment$ = recurse-check-segment ($s$, $b_i$, $B$)
8:   if isAboveThreshold($bestSegment$) then
9:     for each Blob $b$ in $s$ do
10:      $B$.remove($b$)
11:    end for
12:    $S$.add($s$)
13:  end if
14: end for
15:
16: function recurse-check-segment ($s$, $b_i$, $B$)
17: $bestSoFar$ = new $LineSegment$
18: for each Blob $b_j$ in $B$ s.t $j > i$ do
19:   if $b_j \approx s.predictb_j.x_s$ then
20:     $s.updateABC(b_j)$
21:     $s.add(b_j)$
22:     $current$ = recurse-check-segment ($s$, $b_j$, $B$)
23:     if is-better($current$, $bestSoFar$) then
24:       $bestSoFar$ = $current$
25:     end if
26:   end if
27: end for
28: return bestSoFar
29: end function
```

$$y = ax^2 + bx + c \tag{1}$$

The advantage of this assumption is that it gives simple expressions for the derivatives of this function. This allows us to reconstruct the curve given a point on the curve (x_1, y_1), and the slope and the rate of change of slope at that point $(\dot{y}_1, \ddot{y}_1)$:

$$\dot{y} = 2ax + b, \quad \ddot{y} = 2a, \quad \text{which gives} \tag{2}$$

$$a = \frac{\ddot{y}_1}{2}, \quad b = \dot{y}_1 - 2ax_1, \quad c = y_1 - ax_1^2 - bx_1 \tag{3}$$



(a) Raw Image (b) Detected Segments (c) Transformed Image

Fig. 7: For the raw image in Fig. 7a, the detected objects are shown in Fig. 7b. The circle is detected in 2 separate segments. The transformation to the ground plane is shown in 7c.

The procedure for merging the blobs together is presented in Algorithm 2. To merge these blobs together, we traverse over the set of sorted blobs using a stack. We initialize a segment (the stack) with a single blob b, and calculate the parameters a, b and c for that segment, using the values $(x_e, y_e, \dot{y}_e, \ddot{y})$ calculated for the blob (Alg. 2 Line 6).

Once initialized, we start the recursion process (Alg. 2 Line 7). Using the current a, b and c values from the segment, we can predict the curve for any given x (using Eqns. 1 and 2). We then check if any blobs in the global list match our prediction. Once a match is found, we add that blob to the stack, recalculate the a, b and c values (using Eqn. 3), and recursively repeat the procedure.

We backtrack along the stack if we are unable to find a suitable blob to add, or when all possible options have been explored. While backtracking, we keep track of the best candidate found in front of every node, in a form of dynamic programming. Once the entire tree has been explored, the best candidate at the root node is the most suitable line segment. If it satisfies the threshold for being a line segment, we add it to the list of candidate line segments and remove all the blobs in this segment from the overall blob list. The final object detection is shown in Fig. 7b.

C. *Transformation*

For the purpose of localization, it is important to be able to distinguish between lines and curves. Otherwise, a robot observing the center circle could mistakenly believe that it is observing the center line or vice versa. Distinguishing between lines and curves can be difficult in the vision frame because the projections of lines and circles on the camera image often look fairly similar. The inevitable noise and incorrectly formed line segments exacerbate the problem.

For the purpose of distinguishing lines from curves, we use transformations. Based on the current pose of the robot, it is possible to construct a transformation matrix from a point in the vision frame onto the ground plane [5]. This transformation is possible because the height and orientation of the camera can be estimated relative to the ground using the robot's sensors. Since this matrix is calculated once every frame, we can efficiently transform from the vision frame to the ground plane.

After obtaining the transformation for each segment, we use a simple metric for first classifying whether a detected segment is a line or not. We calculate the angle between the first and second halves of the segment, and classify the segment as a curve if this angle is above some threshold. For all curves, we perform circle fitting by calculating the center and radius using 3 seed points. We then verify if the circle has roughly the same radius as the center-circle on the field.

This process is illustrated in Fig. 7. Given the raw image (Fig. 7a), we can apply the vision system to generate the final detected segments (Fig. 7b - note that two segments are detected from the circle). We transform these segments (Fig. 7c) and perform circle fitting on the curves. The green colored segment shows a curve that was fit to the circle shown in green. The red colored segment shows a curve which was not detected as a circle, due to an incorrect projection.

V. CONCLUSION

In this paper we presented aspects of the vision system used by the UT Austin Villa's 2010 robot soccer team. In the process of the development of this vision system, we encountered obstacles which motivated novel solutions to the tasks of camera calibration and more efficient vision processing. We believe that these extensions played an important role in UT Austin Villa securing third place at RoboCup 2010.

VI. ACKNOWLEDGMENTS

This work has taken place in the Learning Agents Research Group (LARG) at the Artificial Intelligence Laboratory, The University of Texas at Austin. LARG research is supported in part by grants from the National Science Foundation (IIS-0917122), ONR (N00014-09-1-0658), DARPA (FA8650-08-C-7812), and the Federal Highway Administration (DTFH61-07-H-00030). Matthew Hausknecht is supported by an NSF Graduate Fellowship.

REFERENCES

[1] H. Kitano, M. Asada, I. Noda, and H. Matsubara, "RoboCup: Robot world cup," *Robotics & Automation Magazine, IEEE*, vol. 5, no. 3, pp. 30–36, 2002.

[2] J. Bruce, T. Balch, and M. Veloso, "Fast and cheap color image segmentation for interactive robots," in *Proceedings of IROS-2000*. Citeseer, 2000.

[3] M. Sridharan and P. Stone, "Real-time vision on a mobile robot platform," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, August 2005.

[4] M. Sridharan and P. Stone, "Autonomous color learning on a mobile robot," in *Proceedings of the Twentieth National Conference on Artificial Intelligence*, July 2005.

[5] T. Hester, M. Quinlan, P. Stone, and M. Sridharan, "TT-UT Austin Villa 2009: Naos across Texas," The University of Texas at Austin, Department of Computer Science, AI Laboratory, Tech. Rep. UT-AI-TR-09-08, December 2009.