

# RoboCup SSL 2005 Team Description: Wingers (University at Buffalo)

Stefan Zickler and Michael Licitra

UB Robotics, University at Buffalo, NY 14261, USA,  
zickler@gmail.com,  
mlicitra@buffalo.edu,

WWW home page: <http://www.eng.buffalo.edu/ubr/robocup.php>

**Abstract.** Since the reception of the Small Size League Championship at the RoboCup 2004 US Open, the Wingers' team members have designed and implemented several dramatic enhancements to both our robotic hardware as well as to our entire software platform. In this team description paper we will demonstrate and explain the most relevant components of our current system and focus especially on the details of the most innovative improvements, such as our new lighting-adaptive computer vision algorithm and our dynamically layered behavioral A.I. model.

## 1 Team Outline

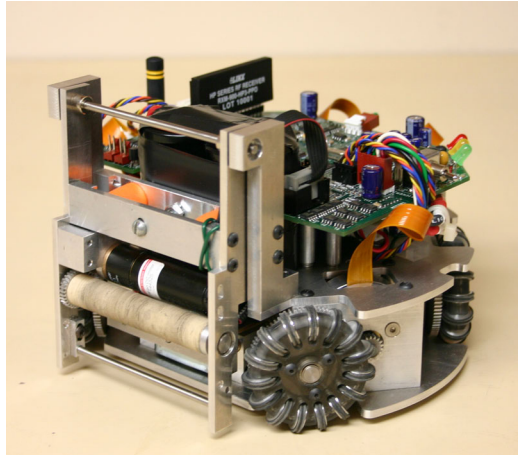
Our team belongs to UB Robotics, an undergraduate engineering club of the University at Buffalo, New York. Team leaders are Michael Licitra (Hardware Leader, Computer Engineering Major) and Stefan Zickler (Software Leader, Cognitive Science Major). The rest of our team consists of several other undergraduate students from various fields. We have been the 2004 Champion of the RoboCup US Open and are looking forward to participate in our first RoboCup World Cup.

For this year, our system has already undergone major redesign and improvement. A completely new robot hardware has been designed (Figure 1) and is currently being built. Most of our software has been rewritten and has seen tremendous increases in both speed and robustness. Especially our Vision and Behavioral control systems have been improved dramatically and should now be considered mature platforms that can be further expanded in the future.

Our software follows a modular approach and consists of a dual-camera vision system and a dynamically layered behavioral A.I. module. We make use of several cutting-edge solutions to achieve a high overall system performance, such as adaptive lighting vision, Kalman-filtered signal analysis, a simple predictive world model and extended RRT path planning [1]. Additionally it is planned to implement methods of statistical adaptability to our behavioral A.I. control in order to adapt to our opponent more dynamically during games.

Our 4-wheeled robots make use of a brushless motor powered drive system with

customized omni-directional wheels. Our ball handling unit features a brand-new dampening system and an electro-magnetic solenoid as high speed kicking device and it furthermore includes a simple rubber-coated dribbler-bar for increased ball control.



**Fig. 1.** Prototype of our 2005 robot

## 2 High Speed and Lighting Adaptive Computer Vision

### 2.1 Introduction to our Vision System

Our 2005 Vision System has benefited from major improvements in both speed and robustness. We have focused on the development of algorithms that feature the ability to make the system handle any possible changes in lighting without the need of any human recalibration. This adaptability includes both the handling of local changes (such as shadows or dark/bright areas on the field), as well as rather global ones (such as a light being turned on/off or slight camera adjustments). We believe that our new system has achieved this goal surprisingly well which is why we made it the prime topic of this paper.

Our system currently has a two-camera configuration with two consumer class camcorders interfaced via an analog cable to two low-cost capture cards. Our software has been implemented in C++ (using Video4Linux 2, QT3 and SDL).

### 2.2 Color Calibration Model

The first underlying foundation of our vision system is a standard color classification routine which converts the images' raw-colors (in our case it's RGB

space, but our concepts should be equally applicable to YUV or any other color model of choice) to their numeric RoboCup color labels (e.g. 0 for Blue, 1 for Yellow. . .). For this purpose we make use of a fairly large ( $256^3$  bytes) RGB lookup table (LUT) which does that mapping for us.

As we will see later however, this RGB LUT will only act as a *guideline* to which pixels are 'interesting' and to which color blob they *most likely* belong to. One of the main advantages of our new vision system is that we will no longer be fully relying on this table as our only means of segmentation. This is because this type of simple region-based color segmentation can only provide a fairly static per-pixel color mapping which might be very misleading in certain situations. In fact, all per-pixel color segmentation algorithms (no matter which color-space they work in) are often too intolerant to local color-noise. For example, an orange blob in our image might contain a pixel which has the exactly same color as a certain pixel within a yellow blob, possibly due to color-noise or differences in lighting.

### 2.3 Visually Guided Region Calibration

The interesting part of course begins with the calibration of this RGB LUT. In order to establish region labels, we have implemented a graphical calibration system which operates in HSV space (Hue, Saturation, Value). Internally, this calibration table is an independent LUT which maps HSV (not RGB) to numeric RoboCup color labels. The user is being presented a 2D drawable surface which represents H and V as its two dimensions. The user can furthermore 'scroll' through the different slices of the S-dimension which gives him a complete and graphically editable representation of HSV space. In order for the user to calibrate the table, he or she can scan sample pixels from the image and then establish region mappings by simply 'drawing' a color region onto the GUI. This visual system naturally allows for a very comfortable re-calibration of the table since one can apply slight changes to regions without the need to fully redraw them. We have chosen the HSV color model for this graphical calibration because of its geometric color distribution properties and because it is such an intuitively understandable model. After the user has finished modifying the HSV calibration table, the system needs to do a quick one-time computation to construct the main RGB LUT (using a standard HSV to RGB conversion algorithm). A screenshot of this visual calibration system can be seen in figure 3 (b).

### 2.4 Optimistically Expanding Blob Algorithm

The new key in our system's adaptability lies within its blob-building algorithm. Its general idea is that we will first attempt to locate 'interesting' pixels by using the above described RGB LUT as our entry point. We will then start scanning connected pixels by using an efficient flood fill algorithm. The trick to this blob expansion algorithm is that instead of using the static RGB segmentation LUT we will rather analyze the relative difference in terms of Hue and Value of each new hypothetical pixel that is to be added to the blob. In order to do so, we

constantly need to keep an accurate average of the blobs Hue and Value while we are expanding it.

In other words, the main decision rule of whether a connected pixel should be added to a blob is based on how much its color and intensity differs from the current blob. If this deviation is within a certain limit, the pixel can be added to the blob and the new average of the blob will be adjusted accordingly. The following pseudocode describes this decision process as it occurs during the expansion of a blob:

```
1 // Assume that the following function is called for each pixel P
2 // in order to decide whether it should be added to blob B.
3 // It returns true if P is considered a part of B; false otherwise.
4 bool checkConnectedPixel(blob & B, pixel P) {
5     if ( P.saturation > min_saturation ) {
6         if ( RGB_LUT[P.red][P.green][P.blue] == B.firstColorLabel ||
7             ( diff(P.hue,B.avg_hue) < max_hue_diff
8               && diff(P.val,B.avg_val) < max_val_diff ) ) {
9             adjustHueAverage(B.avg_hue,P.hue);
10            adjustValAverage(B.avg_val,P.val);
11            adjustSatAverage(B.avg_sat,P.sat);
12            return true;
13        }
14    }
15    return false;
16 }
```

**Fig. 2.** Pseudocode of Blob Expansion Decision

Looking at figure 2, Line 5 simply makes sure that the new pixel is sufficiently saturated. Other, more restrictive filters may be added at the same line. Line 6 essentially preserves the traditional method of segmentation, meaning if the new pixel is classified in the LUT as having the same RoboCup color label as the very first pixel of the blob, then it should be added to the blob. Line 7 and 8 represent the new main decision rule which checks whether the difference in hue and value are below a certain user-defined threshold. If the pixel is added to the blob, the blob's HSV averages need to be updated as it is done in lines 9-11.

Since Hue is a circular scale (often represented as degrees from 0 to 359) one cannot simply use a traditional average calculation (e.g. the average of Hue 5 and 355 is not 180, but rather 0). By converting Hue to radians and by separating it into its independent 2D vector components we can safely calculate the correct Hue average.

It is to be noted that in some cases, slightly better results might be achieved when the median is being used instead of the mean for determining the relative differences of H and V. Furthermore, instead of a relative, percentage-based difference, one could make use of the statistical standard deviation which might

be a mathematically more sound solution. Naturally, these considerations present a trade-off between precision and computational intensity. In our testings we have found that the more intensive computations by using a median calculation do not offer significant enough adaptability improvements to justify this increase in latency to the system.

One more very important thing to note is that this HSV difference model will fail for unsaturated colors, such as black or white. This is because saturation will be 0 for which Hue will simply be undefined and its average therefore becomes meaningless. Since our patterns did not include any colorless blobs this was never an issue for our team.

## 2.5 Fail-Safe Blob Average Matching using Euclidean Minimum Distance Classification

We are now done expanding our blob and the next step will be to decide which RoboCup color label we should really assign to it. Since we have built its precise HSV average information over all of its pixels, we can simply look up these values in our hand-made HSV calibration LUT which will immediately return a RoboCup color label. In some cases however, it might turn out that our HSV averages are simply not defined in our calibration table, meaning they don't map onto any color label at all. In these special cases we can simply perform a minimum distance algorithm which will find the *closest* defined color region within the HSV LUT. For example, if the average blob's Hue ends up somewhere between the calibrated regions of 'pink' and 'orange' in our LUT, then we will simply return the label of the region which is closest in a 3D-euclidean sense. Naturally, we don't have time to do a 3D search for the closest neighboring HSV region in real time. This is why we have created an additional HSV LUT which is *fully mapped*. In this table, each HSV value which has not been defined in the original calibration LUT will return the color label of the region with the minimum distance in HSV space. This LUT is constructed automatically each time after the main calibration LUT has been constructed.

## 2.6 Further Processing of Vision Data

After locating all blobs and building their centroid coordinates, we naturally perform quite a variety of different filters to get rid of any irrelevant blobs (such as very large or distorted ones which cannot possibly represent a RoboCup color-pattern). Like most other teams, we then apply a very straightforward pattern matching algorithm to identify our robots' positions and orientations.

In order to accommodate for lens distortion, camera transformations and robot height, we make use of Roger Tsai's coordinate transformation algorithm [2]. Its calibration data is being gathered through capturing black and white dot patterns for which we have derived a special mode in our vision system.

## 2.7 Results

In terms of adaptability, we have performed several tests to analyze our algorithm's performance. We have for instance created extremely uneven lighting and shadows during games. Most of these changes have been handled very well by the new blob expansion algorithm. It seemed as if the new system recognizes all robots perfectly even in the strangest of lighting conditions. However, we should note that one needs to carefully pick the exact relative H and V differences which specify whether a pixel should still be counted toward a blob. Otherwise, the undesired effect of blob over-expansion might appear. Luckily, this fine-tuning is a one-time process which should not need further adjustments as the field's environment changes.

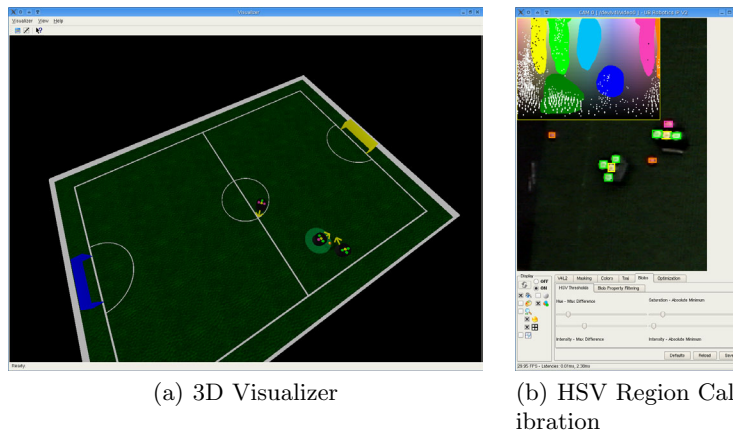


Fig. 3. Screenshots of our System

## 2.8 Emphasis on a High Speed Implementation

It's almost needless to say that all of the above theory has been implemented with a great amount of tweaking and optimization. In our implementation, nearly every complex calculation which has a reasonably limited input-range has been stored within LUTs for increased processing speed. Besides algorithm design, a wise selection of data structures and a limitation of CPU intensive calculations are what led us to the success of achieving very competitive speeds for our system.

## 2.9 Benchmarking

Finally, to give some real data in terms of speed: Using a single CPU Pentium 4 clocked at 3GHz (with HyperThreading enabled), running both visions systems

simultaneously, each at 30 frames per second on a 640\*480\*24bit resolution, we were constantly utilizing less than 10% of available CPU time. The average latency on each vision system, meaning the time from the point of capture (V4L timestamp) to the point where all coordinates are being sent to the A.I. module was usually in the range of 3 to 5 milliseconds, which is an increase of almost 800% compared to our system's last year's performance.

### 3 Dynamically Layered, Behavior-Based Artificial Intelligence

#### 3.1 Noise Reduction through Independent Kalman Filters

Once after all vision data has been retrieved, it is being transported to the A.I. module. Even though our described vision system is generally precise, it is not immune to things like sensory noise or blurred images. For a further reduction of sensory noise we make use of a 3rd party bayesian filtering library, in particular its implementation of the linear Kalman filter. We apply several independent instances of Kalman filters (one for each degree of freedom of each robot). The positive effects of this filter usually become clearly visible to us as the robots' movements tend to look much smoother and more natural on our visualization system than without filtering. As many other teams (especially from the Legged league) have described in their papers, a sensitive tweaking of the filter's values is often crucial to gain any visible effects in performance [3].

#### 3.2 A Simple, Predictive World Model

The next step in our processing flow will be to synchronize both camera's observations into one common world model. Naturally, the goal will be to gain the best possible estimate of the *current* state of the world. As each of the vision data packages is already approximately 5 milliseconds outdated by the time of retrieval, some additional computation is needed to gain a better state estimate. Thanks to frame time stamping, a feature inherent to Video4Linux2 (as it comes with the Linux 2.6 Kernel), we are able to determine the elapsed time from the point of capture to the point of processing in the A.I. system. Even though both cameras run at the same frame rate of 30fps, it is generally unpredictable which image will be delivered to us first. In order to bring both datasets to the current state of the world, we need to equalize both of them to represent the same point in time, namely *now*. This is done by extrapolating each of the datasets to the current point in time. For performing this extrapolation we make use of a quite simple, predictive physics model. For the ball and our opponents we assume linear motion at their observed speeds and directions. We also take ball deceleration and solidness into consideration when we do these predictions. The movements of our own robots are predicted similarly, but with the minor addition of also taking the last movement command that has been sent to the robots into consideration. This kind of extrapolation is of course prone to an

exponentially increasing error, the further we attempt to predict into the future. But since the latencies that we need to predict usually never exceed 15ms it is well within an acceptable range and still much better than simply assuming the world from the point of video capture.

### **3.3 Variably Layered Behavior Model**

Trying to explain it in few words, one could best describe our artificial intelligence system as a hybrid of a finite state machine and an object oriented behavior-based model. All of our behavior classes have access to several helper functions such as the path planner, the world prediction system which was described in the previous paragraph, and several other evaluation routines that for example perform searches for good positions on the field.

### **3.4 Dynamic Behavior Groupings**

Unlike several other teams which often have a fairly strict separation of their behavior's levels (such as the three levels of skills, tactics, and strategy) [4] our system does not natively enforce such a division, because it often is difficult to define whether a certain behavior should i.e. be considered a skill or a tactic. Our layering is what we call dynamic: any behavior can make use of any other sub-behavior(s) that it needs to achieve its task. However, we still draw a slight line of separation, namely between the behaviors which only act upon one robot at a time (such as 'drive to point x') and the ones which could possibly be assigned to more than one robot (such as passing games or attack patterns). As mentioned before, these behaviors can be assigned recursively. For example, we could have an attack behavior which takes control of four robots. This behavior might now sub-assign a passing-game behavior to two of its assigned robots and a defense pattern to the other two robots that it controls.

The way in which we decide what behaviors to apply is currently done by simple success estimation functions that are implemented in each of the behaviors. Since some behaviors are able to take a variable amount of robots into account, there exists a certain competitiveness of how many robots are being assigned to different behaviors. Generally, each behavior will demand a certain set of possible robot combinations and provide success estimations for each of them. We then assign our behaviors on a statistically evaluated priority basis depending on the game's global situation (i.e. offense vs. defense).

### **3.5 Path Planning Components**

An intelligent path planning algorithm is generally a critical component in a RoboCup system. Since the US Open we have made use of the extended RRT algorithm as it was suggested in a paper by James Bruce from the CMU team [1]. The results of this algorithm have always been acceptable during the 2004 US Open, however we are still focusing on issues of optimization and post processing of this algorithm for a further reduction of latency and hysteresis effects.

### **3.6 Statistical Learning**

One of our considerations which has not yet been implemented is the idea of an adaptive, statistical learning algorithm which will influence behavior assignments as the game progresses (i.e. it will assign successful behaviors more frequently and unsuccessful ones less frequently). A behavior's success can be determined by various factors, but most importantly by goals scored against the opponent team (or against our own team in case of negative reinforcement). This learning algorithm will allow our system to adapt autonomously to each unique opponent team simply by changing the strategy of behavior assignments. To our knowledge, some similar approaches have already been implemented successfully by several other teams (to our believe at least by CMU [5] and possibly by FuFighters).

### **3.7 Autonomous Gyrometric Robot Control**

One of the probably most innovative features of this year's robot hardware is the on-board use of accelerometers and gyrometers for achieving a more stable drive-control. The basic idea is to provide the robot with a basic sensory system for stabilized navigation during the dead-time between two command intervals. 33 milliseconds (which is the time between two vision/command frames on our system) can be an awfully long time for a robot driving autonomously at full speed. In the past we have quite frequently observed unpredictable drifts of the robot possibly due to wheel slippage or carpet unevenness. A passive correction through better wheels and a high-power drive system (which have both been implemented this year) will certainly help against these issues, however we figured that an active intervention by the robot might lead to further significant improvements. The new system will compare the robots actual accelerations to the ones assigned by its movement command and then perform any necessary corrections to the drive system's vectors. Since this electronic stability system will be natively embedded onto the robot, it is able to perform corrections at a much higher rate than our external A.I. system ever could. Because we're still actively working on the software component of this stability control system we cannot make any claims about its effectiveness yet, but first readings from the motion sensors' data do in fact look very promising.

## **4 Acknowledgements**

Our team is being funded and supported by the Undergraduate Student Association and by the Energy Systems Institute (ESI) of the University at Buffalo. We furthermore would like to thank all of our additional sponsors: Fisher Price, Northrup Grumman, and the UB College of Arts and Sciences Instrument Machine Shop. Special thanks are also pronounced to Jennifer Zirnheld and Kevin Burke for their constant support to our club's efforts.

## References

1. Bruce, J., Veloso, M.M.: Real-time randomized path planning for robot navigation. *Lecture Notes in Computer Science* **2752** (2003) 288–295
2. Tsai, R.Y.: A versatile camera calibration technique for high-accuracy 3d machine vision metrology using off-the-shelf tv cameras and lenses. *IEEE Journal Of Robotics And Automation* (1987) 323–344
3. Freeston, L.: Applications of the kalman filter algorithm to robot localisation and world modelling. (2002)
4. Browning, B., Bruce, J., Bowling, M., Veloso, M.: Stp: Skills, tactics and plays for multi-robot control in adversarial environments. *IEEE Journal of Control and Systems Engineering* (2004) accepted.
5. Bowling, M., Browning, B., Veloso, M.: Plays as effective multiagent plans enabling opponent-adaptive play selection. In: *Proceedings of International Conference on Automated Planning and Scheduling (ICAPS'04)*. (2004) in press.