Real-Time Foosball Game State Tracking

Sven Bambach, Stefan Lee

School of Informatics and Computing
Indiana University
Bloomington, IN

sbambach@indiana.edu
steflee@indiana.edu

***Abstract*—In this report we present our project for the B657 computer vision class in the Spring 2012 semester. We implemented a system that takes as input an overhead video stream of a foosball match and, without manual assistance and in real-time, outputs the per frame relevant data to describe the state of the game: the current position and rotational angle of each player and the current position of the ball.**

I. Introduction

In recent years, computer vision techniques have found commercial and technical success when applied to the domain of organized sports; providing dynamic visualizations that aid in interpreting the full breadth of information in a given frame of a sports video stream. In designing this semester project, we wished to apply this same paradigm of information gathering and visualization to the game of foosball. Our overall goal for the project was to produce a real-time system, which would compute and display the state-of-play for a foosball game. We define the state-of-play for a foosball game as the position of the ball as well as the position and angular rotation of the player bars. As our time available to execute this project was short, we limited our ambition to a single Tornado F-5 foosball table located in a lounge at the Indiana University Informatics West building.

Related work on ball tracking for foosball has been done by Janssen et. al. at the Eindhoven University of Technology in the context of a semi-automated foosball table [1]. Their work did produce a real-time tracking algorithm for the ball but made no attempt to track the players or to provide a reasonable visualization of the collected data.

Fig. 1. **left:** The Tornado F-5 foosball table at the Informatics West building with the camera rig mounted on it. **right:** The table has eight bars with a total of 13 yellow and 13 black players for each team.

II. Data Acquisition

We made use of a Canon EOS Rebel T3i camera with an 18-55mm lens to capture the gameplay video from a fixed height and centered vantage point. In order to maintain this position, we constructed a quad-legged camera rig, which fits into the four cup-holders built into the foosball table. An image of our rig is presented in figure 1. This construct has sufficient height to capture the entire field of play.

We recorded two complete games containing 15 goals with a total length of about five minutes. In post-production, the video was cropped so that just the foosball field is visible and cut into 15 single videos; one for each goal, starting when the ball enters the field and ending at the actual goal. To reduce the volume of data, we also decreased the video resolution from 1080 lines to 240 lines, while retaining the 16:9 aspect ratio and 30p frame rate of the source video.

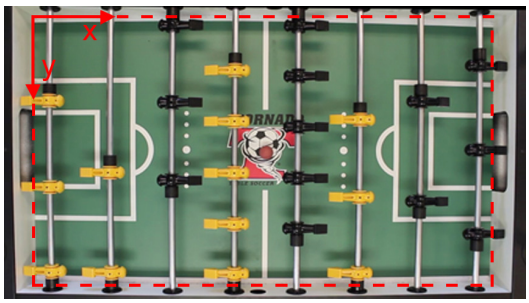

Fig. 2. An example frame of the video stream after post-production, overlaid with the image coordinate system for the foosball field.

III. Our Approach

The goal of our project can be logically divided into the following modules: we use a Hough transform to locate the bars, a color matching approach to identify players and determine the lengths and a template matching method in a small window, localized by a Kalman filter, to find the ball. We also implemented background removal to aid in the localization of the ball and to provide easier detection of occlusion. The following sections go into the details of these methods.

A. Bar Localization

The first step in our system is the localization of the eight bars on the table. To do this, we use a Hough transform line

detection algorithm, supported by some prior knowledge about the table. First, when creating the edge map, we only look at the horizontal derivatives of the image, as we know that all bars are vertical. Second, when searching for lines, we ignore a certain percentage of the center part and both the left and right ends of the image in order to suppress potential noise from the field lines and table edges. We then claim the eight highest peaks in the Hough space to be our bars. As the bars have a certain width and could potentially be somewhat bent due to camera distortion, we need to slightly adjust the initial line parameters to ensure that the lines we describe are as well centered on the bars as possible. In addition to the obvious increase in accuracy gained for the step, better fitting lines also ease the problem of player localization and rotation estimation in later stages. We improve the fit by defining an error function for every potential vertical line within a close neighborhood of the original line, where we check how much each pixel of the line differs from the grey value that we expect the bar center to have. We then pick the line with the lowest median difference.

As the Hough transform is computationally expensive, we only perform it once based on a "calibration image" of the table. This means the bars are located before the first frame of the actual video stream is analyzed and we assume that the bars will not change their position during the video. As the camera rig is attached to the table, we feel that this is a reasonable assumption.

B. Player Detection

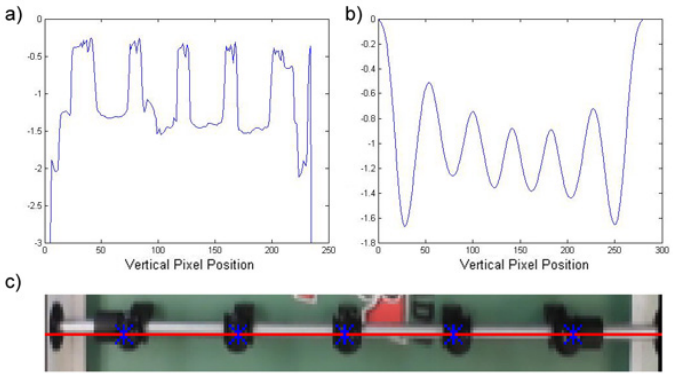

Fig. 3. The raw distribution in a) is very noisy and finding a peak proved difficult. In b) the smoothed distribution has five very clear peaks. The peak locations have been plotted as blue stars on the corresponding bar in c).

Using the bar locations calculated in the previous step, we localize the players on each bar. To accomplish this for each bar we take a 10 pixel wide vertical strip centered on the bar location and for each row of the strip compute the negative mean distance between the pixel colors in the row and the known player color. This results in a distribution of the likelihood of each row containing a part of a player. The distribution is very noisy however and varies in magnitude due to specularities and motion (see figure 3a), so we convolve it with a Gaussian kernel to produce a distribution with well-defined peaks (see figure 3b). These peaks (except for an offset due to the smoothing) are the vertical center locations of the players on that bar and can be extracted in linear time. We perform this algorithm for each bar each frame.

C. Rotation Estimation

The basic idea behind the rotation estimation is to determine the rotation angle of each player trigonometrically based on its projective length in the image.

We already determined the center point of each player along a bar, as described in section III-B. Now, for each player p, we sample a one pixel wide and eleven pixel tall strip from the center point of the player and save the values as $\mathbf{x^P_{ref}}$. We then iteratively sample the same sized strip from consecutive columns to the left and right and save them as $\mathbf{x^P_{left}}$ and $\mathbf{x^P_{right}}$ respectively; each iteration moves the sample areas one column further away from the center. During each iteration we check how likely it is that the sampled pixels still belong to the player. We do so by first calculating the delta terms for each player individually:

$$\Delta^p_{left} = \|\mathbf{x^P_{left}} - \mathbf{x^P_{ref}}\|_2 \quad (1)$$

$$\Delta^p_{right} = \|\mathbf{x^P_{right}} - \mathbf{x^P_{ref}}\|_2 \quad (2)$$

Then we determine the delta terms for the whole bar as

$$\Delta^b_{left} = \sum_{i=1}^{N} \frac{\Delta^p_{left_i}}{N} \quad (3)$$

$$\Delta^b_{right} = \sum_{i=1}^{N} \frac{\Delta^p_{right_i}}{N} \quad (4)$$

where N is the number of players per bar. If Δ^b_{left} or Δ^b_{right} is above a certain threshold, we stop sampling further for that direction. Once we stop iterating in both directions, we assume the current distance between our endpoints to be the length of the player. Estimating the player length based on this global observation rather than looking at each player individually decreases the sensitivity to noise and is a reasonable approach, as the projective lengths of all players that belong to the same bar should be approximately equal.

Once we have the player length l_b for a bar, the corresponding rotation angle is given as:

$$\theta_b = |arcsin(\frac{l_b - l_{min}}{l_{max} - l_{min}})| \quad (5)$$

where l_{min} and l_{max} are predefined values that we manually measured in advance. With respect to the image coordinate system, we define a rotation to the right to be positive and to the left to be negative. We can easily determine the sign of the rotation angle by comparing how far we moved to the left and to the right while estimating the player length.

We do not make special considerations for rotational angles that would place the player foot above the horizontal plane and as such, our minimal and maximal angle measures are -90 and 90 degrees. If such a rotation were to occur, our system would view it the same as if the player was headed back to zero degrees.

D. Ball Tracking

To track the ball, we essentially use a color template matching method in combination with background masking and Kalman filtering. We use the Kalman filter to both improve the accuracy of our measurements and constrain the search space for the ball to a local window based on the state prediction of the filter.

1) Background Masking: To simplify the task of our template matching algorithm, we initially try to eliminate everything in the image that is not the ball by replacing it with a cyan colored mask. We choose a cyan mask for the stark visual contrast and high red channel difference with respect to the red(ish) ball. We accomplish this by overwriting every pixel in the image that is outside the interval $[R_{ball} \pm \Delta_R, G_{ball} \pm \Delta_G, B_{ball} \pm \Delta_B]$, where $[R_{ball}, G_{ball}, B_{ball}]$ is the reference color of the ball. The background-masking step is also essential for handling situations where the ball is occluded, as will be explained in section III-D4. An example of the masked image can be seen in figure 4.

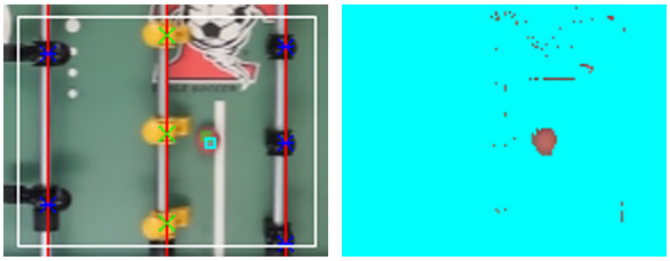

Fig. 4. **left:** The white rectangle indicates the search window for the ball, which is slowly moving towards the left. The light green rectangle indicates the measured position of the ball while the cyan rectangle indicates the updated position estimation of the Kalman filter. **right:** The corresponding masked image on which the template matching is performed.

2) Template Matching: As a part of the tracking scheme we need to locate the best match for the ball within some search window (the details of how this window is selected are given in section III-D3). Our template is a 5-by-5 red square which is smaller than the ball. We use this template because it is robust to partial occlusions and the bright red color will match horribly with the cyan background masking discussed above, allowing us a sizable difference between occlusion and non-occlusion states. We only examine differences in the red channel when evaluating the fitness of the template at a location. We do this as a time efficient matching strategy that works well in practice due in no small part to the cyan background masking. To find the best fitting location of the template within the region we calculate the sum of absolute differences of the red values between the template and each possible template location in the region.

3) Kalman Filtering: Our raw observations of template location might be poor so we turn to a Kalman filter to model the linear dynamics of the moving foosball. We define the state of the ball as its position and velocity which we describe at time t as a normal distribution about the column-vector $S_t = [x, y, v_x, v_y]^T$ with covariance H_t. We borrow from the basic kinematics equations to obtain a state prediction matrix as:

$$A = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 0 & drag & 0 \\ 0 & 0 & 0 & drag \end{bmatrix} \tag{6}$$

In our work the fundamental unit of time is 1 frame, so all speeds are pixels per frame. The $drag$ value is set to 0.85 in our experiments and indicates a 15% velocity loss per frame. As we are only observing the position our observation matrix is simply:

$$C = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \tag{7}$$

We experimentally derived a prediction noise matrix as:

$$Q = \begin{bmatrix} 0.2 & 0 & 0 & 0 \\ 0 & 0.2 & 0 & 0 \\ 0 & 0 & 0.2 & 0 \\ 0 & 0 & 0 & 0.2 \end{bmatrix} \tag{8}$$

Using these matrices we can calculate a predicted state using the standard Kalman equations [2] as:

$$S_{t_{pred}} = AS_{t-1} \tag{9}$$

$$H_{t_{pred}} = AC_{t-1}A^T + Q \tag{10}$$

We use this predicted state to designate the search space for our template matching. We first center a 40-by-40 square on the predicted location and extend this box in the direction of the ball's movement as a linear function of the predicted velocity. Using the template matching we get an observation of the state:

$$Z_t = \begin{bmatrix} z_x \\ z_y \end{bmatrix} \tag{11}$$

In addition to this point, the template matching also returns the associated error value. We use this error value to calculate an estimate for the observation covariance as:

$$R = \frac{error}{100} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{12}$$

The denominator was derived experimentally but it makes intuitive sense. As a good match for the template reports an error of around 10 and a poor match moves towards 40, this divisor provides low or high covariance in response to the quality of the template match. With this final set of matrices we can compute the final Kalman filter state estimate as:

$$k = H_{t_{pred}}C^T(H_{t_{pred}}C^T + R)^{-1} \tag{13}$$

$$S_t = S_{t_{pred}} + k(Z_t - S_{t_{pred}}) \tag{14}$$

$$H_t = (I - kC)H_{t_{pred}} \tag{15}$$

4) Occlusion Handling: The template matching procedure, as described in section III-D2, always provides an error term that directly tells us how certain we are that whatever we found as the best location is actually the ball. Due to the cyan background masking, this error term should be fairly large whenever we do not find the ball due to occlusion. This allows us to introduce an error threshold to decide whether the ball is occluded or not. We adjust our threshold as a linear function based on the ball velocity. This adaptive threshold allows for very sensitive occlusion detection for slow moving, non-blurred balls and a looser condition at higher velocities to account for the higher error of the motion blurred balls.

Once we detect an occlusion, we stop updating the ball state (as described in section III-D3), i.e. we assume that the ball has the same position and velocity as in the previous frame. This is a reasonable assumption, as there are no large-scale spatial occlusions possible (in fact, the only sources of occlusion are players or bars), so it is unlikely that the ball will be occluded for a long period of time while at the same time drastically changing its position or velocity.

IV. Results

We implemented our approach in MATLAB Simulink 7.7 (R2011a) with the Computer Vision Toolbox installed. We made extensive use of the built-in modules for result visualization (see figure 5). All of our experiments were run on a personal laptop computer with a 2 GHz Intel Core i7 processor, 4GB of DDR3 memory, and a AMD Radeon HD 6490M 256MB graphics card. In this environment our application ran at an average of 27 frames per second. We feel that this speed performs at near real-time and demonstrates no significant algorithmic hurdles for real-time given upgraded hardware. In the following sections we provide both qualitative and quantitative analysis of our work. Additionally, the complete video footage of our results can be found online [3].

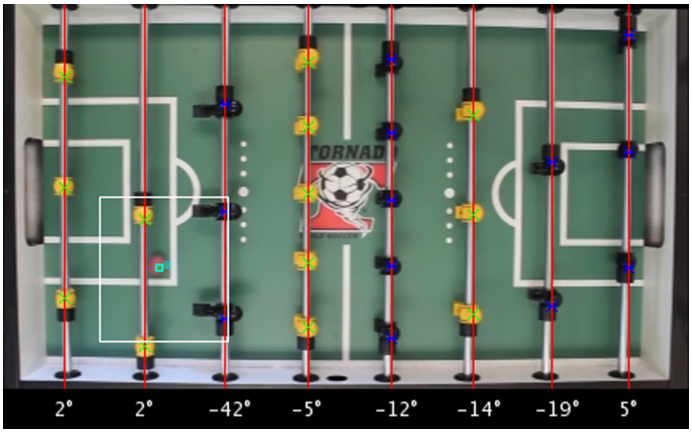

Fig. 5. A screenshot of our game state data output: Red lines mark the bars. Green crosses and blue stars mark the location of the detected yellow and black players respectively. The rotational angles are shown at the bottom of each bar. The white rectangle indicates the current search window of the ball. Semi-transparent blue and green squares indicate predicted and measured ball position, while the cyan bordered square marks the ball position estimation.

A. Player Detection

To evaluate the performance of our player detection algorithm, we did a manual frame-per-frame inspection and defined a player as not found if its position marker misses the player. It turns out that, for all 15 videos, we do not miss any of the players in any frame. We accredit this success to the highly constrained nature of the player locations along the bar and the wide disparity between the player colors and the field. We do not, however, always detect the exact center of each player along the bar. This issue mainly affects the outer black players, as each one is close to a black stopper that resembles the player.

B. Rotation Estimation

We provide some qualitative analysis and general qualitative results for our rotation estimation approach. We do not provide qualitative results because we found producing them to be troublesome, as we had no ground truth and extracting estimates from the video would require tedious pixel counting to determine player length for every bar on every frame. In addition, any method involving player length including our method presented here suffers from a discretization of the arcsine function based on video resolution. If we were to take hand calculations based on pixel counting, this would improperly inflate our real world accuracy figures, as the discretization would exactly line up with that of our approach. In figure 6 below we plot the arcsine value for each possible player length ratio in our videos. This discretization causes uneven jumps between sample points with magnitudes ranging from three or four degrees up to a gap of seventeen degrees between a player length of 24 and 25.

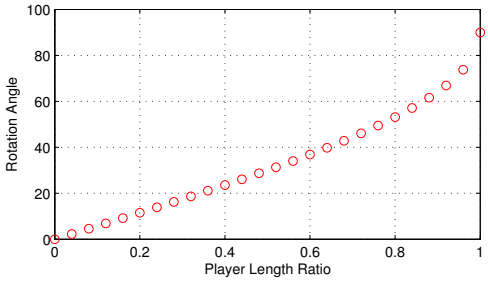

Fig. 6. Rotation angles for all possible discrete player lengths.

Qualitatively we found the results to be fairly good with some noise due to the video quality and this discretization problem but few glaring failures presented themselves and we feel it acts as a reasonable estimate.

C. Ball Tracking

To evaluate our ball-tracking performance, we define two kinds of criteria:

First, whenever we closely miss the non-occluded ball based on a manual frame-per-frame inspection, we consider this to be a minor loss and declare the frame to be a failure. We

measure the accuracy for each video as the ratio of successful frames over total frames, as shown in table I.

Second, whenever the ball exits the search window and we lose track of it entirely, we consider this to be a major loss. In all videos, we had one major loss during goal six of the first game (see table 1). For this case, the accuracy is expressed as the ratio of successful frames over the number of frames until we lost the ball (380).

TABLE I
BALL TRACKING ACCURACY

Video		**Total Frames**	**Failed Frames**	**Accuracy**
Game	**Goal**			
1	1	239	12	0.950
1	2	162	3	0.981
1	3	888	8	0.990
1	4	546	9	0.983
1	5	201	3	0.985
1	***6***	***380***	***9***	***0.976***
1	7	885	14	0.984
1	8	819	27	0.967
2	1	200	3	0.985
2	2	1379	23	0.983
2	3	314	2	0.994
2	4	485	16	0.967
2	5	62	3	0.952
2	6	566	7	0.988
2	7	761	7	0.991
total		7887	146	0.981

The total loss is due to a combination of occlusion and rapid acceleration changes. In the frames leading up to the failure, the ball is traveling upward in-between the third and forth player bars. It is then occluded by the very end of the black player to its left, which hits it and accelerates it to the left at a high velocity. The ball is moving so quickly that in the frame that it exits the occlusion it approaches the next player bar and is well outside the search region.

V. CONCLUSION

We both enjoyed working on the project but found some of our initial time estimates to be poor with portions of the project requiring far more attention than we had previously assumed. Our unfamiliarity with Simulink presented a rather steep learning curve and resulted in periods of time when our algorithm ran well below real time due mainly to poor simulation configurations and costly display mechanisms.

All in all the final output of our project matches what we initially imagined quite well in terms of accuracy, visualization, and speed, so we are fairly pleased.

REFERENCES

[1] Rob Janssen, Jeroen de Best, and Rene van de Molengraft *Real-Time Ball Tracking in a Semi-automated Foosball Table* RoboCup 2009, LNAI 5949, pp. 128-139, 2010

[2] Tristan Fletcher, *The Kalman Filter Explained*, December 2010

[3] http://www.youtube.com/watch?v=Q80WH4OIqJM