

FireVision Image Processing Pipelines

Tim Niemueller

September 20, 2007

Abstract

Besides all the infrastructure code needed to make the application work the real functionality for each vision is encapsulated in a processing pipeline. Here we are going to describe the three pipelines of our most important vision programs.

1 General

In this section we are going to describe some basic elements that we will see in all processing pipelines.

1.1 YUV Colorspace

At the very beginning of the creation of FireVision it was decided that the YUV colorspace would be used. This was done because it has some advantageous side effects to use YUV. Firstly since a sub-sampled YUV 4:2:2 planar format is used the images are smaller in memory than if RGB would have been used (saves one third). Secondly it supplies the gray scale image without the need of a transformation (just ignore U and V chrominance planes). Last but not least the original camera supplied only YUV images and images would have had to be converted to RGB.

YUV uses three values to define a pixel: a luminance (brightness) value Y, and two chrominance (color) values U and V. The YUV color model originates from the appearance of color TVs. The task was here to supply colored TV while still providing the b/w image for the older TVs. This explains why you can just use the luminance plane and get the b/w image. The U/V values are just “add-ons” for color support.

1.2 Colormap

The colormap is a mapping from the U/V color values to an object ID. Throughout the software colormaps up to now have only been defined for one object, so we have two possible IDs: background and object. Since the colormaps originally described the ball we use grey as the background color and orange as the object color.

Colormaps are trained offline with a bayesian approach. As input data images from the camera are transferred to a laptop besides the field to an application called FireStation. There with the fuzzy select tool (“Zauberstab”) you select regions in the image that belong to the desired object. The image with the selection is then passed on to the colormap learning process.

1.3 Classifier

The classifier is used to analyse an image for interesting regions. These interesting regions are identified by looking for a specified object color. The classifier gets an image, a scanline model and a color model. The scanline model is determined by the image processing application and it defines a set of points that shall be analysed. Several models have been implemented (see below). The color model can be any arbitrary model mapping U/V values to object IDs. Throughout the software only colormaps are used as color model.

The output of the classifier is a set with zero, one or more regions of interest (ROI). If no ROI could be found the object cannot be detected in this cycle. ROIs are defined as a rectangular region of the image with the upper left as starting point (x, y) , width, height and a weight. The weight is the number of scanline points in the ROI identified to be of an object color. The list of ROIs is ordered by the weight with the highest weight as the first entry.

2 Pipelines

The image processing is done in so-called pipelines. It is a list of tasks done strictly sequential to get the desired data from the image. Although there are many vision applications already implemented we going to describe the most important three: the front, omni and stereo pipelines. I will describe them in the chronological order in which we developed them. As you will see there are many similarities between the pipelines which allowed us to develop the infrastructure once and then concentrate on the real work.

2.1 Front Vision

The front vision has been used on our old robots. It was used to detect the position and velocity of the ball. It uses the color and shape of the ball to detect it and estimate its position. From

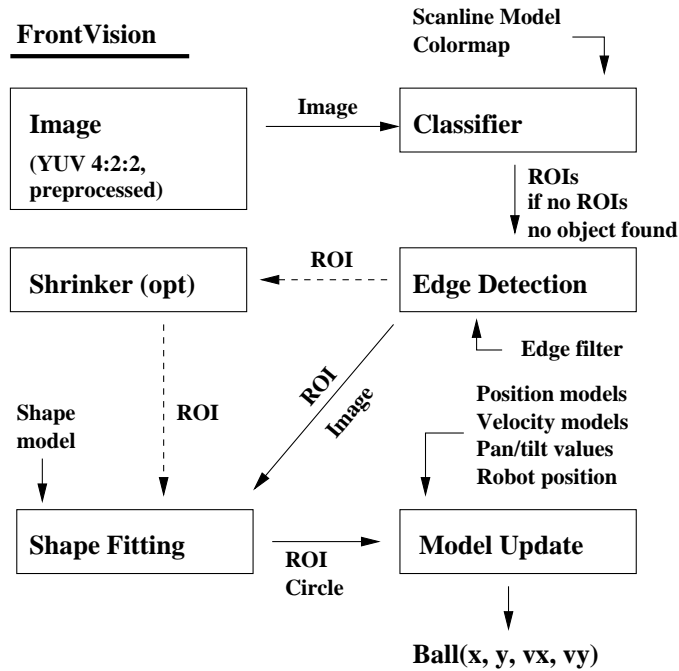


Figure 1: Diagram of the front vision

these positions the velocity can be derived by comparing values over time. One of the major assumptions is that the ball is only rolling on the floor. If the ball is kicked up the height is not calculated. An interesting feature of this pipeline is that it uses the pan/tilt-unit of the camera. This way the ball can be tracked around the robot with a viewing angle of about 210° . A problem which arises from this is the occlusion of objects by the viewing robot itself. This has been solved for using shape recognition to be able to detect only partly visible balls.

2.1.1 The Procedure

The image is gathered from a Sony EviD100P camera which produces PAL images of the resolution of 768×576 pixels. The image is acquired using the Leutron frame grabber available in the LvmPCs in the robots. The images are then converted from the read YUV 4:1:1 packed format images to YUV 4:2:2 planar format.

These images along with a scanline and a color model are then fed to the classifier. The scanline model for the front vision is a scanline grid. It consists of a 5×5 grid. Each intersection of the lines is then considered by the classifier. The supplied color model is a colormap trained earlier.

Only the first ROI produced by the classifier is processed since there is not enough time to process more as we are going to see. The first ROI is then passed to an edge filter. Many experiments have been done to find a good combination of filters. But it turned out that the classic filters like Laplacian of Gaussian and Sobel filters were not good enough. If more

directions of the linear filters were used the results got better but these were too slow. Eventually we came up with the idea of exploiting the color information we have, again: we just search from the border of the ROI to the inside (from left/right and top/bottom to the center. At the very first occasion where there is a flip from the background to the object color we mark it as an edge pixel. This way we get a pretty good and almost noise-free result. The edge filter outputs the ROI and the image with the ROI processed.

This can then optionally be passed to a so-called shrinker. This shrinker takes the ROI and shrinks it as needed. This is for example used to get rid of reflections that we had on the lobby floor during testing. This is done by shrinking the ROI to the biggest quadrat that fits into the rectangle.

Either from the shrinker or from the edge detection directly the processed ROI is passed to the shape fitting. The shape fitting is used for verification of the result if the object is in this ROI or if there was something else that was just of the same color. Secondly the fitting is used to gather data needed to calculate the position of the ball in the next step. In this pipeline we used a circle fitting for the round shape of the ball. After evaluating several approaches like Hough Transform and Randomized Hough Transform we found Randomized Circle Detection ([CC01]). For this algorithm from the set of edge pixels four pixels are drawn randomly. Through the first three pixels a circle is fitted using least squares fitting. The fourth pixel is a control pixel. Only if this pixel is close enough to the fitted circle this circle is considered valid. If a circle is valid the average distance of all pixels to the circle is calculated. If it is below a given threshold the circle is taken into the result set. Computation of all combinations would take too long. So to fit in our desired real-time constraints we implemented RCD as an any-time algorithm by stopping the processing after 10ms and taking the best result then available. During testing it became obvious that we get the best (and most stable) result if we always take the biggest fitted circle.

This circle is then fed to the position and velocity models. From the diameter of the circle and the position of the center point in the image we calculate the relative position of the ball to the robot which can then be transformed into the global coordinate system.

2.2 Omni Vision

The omni vision started as an experiment using Apple iSight firewire cameras and light bulb with silver coating on the top. The camera points upwards towards the bulb which acts as a mirror and thus gives us a 360° round-view.

2.2.1 Mirror Model

The omni vision uses a so-called mirror model. The mirror model is used to map the distorted image coordinates to world coordinates. Since the mirror has a bulb shape which is conic there is no easy mapping. Especially for the used light bulb there are many irregularities in the surface

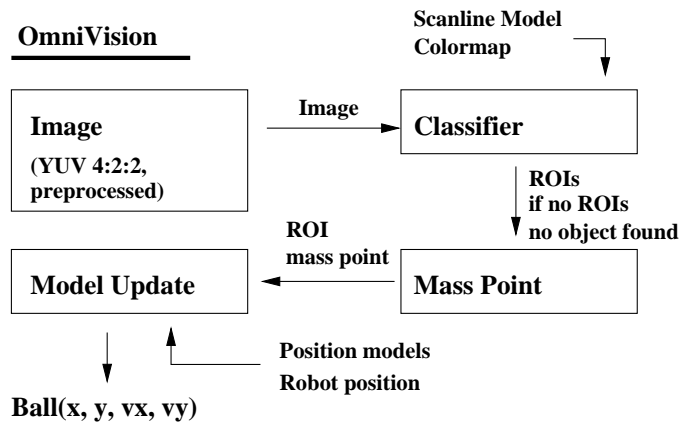


Figure 2: Diagram of the omni-vision

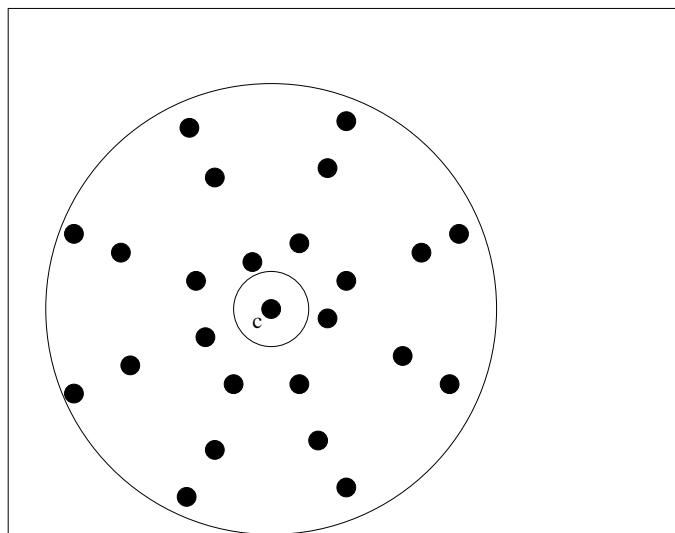


Figure 3: Schematic example of a raw lookup table. The black points indicate the sample points.

which make a measured mapping needed. This mapping has been implemented as a lookup table mapping an image pixel (x, y) (cartesian pixel coordinates) to the world coordinate (d, θ) (polar world coordinates).

The mirror model is generated by using the localization of the robot to move it on a straight line on the field to four to six places (usually side line from the corner of the field to the middle line). At the end of this line (intersection side line and middle line) the ball is placed detectable by the vision at each position. Then the robot visits each place and turns at each place one time around in eight steps. At each step an image is taken and the ball detected. This results in the sample point set that you can see in figure 3. Then four neighbouring points are used to interpolate all points enclosed by the trapezoid of these four points.

2.2.2 The Procedure

The acquired VGA (640×480 pixels) is transformed to the needed colorspace and piped through the classifier. This is the same as for the first steps in the front vision.

The processing is done on the distorted (“warped”) image. No intermediate rectified image is calculated during the process.

Again only the first ROI of the ROIs resulting from the classifier run is processed. Here not because of insufficient processing time but because no good quality measure is at hand to decide on the ROI besides the ROI’s weight. First all object colored pixels in the ROI are marked. Then the mass point of these marked pixels is calculated. This pixel is already considered to be the ball position in the image.

Then using the mirror model calibrated before starting the omni vision this pixel coordinate is transformed to the world coordinate of the ball’s position. Note that here the general assumption is that the ball is lying on the ground and not in the air. A flying ball will result in an error of a far-away looking ball.

2.3 Stereo Vision

The stereo vision is the youngest image processing pipeline and has been used for the first time at the German Open 2007 in Hannover. It uses a Point Grey Research, Inc. Bumblebee 2 camera which provides two pre-calibrated images with a fixed distance. The Triclops software development kit (SDK) has been used to calculate the disparity (depth) image from the two images. For this vision for the first time a lot faster machines have been used (Intel Core2Duo 2 GHz) which made it possible to run this in a timely manner. We are not going to describe the stereo processing in detail here.

The stereo vision has been used to determine the position of a cup which was then grasped with a manipulator.

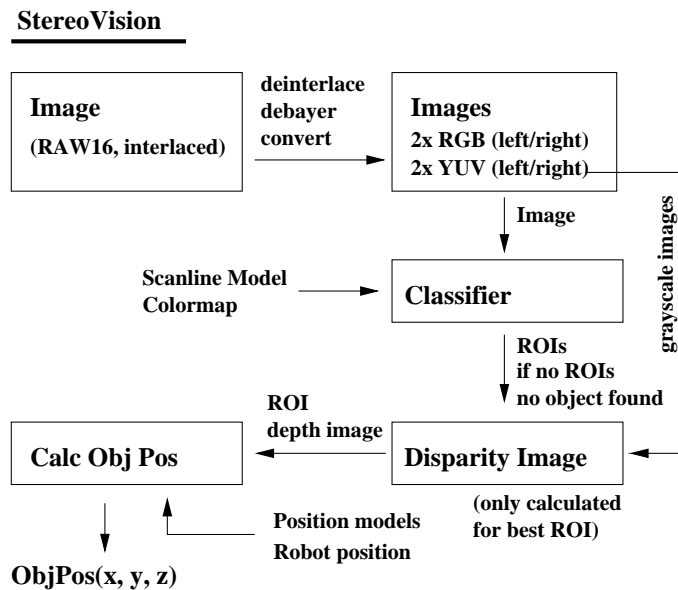


Figure 4: Diagram of the stereo vision

2.3.1 The Procedure

Once again the beginning is very similar to front and omni vision. The image is acquired, pre-processed and then fed to the classifier. Here the preprocessing involves a little bit more since the two images have to be deinterlaced, then calculate RGB from the Bayer pattern images and convert these to YUV images.

The first ROI is then processed to calculate the disparity image. An improvement for the RoboCup 2007 in Atlanta was to only process the ROI and not the full image, which made the overall process three to four times faster in average (now resulting in 15 to 20 frames/second). The disparity images gives information about positions in 3D space. The resulting image is a gray-scale image where different shades of gray denote different distances from the camera. There are also regions where no disparity image cannot be calculated because correlation of edges fails.

Then a new scanline model is used to take pixels spread over the whole ROI. For each of these pixels the corresponding point in camera coordinates is taken. The list of these points is then ordered by the distance from the camera and the median is taken (which was done to provide more stable results). This position is then transformed to world coordinates and the task is done.

In Atlanta we even did this detection task several times (about 10 times) and then averaged the results to get even more stable results. This was needed to have the robot rotate correctly to grasp the cup.

References

- [CC01] Teh-Chuan Chen and Kuo-Liang Chung. An Efficient Randomized Algorithm for Detecting Circles. *Computer Vision and Image Understanding*, 83:172–191, 2001.