SAPHIRA TUTORIAL

Using the ACTS Vision System in Saphira
Software version 8.0 (Saphira/Aria)
September 2001

©Kurt Konolige
SRI International
konolige@ai.sri.com
http://www.ai.sri.com/~konolige

Contents

1 Introduction

This tutorial introduces the ACTS Vision System interface to Saphira. ACTS is a high-performance color blob tracker that works with standard color cameras, developed by Paul Rybski at the University of Minnesota, and distributed by ActivMedia Robotics. ACTS runs as a standalone process, and can be used to train the color tracker, and to report color blob information to other processes; please consult the ACTS manual for more information. The Saphira interface to ACTS can open a connection to the ACTS server, and import information from the color blob tracker. The source and makefile for the sample object file can be found in the `Saphira/tutor/sfacts` directory.

In addition to material directly related to connecting to the ACTS server, this tutorial also discusses some issues in visual perception, including using color blob tracking to determine to position of objects relative to the robot.

2 Saphira Interface to ACTS

Saphira has a loadable module that performs the basic housekeeping for connecting to ACTS, and receiving color blob information. The information comes in every sync cycle (100 ms), and is deposited into data structures that can be accessed by other Saphira programs, or with Colbert.

To facilitate debugging, and for an example of how to use the blob information, the largest blob on color channel 1 is tracked in the LPS with a point artifact.

2.1 Starting ACTS

The Saphira interface requires the ACTS server. The ACTS server can be started at any point, but Saphira will not be able to connect to or retrieve any color blob information before ACTS is started.

Please see the ACTS manual for information about running ACTS. Saphira does not provide an interface for loading parameter files into ACTS; this must be done at startup time for ACTS, for via its GUI.

2.2 Loading the sfActs Tutorial File

In the Saphira distribution, the `sfacts.so/dll` file is already present in the `lib/` directory. Load it with the Colbert command:

```
loadlib sfacts;
```

Loading the library will automatically create a micro-task for communication with the ACTS server, and for drawing on the LPS the largest blob on color channel 1.

The tutorial can also be compiled; go to `tutor/sfActs`, and run the `makefile` or MSVC++ project for the program. The library is deposited into `lib/`.

2.3 Connecting to ACTS and Retrieving Blob Info

Once `sfacts.so/dll` is loaded, Saphira will connect to an ACTS server running on the same machine, if the `sfActsConnectFlag` is set to 1, from either Colbert or C++ code. The sfActs microtask will check this variable, and try to open a connection to the ACTS server. If it does not succeed, a message will be printed on the Colbert console, and the variable will be set back to 0. Typically, the only reasons not to succeed are if the ACTS server is not started, or if another program is connected to it.

Once Saphira is connected to ACTS, it receives information back about all the color channels that ACTS is monitoring. The sfActs microtask will place this information into an array of `blob` structures, available to other loaded program within Saphira, and to Colbert.

The format of the `blob` structure is:

```
typedef struct blob_str
{
  int area;
  int xcg, ycg;
  int left, right, top, bottom;
} blob;
```

Each blob structure contains information about a connected color blob that ACTS found in the most recent image. The fields in the structure describe the image geometry of the blob. For reference, the upper left corner of the image is at the coordinates (0,0). X image coordinates increase to the right, and Y image coordinates increase down. Figure 2-1 shows how the blob structure describes a typical blob. The area of the blob is defined by the number of pixels that make up the blob.

ACTS defines 32 color lookup channels, by which 32 separate color types may be tracked. Within each channel, ACTS segments the image into *blobs*, that is, regions that have the color characteristics that the channel has been trained on. Saphira imports the 10 largest blobs for each channel, putting them into an array of blob structures.

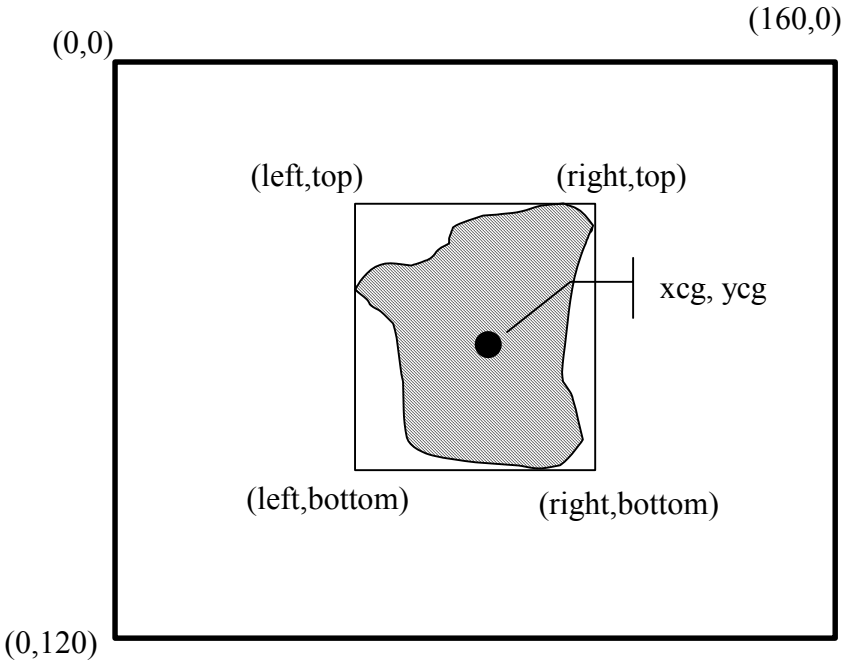

Figure 2-1 Blob image geometry. The bounding box of the blob is given by the left, right, top, bottom fields (in pixels). The center of gravity is (xcg, ycg). The image coordinates originate from the top left-hand corner of the image.

There are a number of functions and variables for getting the blob structures from the array. The following table summarizes them.

C++ and Colbert Function or Variable	Description
`blob * sfGetBlob(int ch, int n)`	Returns the `n`th blob on channel `ch`.
`int sfNumBlobs(int ch)`	Returns the number of blobs on the channel `ch`.
`int SfBlobValid(int ch, int n)`	Returns 1 if there is a valid blob on channel `ch` at index `n`.
`int sfActsConnectFlag`	Set to 1 to connect to ACTS server, 0 to disconnect.
`int focalLength`	Focal length of the lens, in pixels. This parameter is used by the draw() function to determine object distance and angle.
`int objectWidth`	Width of the object, in mm. This parameter is used by the draw() function to determine object distance.
`int imageWidth, imageHeight`	Width and height of the image, in pixels. Default is 160 x 120.

3 From Image Blobs to World Positions

Unlike sonars, image sensors do not give direct information about range to objects in the world. Instead, a vision sensor maps the 3D world into a 2D perspective projection. Objects along the same line of sight through the viewpoint of the image sensor all appear at the same image position, no matter what their distance from the sensor.

In this Section we discuss several ways to convert image information into 3D information. All of these methods involve knowing something about the object being imaged; for example, its height or distance from the ground plane. In addition, we have to know the *internal camera parameters* --- the parameters that are relevant to forming an image in the camera.

3.1 Ideal Pinhole Camera

We treat the imaging system as an ideal pinhole camera, where an image is formed on light-sensitive imaging plane by the action of a pinhole aperture. Figure 3-1 shows the relevant geometry, for a plan view of the camera and object.

From the figure, we can write down the relationship between the object 3D values and the image values:

$$\frac{\Delta x}{f} = \frac{W}{R} \qquad (1)$$

Since we are interested in how far away the object is, we rearrange this equation to isolate *R*:

$$R = \frac{fW}{\Delta x} \qquad (2)$$

To determine the range, we have to know the actual width W of the object; we won't be able to find the range to unknown objects. The focal length *f* is an internal parameter of the camera; we'll discuss how to estimate it in the section on *calibration* below.

For Equations 1 and 2, we assumed that one edge of the object was on the optic axis of the camera. However, this assumption is not necessary: as long as Δx is the image size of the object's width, it doesn't matter where the object is.

We can make another generalization. While we have looked at a plan projection in Figure 3-1, the same geometry holds for any planar projection, so we can use any linear feature of the object with known size: height, width, diagonal, etc. The relationship expressed by Equation 2 still holds.

What about the area of object image? Consider a rectangle of whose height and width are H and W,

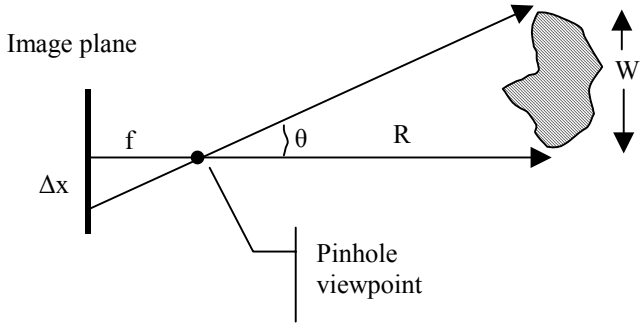

Figure 3-1 Pinhole camera geometry. Light is focused through a pinhole to fall on the image plane. An object of width W at a distance R projects onto a line of width Δx in the image. The focal length of the camera is f, and the line through f, perpendicular to the image plane, is called the *optic axis*.

respectively. Then, from Equation 2, we can write down:

$$R^2 = \frac{fW}{\Delta x}\frac{fH}{\Delta y} = \frac{f^2 A}{a} \qquad (3)$$

so that:

$$R = f\sqrt{\frac{A}{a}} \qquad (4)$$

Here, the range depends on the square root of the ratio of the known area of the object to its projected image area. Equation 4 holds for an arbitrary shape area.

Another useful quantity to measure is the angle at which an object appears, relative to the optic axis. Actually, we have to measure the angle for the *edge* of the object, or some other characteristic point such as the center of the object. For this, we don't need any object characteristics at all. Referring to Figure 3-1, the angle θ is given by:

$$\theta = \tan^{-1}\left(\frac{\Delta x}{f}\right) \qquad (5)$$

In this equation, we are measuring the angle from the optic axis, and Δx is the image distance from the center of the image to the project object point.

3.2 Calibration

Calibration is the process of determining the parameters of the camera. For the ideal pinhole camera, the only parameters we need are the focal length and the size of pixels. Real cameras, with non-ideal lenses, have more parameters: lens center and lens distortion are the important ones. We will assume we can ignore these, and that our camera is close to the pinhole model.

3.2.1 Focal Length

One thing to note is that, in the above Equations, the focal length *f* always appears in a ratio with an image distance, e.g., $f/\Delta x$. Instead of determining pixel size and focal length separately, we will just find the focal length in units of pixels. Even though we won't know the absolute size of the focal length or pixels, we can compute range just by knowing their relative size.

We can use the basic Equation 1 for calibration. Take a ruler, and place it so that it exactly fills the image horizontally. Then measure the distance from the camera to the ruler. We now have three of the four variables of Equation 1, and can determine the focal length *f* in terms of the number of pixels.

There is one further complication: the pixels of the imaging system may not be square. And even if they are square, the framegrabber may introduce stretching or shrinking of the pixel width and height, as it digitizes the signal from the camera. We can generalize Equations 1-5 to take this into account, by subscripting the focal length with the direction, *x* or *y*, of the object feature we are working with.

For the built-in camera of the Sony C1VN PictureBook we use in Stanford's CS225B Robot Lab, the focal length is approximately 175 pixels, in the *x* direction. In the *y* direction, a slightly larger focal length was obtained, but it was within 2% of the *x* direction, and so it was deemed not to matter, and a single focal length suffices. The field of view of the PictureBook camera is approximately 50 degrees in the horizontal direction.

3.2.2 From Camera to Real-World Coordinates

In Equations 1-5, the range and angle to an object are computed relative to the camera viewpoint and optic ray. If the camera is located in the center of the robot, and the optic ray points straight ahead, then

the range and angle are also in robot-centric coordinates. But, it the camera is offset from the center of the robot, or pointing in a different direction, then a further transformation is needed from camera-centric coordinates to robot-centric coordinates. Then, given the robot's real-world pose, a final transformation to real-world coordinates can be made.

3.3 Finding Good Features

One of the practical problems in going from 2D color blobs to 3D properties such as range and angle is finding good features for this purpose. In most cases we can restrict the environment so that some features will be suitable.

3.3.1 Feature Properties

A good feature should have these properties:

1. View independence. No matter what angle the object presents itself to the camera, the feature should have the same value.
2. Unambiguous. The only objects with the given color should be the ones we are interested in.
3. Lighting independence. The feature should be stable under a variety of lighting conditions.

There are several types of object features that can potentially satisfy these criteria. Obviously, objects with heavily saturated colors are the best: neon green and orange work particularly well. Objects with very bright specular reflections are a problem, because these bright spots tend to show up as the color of the light, rather than the color of the object.

Object shape is also very important. Planar objects are very convenient. A rectangular sheet of colored paper makes a great landmark. Assuming the camera always point parallel to the floor plane, putting the sheet on a vertical wall means that the height of the sheet is invariant, no matter what angle is chosen. The *x* center of gravity, or one of the vertical edges, can be used to judge an angle to the object.

Another good object is a spherical ball, which has the most symmetry of any object. The width or height of the ball is a great feature to use. In general, for balls on the floor, the width tends to be a better feature, because sometimes the bottom of the ball is in shadow, and only the upper half of the ball is recognized as a blob. The *x* center of gravity is a good feature for judging the angle to the ball.

For several reasons, image area is not a good feature to use for planar and spherical objects. Planar objects can change their area, depending on the view angle. Spherical objects suffer from poor lighting of the bottom of the sphere (with overhead lighting), leading to fluctuations of the image area as the ball moves to places with different lighting.

Vertical cylindrical objects, however, lend themselves to good area features. Their area is view independent, and they do not suffer from the same shading problems as a spherical object.

3.3.2 Aperture Problem

When an object is projected near the edge of the camera image, some part of it can be cut off. This phenomenon is referred to as the *aperture problem*. It can lead to mistaken estimates of range and angle. For example, suppose we are using the width of a spherical ball as the feature for determining range. If the camera sees only half the ball at the edge of the image, then using the image width of the ball to calculate the range of the ball will give an answer that is much farther than the real distance. Potential aperture problems can be readily detected by examining the bounding box of the blob, and seeing if it impinges on any of the borders of the image.

3.3.3 Filtering

It is important to recognize when a feature may be corrupted, either by the aperture problem, or perhaps from occlusion with another object. Using multiple features to check for compliance with an expected distance or angle can provide a valuable confidence measure. For example, suppose we are trying to recognize the distance to a ball, based on its width. Generally, we should also be able to see the top of the ball (given by the top of the bounding box). Depending on the distance to the ball, the top of the ball should appear at a different height in the image. We can use this information to check that the distance measure generated from the image width feature conforms to the expected appearance of the top of the ball.