Requirements for Robot Vision System

Hardware Description:

The robot has an onboard PC running RedHat linux 7.3. The PC is 500MHz and there is 256MB memory. The robot is connected with wireless 802.11b to a base PC, which is 1.7GHz with 512MB memory, running Debian Sarge 3.1. It is possible to upgrade/replace the base PC if necessary.

There are two firewire cameras (Canon VC-C4) and two framegrabbers. The cameras are mounted on a Directed Perceptions pan-tilt unit. There are also other modalities: a microphone, two speakers, sixteen sonars, and five front and five rear bump sensors.

The lab website is http://csis.pace.edu/robotlab

Software Description:

All the drivers for the above are installed and functioning. The base PC is running RedHat 9 right now. We have another similar PC that can replace it, which is running Debian Sarge 3.1.

The overall structure of the software is as follows. The base PC runs Soar. The robot driver software (called Aria) is wrapped in Java by SWIG. A Java interface program called RunRobot starts Soar and also starts an instance of the robot driver (called ArRobot). In each cycle, the data from the robot is passed to Soar, which fires rules to

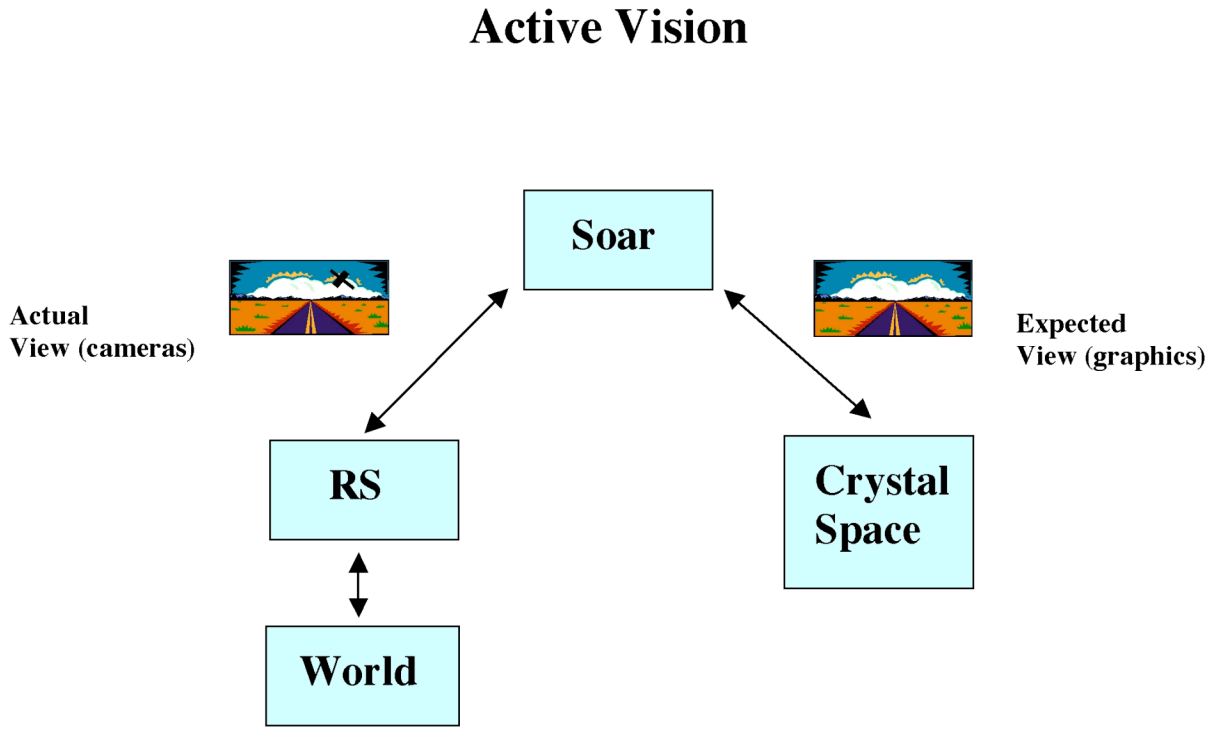
determine what the robot should do; the resulting command is then passed to the robot, which does it. This is all functional.

Sonar data and bumper is passed from the robot in this way and used to make decisions about how to move. This is functional.

The Vision System:

Currently, the vision system is not used at all by Soar. No visual data is being passed into Soar. The overall goal is to make visual data available to Soar so that it can fire rules to detect patterns and objects and then tell the robot what to do.

The purpose of the vision system is that it produces a representation of what the environment actually holds. This is compared with what Soar expects to be there. This is done through the use of an internal world model that is based on a 3D gaming platform called CrystalSpace. Soar creates and manipulates a simulation of the robot and its environment. Differences between what it expects to see and what it actually sees cause Soar to pay attention to these differences and process them, e.g. look at something unexpected and try to recognize it.



CrystalSpace is currently being connected to Soar

.

The planned structure of the vision system is for it to have two main components: a bottom-up component and a top-down component. The bottom-up component will be

always on, and will be continually producing three kinds of output to Soar:
the segments (blobs) in the visual data
stereo disparity information
optical flow

The top-down component will be controlled by Soar, i.e. Soar will fire rules that cause visual processing to take place. The primary kind of visual processing that is controlled this way is object recognition.

So the overall functioning of the visual system would be:
the bottom-up component places segments, stereo info and motion info into Soar,
Soar fires rules that notice a new blob it has never detected before
or notices new depth or motion info,
Soar fires rules that tell the robot to turn its cameras to bring the noticed
info into the center of the visual field,
Soar fires rules that process the new info in the appropriate way, e.g. for a new
blob it grabs a small rectangle containing the new info and sends it to
the object recognition software, which places its result into Soar.

Vision System Requirements:

The software for the segmentation needs to be selected and configured.
One possibility is the Intel Vision Library.
The software for the stereo computations needs to be selected and configured.
One possibility is svs (already configured and running on the robot).
The software for computation of optical flow needs to be selected and
configured.
One possibility is the Intel Vision Library.
The output of the above three systems needs to be connected to Soar.
The object recognition software (ERVision) needs to be configured and then
connected to Soar. This requires some upgrading of packages.

The current idea is that the bottom-up component will run on the onboard computer and the top-down will run on the base PC. This seems like a logical division of labor, and greatly reduces the info transmitted to the base PC, which is good because the wireless connection is of limited bandwidth. This approachwill require a connection to be programmed between the bottom-up and top-down components.