Advanced Mobile Robotics

Activmedia Pioneer Mobile Robot
--- Hardware/Software Systems

John (Jizhong) Xiao
Department of Electrical Engineering
City College of New York
jxiao@ccny.cuny.edu
Outline

• Homework Update
• Hardware/Software Systems of P2 Robot
  – Pioneer Robots
  – Aria/Saphira Architecture
  – Colbert Interface
  – Invoking C++ from Colbert
  – Simulator
• Lab Exercise
Homework 3 update

• How to do simulation?
  – Understand the algorithms
  – Control Structure
  – Matlab function: ODE45

• Topics for discussion
  – Implement both algorithm 1 and 2
  – Change parameters and compare results
Mobile Robot Platform
Pioneer-AT Robot

four-wheel skid-steer machine
Containing basic components for
Sensing and navigating in a real-world environment
Robot Server

• Robot Client/Server model
  – Isolates client from details of the hardware
  – Low-bandwidth information and control

• Services
  – Movement control
  – Position integration
  – Sonar and other sensors
  – Communication – 100 ms packet cycle
The Pioneer 2-AT8 has an on-board micro controller for motor control and low-level access to sonar and bumper switches. The microcontroller is a Hitachi H8S-based microcontroller with ActivMedia Robotics Operating System (AROS)/P2OS software.
Microcontroller

- 20 MHz Siemens 88C166 microprocessor with integrated 32K FLASH-ROM.
- 32K of dynamic RAM
- Two RS232-compatible serial ports
- Several digital and analog-to-digital
- PSU I/O user-accessible ports
- An eight-bit expansion bus (See Appendix A for I/O port details.)
- All of the I/O ports, except those used for the motors, encoders, and sonar, are available to the user for accessory hardware.
- The embedded operating software (P2OS) lets you support and manage each of these I/O ports.
- Connector pin outs and electronics details appear in the Appendices.
Saphira/Aria System Overview

• Saphira is an architecture for mobile robot control.
• Originally, it was developed for the research robot Flakey2 at SRI International.
• Saphira and Flakey appeared in the October 1994 show Scientific American Frontiers.
• Saphira and the Pioneer robots placed first in the AAAI robot competition “Call a Meeting” in August 1996, which also appeared in an April 1997 segment of the same program.
• With Saphira 8.x, the Saphira system has been split into two parts. Lower-level routines have been reorganized and re-implemented as a separate software system, Aria.
ARIA

• The ActivMedia Robotics Interface for Applications (ARIA)
• The ActivMedia Robotics Interface for Applications (ARIA) is C++-based open-source development environment that provides a robust client-side interface to a variety of intelligent robotics systems, including your ActivMedia robot’s controller and accessory systems.
• ARIA is the ideal platform for integration of your own robot-control software, since it neatly handles the lower-level details of client-server interactions, including serial communications, command and server-information packet processing, cycle timing, and multithreading, as well as a variety of accessory controls, such as for the PTZ robotic camera, scanning laser-range finder, and motion gyros, among many others.
• What’s more, it comes with source code so that you may examine the software and modify it for your own sensors and applications.
• Where is ARIA?
ARIA Structure
Packet Communications

• Aria supports a packet-based communications protocol for sending commands to the robot server and receiving information back from the robot.

• Typical clients will send an average of one to four commands a second, although the robot server can handle up to 10 or more per cycle (100+ per second) depending on the serial communication rate and the average command packet size.

• All clients automatically receive 10 or more server-information packets a second back from the robot. These information packets contain sensor readings and motor movement information, among other details.
State Reflector

• It is tedious for robot control programs to deal with the issues of packet communication. So, Saphira incorporates an internal state reflector to mirror the robot’s state on the host computer.

• Essentially, the state reflector is an abstract view of the actual robot’s internal state. There is information about the robot’s movement and sensors, all conveniently packaged into data structures available to any micro-task or asynchronous user routine.

• Similarly, to control the robot, a routine sets the appropriate control variable in the state reflector, and the communication routines will send the appropriate command to the robot.
Saphira

• Saphira, including the Colbert language, is a full-featured robotics control environment developed at SRI International’s Artificial Intelligence Center.

• Saphira is based on ARIA and together they form the robotics-control and applications-development foundation for much of the ActivMedia Robotics.

• The complete, licensed Saphira robotics development environment, including C/C++ libraries, GUI interface and Simulator, comes bundled with the ActivMedia robot

• It includes advanced packages such as gradient navigation and localization software
Saphira / Aria Architecture

Component Description

Robot OS Description
Direct & Behavioral Actions in Saphira

Direct Control

Behavior Control

Sensors → Computation → Motor Controller

Sensors → Computation → Resolver → Motor Controller

Resolver
# Direct Actions in Saphira

## Direct motion control commands

<table>
<thead>
<tr>
<th></th>
<th>Translation</th>
<th>Rotation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Velocity</strong></td>
<td>speed(v)</td>
<td>rotate(v)</td>
</tr>
<tr>
<td><em>mm/sec</em></td>
<td>setVel(int v)</td>
<td>setRotVel(int ( \omega ))</td>
</tr>
<tr>
<td><em>deg/sec</em></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Position</strong></td>
<td>move(d)</td>
<td>turnto(( \phi ))</td>
</tr>
<tr>
<td><em>mm</em></td>
<td>move(int d)</td>
<td>setHeading(int ( \phi ))</td>
</tr>
<tr>
<td><em>deg</em></td>
<td></td>
<td>turn(( \phi ))</td>
</tr>
<tr>
<td></td>
<td></td>
<td>setDeltaHeading(int ( \phi ))</td>
</tr>
<tr>
<td><strong>2 Wheels</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><em>mm/sec</em></td>
<td>setVel2(left, right)</td>
<td></td>
</tr>
<tr>
<td><strong>Stop</strong></td>
<td>stop</td>
<td>stop()</td>
</tr>
</tbody>
</table>
Each behavior is a *subclass* that computes
- Forward velocity and heading setpoint
- Activation level in [0,1]—how strong the action is

Behaviors are invoked with a priority class (integer)
- Behaviors with the same priority class *compete*
- Behaviors in a higher priority class *dominate*
Colbert Robot Programming Language

- C-like language for writing robot control programs.
- Users can quickly write and debug complex control procedures, called *activities*.
- A runtime evaluation environment in which users can interactively view their programs, edit and rerun them, and link in additional C++ code.
- Activities have a finite-state semantics that makes them particularly suited to representing procedural knowledge of sequences of action. Activities can start and stop direct robot actions, low-level behaviors, and other activities.
- Activities are coordinated by the Colbert executive, which supports concurrent processing of activities.
Activities in Colbert

Finite State Semantics

act patrol(int a)
{
    while (a != 0)
    {
        a = a-1;
        turno(180);
        move(1000);
        turno(0);
    }
}
Action Evaluation Cycle

The set of currently active actions is held on a list in the robot object `SfROBOT`. On every cycle (100 ms), each action object is evaluated to produce a translational and rotational output, along with a strength for each.

The strength, which varies from 0 to 1, indicates how strongly the action prefers to have this motion executed. The output values for behavioral actions are described by a structure, `ArActionDesired` (see Aria/include/ArActionDesired.h).

Once the outputs of all current actions have been computed, they are given to a resolver to determine what the final output will be. There are many possible types of resolution strategies: averaging, winner-take-all, competition, etc.

Users are free to define their own resolution strategies to fit particular application needs; these strategies are defined by subclassing the `ArResolver` class. Aria’s standard resolution strategy is a two-part resolution strategy (`ArPriorityResolver`).
<table>
<thead>
<tr>
<th>Control Actions</th>
<th>Example</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Primitive Action</strong></td>
<td><code>move(1000) timeout 30;</code></td>
<td>Start a primitive action</td>
</tr>
<tr>
<td><code>start act</code></td>
<td></td>
<td>Start an activity</td>
</tr>
<tr>
<td><code>&lt;signal&gt; act</code></td>
<td><code>suspend patrol;</code></td>
<td>Signal an activity</td>
</tr>
</tbody>
</table>

| Activity State Tests   |                          |                                            |
| `<state>(act)`         | `sfGetTaskState("patrol")` | Test the state of an activity             |

| Internal State         |                          |                                            |
| *C assignments and functions* | `x = ObjInFront()+10;` | Test or set the state of the database     |

| Sequencing Actions     |                          |                                            |
| `goto`                 | `goto start;`            | Go to a state                              |
| `while, if`            | `if (a == 0) goto start;`| Iterative and conditional execution        |
| `waitfor`              | `waitfor(timedout(act)||a<0);` | Conditional suspension                   |
| `wait <int>`           | `wait 30;`               | Wait \(n\) cycles                         |
Representation of Space

- Mobile robots operate in a geometric space, and the representation of that space is critical to their performance.
- Two main geometrical representations in Saphira:
  - The Local Perceptual Space (LPS) is an egocentric coordinate system a few meters in radius centered on the robot.
  - Global Map Space (GMS) is used to represent objects that are part of the robot’s environment, in absolute (global) coordinates.
- The LPS is useful for keeping track of the robot’s motion over short space-time intervals, fusing sensor readings, and registering obstacles to be avoided. The LPS gives the robot a sense of its local surroundings.
- The main Saphira interface window displays the robot’s LPS.
Local Perceptual Space
Perceptual Structure

Local Perceptual Space (LPS)

Sensors

planning

sequencing

behaviors

Actions

object recognition

surface construction

depth info
Robot Simulator

• The simulator allows developers to debug applications conveniently on a computer without using a physical robot.
• The simulator has realistic error models for the sonar sensors, laser range-finder, and wheel encoders.
• Even its communication interface is the same as for a physical robot, so developers won’t need to reprogram or make any special changes to the client to have it run with either the real robot or the simulator.
• The simulator also lets you construct 2-D models of real or imagined environments, called *worlds*.
• World models are abstractions of the real world, with linear segments representing the vertical surfaces of corridors, hallways, and the objects in them.
Robot Simulator
Code Your Own Behavioral Actions

class SfMovitAction : public ArAction
{
    public:
        SFEXPORT SfMovitAction(int distance, int heading); // constructor
        virtual ~SfMovitAction() {}; // nothing doing
        SFEXPORT virtual ArActionDesired *fire(ArActionDesired currentDesired); // this defines the action
        SFEXPORT void reset() { gone = 0; ax = SfROBOT->getX(); ay = SfROBOT->getY(); }
        SFEXPORT void set(int distance, int heading) // let us set this on the fly
            { reset(); myDistance = distance; myHeading = heading; }
        static SfMovitAction *invoke(int distance, int heading); // interface to Colbert
            …local vars…
};

SFEXPORT ArActionDesired *SfMovitAction::fire(ArActionDesired d)
{
    // reset the actionDesired (must be done)
    myDesired.reset();
    // check the distance to be traveled
    double dx = ax - SfROBOT->getX();
    double dy = ay - SfROBOT->getY();
    ax = SfROBOT->getX(); // set new values
    ay = SfROBOT->getY();
    int ds = (int)sqrt(dx*dx + dy*dy);
    gone += (int)ds;
    sfMessage("Running Movit, gone %d", gone);

    if (gone >= myDistance) {
        sfMessage("Finished Movit");
        deactivate(); // turn off when done
        return NULL;
    } else {
        myDesired.setHeading(myHeading); // control the heading
        myDesired.setVel(200); // moderate speed
        return &myDesired; // return the desired controls
    }
}
Invoking Behavioral Actions from Colbert

class SfMovitAction : public ArAction
{
    public:
        SFEXPORT SfMovitAction(int distance, int heading); // constructor
        virtual ~SfMovitAction() {} ;  // nothing doing
        SFEXPORT virtual ArActionDesired *fire(ArActionDesired currentDesired); // this defines the action
        SFEXPORT void reset() { gone = 0; ax = SfROBOT->getX(); ay = SfROBOT->getY(); }  
        SFEXPORT void set(int distance, int heading) // let us set this on the fly  
            { reset(); myDistance = distance; myHeading = heading; }  
        static SfMovitAction *invoke(int distance, int heading); // interface to Colbert

        ...local vars...
};

SfMovitAction *
SfMovitAction::invoke(int distance, int heading)
{  return new SfMovitAction(distance, heading); }

sfLoadInit () {  
    sfAddEvalAction("Movit", (void *)SfMovitAction::invoke, 2, sfINT, sfINT); }

In Colbert:
start Movit(1000,95);
Call Your Actions in Colbert

• Steps:
  – Write a C++ program containing your code, including calls to Saphira library functions.
  – Compile the program to produce an object file.
  – Link the object file together with the relevant Saphira library to create a shared object file.
  – Write an activity in Colbert and call the action.
  – Make sure the paths are set right
Localization and Navigation

- Saphira incorporates sophisticated algorithms for some difficult robot tasks.
- *Localization* is the task of keeping track of robot position within an environment. Saphira has facilities for both sonar and laser rangefinder based localization. It uses efficient probabilistic techniques developed recently by Dieter Fox and his colleagues.
- *Navigation* is the task of determining a good path for the robot to follow to a goal, and also keeping the robot out of trouble as it moves. Saphira uses the gradient method for this task. Developed by Kurt Konolige, it is an optimal realtime path-planner for the robot.
World Maps

• Saphira uses line-drawing maps of the environment for localization and navigation. These maps can be input by hand from textual coordinate files for simple environments.

• ActivMedia Robotics has two more advanced map input methods. A **graphical mapping interface** allows the user to interactively create maps using a GUI tool.

• For automatic map-building, ActivMedia Robotics has deployed Steffen Gutmann’s *ScanStudio*, a sophisticated algorithm that builds maps automatically from laser range scans.
How to Run the Real Robot?

• Remotely connect from your computer
  – Linux
  – Windows
  – Communication

• From the onboard embedded computer
  – Linux Redhat 7.3.
  – What are the user name and password

• Simulation first, then test on real robots! (Safety First)
Safety Watchdog & Configuration

• What will happen if communication is lost?
  – Pioneer 2’s and PeopleBot’s standard onboard software, P2OS, contains a communications watchdog that will halt motion if communications between a client computer and the server are disrupted for a set time interval, nominally two seconds (watchdog parameter). The robot will automatically resume activity, including motion, as soon as communications are restored.
  – P2OS also contains a stall monitor. If the drive exerts a PWM pulse that equals or exceeds a configurable level and the wheels fail to turn (stallval), motor power is cut off for a configurable amount of time (stallwait). The server software also notifies the client which motor is stalled. When the stallwait time elapses, motor power automatically switches back ON and motion continues under server control.
How to Run the Real Robot?

• Linux
  – g++
  – Basic Commands

• Start from examples

• User’s Group

• Exercise Schedule ??
User Group--Ask for Help

To: saphira-users-requests@activmedia.com
From: <your return e-mail address goes here>
Subject: <choose one command:>
help (returns instructions)
lists (returns list of newsgroups)
subscribe
unsubscribe
Thank you!

Next Class: Reading Discussion, Presentation Preparation

Control architectures: hierarchical/reactive/hybrid paradigms

You must read Part I Robotic Paradigms (Overview, Chapter 1, 2, 3, 4 and 7) of the textbook (Intro. To AI Robotics) before entering the class