

# Teaching Cognitive Robotics With **Tekkotsu**

**David S. Touretzky**

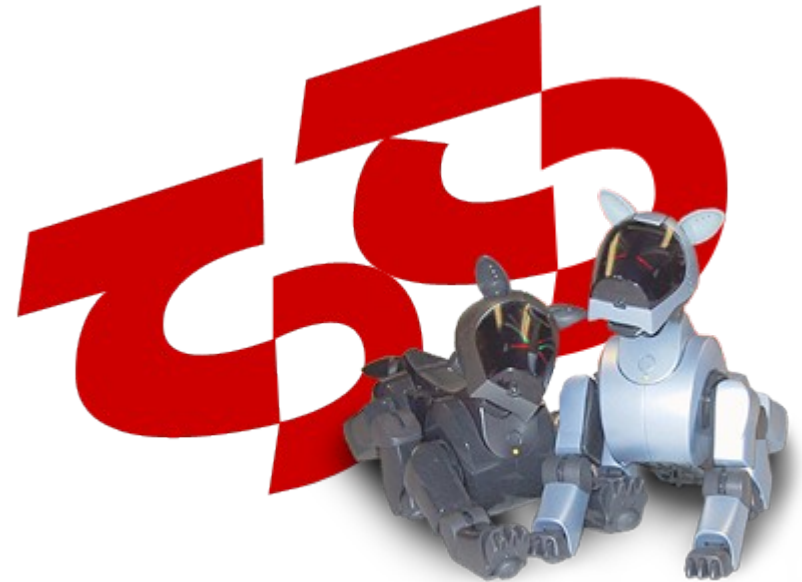
Computer Science Department  
Carnegie Mellon

**Ethan J. Tira-Thompson**

Robotics Institute  
Carnegie Mellon

**Andrew B. Williams**

Department of Computer &  
Information Sciences  
Spelman College



**[www.Tekkotsu.org](http://www.Tekkotsu.org)**

A workshop  
presented at  
SIGCSE 2007  
Covington, Kentucky

March 7, 2007

Funded in part by National Science Foundation awards 0540521 to Carnegie Mellon University and 0540560 to Spelman College. All opinions and conclusions expressed in this document are those of the authors and do not necessarily reflect the views of the National Science Foundation.

# What Is Cognitive Robotics?

A new approach to programming robots:



- Borrowing ideas from cognitive science to make robots smarter
- Creating tools to make robot behavior *intuitive and transparent*

# Why Is Robot Programming Hard?

- It's done at too low a level:
  - Joint angles and motor torques instead of gestures and manipulation strategies
  - Pixels instead of objects
- It's like coding in assembly language, when what you really want is Java or ALICE or Mathematica.



# What If Robots Were Smarter?



- Suppose your robot could already see a bit, and navigate a bit, and manipulate objects.
- What could you do with such a robot?

We're going to find out!

- What primitives would allow you to easily program it to accomplish interesting tasks?

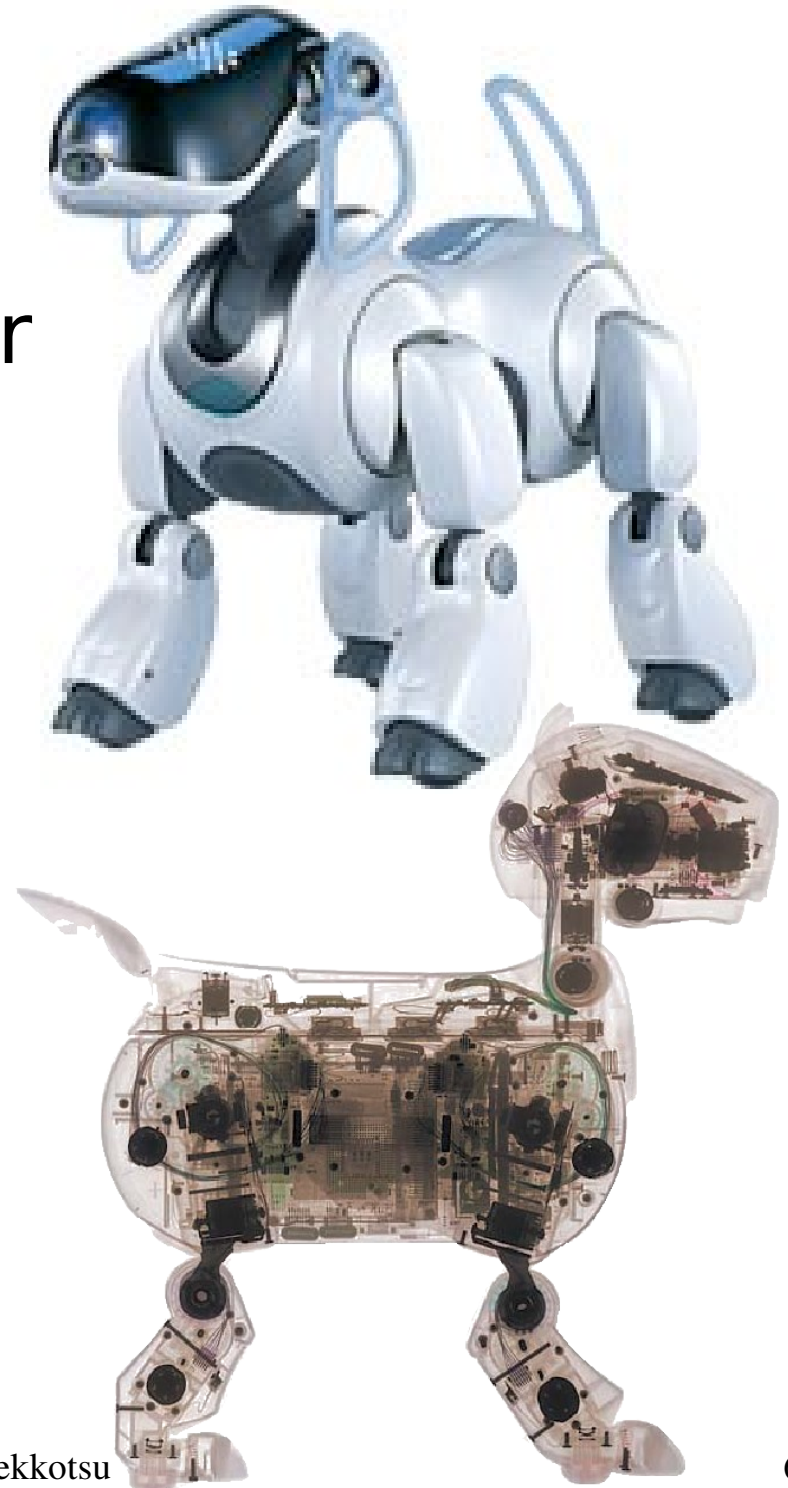
This is what cognitive robotics is about.

# Primitives for Cognitive Robotics

- **Perception**: see shapes, objects
- **Mapping**: where are those objects?
- **Localization**: where am I?
- **Navigation**: go there
- **Manipulation**: put that there
- **Control**: what should I do now?
- **Learning**: how can I do better?
- **Human-robot interaction**: can we talk?

# Sony AIBO ERS-7

- 576 MHz RISC processor
- 64 MB of RAM
- Programmed in C++
- Color camera: 208x160
- 18 degrees of freedom:
  - Four legs (3 degs. Each)
  - Head (3), tail (2), mouth
- Wireless Ethernet



# Potential New Platforms

- Qwerkbot+ developed by Illah Nourbakhsh at CMU.
- Uses TeRK controller board from Charmed Labs.
- Robot recipes on the web:  
<http://www.terk.ri.cmu.edu>



# Potential New Platforms: Lynx Motion



One possible strategy:

Mobile base and arm from Lynx Motion. TeRK controller board (CMU & Charmed Labs) or Gumstix for webcam, wireless, and serial port interfaces.

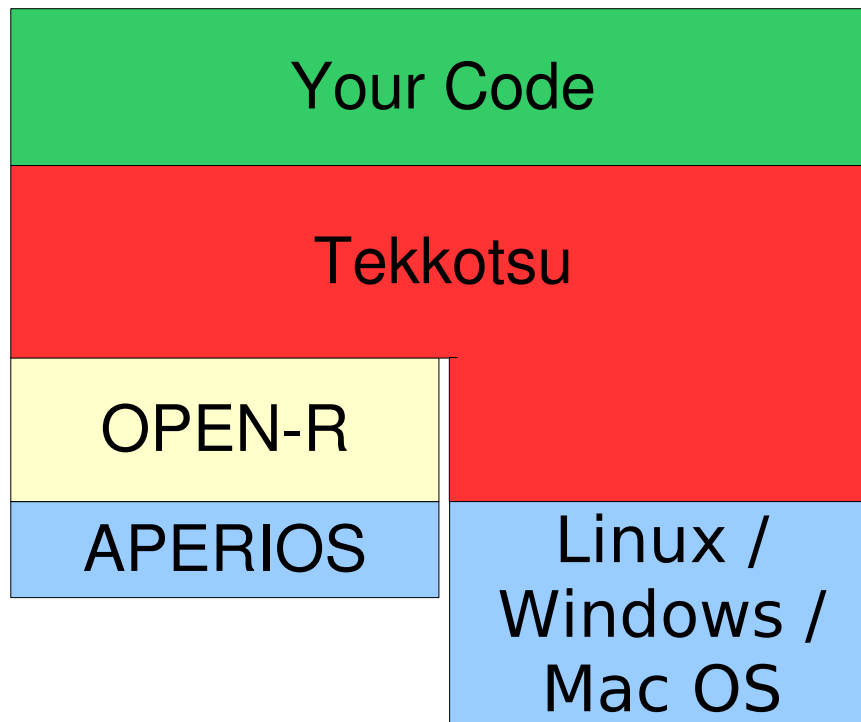


# Tekkotsu Means “Framework” in Japanese

(Literally “iron bones”)



**Tekkotsu.org**



Tekkotsu features:

- Open source, LGPLed
- Event-based architecture
- Powerful GUI interface
- Documented with doxygen
- Extensive use of C++ templates, inheritance, and operator overloading

# Some Early Demos From Our Lab

Implementing learning algorithms  
on the robot:

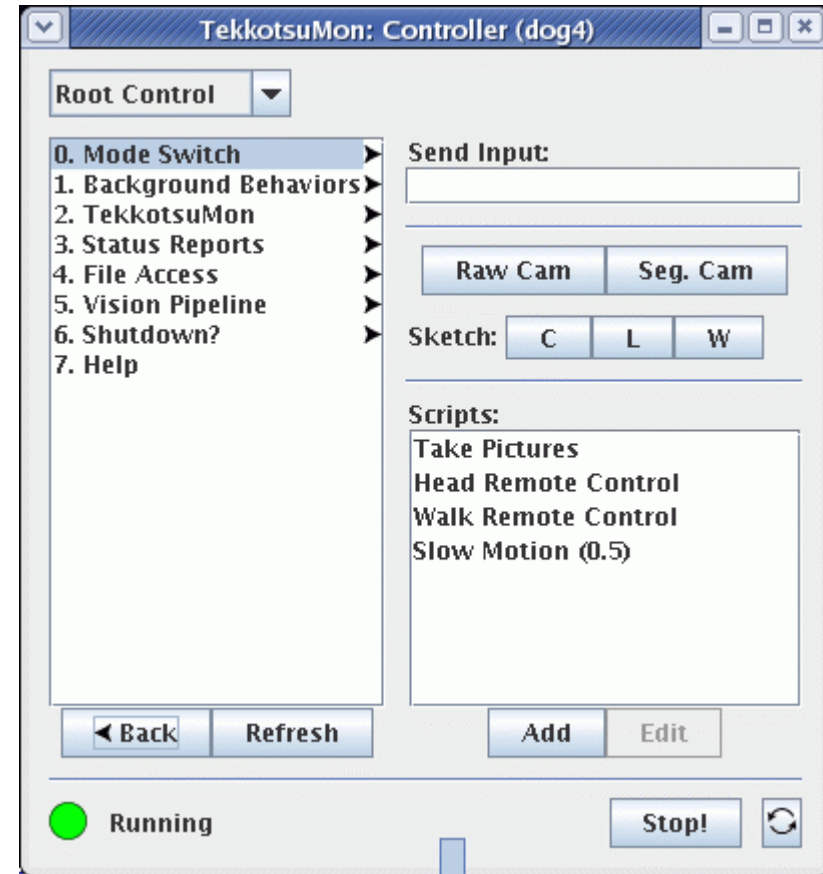
- TD learning for classical conditioning
- Two-armed bandit learning problem



Video  
demos  
from  
Tekkotsu  
web site  
(Videos  
and Screen  
Shots  
section)

# Getting Started With AIBO

- Boot the dog
- Obtain its IP address
  - Turn off Emergency Stop
  - Press and hold the head & chin buttons
- Start ControllerGUI
  - ControllerGUI **172.16.1.xxx**
- Open a telnet connection
  - telnet **172.16.1.xxx** **59000**



# Teleoperation

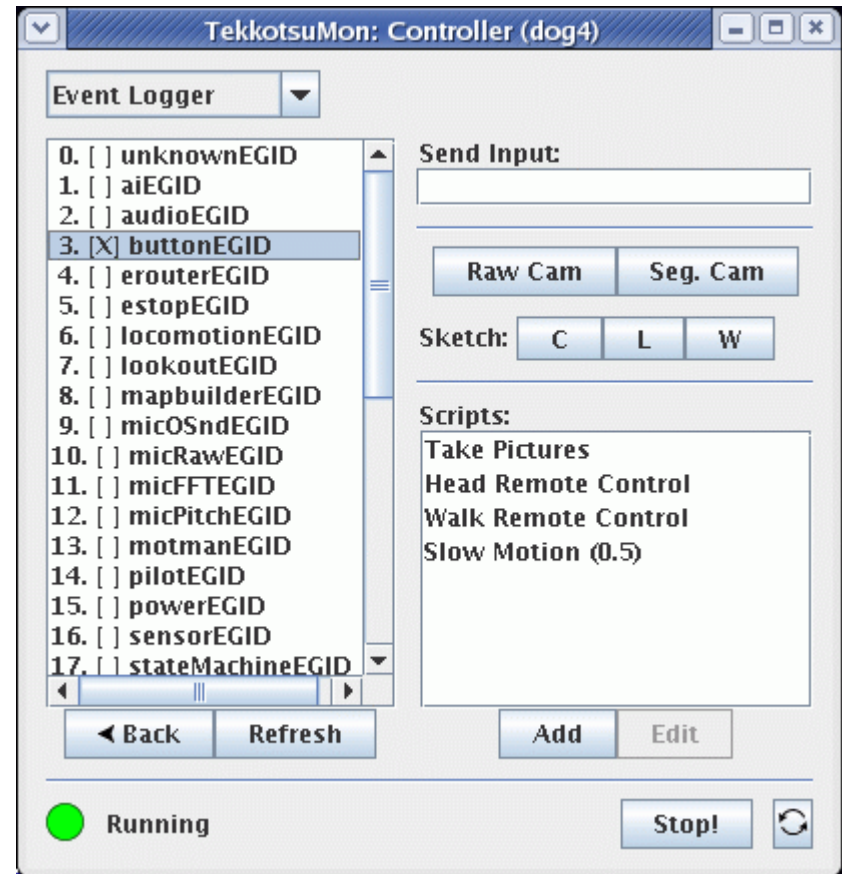
- Click on “Raw Cam” button
- Click on “Head Remote Control”
  - Make sure you're in Run mode (green light), not Stop mode
  - Use the mouse to move the head
- Click on “Walk Remote Control”
  - Put the AIBO on the floor
  - Use the mouse to drive the robot around

# Transparency

- “Transparency” means every aspect of the robot's state should be visible to the user.
  - What is the robot sensing now?
  - What is the robot “thinking” now?
- Achieving transparency requires:
  - A fast connection (preferably wireless)
  - A good set of GUI tools:
    - Event logger, Sensor observer, SketchGUI, Storyboard, etc.

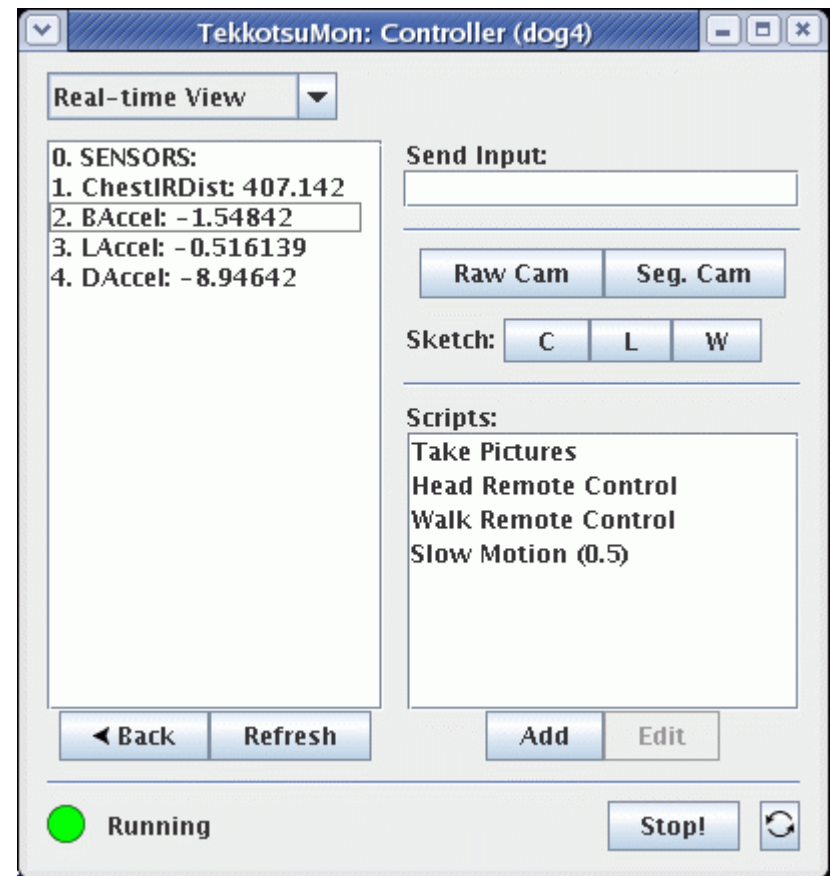
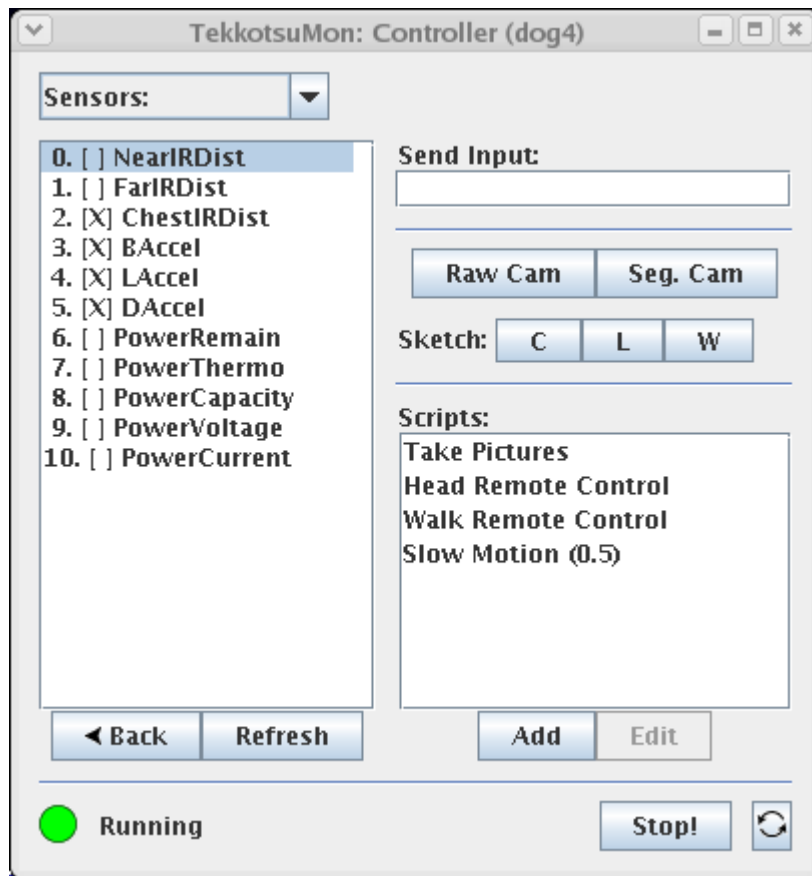
# Event Logger

- Root Control >  
Status Reports >  
Event Logger
- Turn on logging for  
buttonEGID, and select  
Console Output
- Press some buttons, and  
check console output.
- There are over 30 types  
of events in Tekkotsu.



# Sensor Observer

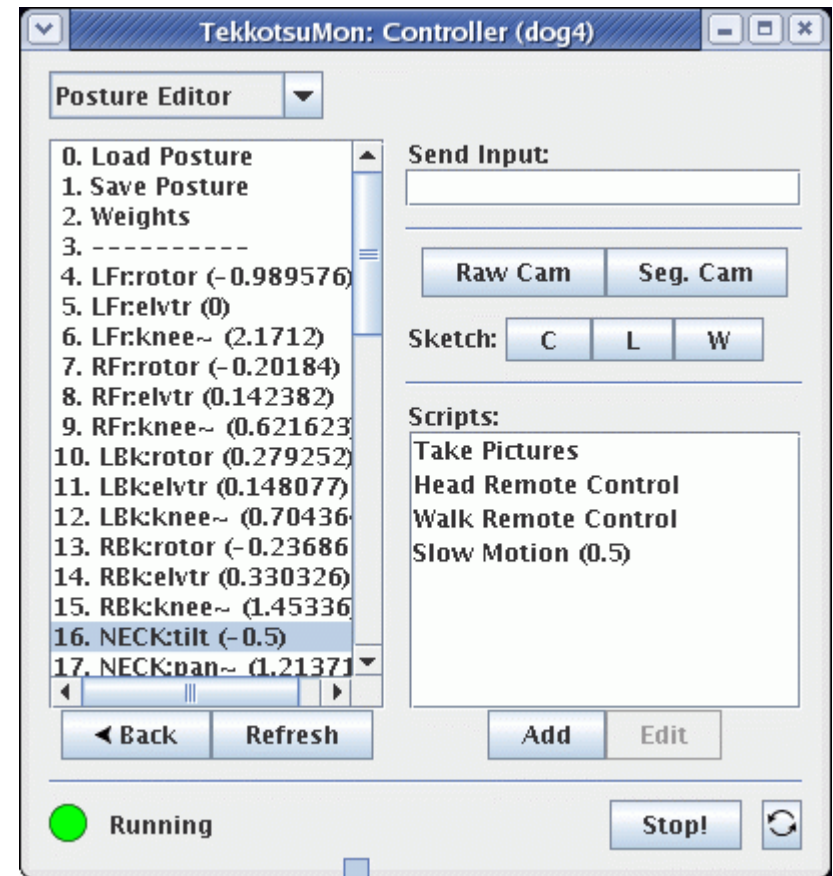
- Root Control > Status Reports > Sensor Observer
- Click on “Sensors”, choose, then go to “Real Time View”



Wave the dog around and watch the accelerometers change!

# Posture Editor

- Root Control > File Access > Posture Editor
- Load Posture
  - STAND.POS
- Select “NECK:tilt”
  - Set value to -0.5 using “Send Input” box
- Hit “Stop!” and move joints manually





# Transparency: Storyboard tool monitors state machines.

The screenshot displays the Tekkotsu Viewer interface, which is used for monitoring and debugging state machines. The main window shows a state machine diagram with various states and transitions. The states include 'Punch', 'Follow', 'Sit', 'Up', 'Down', 'Sniff', 'Time', 'Funny', and 'Look'. The transitions are labeled with actions like 'Punch', 'Follow', 'Sound', and 'Up'. The diagram is titled 'Explore State Machine' and is running on 'localhost' at port '10080'. The 'Properties' panel on the right shows the current selection at 46.875s, with a list of states and transitions: 'Up' (state), 'Up--:PunchLock' (transition), 'Punch' (state), and 'Look' (state). The 'Storyboard' panel at the bottom shows a sequence of states and transitions over time, with a timeline from 0 to 60 seconds. The 'Image Preview' panel at the bottom right shows a 3D rendering of a robot on a green field with a pink ball and a yellow disc.

# An Example Behavior

- Task: *“Respond to a paw button press by turning the head in that direction.”*
- **DoStart()** subscribes to button press events; we're only interested in the two front paws.
- **processEvent()** receives the event and issues a motion command, called a **HeadPointerMC**, to move the head.
- The **HeadPointerMC** runs in a real-time process called **Motion**.

# Demo1 Code (1/2)

```
class Demo1 : public BehaviorBase {
public:
    Demo1() : BehaviorBase("Demo1") {}
    virtual void DoStart() {
        BehaviorBase::DoStart();
        erouter->addListener(this,
                             EventBase::buttonEGID,
                             RobotInfo::LFrPawOffset,
                             EventBase::activateETID);
        erouter->addListener(this,
                             EventBase::buttonEGID,
                             RobotInfo::RFrPawOffset,
                             EventBase::activateETID);
    }
}
```

# Demo1 Code (2/2)

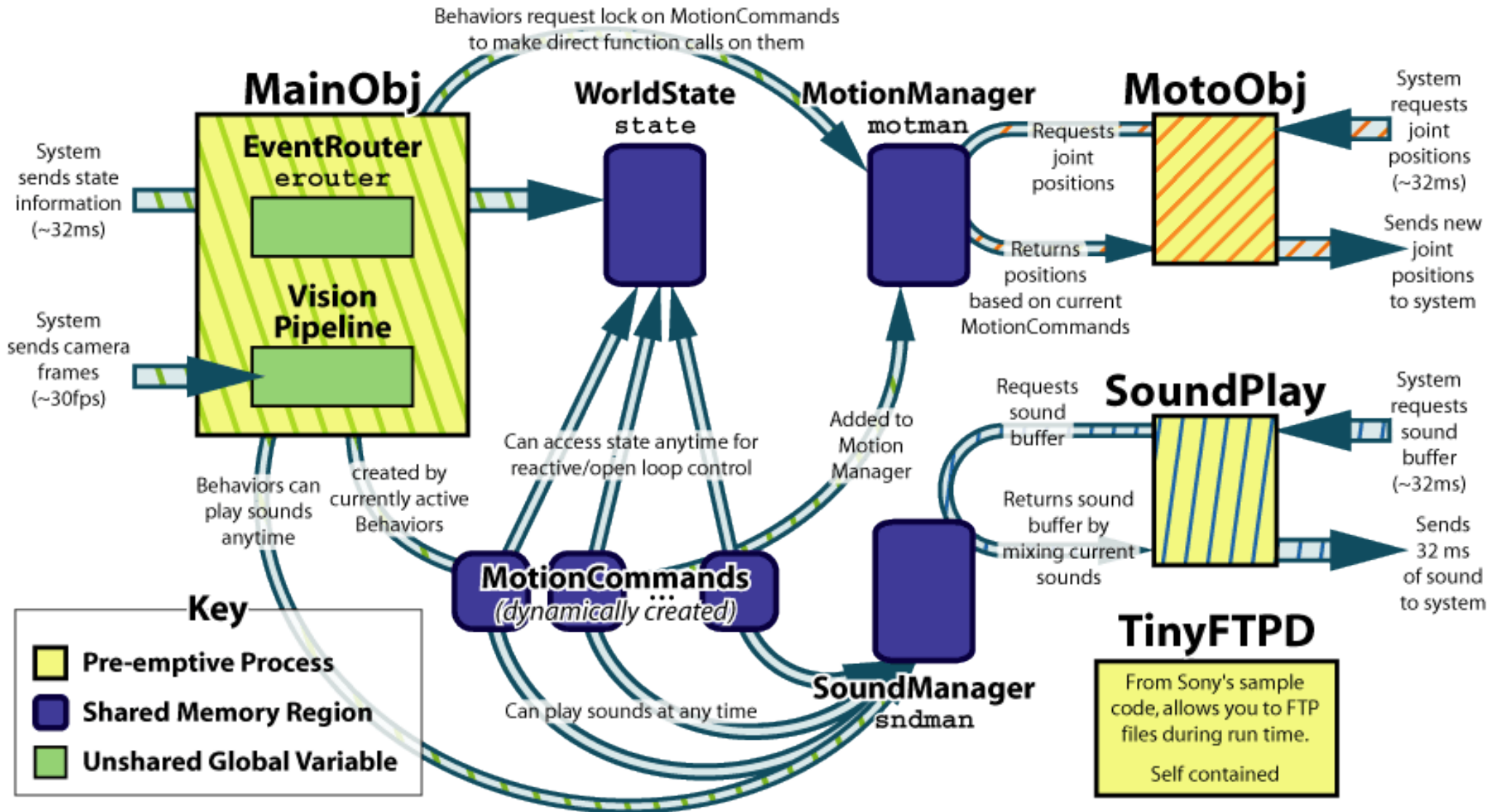
```
virtual void processEvent(const EventBase& event) {
    float pan_value;    // radians

    if ( event.getSourceID() == RobotInfo::LFrPawOffset ) {
        cout << "Go left!" << endl;
        pan_value = +1;
    }
    else {
        cout << "Go right!" << endl;
        pan_value = -1;
    }

    SharedObject<HeadPointerMC> head_mc;
    head_mc->setJoints(0, pan_value, 0);
    motman->addPrunableMotion(head_mc);
}

};
```

# Main vs. Motion



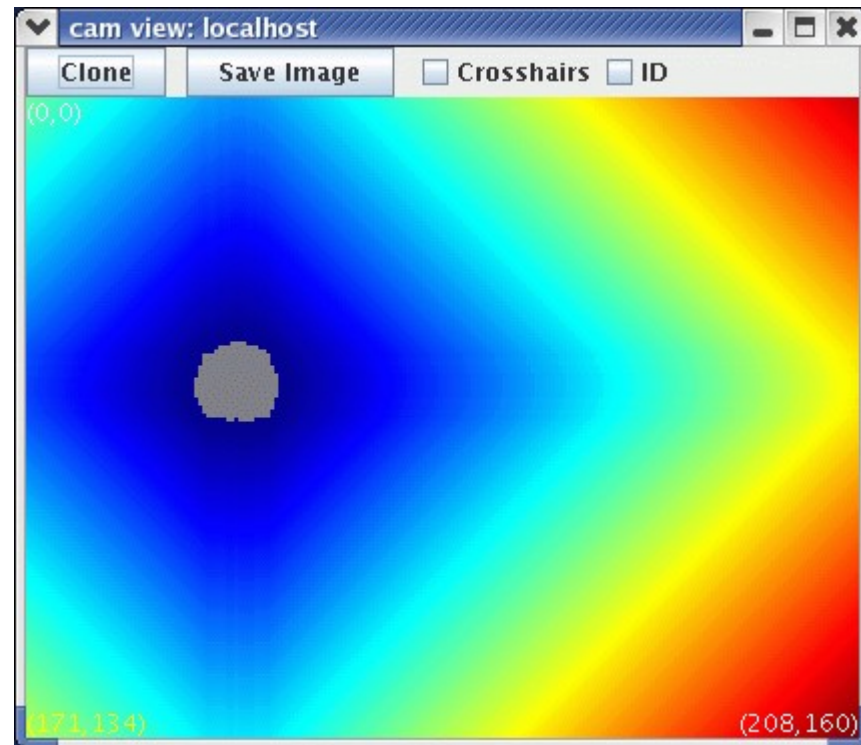
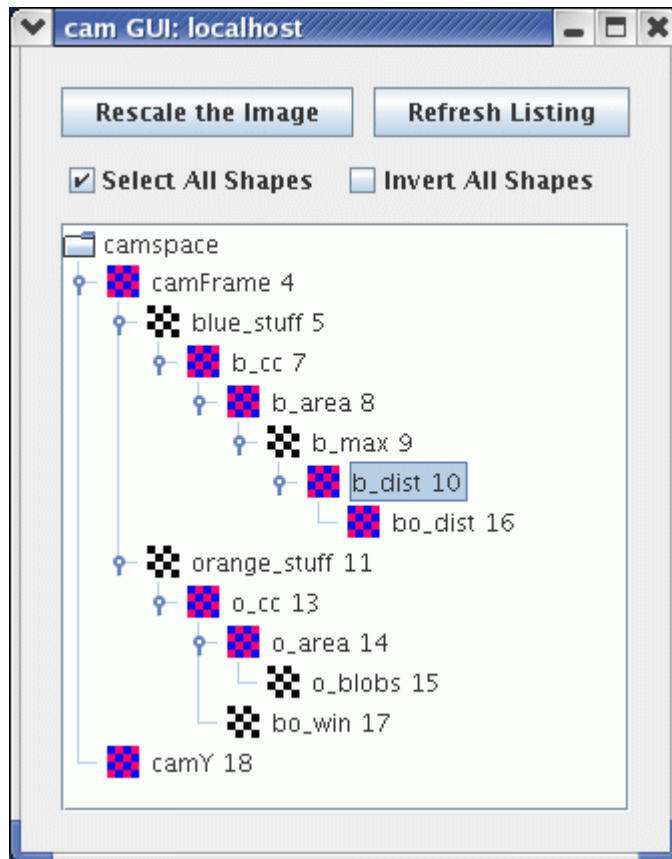
# Ullman (1984): Visual Routines

- Fixed set of composable operators.
- Wired into our brains.
- Operate on “base representations”, produce “incremental representations”.
- Can also operate on incremental representations.
- Examples:
  - marking
  - bounded activation (coloring)
  - boundary tracing
  - indexing (odd-man-out)
  - shift of processing focus

# Sketches in Tekkotsu

- A sketch is a 2-D iconic (pixel) representation.
- Templated class:
  - `Sketch<uchar>`                    *unsigned char*: can hold a color index
  - `Sketch<bool>`                      true if a property holds at image loc.
  - `Sketch<usint>`                    *unsigned short int*: distance, area, etc.
- Visual routines operate on sketches.
- Sketches live in a SketchSpace: fixed width and height, so all sketches are in register.
- A built-in sketch space: `camSkS`.

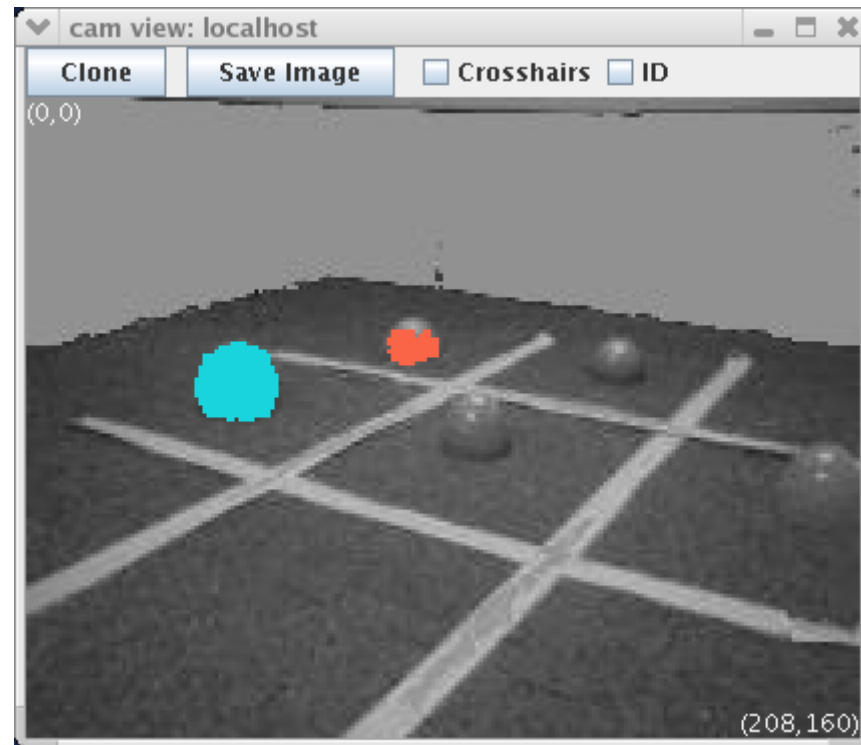
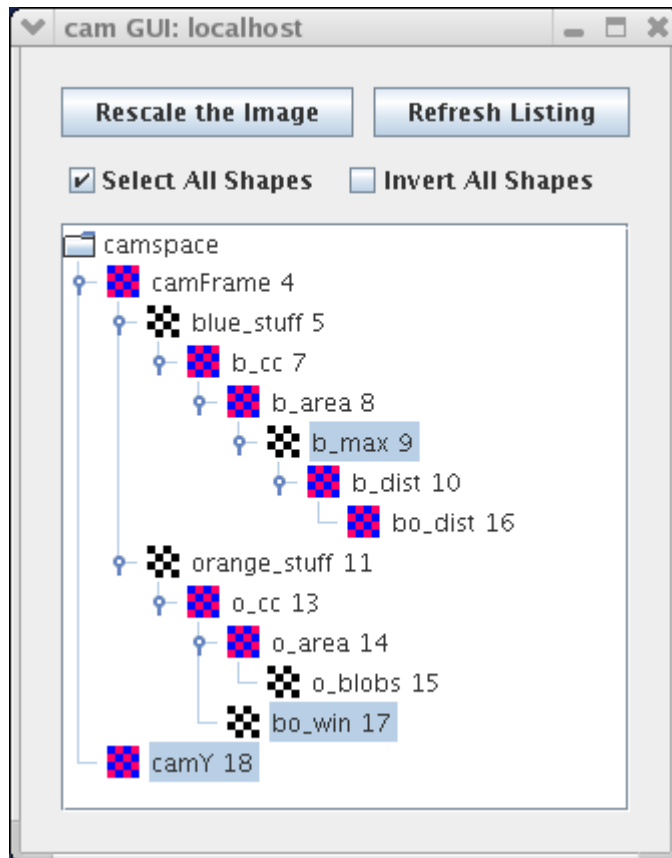
# Distance From Largest Blue Blob





# Orange Blob Closest to Largest Blue Blob

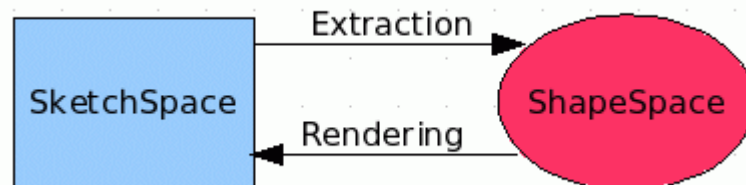
```
NEW_SKETCH(bo_win, bool, o_cc == min_label);
```



# Dual-Coding Representation


- Paivio's “dual-coding theory”:
  - People use both iconic (picture) and lexical (symbolic) mental representations.
  - They can convert between them when necessary, but at a cost of increased processing time.

- Tekkotsu implements this idea:



- What would Ullman say? Visual routines mostly operate on sketches, but not exclusively.

# Mixing Sketches and Shapes

- The strength of the dual-coding approach comes from mixing sketch and shape operations.
- Problem: which side of an orange line has more yellow blobs?
- If all we have is a line segment, people can still interpret it as a “barrier”.
- How do we make the robot do this?

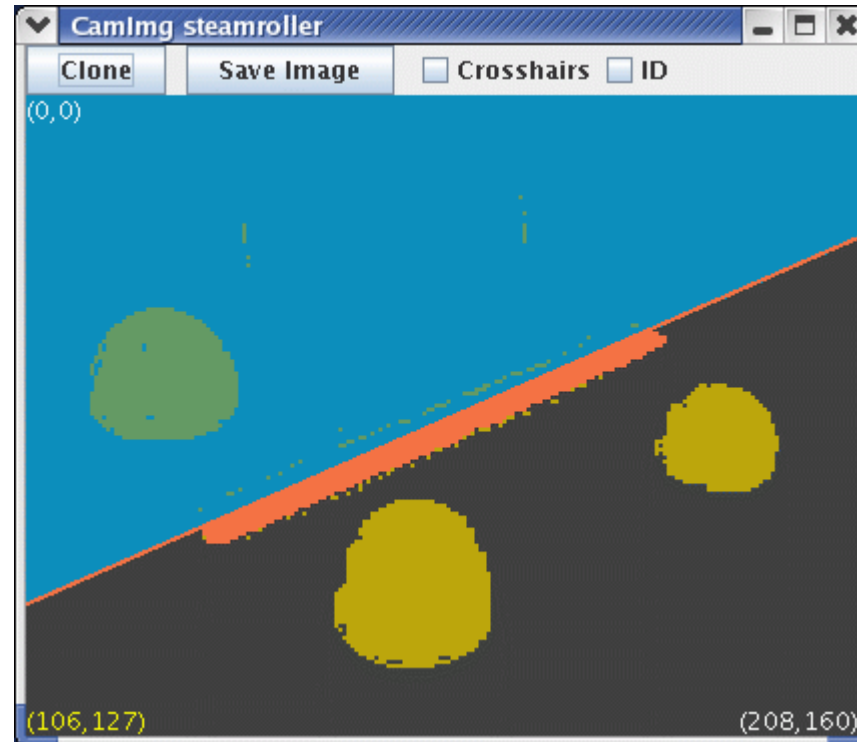
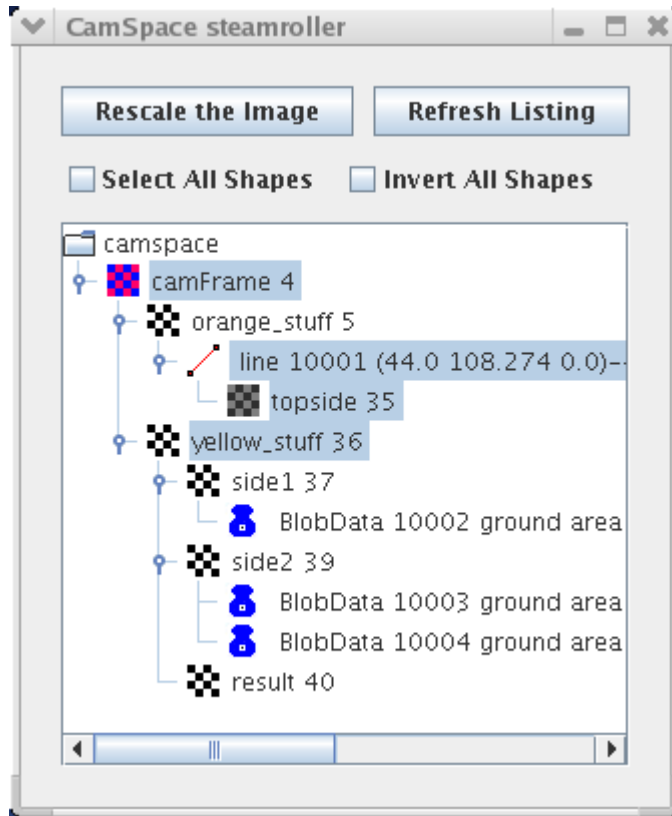
# Lines as Barriers

```
void DoStart() {  
    VisualRoutinesBehavior::DoStart();  
    NEW_SKETCH(camFrame, uchar, sketchFromSeg());  
    NEW_SKETCH(orange_stuff, bool,  
               visops::colormask(camFrame, "orange"));  
    NEW_SKETCH(yellow_stuff, bool,  
               visops::colormask(camFrame, "yellow"));  
  
    NEW_SHAPE(boundary_line, LineData,  
              LineData::extractLine(orange_stuff));  
  
    NEW_SKETCH(topside, bool,  
               visops::topHalfPlane(boundary_line));  
  
    NEW_SKETCH(side1, bool, yellow_stuff & topside);  
    NEW_SKETCH(side2, bool, yellow_stuff & !topside);  
}
```

# Lines as Barriers (cont.)

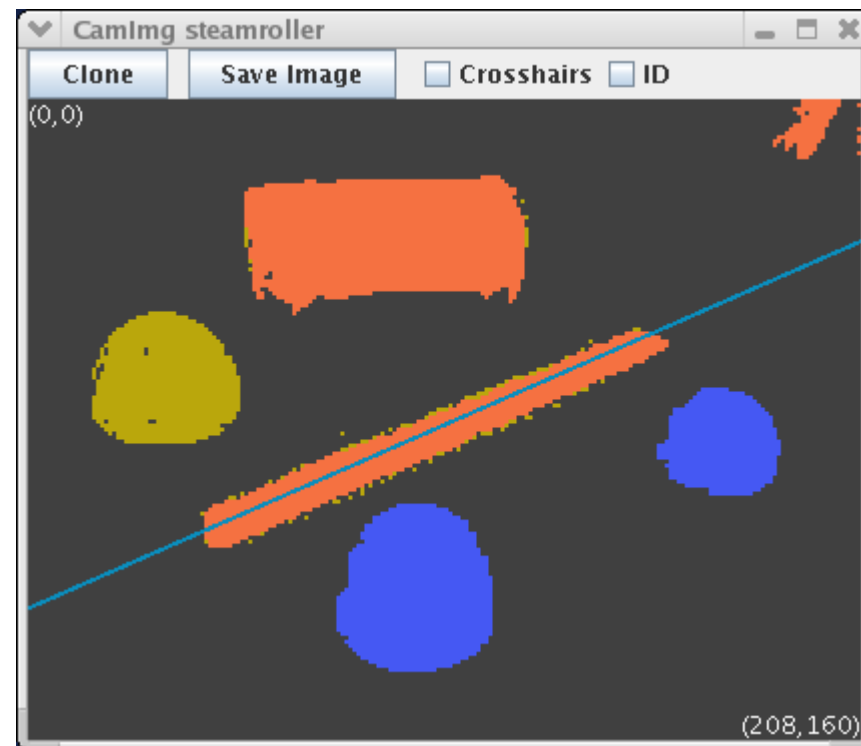
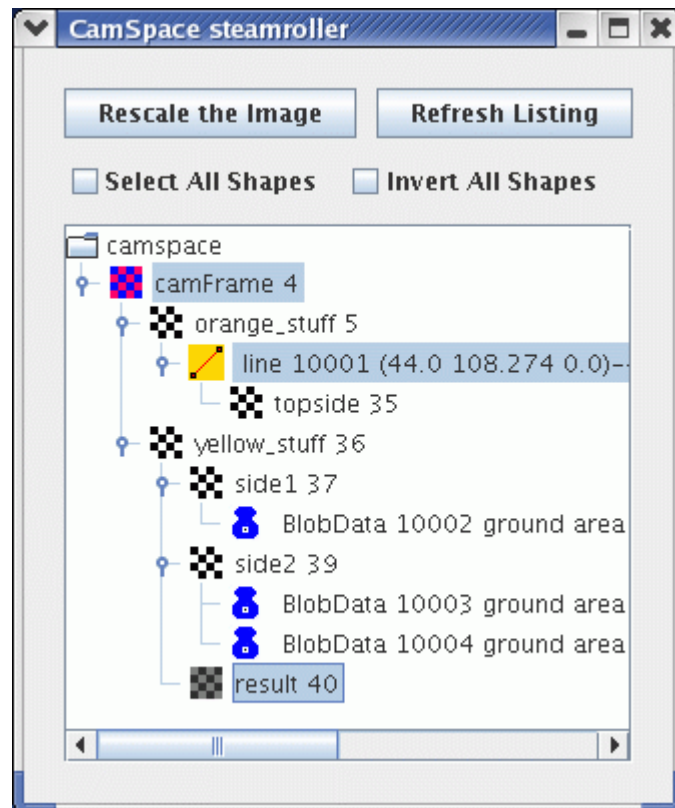
```
NEW_SHAPEVEC(side1blobs, BlobData,  
             BlobData::extractBlobs(side1,50));  
NEW_SHAPEVEC(side2blobs, BlobData,  
             BlobData::extractBlobs(side2,50));  
  
vector<Shape<BlobData> > &winners =  
    side1blobs.size() > side2blobs.size() ?  
    side1blobs : side2blobs;  
  
NEW_SKETCH(result, bool, visops::zeros(yellow_stuff));  
  
SHAPEVEC_ITERATE(winners, BlobData, b)  
    result |= b->getRendering();  
END_ITERATE;  
  
boundary_line->setInfinite();    // for display purposes  
  
}
```

# Lines As Barriers



Subtle point: bool overrides uchar in the SketchGUI, so selecting yellow\_stuff allows the top yellow blob to display even though the inverted (orange) topside is covering its appearance in camFrame. (Competing bools are averaged.)

# Lines As Barriers



# Barrier Demo

- Try running the Barrier demo.
- We'll use different colors:
  - pink for the line
  - orange for the blobs
- In the SketchGUI, hold down the Control key when clicking on a sketch to superimpose multiple sketches.

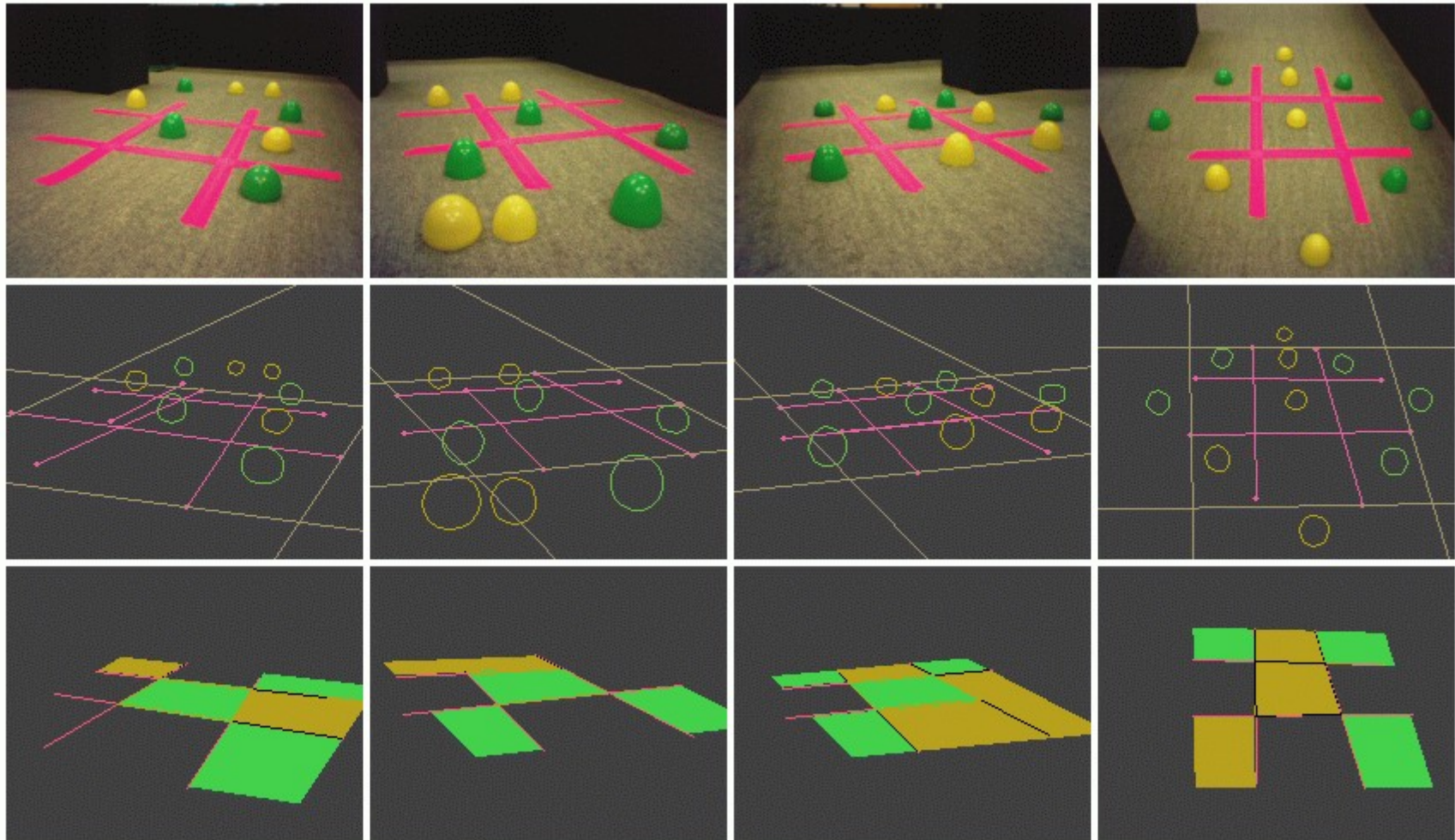


# The MapBuilder

- We can automate the task of object extraction using the MapBuilder.
- Let's look at a tic-tac-toe board:



# Parsing Tic-Tac-Toe Boards



# Programming the MapBuilder

```
const color_index pink_index = ProjectInterface::getColorIndex("pink");  
const color_index blue_index = ProjectInterface::getColorIndex("blue");  
const color_index orange_index =  
    ProjectInterface::getColorIndex("orange");
```

```
MapBuilderRequest req(MapBuilderRequest::localMap);
```

```
req.numSamples = 5; // take mode of 5 images to filter out noise  
req.maxDist = 1200; // maximum shape distance 1200 mm
```

```
req.objectColors[lineDataType].insert(pink_index);
```

```
req.occluderColors[lineDataType].insert(blue_index);  
req.occluderColors[lineDataType].insert(orange_index);
```

```
req.objectColors[ellipseDataType].insert(blue_index);  
req.objectColors[ellipseDataType].insert(orange_index);
```

```
unsigned int mapreq_id = MapBuilder::executeRequest(req);
```

```
erouter->addListener(this, EventBase::mapbuilderEGID,  
    mapreq_id, EventBase::statusETID);
```

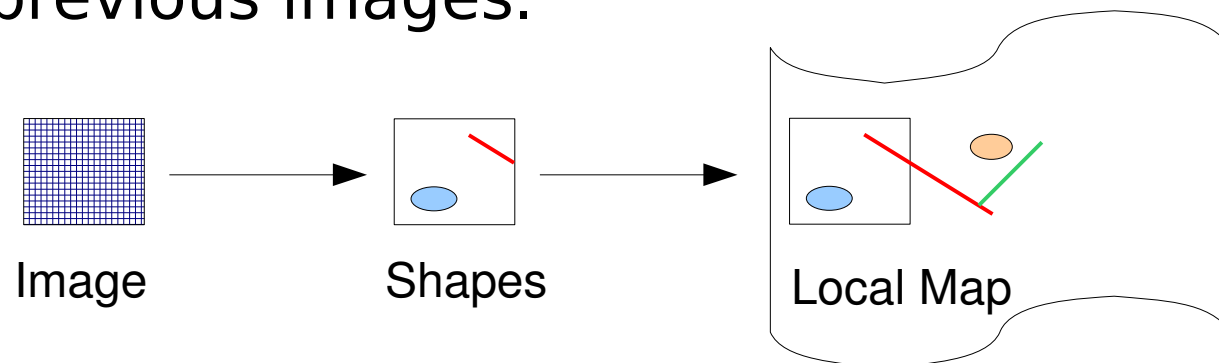


# SimpleTicTacToe Demo

- The MapBuilder extracts lines and ellipses, suppresses noise, handles occlusion.
- User code constructs the parse using a combination of sketches and shapes.
- Look at the parsedBoard sketch for the result.

# Seeing A Bigger Picture

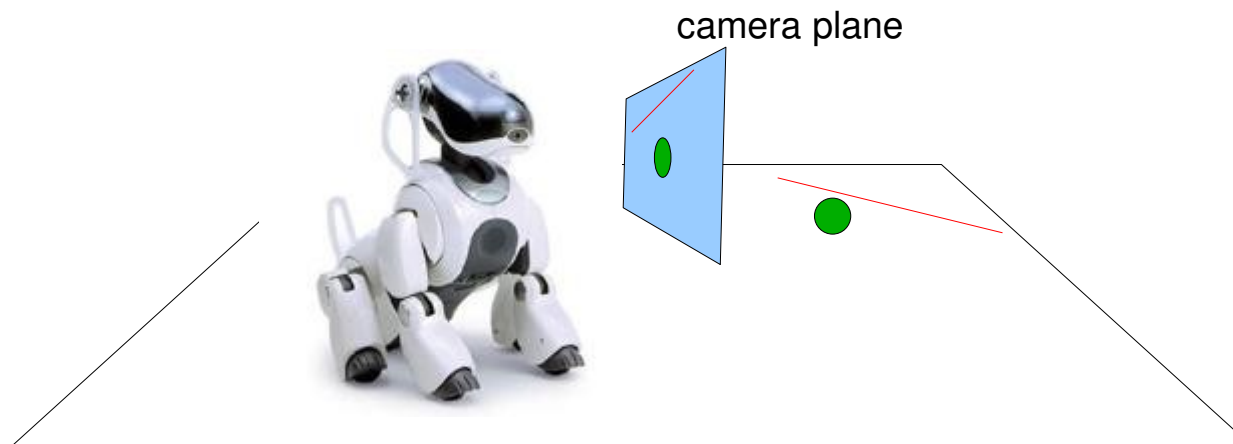
- How can we assemble an accurate view of the robot's surroundings from a series of narrow camera frames?
- First, convert each image to symbolic form: shapes.
- Then, match the shapes in one image against the shapes in previous images.



- Construct a “local map” by matching up a series of camera images.

# Can't Match in Camera Space

- We can't match up shapes from one image to the next if the shapes are in camera coordinates. Every time the head moves, the coordinates of the shapes in the camera image change.
- Solution: switch to a body-centered reference frame.
- If we keep the body stationary and only move the head, the coordinates of objects won't change (much) in the body reference frame.



# Planar World Assumption

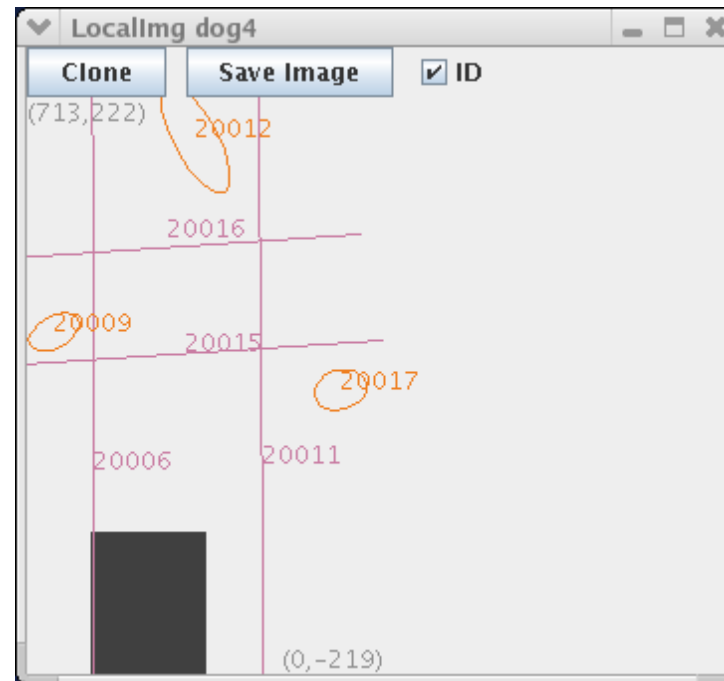
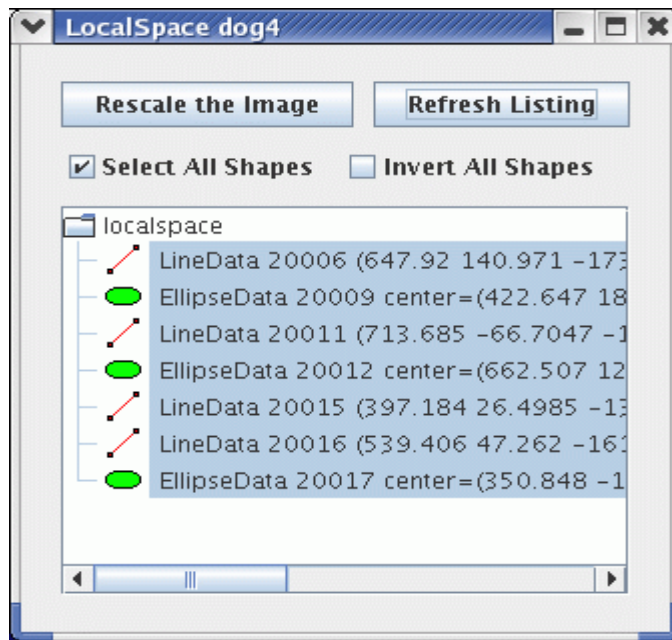
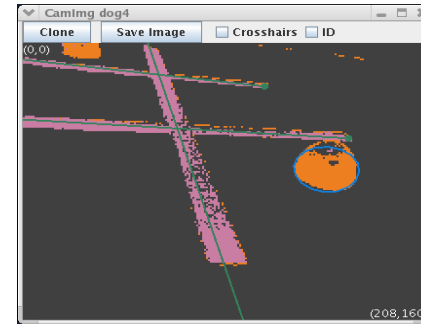
- How do we convert from camera-centered coordinates to body-centered coordinates?
- Need to know the camera pose: can get that from the kinematics system.
- Is that enough? Unfortunately, no.
- Add a planar world assumption: objects lie in the plane. The robot is standing on that plane.
- Now we can get object coordinates in the body frame.

# Shape Spaces

- **camShS** = camera space
- **groundShS** = camera shapes projected to ground plane
- **localShS** = body-centered (egocentric space);  
constructed by matching and importing shapes  
from groundShS
- **worldShS** = world space (allocentric space);  
constructed by matching and importing shapes  
from localShS
- The robot is explicitly represented in worldShS



# Local Map, After Frame 5

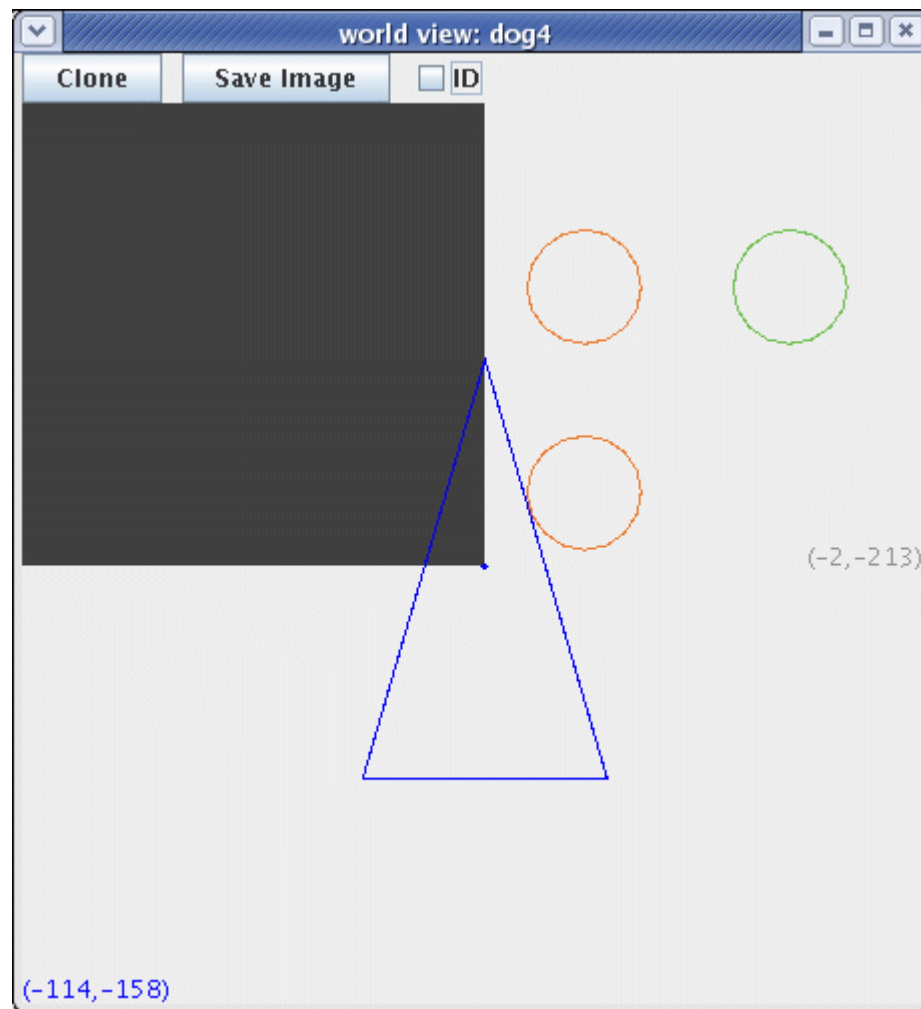


# The Crew

- MapBuilder constructs camera, local, and world maps.
- Lookout moves the head, collects images; can also do fast scanning and tracking.
- Pilot moves the body.
- Particle filter does localization.

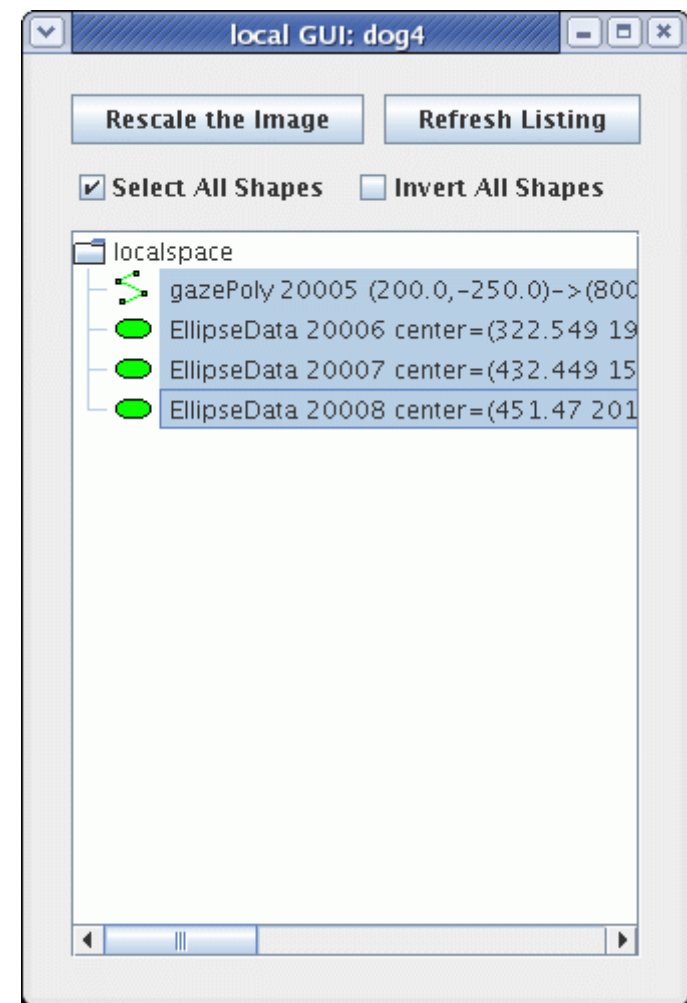
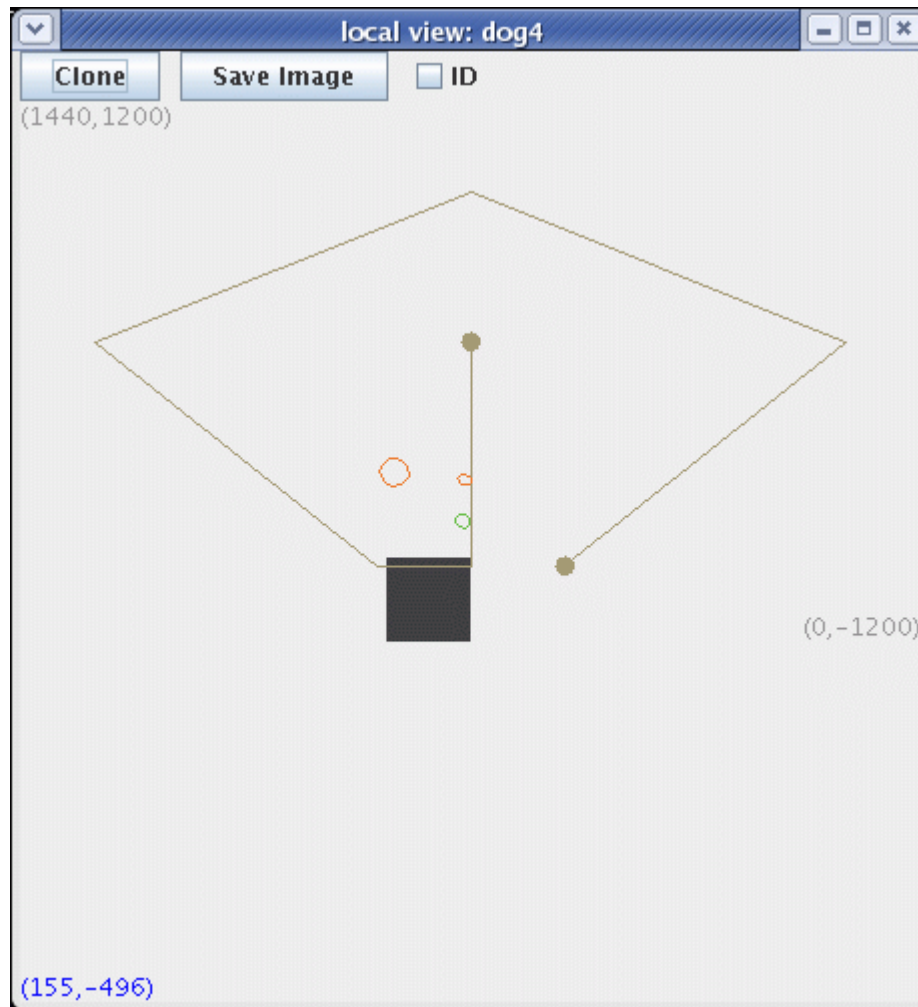
# Localization

The world has three landmarks (worldShS):

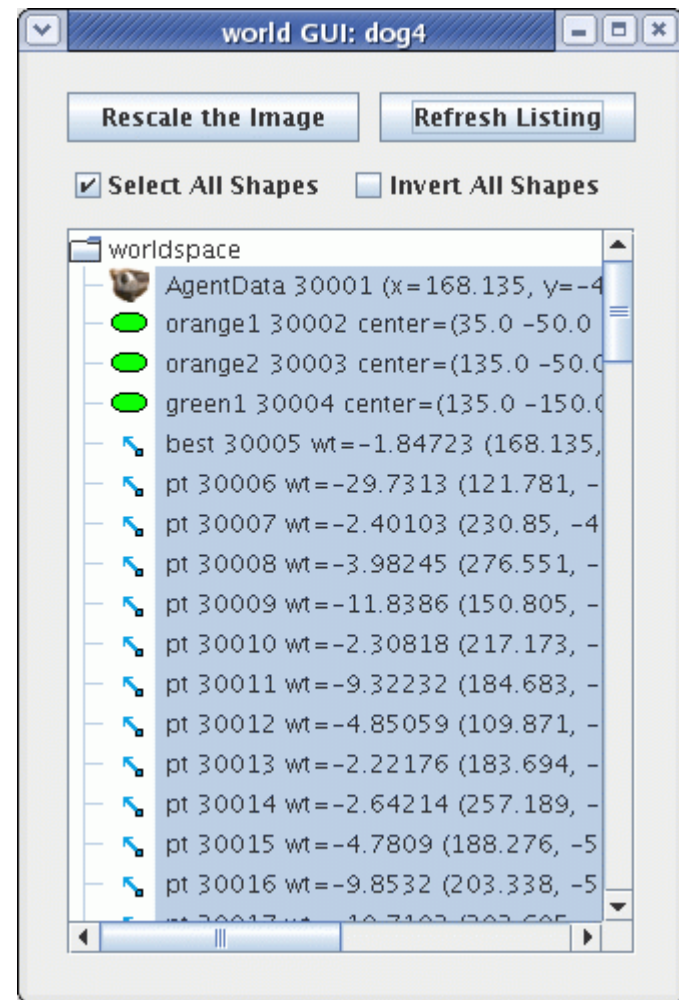
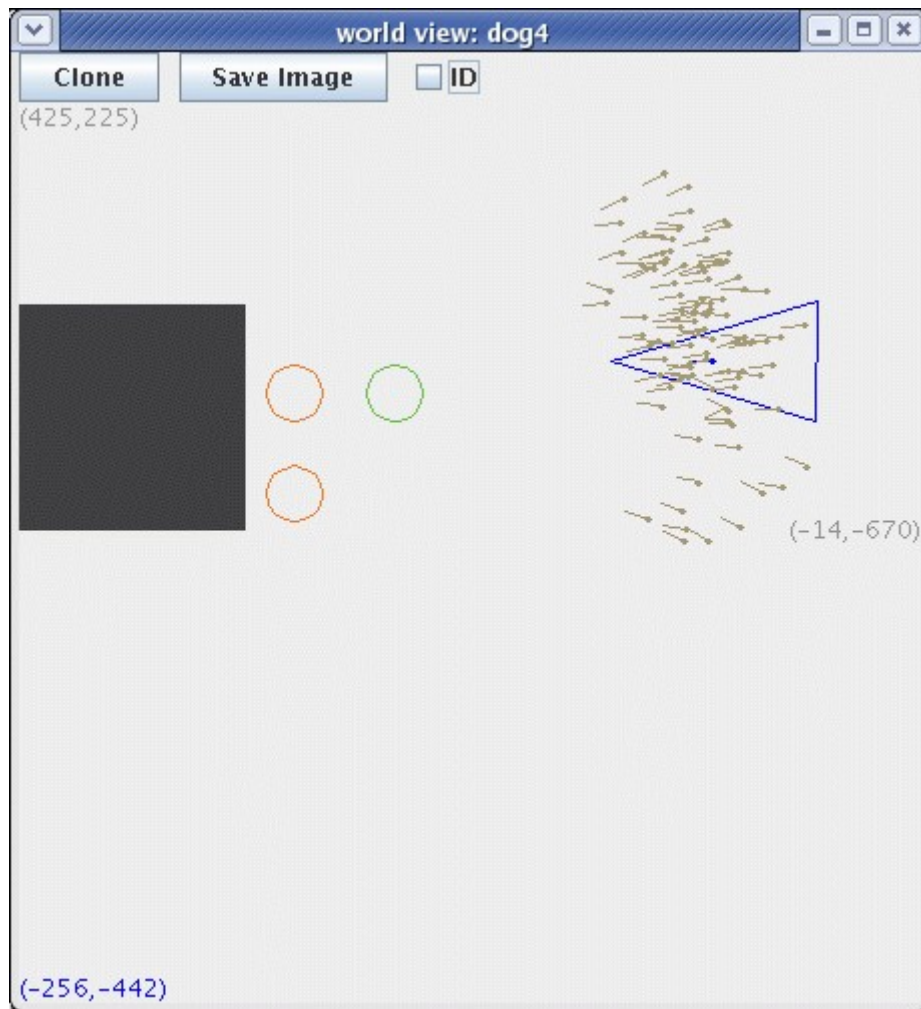


# Define a Scan Pattern and Use MapBuilder to Look Around

Results are constructed in localShS:



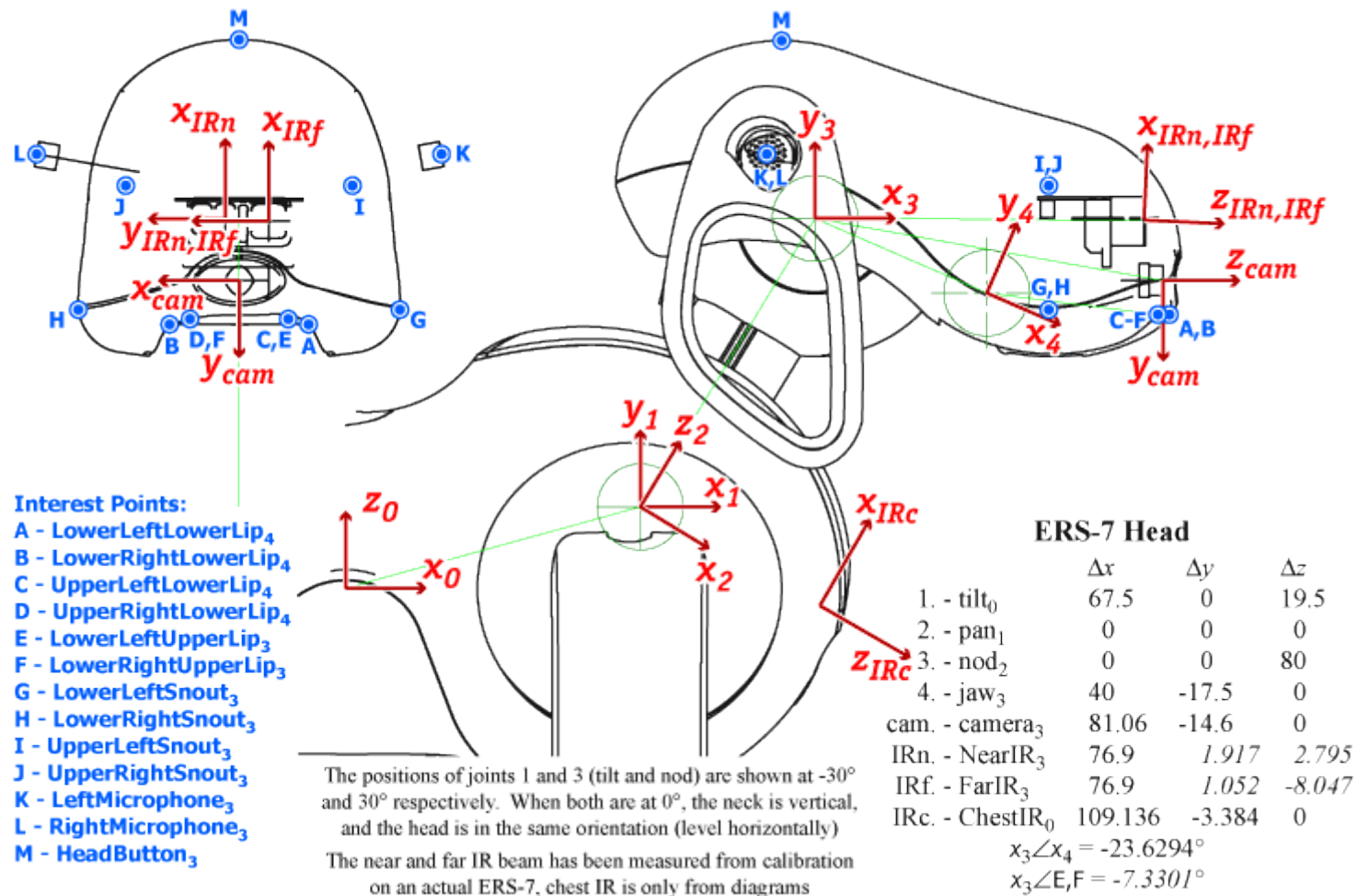
# Use Particle Filter to Localize on the World Map



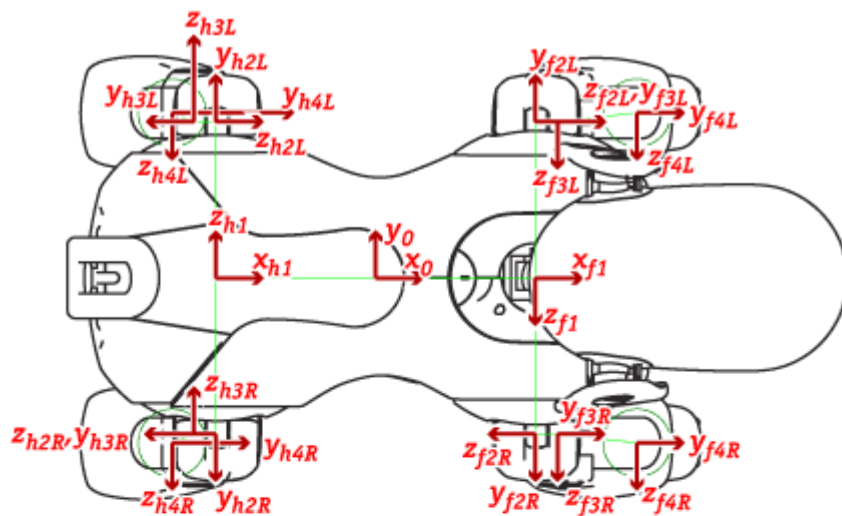
# ParticleDemo

- Set up three landmarks as in the figure on the preceding page.
- Run ParticleDemo on the dog.
- Examine local and world spaces in the SketchGUI.

# Kinematics: Reference Frames and Interest Points



# Kinematics: Body and Legs

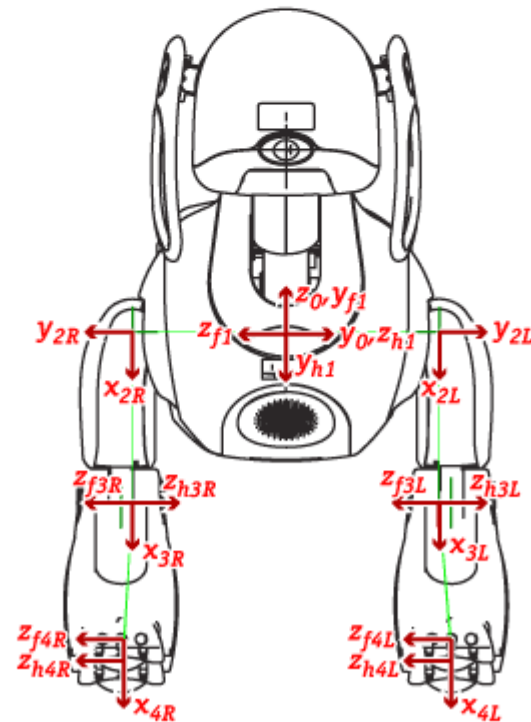
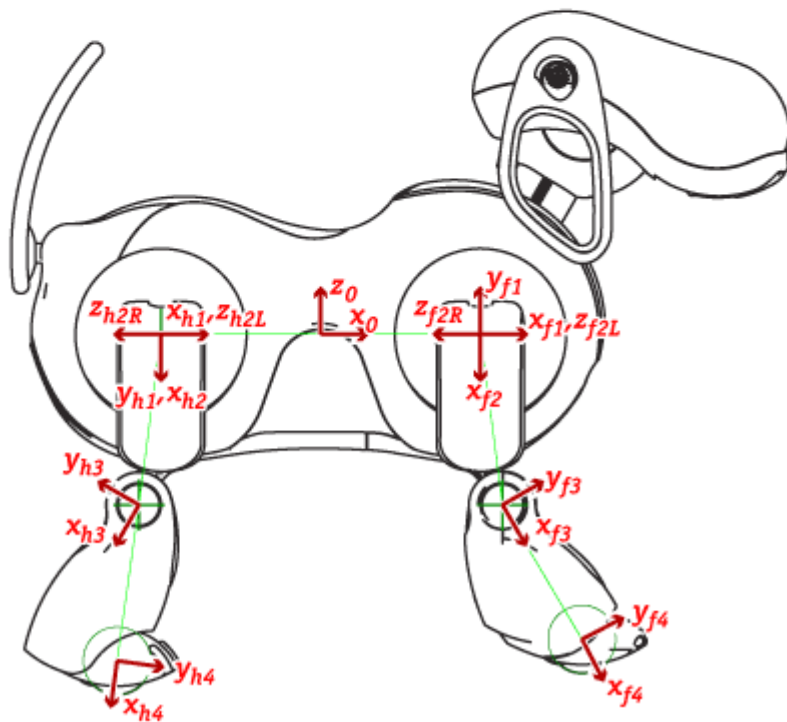


ERS-7 Legs

	$\Delta x$	$\Delta y$	$\Delta z$
1. - shoulder	65	0	0
2. - elevator	0	0	62.5
3. - knee	69.5	0	9
f4. - ball	69.987	-4.993	4.7
h4. - ball	67.681	-18.503	4.7

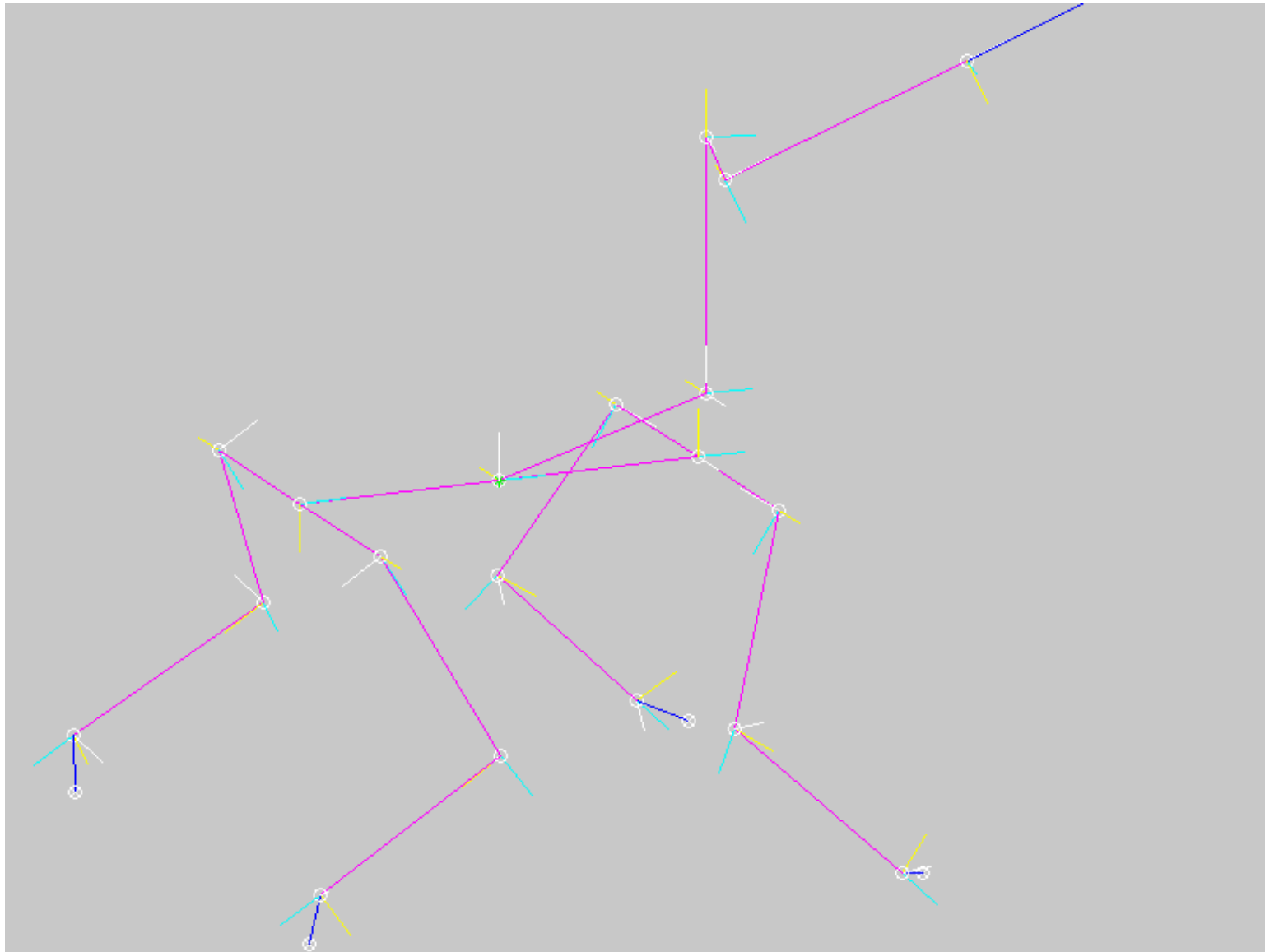
Diameter of ball of foot is 23.433mm  
Each link offset is relative to previous link

The shins shown in this diagram appear to be slightly distorted compared to a real robot. Corresponding measurements have been taken from actual models.



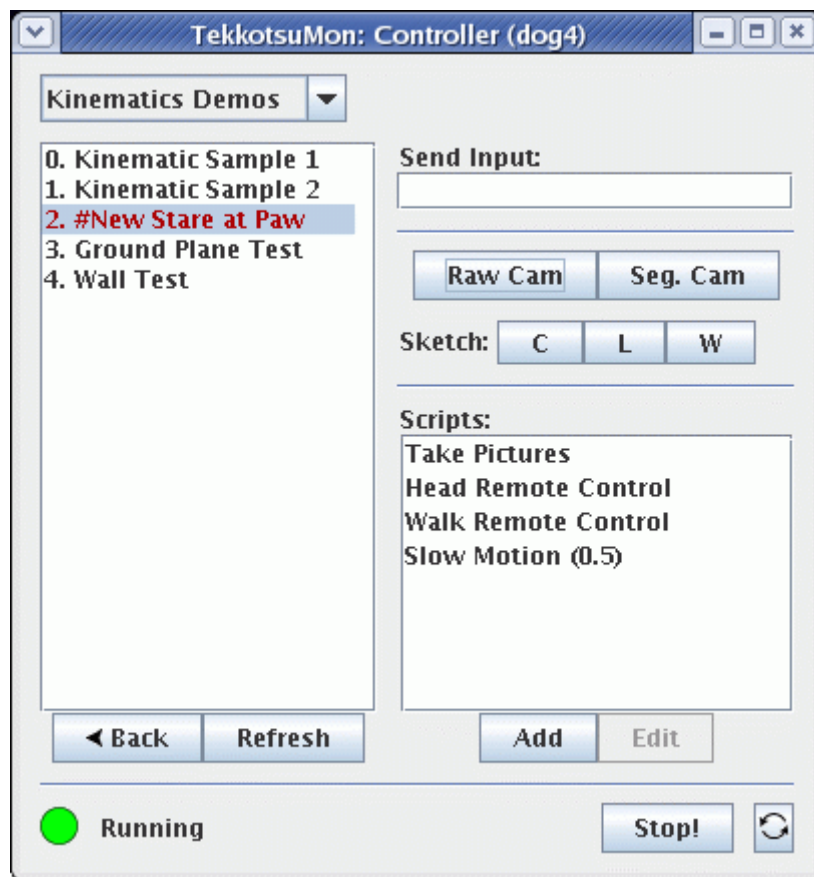


# Kinematic Chains



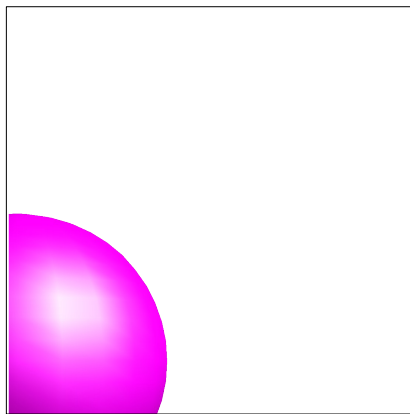
# Kinematics Demo: Keep Camera Pointed at Moving Paw

- Root Control > Mode Switch > Kinematics Demos > New Stare At Paw

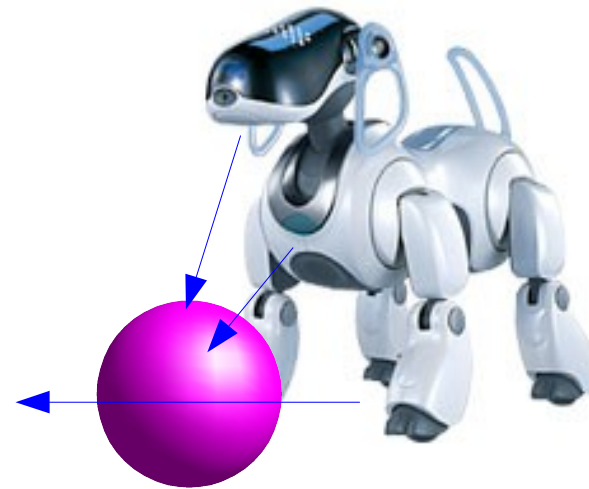


# Ideas from Cognitive Science

- Visual routines, dual coding theory, gestalt perception, affordances, ...
- Active research area for cognitive robotics.

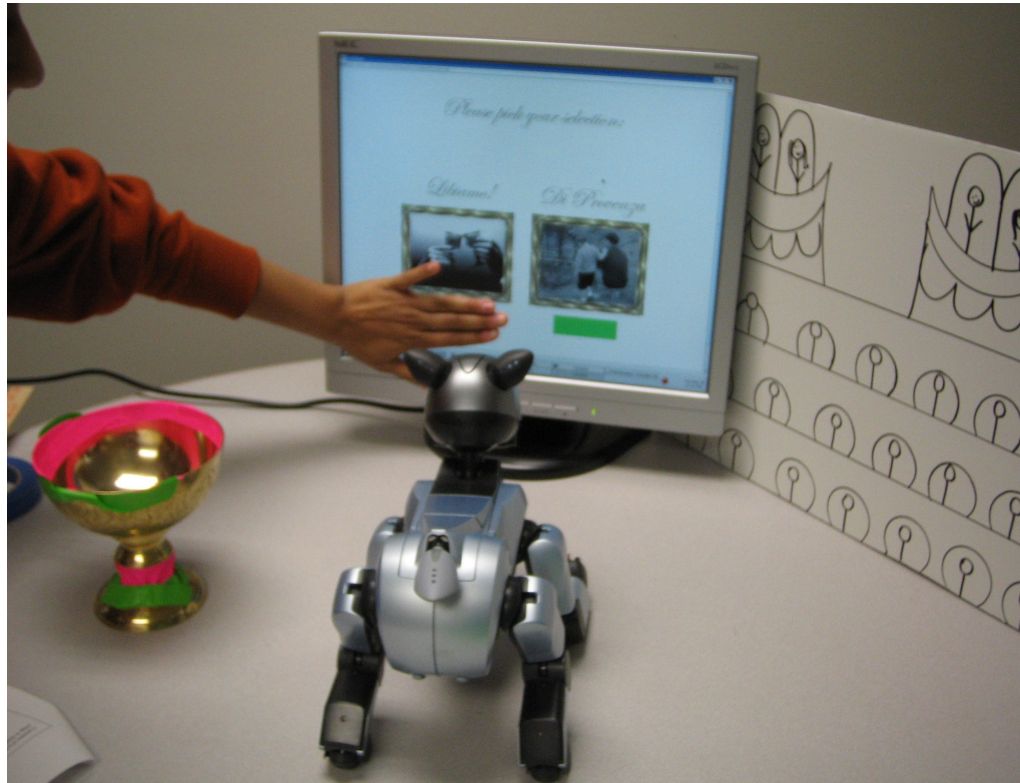


Camera view:  
"I see a pink blob"



Affordances:  
"I see something I can push"

# Human-Robot Interaction



A duet from Verdi's *La Traviata*  
(LookingGlass project by Kirtane & Libby)

# CMU's Cognitive Robotics Course

- First taught Spring 2006 (11 students).  
Second time Spring 2007 (12 students).
- Targeted at juniors and seniors, but some advanced sophomores enroll.
- Two 50 minute lectures and one 80 minute lab per week. 10 labs total.
- Last 3-4 weeks are devoted to project clinics and final project presentations.

# Lecture Topics

- AIBO Overview
- C++ for Java programmers
- Tekkotsu Behaviors and Events
- Motion Commands
- Vision pipeline
- Color image segmentation
- Ullman's visual routines
- Tekkotsu's sketches
- Paivio's dual coding theory
- Tekkotsu shape primitives
- Visual parsing
- Map building
- State machines
- Architectures for robot control
- World maps and localization (particle filtering)
- Navigation with the Pilot

# Lecture Topics (cont.)

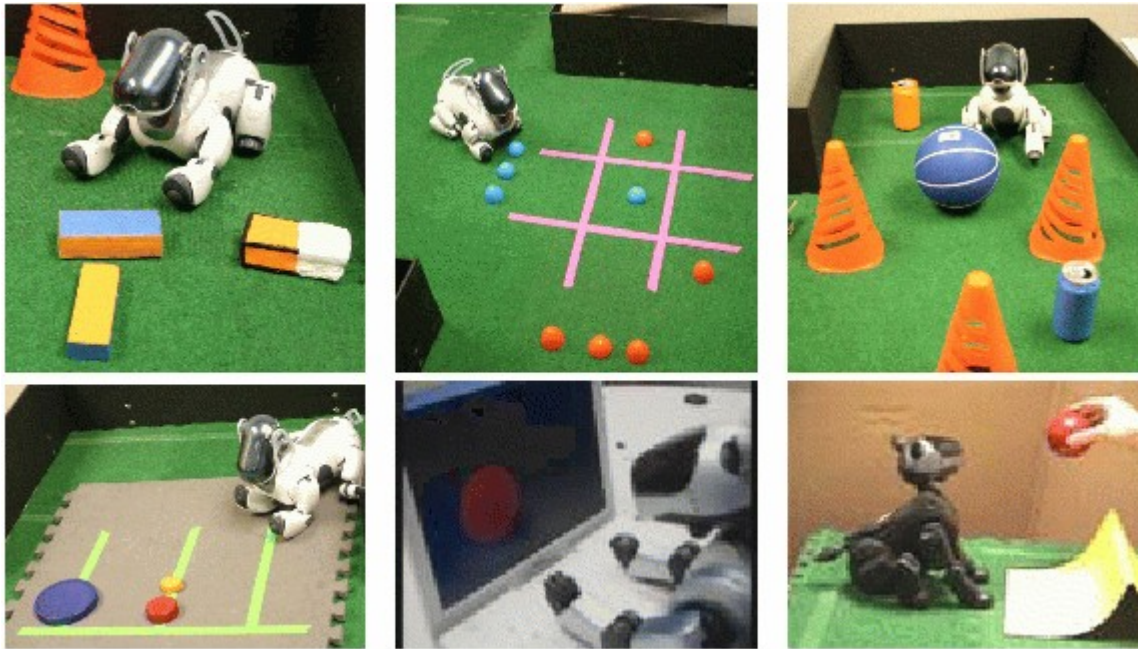
- Object recognition
- Gestalt perception
- Kuipers' “tracker” theory of consciousness
- Postures and motion sequences
- Body representation
- Manipulation by pushing
- Manipulation with friction
- Gibson's theory of affordances
- Human-robot interaction
- The Looking Glass tool
- Robot learning

# Cognitive Robotics Labs

- 1) Compiling and running Tekkotsu programs
- 2) Motion commands: moving the body
- 3) Visual routines and the SketchGUI tool
- 4) Using the map builder: camera vs. body coordinates
- 5) State machines and the Storyboard tool
- 6) Navigation with the Pilot
- 7) Gestalt perception for robots
- 8) Kinematics: body representations; inverse kin. solver
- 9) Human-robot interaction & the Looking Glass display
- 10) Manipulation: grasping and transporting objects



# Robot Tasks



# What Do Students Learn?

- Reference frames and coordinate transforms.
- Kinematics, Denavit-Hartenberg conventions, coordinate transformation matrices.
- Machine vision: color segmentation, line extraction, object recognition, etc. Learn by using the primitives.
- Dealing with sensor limitations: camera field of view, lighting adaptation, pixel noise, encoder error, etc.
- Object-oriented and event-based programming.
- Working with large software systems.
- Mechanisms for specifying robot behaviors.
- A touch of cognitive science.

# Broadening Participation in Computing (NSF)

- Robots are appealing to both male and female undergraduate CS majors.
- Spelman College is using Tekkotsu to introduce African-American women to robotics research.
- The Spelbots made RoboCup history.
- Three other HBCUs are now using Tekkotsu.

# Spelman College Spelbots Robot Soccer Team



# C.A.R.E. Project

- C.A.R.E. = “Computer and Robotics Education for African-American Students”.
- Joint project between CMU and Spelman, funded by NSF BPC program.
- Establish Tekkotsu robotics labs at HBCUs.
  - Set up equipment, install software, train staff.
- Build a community of educators and students who are proficient at cognitive robotics programming.
  - Share educational materials, software, and ideas.

# C.A.R.E Schools

- Hampton University (Hampton, VA)
  - Chutima Boonthum: introducing robotics into an undergraduate AI course.
- University of the District of Columbia
  - LaVonne Manning: LSAMP summer program; cognitive robotics course.
- Florida A&M (Tallahassee)
  - Clement Allen: will use Tekkotsu in an introductory data structures course to “prime the pump” for a robotics/AI course.

# How to Obtain Tekkotsu

- Stable release:

<http://Tekkotsu.org>



- “Bleeding-edge” development version:  
see CVS instructions at  
<http://tekkotsu.no-ip.org>  
<http://cvs.tekkotsu.org>
- Contact us for Mac installation goodies  
(Xcode templates, etc.)

# Tekkotsu Resources

- Extensive on-line documentation at [Tekkotsu.org](http://Tekkotsu.org)
- Tutorial:  
<http://www.cs.cmu.edu/~dst/Tekkotsu/Tutorial>
- Cognitive robotics course:  
<http://www.cs.cmu.edu/afs/cs/academic/class/15494-s07>
- Tekkotsu users group:  
[groups.yahoo.com/groups/tekkotsu\\_dev](http://groups.yahoo.com/groups/tekkotsu_dev)



# Setting Up A Tekkotsu Lab

- Six robot / workstation pairs; extra workstation for the instructor.
- Wireless access point with cabled connections to the workstations.
- 24/7 keycard access for everyone.
- “Playpen” for robots to run around in.



# How to Buy an AIBO on eBay

- Only buy an ERS-7. No significant differences among ERS7-M1 to -M3.
- Market price \$2500 to \$3000 (new ERS-7)
- One-day auction = scam.
- Multiple AIBOs for sale = scam.
- Request for Western Union money order payment = scam. Pay only via PayPal.
- Massive electronics sales by people who formerly sold dolls = hijacked account.

# Timeline for Other Platforms

- Qwerkbot+ interface (almost) ready.
- Initial Lynx Motion arm support running.
- Summer goal:  
a mobile arm.
- Public release planned  
for the fall.



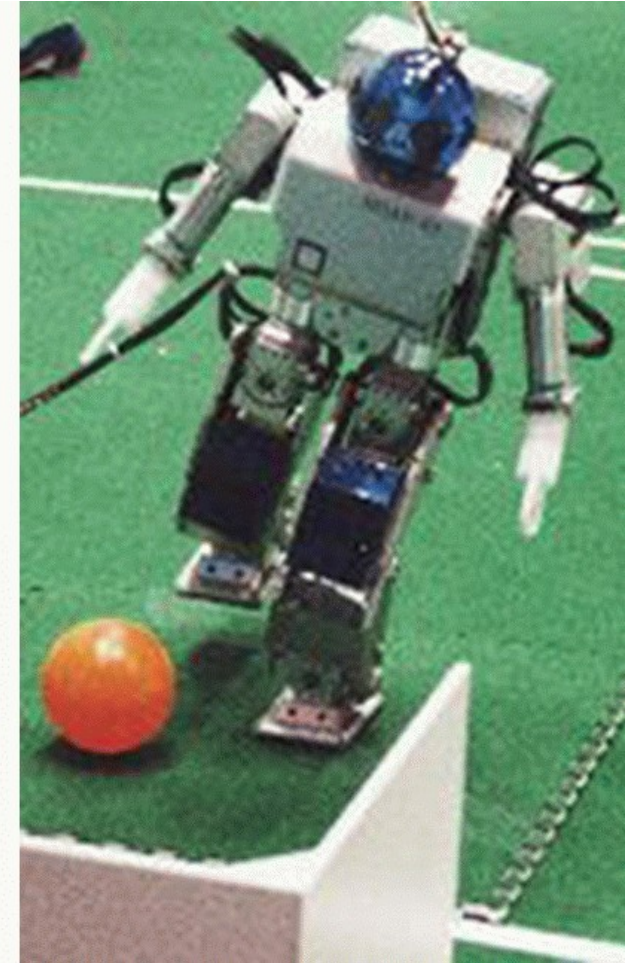
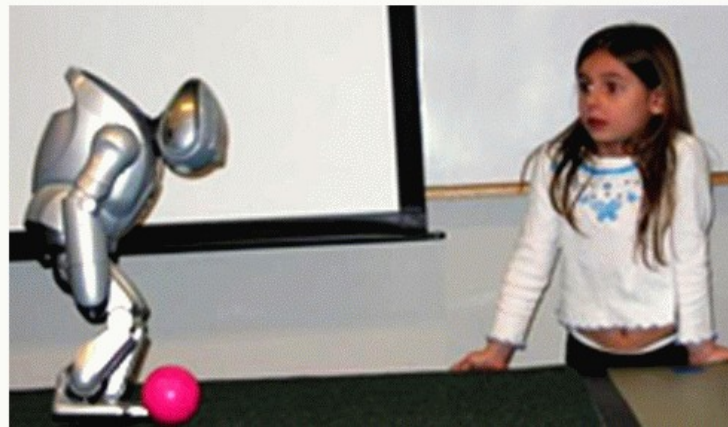
# The Future of Cognitive Robotics



Honda's Asimo



Sony's QRIO (defunct)



Fujitsu's HOAP-2