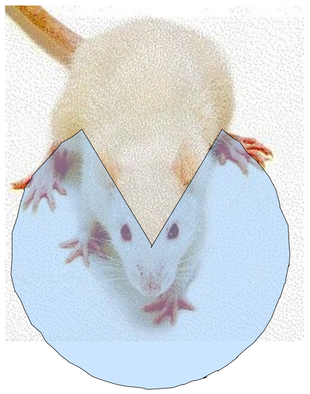
The Map Builder

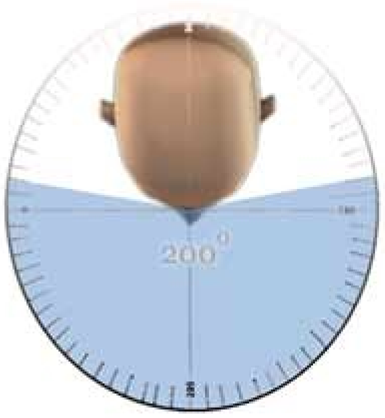
15-494 Cognitive Robotics
David S. Touretzky &
Ethan Tira-Thompson

Carnegie Mellon
Spring 2009

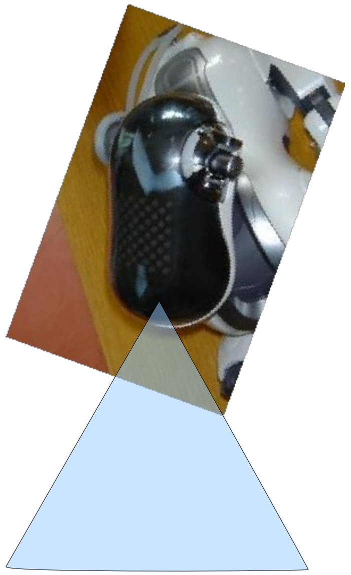
Horizontal Field of View



Rat: 300 deg.



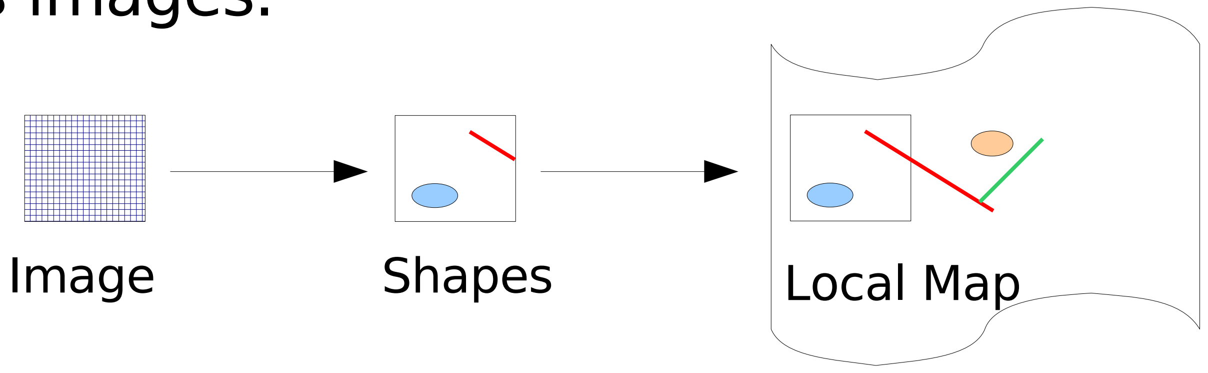
Human: 200 deg.



Typical robot: 60 deg.

Seeing A Bigger Picture

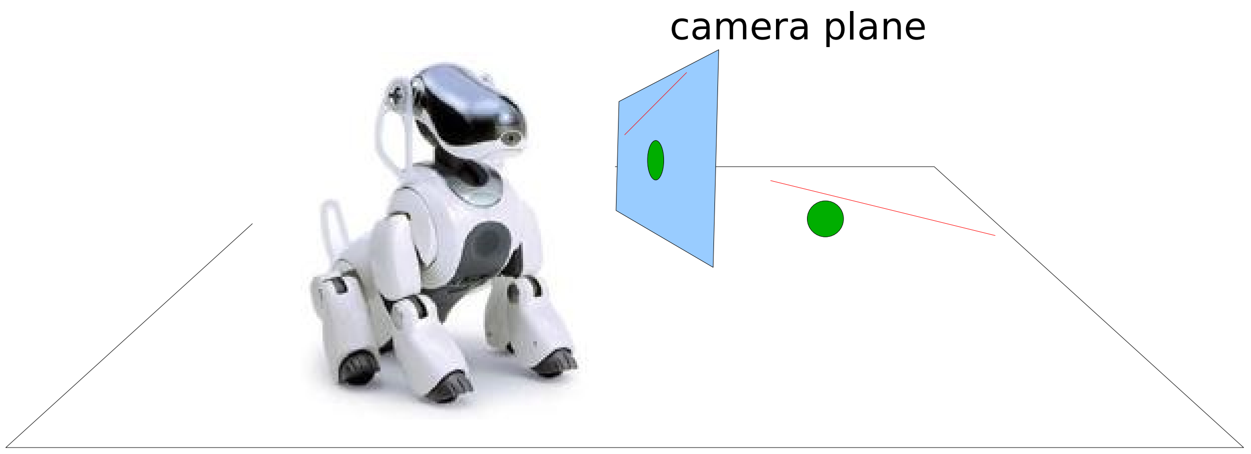
- How can we assemble an accurate view of the robot's surroundings from a series of narrow camera frames?
- First, convert each image to symbolic form: shapes.
- Then, match the shapes in one image against the shapes in previous images.



- 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

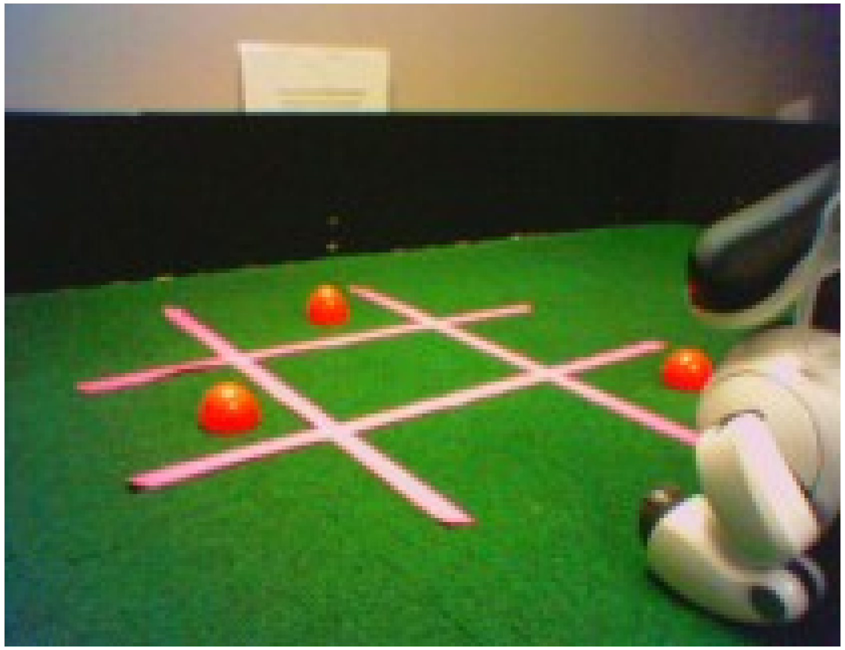
- How do we convert from camera-centered coordinates to body-centered coordinates?
- Need to know the camera pose: can get that from the kinematics system.
- Unfortunately, that's not enough.
- Add a planar world assumption: objects lie in the plane. The robot is standing on that plane.
- Now we can get object coordinates in the body frame.

Shape Spaces

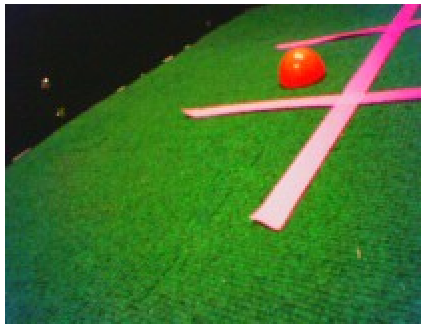
- camShS = camera space
- groundShS = camera shapes projected to ground plane
- localShS = body-centered (egocentric space);
  constructed by matching and importing shapes
  from groundShS across multiple images
- worldShS = world space (allocentric space);
  constructed by matching and importing shapes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

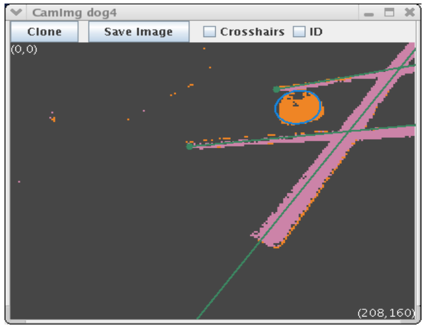
Invoking The Map Builder

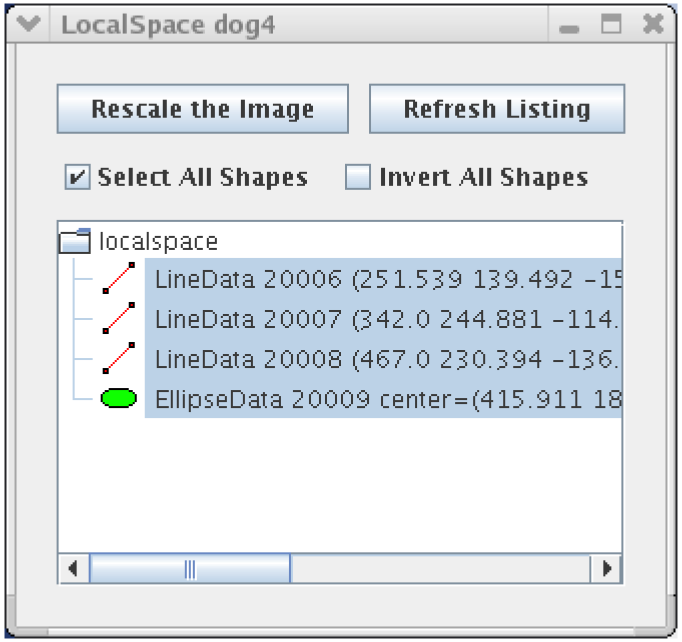
- Let's map the tic-tac-toe board:



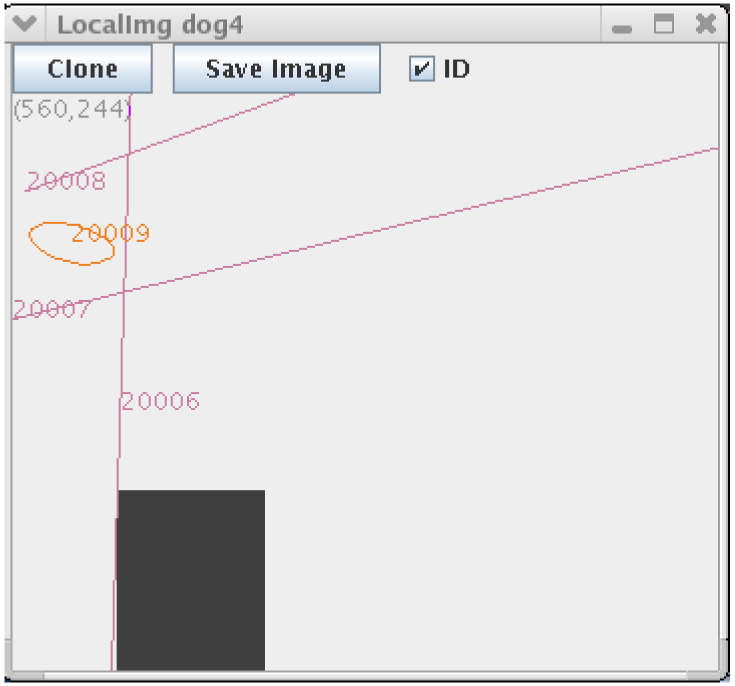
Frame 1



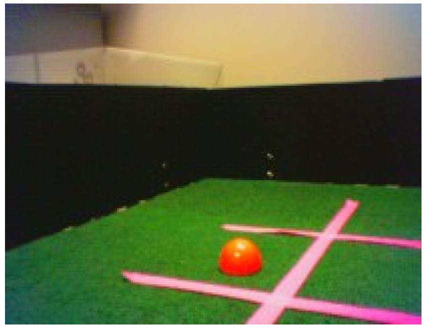


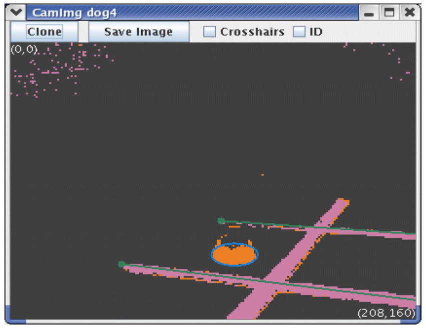


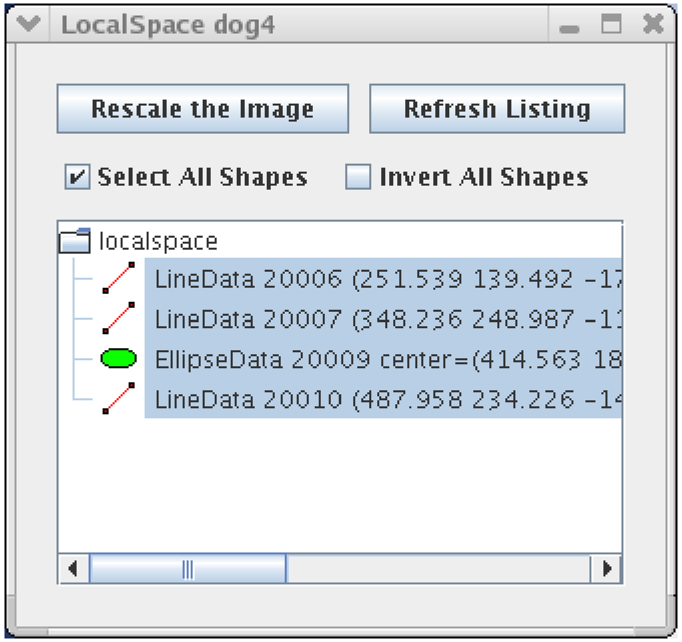
localspace
LineData 20006 (251.539 139.492 -15
LineData 20007 (342.0 244.881 -114.
LineData 20008 (467.0 230.394 -136.
EllipseData 20009 center=(415.911 18



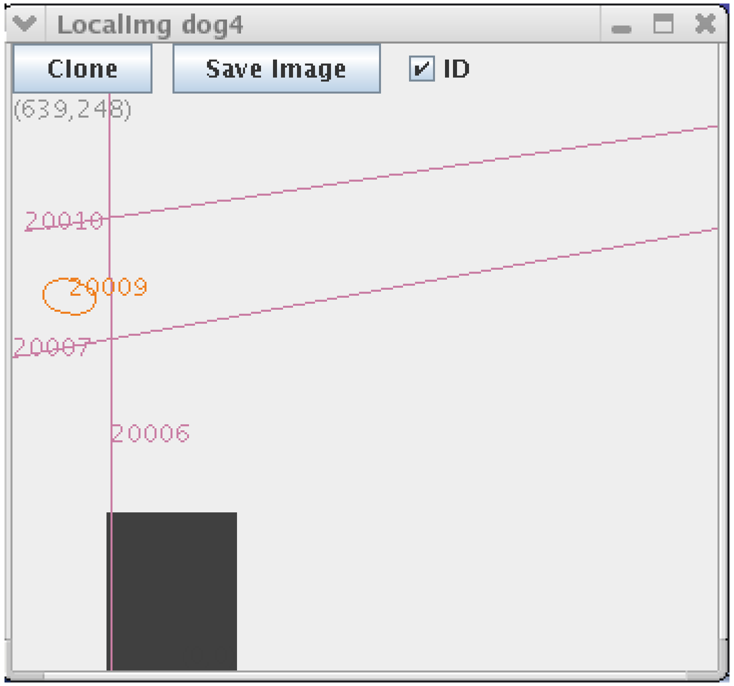
Frame 2



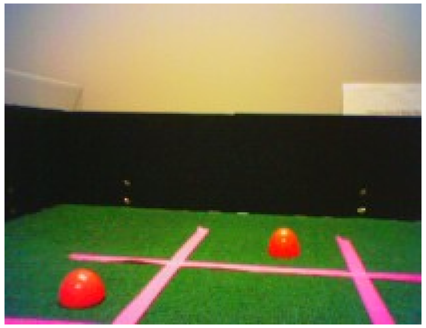


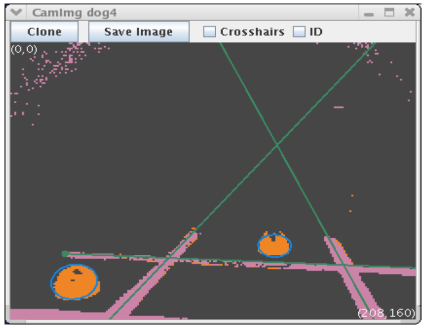


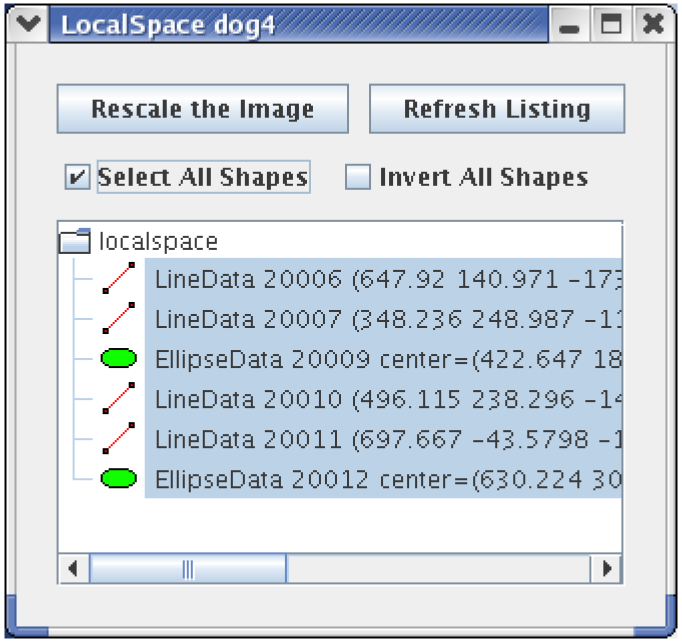
```
LineData 20006 (251.539 139.492 -17
LineData 20007 (348.236 248.987 -1
EllipseData 20009 center=(414.563 18
LineData 20010 (487.958 234.226 -14
```



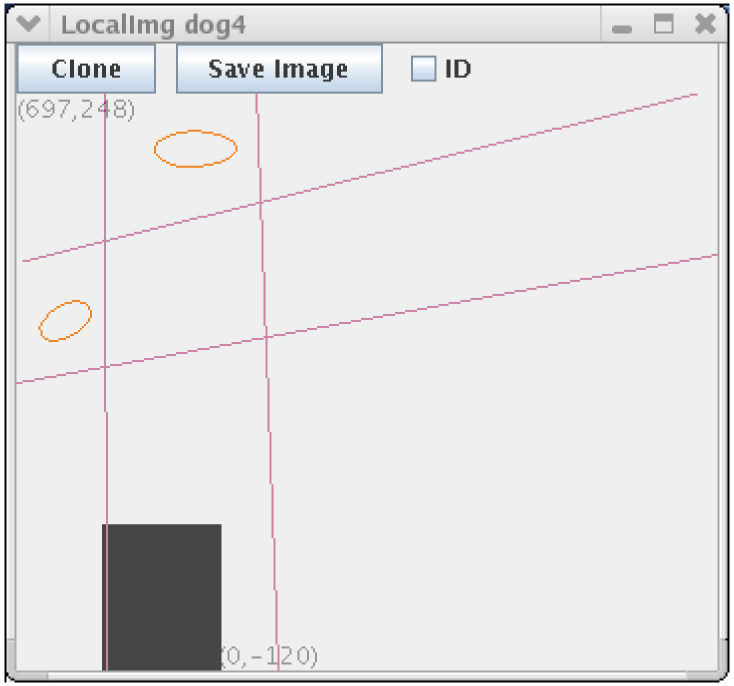
Frame 3



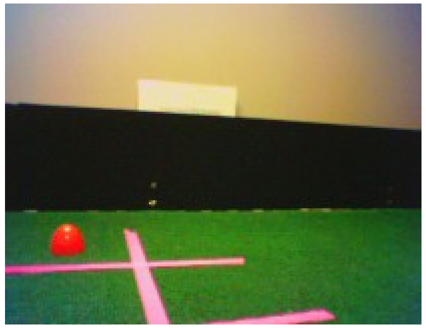


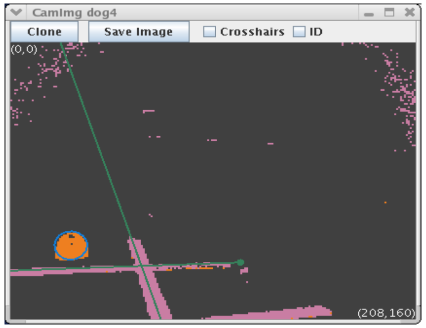


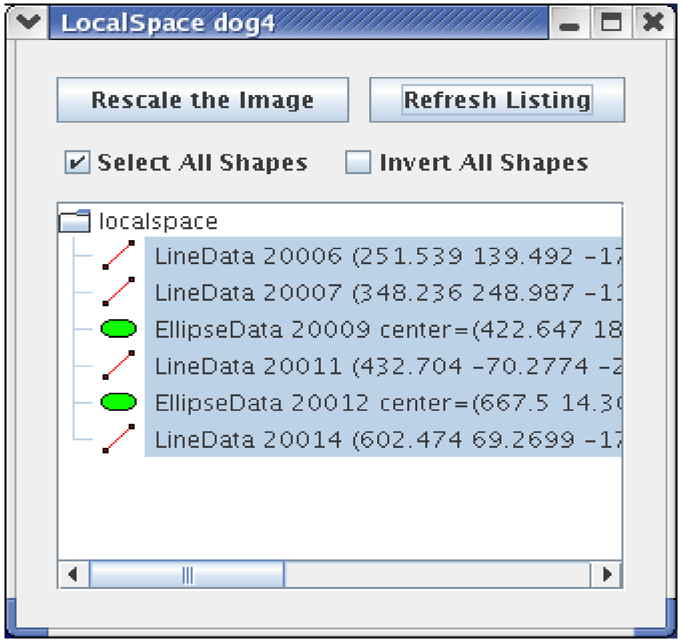
localspace
- LineData 20006 (647.92 140.971 -17
- LineData 20007 (348.236 248.987 -1
- EllipseData 20009 center=(422.647 18
- LineData 20010 (496.115 238.296 -1
- LineData 20011 (697.667 -43.5798 -1
- EllipseData 20012 center=(630.224 30



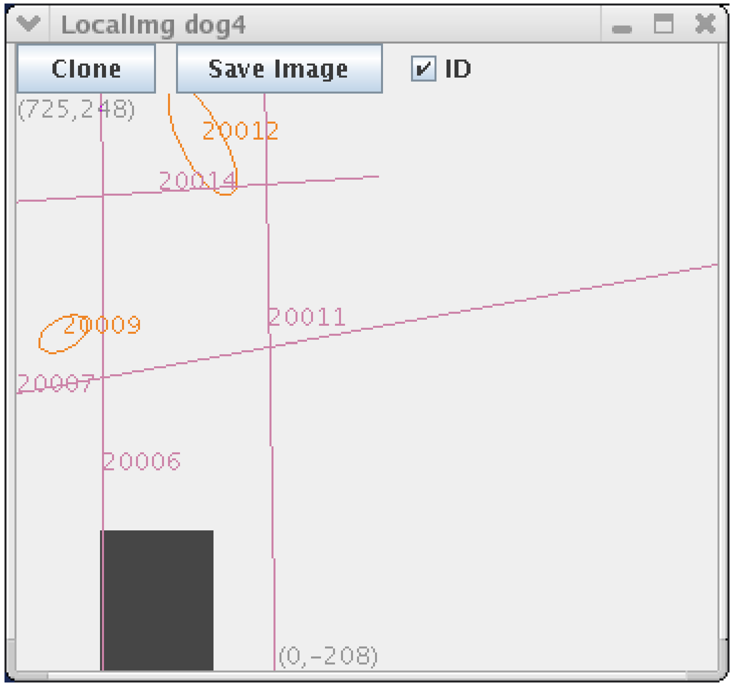
Frame 4



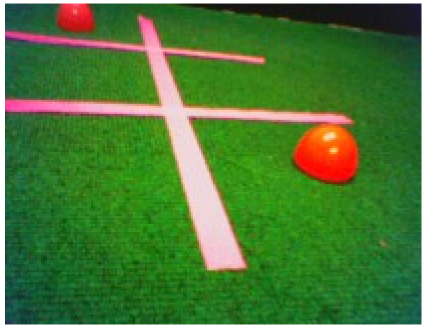


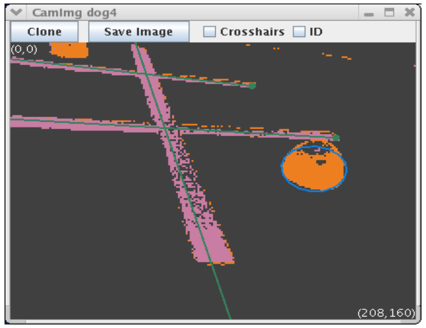


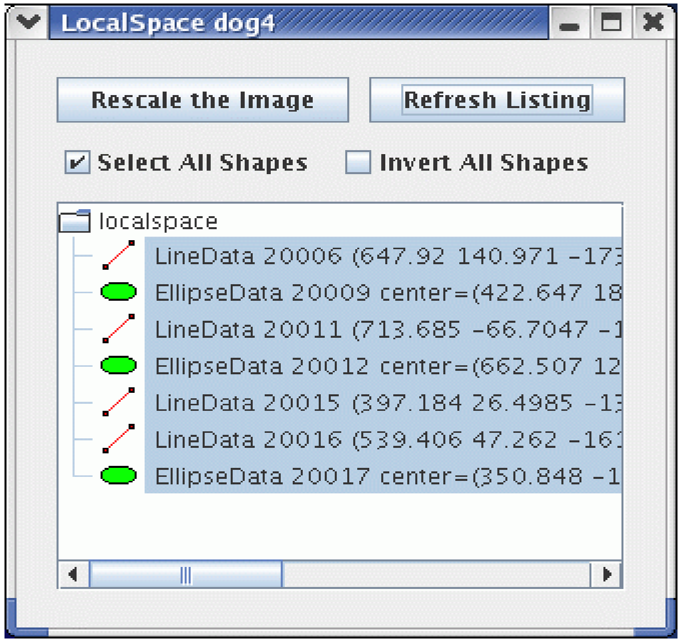
localspace
- LineData 20006 (251.539 139.492 -17
- LineData 20007 (348.236 248.987 -1
- EllipseData 20009 center=(422.647 18
- LineData 20011 (432.704 -70.2774 -2
- EllipseData 20012 center=(667.5 14.3
- LineData 20014 (602.474 69.2699 -17



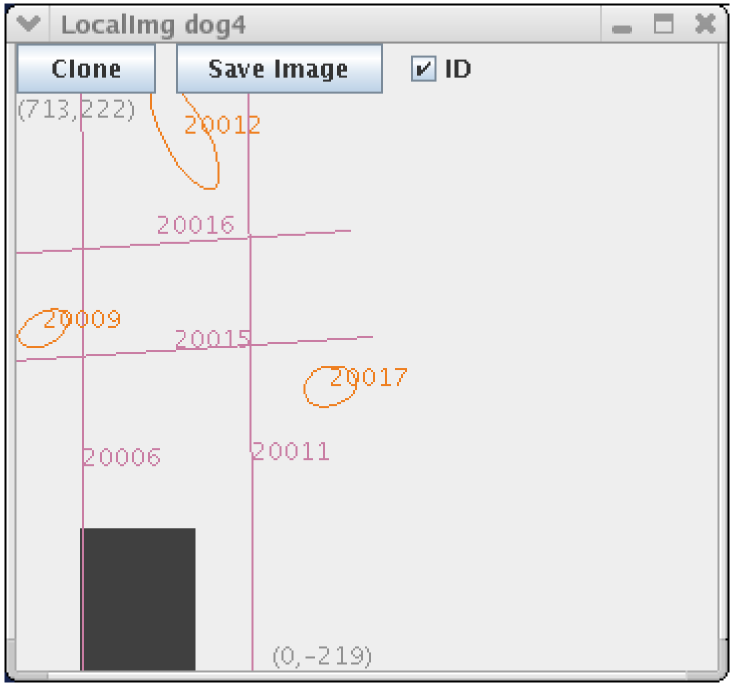
Frame 5



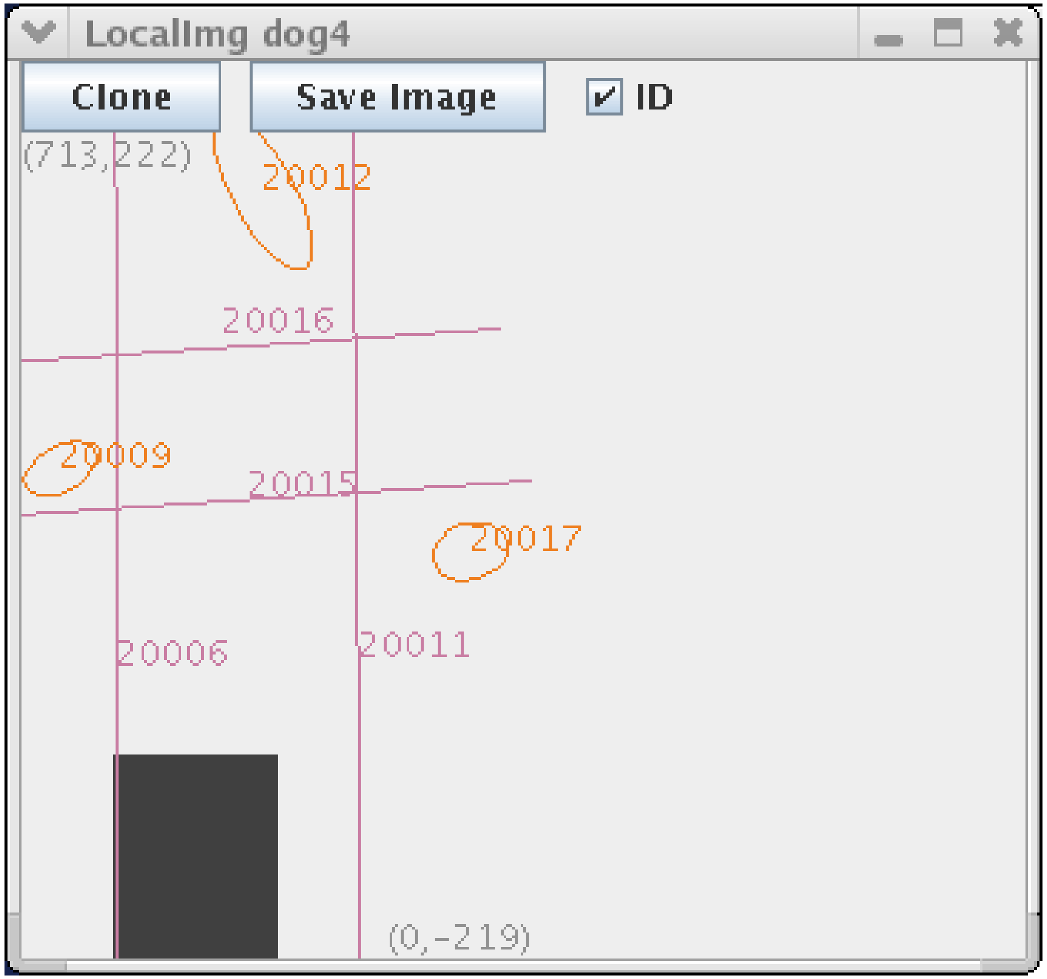




LineData 20006 (647.92 140.971 -17
EllipseData 20009 center=(422.647 18
LineData 20011 (713.685 -66.7047 -1
EllipseData 20012 center=(662.507 12
LineData 20015 (397.184 26.4985 -1
LineData 20016 (539.406 47.262 -16
EllipseData 20017 center=(350.848 -1



Final Local Map



Shape Matching Algorithm

- Shape type and color must match exactly.
- Coordinates must be a reasonably close match for points, blobs, and ellipses.
- Lines are special, because endpoints may be invalid:
  - If endpoints are valid, coordinates should match.
  - If invalid in local map but valid in ground space, update the local map to reflect the true endpoint location.
- Coordinates are updated by weighted averaging.

Noise Removal

- Noise in the image can cause spurious shapes. A long line might appear as 2 short lines separated by a gap, or a noisy region might appear as a short line.
- Assign a confidence value to each shape in local map.
- Each time a shape is seen: increase its confidence.
- If a shape *should* be seen but is not, decrease its confidence.
- Delete shapes with negative confidence.

Where to Look?

- Start with the shapes visible in the camera frame.
- Move the camera to fixate each shape: get a better look.
- If a line runs off the edge of the camera frame, move the camera to try to find the line's endpoints.
  - If the head can't rotate any further, give up on that endpoint.
- If an object is partially cut off by the camera frame, don't add it to the map because we don't know its true shape.
  - Move the camera to bring the object into view.

Programming the MapBuilder

- A instance of MapBuilder called mapbuilder is included as a member of VisualRoutinesBehavior, and VisualRoutinesStateNode is a child of that class.

```
#include "DualCoding/DualCoding.h"

class LocalMapDemo : public VisualRoutinesStateNode {
public:
  LocalMapDemo() : VisualRoutinesStateNode() {}

  virtual void DoStart() {

        MapBuilderRequest req;

         ... program the mapbuilder instructions

        mapbuilder.executeRequest(req);
  }

};
```

Two Ways To Get Results From the MapBuilder

- If only the current camera image is to be processed, the the results can be available immediately:

```
req.immediateRequest = true;
mapbuilder.executeRequest(req);
NEW_SHAPEVEC(... process the results from the MapBuilder...);
```

- If multiple camera images or any head motion is required, you must wait for a MapBuilder event:

```
mapreq_id = mapbuilder.executeRequest(req);
erouter->addListener(this, EventBase::mapbuilderEGID, mapreq_id);
...

void processEvent (const EventBase &e) {
  if ( e.getGeneratorID() == EventBase::mapbuilderEGID &&
       e.getSourceID() == mapreq_id ) {
   postStateCompletion();
  }
```

MapBuilderRequest Parameters

- RequestType
  - cameraMap
  - groundMap
  - localMap
  - worldMap
- Shape parameters:
  - objectColors
  - occluderColors
  - maxDist
  - minBlobArea
  - markerTypes
- Utility functions:
  - clearShapes
  - RawY
  - immediateRequest
- Lookout control:
  - motionSettleTime
  - numSamples
  - sampleInterval
  - pursueShapes
  - searchArea
  - doScan, dTheta
  - manualHeadMotion

Programming the MapBuilder

```
const int pink_index = ProjectInterface::getColorIndex("pink");
const int blue_index = ProjectInterface::getColorIndex("blue");
const int 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.pursueShapes = true;

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);
```

Examine the Results with Another VisualRoutinesStateNode

```
class ExamineMap : public VisualRoutinesStateNode {
  public:

    ExamineMap(const std::string& name) :
        VisualRoutinesStateNode(name) {}

    void DoStart() {
       cout << "MapBuilder found " << localShS.allShapes().size()
               << " shapes." << endl;
    }
};
```

Link the Nodes Together

```
class MyMapDemo : public VisualRoutinesStateNode {
    public:

      MyMapDemo(const std::string &name) :
        VisualRoutinesStateNode(name) {}

      virtual void setup() {

#statemachine
     startnode: BuildMap =C=> ExamineMap
#endstatemachine

     }

};
```

Parent state machine must be a VisualRoutinesStateNode if any of its children are.

Qualitative Spatial Reasoning

- Reading for today:
  How qualitative spatial reasoning can improve strategy game AIs
  Ken Forbus, James Mahoney, and Kevin Dill (2002)
- Uses visual routines to "reason about" maps, e.g., compute reachability, calculate paths, etc.
- Possible research topic: applying these ideas to world maps in Tekkotsu.