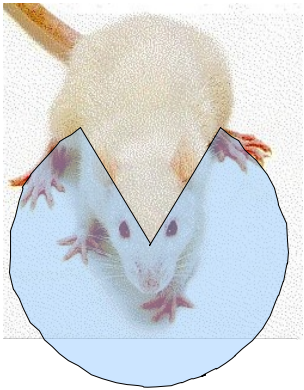


Local and World Maps

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

Carnegie Mellon
Spring 2012

Horizontal Field of View



Rat: 300 deg.



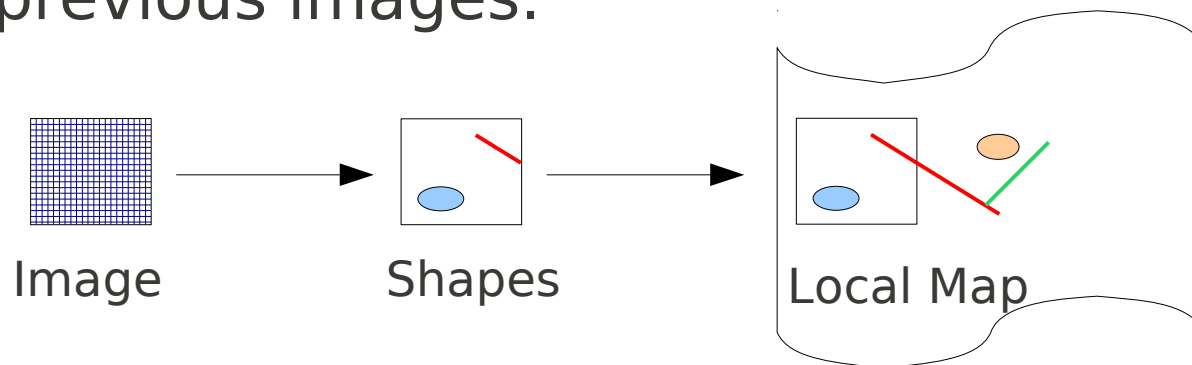
Human: 200 deg.



Typical webcam:
60 deg.

Seeing A Bigger Picture

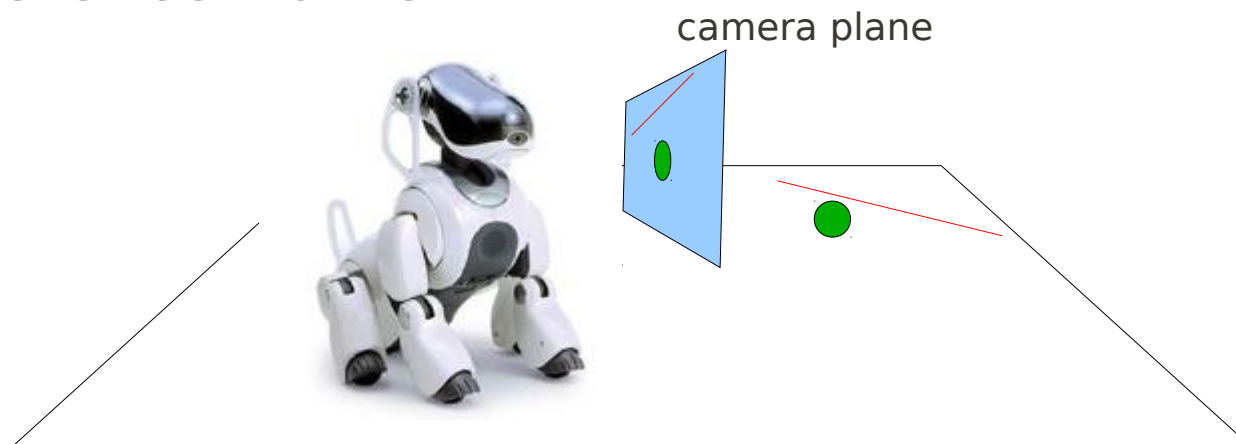
- How can we assemble an accurate view of the robot's surroundings from a series of narrow camera frames?
- First, convert each image to symbolic form: shapes.
- Then, match the shapes in one image against the shapes in previous images.



- 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

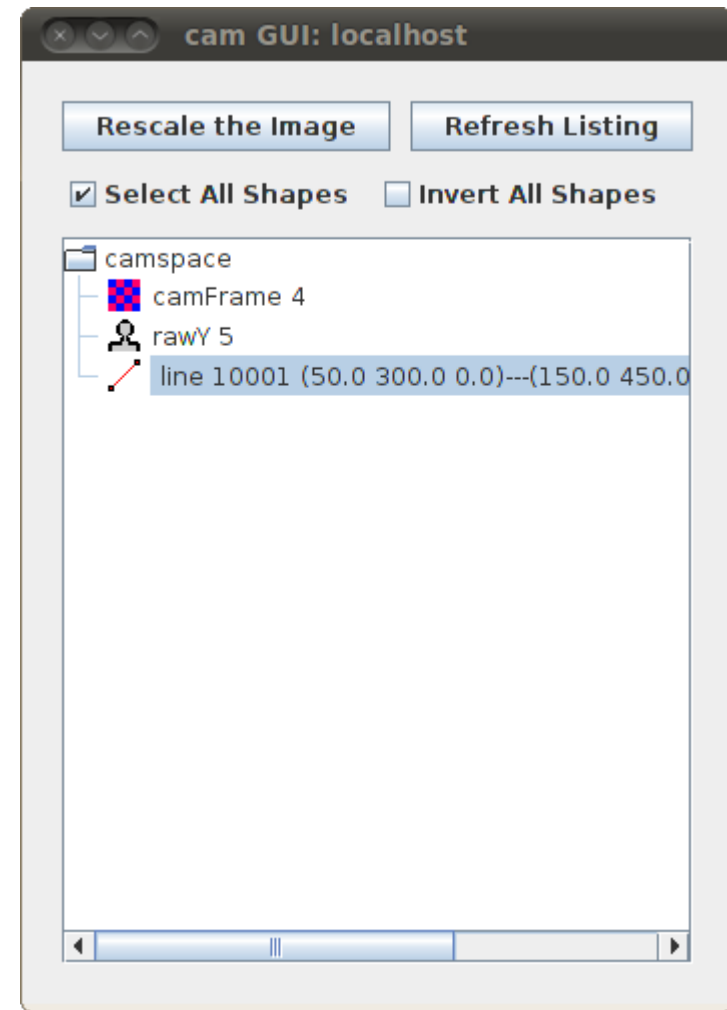
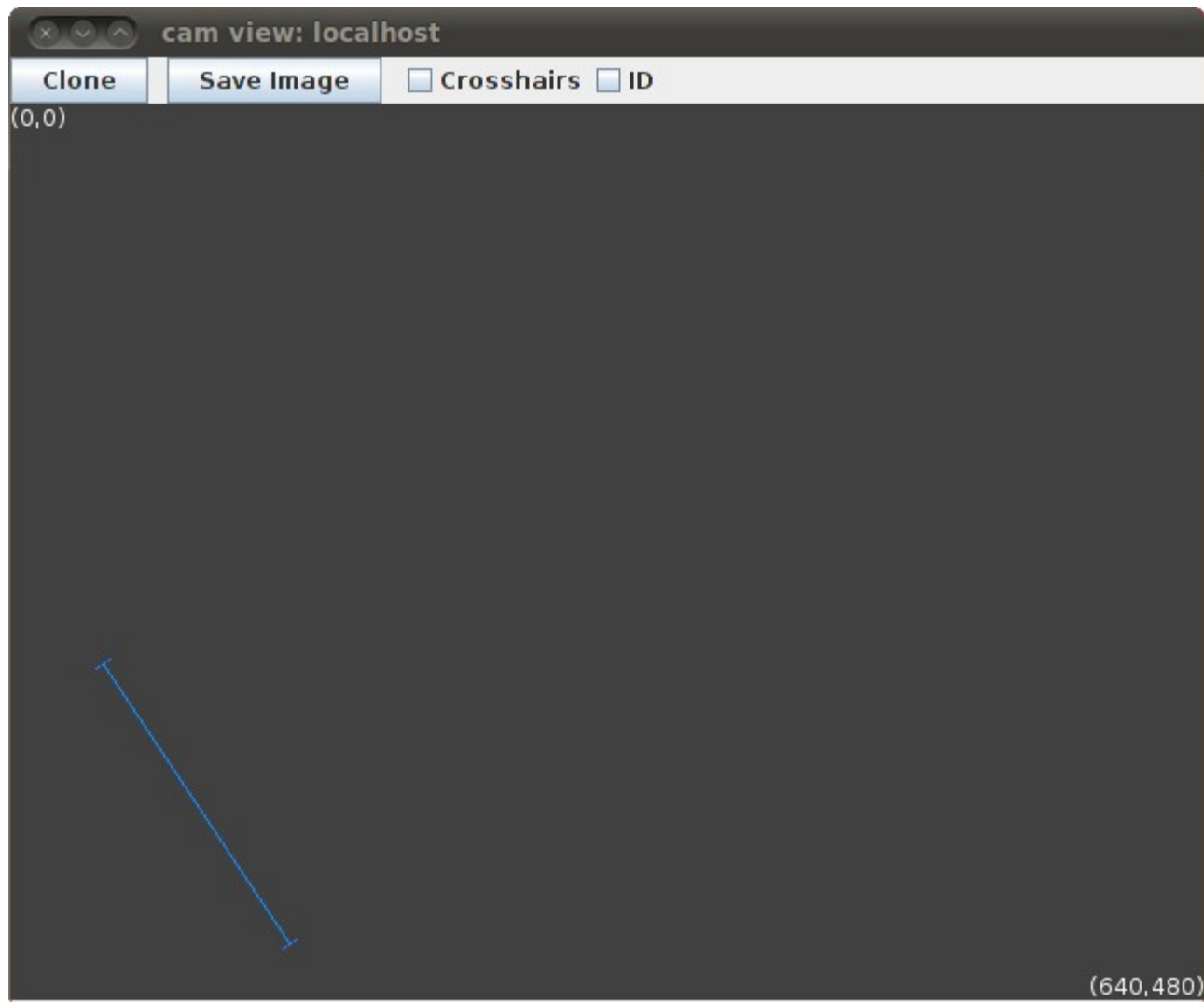
MapBuilderNode

- MapBuilderNode takes a constructor argument specifying the kind of map to build.
- Types of maps:
 - cameraMap (default)
 - localMap
 - worldMap
- If we ask for a localMap, the MapBuilder will project shapes from camera space to ground space and then import them into the local map.

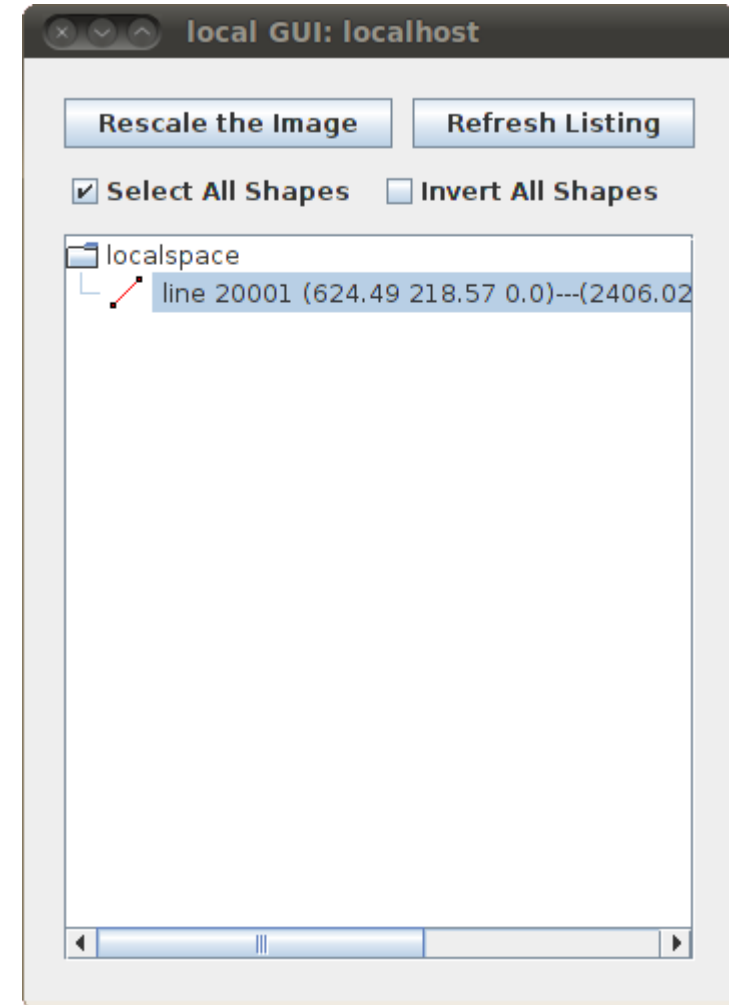
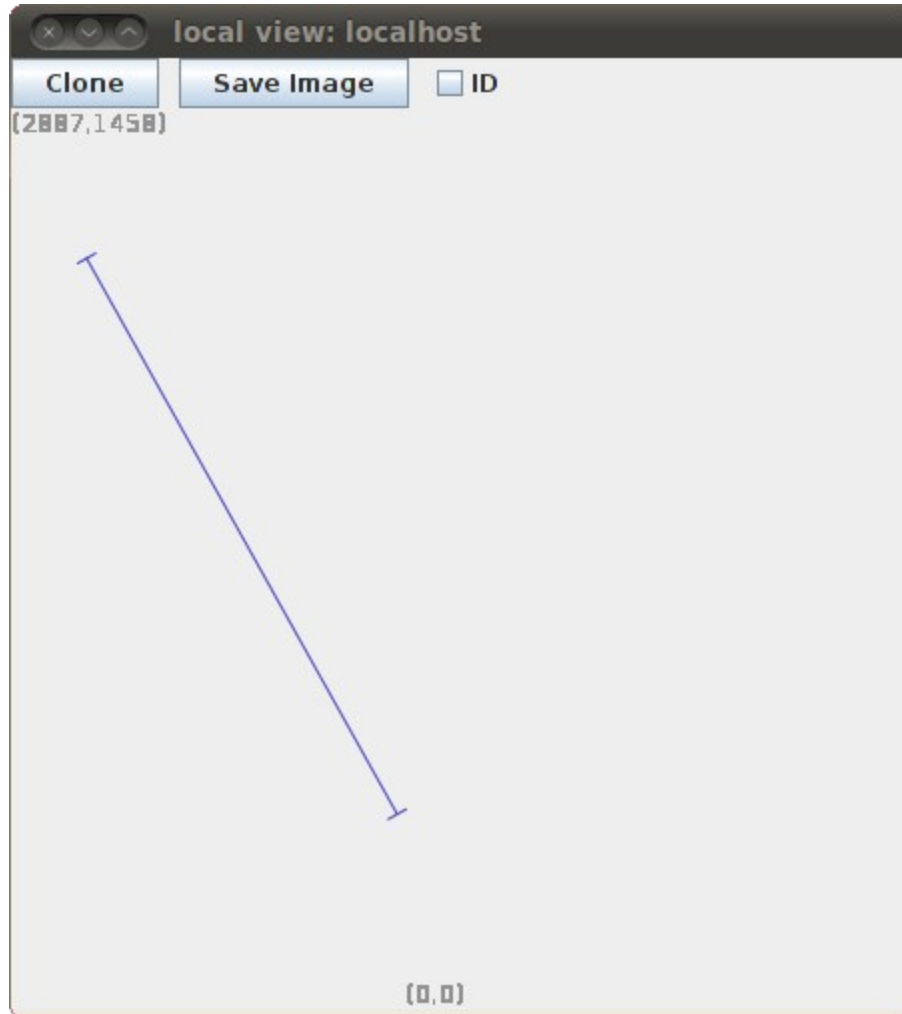
Projecting To Ground

```
$nodeclass Example : VisualRoutinesStateNode {  
  // Construct line in camera space; MapBuilder projects to local space  
  
  $nodeclass BuildIt : VisualRoutinesStateNode : doStart {  
    NEW_SHAPE(line, LineData, new LineData(camShS, Point(50,300),  
                                              Point(150,450)));  
    line->setColor("blue");  
  }  
  
  $nodeclass ProjectIt :  
    MapBuilderNode(MapBuilderRequest::localMap) : doStart {  
    mapreq.clearCamera = false;  
  }  
  
  $setupmachine{  
    BuildIt =N=> ProjectIt =C=> SpeechNode("Done")  
  }  
}
```


Camera Space



Local Space



pursueShapes

- By default, the MapBuilder uses one image: whatever view the camera is currently providing.
- We can ask it to move the camera around and construct a local view from multiple images.
- Set `pursueShapes = true` to give the MapBuilder permission to move the camera..

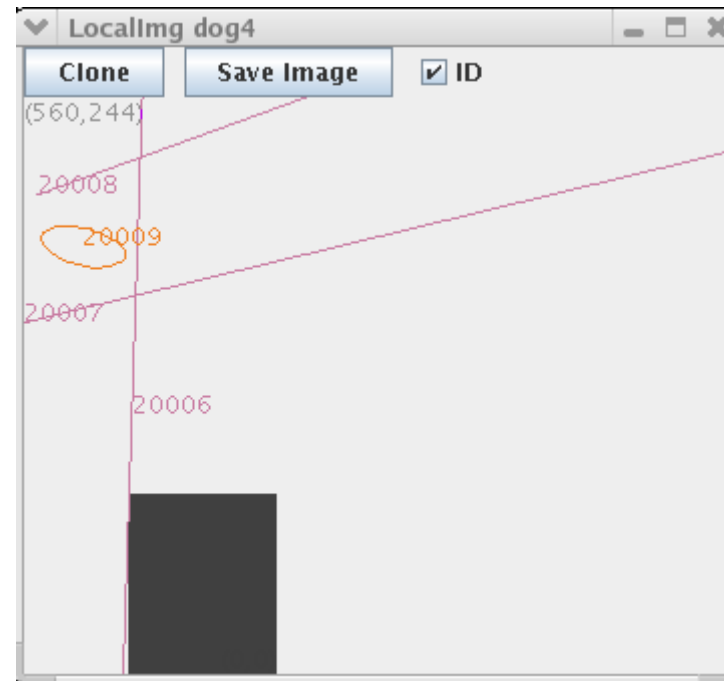
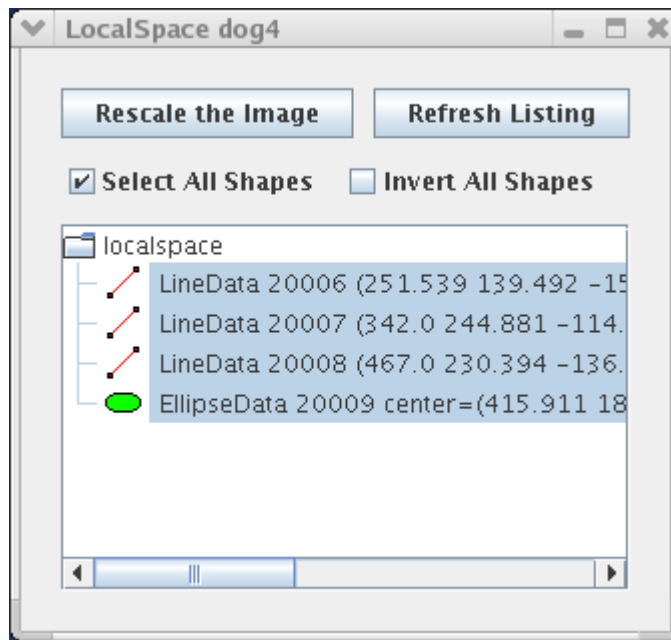
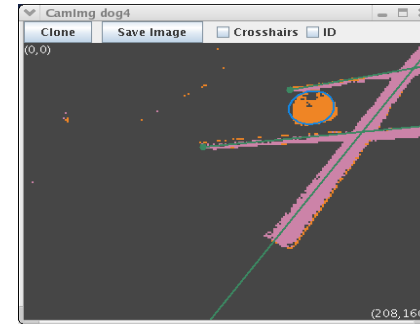
```
$nodeclass FindLines :  
    MapBuilderNode(MapBuilderRequest::localMap) : doStart {  
    mapreq.addObjectColor(lineDataType,"blue");  
    mapreq.pursueShapes = true;  
}
```

Invoking The Map Builder

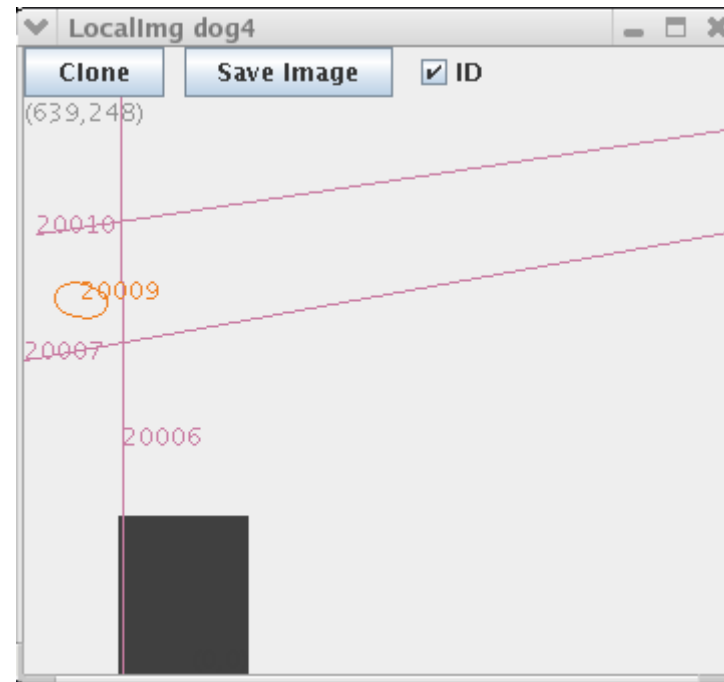
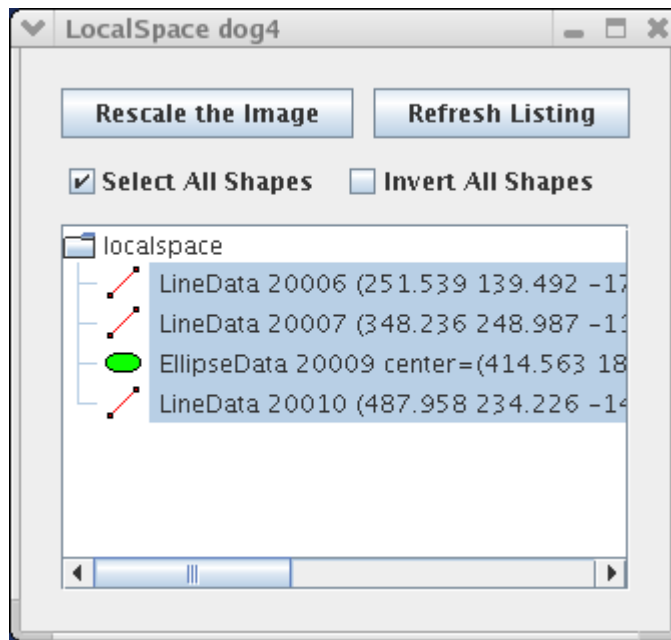
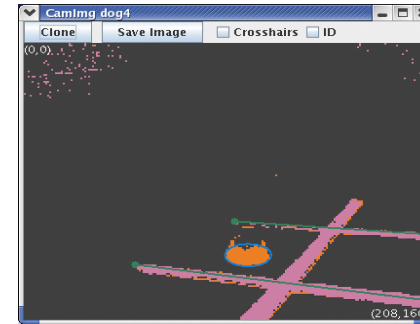
- Let's map the tic-tac-toe board:



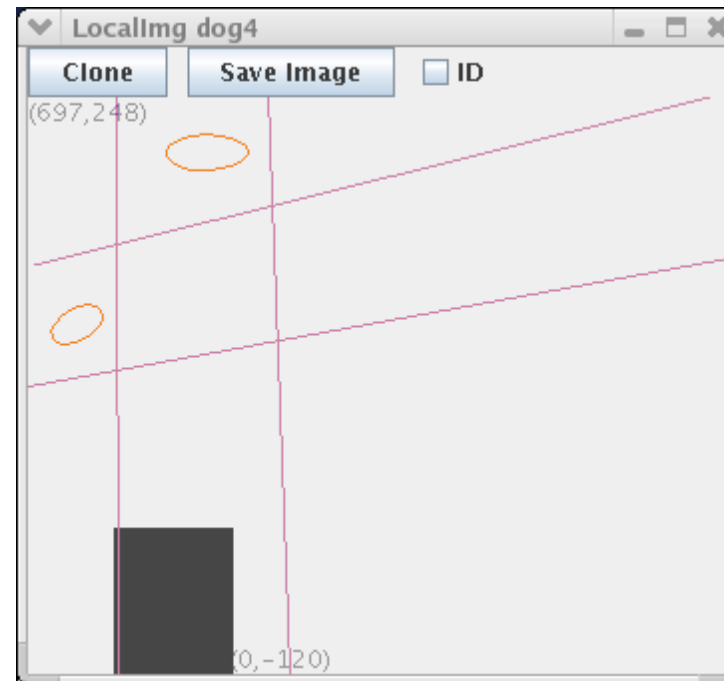
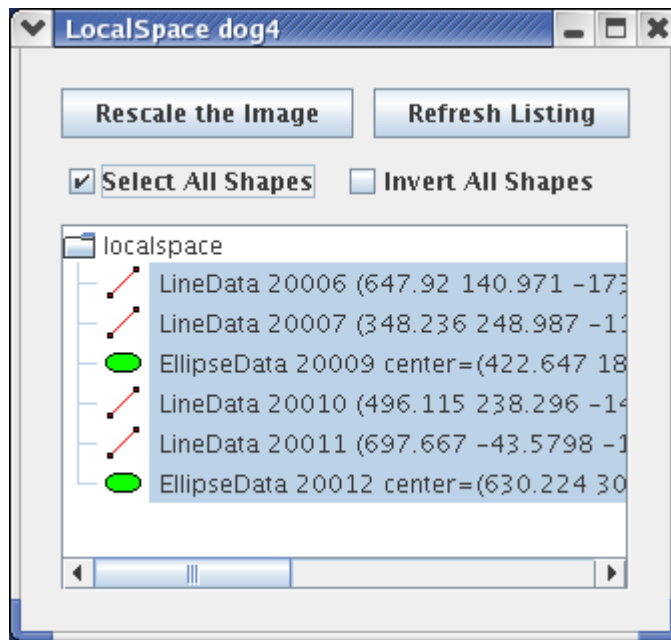
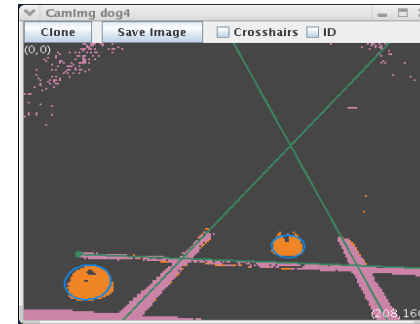
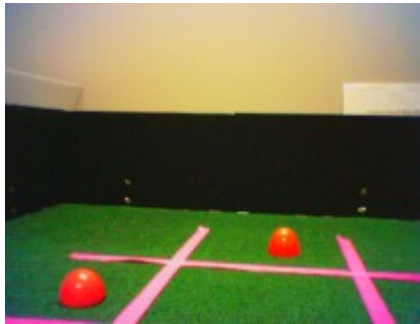
Frame 1



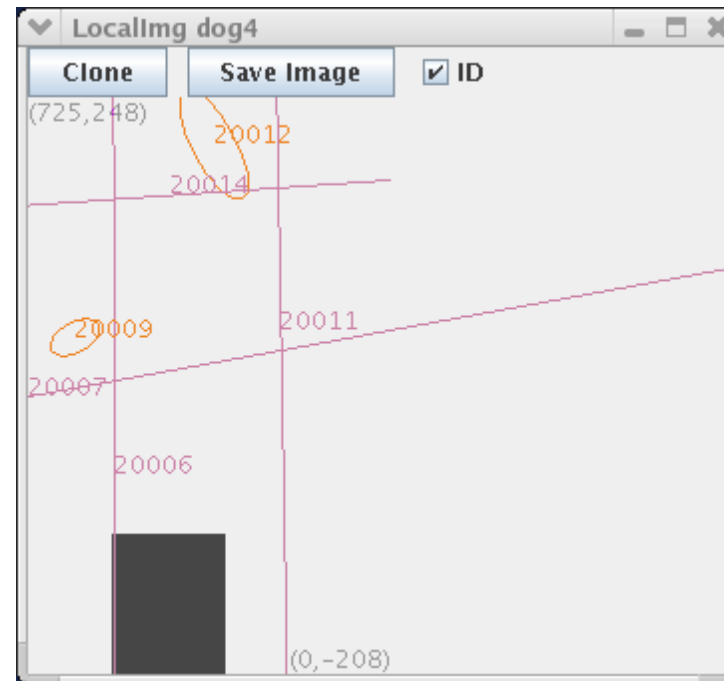
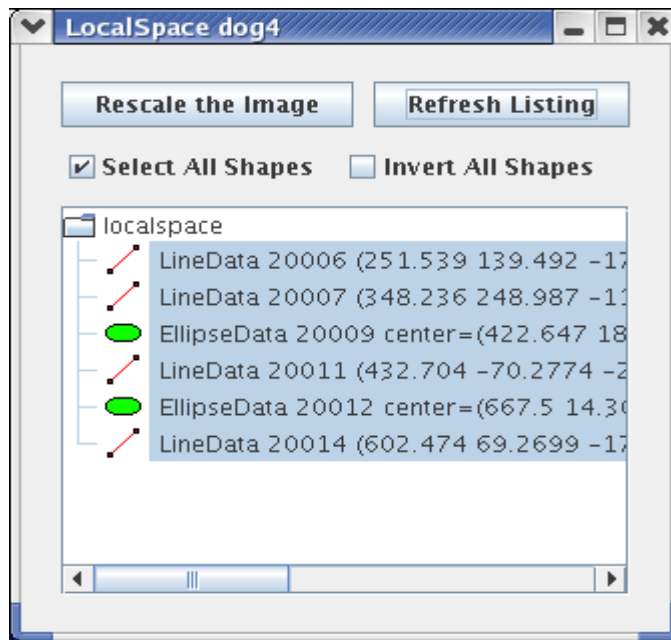
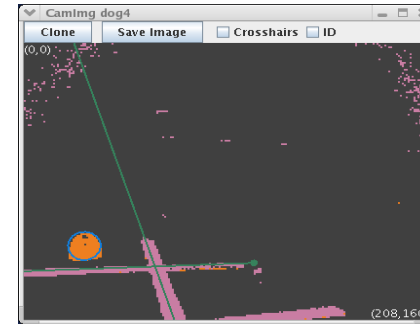
Frame 2



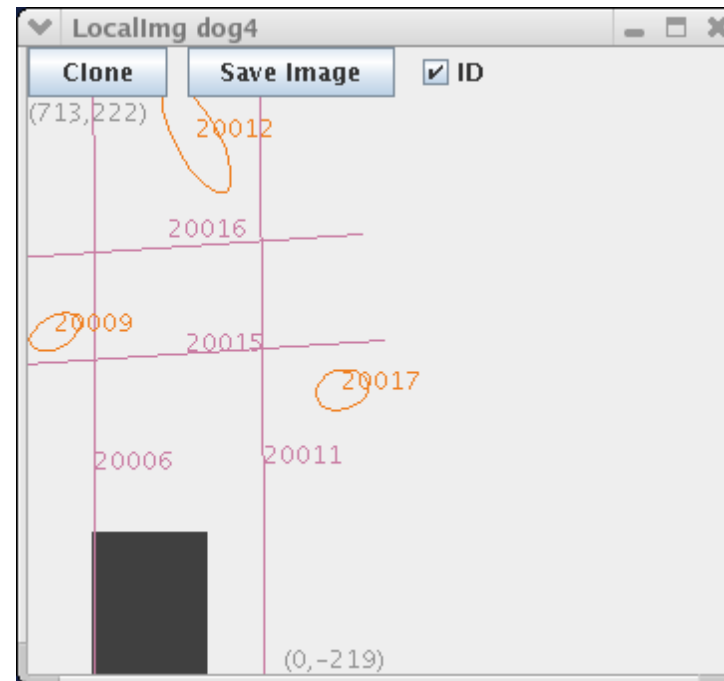
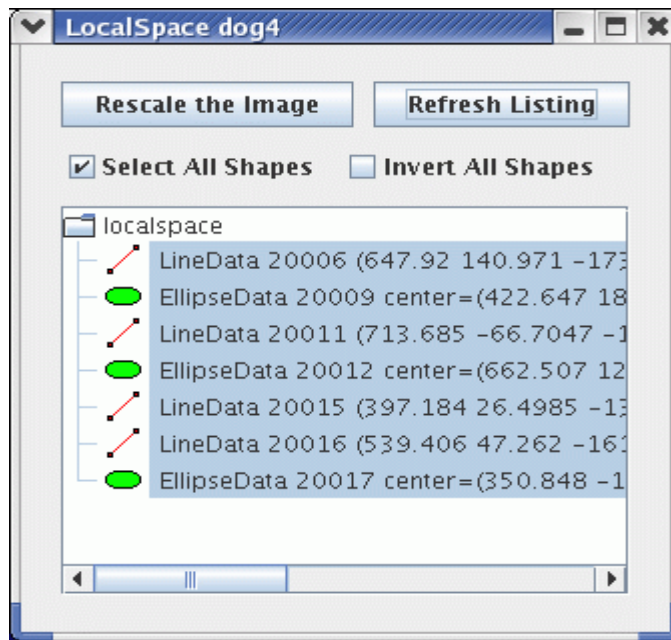
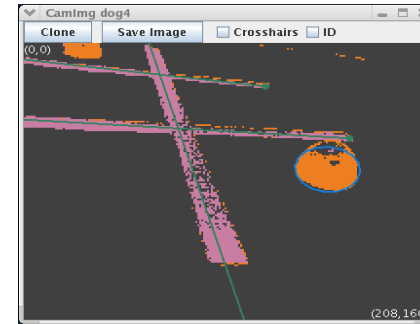
Frame 3



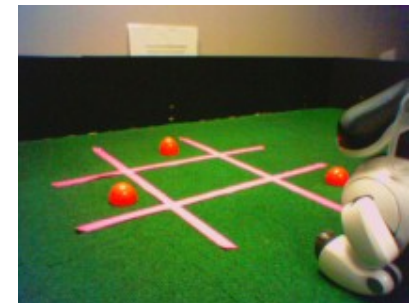
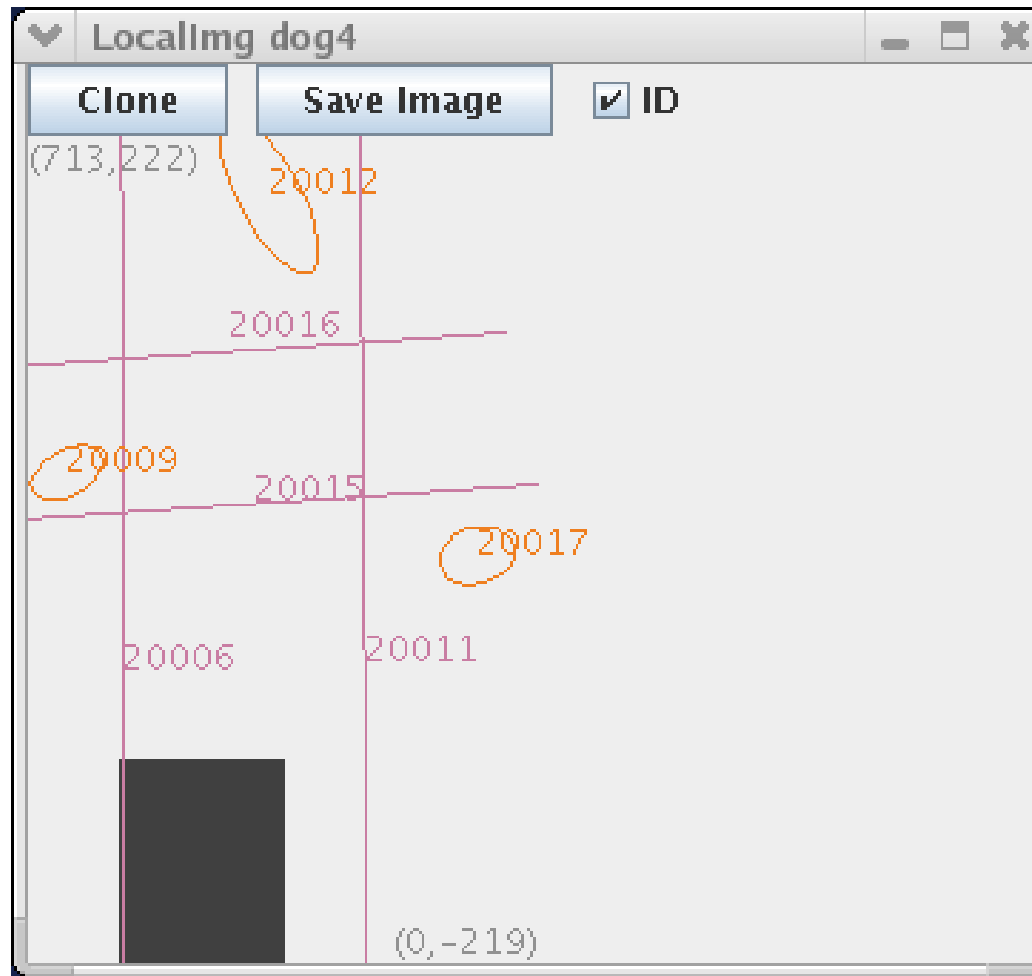
Frame 4



Frame 5



Final Local Map



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.

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.

MapBuilderRequest Parameters

- RequestType
 - cameraMap
 - groundMap
 - localMap
 - worldMap
- Shape parameters:
 - objectColors
 - occluderColors
 - maxDist
 - MinBlobArea
 - aprilTagFamily
 - markerTypes
- Utility functions:
 - clearCamera, clearLocal, clearWorld
 - RawY
- Lookout control:
 - motionSettleTime
 - numSamples
 - sampleInterval
 - pursueShapes
 - searchArea
 - doScan, dTheta
 - manualHeadMotion

World Maps

- worldShS (the world shape space) holds the robot's world map for navigation and localization.
- Examples of pre-specified world maps:
 - Tic-tac-toe board of known size.
 - Maze with known layout.
- Dynamically constructed world maps:
 - Initialize from what's locally visible (localShS).
 - As the robot moves, extend the world map.
 - Transformation matrix maps local to world coordinates.
 - SLAM: Simultaneous Localization and Matching
Not currently implemented; will get there some day.
- Updating the world map when objects move?