# Multi-Level Learning in Hybrid Deliberative/Reactive Mobile Robot Architectural Software Systems

Georgia Institute of Technology

**DARPA MARS Review Meeting - May 2000**

approved for public release: distribution unlimited

## Participants

- Georgia Tech
  - College of Computing
    - Prof. Ron Arkin
    - Prof. Chris Atkeson
    - Prof. Sven Koenig
  - Georgia Tech Research Institute
    - Dr. Tom Collins
- Mobile Intelligence Inc.
    - Dr. Doug MacKenzie

- Students
  - Amin Atrash
  - Bhaskar Dutt
  - Brian Ellenberger
  - Mel Eriksen
  - Max Likachev
  - Brian Lee
  - Sapan Mehta

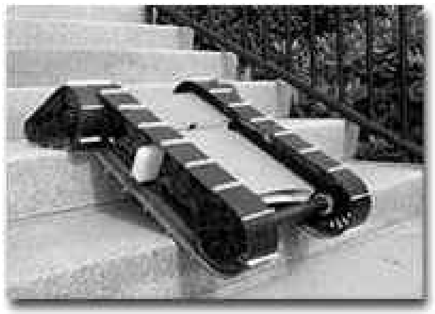
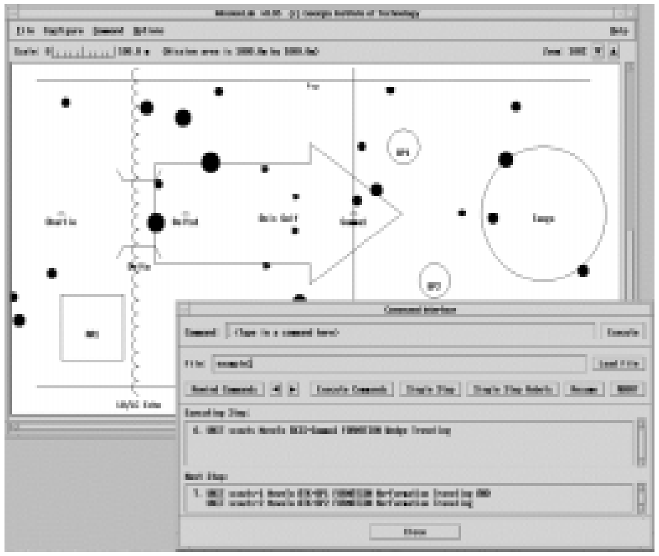
Georgia Tech College of Computing

Georgia Tech Research Institute

# Adaptation and Learning Methods

- Case-based Reasoning for:
  - deliberative guidance (“wizardry”)
  - reactive situation-dependent behavioral configuration
- Reinforcement Learning for:
  - run-time behavior adjustment
  - behavioral assemblage selection
- Probabilistic Behavioral Transitions for:
  - gentler context switching
  - experience-based planning guidance

Available Robots and *MissionLab* Console

# 1. Learning Momentum

- Reactive learning via dynamic gain alteration (parametric adjustment)
- Continuous adaptation based on recent experience
- Situational analyses required
- In a nutshell: If it works, keep doing it a bit harder; if it doesn't, try something different

# Overview

Learning momentum (LM) is a process by which a robot, at runtime, changes values that dictate how it reacts to the environment. Values include weights given to vectors pointing towards the goal, away from obstacles, and in random directions. Also included are the robot's wander persistence and sphere of influence (the maximum distance an obstacle must be from a robot before the obstacle is ignored). A short running history is kept to see if the robot is making progress or is stuck. If the robot determines that it is stuck (the distance it's moving is below a certain threshold), it will take action. For example, it will increase the weight of its random vector and decrease the weight of its goal vector.

## Altered Values

- Move to Goal Vector Weight
- Avoid Obstacles Vector Weight
- Wander Vector Weight
- Wander Persistence
- Obstacle Sphere of Influence

## Goals

- Improved Completion Rate
- Improved Completion Time

# Experiments

Four sets of tests were run. Each set consisted of five series of just over one hundred runs each. The robot differences for each series are summarized in table 1. Each set of tests were run on a different environment. The first two sets were run on environments with a 15% obstacle density, and the last two sets were run on environments with a 20% obstacle density. Results for each run in a series were averaged to represent the overall results for the series. Two specific strategies, ballooning and squeezing, were tested.

| | LM present | Wander Gain | Max Wander Persistence | Navigation Strategy |
|---|---|---|---|---|
| Series 1 | No | .3 | 10 | NA |
| Series 2 | No | .5 | 10 | NA |
| Series 3 | Yes | NA | 15 | Ballooning |
| Series 4 | Yes | NA | 10 | Ballooning |
| Series 5 | Yes | NA | 15 | Squeezing |

Table 1

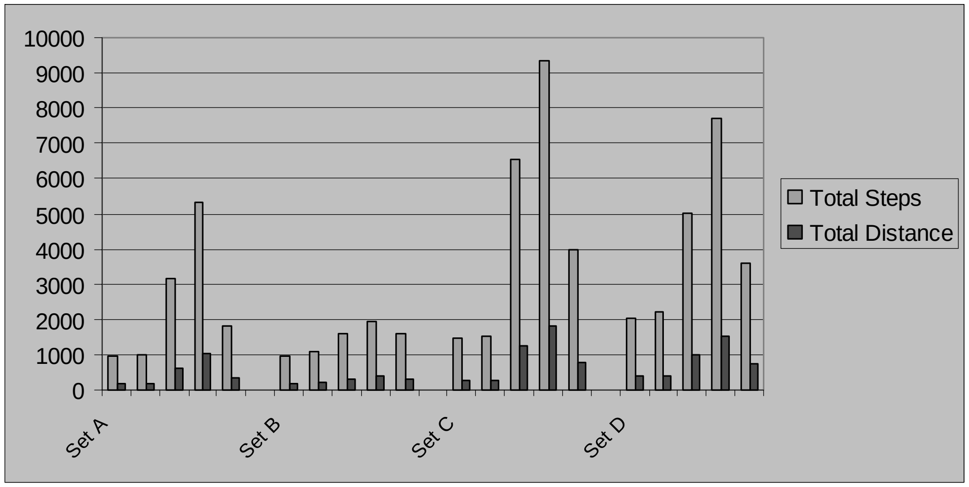
Strategies:

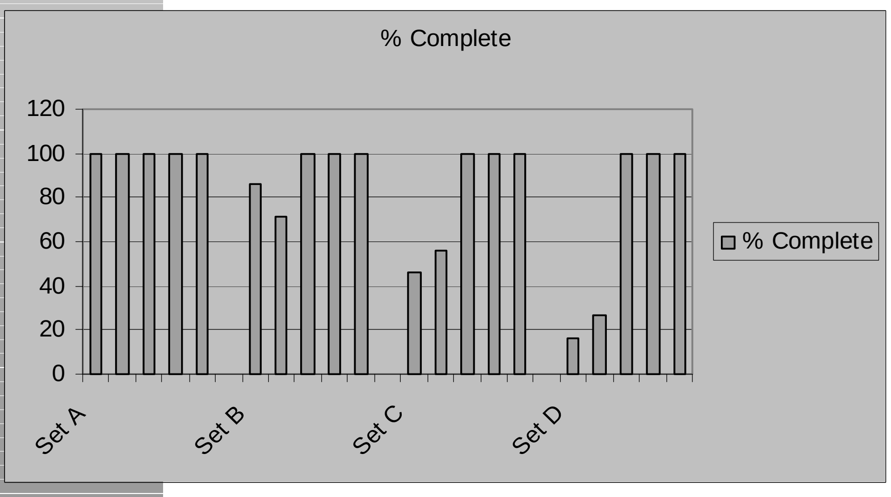
Ballooning - The sphere of influence is increased when the robot comes into contact with obstacles to push the robot around clusters of obstacles and out of box canyon situations.

Squeezing - The sphere of influence is decreased when the robot comes into contact with obstacles so the robot can squeeze between tightly spaced obstacles.

## Results

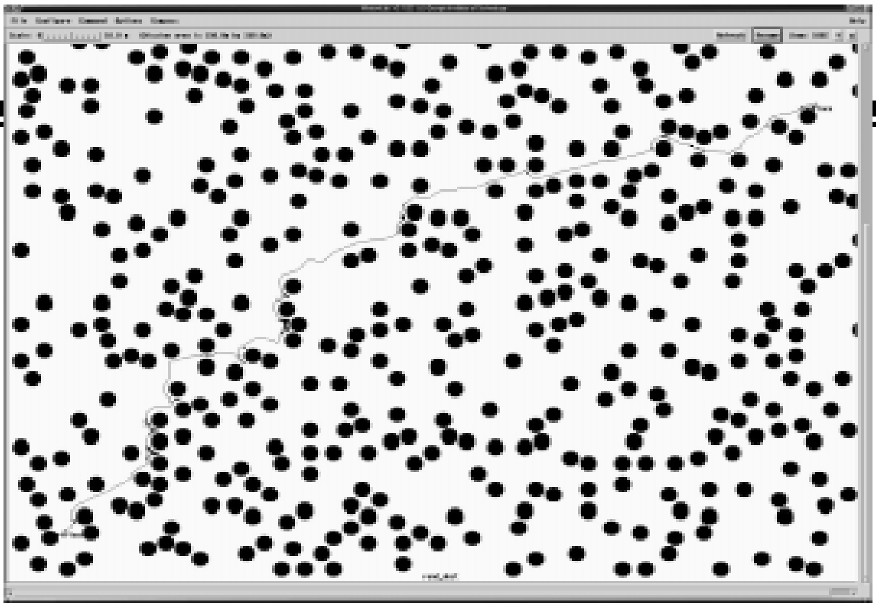
The percentage of trials completed increased to 100% when learning momentum was added to all environments. The success rate of robots without learning momentum decreased as the obstacle density increased. The first two series in each set in the chart above did not utilize learning momentum.

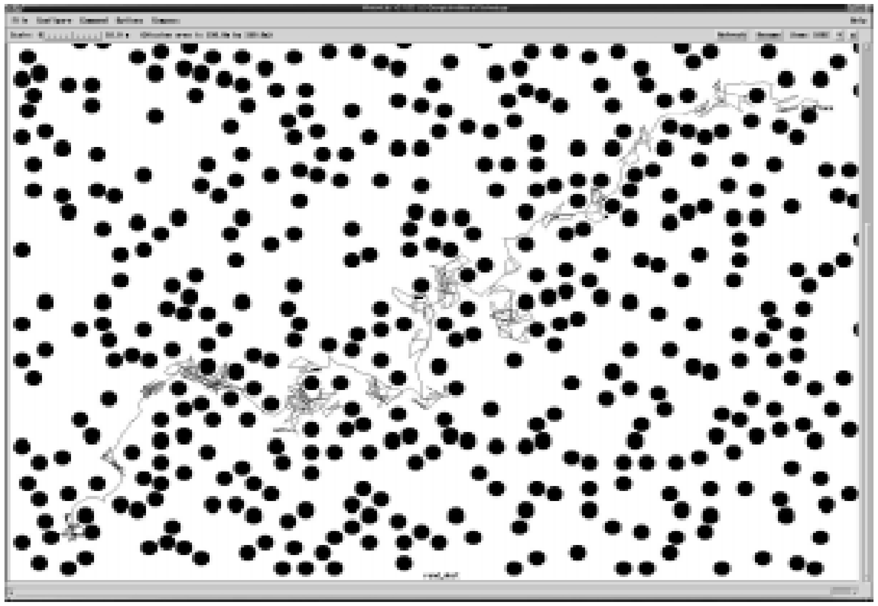




Robots using learning momentum were usually much slower than successful robots not using it. Only results from successful runs were used. The squeezing strategy (series 5) produced better results than the ballooning strategy (series 3 and 4) in the tested environments. In continuous obstacle fields, ballooning around one cluster of objects simply pushes the robot into another cluster.

## Screen Shots

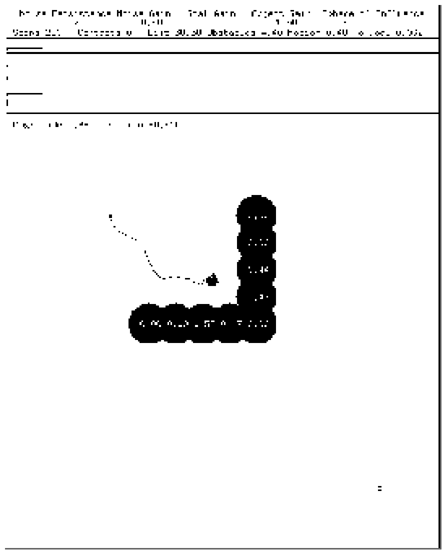


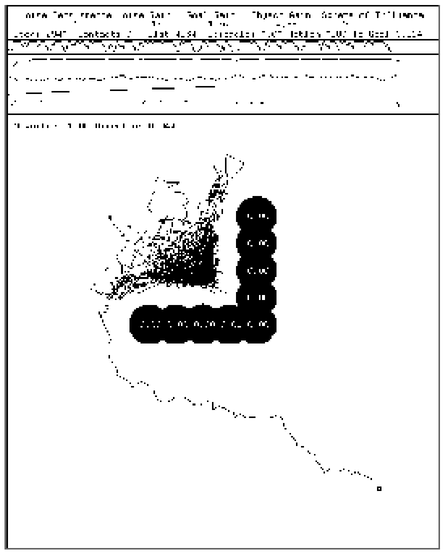


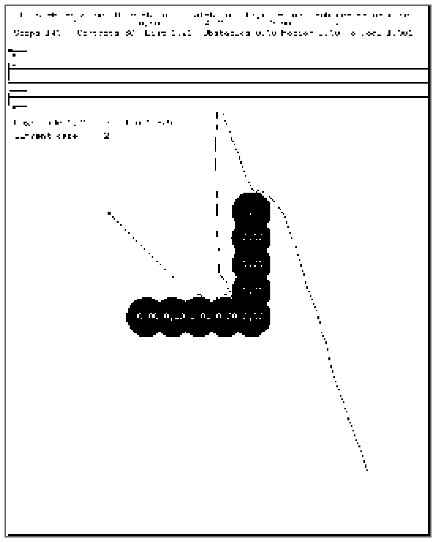
The top figure shows a sample run of a robot using the squeezing strategy. The bottom figure shows another robot traversing the same environment using the ballooning strategy. Even though both approaches are successful, the squeezing strategy is much more direct. It seems that, in environments such as this, learning momentum provides increased success, but there is a cost of time and distance traveled.

# 2. Case-Based Reasoning for Behavioral Selection

- Another form of reactive learning
- Previous systems include: ACBARR and SINS
- Discontinuous behavioral switching



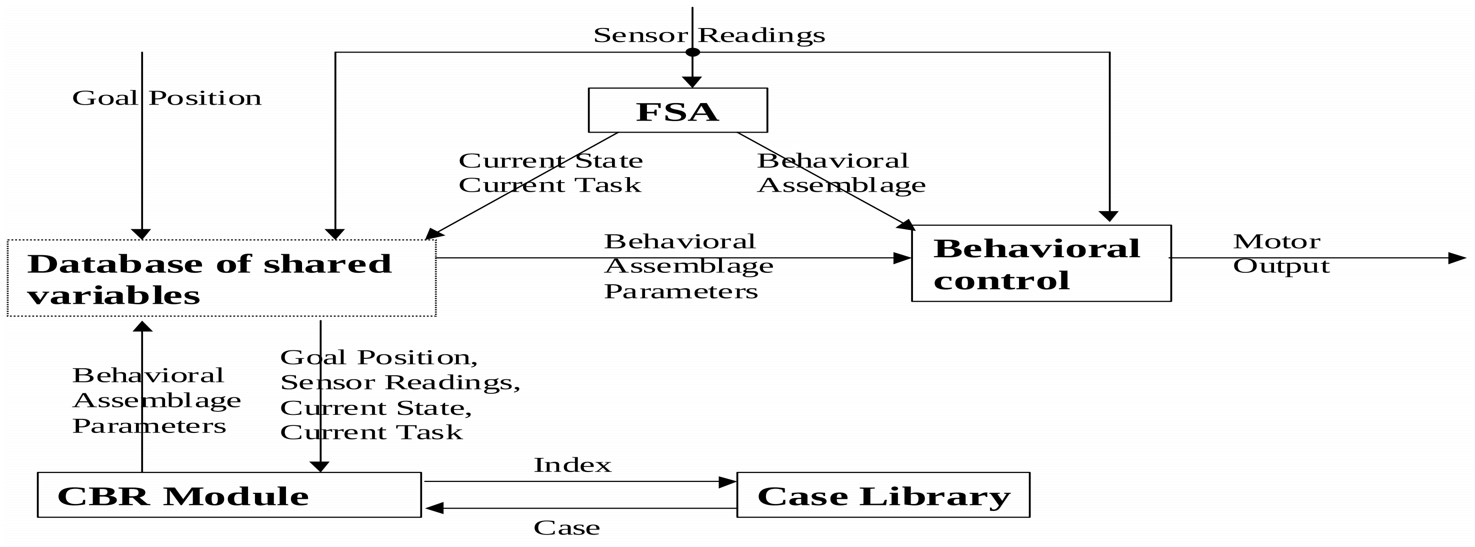




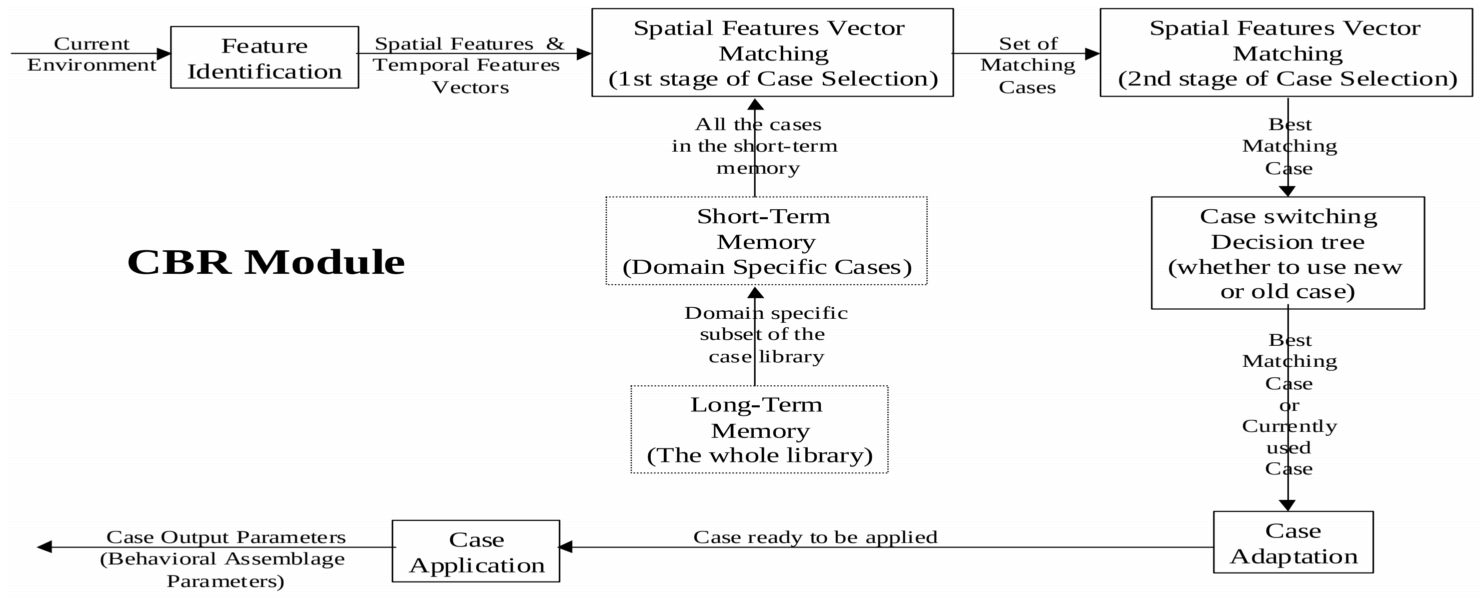
## Overview

- Redesigned the CBR module for the robust feature identification, case selection, and adaptation process:
  - Features are extracted into two vectors:
    - Spatial characteristics that represent the density function discreditized around the robot with configurable resolution
    - Temporal characteristics that represent the short and long term movement of the robot
  - Two-stage selection mechanism:
    - spatial characteristics vector biased matching at a first stage
    - temporal characteristics vector biased matching at a second stage
  - Case switching decision tree to control case switching in order to prevent thrashing and overuse of cases
  - Fine-tuning of the case parameters to decrease the size of the case library
- Support for simple feature and output vectors extensions
- Support for probabilistic feature vectors

# Integration (1)

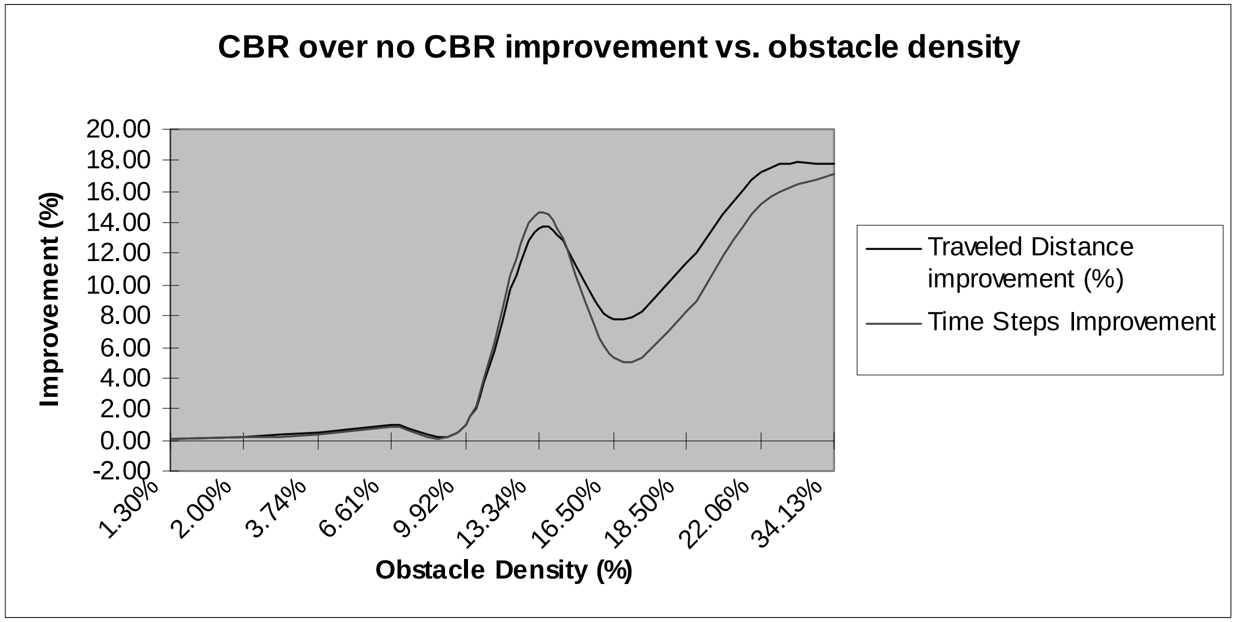


# Integration (2)


Current Environment
Feature Identification
Spatial Features & Temporal Features Vectors
Spatial Features Vector Matching (1st stage of Case Selection)
Set of Matching Cases
Spatial Features Vector Matching (2nd stage of Case Selection)
All the cases in the short-term memory
Best Matching Case
CBR Module
Short-Term Memory (Domain Specific Cases)
Case switching Decision tree (whether to use new or old case)
Domain specific subset of the case library
Long-Term Memory (The whole library)
Best Matching Case or Currently used Case
Case Output Parameters (Behavioral Assemblage Parameters)
Case Application
Case ready to be applied
Case Adaptation

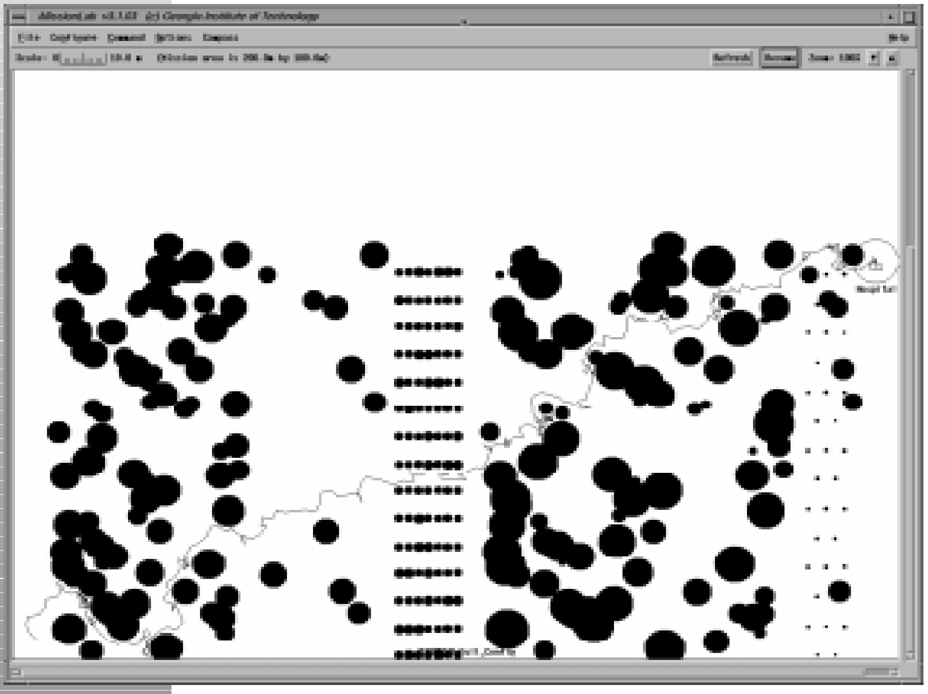
# Experiments and Results

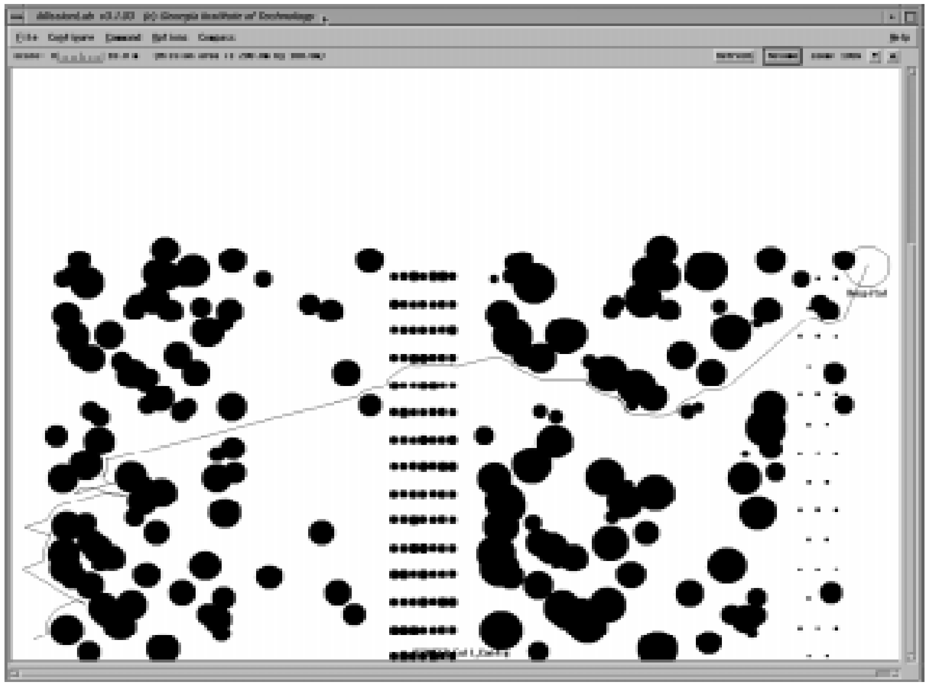
- About 17% decrease on average in the traveling distance and time steps for a MovetoGoal behavior with CBR module over MovetoGoal behavior without CBR module (measured over 40+ runs in environments of different types and varying in densities, with the best set of parameters chosen manually for non-CBR behavior).
- Significant increase in the number of solved environments
- Results of 11 runs with obstacle density varying from 1% to 35 % (the best set of parameters is chosen manually for MoveToGoal without CBR module for each run)



# Screen Shots

- Hospital Approach Scenario. The Environment has five different homogeneous regions in order to exercise the cases fully.
- On the left - MoveToGoal without CBR Module
- On the Right - MoveToGoal with CBR Module





## Future Plans

- Add second level of operation:
  selection and adaptation of the whole new behavioral assemblage
- Automatic learning and adjustment of cases through experience
- Implementation of probabilistic feature identification
- Integration with Q-learning and momentum learning
- Significant statistical results on real robots

## Behavioral Assemblage Selection

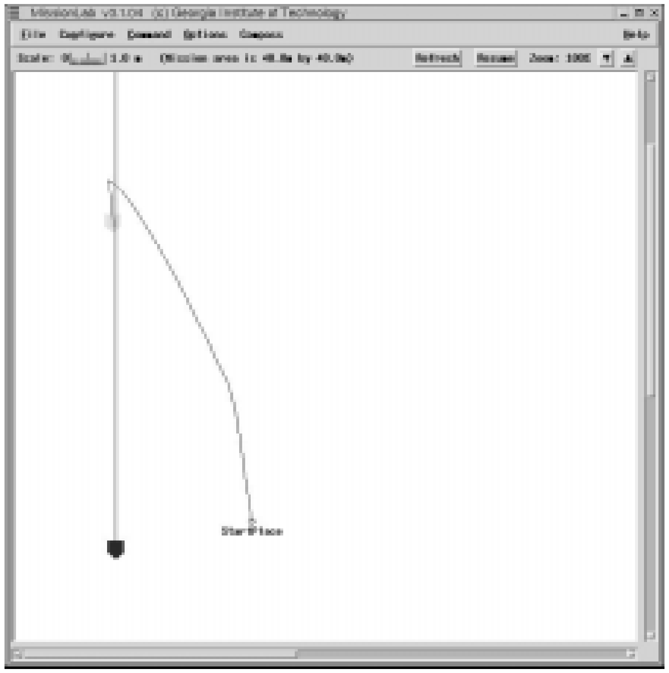
- Reinforcement learning at coarse granularity (behavioral assemblage selection)
- State space tractable
- Operates at level above learning momentum (selection as opposed to adjustment)
- Have added the ability to dynamically choose which behavioral assemblage to execute
- Ability to learn which assemblage to choose using wide variety of reinforcement learning methods: Q-learning, value iteration, policy iteration

## Overview

•Implementation of Assemblage Selection Learning

•Implementation of *Rolling Thunder* Scenario

•Preliminary results of Assemblage Selection Learning using Q-learning


MissionLab v3.1.01 (c) Georgia Institute of Technology
File
Configure
Command
Options
Compass
Help
Scale:
1.0 m
(Mission area is 40.0m by 40.0m)
Refresh
Resume
Zoom: 100%
StartPlace

## Selecting Behavioral Assemblages - Specifics

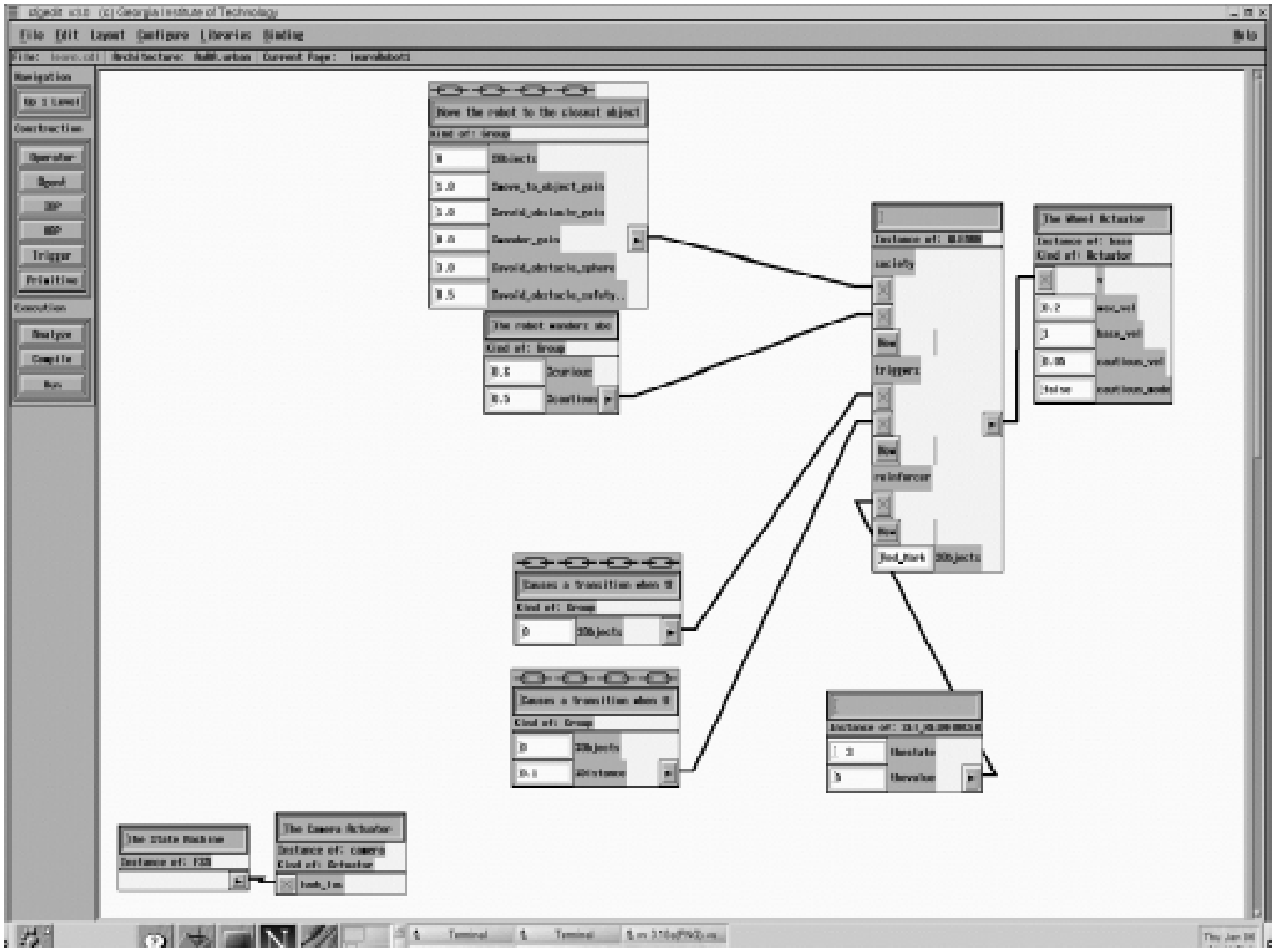
• Replace the FSA with an interface allowing user to specify the environmental and behavioral states

•Agent learns transitions between behavior states

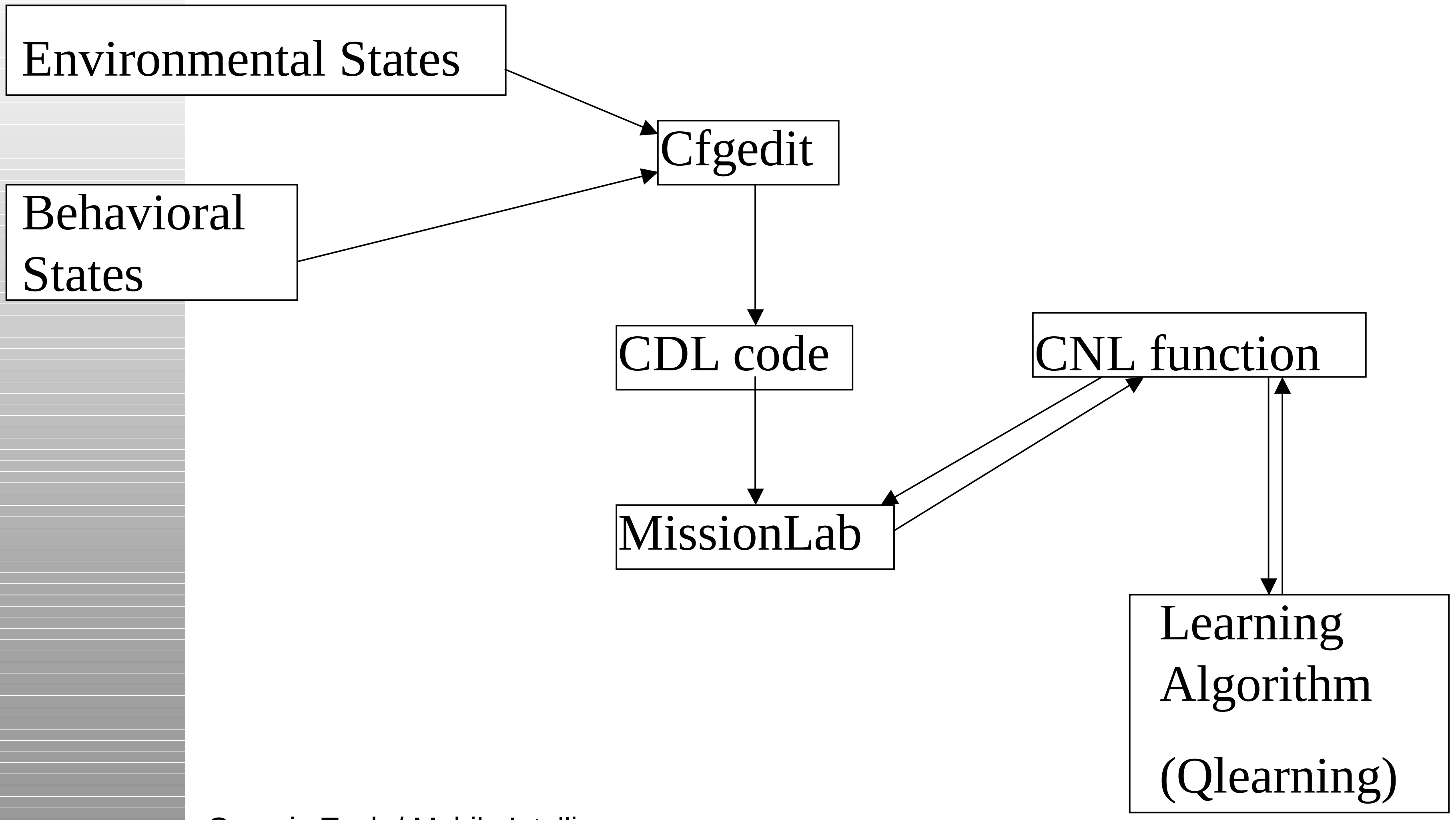
•Learning algorithm is implemented as an abstract module and different learning algorithms can be swapped in and out as desired.

•CNL function interfaces robot executable and learning algorithm

# Integrated System



# Architecture



# RL - Next Steps

•Change implementation of Behavioral Assemblages in *Missionlab* from simply being statically compiled into the CDL code to a more dynamic representation.

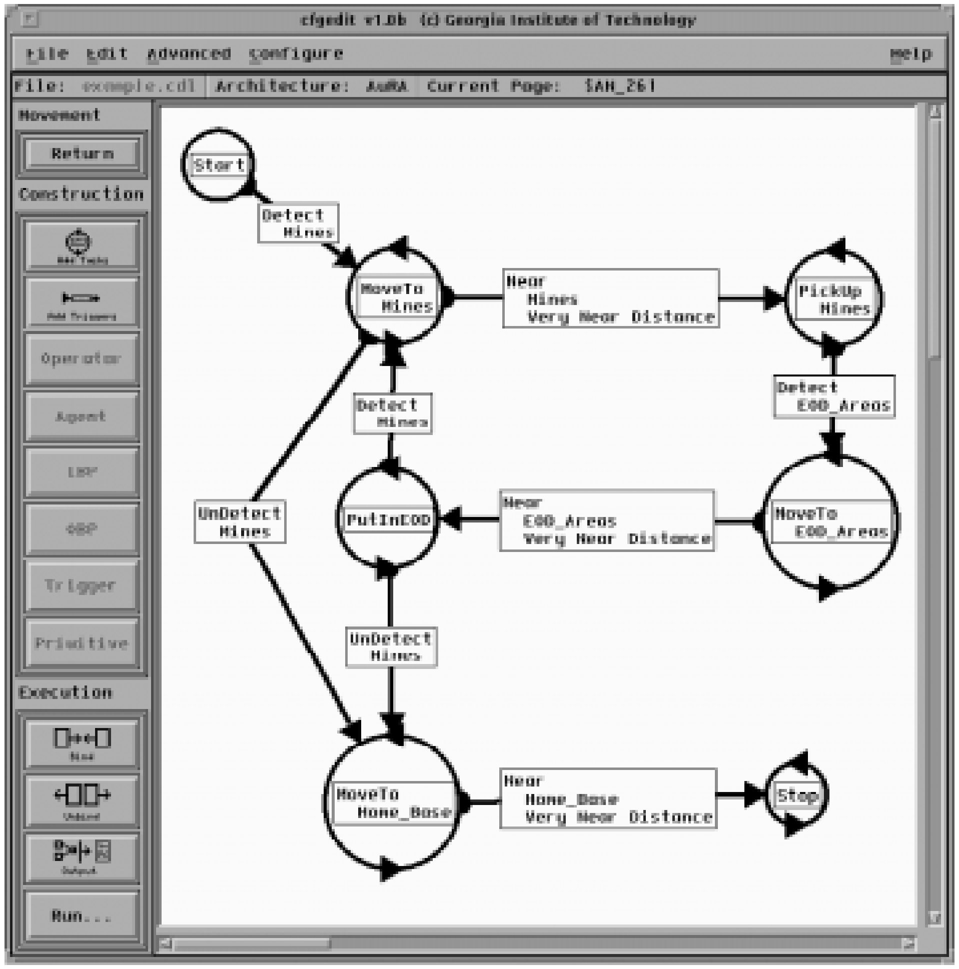
•Create relevant scenarios and test *Missionlab*'s ability to learn good solutions

•Look at new learning algorithms to exploit the advantages of Behavioral Assemblages selection

•Conduct extensive simulation studies then implement on robot platforms

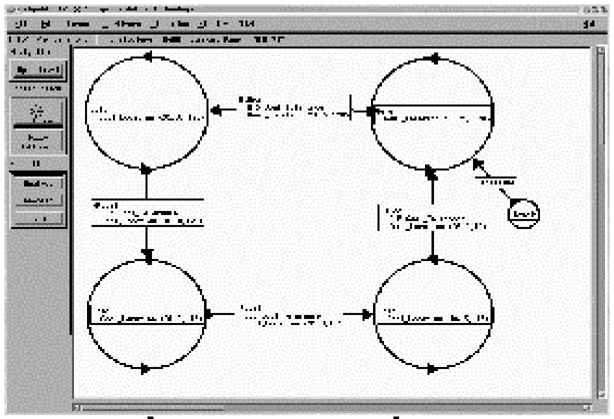
# 4. CBR “Wizardry”

- Experience-driven assistance in mission specification
- At deliberative level above existing plan representation (FSA)
- Provides mission planning support in context



## CBR Wizardry / Usability Improvements

- Current Methods: Using GUI to construct FSA - may be difficult for inexperienced users.

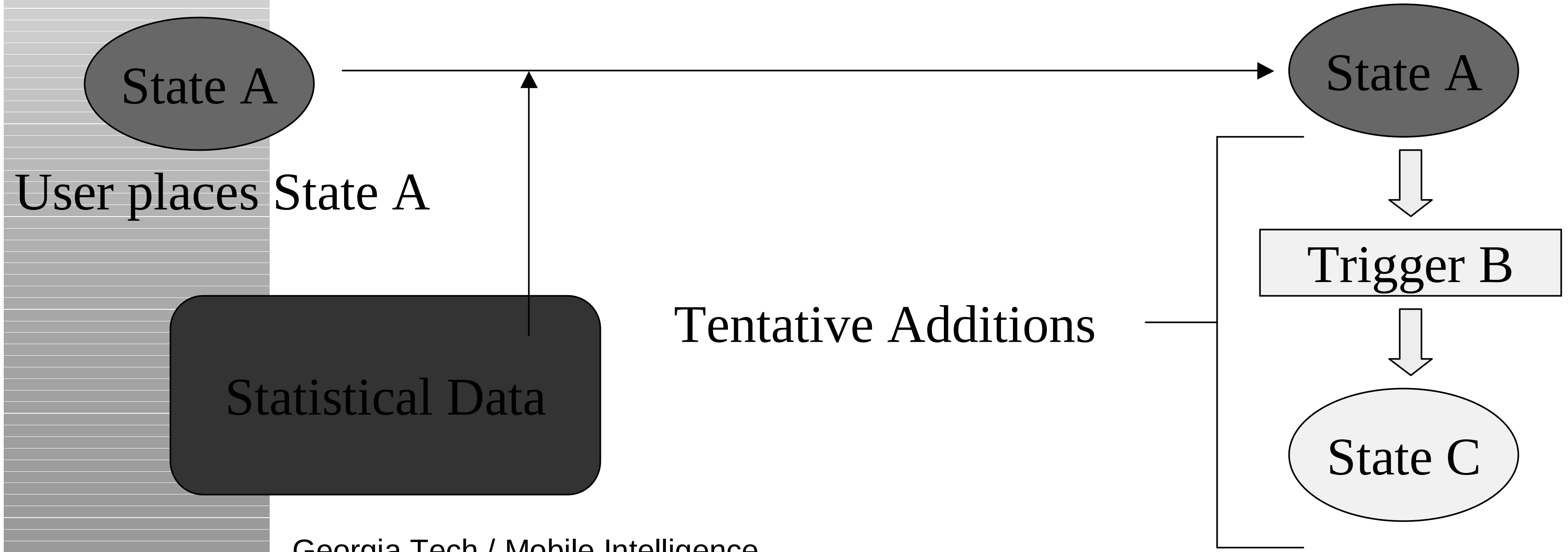


- Goal: Automate plan creation as much as possible while providing unobtrusive support to user.

# Tentative Insertion of FSA Elements:

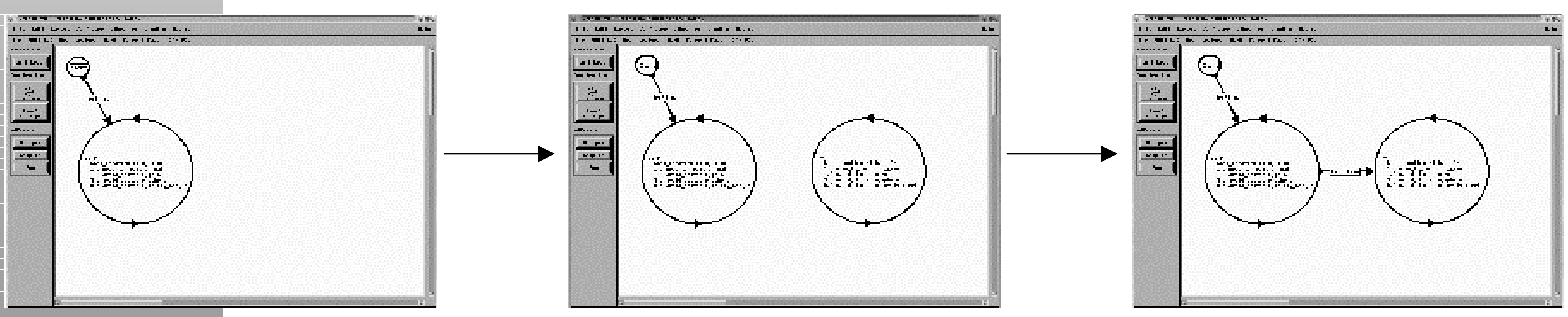
## A user support mechanism currently being worked on

- Some FSA elements very often occur together.
- Statistical data on this can be gathered.
- When user places a state, a trigger and state that follow this state often enough can be tentatively inserted into the FSA.
- Comparable to URL completion features in web browsers.



## Recording Plan Creation Process

- Pinpointing where user has trouble during plan creation is important prerequisite to improving software usability.
- There was no way to record plan creation process in MissionLab.
- Module now created that records user's actions as (s)he creates the plan. This recording can later be played back and points where the user stumbled can thus be identified.



The Creation of a Plan

## Wizardry - Future Work

- Use of plan creation recordings during usability studies to identify stumbling blocks in process.
- Creation of plan templates (frameworks of some commonly used plan types e.g. reconnaissance missions)
- Collection of library of plans which can be placed at different points in “plan creation tree”. This can then be used in a plan creation wizard.

$$\begin{tikzcd}
 & & \text{Plan 1} \arrow[dl] \arrow[d] \arrow[dr] & & \\
 & \text{Plan 2} \arrow[dl] & \text{Plan 3} \arrow[d] & \text{Plan 4} \arrow[d] \arrow[dr] & \\
\text{Plan 5} & & \text{Plan 6} & \text{Plan 7} & \text{Plan 8}
\end{tikzcd}$$

Plan Creation Tree

## 5. Probabilistic Planning and Execution

- "Softer, kinder" method for matching situations and their perceptual triggers
- Expectations generated based on situational probabilities regarding behavioral performance (e.g., obstacle densities and traversability), using them at planning stages for behavioral selection
- Markov Decision Process and other Bayesian methods to be investigated

## Overview

### Purpose

Integration of probabilistic planning into a behavior-based system

### Theory

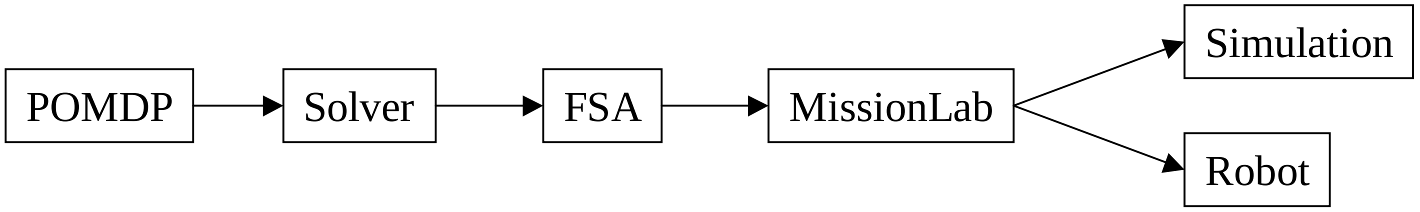
Probabilistic planning can be used to address issues such as sensor uncertainty, actuator uncertainty, and environmental uncertainty.

POMDPs (Partially Observable Markov Decision Processes) can be used to plan based on models of the environment. These models consist of states, actions, costs, transition probabilities, and observation probabilities.

By mapping the policy graph to a finite state automaton, the resulting plan can be used in behavior-based systems.

Different costs and probabilities result in different plans. Our working hypothesis is that humans are bad in determining optimal plans, which is why planning should be automated.

# Integration (1)

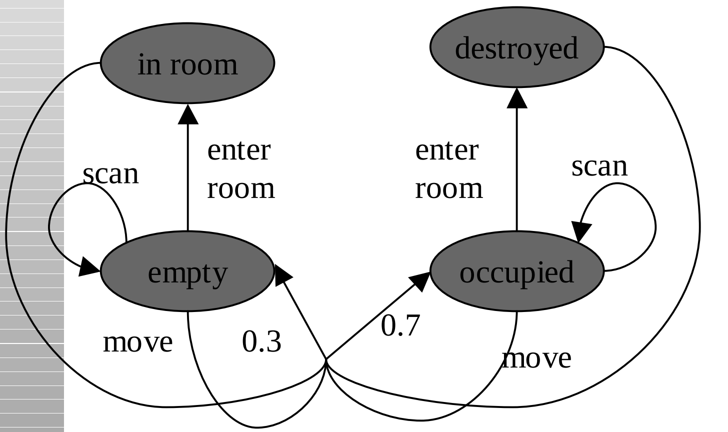


1. The POMDP is specified.
2. The POMDP is solved resulting in an FSA (finite state automaton).
3. The FSA is converted into .cdl and loaded into MissionLab.
4. The FSA is compiled and executed in simulation or on a physical robot.

# Integration (2)

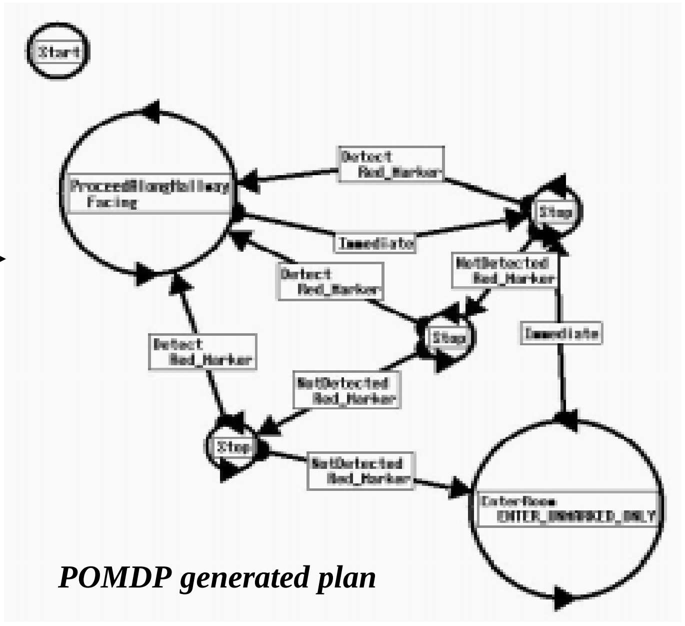
Policy Graph

FSA



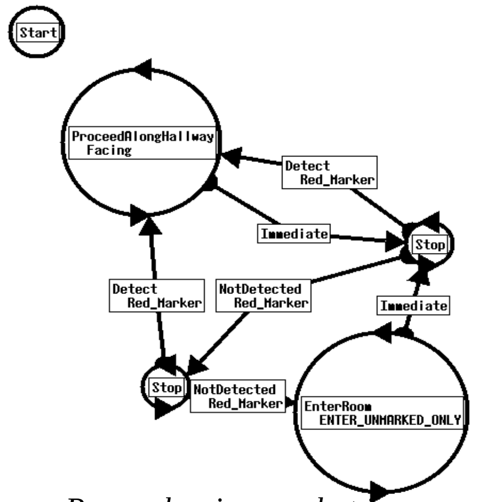
*Sensor model for scan:*
*P(detect-occupied|occupied) = 0.8*
*P(detect-empty|empty = 1.0*

***POMDP Model***

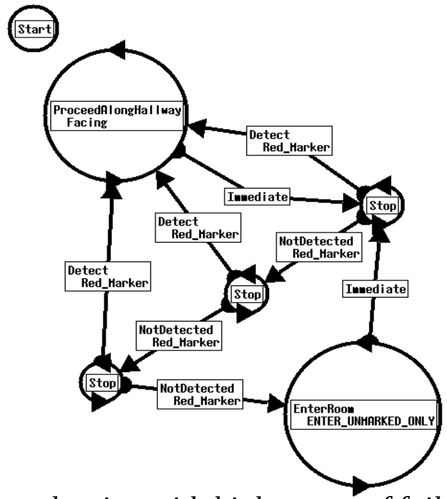


***POMDP generated plan***

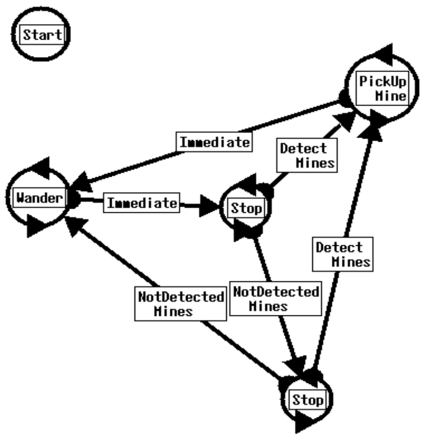
# Experiments and Results



*Room clearing – robot is traversing a hallway to determine which rooms are safe to enter. (POMDP model on previous slide)*



*Room clearing with higher cost of failure*



*Mine removal*

These simple examples demonstrate the ability of the POMDP solver to generate different plans by weighing costs against probabilities and the ability of the compiler to integrate the resulting policy graphs into MissionLab.

Currently, the compiler generates “modules” which the user integrates into complete plans.

## Issues and Future Plans

**Issues**

It is currently difficult to solve large POMDPs. This restricts the application domain to small problems.

The mapping of states and transitions of the policy graph to triggers and behaviors of the finite state automaton may give rise to semantic problems. This mapping must be taken into account during modeling based on the scenario, the details of the simulation used, and the capabilities of the robot.

**Future Plans**

Gather more numerical data based on simulation runs.

Test on real robots. Sensor models must be developed (based on sampling) for the sensors being used. A microphone, for example, maybe used for the room searching scenario.

Develop a simpler interface for modeling the POMDP.

## Role of Mobile Intelligence Inc.

- Develop a conceptual plan for integrating learning algorithms into *MissionLab*
- Guide students performing integration
- Assist in designing usability studies to evaluate the integrated system
- Guide performance and evaluation of usability studies
- Identify key technologies in *MissionLab* which could be commercialized
- Support technology transfer to a designated company for commercialization

# Schedule

| Milestone | | GFY01 | | | | GFY02 | | | | GFY03 | | | | GFY04 | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Jul | Oct | Jan | Apr | Jul | Oct | Jan | Apr | Jul | Oct | Jan | Apr | Jul | Oct | Jan | Apr |
| Demonstration of all learning algorithms in simulation | | | ♦ | | | | | | | | | | | | | |
| Initial integration within MissionLab on lab robots | | | | | ♦ | | | | | | | | | | | |
| Learning algorithms demonstrated in relevant scenarios | | | | | | | | | ♦ | | | | | | | |
| MissionLab demonstration on government platforms | | | | | | | | | | | | ♦ | | | | |
| Enhanced learning algorithms on government platforms | | | | | | | | | | | | | | ♦ | | |
| Final demonstrations of relevant scenarios with govt. platforms | | | | | | | | | | | | | | | | ♦ |