# Improving Adaptability of Multi-Mode Systems via Program Steering

Lee Lin
Michael D. Ernst

MIT CSAIL

# Multi-Mode Systems

- A multi-mode system's behavior depends on its environment and internal state
- Examples of multi-mode systems:
  - Web server: polling / interrupt
  - Cell phone: AMPS / TDMA / CDMA
  - Router congestion control: normal / intentional drops
  - Graphics program: high detail / low detail

# Controllers

- Controller chooses which mode to use
- Examples of factors that determine modes:
  - Web server: heavy traffic vs. light traffic
  - Cell phone: rural area vs. urban area; interference
  - Router congestion control: preconfigured policy files
  - Graphics program: frame rate constraints

# Controller Example

```
while (true) {
    if ( checkForCarpet() )
        indoorNavigation();
    else if ( checkForPavement() )
        outdoorNavigation();
    else
        cautiousNavigation();
}
```

- Do the predicates handle all situations well?
- Is any more information available?
- Does the controller ever fail?

# Improving Built-in Controllers

- Built-in controllers do well in expected situations
- Goal: Create a controller that adapts well to unanticipated situations
  - Utilize redundant sensors during hardware failures
  - Sense environmental changes
  - Avoid default modes if other modes are more appropriate
  - Continue operation if controller fails

# Why Make Systems Adaptive?

- Testing all situations is impossible
- Programmers make mistakes
  - Bad intuition
  - Bugs
- The real world is unpredictable
  - Hardware failures
  - External environmental changes
- Human maintenance is costly
  - Reduce need for user intervention
  - Issue fewer software patches

# Overview

- Program Steering Technique
- Mode Selection Example
- Program Steering Implementation
- Experimental Results
- Conclusions

# Program Steering Goals

- Create more adaptive systems without creating new modes
- Allow systems to extrapolate knowledge from successful training examples
- Choose appropriate modes in unexpected situations

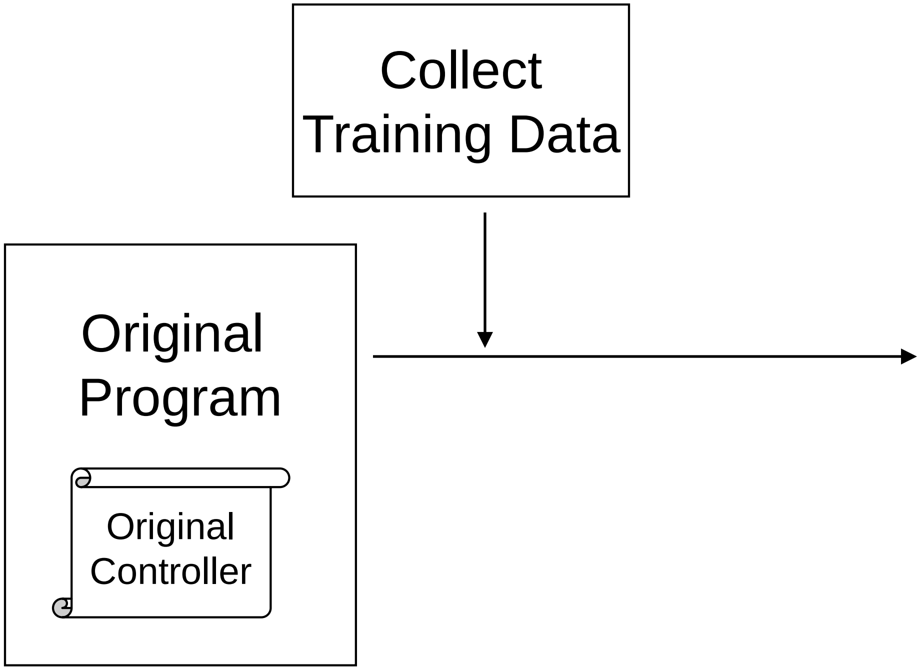
# Program Steering Overview

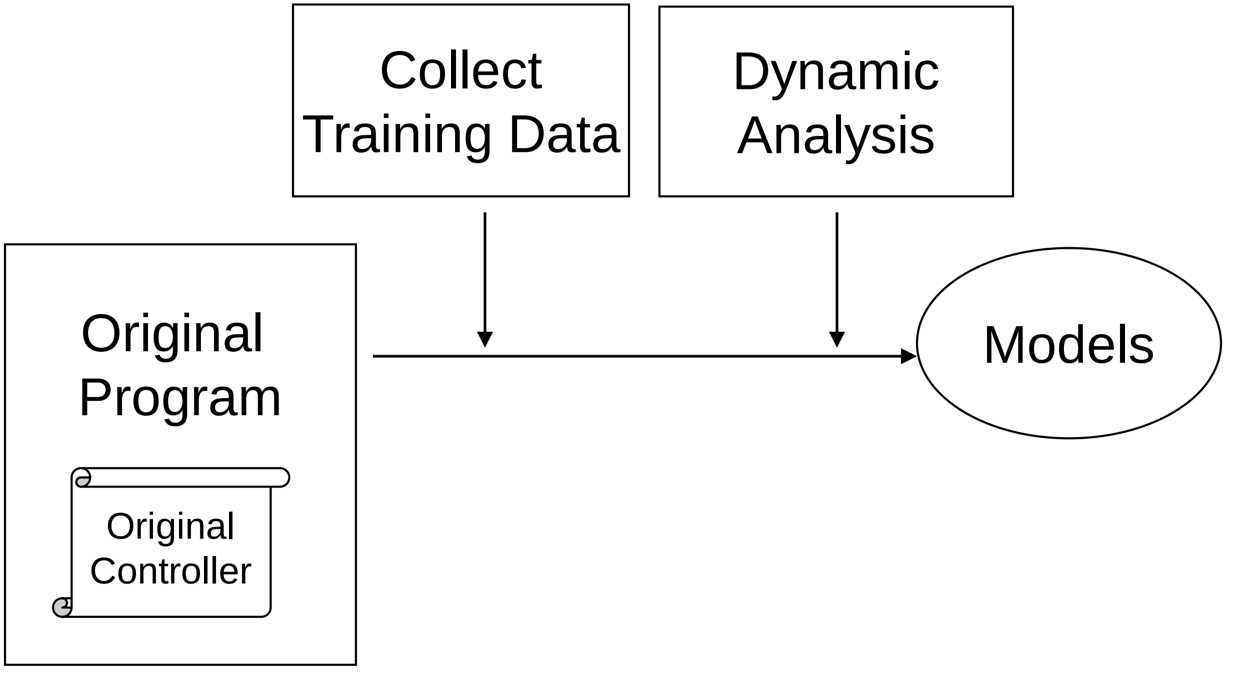
1. Select representative training runs

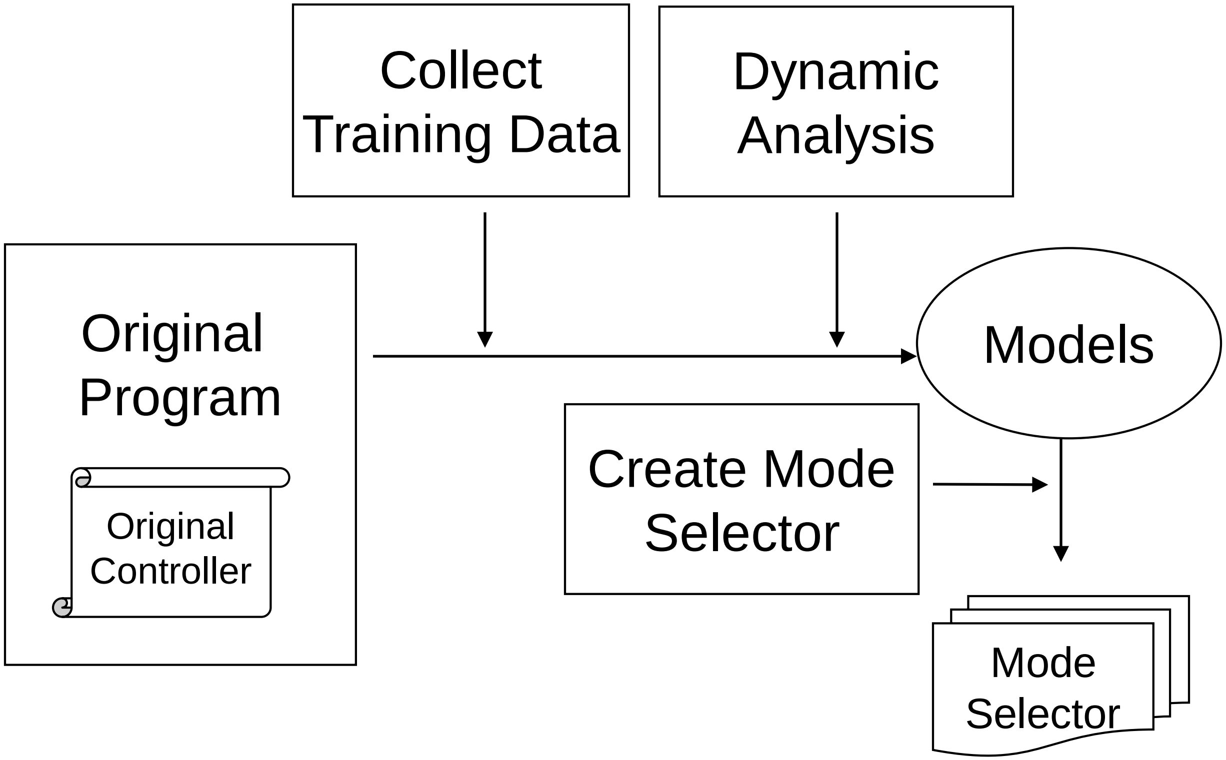
2. Create models describing each mode using dynamic program analysis

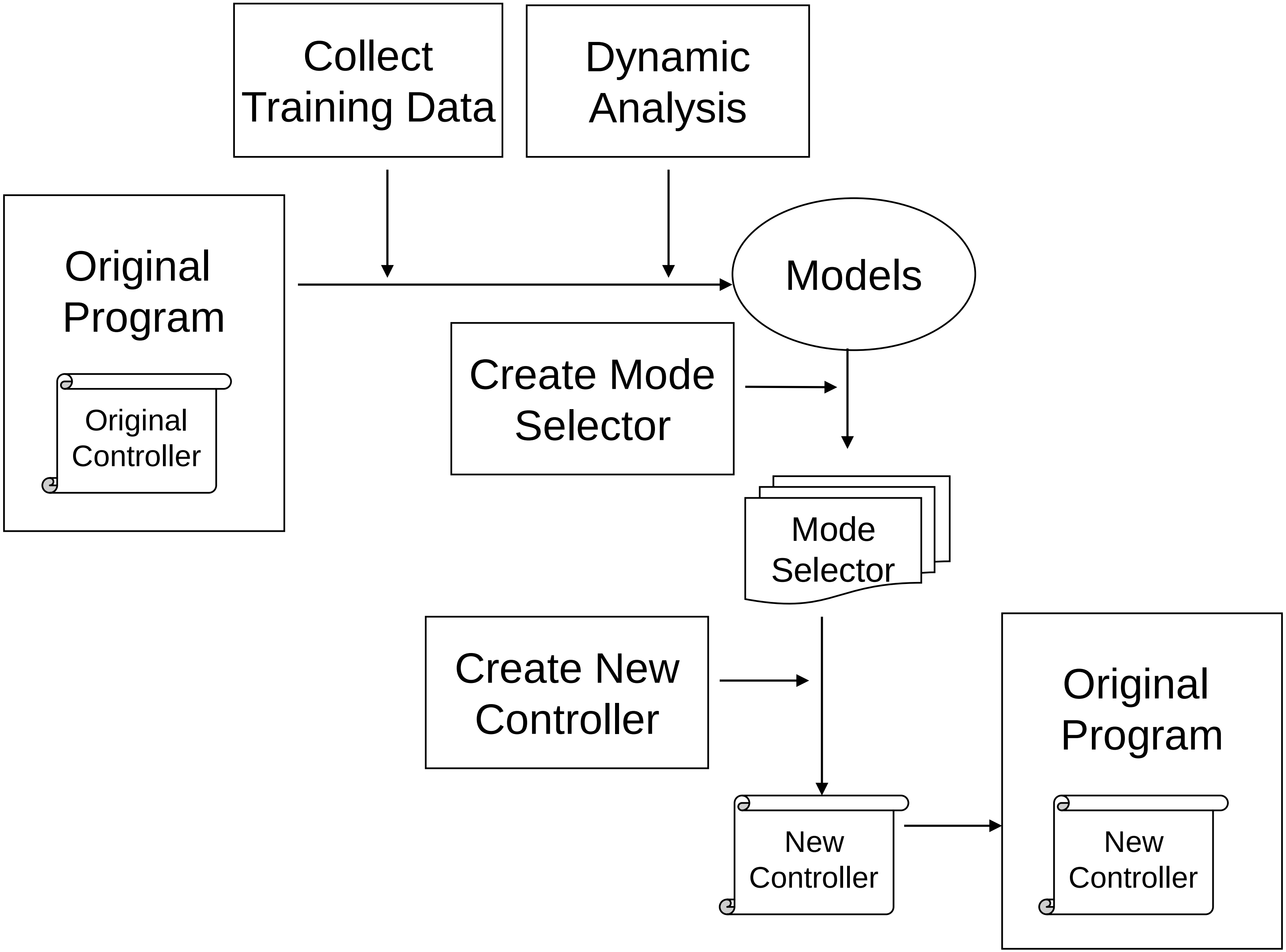
3. Create a mode selector using the models

4. Augment the original program to utilize the new mode selector


Collect
Training Data
Original
Program
Original
Controller


Collect
Training Data
Dynamic
Analysis
Original
Program
Original
Controller
Models


Collect Training Data
Dynamic Analysis
Original Program
Original Controller
Models
Create Mode Selector
Mode Selector


Collect Training Data
Dynamic Analysis
Original Program
Original Controller
Models
Create Mode Selector
Mode Selector
Create New Controller
New Controller
Original Program
New Controller

# Overview

- Program Steering Technique
- Mode Selection Example
- Program Steering Implementation
- Experimental Results
- Conclusions

# Laptop Display Controller

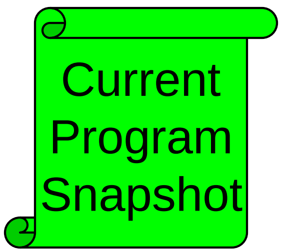
- Three modes
  - Normal Mode
  - Power Saver Mode
  - Sleep Mode
- Available Data:
  - Inputs:  battery life and DC power availability
  - Outputs: brightness

# Properties Observed from Training Runs

| Standard Mode | Power Saver Mode | Sleep Mode |
|---|---|---|
| Brightness >= 0 | Brightness >= 0 | Brightness == 0 |
| Brightness <= 10 | Brightness <= 4 | |
| Battery > 0.15 | Battery > 0.00 | Battery > 0.00 |
| Battery <= 1.00 | Battery <= 0.15 | Battery <= 1.00 |
| | DCPower == false | |

# Mode Selection Problem

What mode is most appropriate?



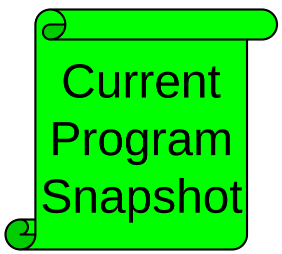

Brightness == 8

Battery == 0.10

DCPower == true

# Mode Selection

Mode selection policy: Choose the mode with the highest percentage of matching properties.



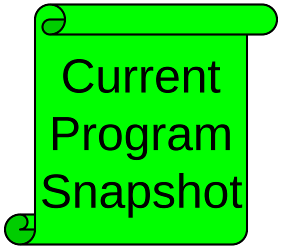

Brightness == 8

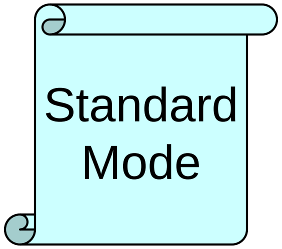
Battery == 0.10

DCPower == true

# Mode Selection

Mode selection policy: Choose the mode with the highest percentage of matching properties.




| | |
|---|---|
| Brightness == 8 | BRT >= 0 |
| | BRT <= 10 |
| Battery == 0.10 | BAT > 0.15 |
| | BAT <= 1.00 |
| DCPower == true | |

Score: 75%

# Mode Selection

Mode selection policy: Choose the mode with the highest percentage of matching properties.

| Current Program Snapshot | Standard Mode | Power Saver Mode |
|---|---|---|
| Brightness == 8 | BRT >= 0 | BRT >= 0 |
| | BRT <= 10 | BRT <= 4 |
| Battery == 0.10 | BAT > 0.15 | BAT > 0.00 |
| | BAT <= 1.00 | BAT <= 0.15 |
| DCPower == true | | DC == false |
| | Score: 75% | Score: 60% |

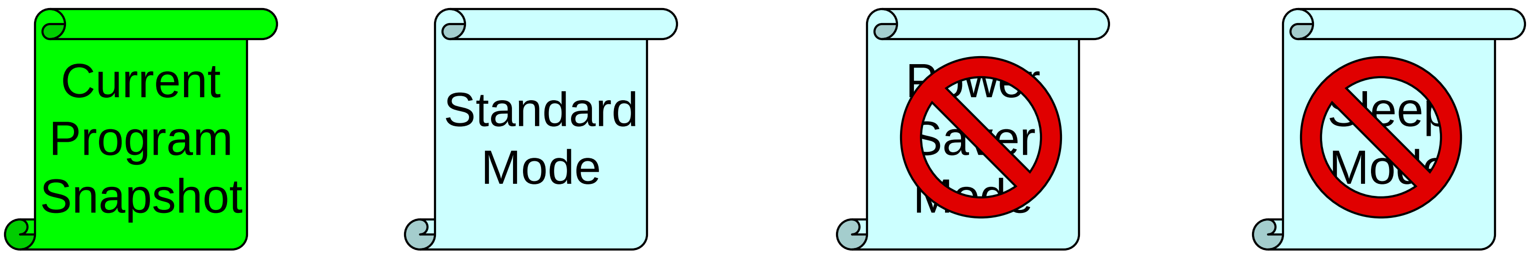
# Mode Selection

Mode selection policy: Choose the mode with the highest percentage of matching properties.

| Current Program Snapshot | Standard Mode | Power Saver Mode | Sleep Mode |
|---|---|---|---|
| Brightness == 8 | BRT >= 0 | BRT >= 0 | BRT == 0 |
| | BRT <= 10 | BRT <= 4 | |
| Battery == 0.10 | BAT > 0.15 | BAT > 0.00 | BAT > 0.00 |
| | BAT <= 1.00 | BAT <= 0.15 | BAT <= 1.00 |
| DCPower == true | | DC == false | |
| | Score: 75% | Score: 60% | Score: 66% |

# Mode Selection

Mode selection policy: Choose the mode with the highest percentage of matching properties.




| | | | |
|---|---|---|---|
| Brightness == 8 | BRT >= 0 | BRT >= 0 | BRT == 0 |
| | BRT <= 10 | BRT <= 4 | |
| Battery == 0.10 | BAT > 0.15 | BAT > 0.00 | BAT > 0.00 |
| | BAT <= 1.00 | BAT <= 0.15 | BAT <= 1.00 |
| DCPower == true | | DC == false | |
| | Score: 75% | Score: 60% | Score: 66% |

# Second Example

Mode selection policy: Choose the mode with the highest percentage of matching properties.

Current Program Snapshot

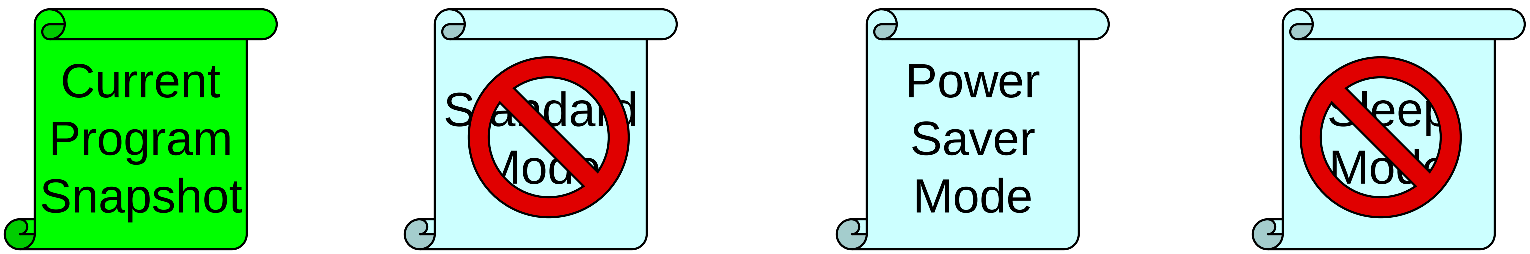
Brightness == 8

Battery == 0.10

DCPower == false

# Second Example

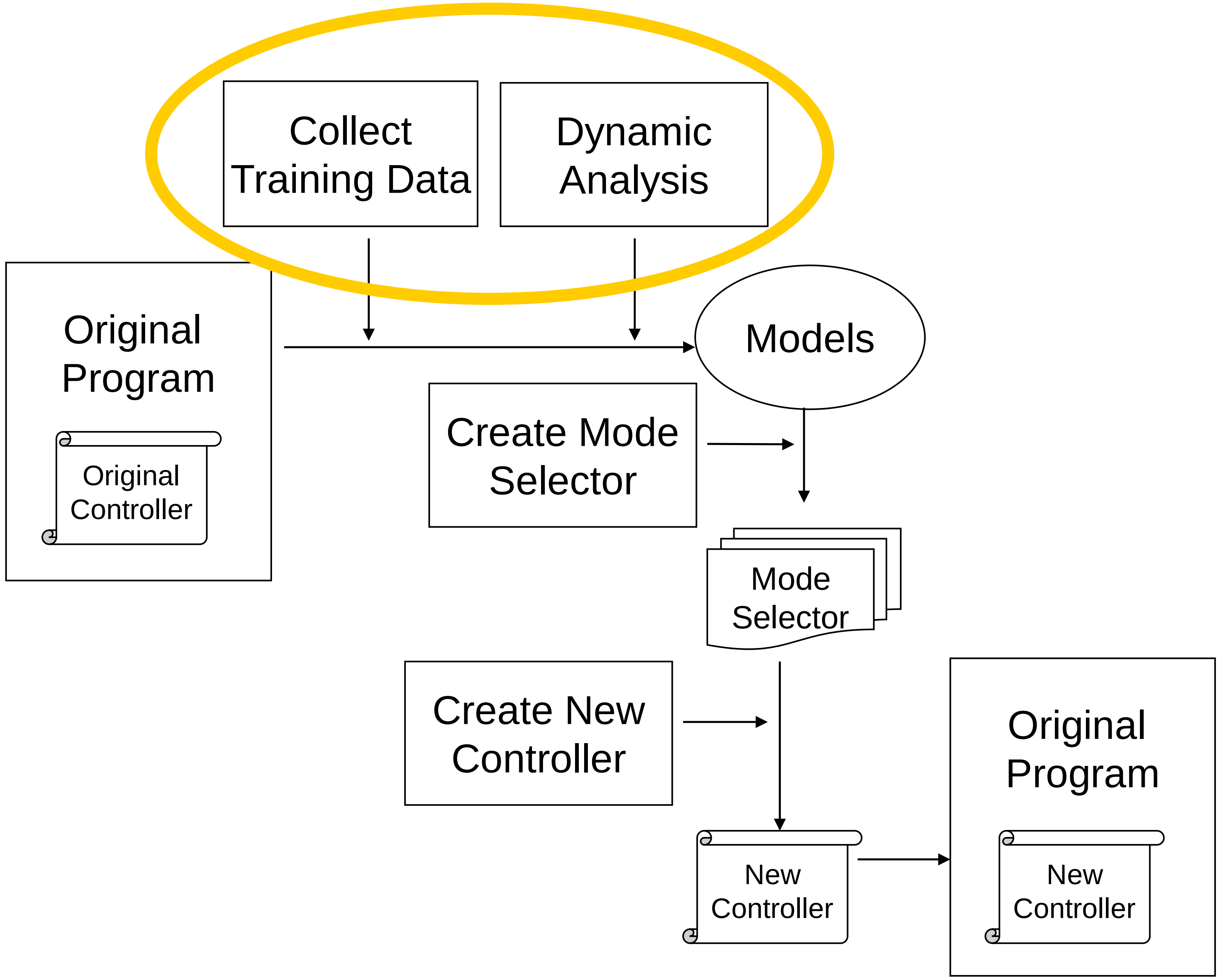
Mode selection policy: Choose the mode with the highest percentage of matching properties.




| | | | |
|---|---|---|---|
| Brightness == 8 | BRT >= 0 | BRT >= 0 | BRT == 0 |
| | BRT <= 10 | BRT <= 4 | |
| Battery == 0.10 | BAT > 0.15 | BAT > 0.00 | BAT > 0.00 |
| | BAT <= 1.00 | BAT <= 0.15 | BAT <= 1.0 |
| DCPower == false | | DC == false | |
| | Score: 75% | Score: 80% | Score: 66% |

# Overview

- Program Steering Technique
- Mode Selection Example
- Program Steering Implementation
- Experimental Results
- Conclusions



Collect Training Data
Dynamic Analysis
Original Program
Original Controller
Models
Create Mode Selector
Mode Selector
Create New Controller
New Controller
Original Program
New Controller

# Training

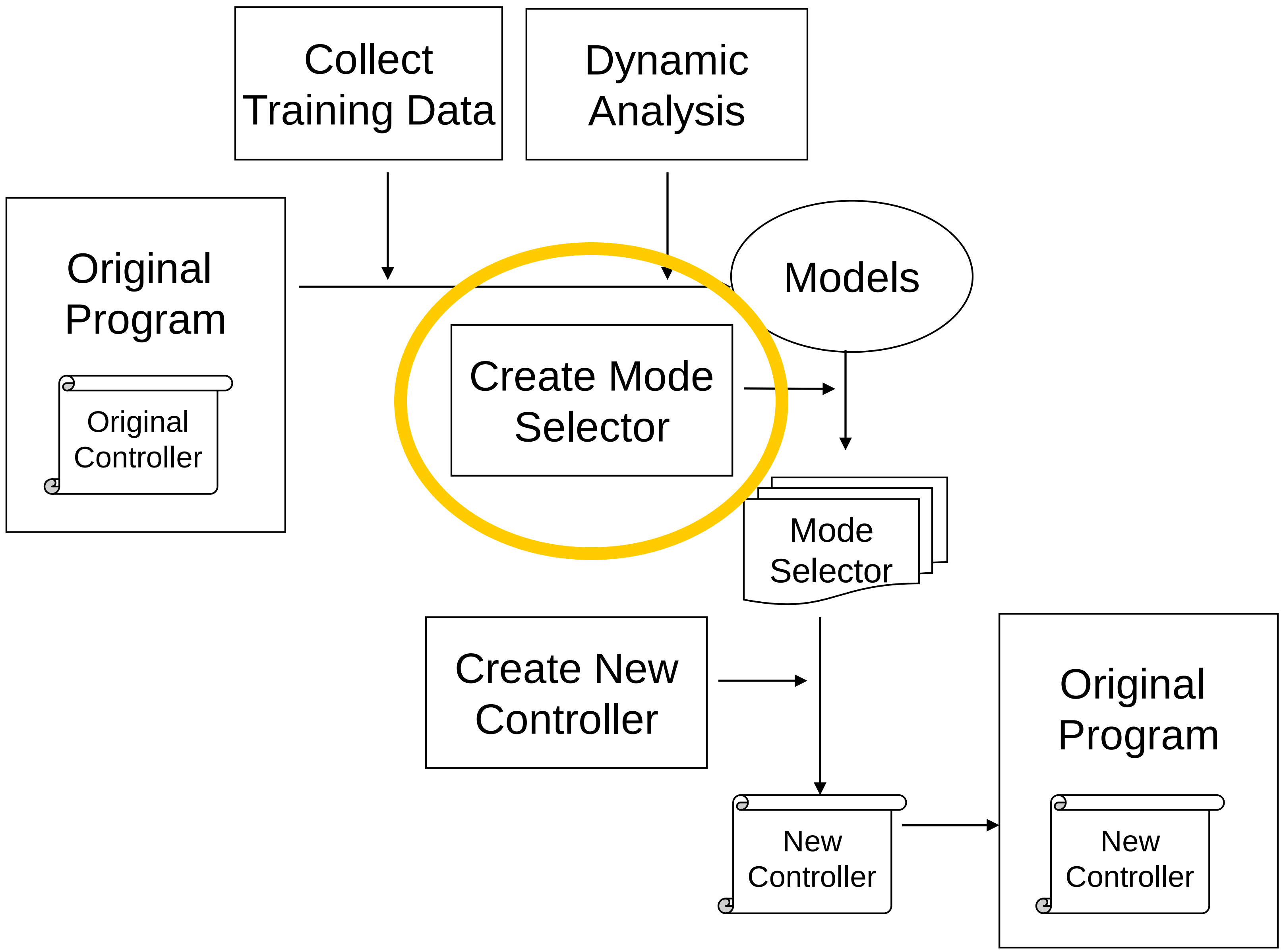
- Train on successful runs
  - Passing test cases
  - High performing trials

- Amount of training data:
  - Depends on modeling technique
  - Cover all modes

# Dynamic Analysis

- Create one set of properties per mode
- Daikon Tool
  - Supply program and execute training runs
  - Infers properties involving inputs and outputs
  - Properties were true for every training run
    - this.next.prev == this
    - currDestination is an element of visitQueue[]
    - n < mArray.length

http://pag.csail.mit.edu/daikon/



Collect Training Data
Dynamic Analysis
Original Program
Original Controller
Models
Create Mode Selector
Mode Selector
Create New Controller
New Controller
Original Program
New Controller

# Mode Selection Policy

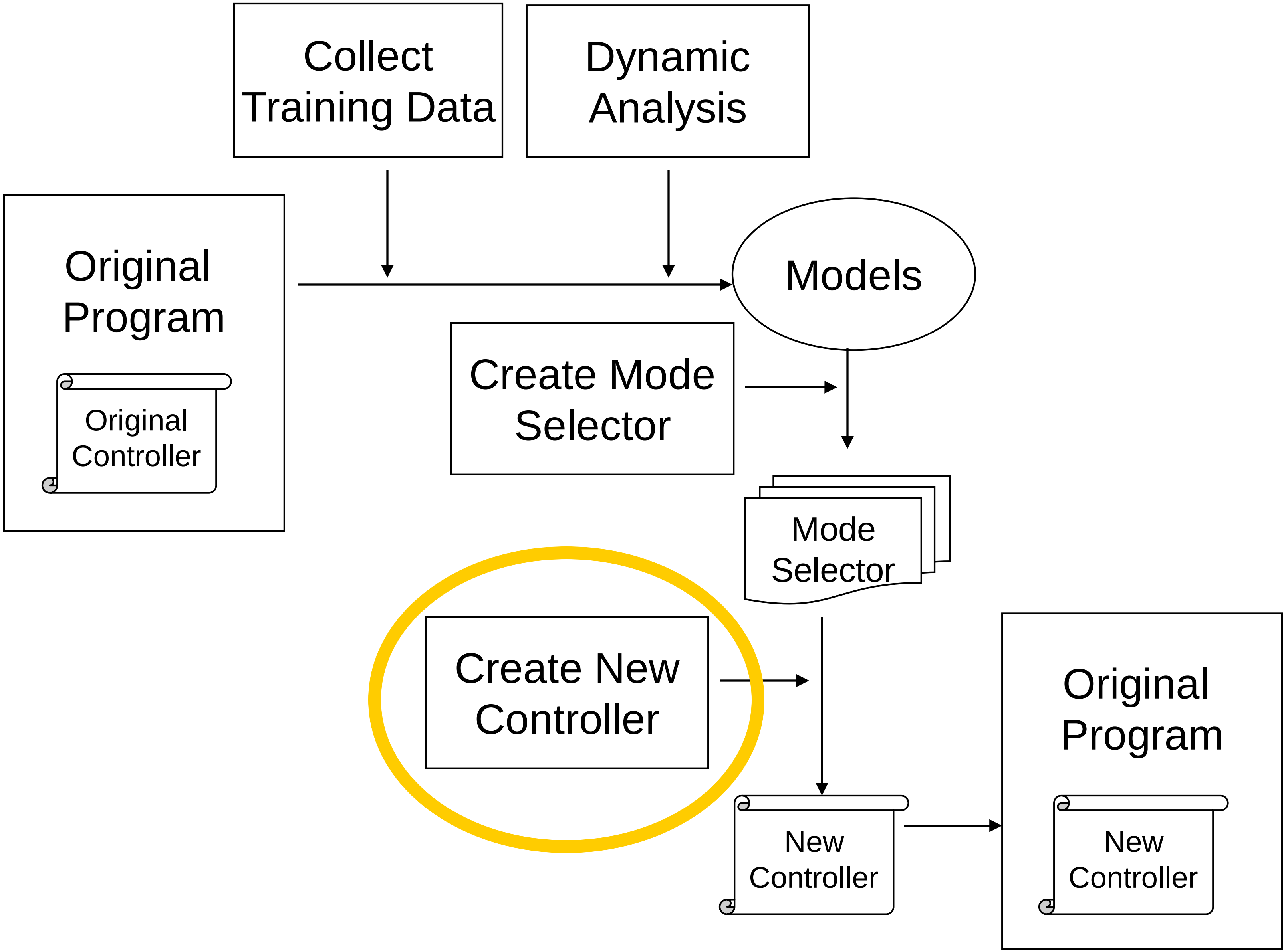
Check which properties in the models are true in the current program state.

For each mode, calculate a similarity score (percent of matching properties).

Choose the mode with the highest score.

Can also accept constraints, for example

- Don’t select Travel Mode when destination null
- Must switch to new mode after Exception


Collect Training Data
Dynamic Analysis
Original Program
Original Controller
Models
Create Mode Selector
Mode Selector
Create New Controller
New Controller
Original Program
New Controller

# Controller Augmentation

Call the new mode selector during:

- Uncaught Exceptions
- Timeouts
- Default / passive mode
- Randomly during mode transitions

Otherwise, the controller is unchanged

# Why Consider Mode Outputs?

- Mode selection considers all properties
- Output properties measure whether mode is behaving as expected
- Provides inertia, avoids rapid switching

- Suppose brightness is stuck at 3 (damaged).
  - No output benefit for Standard Mode.
  - More reason to prefer Power Saver to Standard.

# Why Should Program Steering Improve Mode Selection?

- Eliminates programmer assumptions about what is important
- Extrapolates knowledge from successful runs
- Considers all accessible information
- Every program state is handled

- The technique requires no domain-specific knowledge

# What Systems Can Benefit from Program Steering?

- Discrete transitions between modes
- Deployed in unpredictable environments
- Multiple modes often applicable

# Overview

- Program Steering Technique
- Mode Selection Example
- Program Steering Implementation
- Experimental Results
- Conclusions

# Droid Wars Experiments

- Month-long programming competition
- Teams of simulated robots are at war (27 teams total)
- Robots are autonomous
- Example modes found in contestant code: Attack, defend, transport, scout enemy, relay message, gather resources

# Program Steering Upgrades

- Selected 5 teams with identifiable modes
- Ran in the original contest environment
- Trained on victorious matches
- Modeling captured sensor data, internal state, radio messages, time elapsed

# Upgraded Teams

| Team | NCNB Lines of Code | Number of Modes | Properties Per Mode |
|---|---|---|---|
| Team04 | 658 | 9 | 56 |
| Team10 | 1275 | 5 | 225 |
| Team17 | 846 | 11 | 11 |
| Team20 | 1255 | 11 | 26 |
| Team26 | 1850 | 8 | 14 |

The new mode selectors considered many more properties than the original mode selectors

# Evaluation

- Ran the upgraded teams in the original environments (performed same or better)
- Created 6 new environments
  - Hardware failures: random rebooting
  - Deceptive GPS: navigation unreliable
  - Radio Spoofing: simulate replay attacks
  - Radio Jamming: some radio messages dropped
  - Increased Resources: faster building, larger army
  - New Maps: randomized item placement

# Examples of Environmental Effects

- Hardware Failures
  - Robot did not expect to reboot mid-task, far from base
  - Upgraded robots could deduce and complete task
- Radio Spoofing
  - Replay attacks resulted in unproductive team
  - Upgraded robots used other info for decision making

# Program Steering Effects on Tournament Rank

## Hardware Failures

| Team | Original | Upgrade | |
|---|---|---|---|
| Team04 | 11 | 5 | +6 |
| Team10 | 20 | 16 | +4 |
| Team17 | 15 | 9 | +6 |
| Team20 | 21 | 6 | +15 |
| Team26 | 17 | 13 | +4 |

## Deceptive GPS

| Team | Original | Upgrade | |
|---|---|---|---|
| Team04 | 12 | 9 | +3 |
| Team10 | 23 | 8 | +15 |
| Team17 | 15 | 9 | +6 |
| Team20 | 22 | 7 | +15 |
| Team26 | 16 | 13 | +3 |

# Original Team20

- Centralized Intelligence
- Queen Robot
  - Pools information from sensors and radio messages
  - Determines the best course of action
  - Issues commands to worker robots
- Worker Robot
  - Capable of completing several tasks
  - Always returns to base to await the next order

# Upgraded Team20

- Distributed Intelligence
- Queen Robot (unchanged)
- Worker Robot
  - Capable of deciding next task without queen
  - Might override queen's orders if beneficial

# Understanding the Improvement

Question:

What if the improvements are due to **when** the new controller invokes the new mode selector, not **what** the selector recommends?

Experiment:

- Ran same new controller with a random mode selector.
- Programs with random selector perform poorly.
- Program steering selectors make intelligent choices

# Comparison with Random Selector

## Hardware Failures

| Team | Original | Upgrade | | Random | |
|---|---|---|---|---|---|
| Team04 | 11 | 5 | +6 | 9 | +2 |
| Team10 | 20 | 16 | +4 | 21 | -1 |
| Team17 | 15 | 9 | +6 | 20 | -5 |
| Team20 | 21 | 6 | +15 | 23 | -2 |
| Team26 | 17 | 13 | +4 | 22 | -5 |

## Deceptive GPS

| Team | Original | Upgrade | | Random | |
|---|---|---|---|---|---|
| Team04 | 12 | 9 | +3 | 17 | -5 |
| Team10 | 23 | 8 | +15 | 18 | +5 |
| Team17 | 15 | 9 | +6 | 15 | 0 |
| Team20 | 22 | 7 | +15 | 21 | +1 |
| Team26 | 16 | 13 | +3 | 20 | -4 |

# Overall Averages

| Team | Upgrade | Random |
|---|---|---|
| Team04 | +4.0 | +0.7 |
| Team10 | +3.8 | +1.2 |
| Team17 | +4.2 | -1.5 |
| Team20 | +8.8 | -1.3 |
| Team26 | +1.0 | -3.7 |

# Overview

- Program Steering Technique
- Mode Selection Example
- Program Steering Implementation
- Experimental Results
- Conclusions

# Future Work

- Use other mode selection policies
  - Refine property weights with machine learning
  - Detect anomalies using models
- Try other modeling techniques
  - Model each transition, not just each mode
- Automatically suggest new modes

# Conclusion

- New mode selectors generalize original mode selector via machine learning
- Technique is domain independent
- Program steering can improve adaptability because upgraded teams perform:
  - As well or better in old environment
  - Better in new environments