Autonomous Robust Execution of Complex Robotic Missions

Paul Robertson, Robert T. Effinger, Brian C. Williams
MERS, CSAIL, MIT (papers on http://mers.mit.edu)

Goals

1. Enable development of complex missions with contingencies (Complexity)
2. Provide robustness to component, temporal, and contingency failure (Robustness)
3. Support online optimal temporal planning with contingencies (Optimality)

What I won’t talk about

1. Health Management/Maintenance
2. Mapping/localization
3. Path Planning
4. Distribution over multiple robots
5. Algorithms for Performance
6. Generative Activity Planner

Model-based Programming

- Suspicious
  - Monitors intentions and plans
- Self-Adaptive
  - Exploits and generates contingencies
- State and Fault Aware
  - Specified at level of states.
  - Achieved using failure knowledge.
- Anticipatory
  - Plans and verifies into the future
  - Predicts likely future states
  - Plans contingencies
  - “RMPL” reactive Model-Based Programming Language

What is a TPN?

- A Temporal Plan Network (TPN) is
  - A Simple Temporal Network extended to support:
    - Multiple Redundant Methods
    - Method Deprecation and Regeneration
    - Optimal Planning
  - Adds a choice node
  - Adds exceptions
  - Adds cost/rewards to arcs
  - Adds online replanning
Architecture Walkthrough

1. The human writes a mission program in RMPL.
2. The RMPL Compiler compiles the RMPL code and populates a library of TPN specifications. This library contains the main program, along with any macros that may be called.

Reformulating the TPN into a conditional optimal CSP

- **Step 1:**
  - Walk the TPN and create variables and constraints corresponding to the decision nodes

- **Step 2:**
  - Create variables and constraints corresponding to the non-causal link constraint arcs (these come from arbitrary temporal constraints)
Initialize: Reformulate TPN into CSP

Step 2: Create variables and constraints corresponding to the non-causal link constraint arcs.

V1={   }
V2={   ,   }
V3={   ,   }
V4={   ,   }
V5={   }
V6={   }

Solution Analysis: Temporal Consistency Check

If the temporal consistency checking algorithm detects a negative cycle, the conjunction of variable assignments that contributes to the negative cycle become a new conflict constraint.

Solution Analysis: Tell Consistency Check

Tell Consistency Check ensures that any potentially co-occurring mutually exclusive Tells are ordered so they do not co-occur.

Solution Analysis: Tell Consistency Check

The CSP is updated with a new variable and constraint. The TPN is updated with two new conflict avoidance arcs.
Solution Analysis: Ask Consistency Check

When we detect an Ask, we create a new CSP variable. The Ask variable's domain is empty, because we have not yet identified any satisfying Tells.

Location Consistency

Summary:
- Location Consistency must ensure that mutex location constraints can be ordered (like Tell Consistency)
- Location Consistency must add time-bounds that guarantee feasible travel-time exists between temporally-adjacent location constraints

Architecture Walkthrough

During execution, a primitive activity may fail, triggering an exception and replanning. Before replanning occurs, the Exception Handling module must process the executive's exception.

Solution Analysis: Exception Handling

1. Execution begins...
2. An error occurs, and an exception is thrown
3. The exception-handling code is inserted

The handler is the TPN sub-process corresponding to the RMPL “catch” statement that matches the thrown exception.
Recovering From Failure

1. Enable development of complex missions with contingencies (Complexity)
2. Provide robustness to component, temporal, and contingency failure (Robustness)
3. Support online optimal temporal planning with contingencies (Optimality)

Supports development of very complex missions by (i) raising the level of programming to coaching (from commanding); (ii) Sub-plan and component reuse.

Handles recovery (with graceful degradation) from: (i) Robot health failure; (ii) Temporal plan failure; (iii) Failed contingencies

Uses least commitment temporal planning for optimal solutions

Continuous monitoring and replanning

Wrap up

1. Supports development of very complex missions by (i) raising the level of programming to coaching (from commanding); (ii) Sub-plan and component reuse.
2. Handles recovery (with graceful degradation) from: (i) Robot health failure; (ii) Temporal plan failure; (iii) Failed contingencies
3. Uses least commitment temporal planning for optimal solutions

Continuous monitoring and replanning

Complexity
Robustness
Optimality

Architecture