Advanced Methods in Model-based Autonomy

Outline

• Introduction & Overview
• Model-based Programming
• Execution of Model-based Programs
• Fundamentals of Model-based Reasoning
• Modeling via State Analysis
• Advanced Methods
  – Timed Model-based Programming
  – Hybrid Model-based Programming
  – Model-based Temporal Planning
  – Integration of Activity Planning and Path Planning
  – Verification of Model-based Programs
• Conclusion
Model-based Autonomy Heritage

Directions:
- Model-based Programming & Execution
- Hybrid discrete-continuous systems
- Complex & collaborative behaviors

NMP Deep Space 1 Remote Agent

Related Work

- State-based Specifications
  - StateCharts (Harel, ’87)
  - Timed StateCharts (Kesten & Pnueli, ’92)

- Synchronous Programming
  - Esterel (Berry & Gonthier, ’92)
  - Lustre (Halbwachs, ’93)

- Constraint Programming
  - TCC (Saraswat, Jagadeesan & Gupta, ’94)

- Robotic Execution
  - RAPs (Firby, ’89)
  - ESL (Gat, ’96)
  - TDL (Simmons, ’98)

- Timed Formal Modeling
  - Timed Transition Systems
    - (Henzinger, Manna, & Pnueli, ’92)
  - Timed Automata (Alur & Dill, ’94)

- Model-based Execution
  - GDE, Sherlock (deKleer & Williams, ’87-’89)
  - Livingstone (Williams & Nayak, ’96-’97)
  - Livingstone2 (Kurien & Nayak, ’00)

- Model-based Programming
  - RBurton (Williams & Gupta, ’99)
  - Titan (Williams, Ingham, Chung & Elliott, ’03)

- Mission Data System
  - MDS (Dvorak, Rasmussen, et al., ’00)
Timed Model-based
Execution Architecture

Timed Model-based Program

Timed Control Program

Timed Plant Model

Control Sequencer

Deductive Controller

System Clock

Timers

Clocks

Observations

Commands

Timed Model-based Executive

State estimates

Configuration goals

Mode Estimation

Mode Reconfiguration

State estimate

Plant

Clock

Mode Estimation

Mode Reconfiguration

Timed Model-based Program

Timed Model-based Executive

Clocks

Observations

Commands

Timed Hierarchical Constraint Automata

• Graphical specification language for control programs, in spirit of Timed StateCharts
• Extend Hierarchical Constraint Automata to timed behavior

Mars Entry control program

- clock initialization
- transitions conditioned on clock variables
- clocks provide timing mechanism
- conditioned on time & state constraints
**Timed Concurrent Constraint Automata**

- Variant of Factored POSMDP (state not directly observable, next state depends on current state & time spent in state)

**Concurrent Constraint Automata to timed behavior**

- **Engine:**
  - Off
  - Firing
  - Standby
  - Failed

- **Camera:**
  - Idle
  - Taking Picture
  - Heating
  - Stuck Shutter

**Timed Mode Estimation**

- For physical plants modeled as TCCA (POSMDP):

**Bad news:**
state space gets much larger...

**Good news:**
can leverage existing OPSAT engine!
Hybrid Modeling and Estimation

- Model the system as a network of Probabilistic Hybrid Automata
- Frame fault diagnosis as state estimation in this model

Innovative features:
- non-linear dynamics
- mode transitions dependent on continuous state
- Detect failures or mode changes from subtle symptoms

Continuous state: flow $x$

$$x_{t+1} = q_{out} + N(0, 10^{-4})$$

$$q_{out} = x$$

$$q_{out} = 8.0$$

Flow Regulator

Hybrid Estimation

- Track a set of mode sequences with a bank of Kalman Filters

hybrid ePHa model & initial state information
Experimental Results:
K-best filtering

Hybrid Model-based Execution

- Model-based Program
  - Control Program
  - System Model

- Hybrid Hierarchical Constraint Automata (HHCA)

- Concurrent Probabilistic Hybrid Automata (CPHA)

- Model-based Executive
  - Control Sequencer
  - Mode Estimation
  - Mode Reconfiguration
  - Deductive Controller
Model-based Temporal Planning

High-Level Mission Planning

Temporal Planner

Reactive, Temporal Planner:
- Fast planning performed as graph search
- Encoding of non-deterministic choice
- Conditional planning via encoding of pre/post conditions and maintenance conditions.

Model-based Temporal Planning

Input: Planning models specified as RMPL models

Processing: Transforms RMPL models into intermediate HCA representation, then from HCA to Temporal Plan Network (TPN) representation.

Output: Temporally-constrained network of events
Model-based Temporal Planning

High-Level Mission Planning

Code for describing:
- Concurrent Task Management
- Resource Management
- Temporal Constraints
- Multiple Method Selection

RMPL Constructs:
- constraints \( c \)
- conditional execution \( \text{if } c \text{ then } A \)
- guarded transition \( \text{unless } c \text{ then } A \)
- full concurrency \( A, B \)
- iteration \( \text{always } A \)
- non-deterministic choice \( \text{choose } A, B, \ldots \)
- activity timing \( A \text{ within time } [t^-, t^+] \)
Model-based Temporal Planning

High-Level Mission Planning \(\rightarrow\) Temporal Planner

- TPN Representation
  - 1. STN-like temporal representation of upper & lower bounds between events
  - 2. Choice encoded via choice nodes (i.e. only one trajectory branches from it)
  - 3. Conditions encoded via \(\text{Ask}(C)\) & \(\text{Tell}(C)\)

RMPL Planning Model can be compiled into a Temporal Planning Network

1. 2
2. 3
3. 4
4. 5
5. 6
6. 7
7. 8
8. 9
9. 10
10. 11
11. 12
12. 13
13. 14
14. 15
15. 16
16. 17
17. 18
18. 19
19. 20
20.
Model-based Temporal Planning

High-Level Mission Planning

Deductive Mode Estimator & Reactive Planner

Onboard Sequencer Model

Temporal Planner

Planner traces set of paths from start to end of TPN that satisfy temporal & state constraints. Paths correspond to temporally consistent execution threads.

Programming Teams in RMPL

RMPL Programs

- Describe concurrent sensing, actuation and movements activities.
- **Choose** specifies redundant strategies and contingencies.
- **[A,B]** Specifies timing constraints.

```plaintext
{Group-Enroute()} [l,u] { 
  sequence
  choose {
    (do-watching (PATH1=OK)
      ((Group- Traverse-Path(PATH1_1,PATH1_2,PATH1_3,RE_POS)){1*90%,u*90%})
    )
    (do-watching (PATH2=OK)
      ((Group- Traverse-Path(PATH2_1,PATH2_2,PATH2_3,RE_POS)){1*90%,u*90%})
    )
  }
  (parallel
    ((Group-Transmit(OPS,ARRIVED)){0,2})
    (do-watching(PROCEED=SIGNALLED)
      ((Group-Wait(HOLD1,HOLD2)){0,u*10%}))
  )
}
```

Rendezvous
Corridor 2
Enroute
Corridor 1
**Integrated Activity & Path Planning for Agile Teams**

**Integrated Activity Planning & Path Planning:**
- Search a temporal plan network in best-first order
- Dynamically compute collision-free paths for those plan activities that require moving between locations and the estimated cost of flying along this path
- Continuously interleave activity and path planning to pursue the most promising plan.

**Path Planning Method 1:**
Explore state space using Rapidly-exploring Random Trees (RRTs)

**Maneuver Automaton:** Describes a set of agile maneuvers with respect to the vehicle’s dynamics

**Path Planning Method 2:**
Clausal Linear Programming

---

**Path Planning through Clausal LP**

**A simple example:**

- Mathematically solving the problem of vehicle control normally involves straightforward Linear Programming
- But the addition of obstacle avoidance introduces an Integer Programming element
- This makes the problem difficult to solve “online”: fast enough for actual vehicles in motion
- To resolve this we transform obstacle and collision avoidance into a Constraint Satisfaction Problem: For each obstacle, the domain is split into four regions (above, below, left, right), one of which is selected
- Integrating the selection of domains with the standard vehicle control leads to an algorithm that can be used as a Hybrid CSP/LP Solver

- \[ s_{i+1} = A s_i + B u_i \] State Evolution Equation
- \[ s_{ij} \leq w_{ij}, \text{ etc.} \] State Space Constraints
- \[ x_i \geq x_{\min} \lor x_i \leq x_{\max} \lor y_i \geq y_{\min} \lor y_i \leq y_{\max} \] Obstacle Avoidance (for all time i)
- Similar equation for Collision Avoidance (for all pairs of vehicles)
Motivation:
• Want robust autonomous systems.
• Extend traditional scenario-based testing to verification and validation (V&V).

Goals:
• Verify RMPL model-based programs (control program + plant model) against goal specification.
  e.g., \(((\text{EngineA} = \text{Firing}) \text{ OR } (\text{EngineB} = \text{Firing}))\) for OrbitInsert()
• Extract probabilistic information about program’s possible executions.

Verification of Model-based Programs

Approach:

Verification of Model-based Programs

Motivation:
• Want robust autonomous systems.
• Extend traditional scenario-based testing to verification and validation (V&V).

Goals:
• Verify RMPL model-based programs (control program + plant model) against goal specification.
  e.g., \(((\text{EngineA} = \text{Firing}) \text{ OR } (\text{EngineB} = \text{Firing}))\) for OrbitInsert()
• Extract probabilistic information about program’s possible executions.

Verification of Model-based Programs

Approach:

Up Next…

• Conclusion!