Autonomous Systems, Modeling, and Code Generation

Jonathan Sprinkle
Overview

• Autonomous Systems
  – Online Optimal Control (Pursuit/Evasion Games)
  – Decision Control (Landing Waveoff/Reissue)
  – Decision Authority (Distributed Mixed-Initiative Systems)
  – Composability (Vehicle Software and Components)

• Modeling
  – Domain-Specific Modeling
  – Metamodeling

• Code Generation
  – Controller synthesis
  – Systems modeling

• Cyber-Physical Systems
  – Overview
  – Application
Applications

- SEC Capstone Demonstration
- Pursuit/Evasion of fixed-wing aircraft
  - Joint work with Dr. Mike Eklund, Dr. Jin Kim, Prof. Shankar Sastry
Optimal Control through Model-Predictive Control

See Sprinkle et al., CDC2004

\[
J = \phi(\tilde{y}_N) + \sum_{k=0}^{N-1} L(x, \tilde{y}, u, d),
\]

(7)

where,

\[
\phi(\tilde{y}_N) \triangleq \frac{1}{2} \left( \tilde{y}_N^T P_0 \tilde{y}_N \right),
\]

(8)

and,

\[
L(x_k, \tilde{y}_k, u_k, d_k) \triangleq \frac{1}{2} \left( \tilde{y}_k^T Q \tilde{y}_k + x_k^T S x_k + u_k^T R u_k + \frac{1}{(d_k^T G d_k)^{1/4}} \right)
\]

(9)
(a) An attractive CF.  
(b) A one-sided attractive CF.  
(c) A pointwise repulsive CF  
(d) A linearly repulsive CF  

Fig. 2.  Basic forms of the cost functions used by the NMPTC
\[
L(x_k, \tilde{y}_k, u_k, d_k) \triangleq \begin{align*}
&\frac{1}{2} \tilde{y}_k^T Q \tilde{y}_k + \frac{1}{2} x_k^T S x_k + \frac{1}{2} u_k^T R u_k \\
&\frac{1}{2} p_{1_k}^T B_1 p_{1_k} + \frac{1}{2} p_{2_k}^T B_2 p_{2_k} \\
&\frac{1}{2^n} (d_k^T G d_k) + \frac{1}{2^n} (a_k^T H s_k)
\end{align*}
\]

\[ (8) \]
10-minute time limit (to prevent loitering by pursuer)
## System Architecture

<table>
<thead>
<tr>
<th>Development (Algorithm) by Technology Developer</th>
<th>Control Algorithm(s)</th>
<th>Platform</th>
<th>Testbed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Core OCP</td>
<td>ControlsAPI</td>
<td>Platform 1 ... $N_P$</td>
<td>Testbed 1 ... $N_T$</td>
</tr>
<tr>
<td>Deployment Layer by OCP Developer</td>
<td></td>
<td>Platform Independent Testbed Configurable</td>
<td></td>
</tr>
</tbody>
</table>

- **x86 Laptop, Linux RH9 (perfctr-kernel 2.4)**
- **Boeing T-33 Trainer Jet ca. 1953**
Pilot: “Plane reacted just like pilots are trained to do”

Me: “I couldn’t tell whether it was a computer or person controlling the plane”
• SEC Capstone Demonstration
• Landing/Wave-off scenario (safety calculation)
  • Joint work with Dr. Mike Eklund, Dr. Ian Mitchell, Prof. Shankar Sastry
Motivating Example

- A UAV is waved off, and then after some time redirected to land

Can the decision to safely land:
- be made in real time?
- be guaranteed as true?

See Sprinkle et al., ISSE2006
Fig. 3 Definitions of angular measurements in terms of body motion. Note that $x$ represents the longitudinal axis of the runway, $y$ represents the axis for the width of the runway, and $z$ is altitude. These directions are useful for the direction of motion for designing the control laws, but note that during reachability calculations many of these values will be negative due to the calculations backward in time.
Implementation and Results

See Sprinkle et al., ISSE2006

$$[x, y, \psi]$$
Implementation and Results

See Sprinkle et al., ISSE2006

\[ x, z, \theta \]

All pieces fit together, step size changes by power of 10 to match required resolution

[0,3)

[3,10)

3D rep of data in file ngc3_theta_400_1mm.mat

3D rep of data in file ngc3_theta_400_10mm.mat
Generative Strategy

See Sprinkle et al., ISSE2006

\[ \theta_d(x) = - \left( x_3 - \sin(\theta_G) \sqrt{x_1^2 + x_2^2 + x_3^2} \right) + \theta_G \]  

(29)

\[ \psi_d(x) = -(x_2 - \tan(\psi_G)x_1) + \psi_G \]  

(30)

From this, and utilizing (12), we obtain the desired feedback control law:

\[ P_{II}(x) = -U_{\text{max}}^i \Gamma_i(\theta - \text{Sat}_{\text{max}}^i(\theta_d(x))) \]  

(31)

\[ Q_{II}(x) = -U_{\text{max}}^i \Gamma_i(\psi - \text{Sat}_{\text{max}}^i(\psi_d(x))) \]  

(32)

where \( \theta_d \) and \( \psi_d \) are given as in (29) and (30).

\[ G_0 = \begin{cases} 
\theta_G \in [2.85^\circ, 3.15^\circ], \\
\psi_G \in [-0.2^\circ, +0.2^\circ], \\
x_2 \in [-100, +100] \text{ ft}, \\
x_3 \in [-15, +15] \text{ ft}, \\
x_1 = 0
\end{cases} \]

(c) Recapture of aircraft \( p_i \) to the glideslope, using the recapture controller according to the \( \mathcal{P}_{\text{next}} \) formulation.

(d) Aircraft \( p_i \) uses the “go-around” control law, and circles back into the landing set.
Scenario: Mixed-Initiative UAVs (e.g., AAR)

Joint work with Jerry Ding, Claire Tomlin, Shankar Sastry
Scenario: Automated Aerial Refueling

Target Set for Refueling

human pilot

human operated boom

Joint work with Jerry Ding, Claire Tomlin, Shankar Sastry
Joint work with Jerry Ding, Claire Tomlin, Shankar Sastry

Hybrid Systems Model

FB = Fall back command

$G_{ij}$ = Target Set of Maneuver from Stationary $i$ to Stationary $j$
Simulations: Tanker Waveoff

Joint work with Jerry Ding, Claire Tomlin, Shankar Sastry
Using Reachability Analysis

Joint work with Jerry Ding, Claire Tomlin, Shankar Sastry
Simulation: Full Sequence (with reachability)

Joint work with Jerry Ding, Claire Tomlin, Shankar Sastry
After attaching semantics to the FTL visual language, we will be able to synthesize the MATLAB scripts, based on generalizations of the prototypes which we’ve built by hand. Then, “fallback” states can change, based on the model built, not the static code.

This allows FTL users to design their own protocols for experimentation. We are also working on exploration logic to look for reachable discrete states (not just continuous).
Autonomous Vehicles

Joint work with Ben Upcroft, Hugh Durrant-Whyte (USyd), Will Uther, Robert Fitch (NICTA) Humberto Gonzalez, Esten Grøtli, Shankar Sastry (Berkeley) and MANY OTHERS!!!!

Sydney-Berkeley Driving Team

http://dgc3.eecs.berkeley.edu/
System Architecture: Component-Based Design

RNDF
- Global World Rep.
- Symbolic World Rep.
- Local World Representation

MDF
- Global Path Planning
- Symbolic Decisions
- Local Obstacle Avoidance

Software/Hardware Simulation Interface
- World State
- Vehicle State

Hardware Abstraction Layer
- Sensors
- Actuators

World
Domain-Specific Modeling

- Create *model* of the system
- Perform
  - Analysis
  - Architecture exploration
  - Simulation
- Generate
  - Configuration
  - Code
  - Executables

From the same models!

Example Domains & Environments:
- VLSI Layout (e.g., Altera)
- Engg Drawing (e.g., AutoCAD)
- Physical Modeling (e.g., SolidWorks)
- Signal Processing (e.g., LabVIEW)
- Controls (e.g., Simulink)
Domain-Specific Modeling: An abstract perspective

Domain Concepts

Unrestricted Implementation
Domain Concepts

Defns of Domain Assumptions and Givens
Domain-Specific Code Generation

DS Code Generator

Domain “Instance”
• Advantages:
  – Infer execution structure from domain assumptions
  – Reduce implementation-layer design/input errors
  – Keep implementation details flexible
  – Check design constraints during design
  – Restrict User’s Implementation Space

• Disadvantages:
  – Learning curve for design environment
  – Time to build design environment
  – Re-use cost
Metamodeling--Closing the Loop for Domain-Specific Modeling

Metamodeling Environment

- Formal Specifications
- Meta-Level Translation

DS Modeling Environment

- Model Builder
- Models
- Model Interpreters

Application Domain

- App. 1
- App. 2
- App. 3

Environment Evolution
Application Evolution

Model Interpretation
• Allows:
  – Rapid creation of Modeling Environment
  – Formal structure of Model Builder
  – Strong typing and constraint checking
  – Automatic Modeling Environment Generation

• Advantages:
  – Definition of metamodel strongly reflects system domain
  – Language can be visually defined and implemented
  – Documentation is the metamodel

• End results:
  – System design can be managed by domain experts, not software experts
  – Complex interdependencies checked through structural analysis, not enforced through style guides or memoranda
While Event $e_i$, and in State, $s_c$
After, $e_i . delay$, and in State, $s_c$,
Stop clock
If exists Transition $t_e$: ($src = s_c$, $dst = s_n$), set $s_c = s_n$
Else if $s_c . parent = null$, set $e_i = e_i . amSrc . sequence . dst$
Else transition through $s_c . parent$
Advance clock
Formal Definition of a Domain-Specific Language

\[ L = \langle C, A, S, M_c, M_s \rangle \]

Thanks to Janos Sztopanovits for the inspiration of this slide.
The Power of Modeling
The research initiative on Cyber-Physical Systems seeks new scientific foundations and technologies to enable the rapid and reliable development and integration of computer- and information-centric physical and engineered systems.

From http://varma.ece.cmu.edu/cps/

- A more certain restatement of core embedded systems research
- Aimed at
  - Computers whose sole existence is to control physical processes
  - Systems which cannot be built without computer interaction (not just sensing!)
  - High-confidence systems which are cost/time intensive to produce
    - Avionics
    - Veitronics
    - Health-care device/control
Conclusions

• Autonomous Systems require advanced software and design techniques to become manageable and understandable.

• Domain-Specific Modeling can aid in scoping a large systems problem, or in enabling new capabilities which current software engineering may make laborious.

• Cyber-Physical Systems, as an initiative, is an excellent opportunity to continue to develop the techniques of Domain-Specific Modeling toward unsolved (or expensive) problems.