Computers with Control Theory

Raghavendra Pradyumna Pothukuchi
Don’t Use Formal Control When You…

…hate high-level design

…believe you know *everything*

…love repentance over poor designs

…don’t care about the full system
Goals of the Tutorial

• Know some ways in which formal control can be applied to computers

• Know about the types of formal controllers and their application

• Get a feel for the design of a formal controller
  • Hands-on in a longer version next time!

• Know the strengths and limitations of control theory
A Typical Control Loop

Planner

Targets

High-level goals
min Energy \times \text{Delay}
Constant power

Deviations

Controller

Inputs or Parameters

Frequency
#cores
Cache size
ROB length

Outputs or Goals

Performance
Utilization
Power
Queue utilization

Outputs or Goals

High-level goals
min Energy \times \text{Delay}
Constant power

Inputs or Parameters

Frequency
#cores
Cache size
ROB length

Outputs or Goals

Performance
Utilization
Power
Queue utilization
Using Formal Controllers

Meeting fixed output targets

Video processing
Using Formal Controllers

Meeting fixed output targets
- Video processing

Meeting changing output targets
- Battery optimization

Graph showing:
- Power (% of max)
- Quality of Service (%) vs. Battery level (%)
Using Formal Controllers

Meeting fixed output targets
Video processing

Meeting changing output targets
Battery optimization

Optimization
\( \text{min Energy} \times \text{Delay} \)
First Step for Controller Design

**Dynamic system model**

\[
\text{outputs}(T) = m_1 \times \text{outputs}(T - 1) + m_2 \times \text{outputs}(T - 2) + \cdots + n_0 \times \text{inputs}(T) + n_1 \times \text{inputs}(T - 1) + \cdots
\]

When designing heuristics, the model was called *intuition*, and was *implicit*.

**System Identification**

Get models from experimental data.
System Identification

---

**Input**

- Frequency

---

**Output**

- Power

---

**DVFS**

<table>
<thead>
<tr>
<th>Time</th>
<th>DVFS</th>
<th>Power</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1.2</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>1.8</td>
<td>7.1</td>
</tr>
<tr>
<td></td>
<td>1.1</td>
<td>4.3</td>
</tr>
</tbody>
</table>

---

**DVFS**

<table>
<thead>
<tr>
<th>Time</th>
<th>DVFS</th>
<th>Power</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1.1</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>1.3</td>
<td>5.6</td>
</tr>
<tr>
<td></td>
<td>1.7</td>
<td>7</td>
</tr>
</tbody>
</table>

---

**DVFS**

<table>
<thead>
<tr>
<th>Time</th>
<th>DVFS</th>
<th>Power</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2.1</td>
<td>8.5</td>
</tr>
<tr>
<td></td>
<td>0.5</td>
<td>1.2</td>
</tr>
<tr>
<td></td>
<td>2.8</td>
<td>9</td>
</tr>
</tbody>
</table>

---

**Time**

---

---

---
Choices of Controllers

Controllers differ based on:

• their **policy** (how they generate inputs from deviations)
• the **hyper-parameters** they offer
• the **properties** they guarantee
Properties of Interest

**Stability:** Controller is not crazy
For small deviations, controller does not generate increasingly large input values

**Convergence:** Controlled system meets targets
Deviation will eventually get close to zero

**Robustness:** Controller is not crazy and controlled system meets targets in reality
Stability and convergence hold when design assumptions differ from reality
e.g., 20% model error, or unpredictable changes

**Optimality:** Controller is the best
Controller changes inputs in the best way to minimize deviations
PID Controller

PID: Proportional-Integral-Derivative

Input = \( P \times \) deviation + \( I \times \) (sum of deviations) + \( D \times \) (rate of deviation)

Instantaneous reaction  Smoothened reaction  Anticipatory reaction

Tools (Matlab, Octave, Mathematica) automatically tune \( P, I \) and \( D \)

Report stability, convergence

No optimality

Limited robustness

Guarantees may not hold when inputs can take only a few discrete values

e.g., \#cores : 1, 2, 3, 4
PID Controller: Uses

Applicable for Single Input Single Output (SISO) systems (e.g., RAPL)
LQG Controller

LQG: Linear Quadratic Gaussian
Multiple Input Multiple Output

Matrices \([A, B, C, D]\) and \(state\)

\[
\text{inputs} = C \times \text{state} + D \times \text{deviations}
\]

\[
\text{state}_{\text{new}} = A \times \text{state} + B \times \text{deviations}
\]

Every \(T\) time-units
LQG Controller: Hyper-parameters

Relative output priorities
e.g., power: 10 and performance: 1

Relative input cost
e.g., Frequency: 1 and #cores: 10

Relative ratio of output priorities and input costs determines inertia

Percentage of random error possible with the model
e.g., sensor noise, interrupts
LQG Controller: Properties and Uses

Properties: stability, convergence, optimality

Limited robustness

Non-random modeling errors are common in reality

e.g., multi-controller systems

 Guarantees may not hold when inputs can take only a few discrete values

 e.g., #cores: 1, 2, 3, 4

Useful when models are accurate, and inputs have many values
Robust Controller

Robust Controller

Targets \rightarrow \text{Deviations} \rightarrow \text{ Inputs} \rightarrow \text{Outputs}

External signals

\[ [A, B, C, D] \text{ and } state \]

inputs = C \times \text{ state} + D \times \text{ [deviations, external signals]} \]

\text{state}_{\text{new}} = A \times \text{ state} + B \times \text{ [deviations, external signals]} \]

So far: Controller is designed to work for the nominal specification: model, hyperparameters

Now: Controller works for a range of specifications: model, $\Delta$, hyperparameters
Robust Controller: Hyper-parameters

Uncertainty guardband for modeling error
Worst case, and can be large, e.g., 60%
 Doesn’t matter the reason why the models are wrong

Uncertainty guardband for many other phenomenon
Interference: In multi-controller systems, another controller may override inputs
Input discretization

Deviation bounds
e.g., power deviations must be below 10%

Input weights
Robust Controller: Properties and Uses

Wide applicability and composability because of robustness

Modular inter-operable controllers

Targets \(\pm\) Deviations \(\rightarrow\) Scheduling \(\rightarrow\) Utilization

Robust

Power

Frequency

Temperature

#cores

Robust

Targets

Power

[ISCA’18, CDC’18, CSM’19, MICRO’19]
Robust Controller: Properties and Uses

Wide applicability and composability because of robustness

Interfering controllers

- Other controllers
- Robust
- DVFS #cores
- Power Temperature

Limitations:

- If errors are beyond the guardband, no guarantees
- When guardband is too large, controller becomes slow
Model Predictive Controller (MPC)

At time $T$, generate: inputs($T$)  
inputs($T + 1$)  
⋮  
inputs($T + X$)

deviations will be minimized by this step
use model and current deviations for forecasting

apply inputs($T$)
MPC: Hyper-parameters

Output priorities

Input overheads

Constraints
E.g., range of inputs

Length of control plan
MPC: Properties and Uses

Provides stability, convergence, optimality

Good robustness

However, model accuracy plays a critical part because of prediction

Limitation: computationally intensive

Useful in many scenarios

The continuous re-planning approach makes it suitable for changing conditions
## Summary of Controller Choices

<table>
<thead>
<tr>
<th></th>
<th>PID</th>
<th>LQG</th>
<th>Robust</th>
<th>MPC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stability and Convergence</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>Optimality</td>
<td>✗</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>Robustness</td>
<td>✗</td>
<td>✗</td>
<td>✔️</td>
<td>✗</td>
</tr>
<tr>
<td>Forecasting</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✔️</td>
</tr>
<tr>
<td>Overhead</td>
<td>Very low</td>
<td>Low</td>
<td>Low – Medium</td>
<td>Medium – High</td>
</tr>
<tr>
<td>Uses</td>
<td>SISO e.g., RAPL</td>
<td>MIMO, accurate model, random noise</td>
<td>MIMO, poor models, multi-control systems</td>
<td>MIMO, multi-control systems and forecasting if accurate models</td>
</tr>
</tbody>
</table>
• Inputs:
  • Dynamic Voltage-Frequency Scaling
  • Idleness
Robust Control: Design

- Dynamic model of the system  
  Already done

- Specify hyper-parameters  
  - Guardband for modeling error  
  - Finite and discrete inputs  
  - Deviation bounds  
  - Input overheads  
  Use intuition, tools

- Analyze controller

What’s the difference from heuristic design?
- Able to think at a high-level  
- Only a few obvious settings  
- Easily know impact of choices
Control Theory: When and When Not?

Dynamic systems
Quantitative inputs e.g., #cores: 1, 2, 3, …

Fast and low-overhead decisions

Cannot handle categorical inputs
Scheduling policy A vs policy B
Which cores? (different from, how many cores?)

Less effective when runtime system behavior is very different
Guardbands and robustness help to a good extent – what if we need extreme-efficiency?
Image Sources

This Photo by Unknown Author is licensed under CC BY-NC-ND

This Photo by Unknown Author is licensed under CC BY-NC

This Photo by Unknown Author is licensed under CC BY-SA