From Physics to Interrupt Handlers: The Real to Float Step

Vivien Maisonneuve
CRI, MINES ParisTech

Presentation at Toccata

November 23, 2012

Different levels of description

In control engineering, work on different levels to design and build a control system:

- Format/high-level aspects: system conception, modeling, possibly proof.
- Concrete/low-level aspects: creation of an object implementing the system.
  Quadricopter, DRONE Project, MINES ParisTech & ECP.

Formal aspect

model $\longrightarrow$ object

System definition:

- Inputs: sensors [accelerometer, sonar. . .] + references [operator instructions].
  Outputs: actions to act on environment [rotors rotation speed].
- Modeling in the form of equations to express relations between inputs and outputs: differential equations/transfer functions between IOs.

Formal aspect

model ⟶ object

System definition:

- Inputs: sensors [accelerometer, sonar. . .] + references [operator instructions].
  Outputs: actions to act on environment [rotors rotation speed].
- Modeling in the form of equations to express relations between inputs and outputs: differential equations/transfer functions between IOs.

System requirements:

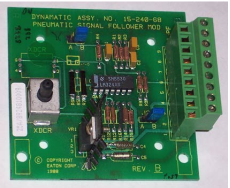
- Stability conditions [bounded rotation speed].
- Pursuit of reference input [try to reach the ordered position].
- Perturbation rejection [wind].

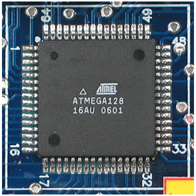
Concrete aspect

model $\longrightarrow$ object

Creation of a real object implementing the system.

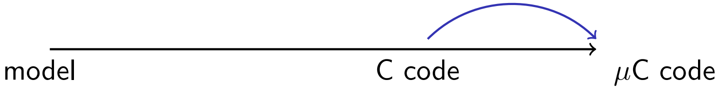
- Electronic circuit that physically computes the transfer function.
- With a **microcontroller**: a small system with processor, memory, I/O devices, that runs a program implementing the transfer function.





[ATMEGA128
Frequency: 16 MHz
RAM: 4 KB
Prog. mem.: 128 KB]

Semantic gap

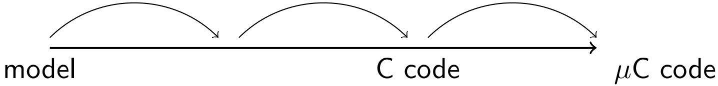


Antagonism:

- Abstract, mathematical model.
- Microcontroller code: program written in C, then compiled.
  Long (thousands of LoC), low-level (elementary operations, hardware management, interruptions).

Series of transformations to go from abstract model to microcontroller code.

Semantic gap



Antagonism:

- Abstract, mathematical model.
- Microcontroller code: program written in C, then compiled.

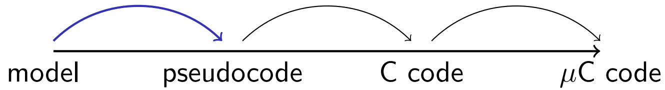
Series of transformations to go from abstract model to microcontroller code.

Problem of proof transposition: Considering a model with correction proofs [stability], how to transpose down these proofs at the code level?

Interest: formally check the code, not only the model.

Difficulties: semantic gap, non-equivalent transformations ($\Rightarrow$ proofs must be checked).

Control-theoretical aspects



Produce a pseudocode from the abstract model:

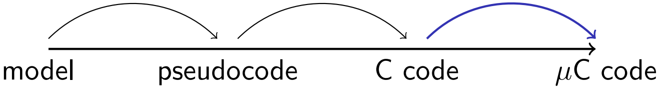
- Solve the model differential equation, get a transfer function.
  (Laplace transform/Z transform, initial conditions problem.)
- If continuous-time model, discretization.
  (Problems with sampling, execution times.)

while transposing the proof.

Usual problems in control engineering.

Once done, discrete-time system with a loop on the transfer function $\Rightarrow$ pseudocode [in MATLAB]. Proof: invariants on this code.

Compilation aspects



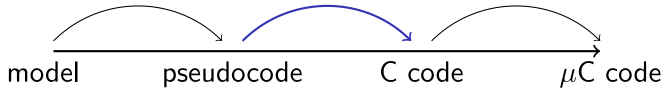
At the other end: compilation of C code to machine code.
Risks of error:

- Important changes in the code: elementary operations, management of registers and of memory stack, instruction jumps.
- Possible optimizations.

Solutions:

- "Existing C compilers are reliable enough."
- Proof-preserving compilation [Barthe].
- Certified compilation [CompCert].

What's between?

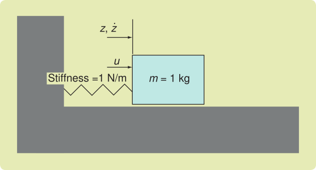


Opener question. Several challenges:

1. ~~High level mathematical operations~~ $\leadsto$ series of elementary instructions [matrices, sinus].
2. ~~Real values~~ $\leadsto$ machine words with limited precision.
3. On a microcontroller, data/events acquisition raises interruptions (real-time answer, energy consumption) $\Rightarrow$ particular code structure.

Example system

Very simple, open-loop, linear system [Feron].



Pseudocode:

```
Ac = [0.4990, -0.0500; 0.0100, 1.0000];
Bc = [1;0];
Cc = [564.48, 0];
Dc = -1280;

xc = zeros(2,1);

receive(y,2); receive(yd,3);
while 1
  yc = max(min(y - yd,1),-1);
  u = Cc*xc + Dc*yc;
  xc = Ac*xc + Bc*yc;
  send(u,1);
  receive(y,2);
  receive(yd,3);
end
```

state matrix (matrice de dynamique)
input matrix (matrice de commande)
output matrix (matrice d'observation)
feedthrough matrix (matrice d'action directe)

$x_c = \begin{pmatrix} x_{c_1} \\ x_{c_2} \end{pmatrix} \in \mathbb{R}^2$: controller state

$y \in \mathbb{R}$: reference input; $y_d \in \mathbb{R}$: real position

$y_c \in [-1, 1]$: bounded gap
$u \in \mathbb{R}$: action to be performed

`send`, `receive`: blocking, 2nd arg. is channel id

Lyapunov theory

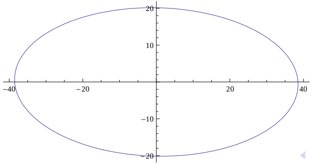
(Lyapunov) stability: all reachable states x_c start near an equilibrium point x_e and stay in a neighborhood V of x_e forever.

Lyapunov theory: NSC on V. On linear systems, provided as an equation that can be solved with LMIs, generally as an ellipsoid.

Here, show that $x_c = \begin{pmatrix} x_{c_1} \\ x_{c_2} \end{pmatrix}$ belongs to the ellipse:

$$\mathcal{E}_P = \{x \in \mathbb{R}^2 \mid x^T \cdot P \cdot x \leq 1\}, \qquad P = 10^{-3} \begin{pmatrix} 0,6742 & 0,0428 \\ 0,0428 & 2,4651 \end{pmatrix}.$$

$$x_c \in \mathcal{E}_P \Longleftrightarrow 0.6742x_{c_1}^2 + 0.0856x_{c_1}x_{c_2} + 2.4651x_{c_2}^2 \leq 1000.$$



Stability proof

```
xc = zeros(2,1);
```
$x_c \in \mathcal{E}_P$
```
receive(y,2); receive(yd,3);
```
$x_c \in \mathcal{E}_P$
```
while 1
```
$x_c \in \mathcal{E}_P$
```
  yc = max(min(y - yd,1),-1);
```
$x_c \in \mathcal{E}_P, \quad y_c^2 \leq 1$

$\begin{pmatrix} x_c \\ y_c \end{pmatrix} \in \mathcal{E}_{Q_\mu} \quad | \quad Q_\mu = \begin{pmatrix} \mu P & 0_{2\times 1} \\ 0_{1\times 2} & 1-\mu \end{pmatrix}, \mu = 0.9991$
```
  u = Cc*xc + Dc*yc;
```
$\begin{pmatrix} x_c \\ y_c \end{pmatrix} \in \mathcal{E}_{Q_\mu}$
```
  xc = Ac*xc + Bc*yc;
```
$x_c \in \mathcal{E}_{\tilde{P}} \quad | \quad \tilde{P} = \left[\begin{pmatrix} A_c & B_c \end{pmatrix} \cdot Q_\mu^{-1} \cdot \begin{pmatrix} A_c & B_c \end{pmatrix}^T \right]^{-1}$
```
  send(u,1);
```
$x_c \in \mathcal{E}_{\tilde{P}}$
```
  receive(y,2);
```
$x_c \in \mathcal{E}_{\tilde{P}}$
```
  receive(yd,3);
```
$x_c \in \mathcal{E}_{\tilde{P}}$

$x_c \in \mathcal{E}_P$
```
end
```

Proof given as code invariants.

Implication (weakening) if two consecutive invariants.

Most of them easy to check, some depend on theorems.

Last implication: $\mathcal{E}_{\tilde{P}} \subseteq \mathcal{E}_P$ closes the loop. Validity relies on parameters A_c, B_c, C_c, D_c, μ: algebric or numerical verification needed.

Digression: with C instructions

~~High level mathematical operations~~ $\rightsquigarrow$ series of scalar elementary instructions.

Here, matrix operations are expanded: the instruction

$\begin{pmatrix} x_c \\ y_c \end{pmatrix} \in \mathcal{E}_{Q_\mu}$

```
xc = Ac*xc + Bc*yc;
```

$x_c \in \mathcal{E}_{\tilde{P}} \quad | \quad \tilde{P} = \left[\begin{pmatrix} A_c & B_c \end{pmatrix} \cdot Q_\mu^{-1} \cdot \begin{pmatrix} A_c & B_c \end{pmatrix}^T \right]^{-1}$

becomes:

$\begin{pmatrix} x_c \\ y_c \end{pmatrix} \in \mathcal{E}_{Q_\mu}$

```
xb[0] = xc[0];
xb[1] = xc[1];
xc[0] = Ac[0][0]*xb[0]+Ac[0][1]*xb[1]+yc;
xc[1] = Ac[1][0]*xb[0]+Ac[1][1]*xb[1];
```

xb: buffer

???

Digression: with C instructions

~~High level mathematical operations~~ $\rightsquigarrow$ series of scalar elementary instructions.

Here, matrix operations are expanded: the instruction

$\begin{pmatrix} x_c \\ y_c \end{pmatrix} \in \mathcal{E}_{Q_\mu}$

```
xc = Ac*xc + Bc*yc;
```

$x_c \in \mathcal{E}_{\tilde{P}} \quad | \quad \tilde{P} = \left[\begin{pmatrix} A_c & B_c \end{pmatrix} \cdot Q_\mu^{-1} \cdot \begin{pmatrix} A_c & B_c \end{pmatrix}^T \right]^{-1}$

becomes:

$\begin{pmatrix} x_c \\ y_c \end{pmatrix} \in \mathcal{E}_{Q_\mu}$

```
xb[0] = xc[0];                              xb: buffer
xb[1] = xc[1];
xc[0] = Ac[0][0]*xb[0]+Ac[0][1]*xb[1]+yc;
xc[1] = Ac[1][0]*xb[0]+Ac[1][1]*xb[1];
```

$x_c \in \mathcal{E}_{\tilde{P}} \quad | \quad \tilde{P} = \left[\begin{pmatrix} A_c & B_c \end{pmatrix} \cdot Q_\mu^{-1} \cdot \begin{pmatrix} A_c & B_c \end{pmatrix}^T \right]^{-1}$

Same computation: output invariant can be found [Feron].

Numerical precision problems

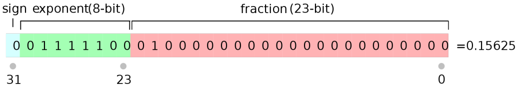
To produce C code: ~~real numbers~~ $\rightsquigarrow$ binary finite-length machine words (32 b. or 64 b.).

$\Rightarrow$ Loss in accuracy, two consequences:

1. Constant values are slightly altered.
2. Rounding errors during computations.

Machine representation of real numbers

1. Floating point: IEEE 754.
   Not usual on microcontrollers.



$$\text{number} = \text{sign} \times 2^{\text{exponent}+\text{cst. offset}} \times \text{fraction}$$

   Correct rounding for base operations: $+$, $-$, $*$, $/$.
   $\Rightarrow$ If [bounds on] operands are known, not special, far enough from extremal values, then rounding error is bounded for $+$, $-$, $*$ (not $/$).

2. Fixed point.
   If operands are not special, far enough from extremal values, then rounding error is bounded for $+$, $-$, $*$.

3. Two integers.

Machine representation of real numbers

1. Floating point.
2. Fixed point.
3. Two integers. Rational representation: numerator, denominator.
   - Base behavior: +, −, ∗, / follow rational definition + fraction simplification:

     $$\frac{p_1}{q_1} + \frac{p_2}{q_2} = \text{simpl}\left(\frac{p_1 q_2 + p_2 q_1}{q_1 q_2}\right), \text{ etc.}$$

     No rounding error.
     Problem: numerator value can easily exceed integer bounds.
   - Approximated behavior to ensure bounded numerator.

Alteration of constants

With IEEE 754, 32 bits, constants

```
Ac = [0.4990, -0.0500; 0.0100, 1.0000];
Bc = [1;0];
Cc = [564.48, 0];
Dc = -1280;
```

become

```
Ac ≈ [0.49900001287460327 , -0.05000000074505806;
      0.009999999776482582,  1.0000];
Bc ≈ [1;0];
Cc ≈ [564.47998046875, 0];
Dc ≈ -1280;
```

Effect on proof

```
xc = zeros(2,1);
```
$x_c \in \mathcal{E}_P$
```
receive(y,2); receive(yd,3);
```
$x_c \in \mathcal{E}_P$
```
while 1
```
$x_c \in \mathcal{E}_P$
```
  yc = max(min(y - yd,1),-1);
```
$x_c \in \mathcal{E}_P, \quad y_c^2 \leq 1$

$\begin{pmatrix} x_c \\ y_c \end{pmatrix} \in \mathcal{E}_{Q_\mu} \quad | \quad Q_\mu = \begin{pmatrix} \mu P & 0_{2\times 1} \\ 0_{1\times 2} & 1-\mu \end{pmatrix}, \mu = 0.9991$
```
  u = Cc*xc + Dc*yc;
```
$\begin{pmatrix} x_c \\ y_c \end{pmatrix} \in \mathcal{E}_{Q_\mu}$
```
  xc = Ac*xc + Bc*yc;
```
$x_c \in \mathcal{E}_{\tilde{P}} \quad | \quad \tilde{P} = \left[\begin{pmatrix} A_c & B_c \end{pmatrix} \cdot Q_\mu^{-1} \cdot \begin{pmatrix} A_c & B_c \end{pmatrix}^T \right]^{-1}$
```
  send(u,1);
```
$x_c \in \mathcal{E}_{\tilde{P}}$
```
  receive(y,2);
```
$x_c \in \mathcal{E}_{\tilde{P}}$
```
  receive(yd,3);
```
$x_c \in \mathcal{E}_{\tilde{P}}$

$x_c \in \mathcal{E}_P$
```
end
```

Rest of the code and proof sketch unchanged.

$\tilde{P}$ depends on A_c, B_c, C_c, D_c, is altered.

$\Rightarrow$ Check that $\mathcal{E}_{\tilde{P}} \subseteq \mathcal{E}_P$ still holds.

Rounding errors

With real numbers, the implication

$\begin{pmatrix} x_c \\ y_c \end{pmatrix} \in \mathcal{E}_{Q_\mu}$

```
xc = Ac*xc + Bc*yc;
```

$x_c \in \mathcal{E}_{\tilde{P}} \quad | \quad \tilde{P} = \left[\begin{pmatrix} A_c & B_c \end{pmatrix} \cdot Q_\mu^{-1} \cdot \begin{pmatrix} A_c & B_c \end{pmatrix}^T \right]^{-1}$

holds.

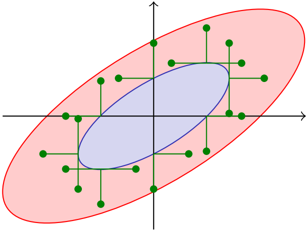
With floats, + and * introduce rounding errors.

As x_c, y_c belong to an ellipsoid, they are bounded so the rounding error on x_c can be bounded by (e_1, e_2).

Super-ellipsoid

Let $\mathcal{E}_{\tilde{F}} \supset \mathcal{E}_{\tilde{P}}$ an ellipse s.t.

$$\forall x_c \in \mathcal{E}_{\tilde{P}},\ \forall x'_c \in \mathbb{R}^2,\ |x'_{c_1} - x_{c_1}| \leq e_1 \wedge |x'_{c_2} - x_{c_2}| \leq e_2 \Longrightarrow x'_c \in \mathcal{E}_{\tilde{F}} \quad (*)$$



Then:

$\begin{pmatrix} x_c \\ y_c \end{pmatrix} \in \mathcal{E}_{Q_\mu}$

```
xc = Ac*xc + Bc*yc;
```

$x_c \in \mathcal{E}_{\tilde{F}}$

$\mathcal{E}_{\tilde{F}}$ can be the smallest magnification of $\mathcal{E}_{\tilde{P}}$ s.t. $(*)$ holds.

Can be computed, whatever number of dimensions.

Effect on proof

```
xc = zeros(2,1);
```
$x_c \in \mathcal{E}_P$
```
receive(y,2); receive(yd,3);
```
$x_c \in \mathcal{E}_P$
```
while 1
```
$x_c \in \mathcal{E}_P$
```
  yc = max(min(y - yd,1),-1);
```
$x_c \in \mathcal{E}_P, \quad y_c^2 \leq 1$

$\begin{pmatrix} x_c \\ y_c \end{pmatrix} \in \mathcal{E}_{Q_\mu} \quad | \quad Q_\mu = \begin{pmatrix} \mu P & 0_{2\times 1} \\ 0_{1\times 2} & 1-\mu \end{pmatrix}, \mu = 0.9991$
```
  u = Cc*xc + Dc*yc;
```
$\begin{pmatrix} x_c \\ y_c \end{pmatrix} \in \mathcal{E}_{Q_\mu}$
```
  xc = Ac*xc + Bc*yc;
```
$x_c \in \mathcal{E}_{\tilde{F}}$
```
  send(u,1);
```
$x_c \in \mathcal{E}_{\tilde{F}}$
```
  receive(y,2);
```
$x_c \in \mathcal{E}_{\tilde{F}}$
```
  receive(yd,3);
```
$x_c \in \mathcal{E}_{\tilde{F}}$

$x_c \in \mathcal{E}_P$
```
end
```

Replace $\mathcal{E}_{\tilde{P}}$ by $\mathcal{E}_{\tilde{F}}$ in proof sketch.

$\Rightarrow$ Check that $\mathcal{E}_{\tilde{F}} \subseteq \mathcal{E}_P$ holds.

Here it works: system stable with floats ☺.

Other functions

Elementary operations +, $*$ are sufficient for linear, invariant systems.
The method applies if the proof sketch fits: no tight assumptions, complex operations on weakened invariants.

1-var, differentiable, periodic functions can be computed

- with an abacus and a polyhedral interpolation function
- with a polyhedral approximation

with a bounded error (sin, cos).

Idem for 1-var, differentiable functions restricted to a finite range. OK if proof ensures the operand is bounded to the range.

Closing the loop

Modeling the result of the effects of the action on the environment, with feedback.

Design: here, two parallel, synchronized programs:

controller + plant (abstract).

```
Ac = [0.4990, -0.0500; 0.0100, 1.0000];
Bc = [1;0];
Cc = [564.48, 0];
Dc = -1280;
xc = zeros(2,1);
receive(y,2); receive(yd,3);
while 1
  yc = max(min(y - yd,1),-1);
  u = Cc*xc + Dc*yc;
  xc = Ac*xc + Bc*yc;
  send(u,1);
  receive(y,2);
  receive(yd,3);
end
```

```
Ap = [1.000, 0.0100; -0.0100, 1.000];
Bp = [0.00005; 0.01];
Cp = [1, 0];

while (1)
  yp = Cp * xp;
  send(yp,2);
  receive(up,1);
  xp = Ap * xp + Bp * up;
end
```

System is not linear.

Proving the system

Lyapunov stability: global state (x_c, x_p) in some ellipsoid $\mathcal{E}_P$.
$\Rightarrow +$ Boundedness of variables in physical system.

Difficulties:

- Non-linearity issues: trickier to find a suitable $\mathcal{E}_P$, post-condition to y_c definition.
  Usual case here, has been dealt.
- Handling concurrency in invariants: switch between system and plant analysis.

Proving the system

```
Ac = [0.4990, -0.0500; 0.0100, 1.0000];
Bc = [1;0];
Cc = [564.48, 0];
Dc = -1280;
xc = zeros(2,1);
receive(y,2); receive(yd,3);
while 1
  yc = max(min(y - yd,1),-1);
  u = Cc*xc + Dc*yc;
  xc = Ac*xc + Bc*yc;
  send(u,1);
  receive(y,2);
  receive(yd,3);
end
```

```
Ap = [1.000, 0.0100; -0.0100, 1.000];
Bp = [0.00005; 0.01];
Cp = [1, 0];


while (1)
  yp = Cp * xp;
  send(yp,2);
  receive(up,1);
  xp = Ap * xp + Bp * up;
end
```

Proving the system

Lyapunov stability: global state (x_c, x_p) in some ellipsoid $\mathcal{E}_P$.
$\Rightarrow +$ Boundedness of variables in physical system.

Difficulties:

- Non-linearity issues: trickier to find a suitable $\mathcal{E}_P$, post-condition to y_c definition.
  Usual case here, has been dealt.
- Handling concurrency in invariants: switch between system and plant analysis.
- Invariants of greater dimension: cannot test algebraically invariant inclusion, fails with floats.
- C code with interrupts.

```
SIGNAL(2)          SIGNAL(3)          while(1) {
  y = ...            yd = ...           sleep();
  ...                ...              }
```

From Physics to Interrupt Handlers: The Real to Float Step

Vivien Maisonneuve
CRI, MINES ParisTech

Presentation at Toccata

November 23, 2012