

## Vision and Control in Hume

**Norman Scaife**, Jocelyn Sérot

LASMEA, Les Cezeaux, F-63177 Aubiere cedex, France

<http://www.lasmea.univ-bpclermont.fr/>  
<http://www.lasmea.univ-bpclermont.fr/>

## Outline

- The EmBounded project
- The Hume language
- Canny edge detector
- PID controller
- CyCab application
- Conclusions

## The EmBounded project

- IST Framework VI project
- St. Andrews, Heriot-Watt, Ludwig Max., LASMEA, AbsInt
- Promote the Hume language for embedded systems work
- Develop compiler and costing/analysis tools
- Verify on applications from vision-based control

## The Hume language

- Slightly restricted functional language
- FSM-like coordination layer, dataflow-ish
- Nodes are “boxes”, triggered by pattern matching
- Determinism artificially enforced by match exception
- Support for embedded systems: timeouts, within, ...
- I/O: URIs, memory, interrupts, ...
- Underlying concept: **Costability**

## Simple Hume program

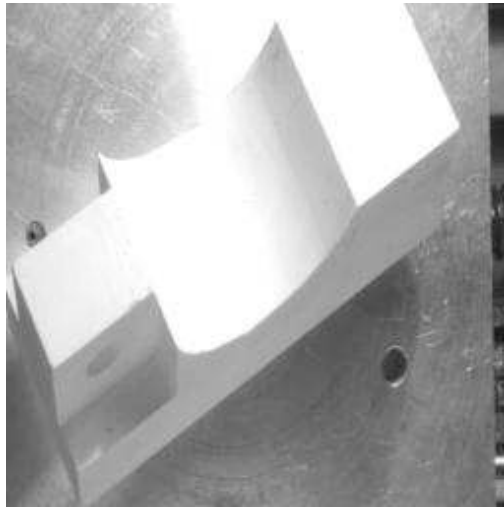
```
stream stdin from "std_in";
stream stdout to "std_out";

get_hello n = ("Hello world " ++ (n as string) ++ "\n", n+1);

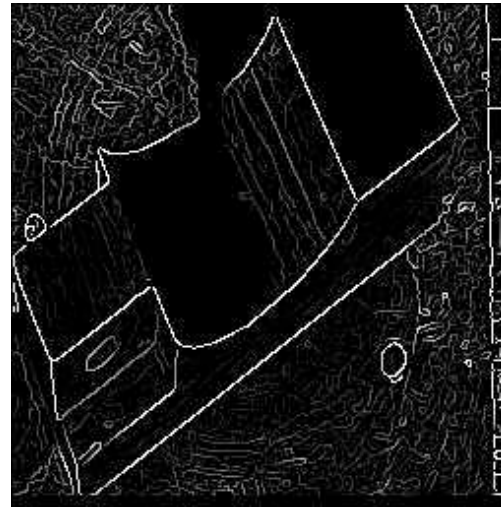
box hello
in (stdin::char, cnt::int 16)
out (stdout::string 32, cnt'::int 16)
match (_, n) -> get_hello n
  | (*, *) -> (*, *);

wire hello (stdin, hello.cnt' initially 0)
          (stdout, hello.cnt);
```

## Application: Canny edge detector



(a) Original Image

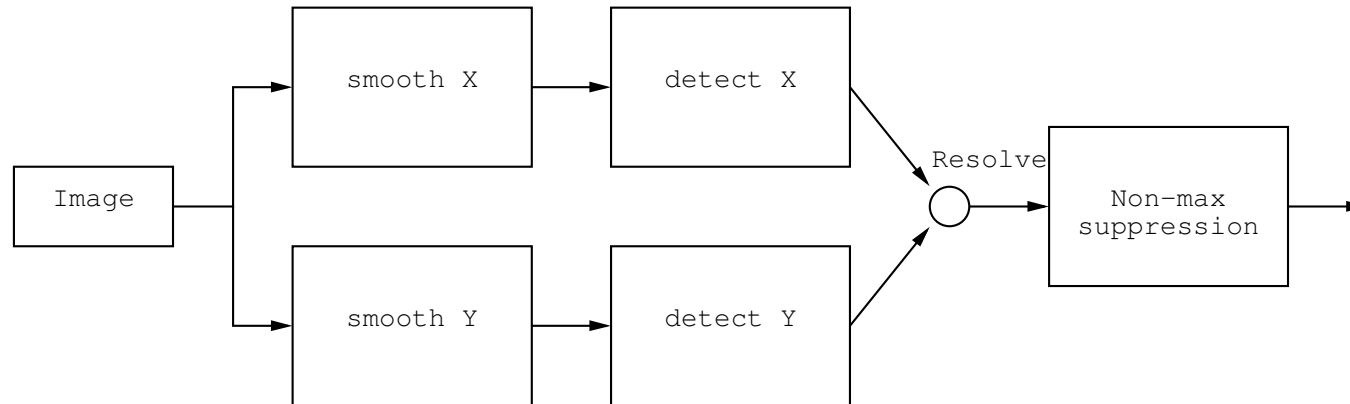


(b) Edge Strength Map



(c) Edge Angle Map

## Canny: Phases



- Three phases:
  1. Image smoothing (X and Y components)
  2. Edge detection (X and Y components)
  3. Non-maximum suppression
- All phases are local window operations

## Canny: Versions

- Translated from legacy SML sources
- Translated into Full Hume
- Three versions implemented:
  1. List-based “directly” translated from SML
  2. Vector-based derived from list-based version
  3. Box-based, almost from scratch

## Canny: List-based from SML

- Hand expansion of code:

```
val convol_RI = mapconv addRR (mapconv addRR multRI 0.0) 0.0;  
convol_RI fltY img;
```

becomes:

```
mapconvaddRRmultRI0 filt img = mapconv addRR multRI 0.0 filt img;  
convol_RIfilty img = mapconv addRR mapconvaddRRmultRI0 0.0 flty img;  
convol_RIfilty img;
```

- Difficult to parameterize program
- Otherwise looks very similar to original code

# Canny: Vector-based from list-based

- Simply replace lists with vectors
- Some problems with, eg. patterns:

```
non_max_sup filts =  
  case filts of  
    (((ux_1_2, uy_1_2) : ((ux_1, uy_1) : ((ux_12, uy_12) : 11)))) :
```

becomes:

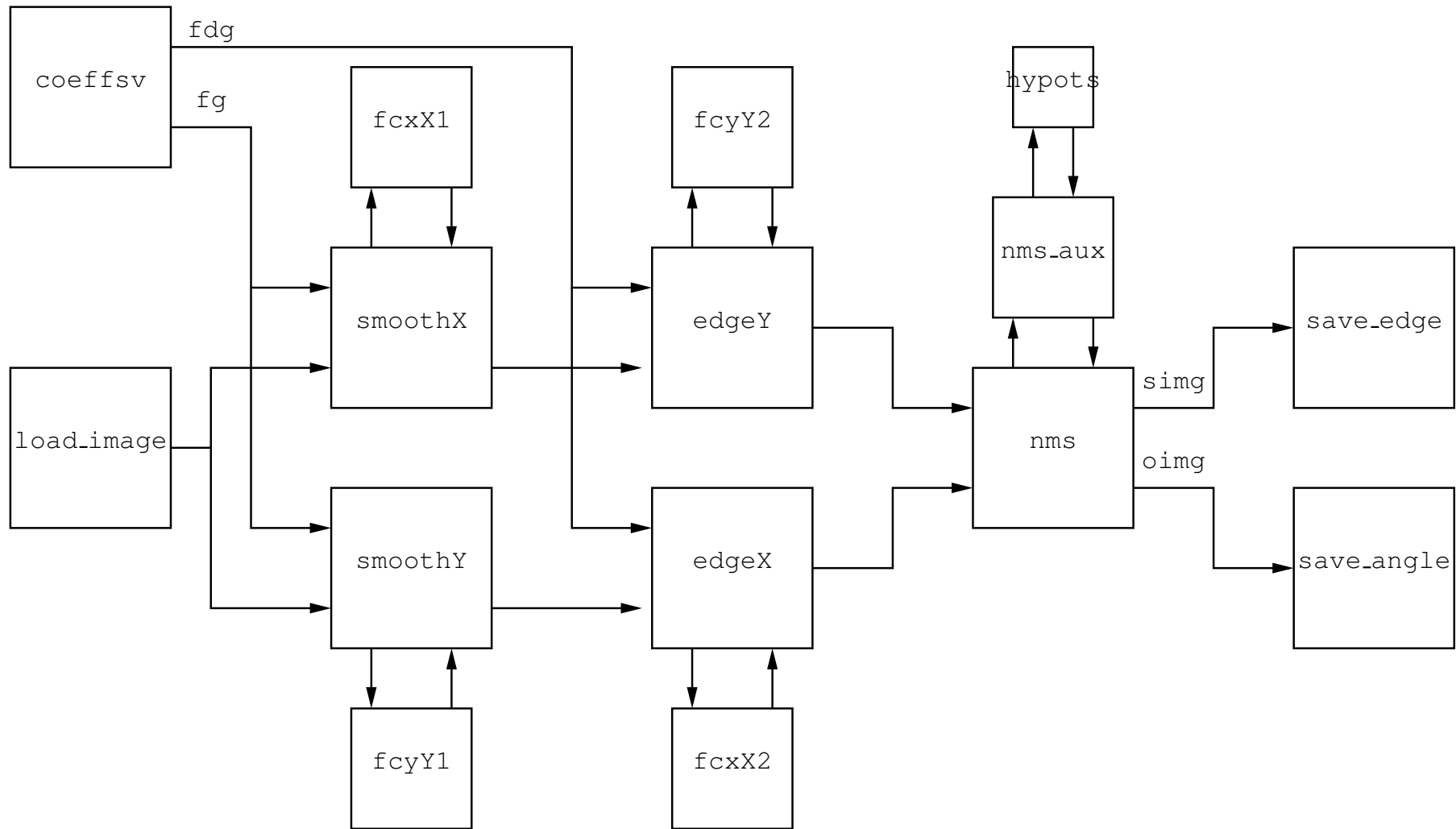
```
non_max_sup imgx imgy x y =  
  let ux_1_2 = (imgx@(y-1))@(x-1) in  
  let ux_1   = (imgx@(y-1))@(x)   in  
  let ux_12  = (imgx@(y-1))@(x+1) in
```

- Not really a “native” vector-based version
- Could rephrase using `vecmap` and `vecfold`

## Canny: Native box-based Hume

- Almost a complete rewrite
- Replace recursive functions with boxes
- Recursion as loopback wires
- Box for each local-window operation
- Use templates to minimize code-bloat
- Need control variables for nested functions

## Canny: Structure of box-based version



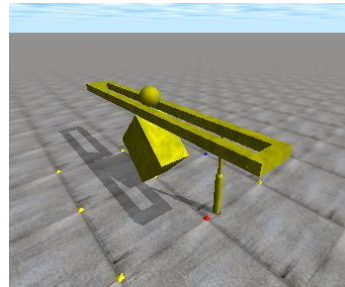
## Canny: Performance

Time/s(+)	Compiler	16x16	32x32	64x64	128x128	256x256
cannyv	hume	1.480	4.776	21.564	140.804	1157.711
	humec	0.068	0.132	0.236	1.208	N/A
cannyl	hume	2.920	12.764	83.377	635.335	N/A
	humec	0.108	0.308	1.192	6.596	N/A
cannyb	hume	6.612	24.553	110.070	427.822	N/A
	humec	0.472	2.332	32.298	638.399	N/A
cannyl size(*)		2x2	18x18	50x50	114x114	N/A

(\*) - size of the image output by cannyl.

(+) - times measured on a 1.80GHz Pentium M, 2048 KB cache.

## Ball and beam plus PID



- Developed B&B simulation using ODE's nano-simulator
  1. Connected to Hume program via sockets
  2. Ball position and beam angle are process variables
  3. Actuator force as control variable
  4. Implemented PID controller in Hume
- Very good demonstration of Hume's strengths

## PID: Equations

$$P_k = K_p * (e_k - e_{k-1})$$

$$I_k = K_i * T * e_k$$

$$D_k = (K_d/T) * (e_k - 2 * e_{k-1} + e_{k-2})$$

$$CV_k = CV_{k-1} + P_k + I_k + D_k$$

$$e_k = SP_k - PV_k$$

## PID: Hume code

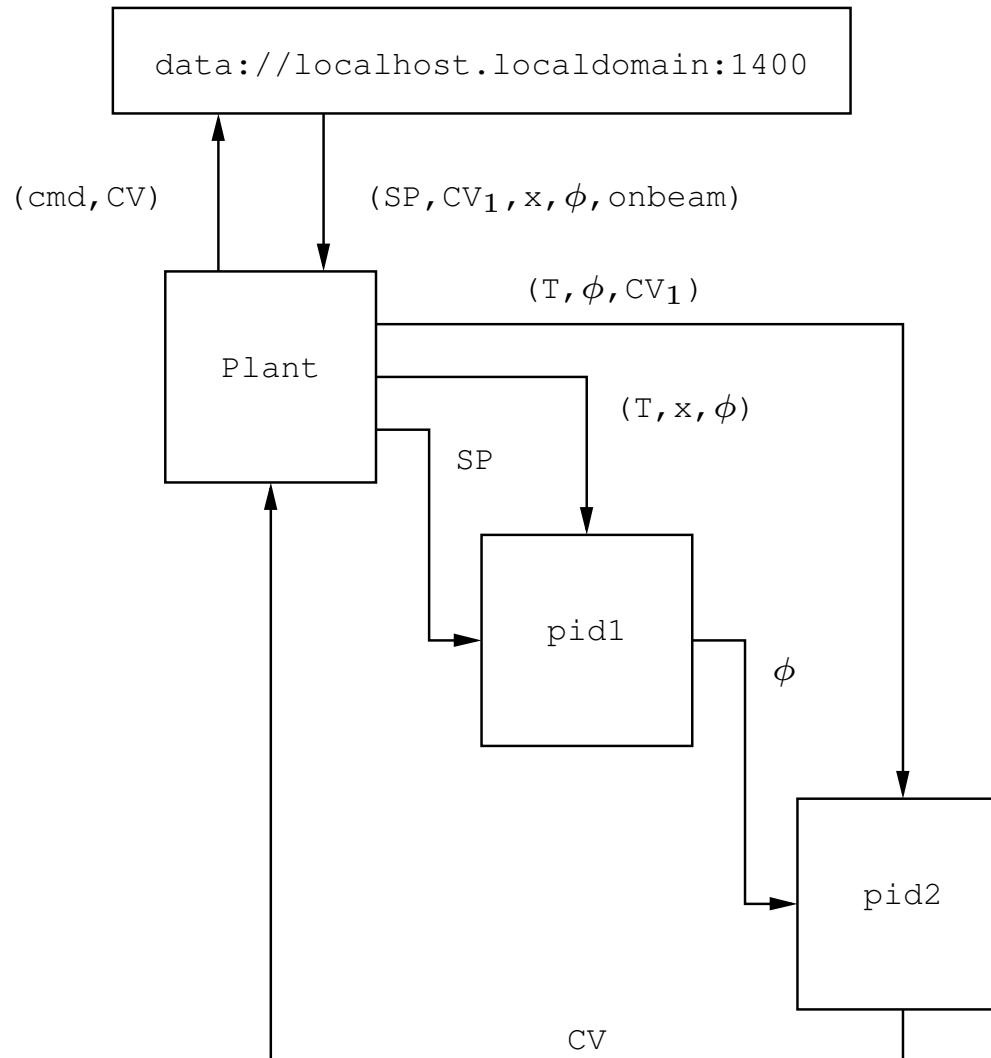
match

```

(c, (false, ek1, ek2, lc), (vSPk, vCVk1, vPVk, phi, onbeam)) ->
  (('x', vCVk1), *, (true, 0.0, 0.0, c))
| (c, (true, ek1, ek2, lc), (vSPk, vCVk1, vPVk, phi, 0)) ->
  (('2', vCVk1), (vSPk, vCVk1, vPVk, phi, 0, c), (true, 0.0, 0.0, c))
| (c, (true, ek1, ek2, lc), (vSPk, vCVk1, vPVk, phi, 1)) ->
  let vT = c -. lc in
  let ek = vPVk -. vSPk in
  let vPk = vKp *. (ek -. ek1) in
  let vIk = vKi *. vT *. ek in
  let vDk = ((vKd/.vT) *. (ek -. 2.0 *. ek1 +. ek2)) in
  let vCVk = vCVk1 +. limitPID(vPk +. vIk +. vDk) in
  (('x', vCVk), (vSPk, vCVk1, vPVk, phi, 1, vT), (true, ek, ek1, c));

```

# PID: Cascaded controller



## The CyCab project

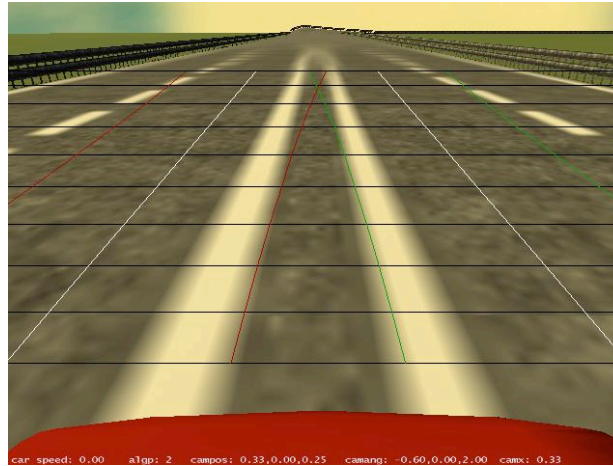


- Small but powerful electric vehicle
- Intended for “urban” transport applications
- Fitted with a range of cameras and rangefinders
- PowerPC platform with RTOS

## CyCab: Role in the EmBounded project

- Several CyCabs available at LASMEA
- Objective to drive CyCab using Hume
- Can target:
  - controller hardware
  - vision component
- Already written simulation in ODE

## CyCab: Simulation

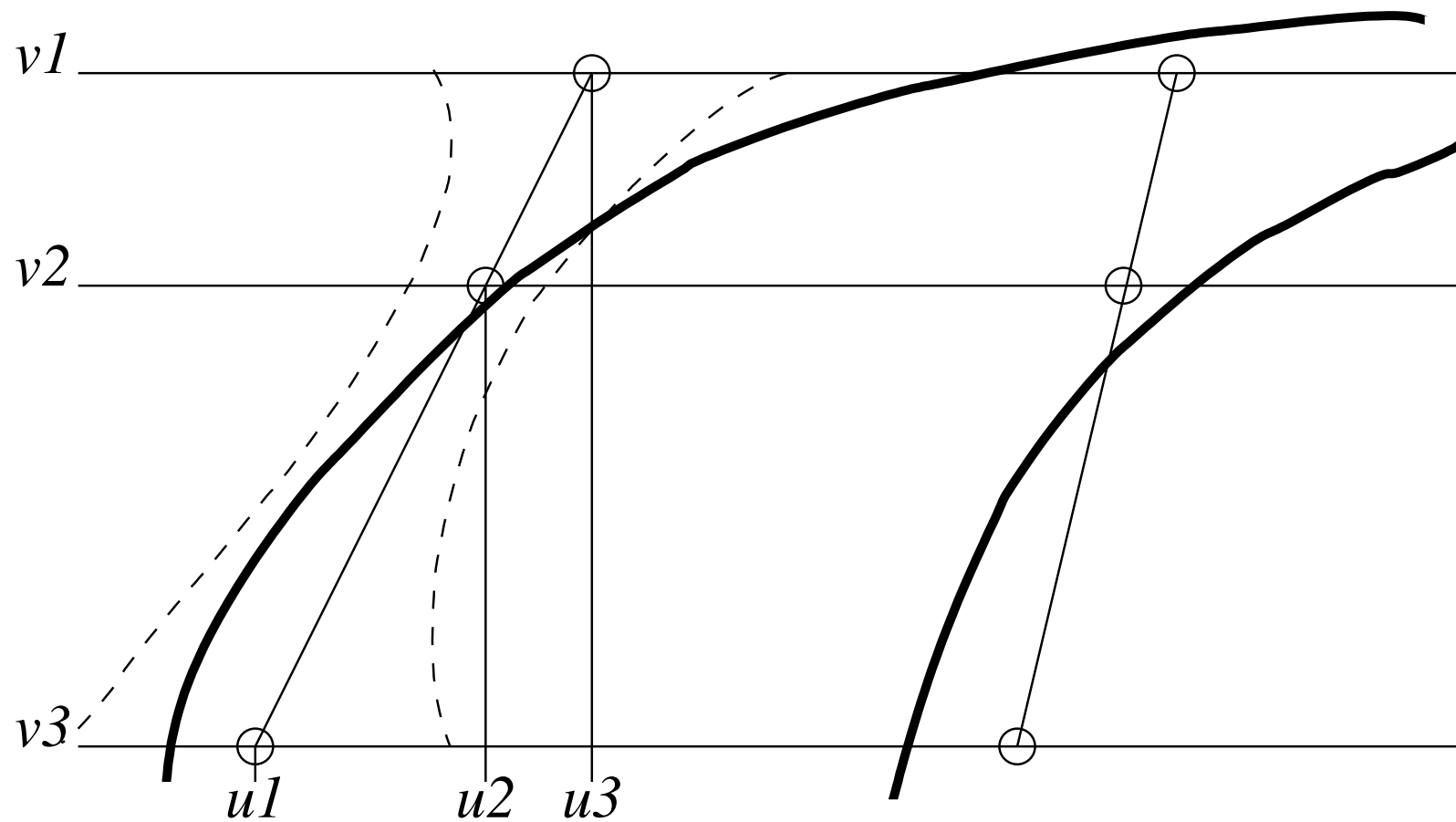


- Simulation of slow-moving vehicle using games engine
- Uses the *raydium* games engine (ODE, OpenGL, openAL, ...)
- Linked to Hume through socket interface

## CyCab: Vision component

- Several possibilities:
  - simple lane-tracking with Hough transform
  - model-driven lane-tracking (Aufrère et al.)
  - sensor fusion with Kalman filtering
  - Bayesian inference with particle filtering
- Won't have time for all of these

# Model-driven lane-tracking



- Control points plus confidence interval

## Lane tracking: Algorithm outline

- Model training (assumptions about camera etc.)
  - State vector  $(x_0, \psi, C_1, L, \alpha)$
- Control loop:
  - recognition (identify lanes in image)
  - localization (position of vehicle in model)
  - tracking (limit search space)

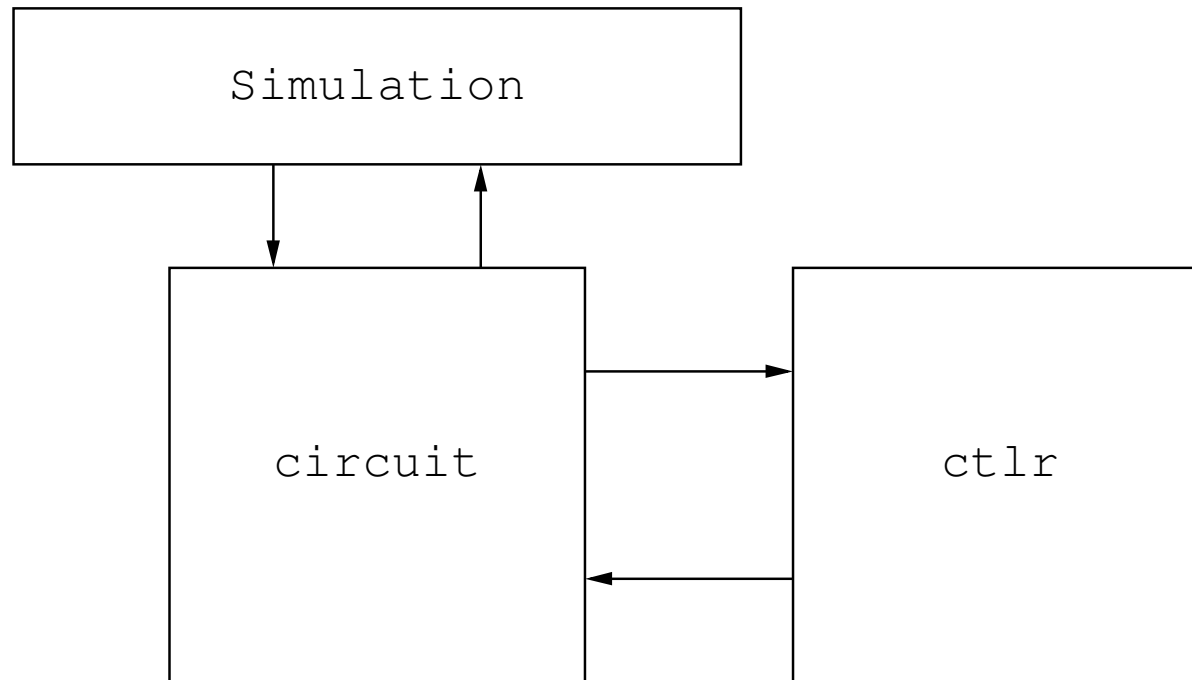
## Lane tracking: Recognition

- Compute interest zone:
  - $v_i$  with minimal variance within confidence interval
- Detection:
  - for each row in IZ: compute max. horiz. grad.
  - fit straight line to segment (LMS)
  - search from lowest to highest variance IZ
- Iterative, update model/covariances (linear algebra)
- Search until 10% of each edge detected

## Lane tracking: Hume implementation

- Simulation stores image data
- Auxiliary functions:
  - training phase
  - `update` process interest zone, update model
  - `localize` run localization algorithm
  - `track` tracking phase
- Foreign functions
  - LAPACK routines for matrix inversion
  - 2D array routines in interpreter

## Lane tracking: Box structure



- `circuit` handles protocol with simulation
- `ctrlr` runs main algorithm

## Lane tracking: Msg datatype

```
type is_t = list int 8;  
type _Xd_Cxd_t = (_Xd_t, _Cxd_t, _Xd_t, _Cxd_t);  
type model_t = (is_t, int 16, int 16, _Xd_Cxd_t);  
data Msg =  
  Start -- initial message from simulation  
| C_sample _Xl_t -- command to sample image data  
| Sample _Xl_t -- onward image sample command  
| Sampled -- image has been refreshed  
| C_model model_t -- update Xd and Cxd matrices  
| Model (v_t, u_t, _Xdc_t) -- model update command  
| Modeled -- model has been updated  
| C_siz int 8 -- scan interest zone number n  
| SIZ (sciz_t, int 8) -- onward IZ process command  
| IZ (iz_t, int 16, int 16, bool, lr) -- result of interest zone processing  
| Reset; -- abort processing
```

# Lane tracking: Circuit box

```

box circuit
  in (simin::Msg, ctrl_in::Msg, pp::PP, lvs::circuit_lvs_t)
  out (pp'::PP, lvs'::circuit_lvs_t, ctrl_out::Msg, simout::Msg)
match
  (Reset, _, _, _) -> (Ping, circuit_lvs_init, Reset, *)
| (Start, *, Ping, lvs) -> (Ping, lvs, *, Sample _Xl_init)
| (Sampled, *, Ping, lvs) -> (Pong, lvs, *, *)
| (Modeled, *, Ping, lvs) -> (Pong, lvs, *, *)
| (IZ pts, *, Ping, lvs) -> (Pong, lvs, IZ pts, *)
| (*, C_sample _Xl, Pong, lvs) -> (Ping, lvs, *, Sample _Xl)
| (*, C_model lvs, Pong, _) -> (Ping, lvs, *, get_model lvs)
| (*, C_siz n, Pong, lvs) -> (Ping, lvs, *, get_siz (lvs, n))
| (*, *, pp, lvs) -> (pp, lvs, *, *);

```

## Lane tracking: Performance

- Hume interpreter
  - Approximately 9 seconds per interest zone
  - About 83 seconds for one iteration
- Currently adapting for Hume compiler
  - Need to implement FFI calls
  - Expect about 100 speedup
  - Still slower than C (60-238ms)

## Conclusions

- Managed to implement low-level vision algorithms
- Very suitable for small control applications
- Qualified success with larger applications:
  - Coordination with boxes easy
  - Dependent upon foreign functions
  - Vision is I/O-bound
- Now looking at Renesas M32C microcontroller
- Analysis tools beginning to mature