Improving Time Estimates in VDM-RT Models

13th Overture Workshop 2015

Presentation by Morten Larsen

Authors: Morten Larsen, Peter W. V. Tran-Jørgensen, and Peter Gorm Larsen

Agenda

- Motivation
- Overture extensions
- Case Study
- Results
- Discussion
- Summary
- Future work
- Special slide

Motivation

- Design of agricultural robotic systems
  - Computational intensive processing of sensor data (LiDaR, GNSS, Camera, etc.)
  - Multiple control loops which may have deadline requirements
  - Distributed nature of robotic system design
    - Reuse of components in different applications

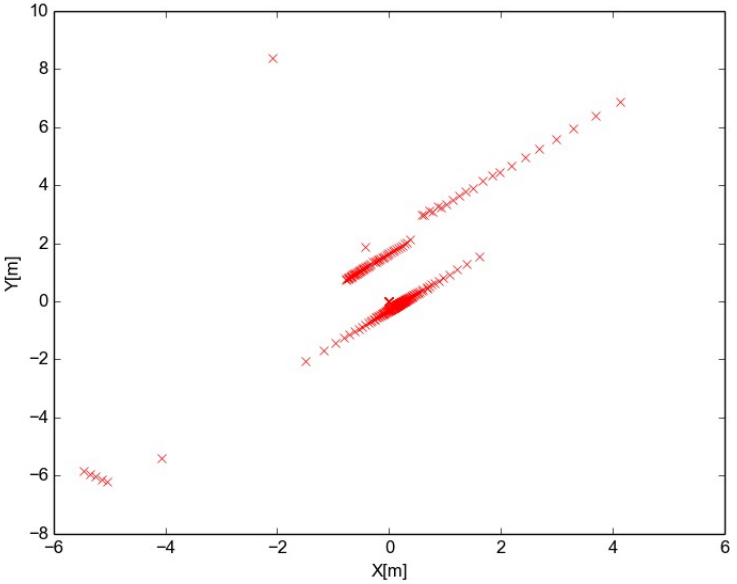
Motivation

- Autonomous mink farm robot
  - Automatic feeding of mink

Source:http://www.minkpapir.dk/

Motivation

- Design row detection algorithm based on LiDaR



Design questions

- Can we switch to a cheaper, lower power platform
- What is the performance of the selected algorithm
- What happens to the performance if we lower the quality of the detected rows

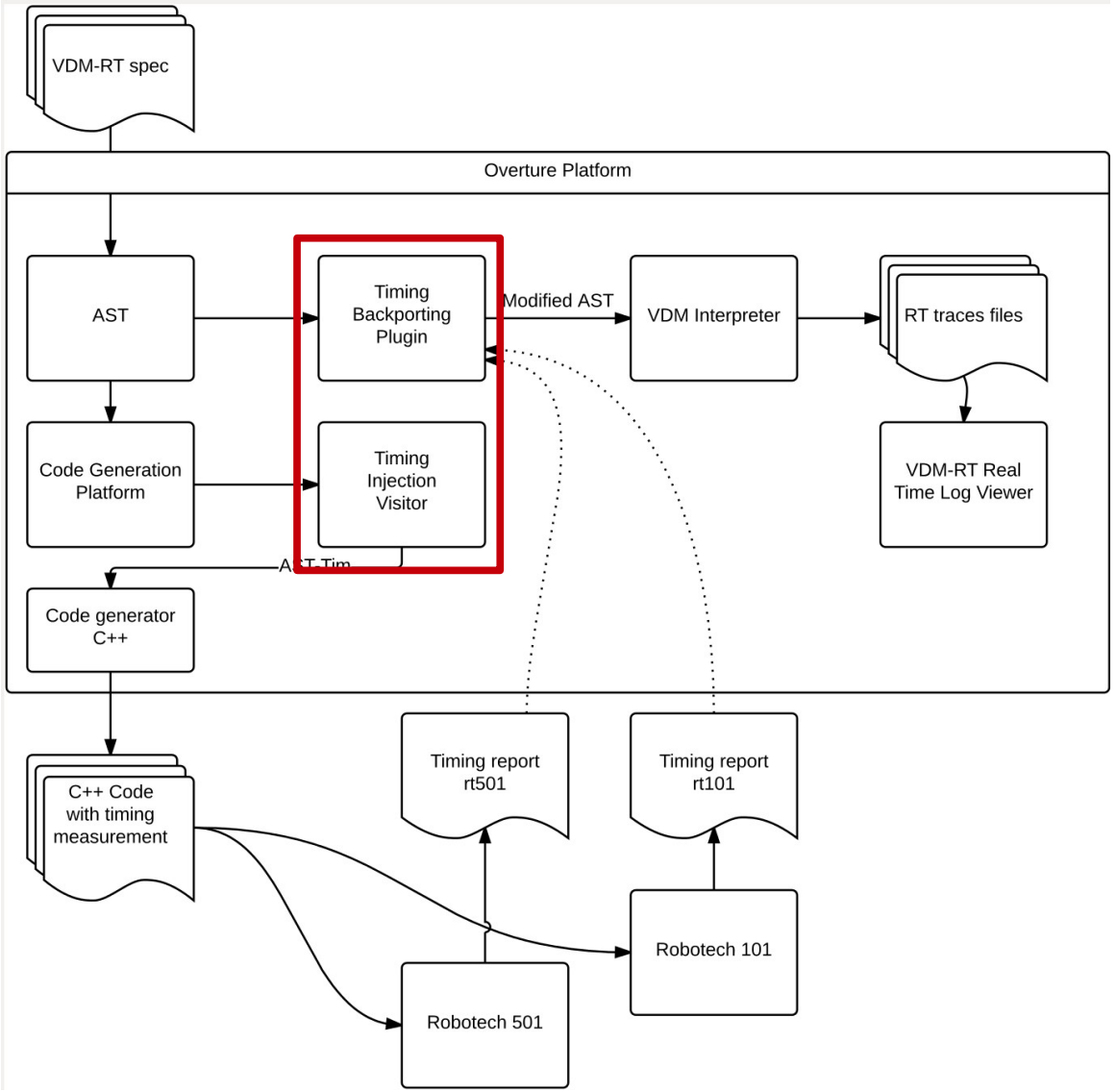
Answering the questions

- We can guesstimate using prior knowledge
- In Overture we can model execution time using **duration** and **cycles**
- No explicit support for switching between multiple platforms

Extensions overview

- Extend overture
  - Obtain timing measurements from real platforms
  - Incorporating these measurements into the model

Extensions overview



Extensions summary

- Many different ways to obtain execution time information from code
  - Static analysis
  - Measurement
  - Simulation
- The incorporation of timing information into the model does not depend on a specific method used for obtaining the information.

Case Study

- Row detection algorithm experiments on two different hardware platforms

Conpleks Robotech 101
I.mx6 Quad Arm @ 1GHz

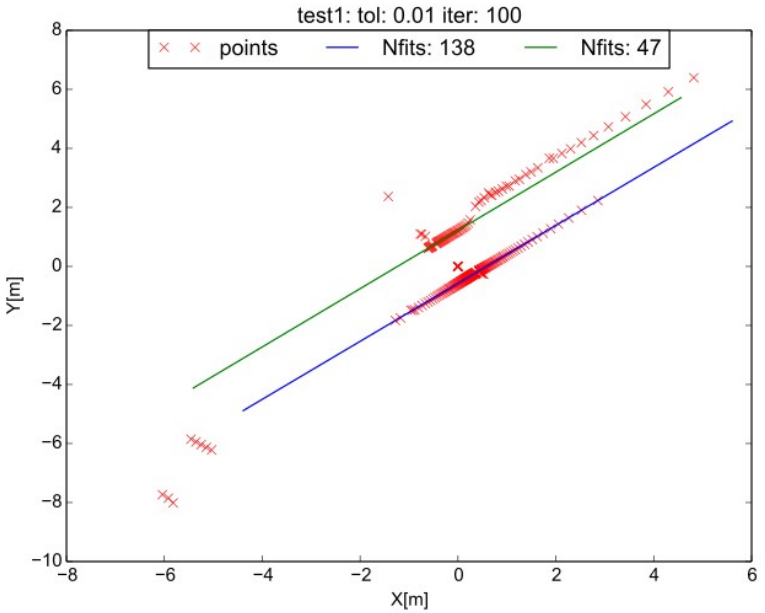
Conpleks Robotech 501
Intel I5 Dual core @ 2.8GHz

Row detection algorithm

- RANSAC based row dectection
  - Random Sample Consensus
  - Pick a sample from the data and construct model
    - For a line model we pick two points at random (p1 != p2)
  - Construct a set of inliers and outliers based on distance to the line
  - Rinse and repeat storing the score for each sampled line

Row detection algorithm

- Final detection algorithm detecting multiple lines
  - Run RANSAC on the data, then remove the inliers of the result and repeat on the reduced data set.



Results

- Corrections for the numbers presented in paper
  - Difference between 2.8GHz and 1GHz is 2.64 on average not 2.8
  - Table 3 is wrong, should have been

Operation	Mean	Median	Min	Max	stddev
getRows	$9.5ms$	$8.3ms$	$6.9ms$	$14.9ms$	$2.3ms$
extractLines	$2.4ms$	$2.1ms$	$253.5\mu s$	$4.5ms$	$1.4ms$
getInliers	$23.6\mu s$	$20.8\mu s$	$2.3\mu s$	$44.8\mu s$	$14.4\mu s$
getRandomLine	$187.2ns$	$184.0ns$	$184.0ns$	$910.0ns$	$19.1ns$
addNewBestFit	$30.0ns$	$30.0ns$	$30.0ns$	$30.0ns$	0

Results

Device	Mean	Std dev
CPU 2.8 GHz	3.6 ms	866 us
CPU 1 GHz	9.5 ms	2.3 ms
RT501	45.9 ms	3.9 ms
RT101	422.2 ms	34.2 ms
RT501-backport	41.4 ms	59.1 us
RT101-backport	376.3 ms	88.1 us

Results

- How well can we predict the execution time when a parameter is changed

Device	Mean	Stddev
RT501	24.2 ms	2.5 ms
RT501-backport	20.8 ms	8.5 us

Discussion

- Timing on the operation/function level makes it difficult to capture variability
- If a parameter is changed the measurements has to be redone in most cases
- The 16.5% accuracy is only for the very specific test done, and is not in any way general.

Summary

- We could make some predictions on the execution time of the row detection algorithm
  - The difference between the two hardware platforms was intially thought to be 2.64 but the measurements showed approx 10 times in difference
- However limited use due to only using mean operation/function execution time

Future work

- We propose to create a benchmark model
  - Can be used to bench mark a given hardware platform
  - The benchmark results can be used by the VDM-RT interpreter for any model

Last Slide

- 1 year
  - Code generation for embedded platforms (C/C++)
    - Bare bone OS
    - RTOS (FreeRTOS)
    - Linux + xenomai/RTAI/RTLINUX
- 5 year
  - Industrial strength libraries with code-generation support
  - Models of Ethernet CAN, Skynet with code-generation support
  - Faster interpreter (JIT?)
  - Model management integrated
- 10 year
  - 100.000 downloads (eclipse IDE for C++ has 600.000+ downloads)