WHY AND HOW TO ACHIEVE DETERMINISTIC TIMING WITH ROS WITH EXAMPLES FROM THE NAVSTACK

ROS-INDUSTRIAL CONFERENCE
3RD NOV 2016

Ingo Lütkebohle (Bosch CR/AE Software Platforms and Technologies)
About Bosch

4 Product Areas

Mobility Solutions

Industrial Technology

Energy and Building Technology

Consumer Goods
About Bosch

Corporate Research / Advance Engineering
About Bosch
(Some of) our Robots
(How)
Can we use ROS for that?
Recap
Bosch CR’s view on ROS

- ROS1 strengths...
  - ...excellent for getting started quickly
  - ...has a lot of components available
  - ...great for working with academia
  - ...offers best-in-class tool support (recording, visualization)

- ROS1 weaknesses
  - ...composition and checking is lot of manual effort
  - ...component quality varies widely
    - many components don’t work outside of very narrow use-cases
  - ...real-time support is very limited
  - ...predictability of system behavior is extremely limited

Limited scalability
No safety support

That’s what we are working on 😊
Agenda

1. Motivation: Predictable execution in an asynchronous system
2. Timing Analysis as a tool to measure and achieve deterministic execution
3. Example: Improvements to ROS move_base
4. Outlook and Discussion
Why we need timing analysis
Basic obstacle avoidance pipeline

- In other words: Get sensor data, do some processing, act
- Real-world issue
  - How long from obstacle sensing to drive stop?
  - And, can we be sure this is always the case?
- Timing view
  - What’s the response time?
  - Is the behavior (and hence response time) deterministic? (“does it always do the same steps in the same order?”)
Why we need timing analysis

Response time experiment

- Three nodes: Laser, move_base, base driver
- No direct info on end-to-end timing available
- Initial analysis: move_base planning stage is good indicator
- Questions
  - How long does the move_base take to plan?
  - How old is the sensor data used for planning?
    (aka, when was the costmap last updated)
Why we need timing analysis

Summary of current situation

- Controller runs at configured rate (e.g., 10Hz)
  - But it often uses old data
- Data age has behavioral effect
  - Little change when map is known and static – assuming odometry is current
  - But delayed reaction to dynamic obstacles can lead to collisions
  - Semi-static obstacles in occluded regions can also be affected
- Varying behavior is problematic for testing
  - Sometimes it works, sometimes it doesn’t...
Objectives

We want to find out *why* this happens

- Could be due to...
  - Differences in computation time from one run to the next
  - Differences in system load during execution
  - Different movement speeds
  - Some change in the environment
  - Data fusion/windowing
  - Sampling between runnables with different rates

- ROS has great tools, but they only look at the interface, not inside
Our Approach
Instrument, analyze, refactor

- **Measure**
  - Framework timing instrumentation – available out-of-the-box
  - Support for custom instrumentation – user-defined, integrated with above

- **Analyze**
  - Measurement-based response time analysis
    - Integrated analysis of generic and custom tracepoints

- **Refactor**
  - Information to guide architectural changes
Our Approach

Generic tracepoints in roscpp
move_base’s internal pipeline

Recap: Basic processing stage

- Receive LaserScan
- Receive Pose
- Merge Sensor & Pose
- Mark occupied and clear space
- Perform local planning
- Output velocity command

Move_base realization: 4 threads, not synchronized

<table>
<thead>
<tr>
<th>Thread</th>
<th>Trigger</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>spin thread</td>
<td>D&amp; T</td>
<td>Receive LaserScan</td>
</tr>
<tr>
<td>TF thread</td>
<td>D</td>
<td>Receive Pose</td>
</tr>
<tr>
<td>costmap update thread</td>
<td>T</td>
<td>Mark occupied and clear space</td>
</tr>
<tr>
<td>move_base thread</td>
<td>T</td>
<td>Perform local planning</td>
</tr>
<tr>
<td>output velocity command</td>
<td></td>
<td>Output velocity command</td>
</tr>
</tbody>
</table>
Case Study
Observations

- Perception and planning are not synchronized
- They run at different rates
  - Laser at 12.5Hz (SICK LMS300)
  - Costmap update at 5Hz
    - And further delayed by TF, until pose for sensor timestamp is available
  - Planner at 10Hz
Our Approach

Custom Tracepoints in move_base
Case Study

Procedure

- Run `move_base` in various configurations with tracing enabled
  1. Standard configuration (as shown before)
  2. Configuration where processing runs at sensor-rate
  3. Refactor as necessary
- Compare response times
Case Study Results

Part 1: Default behavior

Not just delays, but also unpredictable change in execution order
Observation: Overall lower response time, but very jittery

Cause: Slight differences in activation cause execution re-ordering
Case Study Extension
Analysis and Refactoring

▫ We see significant processing jitter
  – This is typical for asynchronous processing
▫ Refactoring to make move_base processing synchronous
  – Map update step and planning executed sequentially
  – Planning (incl. update) invoked triggered by sensor data reception
Case Study Results

Refactored timings, comparison

before

after
Case Study Results

Timing Boxplots

- Drastically reduced mean response time (mean 85ms → mean 9ms)
- Huge jitter reduction (60ms → 5ms std)
- Bounded max delay
Deterministic Timing for ROS

Summary

- Deterministic behavior is not automatic for component-based systems
  - Particularly asynchronous, periodic execution is problematic
- Measurement tools need to look inside the framework and the app
- We have found issues with move_base that have been there for years
  - Performed refactoring with minimal changes
    - Still time-triggered, but activation synchronized
    - Response time reduced almost to pure computation time
- Conclusions
  - Principled measurement and reasoning about execution ordering is essential for robustness (and performance!)
  - Timing analysis can provide this
  - Let’s talk about better built-in support!

Ingo.Luetkebohle@de.bosch.com