Linux-based RTOS Experimental Platform for Constructing Self-driving Vehicles

Jim Huang & ShaoHua Wang,
National Cheng Kung University, Taiwan
Oct 23, 2018
Quote from Intel’s keynote (Oct 22):
Car: the data center on wheels

Arjan van de Ven,
“Software-Defined Everything”
NVIDIA said similar words 4 years ago

"Systems do not have to be re-created from scratch each time, a process which traditionally took five to seven years in the automotive world. Now we are enabling car makers to update the hardware in new cars each model year. Our visual computing module system is as easy as sliding in a new component and updating the software."

-- Danny Shapiro, senior automotive director at Nvidia

New processors are enabling more sustainable design and remote upgrades that could make cars smarter, more efficient and longer lasting. Is the end of auto obsolescence near?

source: https://www.theguardian.com/sustainable-business/nvidia-processors-auto-tech-design-upgrades
The goals of this presentation

- General overview about constructing autonomous vehicle (AV)
  - build from toys to real cars
- Linux adoption with open source stack
- Highlight drawbacks of existing stack and how we enhance in recent prototypes
Sensing is crucial

check video
Components enabling Automated Driving
LiDAR (Light Detection and Ranging) is highly accurate and so can be used as main sensor. It is used for mapping, localization and obstacle avoidance.
Components enabling Automated Driving

Cameras are used for object detection and object tracking. 8+ HD cameras can be installed in AV generating around 1.8 GB of raw data per second.
Components enabling Automated Driving

Radar and Sonar can be used as back up system to avoid collision. They detect the nearest object in front of the vehicle’s path and apply brakes or turn to avoid obstacle and even pre-tension the seat belts.
Components enabling Automated Driving

For best localization results the advantages of GPS and IMU are to be combined and for this Kalman Filter is used.
Reasons why we can not only rely on GPS

• Do not only rely on GPS + IMU combination for localization:
  – Accuracy of about one meter
  – GPS signals can be bounced off by buildings, adding noise
  – GPS requires unobstructed view of the sky, thus cannot work in tunnels
Cameras for Localization

- Triangulating stereo image pairs generates disparity maps giving information of depth for each point.
- By matching salient features between successive image frames, correlation can be established between points in different frames and motion can be estimated.
- Comparing it with the known map, the current position of the vehicle can be determined.

It is very sensitive to lighting conditions.
LiDAR for Localization

- Points clouds generated by LiDAR provides shape description and then it compared with the known map to reduce uncertainty.
- To localize a moving vehicle, a particle filter method is used to correlate the LiDAR measurements with the map and accuracy up to 10 cm is obtained.

When there are many suspended particles in the air like rain drops or dust, the measurement can be extremely noisy.
System Software Architecture

1. Linux + PREEMPT_RT with Realtime extension

2. ROS + Autoware [BSD 3-Clause]
Duckietown: an Open, Inexpensive and Flexible Platform for Autonomy Education and Research

Liam Paull, Jacopo Tani, Heejin Ahn, Javier Alonso-Mora, Luca Carbone, Michal Cap, Yu Fan Chen, Changhyun Choi, Jeff Dusek, Yajun Fang, Daniel Hoehener, Shih-Yuan Liu, Michael Novitzky, Igor Franzoni Okuyama, Jason Pazis, Guy Rosman, Valerio Varricchio, Hsueh-Cheng Wang, Dmitry Yershov, Hang Zhao, Michael Benjamin, Christopher Carr, Maria Zuber, Sertac Karaman, Emilio Frazzoli, Domitilla Del Vecchio, Daniela Rus, Jonathan How, John Leonard, Andrea Censi

Abstract—Duckietown is an open, inexpensive and flexible platform for autonomy education and research. The platform comprises small autonomous vehicles (“Duckiebots”) built from off-the-shelf components, and cities (“Duckietowns”) complete with roads, signage, traffic lights, obstacles, and citizens (duckies) in need of transportation. The Duckietown platform offers a wide range of functionalities at a low cost. Duckiebots sense the world with only one monocular camera and perform all processing onboard with a Raspberry Pi 2 yet are able to: follow lanes while avoiding obstacles, pedestrians (duckies) and other Duckiebots, localize within a global map, navigate a city, and coordinate with other Duckiebots to avoid collisions. Duckietown is a useful tool since educators and researchers can save money and time by not having to develop all of the necessary supporting infrastructure and capabilities. All materials are available as open source, and the hope is that others in the community will adopt the platform for education and research.

Fig. 1. In Duckietown, inhabitants (duckies) are transported via an autonomous mobility service (Duckiebots). Duckietown is designed to be inexpensive and modular, yet still enable many of the research and educational opportunities of a full-scale self-driving car platform.
MIT Duckietown

https://www.duckietown.org/
• Complete (scale) systems: Raspberry Pi
• Software: Python, ROS
• Focus: AI & Machine learning, TensorFlow, OpenCV / R
• Purpose: lectures
• Other resources: DIYrobocars.com
• Hardware
  – LIDAR:, LeddarTech, XenoMatix, Innoviz and Scanse
  – Control systems: PolySync and Comma Neo
  – Sensor fusion: NXP and others
Puyuma: Enhanced prototype w/ RT Capability

Puyuma: Linux-based RTOS Experimental Platform for Constructing Self-Driving Miniature Vehicles

Shao-Hua Wang†, Sheng-Wen Cheng*, Ching-Chun (Jim) Huang†
†Department of Computer Science and Information Engineering, Providence University, Taiwan
‡Department of Computer Science and Information Engineering, National Cheng Kung University, Taiwan

Abstract—A holistic design and cost-efficient platform to construct self-driving systems is presented with an emphasis on Linux-based software architectures for computer vision, control system, and inter-vehicle communication. Starting with an executable specification of autonomous car application, subsequent transformations are performed across different levels of abstraction until the final implementation is achieved. The software partitioning is facilitated through the integration of ROS and OpenCV in the same design environment, as well as closed-loop control algorithms and Linux in the run-time system. We built a rapid prototyping based on fundamentally open source technologies and hardware under 100 dollars USD, which allows developers to be explored and evaluated in realistic conditions efficiently. Using lane departure and the corresponding performance speedup, we show that our platform reduces the design time, while improving the verification efforts, with the aid of tweaked real-time executives.

Keywords—Real-time; self-driving; ROS; BYOD

AUTOSAR (AUTomotive Open System ARchitecture) is a standardized architecture for automotive software systems. Its primary focus is the system development process for electronic control units (ECU) and does not provide specific support for sensor/actuator-based autonomous system development [7]. An implementation of an AUTOSAR-compliant embedded software platform is Arctic Core [8], offering a real-time operating system, memory services, and communication services such as CAN and LIN.
Puyuma: Enhanced prototype w/ RT Capability

- Full source code available: https://github.com/puyuma
- Architecture of a self-driving miniature vehicle based entirely on cost-efficient components
- Context switching performance is important because the number of sensors can be high and we need to read data from different sensors continuously in real-time.
- Linux with real-time extension fulfills all these criteria e.g. performance, reusability, changeability, and maintainability, which have significant value from the software engineer’s perspective.
Preemptive Kernel

- Controlling latency by allowing kernel to be preemptible everywhere
- Increase responsibility; decrease throughput

- preemption: the ability to interrupt tasks at many “preemption points”
- The longer the non-interruptible program units are, the longer is the waiting time of a higher priority task before it can be started or resumed.
- PREEMPT_RT makes system calls preemptible as well

Source: Understanding the Latest Open-Source Implementations of Real-Time Linux for Embedded Processors, Michael Roeder, Future Electronics
Preemption is not allowed in Kernel Mode.

Preemption could happen upon returning to user space.
PREEMPT_VOLUNTARY

Insert explicit preemption point in Kernel: `might_sleep`
Kernel can be preempted only at preemption point

CONFIG_PREEMPT

- Implicit preemption in Kernel
- `preempt_count`
  - Member of thread_info
  - Preemption could happen when `preempt_count == 0`
PREEMPT_RT_FULL
Threaded Interrupts

Reduce non-preemptible cases in kernel: spin_lock, interrupt
Measurements on Real-Time

- **Clocksource and High Resolution Timer**
  Accuracy of timer in Linux depends on the accuracy of hardware and software interrupts. Timer interrupts are not occurring accurately when the system is overloaded. It would cause timer latency in kernel.

- **Task switching cost**
  Process switching cost is significantly larger than thread switching. Process switching needs to flush TLB. If RT application consists of lots of processes, process switching measurement is necessary.

- **Page faults**
  Initial memory access causes page fault, and this causes more latency. Page-out to swap area also causes page faults. Use mlockall and custom memory allocators.

- **Multi-core**
  Task migration from local to remote core causes additional latency. It can be fixed to a specific core by cpuset cgroup.

- **Locks**
  spin_locks are now mutexes, which can sleep. spin_locks must not be in atomic paths. That is, preempt_disable / local_irq_save. RT mutex uses priority inheritance, and no more futexes. cost gets higher.

Further reading: “Effectively Measure and Reduce Kernel Latencies for Real-time Constraints”, ELC 2017
Memory problems in typical real-time

- Dynamic memory allocation is used along with
  - A lot of objects, which are referenced by different threads.
  - Their number and lifetime is unpredictable, therefore they should be allocated and deallocated dynamically.

- Heap operations are in conflict with the main demand of real-time systems
  - Operations in high priority threads must be deterministic.
Recent RT Considerations on Memory

- **Many-Core Era**
  - Computers with tens of cores are available
    - Even valid for mobile phones:
    - e.g. MediaTek helio P10 >> 2 GHz, Octa-core 64-bit ARMv8-A CPU

- **Many-Thread Application**
  - Not only server programs but also desktop applications
    - Even mobile Apps, which consist of various background tasks

- **Many Applications’ performance heavily relies on memory allocator**, such as modern C++ framework
Improvements over low-level allocator

- **Scalability**
  - Synchronization primitive-free critical path
  - Local memory reuse
  - Lock-free global data structure
  - Excessive VM management calls avoidance (mmap, munmap)

- **Latency**
  - Wait-free algorithm within private heap
  - Shorten critical path

- **Locality**
  - Locality-conscious memory chunk management
  - Allocator false-sharing avoidance

Further reading: “Deterministic Memory Allocation for Mission-Critical Linux“, OSS-EU 2017
How autonomous vehicles (AV) work w/ open source stack
How AV Works

- Focus on **ROS navigation stack** package in this presentation.
- There are 3 phases in ROS navigation stack: **Sensing, Computing, Actuation**.
- AV needs to know “*where am I?*”, “*when to go?*” and “*how to go?*”.

![Diagram of AV navigation stack]

- **Sensing**
  - LiDAR
  - IMU

- **Computing**
  - Perception
    - Localization
    - Detection
  - Planning
    - Goal
    - Motion
  - Decision
    - State

- **Actuation**

---

Robot Operating System (ROS)

Linux OS (e.g. Ubuntu 16.04)
How AV Works

• ROS navigation stack uses **natural features navigation**, without retrofitting of the workspace to guide AV.

• Key sensor on natural features navigation is 2D-LiDAR or 3D-LiDAR.

• **LiDAR** = Light Detection and Ranging, measures distance to a target by illuminating the target with pulsed laser light and measuring the reflected pulses with a sensor.


SLAM

- **Simultaneous Localization And Mapping (SLAM)** is **prerequisite** for navigation in an unknown environment. LiDAR is used to generate maps.

Source: Shikai Chen, SLAM based on RPLIDAR and ROS Hector Mapping. Retrieved Oct 10, 2018, from [https://www.youtube.com/watch?v=pCF7P7u8pDk](https://www.youtube.com/watch?v=pCF7P7u8pDk)
Perception: Detection

- To know *where am I*, AV needs detection and localization information.
- AV compares the map produced from **SLAM** with the real-time **LiDAR** sensor data to detect where the obstacles are.

Source: pirobotproductions, SLAM 1: Testing ROS with the gmapping package, Retrieved Oct 11, 2018, from https://www.youtube.com/watch?v=khSrWtB0Xik
Perception: Localization

- Getting the map and perceiving the surrounding is not enough, AV needs to locate itself in the map.
- **Adaptive Monte Carlo Localization** (AMCL) uses AV’s motion and sensor data to compute the **probability distribution** of new location in every iteration.
- One iteration contains 3 actions: **sensor update, resampling, motion update**.
Perception: Localization (cont’d)

- AMCL initializes with a **uniform distribution** of particles. AV considers itself equally to be any point in the space.
Perception: Localization (cont’d)

Sensor Update
Using LiDAR sensor data to compute every particles’ weight.

Resampling
According to particle’s weight, AMCL generates new distribution of particles.

Motion update
When AV moves, each particle also follows the same movement.
Perception: Localization (cont’d)

Sensor Update
Using LiDAR sensor data to compute every particles’ weight.

Resampling
According to particle’s weight, AMCL generates new distribution of particles.

Motion update
When AV moves, each particle also follows the same movement.
Sensor Update
Using LiDAR sensor data to compute every particles’ weight.

Resampling
According to particle’s weight, AMCL generates new distribution of particles.

Motion update
When AV moves, each particle also follows the same movement.
Perception: Localization (cont’d)

**Sensor Update**
Using LiDAR sensor data to compute every particles’ weight.

**Resampling**
According to particle’s weight, AMCL generates new distribution of particles.

**Motion update**
When AV moves, each particle also follows the same movement.
Perception: Localization (cont’d)

Sensor Update
Using LiDAR sensor data to compute every particles’ weight.

Resampling
According to particle’s weight, AMCL generates new distribution of particles.

Motion update
When AV moves, each particle also follows the same movement.
Perception: Localization (cont’d)

Sensor Update
Using LiDAR sensor data to compute every particles’ weight.

Resampling
According to particle’s weight, AMCL generates new distribution of particles.

Motion update
When AV moves, each particle also follows the same movement.
Perception: Localization (cont’d)

Sensor Update
Using LiDAR sensor data to compute every particles’ weight.

Resampling
According to particle’s weight, AMCL generates new distribution of particles.

Motion update
When AV moves, each particle also follows the same movement.
Perception: Localization (cont’d)

Sensor Update
Using LiDAR sensor data to compute every particles’ weight.

Resampling
According to particle’s weight, AMCL generates new distribution of particles.

Motion update
When AV moves, each particle also follows the same movement.
Perception: Localization (cont’d)

**Sensor Update**
Using LiDAR sensor data to compute every particles’ weight.

**Resampling**
According to particle’s weight, AMCL generates new distribution of particles.

**Motion update**
When AV moves, each particle also follows the same movement.
Perception: Localization (cont’d)

Sensor Update
Using LiDAR sensor data to compute every particles’ weight.

Resampling
According to particle’s weight, AMCL generates new distribution of particles.

Motion update
When AV moves, each particle also follows the same movement.
Perception: Localization (cont’d)

- **AMCL** initializes with a **uniform distribution** of particles in the space.
- **Sensor update**: Using LiDAR sensor data to compute every particles’ weight.
- **Resampling**: According to particle’s weight, AMCL generates new distribution of particles.
- **Motion update**: When AV moves, each particle also follows the same movement, then go back **sensor update** to finish one iteration.

Source: Derek Wonnacott, Pioneer AMCL Pose Estimate 2, Retrieved Oct 11, 2018, from [https://www.youtube.com/watch?v=1BGj09VrJOU](https://www.youtube.com/watch?v=1BGj09VrJOU)
To know \textit{when to go}, AV considers its situation to make decision when it receives the information from perception stage.

Once situation is normal, AV goes into planning stage to start mission.
In ROS navigation stack, AV has three states in one control loop.

<table>
<thead>
<tr>
<th>State</th>
<th>Behavior</th>
<th>Situation</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLANNING</td>
<td>Plan a new path to the goal</td>
<td>• Initial state&lt;br&gt;• Path is stuck</td>
</tr>
<tr>
<td>CONTROLLING</td>
<td>Keep following the path to goal</td>
<td>• The most common state&lt;br&gt;• AV works normally</td>
</tr>
<tr>
<td>CLEARING</td>
<td>Launch specific function to troubleshoot&lt;br&gt;(e.g. stop, rotate or retreat)</td>
<td>• Unknown situation&lt;br&gt;• No way to go&lt;br&gt;• Emergency happened</td>
</tr>
</tbody>
</table>
Planning: Global Path Planner

- When the mission arrives, how AV reaches goal with perception data? (How to go?)
- The red line is global path which indicates the whole trajectory to the goal.
- The Dijkstra’s and A* is the most famous path finding algorithms.
• Only considering the shortest path (global path) is not enough, because the dynamic obstacle may stuck global path during navigation.

• **Dynamic Window Approach (DWA)** computes different velocity and angular velocity pairs for predicting the trajectories.

• DWA **scores** every sampled trajectories based on distance to the global path, moving speed and obstacle distance. According to scores, AV chooses the best trajectory.
AV w/ ROS Navigation Stack

- **move_base** package receives the sensor data for computing. It also contains the two key features: the **global (A*)** and **local path planner (DWA)** as mentioned above.

- **costmap2d** package takes in sensor data from the world, builds a 2D/3D occupancy grid of the data.

- **amcl** package handles the AV’s localization.
Enhancements against navigation stack
Revise Resource Usage w/ ROS

- The local path planner in `move_base` and `amcl` package in ROS are **serial programming**. It may lead into poor CPU usage in multi-core device.

- Before improve the ROS navigation stack, we need to find out where the hotspot is.

- The profiling tools (e.g. `perf`, `valgrind`, `Intel VTune`) help the developer analyze the system to figure out what the bottleneck is.

- Once the bottleneck found, we leverage the ROS navigation stack with the **multi-threading** and **heterogeneous computing** technique.
Evaluation: Environment

- We prepared one Intel server platform and one ARM embedded platform.
  - The CPU in Intel server platform is Intel Xeon E5-2650 with **48 cores**.
  - The server has sufficient core number to analyze performance up to 16 threads.
  - The CPU in ARM embedded platform is Dual Denver + Quad ARM A57 with **6 cores**.
  - We compared the ARM embedded performance with Intel server.

- **Gazebo** is the famous robot simulator builds the vehicle, sensor and environment based on its robust physics engine.
Evaluation: Profiling

- As the mentioned above, one control loop sends a control command in that moment. One trajectory needs huge amount of control loop to travel.
- We consider **control frequency**, control loops per second, as the performance index.
- The lower moving distance, which is **moving speed * one control loop time**, means the AV has more time to make decision when emergency happens.
Evaluation: Profiling (system-wide)

- Indicated by `perf` system-wide profiler, `move_base` and `amcl` occupied about 69% and 22% respectively. These two packages significantly affect the ROS navigation stack performance.
- We zoom into what leads heavy workload in these two packages.
- Classify the all algorithm by task, the AV algorithm take about 96% CPU workload in whole system.
Evaluation - Profiling (amcl)

- CPU cycles were occupied by amcl packages when **amcl particles number** increased.
- The profiling runs on Intel Xeno E5-2650 by **perf**.
- The amcl package profiled with 800 sampled paths and 12Hz control frequency.

![AMCL Workload with 12Hz Control Freq.](image-url)
Evaluation - Profiling (move_base)

- The move_base packages workload rise when local planner sampled trajectories increased.
- When AV can’t meet the asked control frequency, it will run as soon as possible for each control loop.
Evaluation - Experiment

- The sampled paths by **DWA local path planner** is **data independency**.
Evaluation - Experiment Result (Intel Platform)

- The experiment achieves **7.98x** speedup at Intel platform with 16 threads.
- The local path planner samples 8000 trajectories toward to get correct path.
The experiment achieves $2.61x$ speed up at ARM platform with 6 threads.

The AMR platform’s CPU is not pure symmetric, it leads to interference in parallelism.

The local path planner samples 4000 trajectories toward to get correct path.
Evaluation - Experiment Result (Intel vs ARM)

- In this experiment, the ARM embedded platform with 2 threads reach the same performance as Intel server platform at serial.
- The ARM with 6 threads is the same as Intel server with 2 threads.
Performance Projection: AMCL on GPU

- AMCL uses Monte Carlo method to orientate, and it is ideal to apply **heterogeneous computing**.
- Performance gains by using Intel i7-920 MT, NVIDIA GTS 450 and Altera Stratix V: **14.425x** speedup on 8 threads, and NVIDIA GTS 450 and Altera Stratix V gain 64.11x and 21.37x speedup respectively.

Observations

- Modern ARM powered devices can have similar performance behavior as Intel middle-end series, that implies cost-efficient and flexible hardware selections.
- Various multithreading and GPU programming techniques can be transparently applied to ROS and/or related open source stack.
- Known hotspots like move_base and amcl are hopefully benefiting from advanced hardware features such as SIMD.
- Due to the improvements from core components, navigation stack can aggregate more algorithms and sensing along with redundant designs.
On-going Autonomous Vehicle Development
Continuous open source improvements

• ROS and Autoware were built for PC class hardware, and it is worthy to analyze overall performance hotspot on ARM based devices
• Previous experiments unveil room for resource allocations with heterogeneous hardware configurations
• Safety, formal verification on essential components
A Spin-based model checking for the simple concurrent program on a preemptive RTOS

• Our long-term objective is to create tools that enable engineers and algorithm designers to use formal verification as part of the normal software development cycle
• Current focus is hypervisor / resource partitioning
Charge station integration

- Prototype by ElaadNL
- Direct payment from EV to charger, Direct communication
- Signal Level Attenuation Characterization (SLAC) to establish a communication link between the EV and the charging station
- Payments via IOTA
Seamless integration in smart grid

- Power is generated from Solar PV from shopping mall rooftop.
- Neighbor A sells the surplus of solar power to neighbor B real-time.
- Autonomous bus charges itself with surplus power generated by the neighborhood's smart charger.
- EV pays smart charger with IOTA tokens held in car eWallet.
- Power is provided to shoppers as part of loyalty discounts.
- Tracking based insurance.
- Municipality owned chargers and parking.
- Connected vehicle pays for toll.

source: ElaadNL
What we learn so far...

- Construct autonomous vehicle (AV)
- Linux adoption with open source stack, considering real-time capability
- Improve existing navigation stack
- Performance analysis over ROS and complex software
Contact information:
Ching-Chun (Jim) Huang [黃敬群]
<jserv@ccns.ncku.edu.tw>