Building a Computer Vision Research Vehicle with ROS

Andreas Fregin, Markus Roth, Markus Braun, Sebastian Krebs & Fabian Flohr
Agenda

1. Introduction
2. History
3. Triggering a Heterogeneous Sensor Setup
4. Our Calibration Solution
5. Enhancing ROS Tools / Handling Data
6. Q&A
About Us

• Daimler is the corporate parent of Mercedes-Benz.
• The authors started in team „Pattern Recognition and Cameras“ as PhDs.
• Main research topics: pedestrian intention recognition, traffic light recognition.
• Interests: object recognition from camera images, machine learning.
• Using ROS as research framework for computer vision.

Sebastian Krebs  Markus Braun  Andreas Fregin  Markus Roth  Fabian Flohr
How I came to ROS

- University
- RoboCup@Work
- Need for a framework
- Used est. automotive framework
- Missed simplicity, introspection and especially the doc. (wiki) of ROS
- Came back to ROS
Our ROSified Research Vehicles
Universal CAN Message Decoder

- Message generator for CAN-bus messages
  - CAN msg. description
    - auto. generation
    - .msg files
      - genpp, genpy, genlisp, ...
    - client library msgs

- Decoding/Publishing
  - CAN hardware
    - raw CAN msg.
  - decoder
    - w221_body_can/velocity
    - w221_body_can/yaw_rate
    - w221_body_can/stw_angle
    - w221_body_can/radar_objects
    - ...

Daimler AG

Building a Computer Vision Research Vehicle with ROS | Andreas Fregin | 2017-09-21 | Page 7
Enabling Low-Level Sensor Fusion

Target
• Capture surrounding at the same moment in time
• ... across different sensors
• Precisely time-stamp sensor-readings

Constraints
• Heterogeneous sensors
• Different sensor nodes
• Maybe different cycle rates
• Un stamped sensor data from CAN-bus
Software Triggering

• Use the host PC to software-trigger all sensors
Hardware Triggering

- Use a trigger generator to hardware-trigger all sensors
- Same acquisition time – but when?

Sensor data
Trigger signal

CAN Busses

Unstamped data
Stamped data

CAN

bag_rec

camera1
camera2
camera3
camera4
Sensors do not know about reference time

- The exposure was triggered at the exact same moment, so the data (images) show the same content

- Processing time of heterogeneous setups will vary -> data (images) arrive at different moments in time

- Timestamping using ros::Time::now() will result in different timestamps

- Timestamping using ros::Time::now() is not correct (arrival vs. acquisition!)

- We need to know the moment of triggering in reference time
PTP Time-Sync (Precision Time Protocol, IEEE1588)

• Trigger (Microcontroller) does not know about reference time
• Time-Synchronization: STM32F4 + LWIP + ROSUDP + PTPd

• With each trigger signal, also a trigger message is generated
• Publishing trigger message (std_msgs/Header)

• Sensor nodes receive the trigger message before the sensor data arrives: proper timestamped images, while ensuring all different sensor data have the exact same timestamp!

• Result: All camera images show the exact same moment AND we know the timestamp of that moment
Hardware Triggering with known time

- Microcontroller does know about reference time (via PTP from PC)
- Microcontroller publishes trigger as ROS std_msgs/Header

![Diagram of hardware triggering]

- Sensor data
- Trigger signal
- Trigger ROS topic

- CAN Busses
- M

- Unstamped data
- Stamped data

- CAN
- bag_rec

- Trigger
- camera1
- camera2
- camera3
- camera4
Synchronization with Velodyne LiDARs

- Microcontroller is now PTP time master using GPS time (NMEA string parsing)
- PC is PTP time slave
Calibration

• Whenever you fuse data you need to know about times AND coordinate frames
• Even small errors (sub-decimal) in orientation result in huge position errors for distant objects
• We need a good extrinsic calibration
  • Cameras
  • Laser scanners
  • ...
Intrinsic Camera Calibration

• cameracalibrator.py comes with OpenCV checkerboard-detector
• Has a informative UI that teaches you where to hold the checkerboard (X/Y/Size)
• Does pick the images from running video: user doesn’t has the chance to hold still to avoid motion blur, etc.
• Does not allow to modify data that is used for the calibration step
• Does not generate a sensor-to-car transformation
Calibration Requirements

- One-man show
- On demand checkerboard detection
- Live detection inspection
- Remove images
- Add specific images
Server-Client Calibration using Car-PC + Linux-Tablet

Topics

- total_st.
- camera

Services

- cal-server
- cal-gui

Car

- Trigger detection
- Trigger calibration
- delete detection
- load data
- save data

Tablet

- image(_left/_right)
- image(_sbs)/compressed

- load data
- save data
Timeshift Recording

- Good example for ROS-tool enhancement
- Start recording in the past
- RAM Buffer
- Trigger topic (delayed start/stop)

Rosbag player enhancements
- Step-topic (play/pause)
- Triggered playback
Rewriting Rosbags

- Don’t be afraid using rospython.rosbag to modify existing ROSbags
  - add sensor data
  - add TF (Calibration)
  - add ground truth
  - correct data (e.g. frame_ids, image_encodings, ...)
- ...
Powerful tools / packages

• Setting up complex image processing setups using nodelets

• Strongly typed messages lead to node exchangeability
  • Example: different detectors all use the same in/output messages

• Extremely powerful packages like image_geometry speed up research

• Having tf as the transformation central

• Launch system is very helpful (especially including other launch files)

• Diagnostics capabilities, ...
Our lessons learned

• ROS already includes the concepts to realize complex, heterogeneous sensor setups

• ROS can handle high data throughput and high cycle rates

• ROS is a good starting point for handling large data

• If your needs exceed what ROS comes with – extend it!
Questions ?