Brief Introduction to CommonRoad-io

Christian Pek

Cyber-Physical Systems Group
Technische Universität München

August 14, 2019

List of Contents

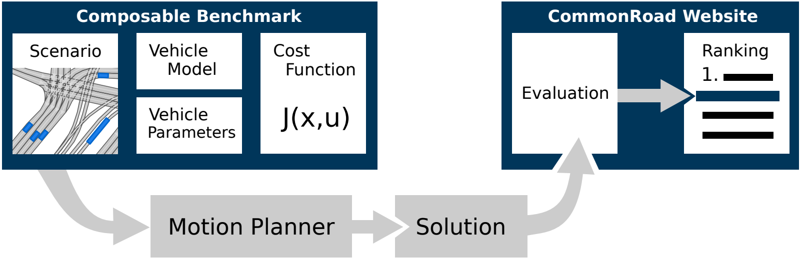
1. Introduction to CommonRoad
2. CommonRoad-Collision-Checker
3. Outlook

1 Introduction to CommonRoad

2 CommonRoad-Collision-Checker

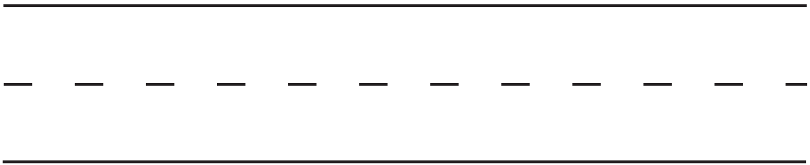
3 Outlook

Composable Benchmarks for Motion Planning



Website: **https://commonroad.in.tum.de**

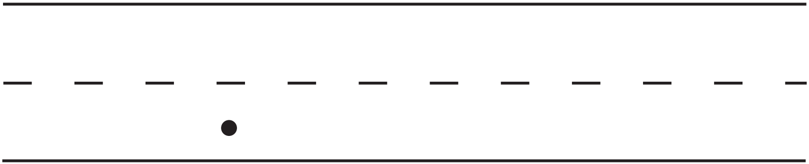
Motion Planning With CommonRoad



Scenario (S)

Road network

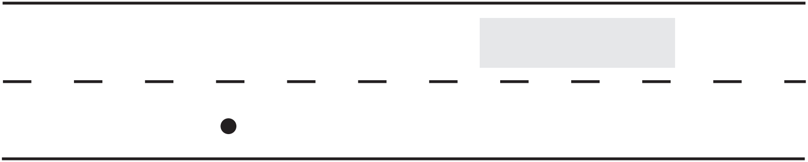
Motion Planning With CommonRoad



Scenario (S)

Road network, initial state x_0

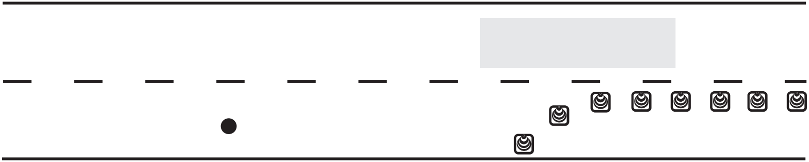
Motion Planning With CommonRoad



Scenario (S)

Road network, initial state x_0, goal region $\mathcal{G}$

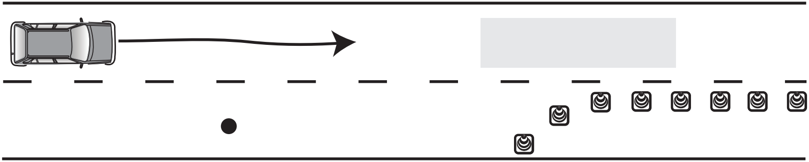
Motion Planning With CommonRoad



Scenario (S)

Road network, initial state x_0, goal region $\mathcal{G}$, static obstacles

Motion Planning With CommonRoad



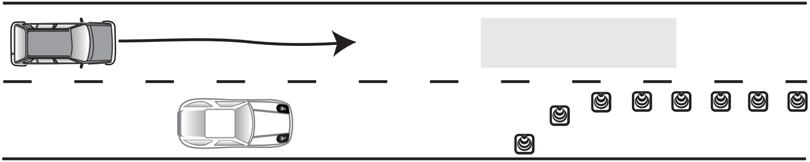
Scenario (S)

Road network, initial state x_0, goal region $\mathcal{G}$, static obstacles, dynamic obstacles (including movement over time)

Motion Planning With CommonRoad

Vehicle model (M)

$\dot{x}(t) = f(x(t), u(t))$
x: state, u: input



Scenario (S)

Road network, initial state x_0, goal region $\mathcal{G}$, static obstacles, dynamic obstacles (including movement over time)

Motion Planning With CommonRoad

Vehicle model (M)

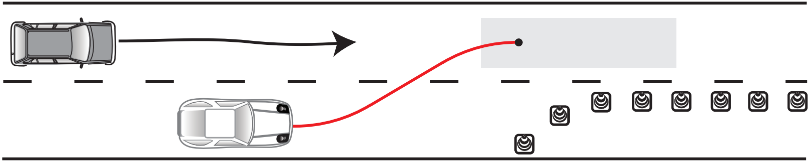
$$\dot{x}(t) = f(x(t), u(t))$$

x: state, u: input

Cost function (C)

$$J_C = \Phi_C(x(t_0), t_0, x(t_f), t_f) + \int_{t_0}^{t_f} L_C(x(t), u(t), t)\,\mathrm{d}t$$

Φ_C: terminal costs,
L_C: running costs



Scenario (S)

Road network, initial state x_0, goal region $\mathcal{G}$, static obstacles, dynamic obstacles (including movement over time)

Motion Planning With CommonRoad

Vehicle model (M)

$$\dot{x}(t) = f(x(t), u(t))$$

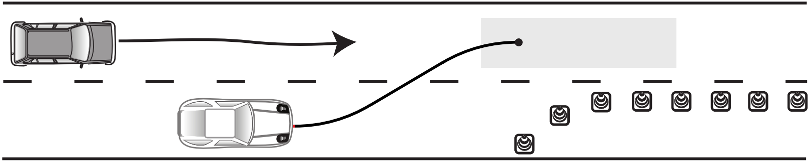
x: state, u: input

Cost function (C)

$$J_C = \Phi_C(x(t_0), t_0, x(t_f), t_f) + \int_{t_0}^{t_f} L_C(x(t), u(t), t)\,\mathrm{d}t$$

Φ_C: terminal costs,

L_C: running costs

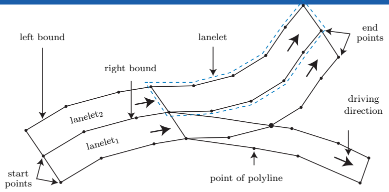


Individual ID: M:C:S

Scenario (S)

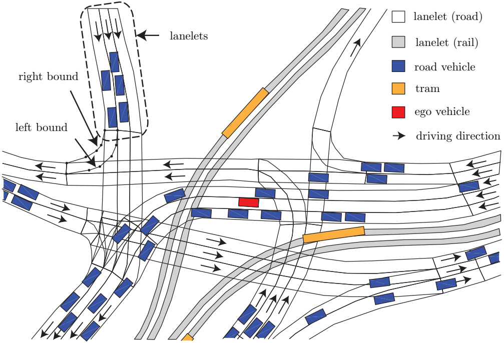
Road network, initial state x_0, goal region $\mathcal{G}$, static obstacles, dynamic obstacles (including movement over time)

Scenarios: Road Network

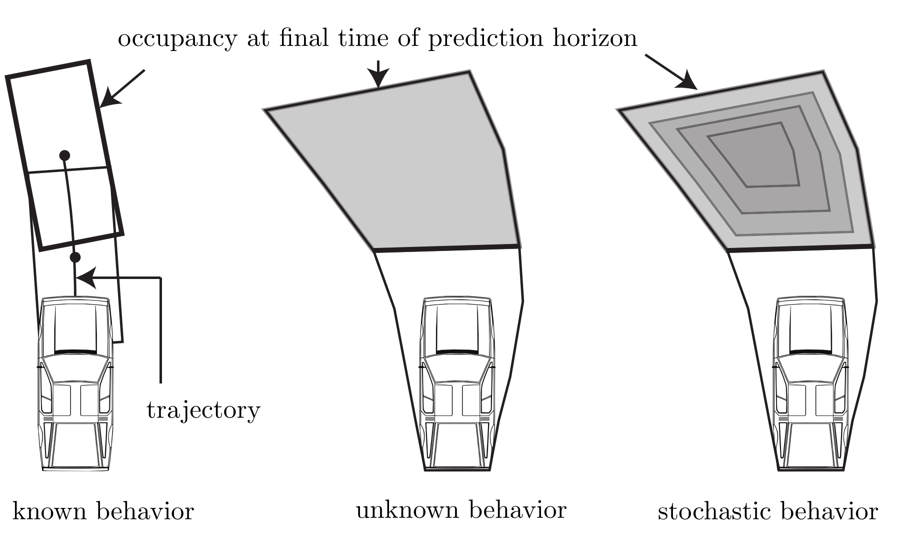


P. Bender, J. Ziegler, and C. Stiller, "Lanelets: Efficient map representation for autonomous driving," in *Proc. of the IEEE Intelligent Vehicles Symposium*, 2014, pp. 420–425.

Example of a complicated crossing in Munich:

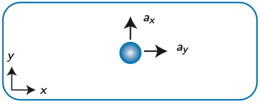


Scenarios: Obstacles



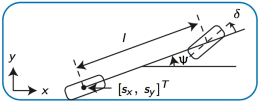
Kinematic Models

Models



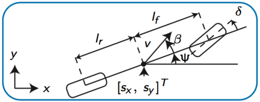
Point-mass model (PM)

- Holonomic system
- $\ddot{x} = a_x, \quad \ddot{x} = a_y$



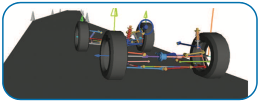
Kinematic single-track model (KS)

- Nonholonomic system
- Considers minimum turning radius
- No tire slip



Single-track model (ST)

- Considers tire slip
- Can explain understeer and oversteer
- No individual tire loads



Multi-body model (MB)

- Individual tire loads
- Effects from yaw, pitch, and roll
- Detailed suspension model

Cost Functions

Like the benchmarks, the cost functions are composable:

$$J_C(x(t), u(t), t_0, t_f) = \sum_{i \in \mathcal{I}} w_i \, J_i(x(t), u(t), t_0, t_f),$$

where $\mathcal{I}$ contains the IDs of partial cost functions and $w_i \in \mathbb{R}^+$ are weights. Examples:

- **Time**: $J_T = t_f$ (see Bobrow et al., 1988).
- **Acceleration**: $J_A = \int_{t_0}^{t_f} a(t)^2 \, \mathrm{d}t$ (see Ziegler et al., 2014b).
- **Jerk**: $J_J = \int_{t_0}^{t_f} \dot{a}(t)^2 \, \mathrm{d}t$ (see Werling et al., 2010).
- **Steering angle**: $J_{SA} = \int_{t_0}^{t_f} \delta(t)^2 \, \mathrm{d}t$ (see Magdici et al., 2016).
- etc.

A set of useful weights is provided by cost-function IDs (e.g. $JB1$, $SA1$, and $WX1$).

Key Features

- **Reproducibility/unambiguity:** Unambiguous information representation & manuals on our website.

Key Features

- **Reproducibility/unambiguity:** Unambiguous information representation & manuals on our website.
- **Composability:** All components (vehicle models, cost functions, and scenarios) are interchangeable.

Key Features

- **Reproducibility/unambiguity:** Unambiguous information representation & manuals on our website.
- **Composability:** All components (vehicle models, cost functions, and scenarios) are interchangeable.
- **Representativeness:** Real traffic and hand-crafted problems (most recorded traffic situations are not critical).

Key Features

- **Reproducibility/unambiguity:** Unambiguous information representation & manuals on our website.
- **Composability:** All components (vehicle models, cost functions, and scenarios) are interchangeable.
- **Representativeness:** Real traffic and hand-crafted problems (most recorded traffic situations are not critical).
- **Portability:** XML for scenarios (platform-independent); Vehicle models in MATLAB and Python (both platform-independent).

Key Features

- **Reproducibility/unambiguity:** Unambiguous information representation & manuals on our website.
- **Composability:** All components (vehicle models, cost functions, and scenarios) are interchangeable.
- **Representativeness:** Real traffic and hand-crafted problems (most recorded traffic situations are not critical).
- **Portability:** XML for scenarios (platform-independent); Vehicle models in MATLAB and Python (both platform-independent).
- **Scalability:** From simple static to complex scenarios with many dynamic obstacles.

Key Features

- **Reproducibility/unambiguity:** Unambiguous information representation & manuals on our website.
- **Composability:** All components (vehicle models, cost functions, and scenarios) are interchangeable.
- **Representativeness:** Real traffic and hand-crafted problems (most recorded traffic situations are not critical).
- **Portability:** XML for scenarios (platform-independent); Vehicle models in MATLAB and Python (both platform-independent).
- **Scalability:** From simple static to complex scenarios with many dynamic obstacles.
- **Openness:** All benchmarks downloadable from our website & possibility to suggest new ones.

Key Features

- **Reproducibility/unambiguity:** Unambiguous information representation & manuals on our website.
- **Composability:** All components (vehicle models, cost functions, and scenarios) are interchangeable.
- **Representativeness:** Real traffic and hand-crafted problems (most recorded traffic situations are not critical).
- **Portability:** XML for scenarios (platform-independent); Vehicle models in MATLAB and Python (both platform-independent).
- **Scalability:** From simple static to complex scenarios with many dynamic obstacles.
- **Openness:** All benchmarks downloadable from our website & possibility to suggest new ones.
- **Independence:** Our benchmarks are independent from planning libraries.

CommonRoad Tools

- OpenDRIVE2Lanelet Converter
  - Python $\geq$3.6
  - Converts OpenDrive maps to CommonRoad road networks which are based on lanelets.
- **commonroad-io**
  - Python $\geq$3.6
  - Provides methods to read, write, and visualize CommonRoad scenarios and planning problems.
  - Integrated solution writer to upload solved benchmark problems to our website.
- **commonroad-collision-checker**
  - C++11 and Python $\geq$3.6
  - Functionality to check if basic geometric shapes and groups of shapes collide.
  - Convenient Python interface for commonroad-io.

1 Introduction to CommonRoad

2 CommonRoad-Collision-Checker

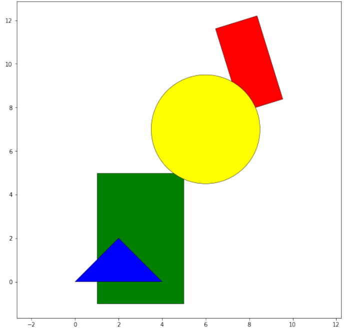
3 Outlook

Introducing the CommonRoad-Collision-Checker

1. Library for fast 2D collision checking,
2. Programming language: C++11 and Python $\geq$3.6
3. Python interface to commonroad-io is available,
4. Collision checks with static and dynamic obstacles are supported,
5. Specialized broadphase collision detection algorithms can be used,
6. Design goals:
   - Easy extensibility to integrate open source collision checking libraries,
   - Easy use with commonroad-io.

Primitive Collision Detection

- Primitive collision objects:
  - axis-aligned rectangles,
  - oriented rectangles,
  - triangles,
  - circles,
  - polygons.
- Complex collision objects:
  - groups of primitive shapes
  - time-variant collision objects



Broad-Phase Collision Detection

- Motion planning: a lot of time is spent on collision checking
- Brute-force collision checking can lead to high runtime $O(n^2)$
- Algorithms for broad-phase collision detection:
  - Bounding Volume Hierarchy
  - Sweep-and-prune
  - Spatial hash map
  - ...
- There exist many open source libraries with specialized broad-phase collision detection algorithms.

Open Source Libraries

1. Comprehensive libraries
   - 3D
     - FCL
     - Bullet
     - ODE
   - 2D
     - Box 2D
     - Chipmunk
2. Smaller libraries
   - OPCODE, SOLID, FreeSOLID, SWIFT
   - RAPID, I-Collide, V-Collide
   - PQP
   - V-Clip
   - KCCD, SELF-CCD, KCD Dynamic
   - QuickCD, Dop-Tree
   - SWIFT++, BoxTree, Q-Collide
   - ColDet, ECDL, ...

Collision Detection Solver

- Currently, we use FCL [Pan et al., 2012] for most primitive collision checks and broad-phase collision checks.
- Broad-phase collision checks are performed with AABB trees.
- Queries can return:
  - binary collision result,
  - list of colliding obstacles,
  - list of colliding object pairs from shape groups and the time of collision.

1 Introduction to CommonRoad

2 CommonRoad-Collision-Checker

3 Outlook

Outlook

- Overview of CommonRoad and its features
- Introduction to CR tools
- Finishing introduction tutorials
- Overview of possibilities for your own research

References I

Pan, J., Chitta, S., and Manocha, D. (2012).

Fcl: A general purpose library for collision and proximity queries.

In *2012 IEEE International Conference on Robotics and Automation*, pages 3859–3866.