An Autonomous Vehicle Control Stack (Benchmark Proposal)

Alena Rodionova[1], Matthew O'Kelly[2], Houssam Abbas[2], Vincent Pacelli[2], and Rahul Mangharam[2]

[1] Technical University of Vienna, Vienna, Austria
alena.rodionova@tuwien.ac.at
[2] University of Pennsylvania, Philadelphia, PA USA
{mokelly,habbas,pacelliv,rahulm}@seas.upenn.edu

Abstract

This benchmark presents an implementation of a standard control stack for an Autonomous Vehicle (AV). The control stack is made up of a behavioral planner (providing waypoints for the AV to visit in sequence), a trajectory planner (which computes smooth trajectory that the AV should follow to go between waypoints) and a trajectory tracker (which actuates the AV to make it follow the planned trajectory as closely as possible). The behavioral planner is purposefully simple, while the trajectory planner is a a high-fidelity approximation of a planner that was tested on a real Prius, and the tracker was validated by others in previous work on a real Cadillac SRX. The interest of this benchmark is that it includes all three components, rather than one AV control subsystem (such as only adaptive cruise control or only a lane-keeper), and the planners are significantly more realistic than most existing benchmarks or models. It can be used as a baseline AV system for verification and testing tools, which must be able to handle at least the complexity of this controller. This includes simple choices made by the behavioral planner when the current waypoint cannot be reached, discrete and continuous nonlinear optimizations solved by the trajectory planner, and nonlinear ODEs solved by the trajectory tracker. The benchmark comes with three road topologies: a free space with obstacles, a curved road, and a roundabout.

1 Introduction

Autonomous Vehicles (AVs) are a very active area of research, both in industry and academia. The range of capabilities that an AV must have has spurred new research in perception (computer vision and signal processing, sensors and VLSI), control theory, and verification and testing. In particular, the verification and testing efforts have centered on sub-systems of the AV, like adaptive cruise control and lane-keeping. Some of the formal verification work that has addressed the whole AV has used very simplified models [8]. This is understandable since, as we will see, a more realistic model of an AV's control stack already has enough complexity that goes beyond today's tools, short of a significant effort to model the various planners mathematically and to validate their models.

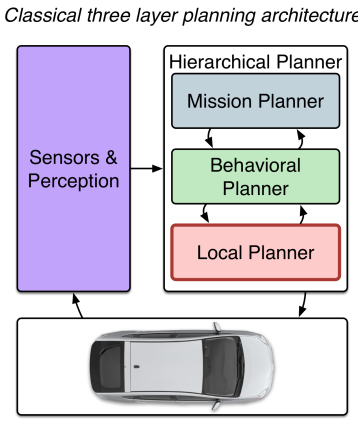


Figure 1: Control architecture of proposed benchmark.

The proposed benchmark presents an implementation of the de facto standard control architecture for AVs, shown in Fig. 1. It consists of three planners and a trajectory tracker. Traversing it from long-term planners to short-term planners, it consists of:

- Mission Planner (MP): this can be roughly thought of as a map service like Google Maps. Based on the desired destination of the AV, it produces the sequence of mission points that the vehicle should reach one-by-one. This is the most high-level planner in the AV control stack. We do not include a Mission Planner in the benchmark. Rather, the user can provide a sequence of mission points explicitly.

- Behavioral Planner (BP): takes the information about the environment and the next mission point provided by MP as an input and outputs the next waypoint which should be reached by the AV. This planner focuses on the local-level tasks generation for an AV, like lane changing or driving straight, which progress toward the next mission point.

- Trajectory Planner (TP): given the environment information, the current position of the AV and the next waypoint produced by the above BP, the TP generates a smooth trajectory using a cubic spline that the AV should follow to get to the next waypoint.

- Trajectory Tracker (TT): this level takes a desired trajectory generated by TP as an input and provides control commands for the AV in order to follow the given trajectory as closely as possible. The control inputs to the system are the longitudinal acceleration and the steering wheel velocity.

2 Brief Description

See Appendix for details of TP and TT.

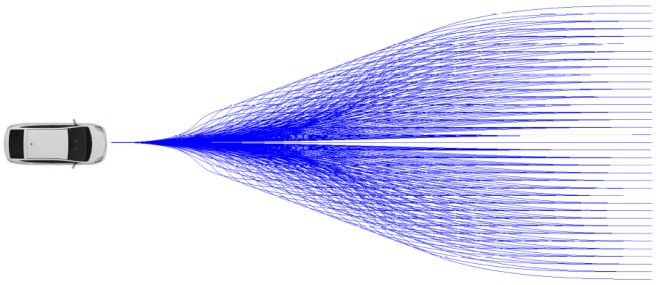


Figure 2: Generated set of feasible trajectories by TP

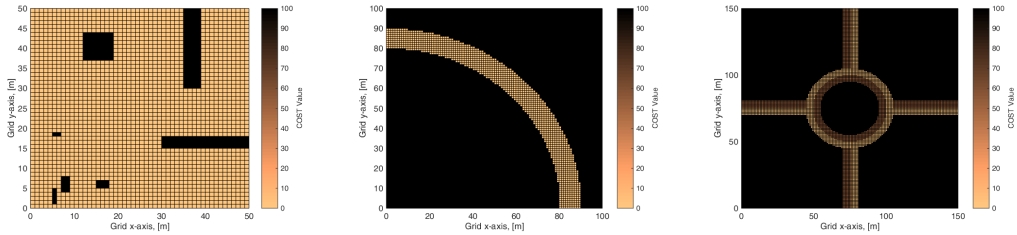
2.1 The planners

The BP operates as follows: it receives from the MP the sequence of mission points $mp_1, mp_2, \ldots, mp_N$. Each mission point can be thought of as a 2D point in some reference system. The BP starts by setting the first waypoint $w_1 = mp_1$, and passes it to the TP. If the TP returns successfully, this means it was able to compute a trajectory leading from the current position to w_1, and the TT was able to track it successfully. The BP then sets $w_2 = mp_2$ and repeats. If at iteration n the TP returns 'Fail', the BP must choose another waypoint and there is a number of ways to do so. We adopted a simple shortening of the horizon: the BP builds a shortest path from the current position c to w_n over a coarse grid that partitions the continuous space. The cost of a cell in the map is a function of the obstacles in it. Let $P = P_1 \ldots P_m$ be the sequence of cells in the shortest path, such that $c \in P_1$ and $w_n \in P_m$. Then BP updates $w_n = P_k$ for some $k, 2 \leq k < m$. In the benchmark, the user controls the details of choosing k. If after the horizon shortening procedure the BP cannot produce any feasible next waypoint ($w_n = P_2$, so the horizon cannot be reduced anymore, and the TP returns 'Fail' for such w_n), then the BP returns a non-feasible next waypoint, returns the 'fail' BP flag, but the simulation still continues. Finally, the BP passes an updated w_n to the TP, and the above procedure repeats.

A fundamental decision we made with this benchmark is that the BP should be purely based on the cost maps. That is, the BP looks at a (coarse) cost map and decides the next waypoint accordingly. Traffic rules, traffic flow directions, etc., are encoded in the way the cost maps are updated rather than in the logic of the BP itself. This allows for a more modular and more maintainable design.

The TP operates as follows: first, it receives from the BP the next waypoint. Second, given the current state of the vehicle, the TP computes a set of smooth and kinematically feasible candidate trajectories that lead to the goal point and to points near it, as detailed in the appendix.

Third, when such set of the trajectory candidates is obtained (see Figure 2), the TP selects a single trajectory based on the information about the environment: it computes the cost of each trajectory in this set based on the cost map, and chooses the lowest cost one. If there are several lowest-cost trajectories, the TP chooses the one that leads to the point closest to the goal point. Finally, if the cost of this single generated trajectory is below some predefined threshold, then the TP passes this trajectory to the TT. Otherwise, then the TP returns 'Fail' to the BP and the BP should choose another way point.



(a) Free space with obstacles (b) Curved road (c) Roundabout

Figure 3: Three pre-programmed cost maps

The TT operates as follows: it receives a desired trajectory from the TP and generates and applies a set of control inputs u, based on the dynamical model of the system. The tracking controller provides the commanded steering wheel velocity v_w and the commanded longitudinal acceleration a_x. Such AV dynamics is described by a 7D first-order non-linear ODE system, so-called bicycle vehicle model [2]. The model ignores roll and pitch, and considers only one front and one rear wheel. See Appendix for details.

2.2 The maps

The benchmark contains three pre-programmed cost maps. The user can add obstacles to any of the maps, and control their sizes as well as the granularity of the grid used by the BP for proposing alternative waypoints.

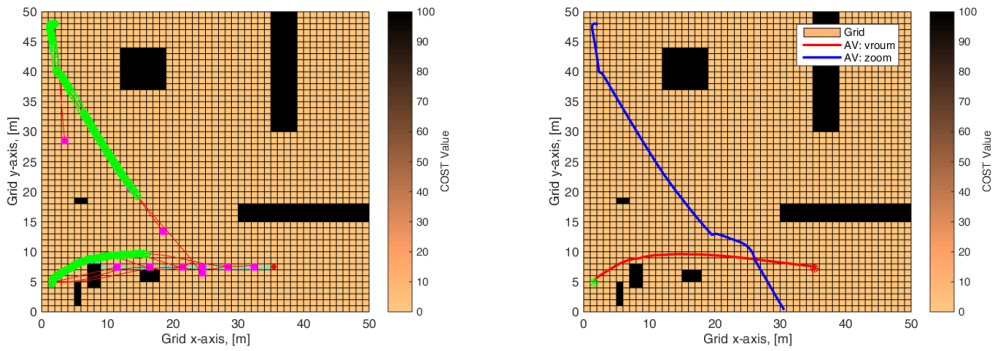
1. Free space with obstacles: This map is a free (navigable) space, with some static obstacles. The user controls the location and size of obstacles. See Fig. 3a.

2. Curved road: Such map is a simple arc of a circle with a constant radius throughout the circle, see Fig. 3b. The road space is free (zero-cost) space with no static obstacles. The user controls the inside radius and the width of the road.

3. Roudnabout: Is a special type of intersection where the road traffic flows only in one direction around a central island, and both directions outside of the circle. Presented map is a two-lane version of a roundabout scenario, see Fig. 3c. Lane width parameter was set up based on the European road standards, but can be controlled by the user.

2.3 Multi-AV simulations

The benchmark supports the simulation of multiple AVs in the same map, with each AV assigning its own costs to the obstacles in the map. An agressive AV will assign very low cost close to the obstacles, thus allowing it to navigate very close to them, whereas a conservative AV will have a more gradual cost decrease, discouraging proximity to obstacles.



(a) AVs dynamics (b) Resulting trajectories

Figure 4: Simulation of the `emptyWithObstacles` map scenario

2.4 Running the benchmark

To run the benchmark, the user should cd folder `examples/` and run `test_xagent`. The variable `mapID` sets up which one of three listed scenarios is simulated: `emptyWithObstacles`, `curve` or `roundabout`. In case of the scenario `emptyWithObstacles`, the user is free to choose the initial state and the target point for the second AV. Such setting is controlled by the value of the variable `situationID`. Depending on a `situationID`, the user is able to simulate such situations as: parallel navigation of AVs (`situationID` $= 1$), navigation with crossing trajectories (`situationID` $= 2$), and collision avoidance navigation (`situationID` $= 3, 4$).

While simulation is running, the dynamics of both AVs is presented graphically, see Fig 4a. The user can see the generation of waypoints produced by BP (magenta color), set of candidate trajectories generated by TP (red curves) and the final tracked trajectories (green color curves). The second figure that the user sees displays the resulting trajectories followed by both AVs, see Fig 4b. Note, since both of these dynamic trajectories are presented as a static figure, trajectories crossing does not always correspond to the collision situation between AVs.

3 Limitations and future improvements

Each AV can have its own planning update period T. Every T milliseconds, the planner updates its cost map with the new positions of other AVs, and re-plans its waypoints and trajectories. Currently, however, the muti-AV simulation requires all AVs to have the same update period, which is the default behavior of the code.

The benchmark ignores an important consideration in real planners, which is that planning takes non-zero time during which the AV continues to move. A more realistic planner will plan a path not from the current position, but from the predicted future position at which the planning computation completes. The current benchmark ignores this and plans from the current position. A future version of the benchmark will take this into account.

Finally, the update of the cost maps currently only takes into consideration the positions of the AVs. Future versions will incorporate traffic rules.

4 Outlook

The objective of this benchmark is to provide a (relatively) simple baseline for an AV control stack, which contains enough ability to navigate 2 or more AVs on a cost map. The literature has *many* motion planners that could be used instead of the ones we provide here, with various guarantees on their performance (although usually the evaluations are empirical). The TP we use is a validated approximation of a TP that has run on a real AV. The value of the approximation is that it takes a very involved optimization, and replaces it by a neural network with a known structure. The TT was validated by others on a real car. The BP uses a very simple horizon shortening heuristic.

To enable verification of such a control stack, the first hurdle is the modeling of the BP and of the cost optimization performed by the TP. The current BP can be modelled as a hybrid system, whose modes indicate the different possible choices of alternative waypoints when the lower planners return Failure. *This assumes that the shortest path on the grid is given.*

The TP cost optimization, on the other hand, is more involved and might require a continuous-limit approximation to model. That is, the grid is replaced by a continuous cost surface with a simplified, parametrized shape.

Because we would then have a nonlinear hybrid system, falsification is likely to play a major role in testing this benchmark, and a tool like dReach will be required for verification. However, we already know that dReach cannnot handle this benchmark in its full generality. Approaches like Robustness-Guided Verification, introduced in [1], should be able to tackle the complexity issue.

A Ego Vehicle Model

The following bicycle model is lifted from [2] and provided here for ease of reference. It is a non-linear 7 degree of freedom bicycle model [6]. See Fig. 5. The input to such a model is steering angle velocity and linear velocity, the output is vehicle state as a function of time.

The state vector describing the vehicle is $x_v = (\beta, \Psi, \dot{\Psi}, v, s_x, s_y, \delta)$. The variable β is the slip angle at the center of mass, ψ is the heading angle, $\dot{\psi}$ is the yaw rate, v is the velocity, s_x and s_y are the x and y positions, and δ is the angle of the front wheel. The inputs to the system are a_x, the longitudinal acceleration, and v_w the rotational speed of the steering angle.

The state equations for the system as described in [2] are:

$$\dot{\beta} = \left(\frac{C_r l_r - C_f l_f}{m v^2}\right)\dot{\psi} + \left(\frac{C_f}{mv}\right)\delta - \left(\frac{C_f + C_r}{mv}\right)\beta \tag{1}$$

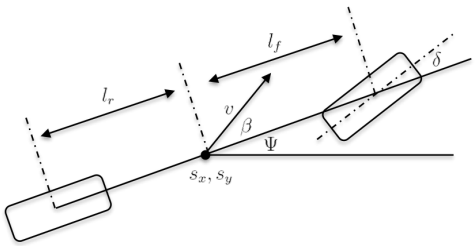


Figure 5: Nonlinear bicycle model

$$\ddot{\psi} = \left(\frac{C_r l_r - C_f l_f}{I_z}\right)\beta - \left(\frac{C_f l_f^2 - C_r l_r^2}{I_z}\right)\left(\frac{\dot{\psi}}{v}\right) + \left(\frac{C_f l_f}{I_z}\right)\delta$$

$$\dot{v} = a_x,\ \dot{s_x} = v\cos(\beta+\psi),\ \dot{s_y} = v\sin(\beta+\psi),\ \dot{\delta} = v_w \tag{2}$$

Vehicle Parameters Table 1 contains the validated vehicle parameters as given in [2].

Table 1: Parameters of Example Ego Vehicle [2]

Vehicle Parameters					
$m(kg)$	$I_z(kg*m^2)$	$C_f(N/rad)$	$C_r(N/rad)$	$l_f(m)$	$l_r(m)$
2273	4423	10.8e4	10.8e4	1.292	1.515

Trajectory Tracker. A good tracker stabilizes the system and minimizes the error between the planned trajectory and actual trajectory. The control inputs v_w and a_x can be computed as follows [2]. Variables subscripted with d are desired values from the planned trajectory.

$$\begin{aligned} v_w = k_1(cos(\Psi_d)(s_{y,d} - s_y - w_y) - sin(\Psi_d)(s_{x,d} - s_x - w_x)) + k_2(\Psi_d - \Psi - w_\Psi) \\ +k_3(\dot{\Psi}_d - \dot{\Psi} - w_\psi) - k_4(\delta - w_\delta) \\ a_x = k_5(cos(\Psi_d)(s_{x,d} - s_x - w_x) + sin(\Psi_d)(s_{y,d} - s_y - w_y)) + k_6(v_d - v - w_v) \end{aligned} \tag{3}$$

The parameters are $k_1 = 2, k_2 = 12, k_3 = 4, k_4 = 2, k_5 = 1, k_6 = 1.515$ They were obtained for the Cadillac SRX research vehicle of Carnegie Mellon University as described in [7].

B TP details

The TP utilizes the methods outlined in [3] commonly known as state-lattice planning with cubic spline trajectory generation.

Each execution of the planner requires as an input the current state of the vehicle and a goal state as defined by the behavioral planner. The vehicle state used in the TP will be written $x_{sl} = (s_x, s_y, v, \Psi, \kappa)$, where s_x and s_y are the x and y positions of the center of mass, v is the velocity, Ψ is the heading angle, and κ is the curvature. Note x_{sl} is not the same as that used in A, because the TP must run online, in real-time, so lower order models are often used.

The state equations of the trajectory are described as (L is the wheelbase of the vehicle, a constant.):

$$\dot{x} = v * cos(\Psi),\ \dot{y} = v * sin(\Psi),\ \dot{\theta} = \kappa * v,\ \dot{\kappa} = \frac{\dot{\Psi}}{L} \tag{4}$$

The TP's objective is then to find a feasible trajectory from the initial state x_{sl} to a goal pose $x_p = (s_x, s_y, \Psi, \kappa)$. We limit trajectories to cubic splines. A cubic spline is defined as a function of arc length s:

$$\kappa(s) = \kappa_0 + a\kappa_1 s + b\kappa_2 s^2 + c\kappa_3 s^3 \tag{5}$$

Note that there are four free parameters (a, b, c, s_f) (where s_f is the total arc length of the trajectory) and our goal posture x_p has four state variables. We first perform a stable reparameterization with new parameter p:

$$\kappa(0) = p_0,\ \kappa(s_f/3) = p_1,\ \kappa(2s_f/3) = p_2,\ \kappa(s_f) = p_3 \tag{6}$$

so that he parameters (a, b, c, s_f) can now be expressed as:

$$a(p) = p_0, b(p) = -\frac{11p_0 - 18p_1 + 9p_2 - 2p_3}{2s_f} \tag{7}$$

$$c(p) = \frac{9 * (2p_0 - 5p_1 + 4p_2 - p_3)}{2s_f^2}, d(p) = -\frac{9(p_0 - 3p_1 + 3p_2 - p_3)}{2s_f^3} \tag{8}$$

For any particular (state, goal) pair two steps are necessary to compute the parameters. First, we produce an initial guess:

$$p_0 = \kappa_0 = \kappa_i, p_1 = \kappa_1 = \frac{1}{49}(8b(s_f - s_i) - 26\kappa_0 - \kappa_3) \tag{9}$$

$$p_2 = \kappa_2 = \frac{1}{4}(\kappa_3 - 2\kappa_0 + 5\kappa_1), p_3 = \kappa_3 = \kappa_f \tag{10}$$

Then, with an initial guess in hand the local planner can solve a simple gradient descent problem to drive the vehicle to the goal posture.

C Example specifications

Formal verification requires both a system model and a specification. Based on the scenario of the interest, the specification to verify could include traffic rules, speed limit, distance to other cars, etc. An example specification for the ego vehicle can be defined as follows:

- The ego vehicle travels at a velocity less than or equal to the speed limit: $G(v_{ego} \leq v_{limit})$
- The ego vehicle does not drive backwards: $G(v_{ego} \geq 0)$
- If the AV started to accelerate then the AV will not start decelerate right after that moment for some time in the future:$G((a_x \geq \theta) \rightarrow G_{(0,\epsilon]} \neg (a_x \leq -\theta))$
- The ego vehicle does not collide with any of the n other vehicles in the environment: $G\Big(\sqrt{(s_{x_{ego}} - s_{x_{env_i}})^2 + (s_{y_{ego}} - s_{y_{env_i}})^2} \geq r\Big)$

References

[1] Houssam Abbas, Matthew O'Kelly, and Rahul Mangharam. Relaxed decidability and the robust semantics of metric temporal logic. In *Hybrid Systems: Computation and Control*, 2017.

[2] Matthias Althoff and John M. Dolan. Online verification of automated road vehicles using reachability analysis. *IEEE Transactions on Robotics*, 4(30):903–918, 2014.

[3] Matthew McNaughton. *Parallel Algorithms for Real-time Motion Planning.* PhD thesis, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, July 2011.

[4] Matthew McNaughton. Parallel algorithms for real-time motion planning, phd thesis, robotics institute, carnegie mellon university, pittsburgh, pa, July 2011.

[5] Bryan Nagy and Alonzo Kelly. Trajectory generation for car-like robots using cubic curvature polynomials. field and service robots, 11, 2001.

[6] Rajesh Rajamani. *Vehicle Dynamics and Control.* Springer, 2006.

[7] Jarrod M. Snider. Automatic steering methods for autonomous automobile path tracking, 2009.

[8] Richard M. Murray Andrew Lamperski Tichakorn Wongpiromsarn, Sayan Mitra. Periodically controlled hybrid systems. In *12th International Conference, HSCC 2009, San Francisco, CA, USA, April 13-15, 2009. Proceedings*, 2009.