

AUTO-TUNING OF SYMBOLIC CYBER-PHYSICAL CONTROL ENVELOPES

MUSTAFA SAMI, STEFAN MITSCH

ABSTRACT

Formal verification produces symbolic safety envelopes whose parameters are left unspecified for real-world use. We automatically tune those values by collecting driving data in CARLA under varied traffic, weather, and manual-control scenarios. This data-driven approach bridges the gap between formal models and simulator behavior, paving the way for integrated runtime-verification workflows.

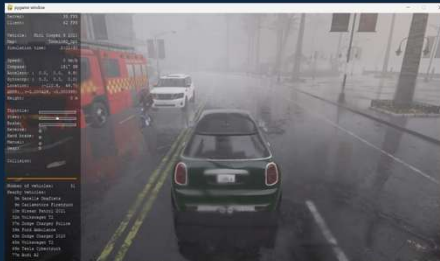
INTRODUCTION

- **Motivation:** CPS need both symbolic safety guarantees and concrete parameter values
- **Gap:** KeYmaera X proofs yield parameters but can't be tested "as is."
- **Goal:** Leverage CARLA to gather traces and auto-tune.

```
1 Theorem "LJRR17/Theorem 1: Static safety"
2
3 Definitions
4 Real ep; /* time limit for control decisions */
5 Real b; /* minimum braking capability of the robot */
6 Real A; /* maximum acceleration -b <= a <= A */
7 Real W; /* maximum steering */
8
9 Real stopDist(Real v) = v^2 / (2*b); /* The straight-line stopping distance from brake start to full stop. */
10 Real accelComp(Real v) = ((A/b + 1) * (A/2 * ep^2 + ep*v)); /* Straight-line distance to compensate acceleration */
11 Real admissibleSeparation(Real v) = stopDist(v) + accelComp(v); /* Separation that allows accelerating on a new curve */
12
13 Bool isWellformedDir(Real dx, Real dy) <-> dx^2 + dy^2 = 1; /* The orientation of the robot is a unit vector. */
14
15 Bool bounds() <-> ( /* Bounds for global constants */
16   A >= 0 /* Working engine */
17   & b > 0 /* Working brakes */
18   & ep > 0 /* Controller reaction time */
19 );
20 Bool initialState(Real x, Real y, Real v, Real dx, Real dy, Real xo, Real yo) <-> ( /* Stopped safe initially */
21   v = 0
22   & (x-xo)^2 + (y-yo)^2 > 0
23   & isWellformedDir(dx, dy)
24 );
```

SIMULATION SCENARIOS

- **Traffic Generation:** Spawned 30 cars + 10 pedestrians via Python API in Town10HD_Opt.
- **Dynamic Weather:** Ran dynamic_weather.py to cycle day \rightleftharpoons night, clouds, rain, wind.
- **Manual Control:** Drove via manual_control.py (pygame); logged speed & collision data.



```
Server: 12 FPS
Client: 62 FPS

Vehicle: Mercedes Sprinter
Map: Town10HD_Opt
Simulation time: 0:01:18

Speed: 0 Km/h
Compass: 316° NW
Accelero: ( 0.5, 0.0, 9.9)
Gyrosco: (-0.1, 0.3, 0.3)
Location: (-51.0, -72.3)
GNSS: ( 0.000656, -0.000463)
Height: 0 m

Throttle: [ ]
Steer: [ ]
Brake: [ ]
Reverse: [ ]
Hand brake: [ ]
Manual: [ ]
Gear: 1

Collision:

Number of vehicles: 31
Nearby vehicles:
15m Ford Crown
27m Audi A2
38m Lincoln Mkt 2017
39m Ford Ambulance
58m Audi Etron
63m Nissan Patrol
73m Tesla Cybertruck
74m Bmw Grandtourer
84m Mitsubishi Fusorosa

Collision with 'Vegetation'
```

DISCUSSIONS

- **Validation:** Simulation traces validate symbolic bounds under real-time control.
- **Performance:** Client FPS \sim 60, Server FPS \sim 30–35—enough for interactive tuning.
- **Limitations:** GPU/DirectX issues on Windows; topology; no pedestrian dynamics.

FUTURE DIRECTIONS

- **Scenic-Driven Scenarios:** Formally define layouts, speed rules, and pedestrian behaviors in concise Scenic scripts.
- **High-Volume Simulation:** Use Scenic's samplers to spawn and run hundreds of CARLA tests automatically.
- **Adaptive Envelope Tuning:** Feed those traces back into the optimizer to refine control envelopes under multi-agent, pedestrian-rich conditions.

REFERENCES

- Dosovitskiy A., Ros G., Codevilla F., López A. M. & Koltun V. (2017). CARLA: An Open Urban Driving Simulator.
- Fremont D. J., Kim E., Dreossi T., Ghosh S., Yue X., Sangiovanni-Vincentelli A. L. & Seshia S. A. (2023). Scenic: a language for scenario specification and data generation.
- Stefan Mitsch and André Platzer. ModelPlex: Verified runtime validation of verified cyber-physical system models.