ROAR XD

A model for Autonomous Mobility-as-a-Service

Wesley, **A**lfredo, **S**ihao, **A**lvin, and **A**man's **P**roject (WASAAP)


brief
\ /
(o.o)
(| | |)
^ ^

what is ROAR?

Robot Open Autonomous Racing:
Racing 1/8 model race cars around a track

2020 version: simulated cars in Carla

original goals \(^o^)/

Big Picture: model of an autonomous Mobility-As-A-Service platform

Sensing: lane detection, obstacle detection, and traffic signal detection using onboard sensors

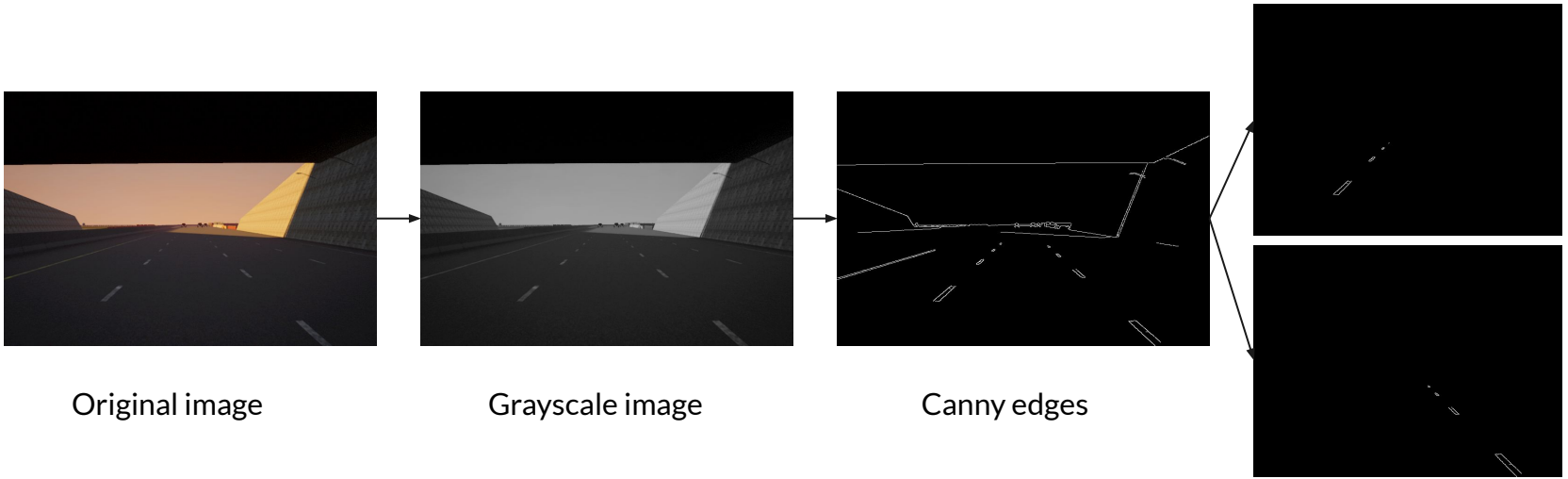
Planning: lane keeping, obstacle avoidance, and adherence to traffic rules

Actuation: PID and LQR controller implementation and tuning for responsive and smooth execution

Racing: make that bad boy go fast without crashing (too much)

what our project actually does
~(o 30)~

sensing



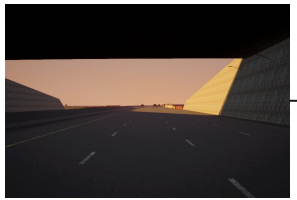
Original image

Grayscale image

Canny edges

Area(s) of interest

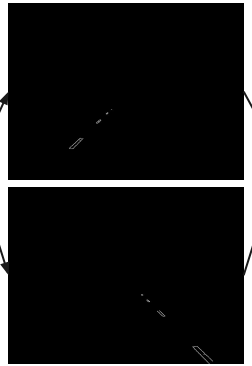
Canny, J. (1986). A computational approach to edge detection. *IEEE Transactions on pattern analysis and machine intelligence*, (6), 679-698.



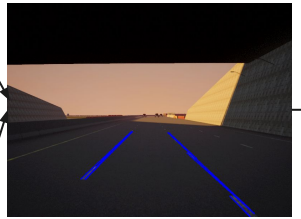
Original image

Grayscale image

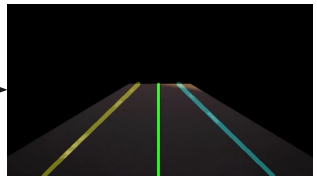
Canny edges



Area(s) of interest

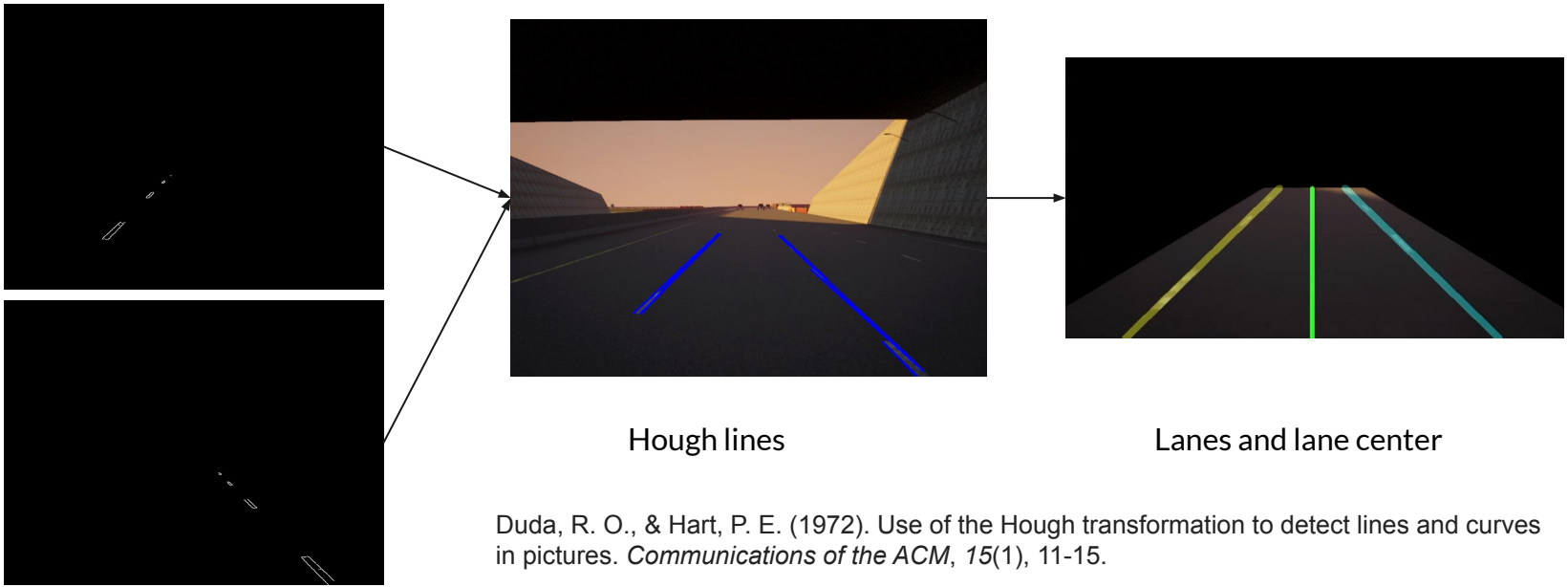


Hough lines



Lanes and lane center

sensing

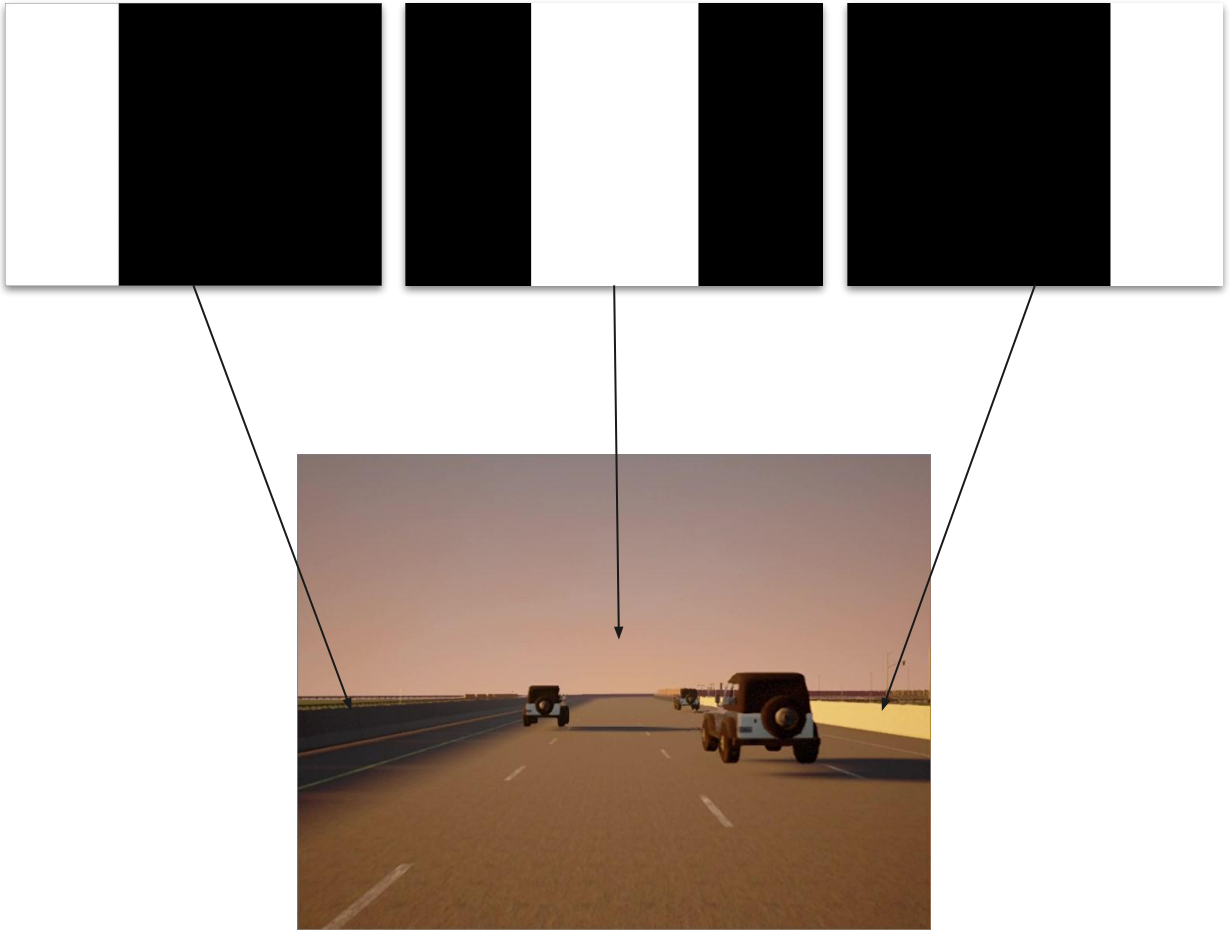


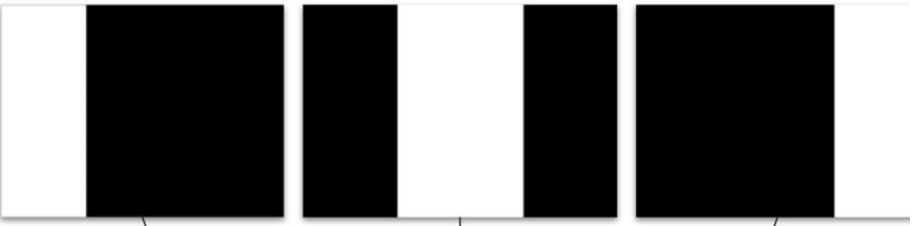
Hough lines

Lanes and lane center

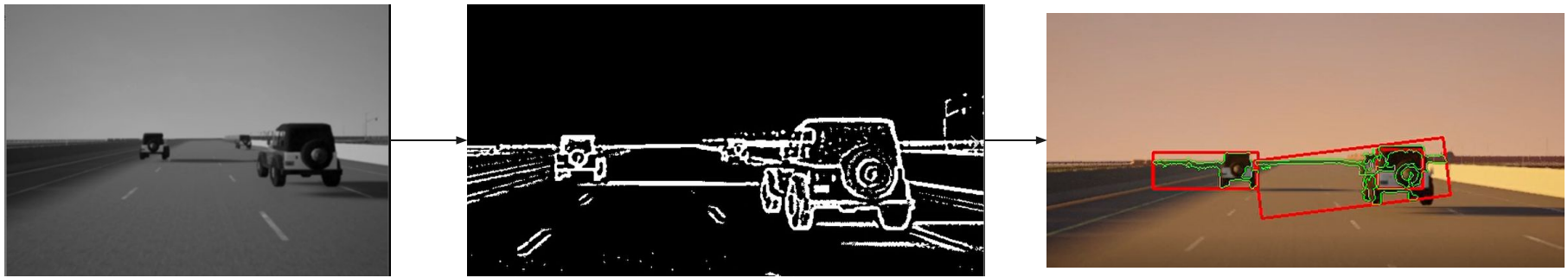
Duda, R. O., & Hart, P. E. (1972). Use of the Hough transformation to detect lines and curves in pictures. *Communications of the ACM*, *15*(1), 11-15.

object detection





continued...



planning

```
def follow_lane(waypoints, α: confidence decay rate 0~1, β: speed limit factor >0):
    confidence = 1
    for each step:
        left_lane, right_lane, lane_center = detect_lanes_from_camera_input()
        if left_lane or right_lane not detected:
            confidence = confidence * α
        else:
            confidence = 1
        target_location = mid_point(lane_center) * confidence +
                              get_next_waypoint(waypoints) * (1 - confidence)
        target_speed = maximum_speed * exp((1 - confidence) * -β)
        controller.run(target_location, target_speed)
```

Simplified lane following algorithm

actuation (before)

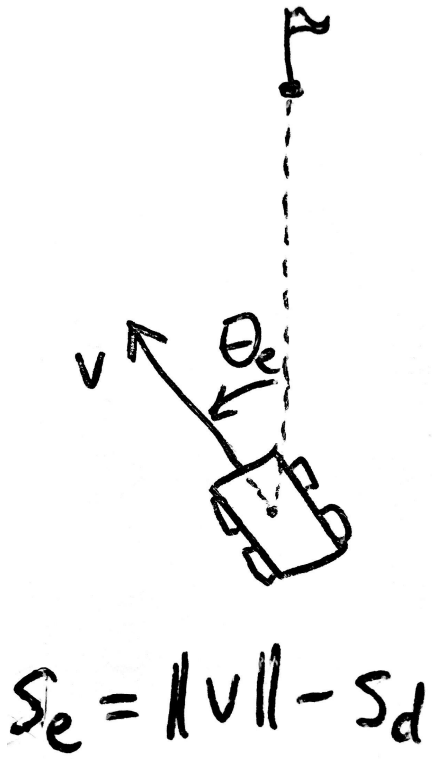
Target direction and speed from planner

Two inputs: steering and throttle

Two separate PID controllers:

1. controlled throttle based on error in speed
2. controlled steering based on error in angle

Problems: rough ride, aggressive turns, slower-than-expected speeds



actuation (changes)

Reactive speed control
- reduced speed when error in direction is large
- smoother turns and recovery

Tuned PID K-values for smoother control

PID shortcomings: unintuitive parameters, no system dynamics

Solution: develop and implement an LQR controller

actuation (after/LQR basics)

Two parts of an LQR controller:

1. system dynamics
2. costs

Pros: accounts for car quirks, intuitive to modify controller behavior

Cons: system dynamics are a pain to tune

Result: significant improvement over the PID controllers, especially at high speeds

System dynamics:

$$\begin{bmatrix} \theta_{t+1} \\ v_{t+1} \end{bmatrix} = \underbrace{\begin{bmatrix} 1 & 0 \\ 0 & 0.995 \end{bmatrix}}_{A} \begin{bmatrix} \theta_t \\ v_t \end{bmatrix} + \underbrace{\begin{bmatrix} 0.1 & 0 \\ 0 & 3 \end{bmatrix}}_{B} \begin{bmatrix} s_t \\ t_t \end{bmatrix}$$

↑ next state ↑ state ↑ input

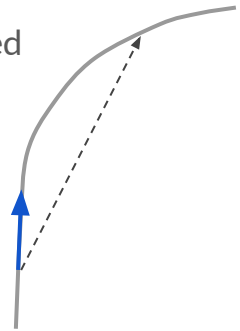
Cost of deviation: $Q = \begin{bmatrix} 0.02 & 0 \\ 0 & 0.01 \end{bmatrix}$

Cost of actuation: $R = \begin{bmatrix} 2.5 & 0 \\ 0 & 2.5 \end{bmatrix}$

racing

Optimize path and speed planning based on previously acquired track waypoints.

- Implement waypoint look-ahead for smoothing and cutting turns
  - Look and sample from N waypoints ahead to reduce noise in previously acquired measurements for position and orientation
- Also proactively adjust target speed based for turns and straight sections
  - Slowdown based on look ahead used to calculate amount of turn that will be required
- Tune LQR for faster speeds, also adjust lookahead based on speed.



Car skids at some turns, but is able to recover. No collisions (most of the times).

Achieved max speed above 185 km/hr and lap time of ~1:15 min (demo).

racing

No look-ahead

Look-ahead

difficulties
(-_- ;)

difficulties

- Lane detection: Areas without lanes
- Hardware: all devices have different speeds, hard to compare and make sure it will work when deployed in race
- Planning and control: Sometimes response for braking before the curve could be slow, which caused to brake during the curve and cause skidding
- Bugs: Found a bug in original implementation where the calculated angle error was absolute error in 3d, instead of the 2d plane. Spent quite a bit of time trying to fix the controllers but the issue was here. This caused error to always be positive in the uphill, hence the controller would never converge since the error kept being high and starts oscillating.
- Carla confusing coordinates: left-handed in Carla and right-handed in ROAR

improvements/extensions
(ノ◕ヮ◕)ノ*:･ﾟ✧

improvements/extensions

Improvements:

- improve lane-keeping so it's robust to turning at high speeds
- fully implement object detection

Extensions:

- build object avoidance into path planning
- include basic traffic signal detection and adherence
- import real-world maps into Carla and utilize third-party routing tools (e.g. Google Maps) to drive the car from point A to point B (i.e. rudimentary implementation of Mobility-As-A-Service)

live demo
(rawr :3)

questions?
(▪◇▪) ?

recorded demo


Server:
Client:
Vehicle:
Map:
Simulation time:
Speed:
Compass:
Location:
Height:
Throttle:
Steer:
Brake:
Reverse:
Hand brake:
Manual:
Gear:
Collision:
Number of particles: