# Augmenting Learning Components for Safety in Resource Constrained Autonomous Robots

Shreyas Ramakrishna[1], Abhishek Dubey[1], Matthew Buruss[1], Charles Hartsell[1], Nagabhushan Mahadevan[1], Saideep Nannapaneni[2], Aaron Laszka[3], and Gabor karsai[1]

[1] Institute for Software Integrated Systems, Vanderbilt University

[2] Wichita State University

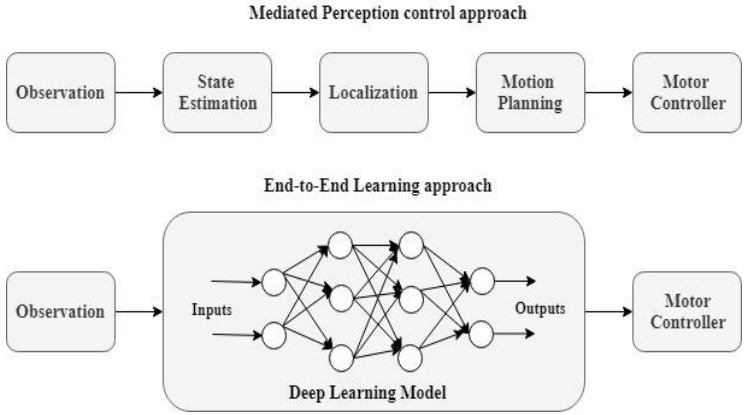
[3] University of Houston

# Outline

- Introduction & Research contribution.
- Testbed – DeepNNCar.
- Our Contributions:
  - Blending Control actions for safety.
  - Resource Management.
  - Confidence Estimator.
  - System Integration.
- Experiments & Results.
- Summary & Ongoing work.

# Autonomous Robots

- Learning Enabled Components (LEC) driven autonomous robots have been used in self driving, UAV's, Navigation, and off road obstacle voidance, etc.
- Recently end-to-end learning approach has gained popularity. (Alpha Go & NVIDIA DAVE-II )
- E2E learning uses data-driven models (Neural Networks) to map sensor inputs to control actions.




Middleware Framework

1. Augment safety controllers & blend control actions to improve safety
2. Resource management & Task Offloading
3. System Level Confidence Estimator


End-to-end reinforcement learning trained AlphaGo.

NVIDIA's DAVE-II performed steering using e2e learning.

**Problem with data driven LEC models**
- Black box with no safety guarantees.
- Testing and verification is challenging.
- Large amounts of data requirements.

# Simplex Architectures (SA)

- SA have been used in Safety critical CPS to improve its assurance.
- Integrates high performance controller with safety controllers and arbitrates between them.
- Arbitration techniques: (1) Linear Matrix Reachability, (2) Hybrid System Reachability.

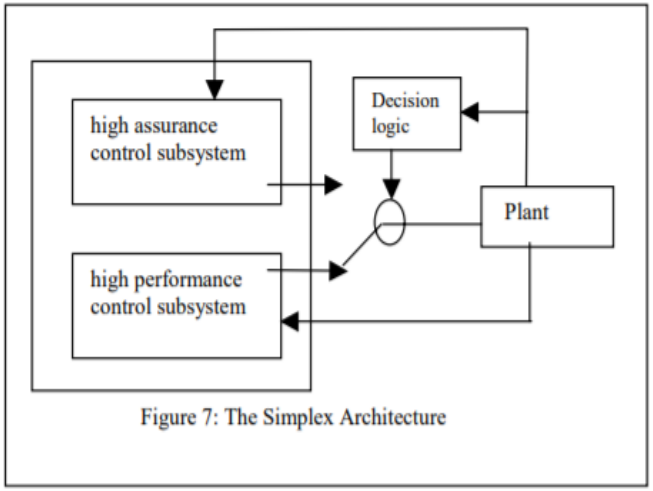


Figure 7: The Simplex Architecture

Simplex Architecture[1]

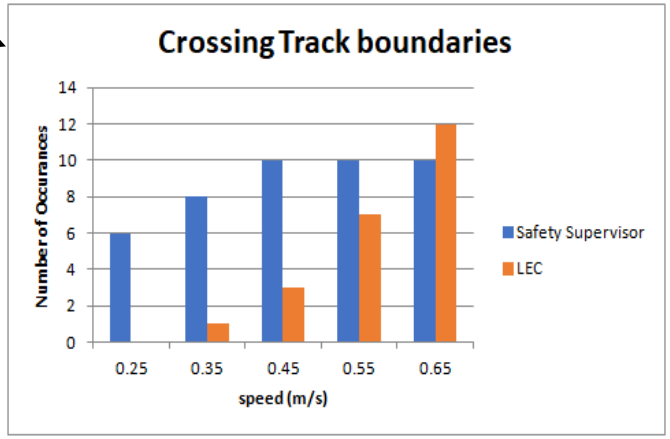
How do we combine the controllers, if we do not have a high assurance controller?

## Weighted Simplex Strategy

**Idea:** Use Weighted Ensemble approach to blend the control actions of the controllers.

$$S_C = w_1 * S_{C1} + w_2 * S_{C2} + \ldots\ldots + w_N * S_{CN}$$

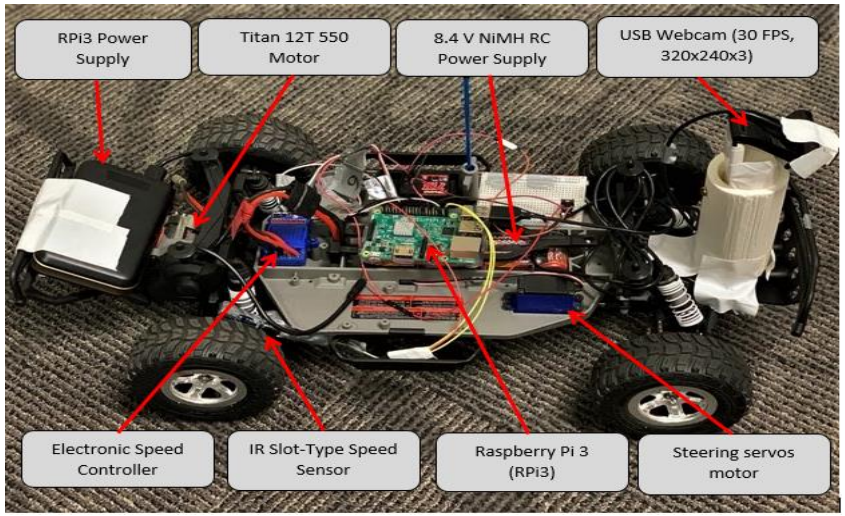
where, w1, w2 are ensemble weights, and $S_C$= control actions

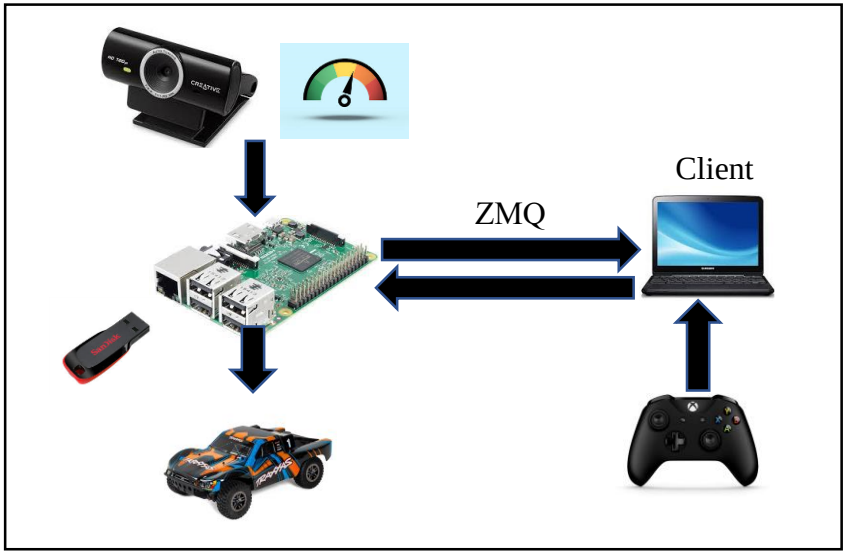



Our controllers do not have high safety assurance

[1] Sha, Lui. "Using simplicity to control complexity." *IEEE Software* 4 (2001): 20-28.

# DeepNNCar: Low cost Autonomous Testbed




DeepNNCar wirelessly connected to Fog node

**Sensors:**

(1) Camera capturing images @ 30 FPS with a resolution (320 x 240 x 3).

(2) (2) Slot type IR Opto-coupler for RPM measurement.

**Computing Unit:**

Raspberry Pi 3B

**Operating modes:**

(1) Data collection, (2) Livestream, and (3) Autonomous.

**Acceleration Protocols:**

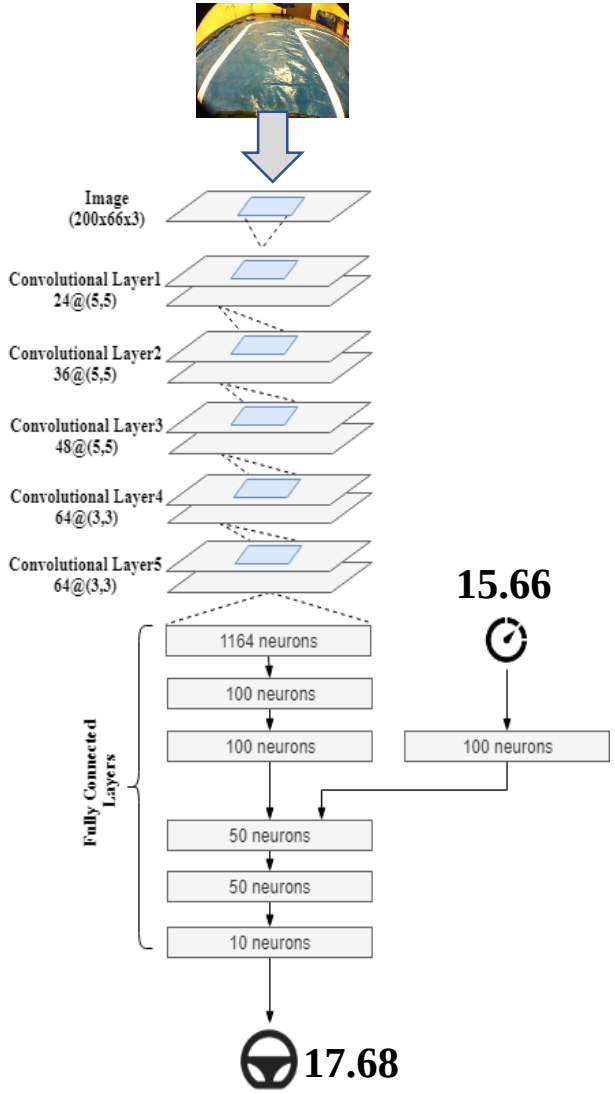
- Default-controlled using Xbox controller.
- Constant throttle.
- Constant throttle adjusted for steering.
- PID controlled cruise control.

**Safety Algorithms:**

- Lane Detection
- Blur detection

github: https://github.com/scope-lab-vu/deep-nn-car

# CNN for Steering DeepNNCar




**Model:**

- Modified NVIDIA's DAVE-II[1] Convolutional Neural Network (CNN) to predict steering.
- **Inputs:** Images (200 x 66 x3) and speed (PWM)
- **Output:** Continuous Steering (PWM)

**Training:**

- 6000 images collected from different tracks.
- Batch size = 128, Epochs = 200, Learning rate = 0.0001

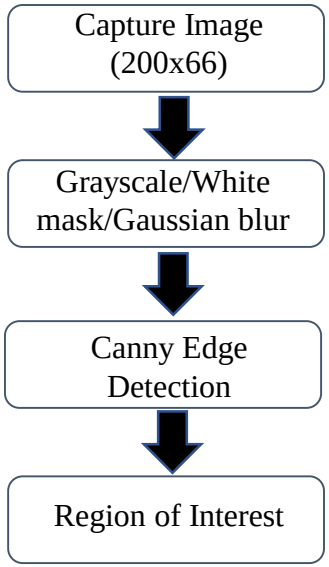

[1] Bojarski, Mariusz, et al. "End to end learning for self-driving cars." *arXiv preprint arXiv:1604.07316* (2016).

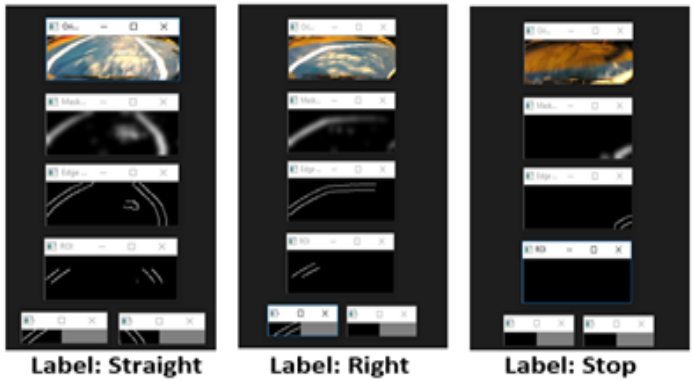
# Safety Supervisor (SS)

- **Lane Detection (LD) algorithm**: OpenCV based image processing to detect the track segments. (straight, left, right, out)
- Track Segment ⟶ Discreet Steering[1]
- LD performs with 89.6% accuracy (tested on 3000 images).

| Track Segment | Discreet Steering (degrees) |
|---|---|
| Straight | 0 |
| Left | -30 |
| Right | +30 |
| Out of track | NA |

Track segment to discreet steering angle conversion



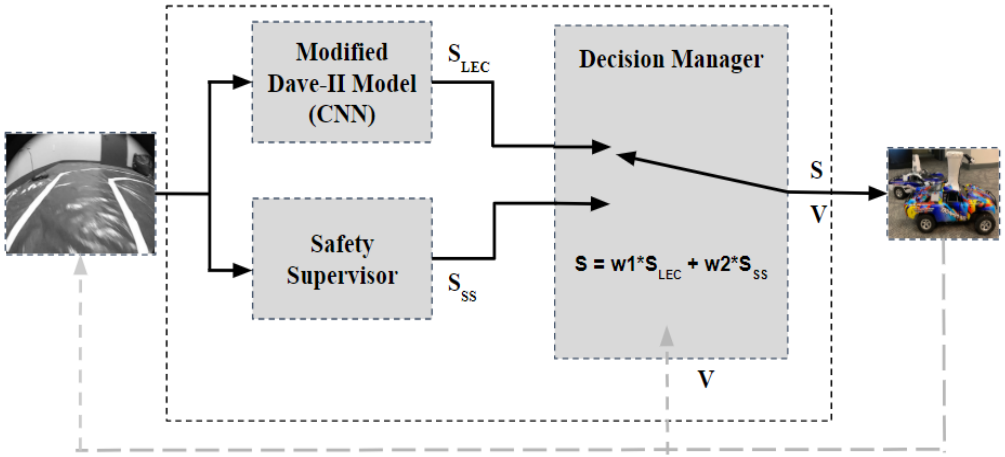




OpenCV image processing steps for Lane Detection

[1] McFall, Kevin. "Using Visual Lane Detection to control steering in a self-driving vehicle." *Smart City 360°*. Springer, Cham, 2016. 861-873.

# Blending Control Actions for DeepNNCar

**Weighted Simplex Strategy for DeepNNCar**




- The steering computation:

**Steering = $w_{LEC}$*$S_{LEC}$ + $w_{SS}$*$S_{SS}$**

- Speed (V) computation:

$$V = V \pm \Delta V$$

How do we compute the weights?

Can we include domain knowledge (like track segments) in performing the weights calculation?

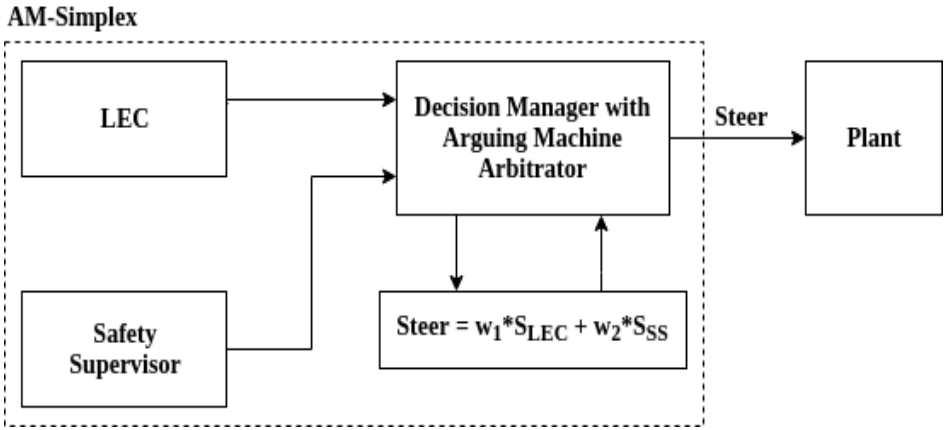
Also, Literatures from Biswas, Gautham et al.[1] and Yau, Stephen S., et al.[2], Nascimento, Nathalia, et al. have addressed the importance of using context-sensitive information in different data-driven applications.


[1] Biswas, Gautam, et al. "An approach to mode and anomaly detection with spacecraft telemetry data." *International Journal of Prognostics and Health Management* (2016).

[2] Yau, Stephen S., et al. "Reconfigurable context-sensitive middleware for pervasive computing." *IEEE Pervasive Computing* 1.3 (2002): 33-40.

# Simple Weighted Simplex Strategy (SW-Simplex)

- **Idea:** Arguing Machines introduced by Fridman et al[1].
- The control action is decided on the argument: $|S_{LEC} - S_{SS}|$.
- If $|S_{LEC} - S_{SS}| > T_{ARG}$ (preset threshold):
  Steer = $w_1 * S_{LEC} + w_2 * S_{SS}$.
  Speed - = $V_{AM}$
- If $|S_{LEC} - S_{SS}| <= T_{ARG}$
  Steer = $S_{LEC}$
  Speed += $V_{AM}$
- Weights: $W_1 = 0.8$, $W_2 = 0.2$ (after several test runs)



Arguing Machine for DeepNNCar

- **Parameter selection:** trial and error experimental runs.
- **Questions:** Can the weights dynamically change? Can we automate the parameter tuning process?

[1] Fridman, Lex, Benedikt Jenik, and Bryan Reimer. "Arguing machines: Perception control system redundancy and edge case discovery in real-world autonomous driving." *arXiv preprint arXiv:1710.04459* (2017).

# Context-Sensitive Weighted Simplex (CSW-Simplex)

- **Reinforcement Learning (RL)**[1] to find context-sensitive dynamically changing weights.
- RL is a type of dynamic programming that trains algorithms using a system of reward and punishment.
- Specifically we use Q-learning[2] algorithm.

**Important:** Designing a reward function to: (1) avoid moving off track, and (2) optimize speed.

**Q-learning Specifics:**

**States: (V,w,1-w)** where,
- V is the velocity of the car, V ϵ [15.58,15.62]
- w is the weight, w ϵ [0,1]
- The number of states in our case was 102

| action space | $\uparrow V$ *by* 0.01 | $\downarrow V$ *by* 0.01 | NOP |
|---|---|---|---|
| $\uparrow W_L$ *by* 0.05 | (0.55,0.45,15.86) | (0.55,0.45,15.84) | (0.55,0.45,15.85) |
| $\downarrow W_L$ *by* 0.05 | (0.45,0.55,15.86) | (0.45,0.55,15.84) | (0.45,0.55,15.85) |
| NOP | (0.50,0.50,15.86) | (0.50,0.50,15.84) | (0.50,0.50,15.85) |

Sample set of actions the car can perform in each state

**Actions: (ΔV,Δw,1-Δw)** where,
- **Δ**V is the change in the velocity (possible **Δ**V ϵ [0,0.010,-0.010])
- **Δ**w is the change in the weights (possible **Δ**V ϵ [0,0.005,-0.05])

# Q-Learning Setup

**Reward:** Formulated to account for speed and deviation from track center.

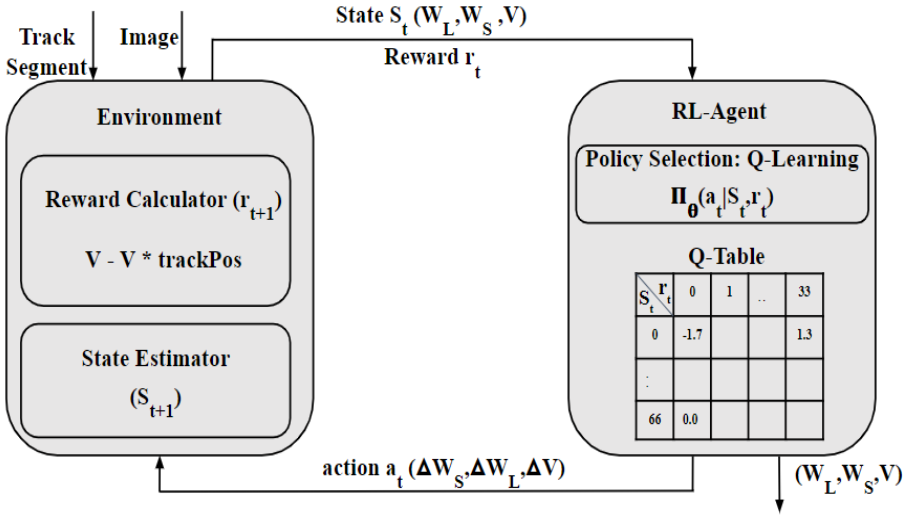
$$R(s_t, a_t) = V - V * trackPos$$

where, $R(s_t, a_t)$: reward.
Vt: velocity of the car.
trackPos: scalar value based on position of the car.

- Q-value for (state, action) is computed using Bellman equation.

$$Q_{new}(s_t, a_t) = Q(s_t, a_t) + \alpha[R(s_t, a_t) + \gamma * maxQ'(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$$

Where, $\alpha$ = learning rate (0.1), $\gamma$ = discount factor (0.4)



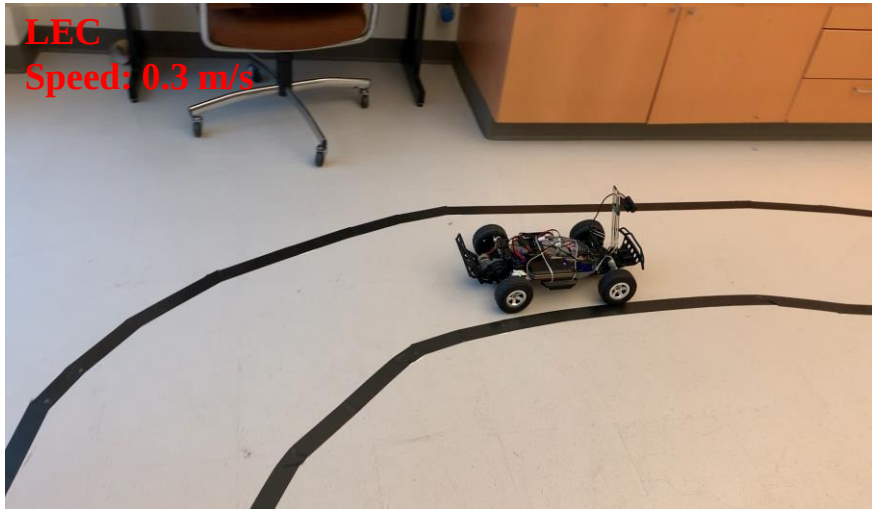
Agent-Environment interaction in q-learning

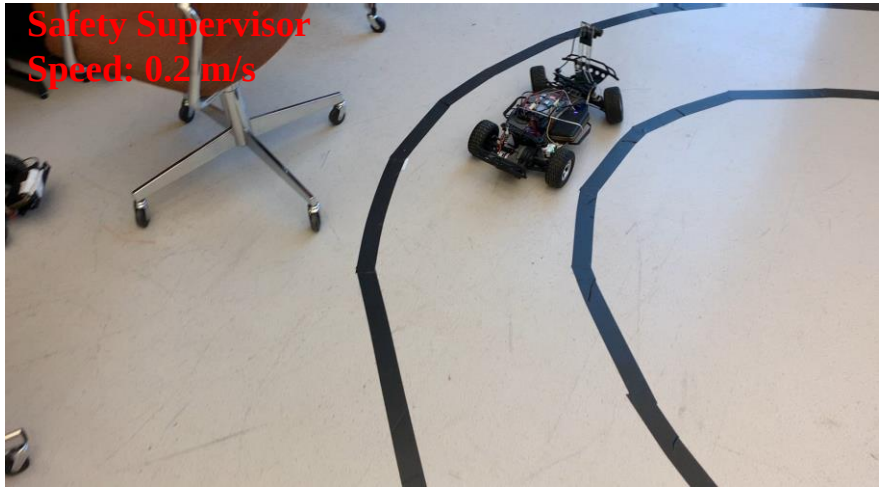
**Exploration:** Stores state, action and q-value in the Q-table. (1000 steps)

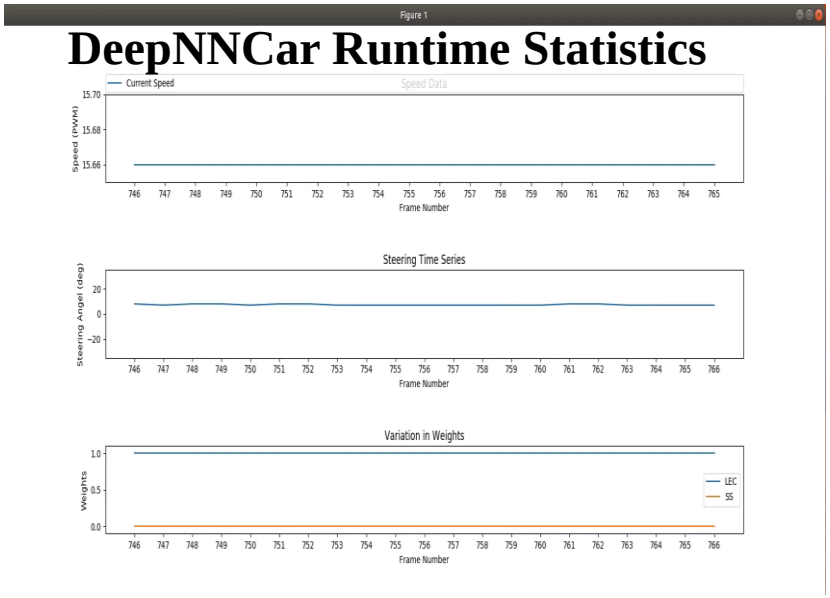
**Exploitation:** RL-Agent uses the Q-table to choose the actions.

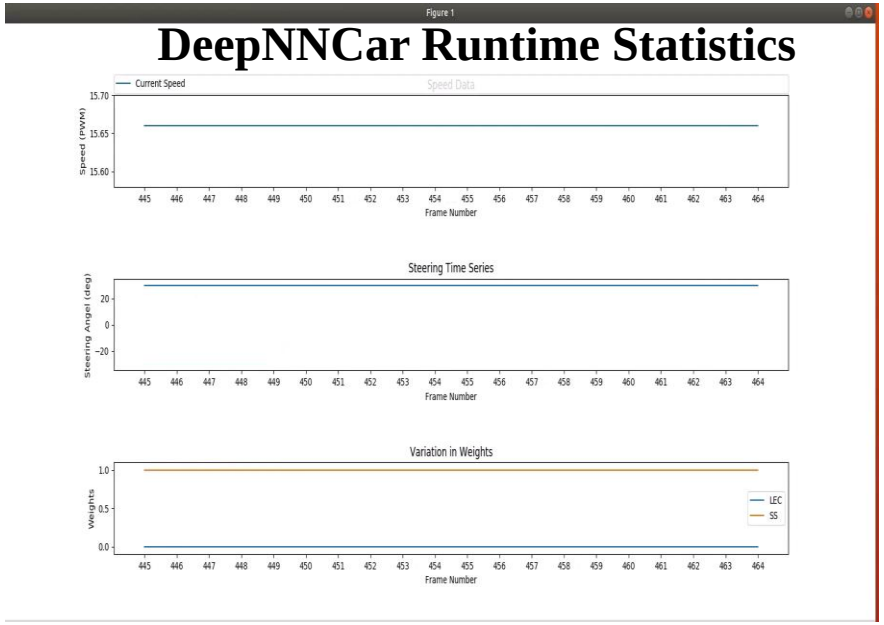
- Designing RL reward function is important to learn driving policy.
- Any RL algorithm can be used to find the weights

# LEC & Safety Supervisor



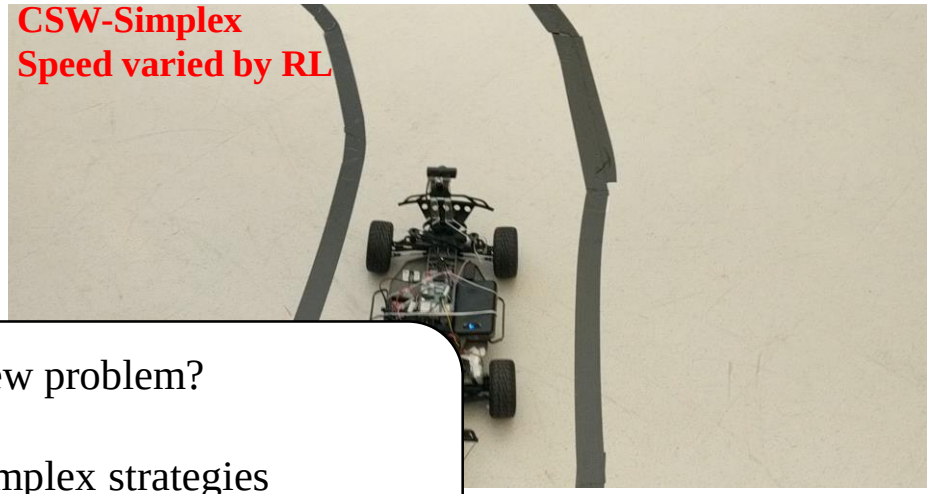



# SW-Simplex & CSW-Simplex

**SW-Simplex**
**Speed varied in fixed steps**

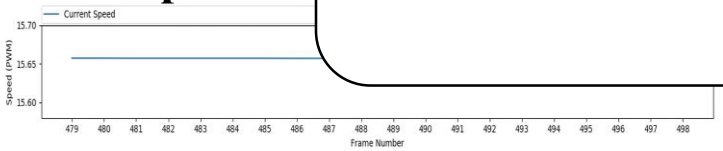
**CSW-Simplex**
**Speed varied by RL**

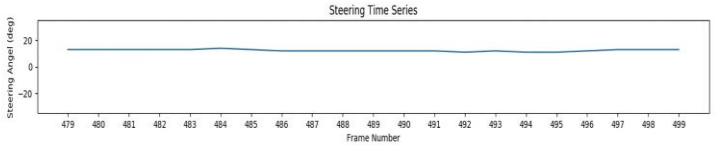


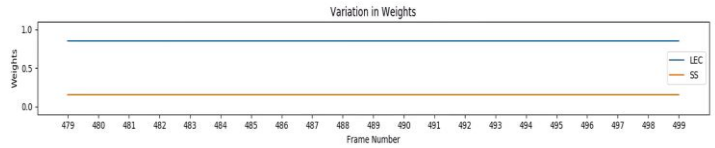
Did we create a new problem?

The complex computations of simplex strategies increases the core temperature and CPU utilization of RPi3.

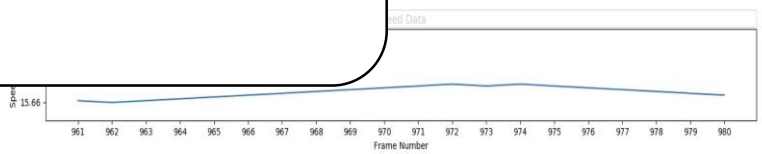
**DeepNNCa**

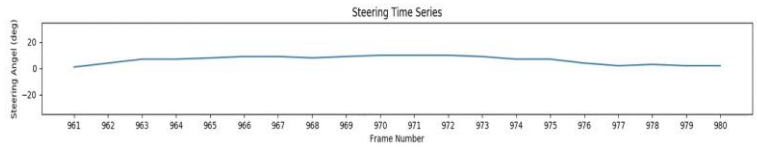


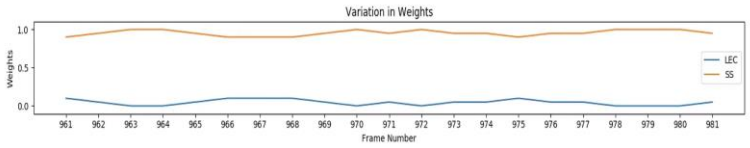




**Runtime Statistics**

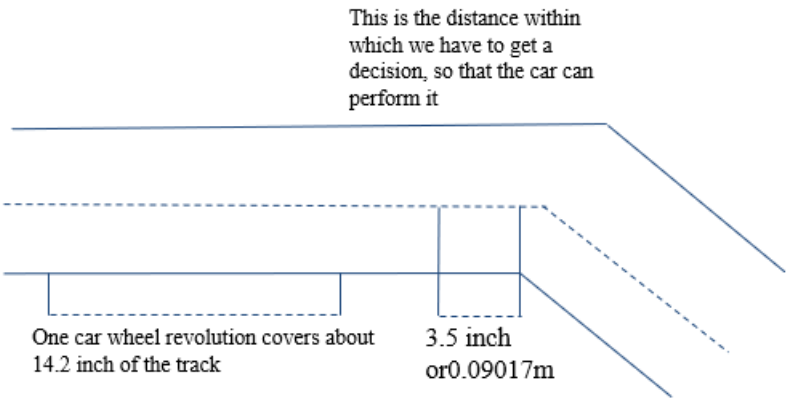


# Resource Manager and Task Offloader

- Complex computations of simplex strategies increases the temperature and cpu utilization of RPi3.
- Continuously monitor the RPi3 temperature.
- Offload tasks to maintain temperature within limits (70°c).
- Task Offloader: selects the fog with least latency.
- To compensate for increased pipeline times, top speed of the




Top speed calculated as a function of added latency + Position of car on track.

## Fog Node Selection

| Fog Node1 (Latency ms) | Fog Node2 (Latency ms) | Fog Node1 (Average Latency ms) | Fog Node2 (Average Latency ms) | Fog Node Selected |
|---|---|---|---|---|
| 13 | 10.8 | 11.87 | 12.17 | Fog Node1 |
| 11.4 | 11.7 | | | |
| 11.2 | 14.0 | | | |
| 11.7 | 11.4 | 11.23 | 10.77 | Fog Node2 |
| 11.9 | 10.6 | | | |
| 10.1 | 10.3 | | | |

Continuous background pings to available fog nodes using iperf every 30 seconds.

## Task movement as temperature varied

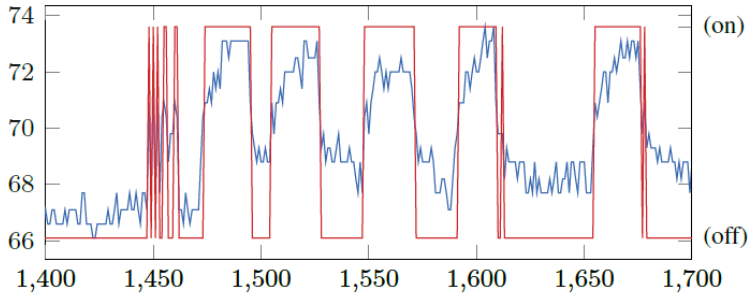


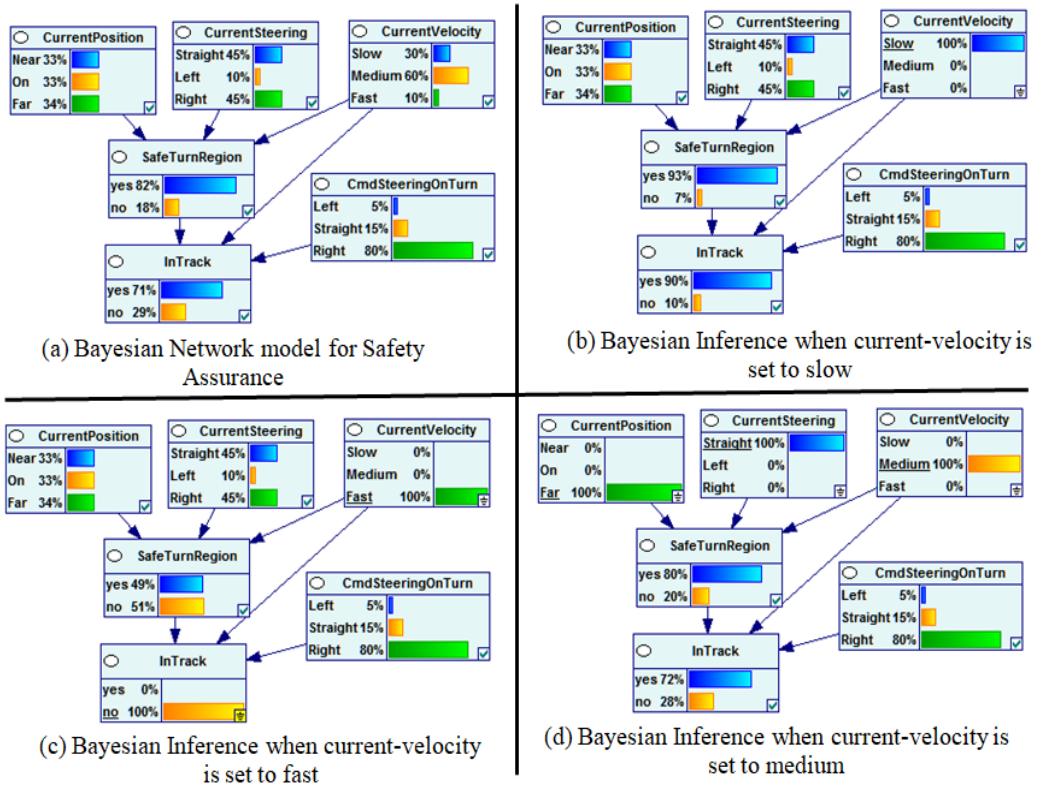

Fig. 12: The effect of offloading the tasks in response to high temperature per iteration of the inference pipeline. The trigger to offload the task is 70°C. The blue line shows the temperature in Celsius. The red line shows when the tasks were offloaded to the fog (on=on fog, off=off fog). The graph shows a subset of iterations (total 10000 iterations).

# System Level Confidence Estimation

- To **monitor the system actions and provide a level of confidence** about its safety in different scenarios.
- Bayesian network model to learn a probability distribution about the different scenarios leading to safety of the car.

We use:

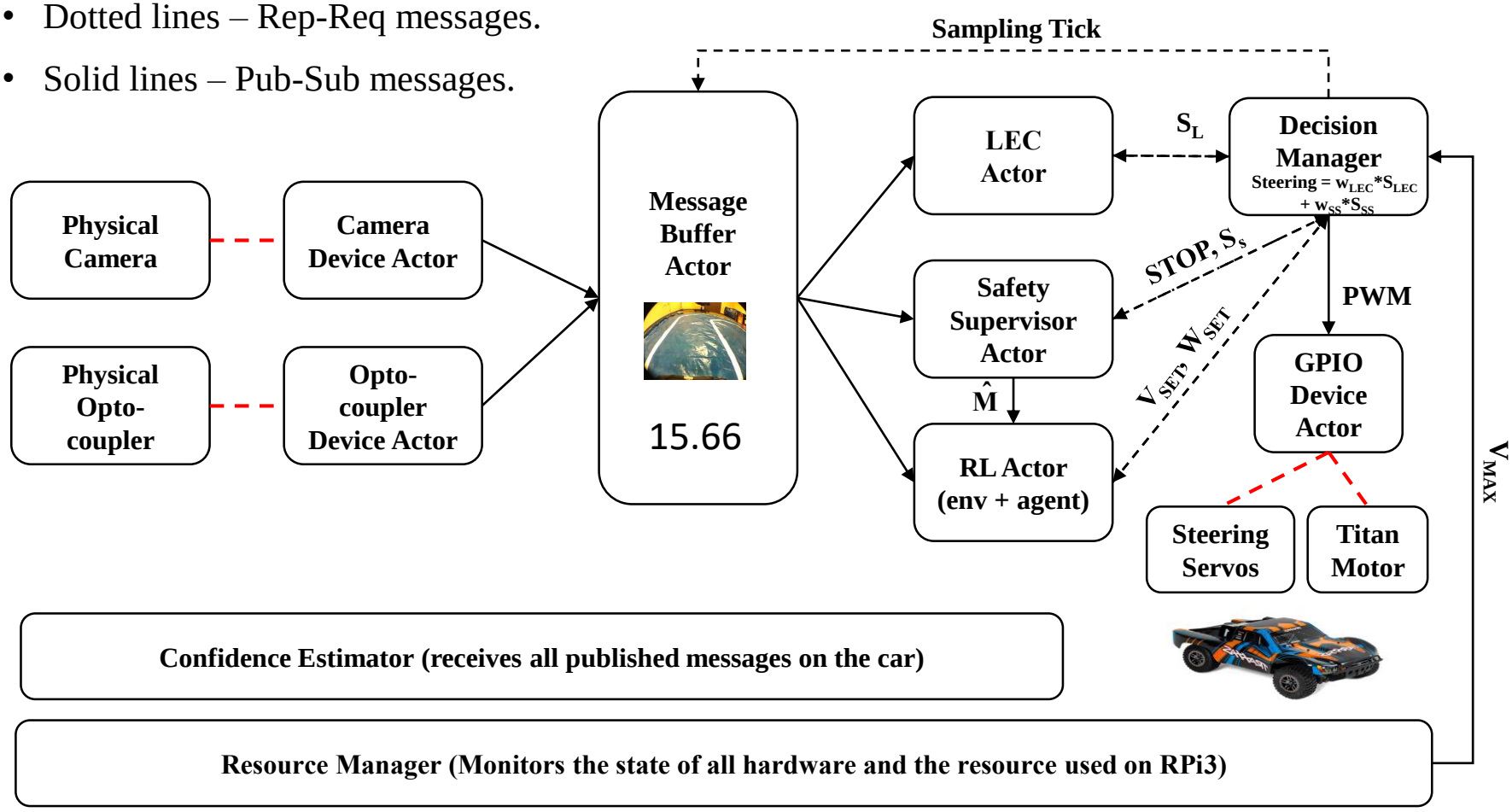
- Car state information: (current-steering, current-velocity, current-position)
- SafeTurnRegion: Calculated from the track, it captures the car being in the non-critical region when it makes the steering decision.




The probability of car remaining on track is computed using current position, velocity and steering action.

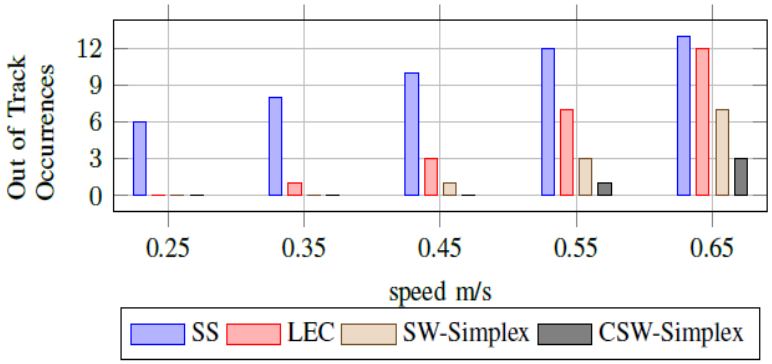
# System Integration

- These components communicate using different messaging patterns of ZeroMQ.
- Red lines – Physical connections.
- Dotted lines – Rep-Req messages.
- Solid lines – Pub-Sub messages.

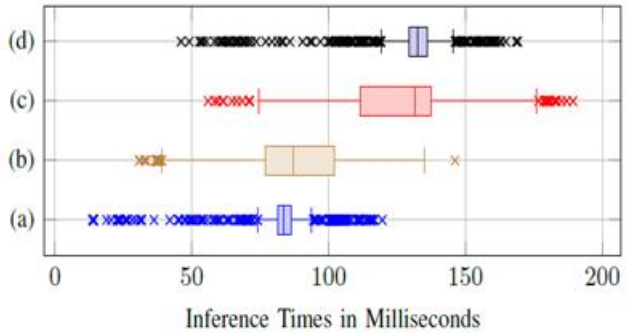


# Performance of weighted simplex strategies

Indoor track designed in our laboratory.

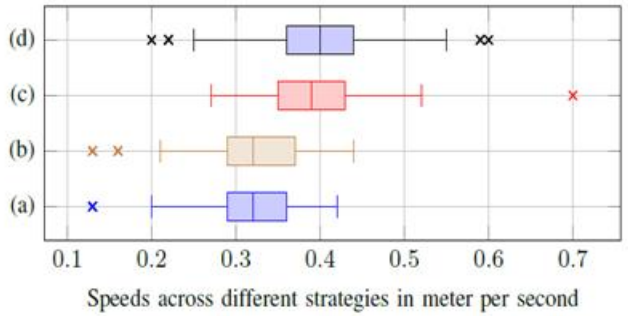



SW-Simplex and CSW-Simplex reduce safety violations by 40% and 60%.




Inference times in milliseconds of (a) Safety Supervisor: driving only with the safety supervisor, (b) LEC: driving only with the modified Dave-II model, (c) SW-Simplex, and (d) CSW-Simplex

SW-Simplex and CSW-Simplex increases the inference pipeline time, due to increased simplex computations.

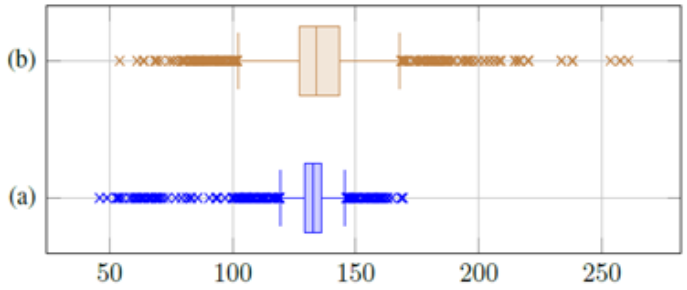



Speeds in meter per second of (a) Safety Supervisor: Driving only with the safety supervisor, (b) LEC: Driving only with the modified Dave-II model, (c) SW-Simplex, and (d) CSW-Simplex

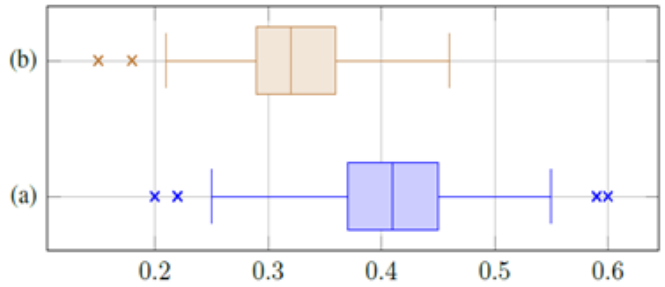
SW-Simplex and CSW-Simplex optimizes speed (~0.45 m/s) on indoor tracks.

# What happens when tasks get offloaded?

## Impact of Latency on speed and inference time




Inference times in milliseconds (a) CSW-Simplex with all tasks executed onboard (b) CSW-Simplex with RL-Manager offloaded.

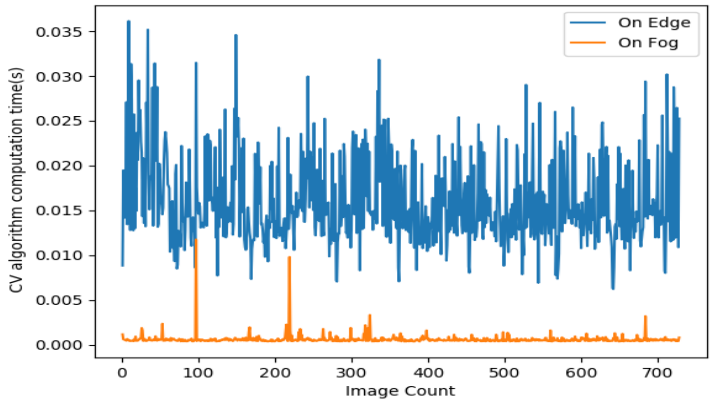



Speed readjustment during offload (m/s) (a) CSW-Simplex with all tasks executed onboard (b) CSW-Simplex with RL-Manager offloaded.

- The impact of added latency reflects as increased pipeline times.

$$T_{Total} = T_{onboard} + T_{offload\_roundtrip}$$

$T_{onboard}$: time for onboard task computation
$T_{offload\_roundtrip}$: round trip time for offloaded tasks




- Top speed is saturated to compensate for increased pipeline time.
- $V_{MAX} = F(T_{Total})$, top speed is saturated based on the latency.

- Computations on the fog are ~ 27 times faster.
- However, the varying latency could cause a problem to the safety of the car.

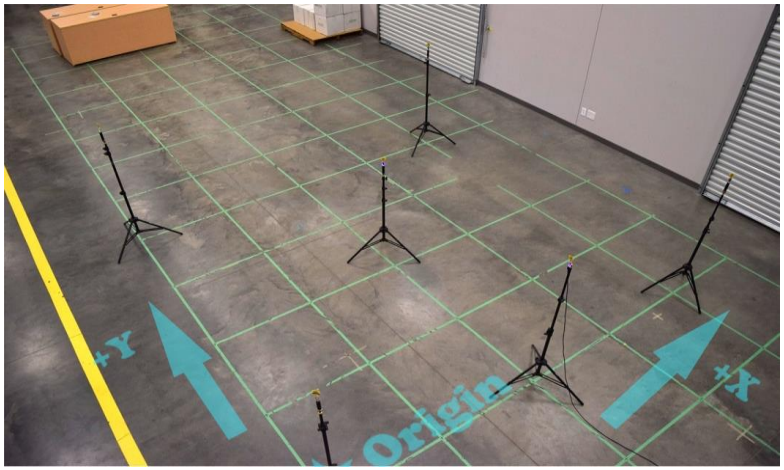
# Summary

Middleware Framework

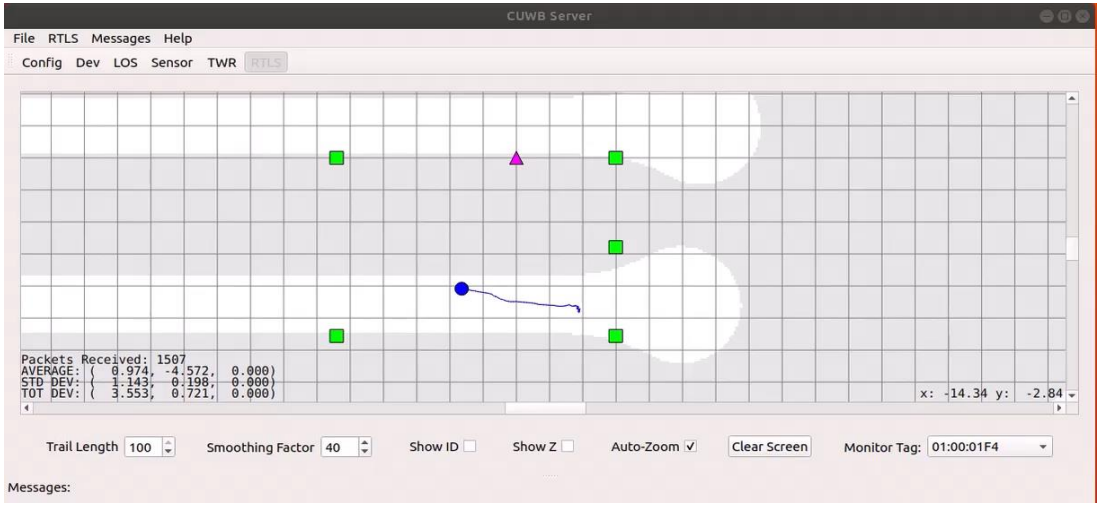
1. We blend the control actions of controllers using Weighted Simplex Strategies, to improve the systems safety guarantees and optimize its performance.
2. We describe a resource management and task offloading scheme to move tasks at runtime to fog nodes.
3. We introduce a system level confidence estimator to assess the confidence of the robot systems current action

# Ongoing Work

- We are currently extending our work to multi-car platoon setup.
- Ciholas ultra-wideband sensors (CUWB) are used to get the position of the cars at runtime.
- The cars use safety contracts to communicate safe distance information, and the follower cars speed is saturated based on the distance between cars.




Ultra-wideband sensors can be placed around the tracks. Masters, Anchors, and tags communicated to provide Real-time position.




Ultra-wideband sensors attached to a single car.
We get the Real-time position.

https://cuwb.io/docs/v2.0/overview/