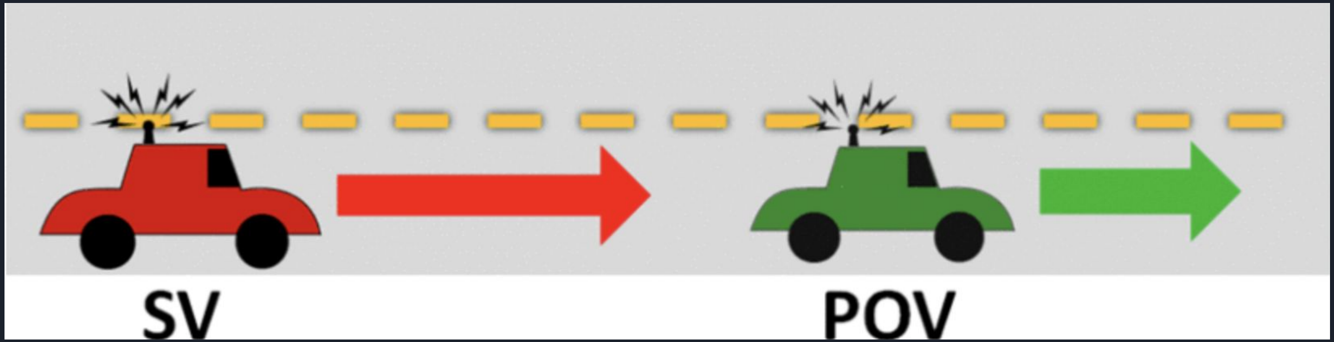
Verified Cruise Control on RC Vehicle

Shashank Ojha and Yufei Wang

Objective

- Implement a verified model (static POV system) on real hardware
- Fill the gap between theory & practice



Motivation

- Cruise Control system is useful in practice:
    - A stepping-stone towards self-driving cars
    - Long straight highway trucking

Summary of Deliverables

- Formal model and proof of system in KeYmaera X
- Implementation of model on an RC vehicle
- Video and Live Demos

Formal Model and Proof

Assumptions

- One-dimensional road
- Static Obstacle
- Constant accelerate with rates $acc = \{A, 0, -B\}$
- LIDAR sensor measures the obstacle distance
- ODOM sensor measures the car's velocity
- Asynchronous read from the sensors & control

Formal Model

Estimate Obstacle Distance

```
if (in_sensor_range(obstacle)):
    sensed_dist = LIDAR_reading
else:
    sensed_dist = sensor_range
```

Estimate Vehicle Velocity

```
sensed_vel = ODOM_reading
```

Control Decision

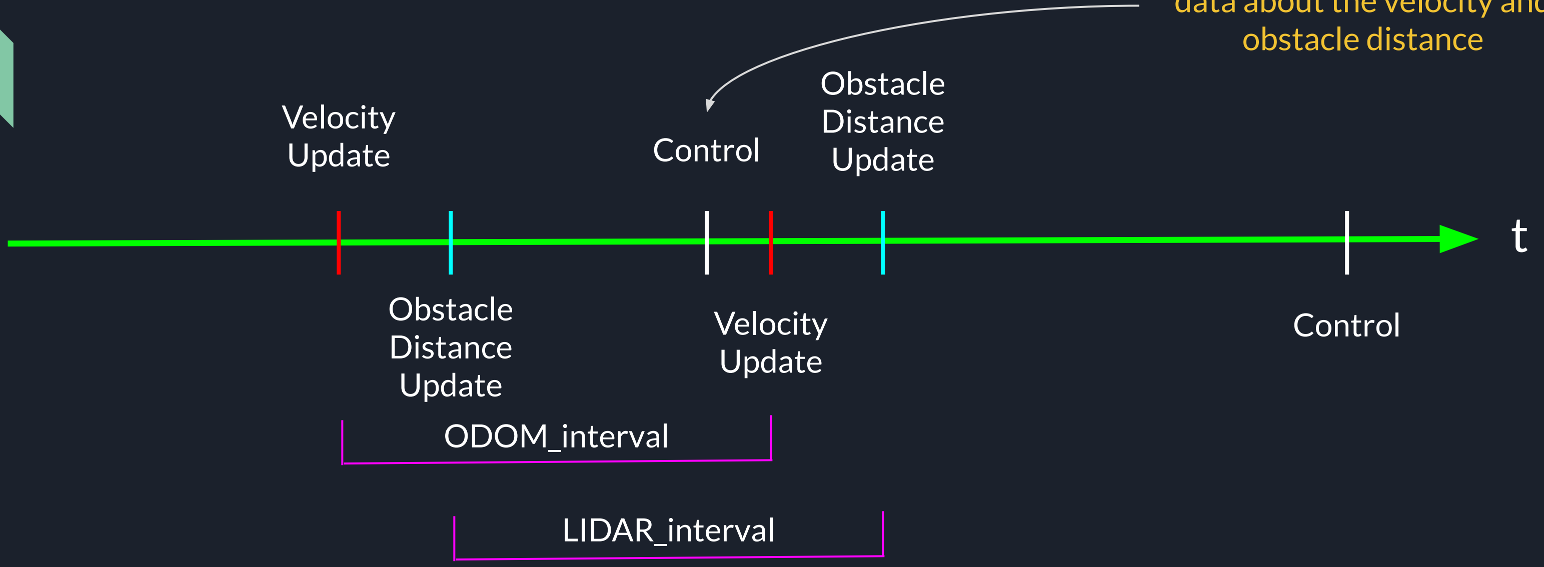
```
if (safe(A)):
    acc = A
elif (safe(0.0)):
    acc = 0.0
else:
    acc = -B
```

Dynamics

```
{ obstacle_dist′ = -v, v′ = acc, t′ = 1 & v ≥ 0 & t ≤ CTRL_T }
```

Timeline of Events

Control must make a decision based on ***stale*** data about the velocity and obstacle distance




```
ub_v = sensed_vel + A * ODOM_interval

lb_obstacle_distance =
sensed_distance - (ub_v * LIDAR_interval + 0.5 * A * LIDAR_interval^2)
```

Safety Condition

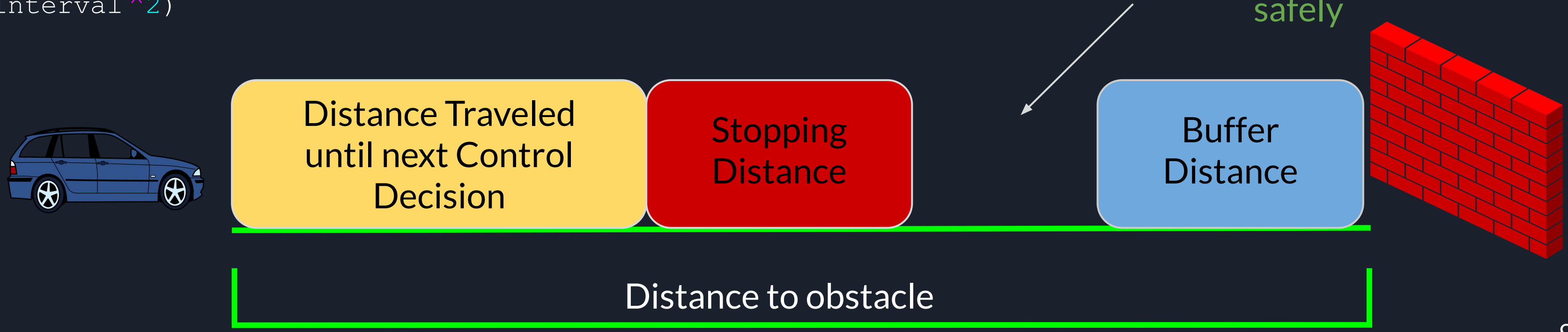
Safety condition is based on ***sensed parameters***, NOT the true values

```
def safe(a) : lb_obstacle_distance >= ub_v * CTRL_T + 0.5 * a * CTRL_T^2
                                     + (ub_v + a * CTRL_T) / (2 * B)
                                     + (BUFFER_DIST)
```

```
ub_v = sensed_vel + A * ODOM_interval

lb_obstacle_distance =
sensed_distance - (ub_v * LIDAR_interval + 0.5 * A * LIDAR_interval ^2)
```

We must have distance left over in order to accelerate safely



Implementation

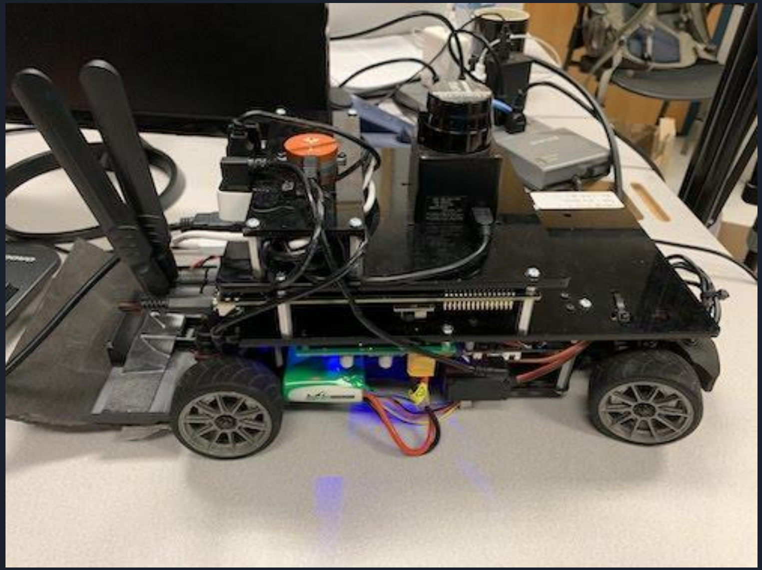
RC Vehicle

Hardware:

- **LIDAR sensor:** *5.6m range, 10Hz*
- **ODOM sensor:** *30Hz*
- **Max velocity:** *6m/s*

Software:

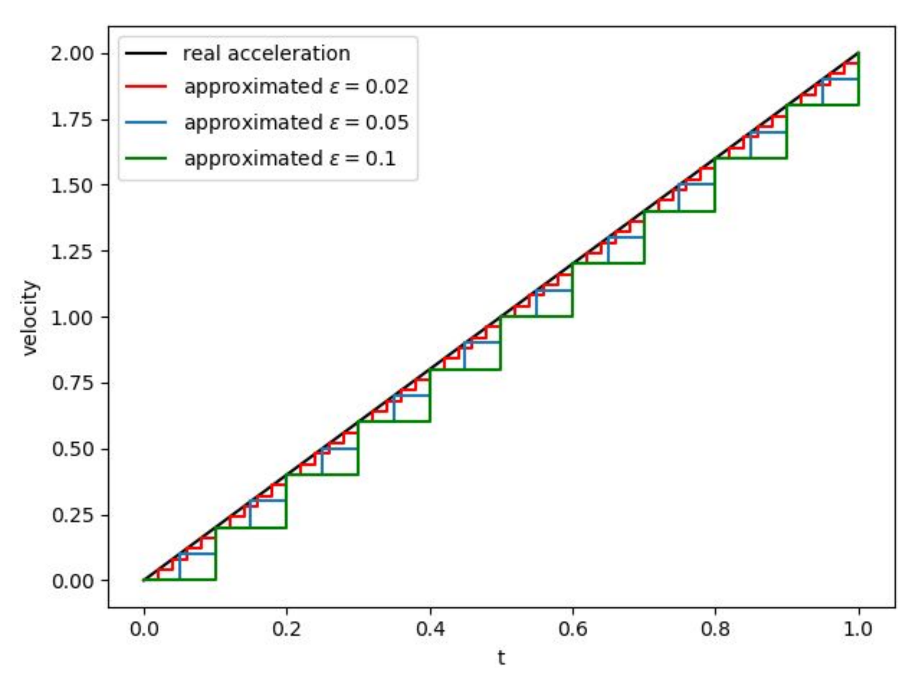
- ROS
- Subscribe to get sensor data
- Publish to command velocity



Implementation Challenges

- Cannot command acceleration directly
  - Approximate acceleration control by velocity control

V_command = V_old_commanded + A * ε



Implementation Challenges

- Very noisy ODOM sensor: imprecise *V_odom*
    - Maintain an analytic velocity *V_command*
    - Use *max(V_command, V_odom)* to upper bound the real velocity
- Steering linkage was also damaged
    - Manually adjust for bias with software

Live Demo

Challenges Ahead

- Move from static obstacle to dynamic obstacle model (need another car)
    - Need to approximate POV’s velocity
- Update hardware: ODOM sensor, direct acceleration control
- Incorporate feedback from sensors to lower the disparity between commanded controls and actual dynamics
- Model other dynamics such as drag and friction forces

Huge Thanks to

- André Platzer
- Katherine Cordwell
- Aman Khurana