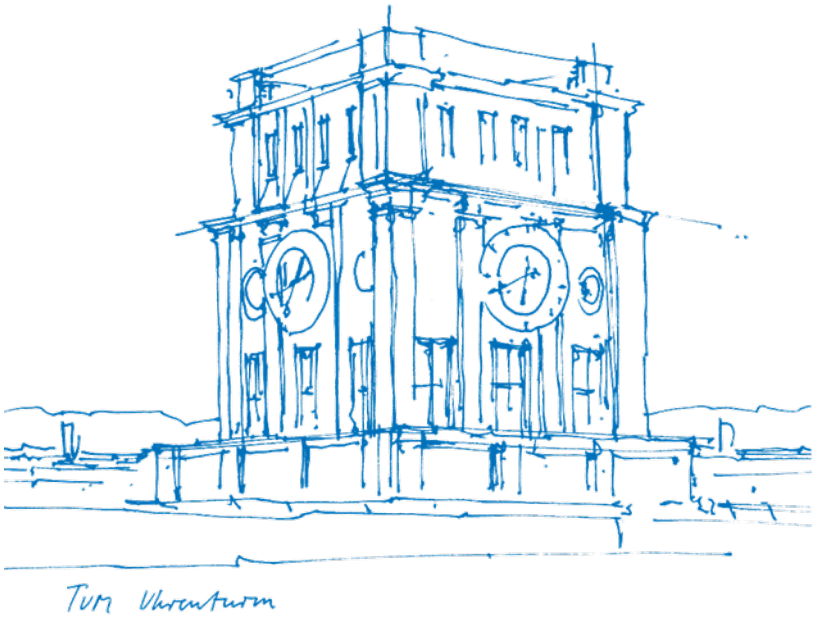
Chair of Business Information Systems
Department of Informatics
Technische Universität München

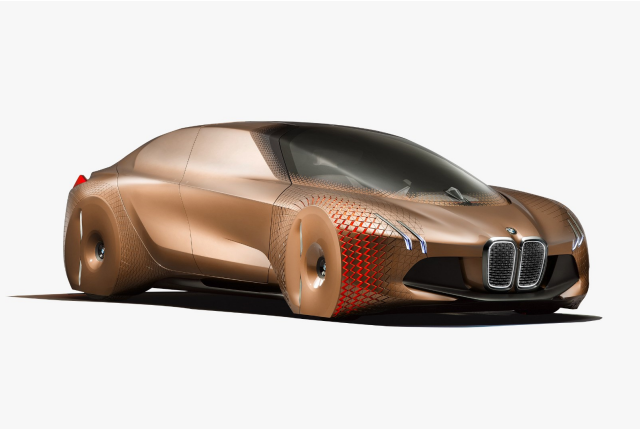
Interactive Front-End for EV Traffic Simulation in Highways

Adrian Thiesen, Martin Wauligmann
Technische Universität München
Department of Informatics
Chair of Business Information Systems
Munich, 18 January 2017



Electric vehicles and their challenges

“In order to have clean air in cities, you have to go electric.” – Elon Musk, MIT’s Aero/Astro Centennial



Electric vehicles are the future

- Electric vehicles are increasing
- Tesla leading company

One big problem they all face

- limited battery capacity
- a lot more recharges needed than regular “petrol” vehicles
- vehicles need efficient charging managment and an EV Driver Interface

Smart scheduling approach for EVs

- **paper:** “Smart Charging Schedules for Highway Travel with Electric Vehicles”
  - authors: Victor del Razo and Hans-Arno Jacobsen
- **idea:** EVs determine their charging stops during a highway trip
- **goal:** reduce the total travel time for each EV
- **summary:** shortest path problem
  - A* search algorithm
  - extended with verification of constraints
- **software:** Python based simulation framework that provides
  - generated trip data
  - time-dependent parameters

Smart scheduling approach for EVs

- **simulation model**
  - electric vehicles (EVs)
  - charging stations (CSs)
  - highway
- **scheduling design**
  - local to the EV
  - communication with charging stations
  - highway-related information system
- **scheduling process**
  - calculate set of charging stops and times
  - submit bookings to the charging stations
  - proceed trip as planned unless an update event is received

Interactive Front-Ends

Our task was to design and implement two front-ends for the simulation framework.

- **Simulation Manager Interface**
  - show current states of EVs and CSs
- **EV Driver Interface**
  - show relevant vehicle information
  - display travel-related information

Research question

What is the most suitable form of presentation for the data that is most relevant during the simulation and while driving respectively?

- **Simulation Manager Interface**
  - data-heavy application
  - structured data access
  - relation between EVs and CSs
  - schedule changes
  - aggregated metrics

- **EV Driver Interface**
  - limited user attention
  - separation of information
  - time-relevant data

Which tools, libraries, frameworks or APIs can be used to implement the two front-ends? Which are most suitable for our purpose?

Requirements EV Driver Interface

- **Performance**
  - real time data without delay
  - if driver's attention is needed reduce distraction to a minimum
- **Functional**
  - functionality of the UI must be verified through measurment or testing
  - UI missfunction etc. can lead to sevear damage e.g. missleading route
- **Design**
  - vehicles self status e.g. battery status at static position
  - trip information like map components can be dynamic
  - $\rightarrow$ simplistic desgin to avoid driver distraction and workload

Requirements Simulation Manager Interface

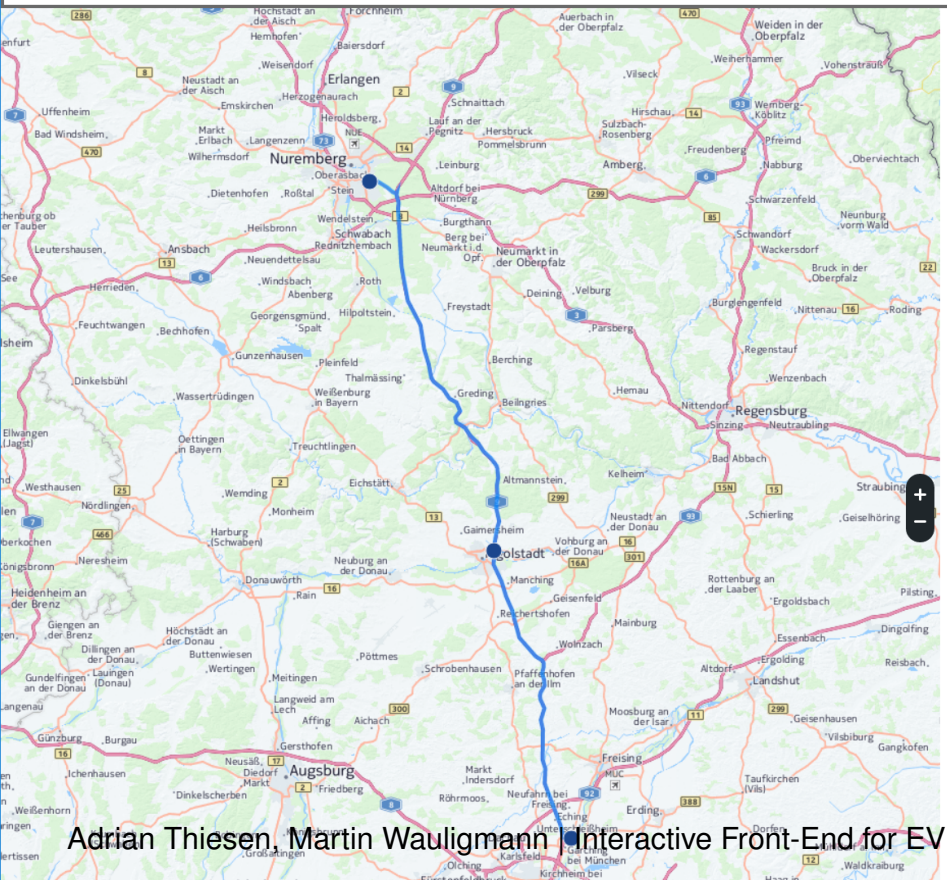
- **Simulation Manager Interface**
  - most important data first in sight e.g. charging stations, EV's
  - detailed information hidden on first sight $\rightarrow$ reduce complexity
  - real time data without much delay

EV Driver Interface

Here JavaScript API

Charging status: charging
Battery level: 1
Discharging time: Infinity

Stadion Nürnberg | Get directions



- Stop: charging station Aral Tankstelle Christiane Wentzlaff - Ziel: Stadion Nürnberg

1. Head toward Boltzmannstraße. Go for 182 m.
2. Turn right onto Boltzmannstraße. Go for 102 m.
3. Continue on Ludwig-Prandtl-Straße. Go for 1.5 km.
4. Turn left and take ramp onto A9 toward Nürnberg/Flughafen München. Go for 61.5 km.
5. Take exit 61 toward Ingolstadt-Nord/Eichstätt onto B16A. Go for 505 m.
6. Turn right onto Römerstraße toward München/A9/Eichstätt/Neuburg a. d. D./Ingolstadt-Nord/Klinikum. Go for 320 m.
7. Turn right onto Römerstraße. Go for 28 m.
8. Arrive at Römerstraße. Your destination is on the left.
9. Head west on Römerstraße. Go for 28 m.
10. Turn right onto Römerstraße. Go for 66 m.
11. Make a U-Turn at Schollstraße onto Römerstraße. Go for 374 m.
12. Turn left onto B16A toward Gewerbegebiet Nord-Ost/Ingolstadt Village/Nürnberg/A9/B16a/Großmehring/Riedenburg. Go for 312 m.
13. Take ramp onto A9 toward Nürnberg. Go for 22.8 km.
14. Keep left onto A9. Go for 4.7 km.
15. Keep left onto A9. Go for 51.0 km.
16. Take exit 52 toward Nürnberg-Fischbach/Nürnberg-Zentrum/Nürnberg-Messe/Nürnberg-Stadion. Go for 328 m.
17. Continue on B4. Go for 5.6 km.
18. Turn left onto Hans-Kalb-Straße toward Messe/Stadion ARENA. Go for 270 m.
19. Turn right onto Zeppelinstraße. Go for 118 m.
20. Turn left onto Norisring. Go for 87 m.
21. Continue on Karl-Steigelmann-Straße. Go for 402 m.
22. Arrive at Karl-Steigelmann-Straße.

Distance to charging Station: 64.102 km.
Total distance: 150.174 km.
Travel Time: 91 minutes 32 seconds. (in current traffic)

Here API vs Google Maps API comparison

Which API is most suitable for the EV Driver interface?

- **Google Maps API**
  - Turn-by-Turn Navigation is not possible
  - Offline use is limited

- **Here API**
  - Developer friendly (full control of maps)
  - Turn-by-Turn Navigation even offline
  - $\rightarrow$ more suitable

main.js

```
function calculateRouteFromAtoB(platform) {
    var router = platform.getRoutingService(),
    routeRequestParams = {
        mode: 'fastest;car',
        representation: 'display',
        routeattributes: 'waypoints,summary,shape,legs',
        maneuverattributes: 'direction,action',
        waypoint0: '48.2626,11.6679', // Brandenburg Gate
        waypoint1: '48.7752,11.4595', // Friedrichstrasse Railway Station
        waypoint2: '49.4268,11.1255' // Stadion Nuremberg
    };

    router.calculateRoute(
    routeRequestParams,
    onSuccess,
    onError
    );
}
```

Simulation Manager Interface

Google Maps JavaScript API

config.js

```
var config = {
    mapID: "map",
    mapCenter: "Germany",
    mapZoom: 7,
    markers: {
        car: {
            url: "img/markers/car.png",
            anchor: new google.maps.Point(24, 18)
        },
        battery: {
            url: "img/markers/battery.png",
            anchor: new google.maps.Point(20, 36)
        }
    }
};
```

main.js

```
$(document).ready(function () {

    // Init map
    var map = new Map();

    // Electric vehicles traveling from A to B
    var ev = [];

    ev.push(new EV(map.map, 1, 0, "Munich", "Berlin"));
    ev.push(new EV(map.map, 2, 10, "Munich", "Berlin"));
    ev.push(new EV(map.map, 3, 25, "Berlin", "Munich"));

    // Charging stations at location C
    var cs = [];

    cs.push(new CS(map.map, 1, "Ingolstadt"));
    cs.push(new CS(map.map, 2, "Nuremberg"));
    cs.push(new CS(map.map, 3, "Bayreuth"));
    cs.push(new CS(map.map, 4, "Osterfeld"));
});
```

map.js

```
function Map() {
    this.init = function () {

        // Create new Google Map
        this.map = new google.maps.Map(document.getElementById(config.mapID), {
            mapTypeId: google.maps.MapTypeId.ROADMAP
        });

        // Center and fit country in viewport
        var geocoder = new google.maps.Geocoder();
        var map = this.map;

        geocoder.geocode({'address': config.mapCenter}, function (results, status) {
            if (status == google.maps.GeocoderStatus.OK) {
                map.setCenter(results[0].geometry.location);
                map.fitBounds(results[0].geometry.viewport);
            }
        });
    };
}
```

cs.js – 1

```
function CS(map, id, location) {
    var CS = this;

    var stats = {
        id: id,
        time: '',
        queue_length: '',
        busy_poles_fc: '',
        busy_poles_tsc: '',
        arrived_cars: '',
        leaving_cars: '',
        queued_cars: '',
        plugged_cars: '',
        energy_consumed: '',
        energy_produced: '',
        energy_stored: '',
        energy_bought: '',
        queue_prediction_fc: '',
        queue_prediction_tsc: ''
    };
```

cs.js – 2

```
this.init = function () {
    var marker = new google.maps.Marker({
        map: map,
        icon: config.markers.battery
    });

    CS.setPosition(marker);

    var panel = new google.maps.InfoWindow({
        content: CS.getStats()
    });

    CS.initStats(panel, marker);
};
```

cs.js – 3

```
this.getStats = function () {
    var info = '';

    for (var key in stats) {
        info += key + ': ' + stats[key] + '<br>';
    }

    return info;
};
```

ev.js – 1

```
function EV(map, id, start_time, origin, destination) {
    var EV = this;

    var stats = {
        id: id,
        time: '',
        position: '',
        geo_position: '',
        distance_travelled: '',
        time_travelled: '',
        time_waited: '',
        time_charged: '',
        time_driven: '',
        battery_level: '',
        speed: '',
        driving_flag: '',
        schedule_status: ''
    };
```

ev.js – 2

```
this.start = function () {
    var directions = new google.maps.DirectionsService();

    var request = {
        origin: origin,
        destination: destination,
        travelMode: google.maps.TravelMode.DRIVING
    };

    setTimeout(function () {
        directions.route(request, function (result, status) {
            if (status == google.maps.DirectionsStatus.OK) {
                EV.autoUpdate(map, result.routes[0].legs);
            }
        });
    }, start_time * 1000);
};
```

ev.js – 3

```
this.autoUpdate = function (map, legs) {
    var route, marker, panel;

    route = new google.maps.Polyline({
        path: [],
        geodesic: true,
        strokeColor: '#000000',
        strokeOpacity: 0.8,
        strokeWeight: 2,
        editable: false,
        map: map
    });

    marker = new google.maps.Marker({
        map: map,
        icon: config.markers.car
    });

    ...
```

ev.js – 4

```
...

var timeUnit = 0;

for (var i = 0; i < legs.length; i++) {
    for (var j = 0; j < legs[i].steps.length; j++) {
        for (var k = 0; k < legs[i].steps[j].path.length; k++) {
            setTimeout(function (coords) {

                route.getPath().push(coords);
                EV.moveMarker(map, marker, coords);
                EV.updateStats(panel);

            }, 50 * timeUnit++, legs[i].steps[j].path[k]);
        }
    }
}
```

Outlook

EV Driver Interface

- evtl. integration with the backend
- dynamic routing

Simulation Manager Interface

- highlight EV schedule entries
- enhancement via visual cues
- aggregated statistics