FORCESPRO: HIGH-LEVEL INTERFACE

Uros Markovic
Optimization Specialist

2022 American Control Conference
Atlanta, GA, USA / June 2022

embotech

PROBLEM CLASS OVERVIEW

High-level interface supports (potentially) **non-convex**, finite-time **nonlinear** optimal control problems

- Provides users with a familiar easy-to-use interface for defining an optimization problem
- Gives advanced users full flexibility when importing external C-coded functions for evaluation
- Includes the class of mixed-integer nonlinear problems
- All functions involved (f_k, c_k, h_k) need to be continuously differentiable
- Supports all problem data to be parametric (unknown at code generation time)
- Supports discretization functions and integration schemes for nonlinear MPC

minimize	$\sum_{k=1}^{N-1} f_k(z_k, p_k)$	(separable objective)
subject to	$z_1(\mathcal{I}) = z_{\text{init}}$	(initial equality)
	$E_k z_{k+1} = c_k(z_k, p_k)$	(inter-stage equality)
	$z_N(\mathcal{N}) = z_{\text{final}}$	(final equality)
	$\underline{z}_k \leq z_k \leq \bar{z}_k$	(upper-lower bounds)
	$F_k z_k \in [\underline{z}_k, \bar{z}_k] \cap \mathbb{Z}$	(integer variables)
	$\underline{h}_k \leq h_k(z_k, p_k) \leq \bar{h}_k$	(nonlinear constraints)

embotech

FUNCTION EVALUATION

FORCESPRO requires **functions f, c, h** and their **derivatives (Jacobians)** to be **evaluated at each iteration**

- **Cost function** terms $\boldsymbol{f_k(z_k)}$and their gradients $\boldsymbol{\nabla f_k(z_k)}$
- **Dynamics** $\boldsymbol{c_k(z_k)}$ and their Jacobians $\boldsymbol{\nabla c_k(z_k)}$
- **Inequality constraints** $\boldsymbol{h_k(z_k)}$ and their Jacobians $\boldsymbol{\nabla h_k(z_k)}$

Code generator supports **two ways of supplying these functions**:

- **Automatic C-code generation** using supported differentiation tools (CasADi or Symbolic Math Toolbox)
- **User-provided C-functions** (source files) derived by hand or another AD tool

embotech

FUNCTION EVALUATION

FORCESPRO requires **functions f, c, h** and their **derivatives (Jacobians)** to be **evaluated at each iteration**

- **Cost function** terms $\boldsymbol{f_k(z_k)}$and their gradients $\boldsymbol{\nabla f_k(z_k)}$
- **Dynamics** $\boldsymbol{c_k(z_k)}$ and their Jacobians $\boldsymbol{\nabla c_k(z_k)}$
- **Inequality constraints** $\boldsymbol{h_k(z_k)}$ and their Jacobians $\boldsymbol{\nabla h_k(z_k)}$

Code generator supports **two ways of supplying these functions**:

- **Automatic C-code generation** using supported differentiation tools (CasADi or Symbolic Math Toolbox)
- **User-provided C-functions** (source files) derived by hand or another AD tool

Recommended: Install CasADi and use it as the **default differentiation tool** for **FORCES**PRO NLP problems

embotech

SQP SOLVER

- **FORCES**PRO real-time **sequential quadratic programming (SQP) algorithm** iteratively solves **convex quadratic approximations** of the (generally non-convex) problem

- The **solution is stored internally** in the solver and **used as an initial guess** for the next time the solver is called

- Provides **fast solve times** (compared to the **interior point method**), particularly suitable for MPC applications with
  - **Small sampling time**
  - **Limited computational power** of the hardware

```
% Select appropriate NLP solver
codeoptions.solvemethod = 'SQP_NLP'; % default setting is 'PDIP'
```

embotech

SQP SOLVER

- **FORCES**PRO real-time **sequential quadratic programming (SQP) algorithm** iteratively solves **convex quadratic approximations** of the (generally non-convex) problem

- The **solution is stored internally** in the solver and **used as an initial guess** for the next time the solver is called

- Provides **fast solve times** (compared to the **interior point method**), particularly suitable for MPC applications with
  - **Small sampling time**
  - **Limited computational power** of the hardware

Note: Currently, there are several **modeling requirements** when using the **FORCES**PRO **SQP solver**:

minimize	$\sum_{k=1}^{N-1} f_k(z_k, p_k)$	(separable objective)
subject to	$z_1(\mathcal{I}) = z_{\text{init}}$	(initial equality)
	$E_k z_{k+1} = c_k(z_k, p_k)$	(inter-stage equality)
	$z_N(\mathcal{N}) = z_{\text{final}}$	(final equality)
	$\underline{z}_k \leq z_k \leq \bar{z}_k$	(upper-lower bounds)
	$F_k z_k \in [\underline{z}_k, \bar{z}_k] \cap \mathbb{Z}$	(integer variables)
	$\underline{h}_k \leq h_k(z_k, p_k) \leq \bar{h}_k$	(nonlinear constraints)

embotech

SQP SOLVER

- **FORCES**PRO real-time **sequential quadratic programming (SQP) algorithm** iteratively solves **convex quadratic approximations** of the (generally non-convex) problem

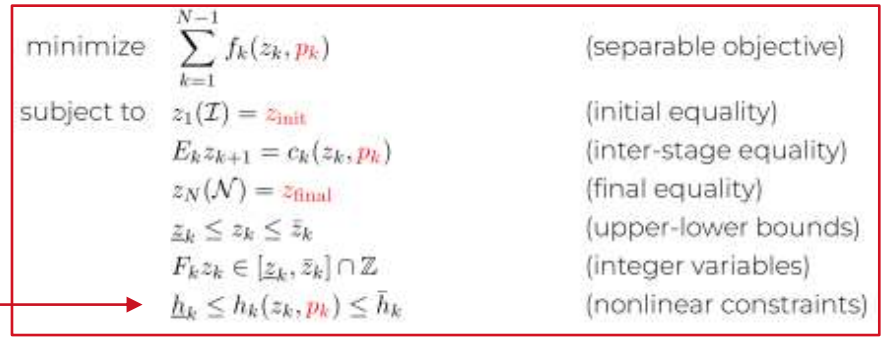
- The **solution is stored internally** in the solver and **used as an initial guess** for the next time the solver is called

- Provides **fast solve times** (compared to the **interior point method**), particularly suitable for MPC applications with
  - **Small sampling time**
  - **Limited computational power** of the hardware

Note: Currently, there are several **modeling requirements** when using the **FORCES**PRO **SQP solver**:

- Algorithm **only supports affine inequalities**

All inequality functions $\boldsymbol{h_k(z_k, p_k)}$must be **affine functions of the variable** $\boldsymbol{z_k}$ (not necessarily of $\boldsymbol{p_k}$)



embotech

SQP SOLVER

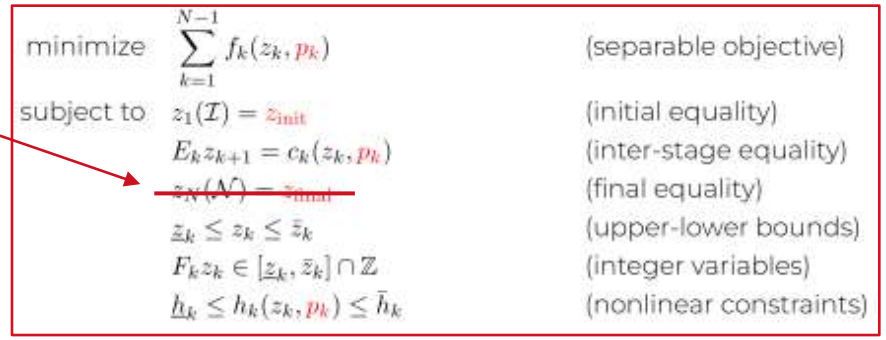
- **FORCES**PRO real-time **sequential quadratic programming (SQP) algorithm** iteratively solves **convex quadratic approximations** of the (generally non-convex) problem

- The **solution is stored internally** in the solver and **used as an initial guess** for the next time the solver is called

- Provides **fast solve times** (compared to the **interior point method**), particularly suitable for MPC applications with
  - **Small sampling time**
  - **Limited computational power** of the hardware

Note: Currently, there are several **modeling requirements** when using the **FORCES**PRO **SQP solver**:

- Algorithm **only supports affine inequalities**
- Algorithm **does not support** problems comprising **final equality constraints** (specified via *model.xfinalidx*)



embotech

SQP SOLVER

- **FORCES**PRO real-time **sequential quadratic programming (SQP) algorithm** iteratively solves **convex quadratic approximations** of the (generally non-convex) problem

- The **solution is stored internally** in the solver and **used as an initial guess** for the next time the solver is called

- Provides **fast solve times** (compared to the **interior point method**), particularly suitable for MPC applications with
  - **Small sampling time**
  - **Limited computational power** of the hardware

Note: Currently, there are several **modeling requirements** when using the **FORCES**PRO **SQP solver**:

- Algorithm **only supports affine inequalities**
- Algorithm **does not support** problems comprising **final equality constraints** (specified via *model.xfinalidx*)
- Algorithm **does not support parallel execution** (setting *codeoptions.parallel* will have no effect)

$$
\begin{aligned}
&\text{minimize} && \sum_{k=1}^{N-1} f_k(z_k, p_k) && \text{(separable objective)}\\
&\text{subject to} && z_1(\mathcal{I}) = z_{\text{init}} && \text{(initial equality)}\\
& && E_k z_{k+1} = c_k(z_k, p_k) && \text{(inter-stage equality)}\\
& && z_N(\mathcal{N}) = z_{\text{final}} && \text{(final equality)}\\
& && \underline{z}_k \le z_k \le \bar{z}_k && \text{(upper-lower bounds)}\\
& && F_k z_k \in [\underline{z}_k, \bar{z}_k] \cap \mathbb{Z} && \text{(integer variables)}\\
& && \underline{h}_k \le h_k(z_k, p_k) \le \bar{h}_k && \text{(nonlinear constraints)}
\end{aligned}
$$

embotech

SQP SOLVER OPTIONS

FORCESPRO **SQP** solves a **single convex quadratic approximation** by default:

- **Fast solve times** compared to a **“full” SQP** solver (solving quadratic NLP approx. until a KKT point is reached)
- **Trade-off** between the **solve time** and control **performance** (user can **allow multiple quadratic approx.**)

```
% Set the number of quadratic approximations to be solved at every call to the solver
codeoptions.sqp_nlp.maxqps = k;
```

embotech

SQP SOLVER OPTIONS

FORCESPRO **SQP** solves a **single convex quadratic approximation** by default:

- **Fast solve times** compared to a **"full" SQP** solver (solving quadratic NLP approx. until a KKT point is reached)
- **Trade-off** between the **solve time** and control **performance** (user can **allow multiple quadratic approx.**)

```
% Set the number of quadratic approximations to be solved at every call to the solver
codeoptions.sqp_nlp.maxqps = k;
```

A good choice of **hessian approximation** can often **improve the number of iterations** required by the SQP solver:

- **Default** option for the SQP solver is the **BFGS** hessian approximation
- **Gauss-Newton** hessian approximation can be **beneficial for least-squares cost** objective functions
- When using Gauss-Newton hessian approximation, one can also **disable the internal line search**

```
% Select the method for approximating the Hessian of the Lagrangian function
codeoptions.nlp.hessian_approximation = 'gauss-newton'; % default setting 'bfgs'

% Disable internal line search (cannot be disabled for the 'bfgs' method)
codeoptions.sqp_nlp.use_line_search = 0; % default setting 1
```

embotech

SQP SOLVER OPTIONS

User can control the **initial SQP solver guess** at run-time

- SQP solver uses the **solution from the previous call** as initial guess in every subsequent solver call (**default**)
- Initial guess **can also be manually set** in subsequent calls to the solver (provided through *problem.x0*)

```
% Manually set the primal initial guess at runtime
problem.reinitialize = 1; % default 0
```

Additional solver options specific to the real-time **FORCES**PRO **SQP** solver:

```
% Set stationarity tolerance required for terminating the algorithm (convergence to a KKT point)
codeoptions.sqp_nlp.TolStat = 1e-6; % default setting

% Set feasibility tolerance required for terminating the algorithm (convergence to a feasible point)
codeoptions.sqp_nlp.TolEq = 1e-6; % default setting

% Set the level of regularization of the hessian approximation
codeoptions.sqp_nlp.reg_hessian = 5e-9; % default setting

% Set the initialization strategy for the internal QP solver
codeoptions.sqp_nlp.qpinit = 0; % 0: cold start (default); 1: centered start
```

embotech

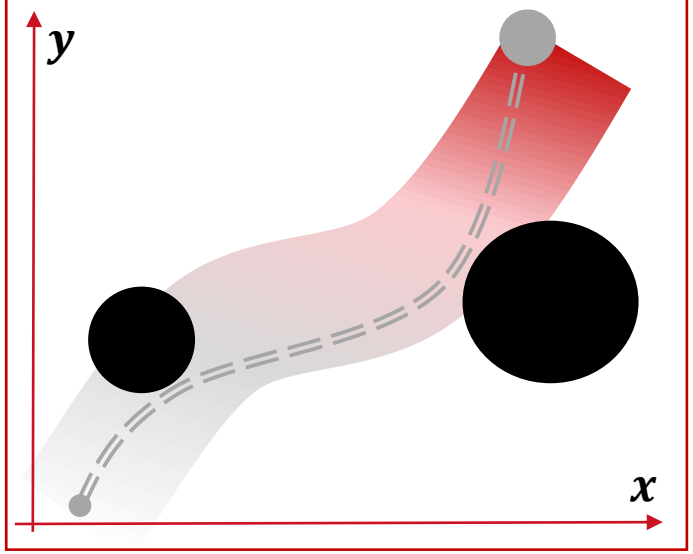
1 PATH PLANNING (OBSTACLE AVOIDANCE)

Simple introductory example to the high-level interface

PROBLEM FORMULATION

AIM

maximize y
subject to car dynamics
actuator limits
bounds on states
avoid ext. obstacles
stay on track



Detailed **example description** and **MATLAB code** are **available for download here**

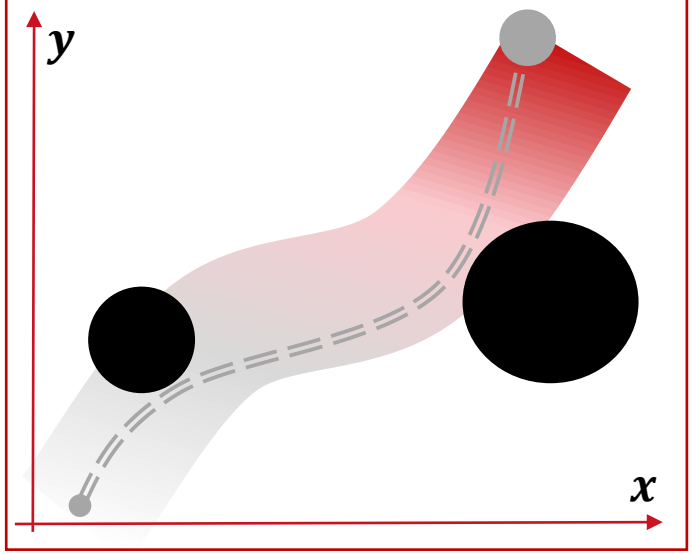
embotech

PROBLEM FORMULATION

AIM

maximize y

subject to
- car dynamics
- actuator limits
- bounds on states
- avoid ext. obstacles
- stay on track



COST FUNCTION

$$f(z) = -100y + 0.1F^2 + 0.01s^2$$

VARIABLES & PARAMETERS

x, y = cartesian coordinates of the car

v = velocity

θ = heading angle

F = acceleration force

s = steering torque

m = mass (1kg)

L = wheel base (1m)

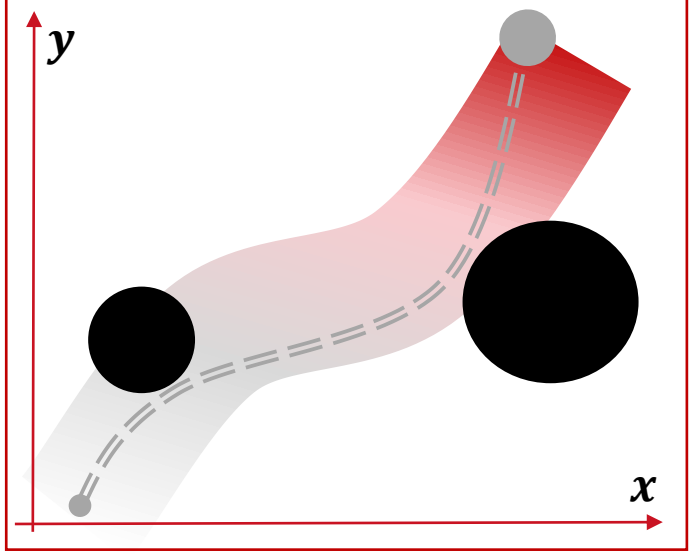
embotech

PROBLEM FORMULATION

AIM

maximize y

subject to
- car dynamics
- actuator limits
- bounds on states
- avoid ext. obstacles
- stay on track



COST FUNCTION

$$f(z) = -100y + 0.1F^2 + 0.01s^2$$

VARIABLES & PARAMETERS

x, y = cartesian coordinates of the car

v = velocity

θ = heading angle

F = acceleration force

s = steering torque

m = mass (1kg)

L = wheel base (1m)

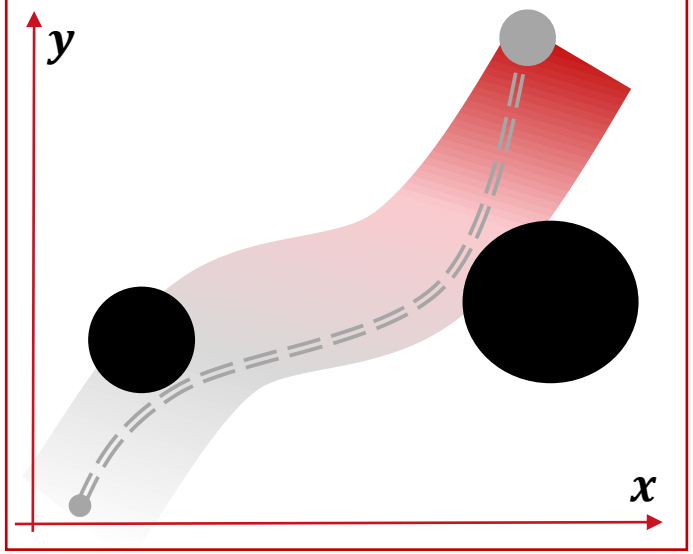
embotech

PROBLEM FORMULATION

AIM

maximize y

subject to **car dynamics**
actuator limits
bounds on states
avoid ext. obstacles
stay on track



COST FUNCTION

$$f(z) = -100y + 0.1F^2 + 0.01s^2$$

VARIABLES & PARAMETERS

states
- x, y = cartesian coordinates of the car
- v = velocity
- θ = heading angle
- F = acceleration force
- s = steering torque
- m = mass (1kg)
- L = wheel base (1m)

CAR DYNAMICS

$$\dot{x} = v\cos(\theta)$$
$$\dot{y} = v\sin(\theta)$$
$$\dot{v} = F/m$$
$$\dot{\theta} = s/L$$

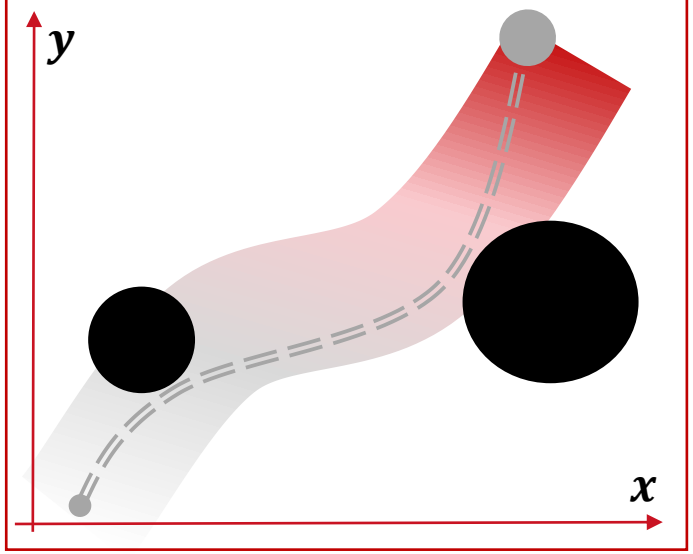
embotech

PROBLEM FORMULATION

AIM

maximize y

subject to **car dynamics**
actuator limits
bounds on states
avoid ext. obstacles
stay on track



COST FUNCTION

$$f(z) = -100y + 0.1F^2 + 0.01s^2$$

VARIABLES & PARAMETERS

x, y = cartesian coordinates of the car

v = velocity

Θ = heading angle

control inputs { F = acceleration force, s = steering torque

m = mass (1kg)

L = wheel base (1m)

CAR DYNAMICS

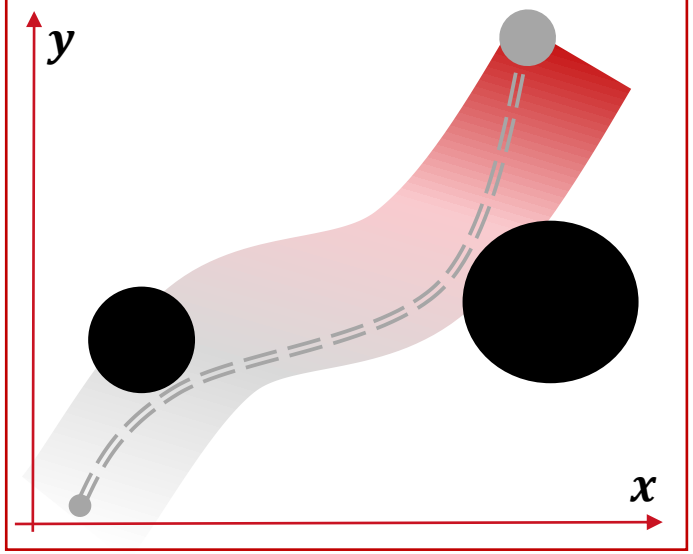
$$\dot{x} = v\cos(\Theta)$$
$$\dot{y} = v\sin(\Theta)$$
$$\dot{v} = F/m$$
$$\dot{\Theta} = s/L$$

PROBLEM FORMULATION

AIM

maximize y

subject to **car dynamics**
actuator limits
bounds on states
avoid ext. obstacles
stay on track



COST FUNCTION

$$f(z) = -100y + 0.1F^2 + 0.01s^2$$

VARIABLES & PARAMETERS

x, y = cartesian coordinates of the car
v = velocity
Θ = heading angle
F = acceleration force
s = steering torque
parameters { m = mass (1kg)
L = wheel base (1m)

CAR DYNAMICS

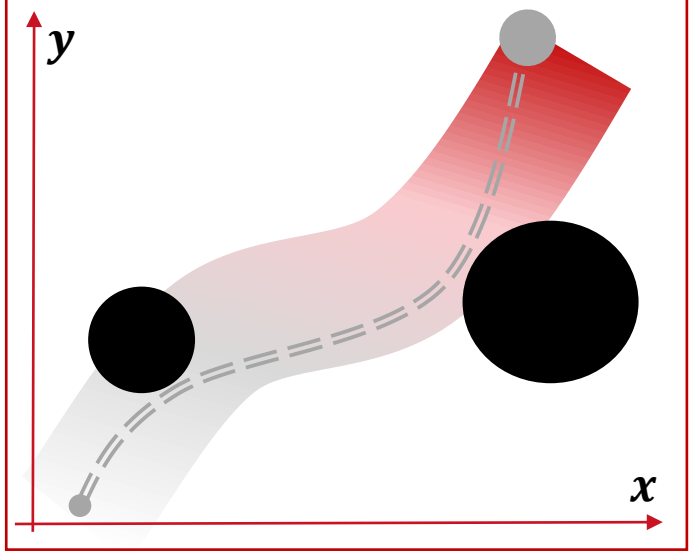
$$\dot{x} = v\cos(\Theta)$$
$$\dot{y} = v\sin(\Theta)$$
$$\dot{v} = F/m$$
$$\dot{\Theta} = s/L$$

embotech

PROBLEM FORMULATION

AIM

maximize y
subject to car dynamics
actuator limits
bounds on states
avoid ext. obstacles
stay on track



COST FUNCTION

$$f(z) = -100y + 0.1F^2 + 0.01s^2$$

CAR DYNAMICS

$$\dot{x} = v\cos(\Theta)$$
$$\dot{y} = v\sin(\Theta)$$
$$\dot{v} = F/m$$
$$\dot{\Theta} = s/L$$

CONSTRAINTS

$$-5\text{N} \leq F \leq 5\text{N}$$
$$-1\text{Nm} \leq s \leq 5\text{Nm}$$

VARIABLES & PARAMETERS

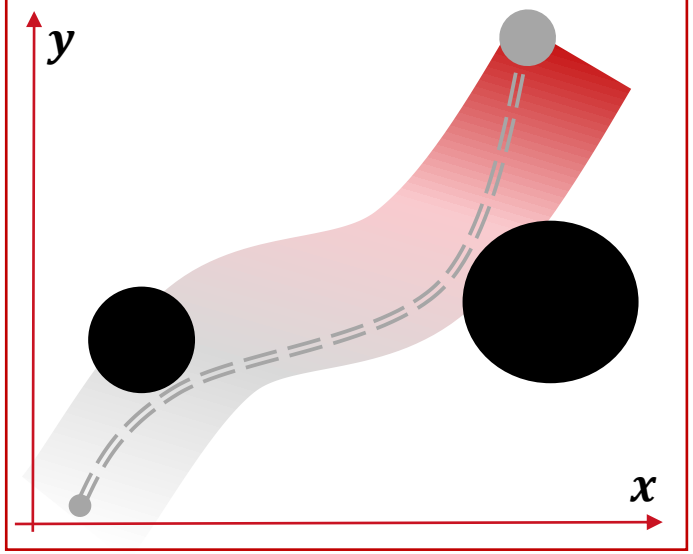
x, y = cartesian coordinates of the car
v = velocity
Θ = heading angle
F = acceleration force
s = steering torque
m = mass (1kg)
L = wheel base (1m)

embotech

PROBLEM FORMULATION

AIM

maximize y
subject to car dynamics
actuator limits
bounds on states
avoid ext. obstacles
stay on track



COST FUNCTION

$$f(z) = -100y + 0.1F^2 + 0.01s^2$$

CAR DYNAMICS

$$\dot{x} = v\cos(\Theta)$$
$$\dot{y} = v\sin(\Theta)$$
$$\dot{v} = F/m$$
$$\dot{\Theta} = s/L$$

CONSTRAINTS

$$-5\text{N} \leq F \leq 5\text{N}$$
$$-1\text{Nm} \leq s \leq 5\text{Nm}$$
$$-3\text{m} \leq x \leq 0\text{m}$$
$$0\text{m} \leq y \leq 3\text{m}$$
$$0\text{m/s} \leq v \leq 2\text{m/s}$$
$$0\text{rad} \leq \Theta \leq \pi\text{rad}$$

VARIABLES & PARAMETERS

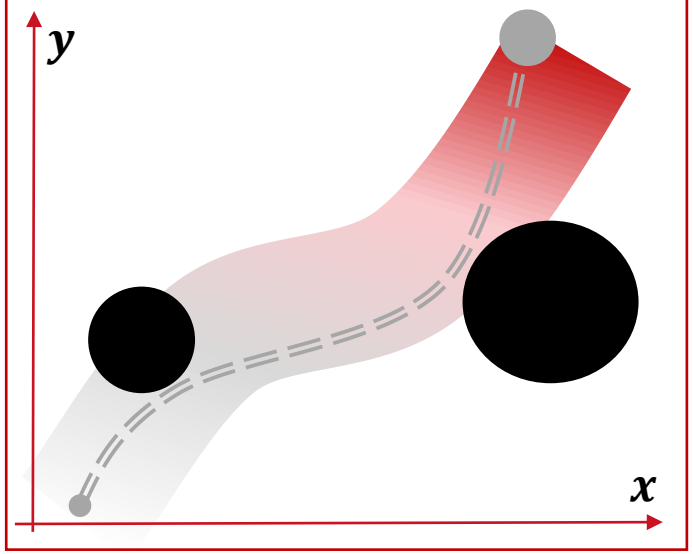
x, y = cartesian coordinates of the car
v = velocity
Θ = heading angle
F = acceleration force
s = steering torque
m = mass (1kg)
L = wheel base (1m)

embotech

PROBLEM FORMULATION

AIM

maximize y
subject to car dynamics
actuator limits
bounds on states
avoid ext. obstacles
stay on track



COST FUNCTION

$$f(z) = -100y + 0.1F^2 + 0.01s^2$$

CAR DYNAMICS

$$\dot{x} = v\cos(\Theta)$$
$$\dot{y} = v\sin(\Theta)$$
$$\dot{v} = F/m$$
$$\dot{\Theta} = s/L$$

CONSTRAINTS

$$-5\text{N} \leq F \leq 5\text{N}$$
$$-1\text{Nm} \leq s \leq 5\text{Nm}$$
$$-3\text{m} \leq x \leq 0\text{m}$$
$$0\text{m} \leq y \leq 3\text{m}$$
$$0\text{m/s} \leq v \leq 2\text{m/s}$$
$$0\text{rad} \leq \Theta \leq \pi\text{rad}$$

VARIABLES & PARAMETERS

x, y = cartesian coordinates of the car
v = velocity
Θ = heading angle
F = acceleration force
s = steering torque
m = mass (1kg)
L = wheel base (1m)

COLLISION CONSTRAINTS

$$1^2 \leq x^2 + y^2 \leq 3^2$$
$$0.95^2 \leq (x+2)^2 + (y-2.5)^2$$

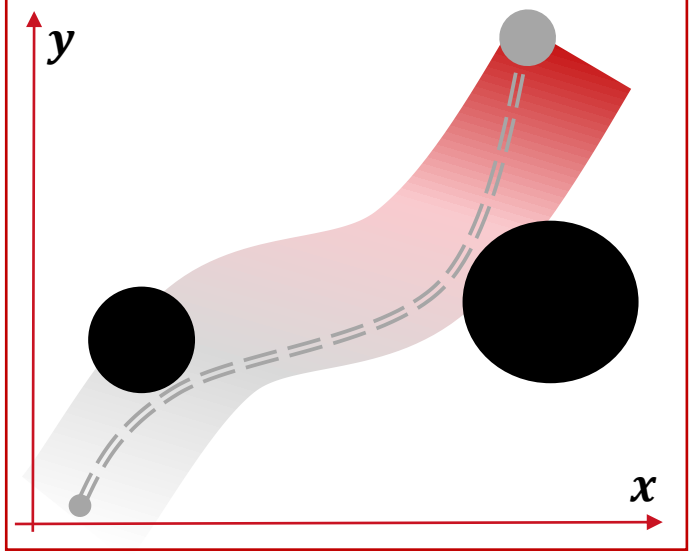
embotech

PROBLEM FORMULATION

AIM

maximize y

subject to
- car dynamics
- actuator limits
- bounds on states
- **avoid ext. obstacles**
- **stay on track**



COST FUNCTION

$$f(z) = -100y + 0.1F^2 + 0.01s^2$$

CAR DYNAMICS

$$\dot{x} = v\cos(\Theta)$$
$$\dot{y} = v\sin(\Theta)$$
$$\dot{v} = F/m$$
$$\dot{\Theta} = s/L$$

CONSTRAINTS

$$-5\text{N} \leq F \leq 5\text{N}$$
$$-1\text{Nm} \leq s \leq 5\text{Nm}$$
$$-3\text{m} \leq x \leq 0\text{m}$$
$$0\text{m} \leq y \leq 3\text{m}$$
$$0\text{m/s} \leq v \leq 2\text{m/s}$$
$$0\text{rad} \leq \Theta \leq \pi\text{rad}$$

VARIABLES & PARAMETERS

x, y = cartesian coordinates of the car

v = velocity

Θ = heading angle

F = acceleration force

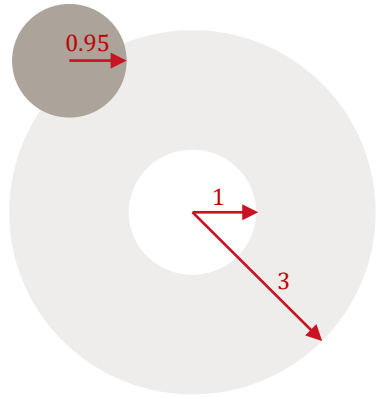
s = steering torque

m = mass (1kg)

L = wheel base (1m)

COLLISION CONSTRAINTS

$$1^2 \leq x^2 + y^2 \leq 3^2$$
$$0.95^2 \leq (x+2)^2 + (y-2.5)^2$$



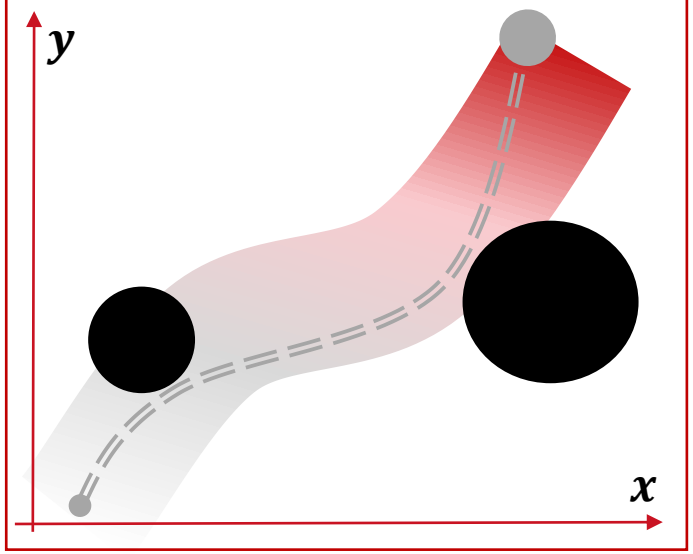
embotech

PROBLEM FORMULATION

AIM

maximize y

subject to
- car dynamics
- actuator limits
- bounds on states
- **avoid ext. obstacles**
- **stay on track**



COST FUNCTION

$$f(z) = -100y + 0.1F^2 + 0.01s^2$$

VARIABLES & PARAMETERS

x, y = cartesian coordinates of the car

v = velocity

Θ = heading angle

F = acceleration force

s = steering torque

m = mass (1kg)

L = wheel base (1m)

CAR DYNAMICS

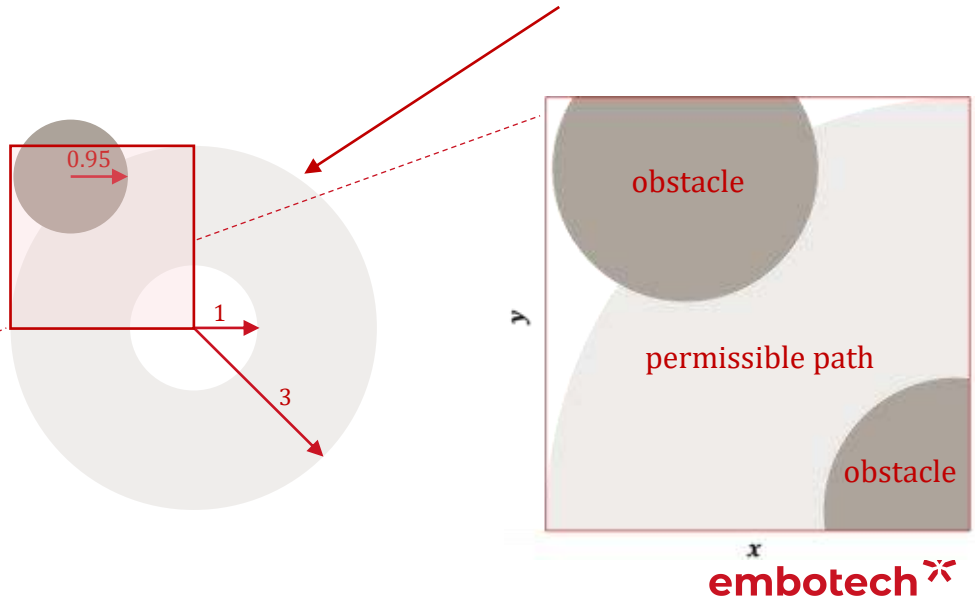
$$\dot{x} = v\cos(\Theta)$$
$$\dot{y} = v\sin(\Theta)$$
$$\dot{v} = F/m$$
$$\dot{\Theta} = s/L$$

CONSTRAINTS

$$-5\text{N} \le F \le 5\text{N}$$
$$-1\text{Nm} \le s \le 5\text{Nm}$$
$$-3\text{m} \le x \le 0\text{m}$$
$$0\text{m} \le y \le 3\text{m}$$
$$0\text{m/s} \le v \le 2\text{m/s}$$
$$0\text{rad} \le \Theta \le \pi\text{rad}$$

COLLISION CONSTRAINTS

$$1^2 \le x^2 + y^2 \le 3^2$$
$$0.95^2 \le (x+2)^2 + (y-2.5)^2$$



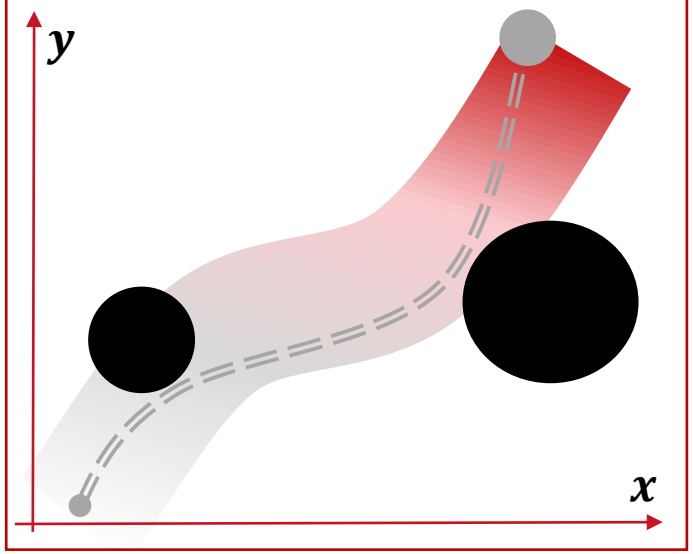
embotech

PROBLEM FORMULATION

AIM

maximize y

subject to
- car dynamics
- actuator limits
- bounds on states
- avoid ext. obstacles
- stay on track



COST FUNCTION

$$f(z) = -100y + 0.1F^2 + 0.01s^2$$

VARIABLES & PARAMETERS

x, y = cartesian coordinates of the car

v = velocity

Θ = heading angle

F = acceleration force

s = steering torque

m = mass (1kg)

L = wheel base (1m)

CAR DYNAMICS

$$\dot{x} = v\cos(\Theta)$$
$$\dot{y} = v\sin(\Theta)$$
$$\dot{v} = F/m$$
$$\dot{\Theta} = s/L$$

COLLISION CONSTRAINTS

$$1^2 \leq x^2 + y^2 \leq 3^2$$
$$0.95^2 \leq (x+2)^2 + (y-2.5)^2$$

CONSTRAINTS

$$-5\text{N} \leq F \leq 5\text{N}$$
$$-1\text{Nm} \leq s \leq 5\text{Nm}$$
$$-3\text{m} \leq x \leq 0\text{m}$$
$$0\text{m} \leq y \leq 3\text{m}$$
$$0\text{m/s} \leq v \leq 2\text{m/s}$$
$$0\text{rad} \leq \Theta \leq \pi\text{rad}$$

INITIAL / FINAL CONDITIONS

$$x_{\text{init}} = -2\text{m}, \quad y_{\text{init}} = 0\text{m},$$
$$v_{\text{init}} = 0\text{m/s}, \quad \Theta_{\text{init}} = 120°$$
$$v_{\text{final}} = 0\text{m/s}, \quad \Theta_{\text{final}} = 0°$$

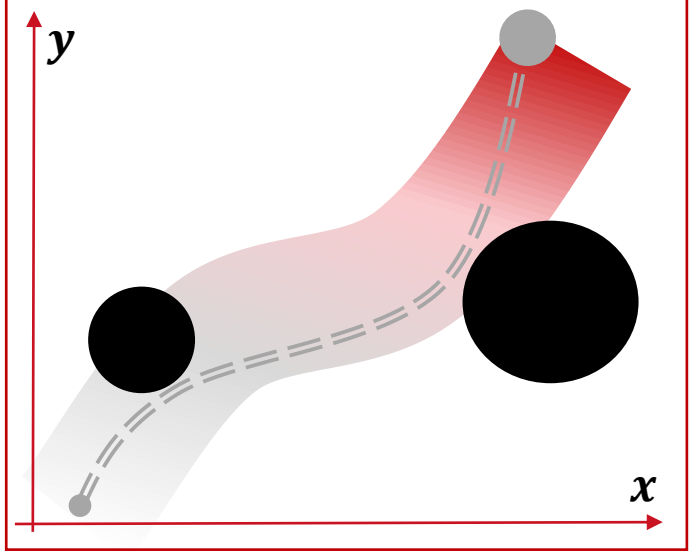
embotech

PROBLEM FORMULATION

AIM

maximize y

subject to
- car dynamics
- actuator limits
- bounds on states
- avoid ext. obstacles
- stay on track



COST FUNCTION

$$f(z) = -100y + 0.1F^2 + 0.01s^2$$

CAR DYNAMICS

$$\dot{x} = v\cos(\Theta)$$
$$\dot{y} = v\sin(\Theta)$$
$$\dot{v} = F/m$$
$$\dot{\Theta} = s/L$$

CONSTRAINTS

$$-5\text{N} \le F \le 5\text{N}$$
$$-1\text{Nm} \le s \le 5\text{Nm}$$
$$-3\text{m} \le x \le 0\text{m}$$
$$0\text{m} \le y \le 3\text{m}$$
$$0\text{m/s} \le v \le 2\text{m/s}$$
$$0\text{rad} \le \Theta \le \pi\text{rad}$$

VARIABLES & PARAMETERS

x, y = cartesian coordinates of the car

v = velocity

Θ = heading angle

F = acceleration force

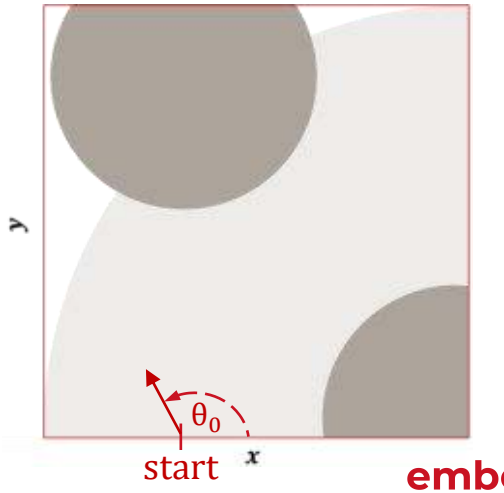
s = steering torque

m = mass (1kg)

L = wheel base (1m)

COLLISION CONSTRAINTS

$$1^2 \le x^2 + y^2 \le 3^2$$
$$0.95^2 \le (x+2)^2 + (y-2.5)^2$$



embotech

SETTING UP THE SOLVER

STEP 1 : PROBLEM DEFINITION

- Problem dimension (horizon length, number of variables/constraints)
- Objective function
- Equality constraints (dynamics)
- Inequality constraints (upper/lower bounds, differentiable nonlinear inequalities)
- Info about initial and final conditions
- Solver options (iteration number, code optimization, progress print, target platform etc.)

STEP 2 : GENERATE THE SOLVER

STEP 3 : CALL THE SOLVER WITH RUNTIME INPUT PARAMETERS

SETTING UP THE SOLVER

STEP 1 : PROBLEM DEFINITION (Variables collected stage-wise into $z = [F, s, x, y, v, \theta]$)

- Problem dimension (horizon length, number of variables/constraints)

```
model.N = 50;              % horizon length
model.nvar = 6;            % number of variables
model.neq = 4;             % number of equality constraints
model.nh = 2;              % number of inequality constraint functions
model.npar = 0;            % number of runtime parameters
```

CAR DYNAMICS

$$\dot{x} = v\cos(\Theta)$$
$$\dot{y} = v\sin(\Theta)$$
$$\dot{v} = F/m$$
$$\dot{\Theta} = s/L$$

SETTING UP THE SOLVER

STEP 1 : PROBLEM DEFINITION (Variables collected stage-wise into $z = [F, s, x, y, v, \theta]$)

- Problem dimension (horizon length, number of variables/constraints)
- Objective function

```
% In this example, we want to maximize position in y direction, with
some penalties on the inputs F and s:
model.objective = @(z) -100*z(4) + 0.1*z(1)^2 + 0.01*z(2)^2;
```

COST FUNCTION

$$f(z) = -100y + 0.1F^2 + 0.01s^2$$

SETTING UP THE SOLVER

STEP 1 : PROBLEM DEFINITION (Variables collected stage-wise into $z = [F, s, x, y, v, \theta]$)

- Problem dimension (horizon length, number of variables/constraints)
- Objective function
- Equality constraints (dynamics)

```
m=1; I=1; % physical constants of the model
integrator_step = 0.1;
cont_dynamics = @(x,u) [x(3)*cos(x(4));  % v*cos(theta)
                        x(3)*sin(x(4));  % v*sin(theta)
                        u(1)/m;          % F/m
                        u(2)/I];         % s/I

% Use an explicit RK4 integrator to discretize continuous dynamics:
model.eq = @(z) RK4( z(3:6), z(1:2), cont_dynamics, integrator_step);

% Indices on LHS of dynamical constraint - efficienct when E = [0 I]:
model.E = [zeros(4,2), eye(4)];
```

CAR DYNAMICS

$$\dot{x} = v\cos(\Theta)$$
$$\dot{y} = v\sin(\Theta)$$
$$\dot{v} = F/m$$
$$\dot{\Theta} = s/L$$

SETTING UP THE SOLVER

STEP 1 : PROBLEM DEFINITION (Variables collected stage-wise into $z = [F, s, x, y, v, \theta]$)

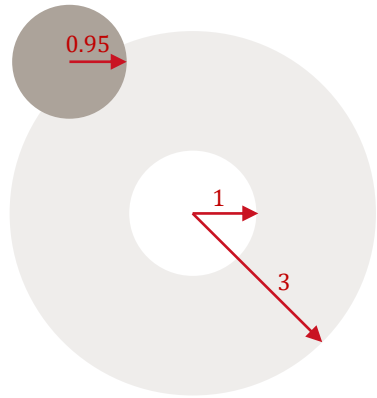
- Problem dimension (horizon length, number of variables/constraints)
- Objective function
- Equality constraints (dynamics)
- Inequality constraints (upper/lower bounds, differentiable nonlinear inequalities)

```
% upper/lower variable bounds lb ≤ x ≤ ub
%             F    s   |    x     y     v    theta
model.lb = [ -5,  -1,      -3,    0     0     0  ];
model.ub = [ +5,  +1,       0,    3     2    +pi ];

% General (differentiable) nonlinear inequalities hl ≤ h(x) ≤ hu
model.ineq = @(z)  [   z(3)^2 + z(4)^2;
                     (z(3)+2)^2 + (z(4)-2.5)^2 ];

% Upper/lower bounds for inequalities
model.hu = [9,  +inf]'; model.hl = [1,  0.95^2]';
```

CONSTRAINTS

$$-5\text{N} \leq F \leq 5\text{N}$$
$$-1\text{Nm} \leq s \leq 5\text{Nm}$$
$$-3\text{m} \leq x \leq 0\text{m}$$
$$0\text{m} \leq y \leq 3\text{m}$$
$$0\text{m/s} \leq v \leq 2\text{m/s}$$
$$0\text{rad} \leq \Theta \leq \pi\text{rad}$$

COLLISION CONSTRAINTS



embotech

SETTING UP THE SOLVER

STEP 1 : PROBLEM DEFINITION (Variables collected stage-wise into $z = [F, s, x, y, v, \theta]$)

- Problem dimension (horizon length, number of variables/constraints)
- Objective function
- Equality constraints (dynamics)
- Inequality constraints (upper/lower bounds, differentiable nonlinear inequalities)
- Info about initial and final conditions

```
% Initial condition on vehicle states
% x=-2, y=0, v=0 (standstill), heading angle=120
model.xinit = [-2, 0, 0, deg2rad(120)]';
model.xinitidx = 3:6; % specify on which variables are imposed

% Final condition on vehicle velocity and heading angle
% v final=0 (standstill), heading angle final=0
model.xfinal = [0, deg2rad(0)]';
model.xfinalidx = 5:6; % specify on which variables are imposed
```

INITIAL / FINAL CONDITIONS

$$x_{\text{init}} = -2\text{m}, \quad y_{\text{init}} = 0\text{m},$$
$$v_{\text{init}} = 0\text{m/s}, \quad \theta_{\text{init}} = 120°$$
$$v_{\text{final}} = 0\text{m/s}, \quad \theta_{\text{final}} = 0°$$

embotech

SETTING UP THE SOLVER

STEP 1 : PROBLEM DEFINITION (Variables collected stage-wise into $z = [F, s, x, y, v, \theta]$)

- Problem dimension (horizon length, number of variables/constraints)
- Objective function
- Equality constraints (dynamics)
- Inequality constraints (upper/lower bounds, differentiable nonlinear inequalities)
- Info about initial and final conditions
- Solver options (iteration number, code optimization, progress print, target platform etc.)

```
codeoptions = getOptions('FORCESNLPsolver');
codeoptions.maxit = 200;            % Maximum number of iterations
codeoptions.optlevel = 2;           % Optimization level
codeoptions.cleanup = false;        % Remove function evaluation code
codeoptions.nlp.compact_code = 0;   % Generate smaller code size
codeoptions.BuildSimulinkBlock = 0; % Don't mex Simulink block
codeoptions.printlevel = 0;         % Print progress
codeoptions.server = 'https://forces.embotech.com';
codeoptions.platform = 'dSPACE-MABXIII'; % target platform
```

embotech

SETTING UP THE SOLVER

STEP 1 : PROBLEM DEFINITION

- Problem dimension (horizon length, number of variables/constraints)
- Objective function
- Equality constraints (dynamics)
- Inequality constraints (upper/lower bounds, differentiable nonlinear inequalities)
- Info about initial and final conditions
- Solver options (iteration number, code optimization, progress print, target platform etc.)

STEP 2 : GENERATE THE SOLVER

```
FORCES_NLP(model, codeoptions);
```

embotech

SETTING UP THE SOLVER

STEP 1 : PROBLEM DEFINITION

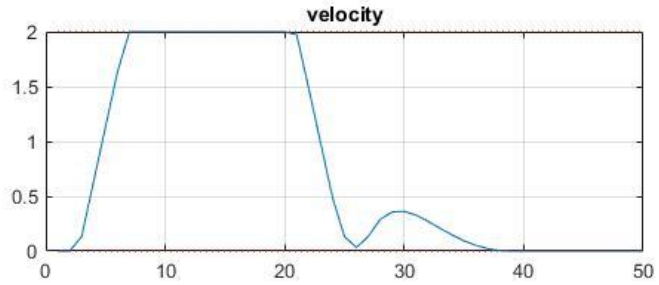
STEP 2 : GENERATE THE SOLVER

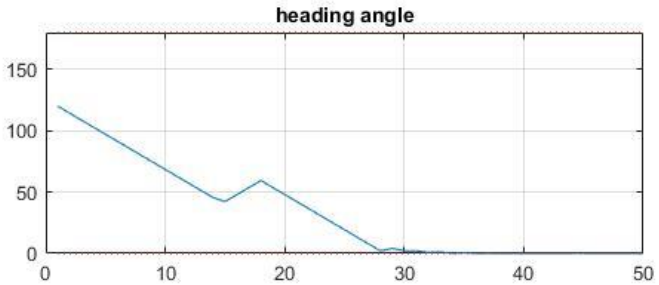
STEP 3 : CALL THE SOLVER WITH RUNTIME INPUT PARAMETERS

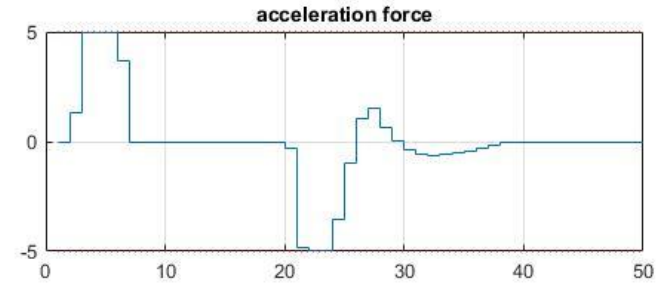
```
% Set initial guess to start solver from:
x0i=model.lb+(model.ub-model.lb)/2;
x0=repmat(x0i',model.N,1);
problem.x0=x0;

% Set initial and final conditions:
problem.xinit = model.xinit;
problem.xfinal = model.xfinal;

% Time to solve the NLP!
[output,exitflag,info] = FORCESNLPsolver(problem);
```

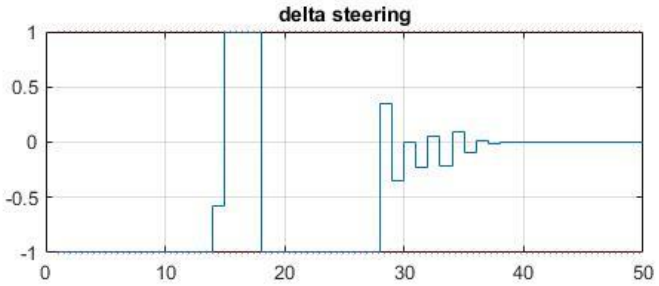
embotech

PLANNED PATH & CONTROL ACTIONS



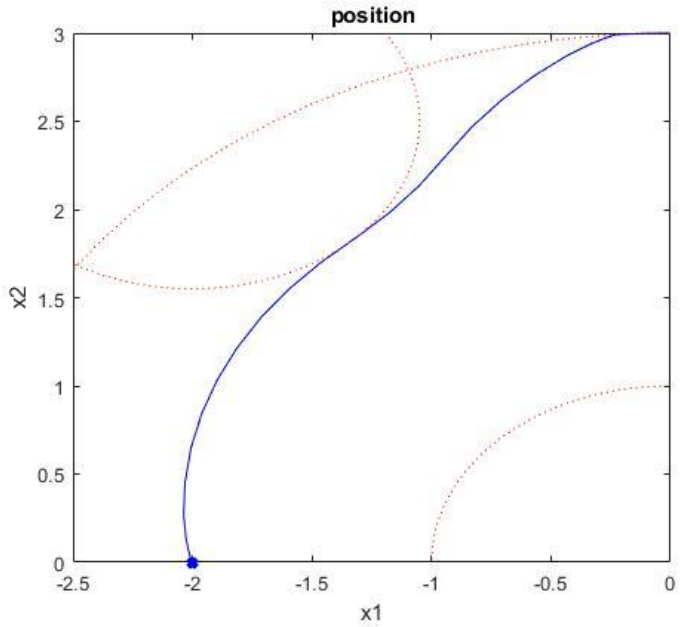






FORCESPRO takes **73 iterations** and **12ms** to solve the problem on average

Simulations performed on x-64-based Intel® Core™ i7-10750H CPU @ 2.60GHz with 16GB RAM



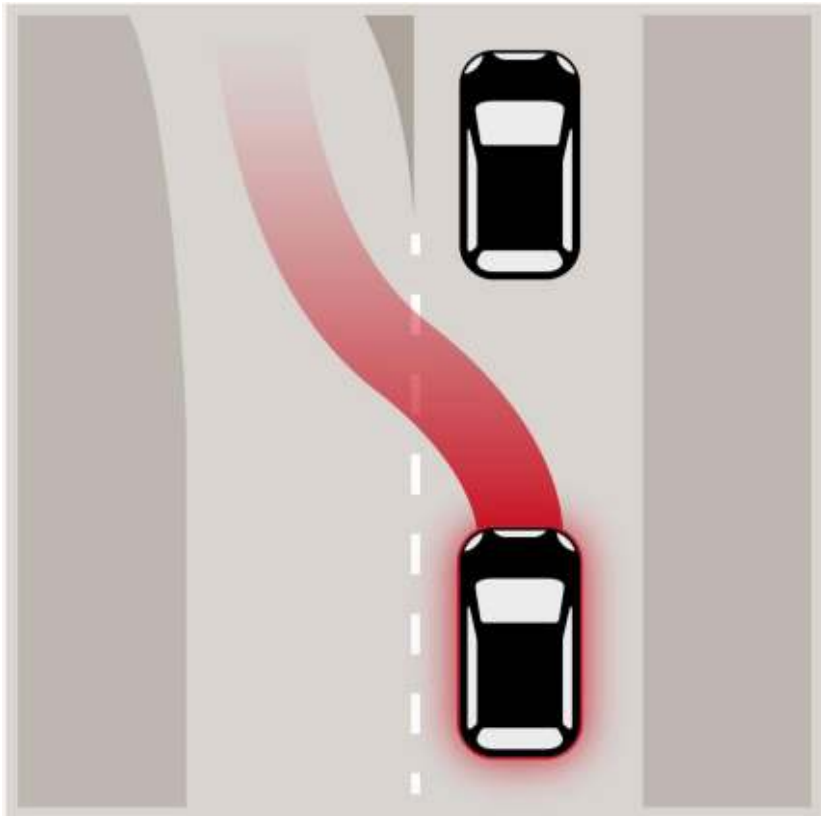
embotech

2 PATH TRACKING

PROBLEM OVERVIEW

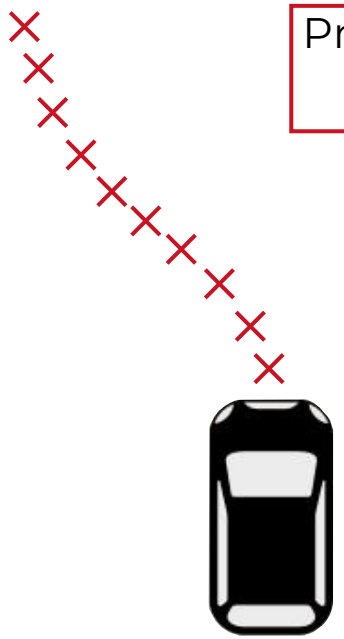
APPLICATION

Lane change (intersection) / passing by another car



MPC PROBLEM

Control the car to track the trajectory in real time



Prediction horizon:
$N = 10$

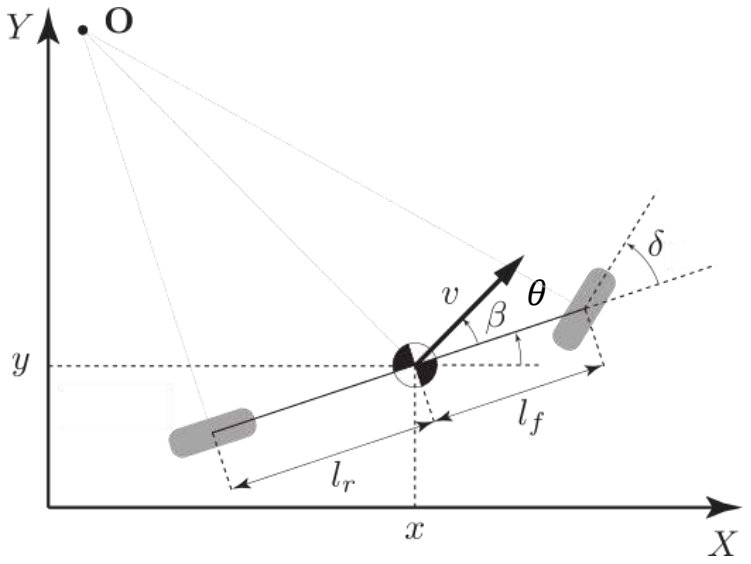
Detailed **example description** and **MATLAB code** are **available for download here**

embotech

PROBLEM FORMULATION: DYNAMIC MODEL

KINEMATIC BICYCLE MODEL

Bicycle model encompassing both **lateral** and **longitudinal** vehicle dynamics



Kong, J., Pfeiffer, M., Schildbach,G., Borelli, F.:
"Kinematic and dynamic models for autonomous driving control design."
2015 IEEE Intelligent Vehices Symposium (IV), pp.1094-1099, 2015.

SYSTEM DYNAMICS

Represented by a **5th-order** vehicle model of the form:

$$\dot{x} = x\cos(\theta + \beta)$$

$$\dot{y} = y\sin(\theta + \beta)$$

$$\dot{v} = \frac{F}{m}$$

$$\dot{\theta} = \frac{v}{l_r}\sin\beta$$

$$\dot{\delta} = \phi$$

with

$$\beta = \tan^{-1}\left(\frac{l_r}{l_r + l_f}\tan\delta\right)$$

VARIABLES & PARAMETERS

STATE VARIABLES

- 2D car position (x, y)
- Car velocity v
- Heading angle θ
- Steering angle δ

CONTROL INPUTS

- Acceleration force F
- Steering rate ϕ

PARAMETERS

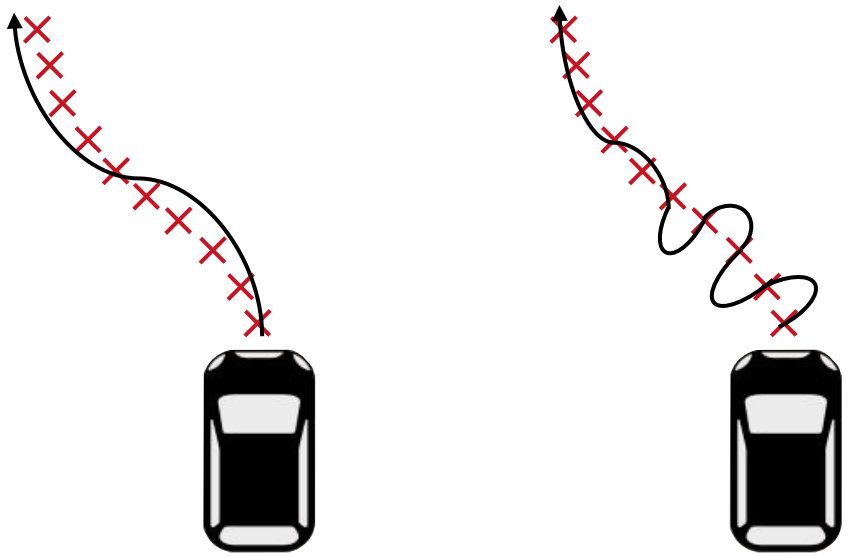
- Car mass m
- Distance l_r rear wheels to COG
- Distance l_f front wheels to COG

embotech

PROBLEM FORMULATION: OBJECTIVE

GOALS

- Drive the car as closely as possible **along the desired trajectory** path points
- **Avoid excessive maneuvers** (ride comfort)



STAGEWISE OBJECTIVE COST FUNCTION

Least square costs on deviating from the path and on the use of control action

$$f_i(z, p_i) = \min_{z_1, z_2} \frac{1}{2} \| r_i(z, p_i) \|_2^2$$

$$r_i(z, p_i) = \begin{bmatrix} \sqrt{200}(z_3 - p_{i,x}) \\ \sqrt{200}(z_4 - p_{i,y}) \\ \sqrt{0.2} z_1 \\ \sqrt{0.02} z_2 \end{bmatrix}$$

- Variables $z = [F, \phi, x, y, v, \theta, \delta]^T$
- Runtime parameters $p_i = [p_{i,x}, p_{i,y}]^T$
- Using **Gauss-Newton hessian approximation** due to a least square objective function

embotech

SOLVER GENERATION

```
function [model, codeoptions, I] = generatePathTrackingSolver(solverDir, timeStep, horizonLength)
    %% Indices - struct I keeps track of variable sorting in z
    % Inputs
    I.FLon = 1; I.steeringRate = 2;
    I.inputs = [I.FLon, I.steeringRate];
    % States
    I.xPos = 3; I.yPos = 4; I.velocity = 5; I.heading = 6; I.steeringAngle = 7;
    I.states = [I.xPos, I.yPos, I.velocity, I.heading, I.steeringAngle];

    %% Problem dimensions
    nInputs = numel(I.inputs);
    nStates = numel(I.states);
    model.N = horizonLength;            % Horizon length
    model.nvar = nInputs+nStates;       % Number of variables
    model.neq  = nStates;               % Number of equality constraints
    model.npar = 2;                     % Number of runtime parameters (waypoint coordinates)

    %% Objective function
    % Definitions of LSobj and LSobjN further below in this file
    model.LSobjective = @(z,p)LSobj(z,p,I);
    model.LSobjectiveN = @(z,p)LSobjN(z,p,I); % Increased costs for the last stage  ...
```

embotech

SOLVER GENERATION

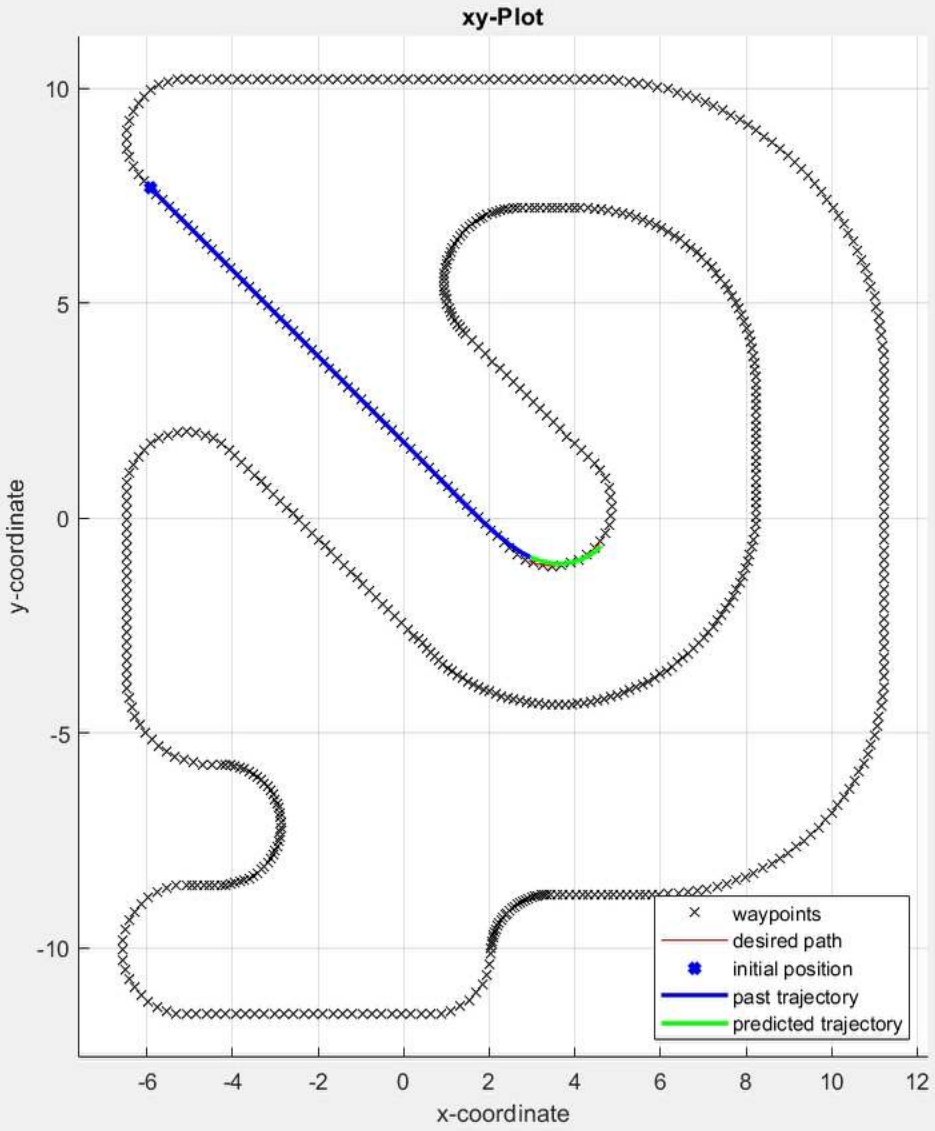
```
...
%% Dynamics, i.e. equality constraints
% We use an explicit RK4 integrator here to discretize continuous dynamics:
model.eq = @(z) RK4( z(I.states), z(I.inputs), @(x,u)continuousDynamics(x,u,I), timeStep);

% Indices on LHS of dynamical constraints
model.E = [zeros(nStates,nInputs), eye(nStates)];

%% Inequality constraints
% Upper/lower variable bounds lb ≤ z ≤ ub
%                 inputs         |                       states
%           FLon   steeringRate  |  xPos    yPos  velocity  heading   steeringAngle
model.lb = [ -5.,  deg2rad(-90),   -100.,  -100.,   0.,     -inf,     deg2rad(-50)];
model.ub = [ +5.,  deg2rad(90),     100.,   100.,   5.,     +inf,     deg2rad(50)];

% Initial conditions on vehicle states
model.xinitidx = I.states; % Use this to specify on which variables initial conditions are imposed ...
```

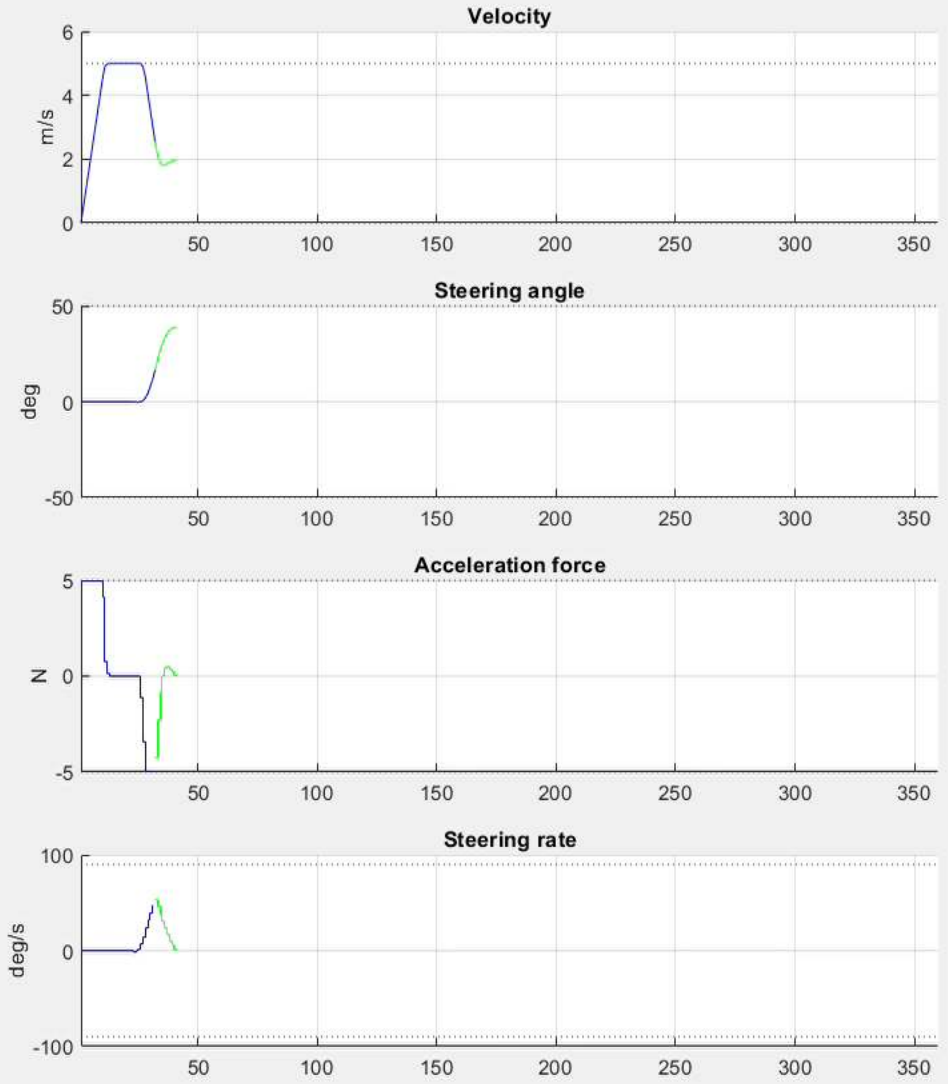
SOLVER GENERATION

```
...
    %% Define solver options
    codeoptions = getOptions('PathTrackingSolver');
    codeoptions.maxit = 200;        % Maximum number of iterations
    codeoptions.optlevel = 3;       % 0: No optimization, good for prototyping
    codeoptions.timing = 1;
    codeoptions.printlevel = 0;
    codeoptions.nohash = 1;         % Enforce solver regeneration
    codeoptions.overwrite = 1;      % Overwrite existing solver
    codeoptions.BuildSimulinkBlock = 0;

    % SQP options
    codeoptions.solvemethod = 'SQP_NLP';
    codeoptions.nlp.hessian_approximation = 'gauss-newton';
    codeoptions.sqp_nlp.maxqps = 1; % Max number of quadratic problems to be solved in one solver call
    codeoptions.sqp_nlp.use_line_search = 0;

    % PDIP options (alternatively)
    codeoptions.solvemethod = 'PDIP_NLP';
    codeoptions.nlp.hessian_approximation = 'bfgs';

    %% Generate FORCESPRO solver
    FORCES_NLP(model, codeoptions);
end
```

embotech

CONTROL PERFORMANCE



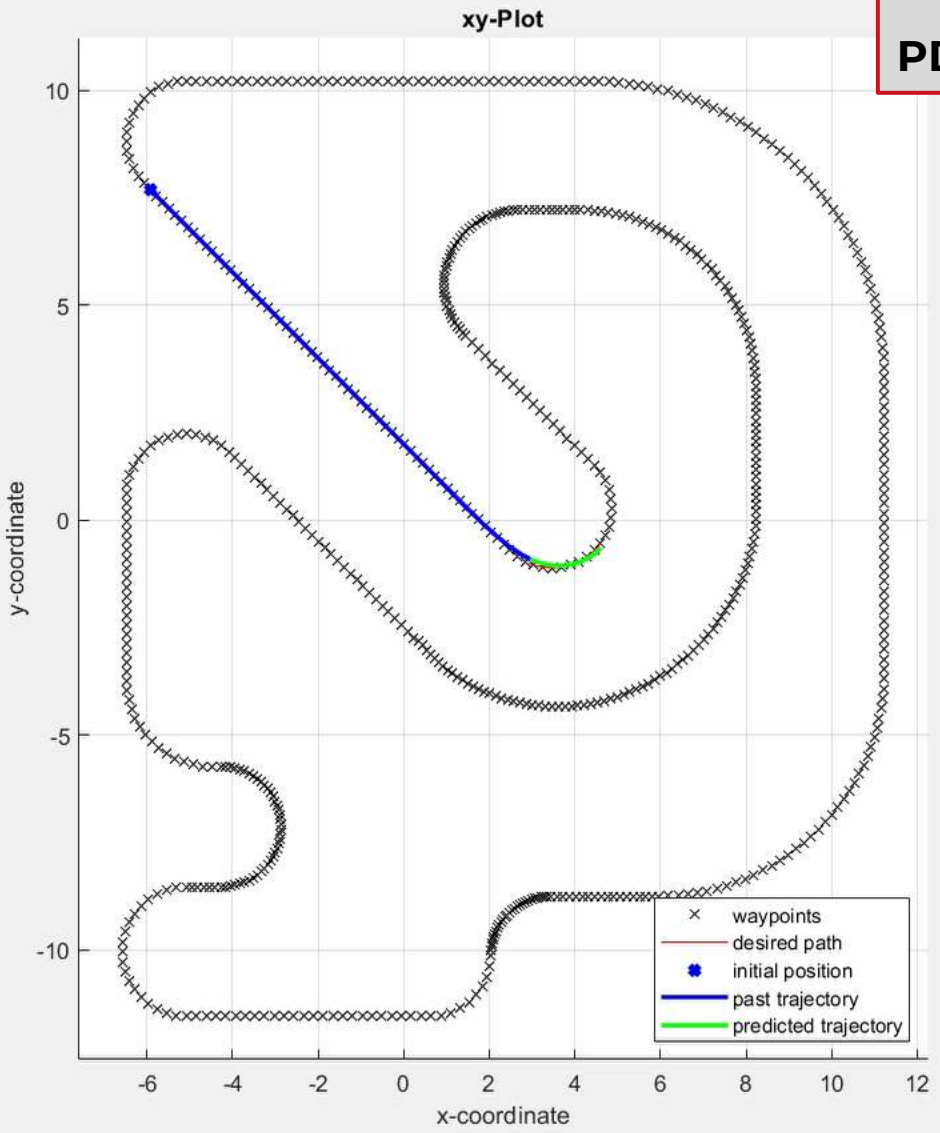


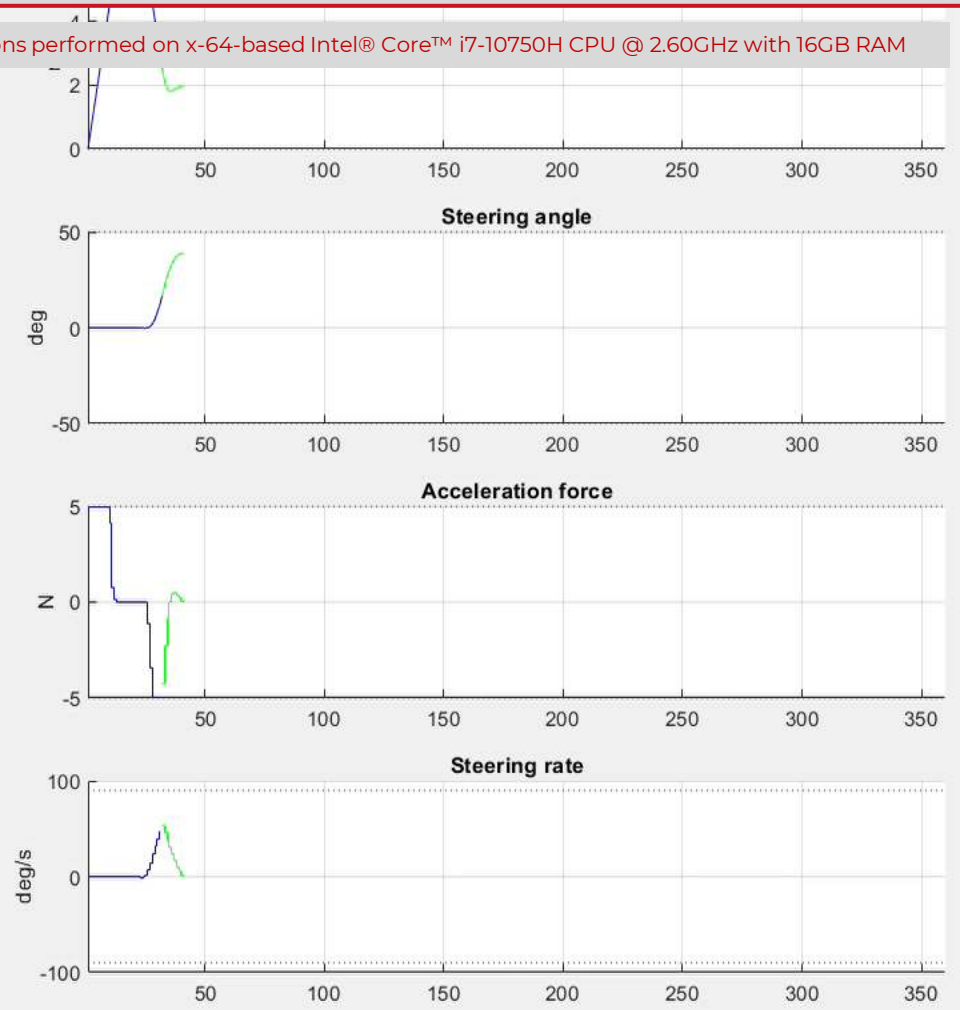
CONTROL PERFORMANCE

1 SQP iteration takes approximately **0.2ms** to solve

PDIP takes **17 iterations** and **0.48ms** to solve on average

Simulations performed on x-64-based Intel® Core™ i7-10750H CPU @ 2.60GHz with 16GB RAM





3 ENERGY-EFFICIENT SHORTEST EV TRIP

PROBLEM OVERVIEW

Goal: Determine EV's **optimal speed** and **charging profiles** in order to minimize **total travel time**, while

- **Reducing** the overall **energy consumption**
- **Selecting** the most suitable **charging locations**
- Considering various **vehicle**, **road**, and **environmental aspects**

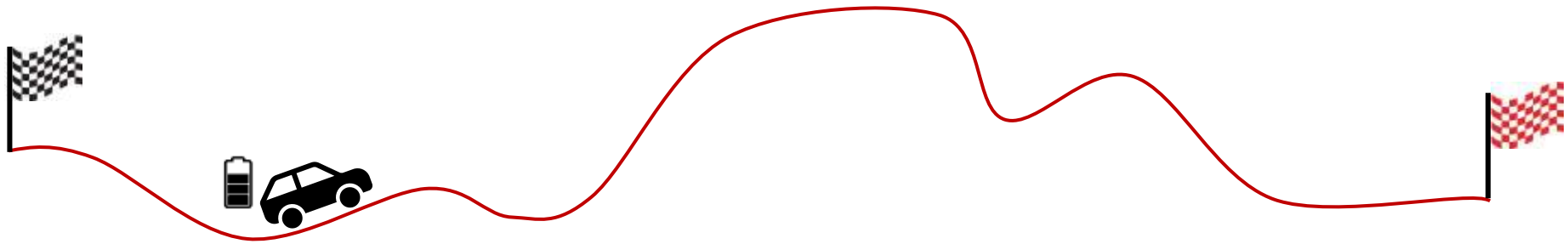
embotech

PROBLEM OVERVIEW

Goal: Determine EV's **optimal speed** and **charging profiles** in order to minimize **total travel time**, while

- **Reducing** the overall **energy consumption**
- **Selecting** the most suitable **charging locations**
- Considering various **vehicle**, **road**, and **environmental aspects**

Electric vehicle traveling between two locations **(predefined route)**



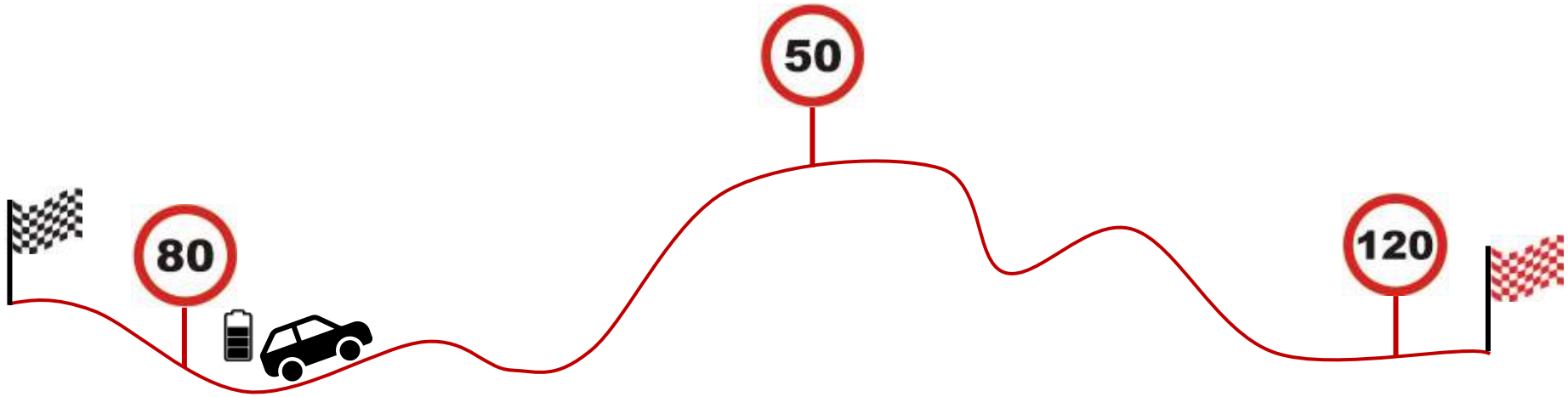
embotech

PROBLEM OVERVIEW

Goal: Determine EV's **optimal speed** and **charging profiles** in order to minimize **total travel time**, while

- **Reducing** the overall **energy consumption**
- **Selecting** the most suitable **charging locations**
- Considering various **vehicle**, **road**, and **environmental aspects**

Route characteristics:
- Speed limits



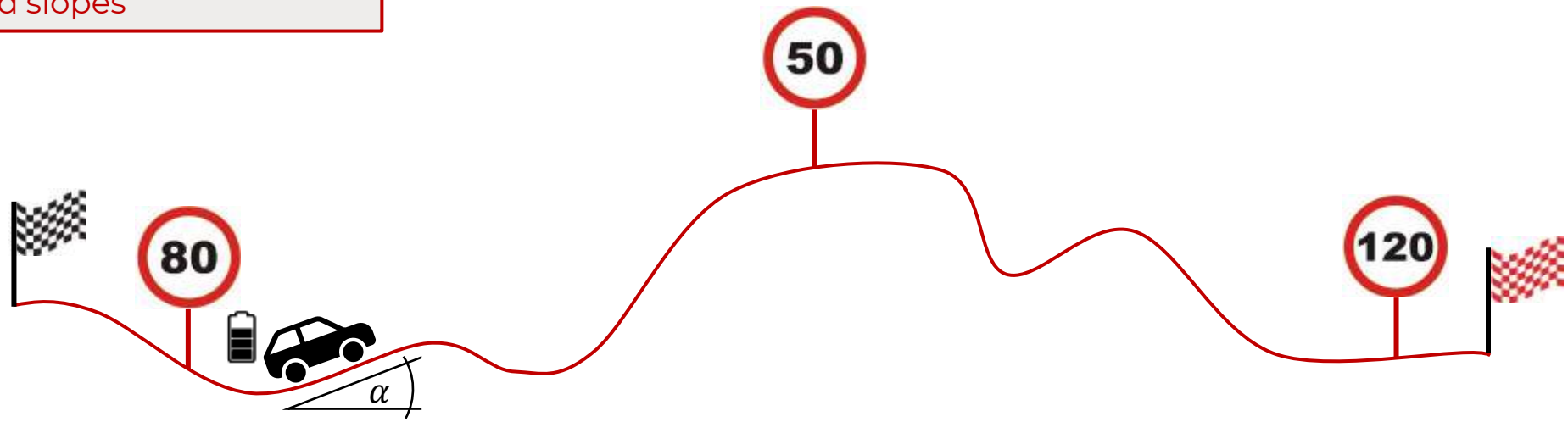
embotech

PROBLEM OVERVIEW

Goal: Determine EV's **optimal speed** and **charging profiles** in order to minimize **total travel time**, while

- **Reducing** the overall **energy consumption**
- **Selecting** the most suitable **charging locations**
- Considering various **vehicle**, **road**, and **environmental aspects**

Route characteristics:
- Speed limits
- Road slopes



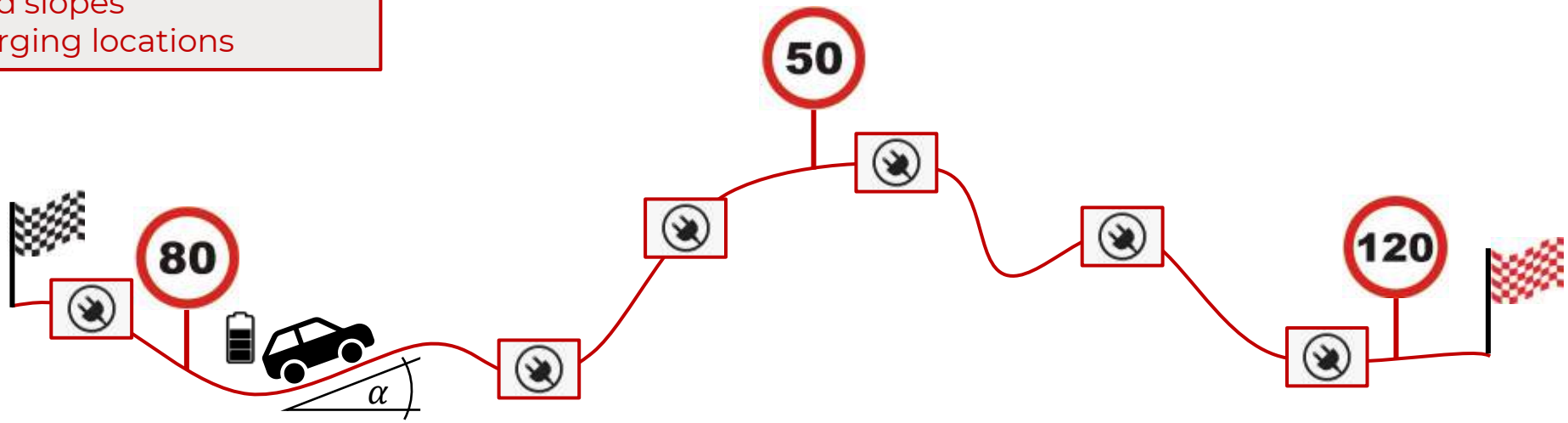
embotech

PROBLEM OVERVIEW

Goal: Determine EV's **optimal speed** and **charging profiles** in order to minimize **total travel time**, while

- **Reducing** the overall **energy consumption**
- **Selecting** the most suitable **charging locations**
- Considering various **vehicle**, **road**, and **environmental aspects**

Route characteristics:
- Speed limits
- Road slopes
- Charging locations



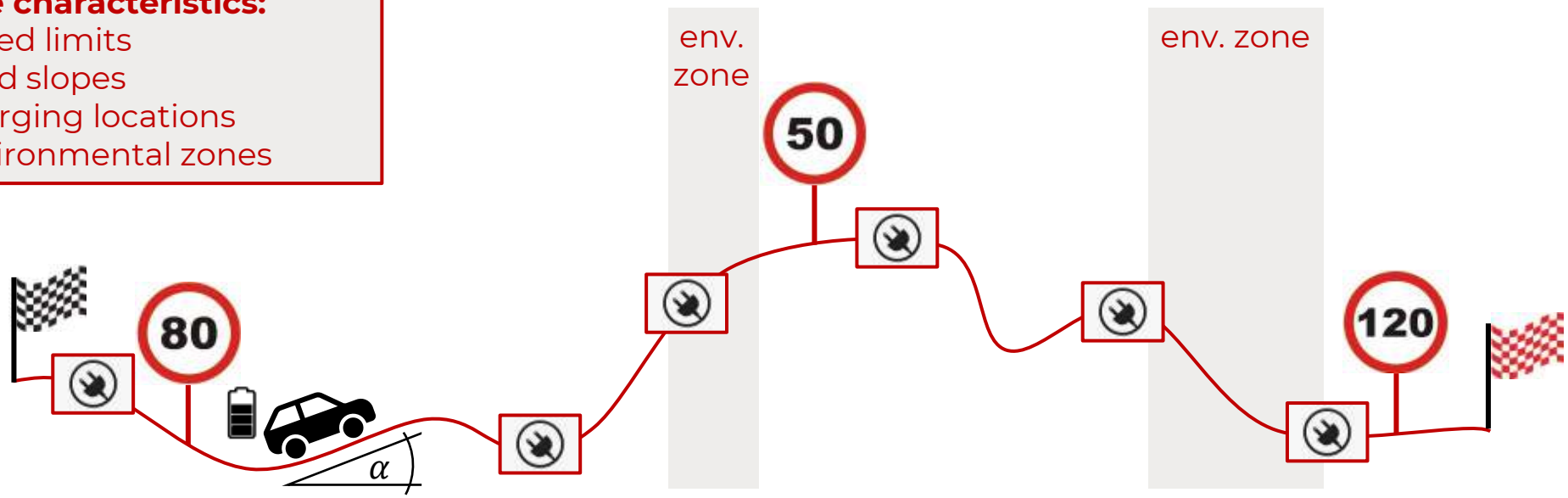
embotech

PROBLEM OVERVIEW

Goal: Determine EV's **optimal speed** and **charging profiles** in order to minimize **total travel time**, while

- **Reducing** the overall **energy consumption**
- **Selecting** the most suitable **charging locations**
- Considering various **vehicle**, **road**, and **environmental aspects**

Route characteristics:
- Speed limits
- Road slopes
- Charging locations
- Environmental zones



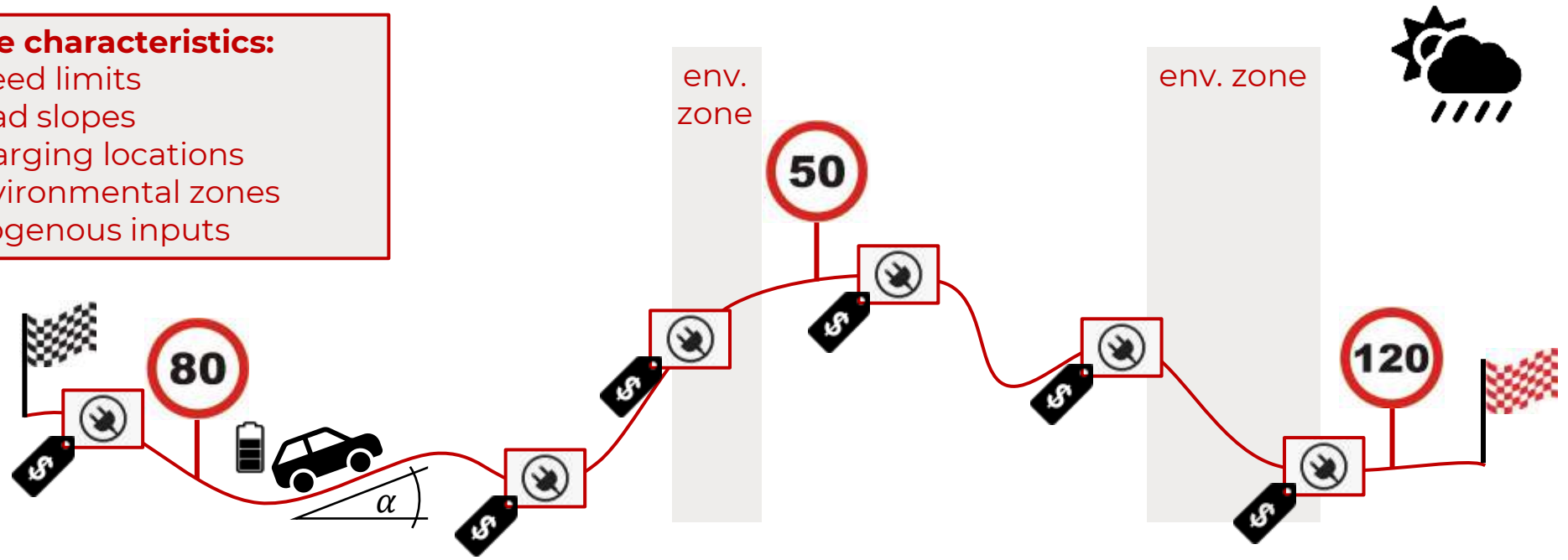
embotech

PROBLEM OVERVIEW

Goal: Determine EV's **optimal speed** and **charging profiles** in order to minimize **total travel time**, while

- **Reducing** the overall **energy consumption**
- **Selecting** the most suitable **charging locations**
- Considering various **vehicle**, **road**, and **environmental aspects**

Route characteristics:
- Speed limits
- Road slopes
- Charging locations
- Environmental zones
- Exogenous inputs

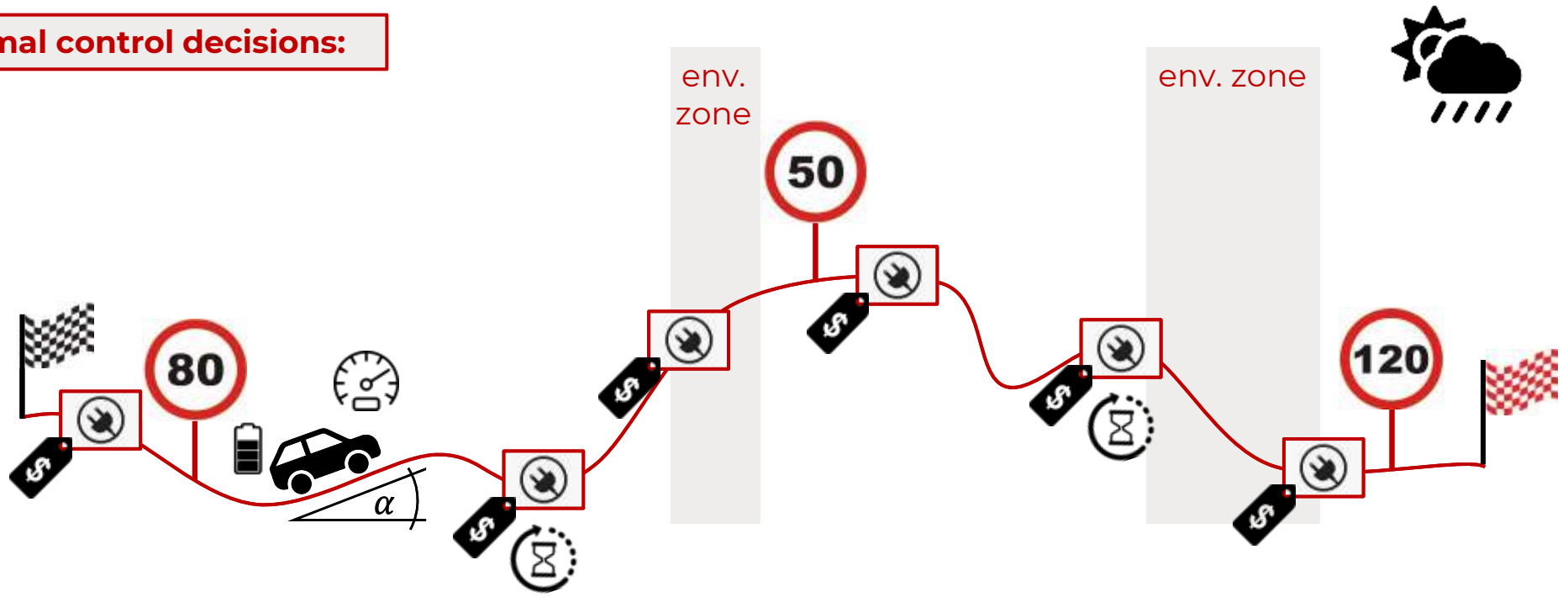


PROBLEM OVERVIEW

Goal: Determine EV's **optimal speed** and **charging profiles** in order to minimize **total travel time**, while

- **Reducing** the overall **energy consumption**
- **Selecting** the most suitable **charging locations**
- Considering various **vehicle**, **road**, and **environmental aspects**

Optimal control decisions:



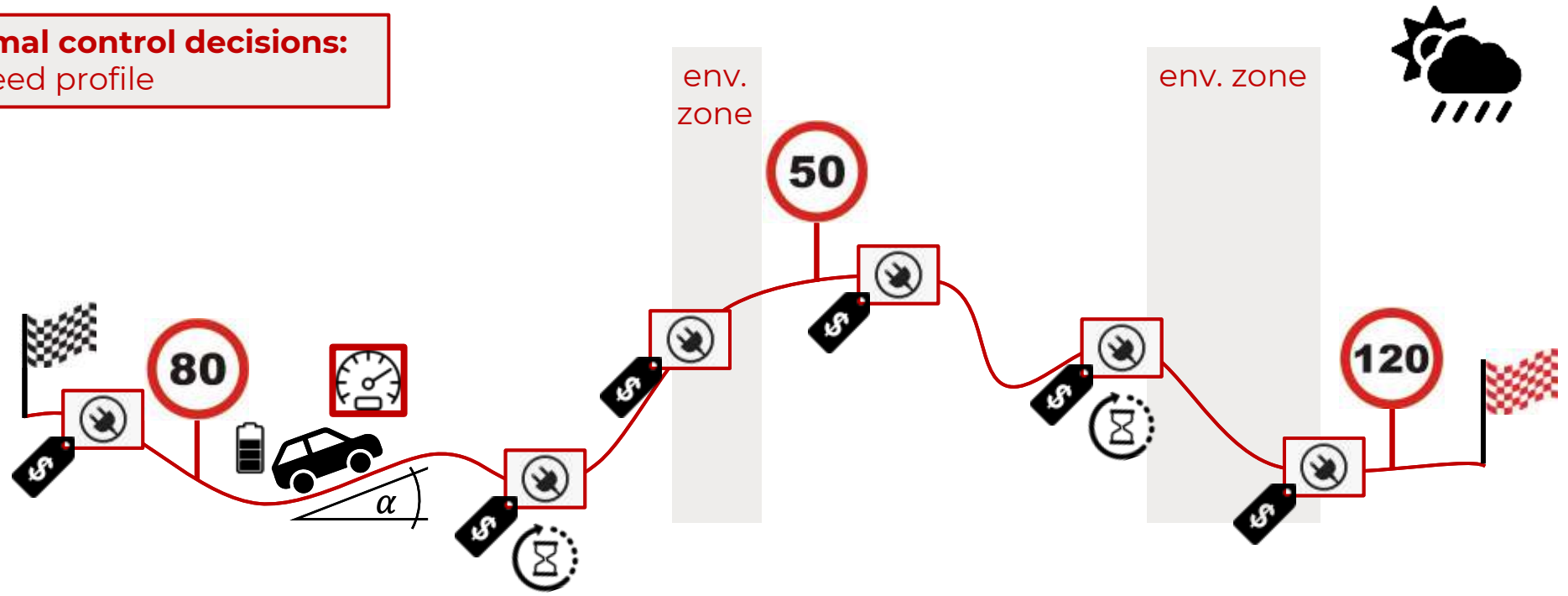
embotech

PROBLEM OVERVIEW

Goal: Determine EV's **optimal speed** and **charging profiles** in order to minimize **total travel time**, while

- **Reducing** the overall **energy consumption**
- **Selecting** the most suitable **charging locations**
- Considering various **vehicle**, **road**, and **environmental aspects**

Optimal control decisions:
- Speed profile



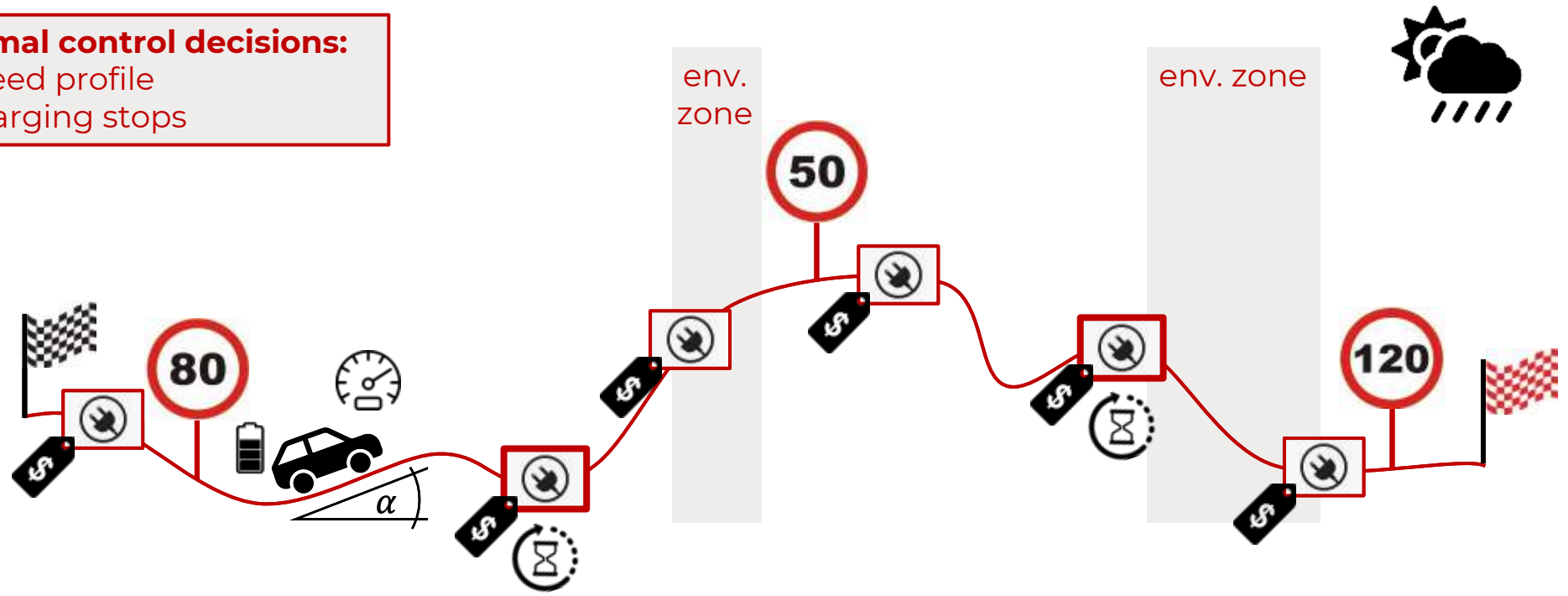
embotech

PROBLEM OVERVIEW

Goal: Determine EV's **optimal speed** and **charging profiles** in order to minimize **total travel time**, while

- **Reducing** the overall **energy consumption**
- **Selecting** the most suitable **charging locations**
- Considering various **vehicle**, **road**, and **environmental aspects**

Optimal control decisions:
- Speed profile
- Charging stops

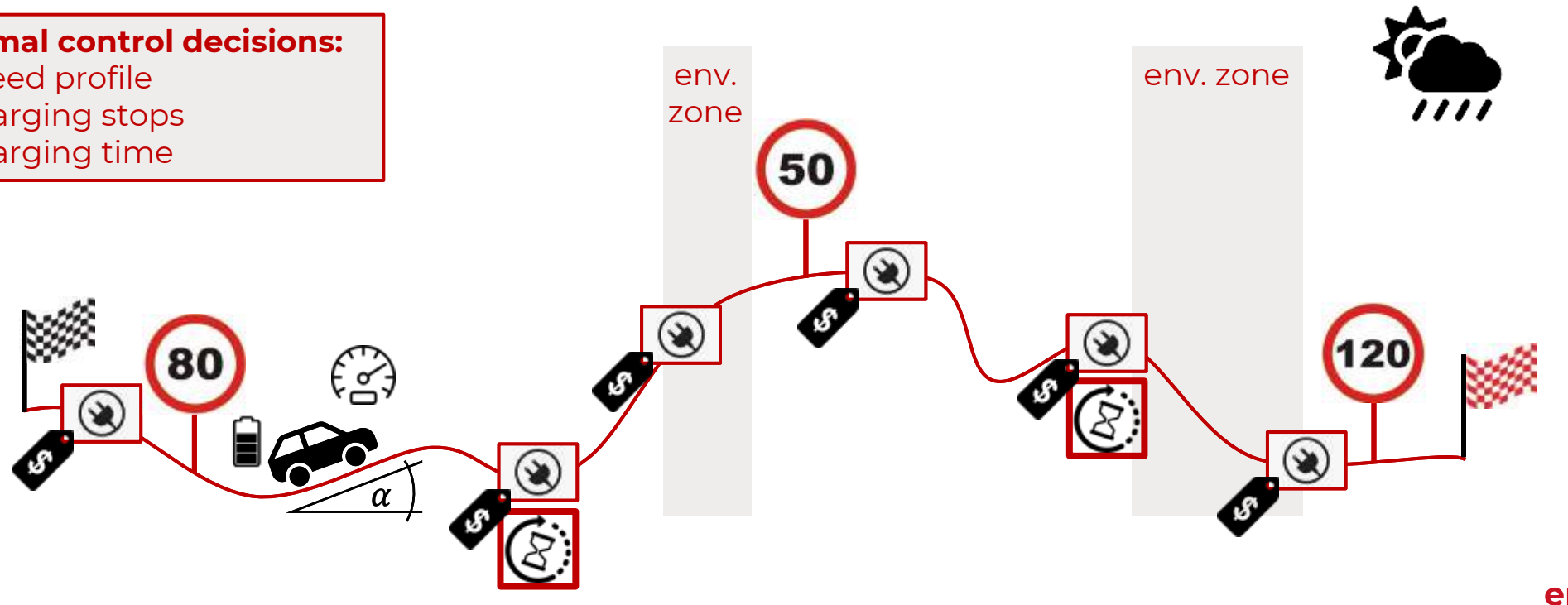


PROBLEM OVERVIEW

Goal: Determine EV's **optimal speed** and **charging profiles** in order to minimize **total travel time**, while

- **Reducing** the overall **energy consumption**
- **Selecting** the most suitable **charging locations**
- Considering various **vehicle**, **road**, and **environmental aspects**

Optimal control decisions:
- Speed profile
- Charging stops
- Charging time



embotech

PROBLEM OVERVIEW

Goal: Determine EV's **optimal speed** and **charging profiles** in order to minimize **total travel time**, while

- **Reducing** the overall **energy consumption**
- **Selecting** the most suitable **charging locations**
- Considering various **vehicle**, **road**, and **environmental aspects**

Input: Route and charging station information, EV specifications, energy prices, weather forecast etc.

embotech

PROBLEM OVERVIEW

Goal: Determine EV's **optimal speed** and **charging profiles** in order to minimize **total travel time**, while

- **Reducing** the overall **energy consumption**
- **Selecting** the most suitable **charging locations**
- Considering various **vehicle**, **road**, and **environmental aspects**

Input: Route and charging station information, EV specifications, ~~energy prices, weather forecast etc.~~

embotech

PROBLEM OVERVIEW

Goal: Determine EV's **optimal speed** and **charging profiles** in order to minimize **total travel time**, while

- **Reducing** the overall **energy consumption**
- **Selecting** the most suitable **charging locations**
- Considering various **vehicle**, **road**, and **environmental aspects**

Input: Route and charging station information, EV specifications, ~~energy prices, weather forecast etc.~~

Method: FORCESPRO NLP solver providing **real-time speed** and **charging profiles**

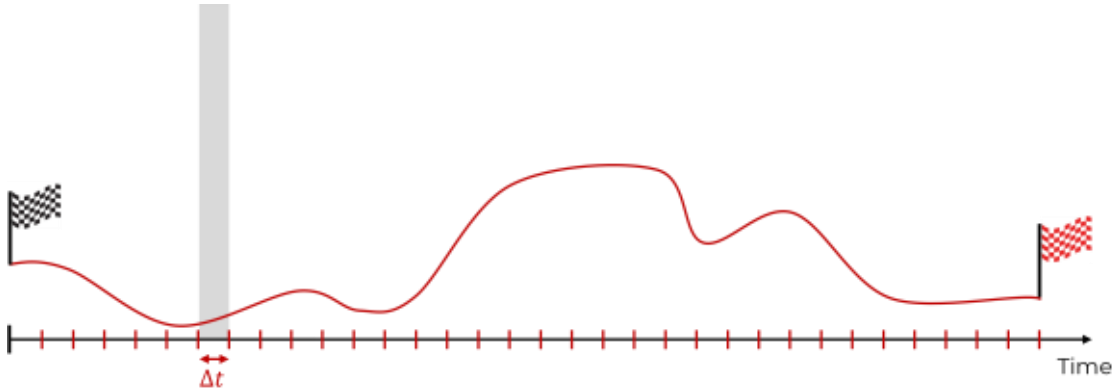
- Implemets a **shrinking horizon MPC** encompassing the whole route
- Discretization in **spatial domain** using kinetic energy laws
- **Highly efficient online computation** (500-stage simulations solved in less than 100ms)
- MPC parameters could be stage dependent, allowing for exogenous (stochastic) inputs

embotech

TIME VS SPACE DOMAIN

TIME DOMAIN

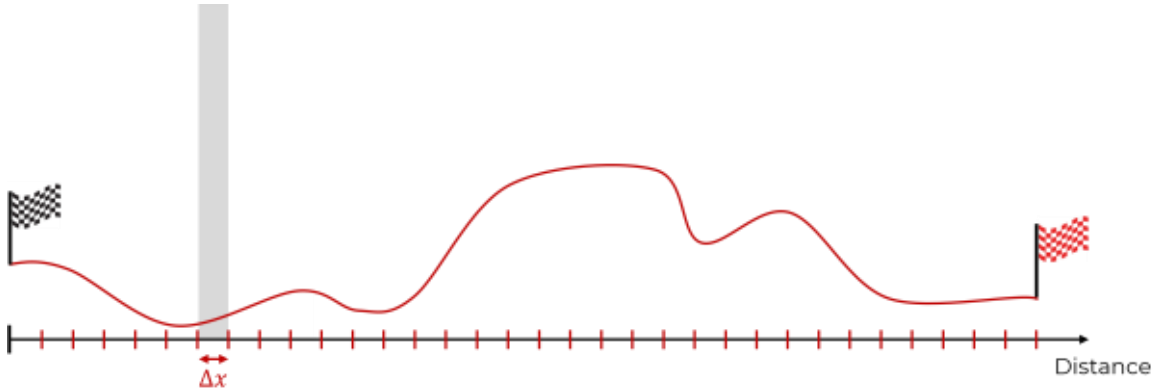
+ **Simple** vehicle dynamics, easily obtainable QP
+ No numerical obstacles for speeds close to zero
- **Inaccurate approximation** of control actions and vehicle energy dynamics



SPACE DOMAIN

+ **More accurate** system representation
+ Allows for **road position-specific parameters**
+ Practical for time-optimal problems
- Introduces **nonlinearity** to vehicle dynamics

(not an issue for **FORCES**PRO NLP solver)

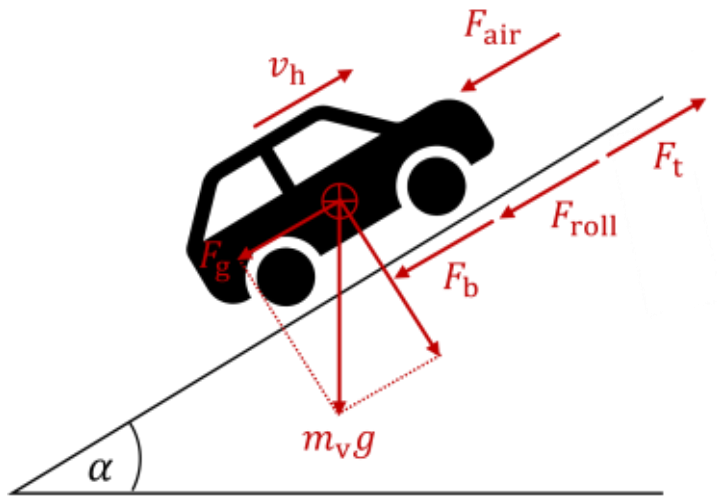


embotech

DYNAMIC VEHICLE MODEL

MECHANICAL MODEL

- **Longitudinal** vehicle dynamics based on **traction**, **rolling resistance**, **gravitational** and **air drag force**
- Velocity determined via **kinetic energy and motion laws** (highly **nonlinear**)
- Use of **nonlinear efficiency maps** for **charging power** and vehicle's **powertrain efficiency**
- **Regenerative breaking** capabilities



$m_v(1+e_I)$ — control decisions

$$\frac{1}{2} m_{eq} \left(v_{i+1}^2 - v_i^2\right) = L_s \left(F_{t,i} - F_{b,i} - F_{r,i}\right)$$

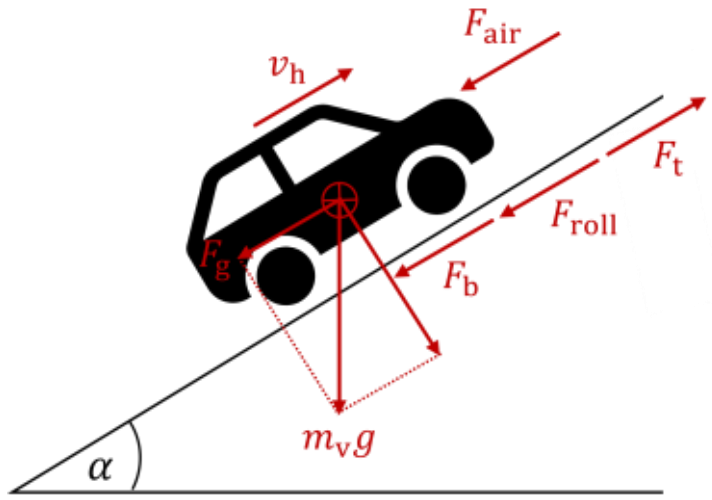
$$F_{r,i} = \underbrace{c_r m_v g \cos(\alpha_i)}_{\text{rolling resistance}} + \underbrace{m_v g \sin(\alpha_i)}_{\text{gravitational force}} + \underbrace{\frac{1}{2} c_a A_f \rho_a v_i^2}_{\text{air drag force}}$$

embotech

DYNAMIC VEHICLE MODEL

MECHANICAL MODEL

- **Longitudinal** vehicle dynamics based on **traction**, **rolling resistance**, **gravitational** and **air drag force**
- Velocity determined via **kinetic energy and motion laws** (highly **nonlinear**)
- Use of **nonlinear efficiency maps** for **charging power** and vehicle's **powertrain efficiency**
- **Regenerative breaking** capabilities



VEHICLE ENERGETICS

- Charging stops can be **optional** or **mandatory**
- Charging speed determined by vehicle's SoC
- **Charging decisions** defined as **charging time** associated with **each discrete spatial step**

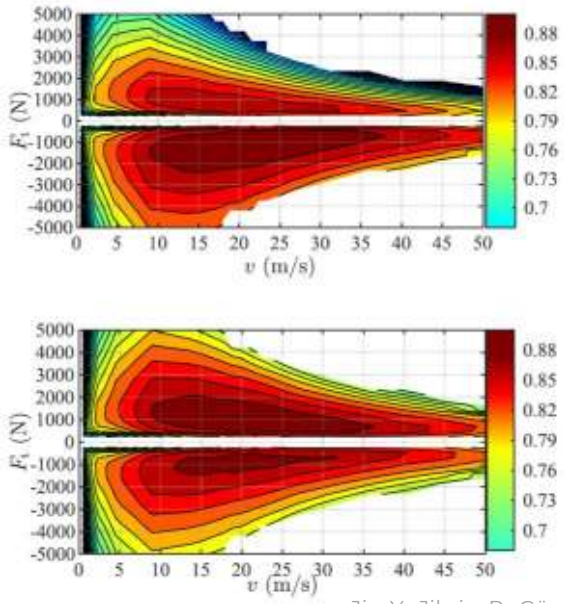
$$v_{i+1} = \sqrt{\frac{2\left(F_{\mathrm{t},i} - F_{\mathrm{b},i} - F_{\mathrm{r},i}\right)L_{\mathrm{s}}}{m_{\mathrm{eq}}} + v_i^2}$$

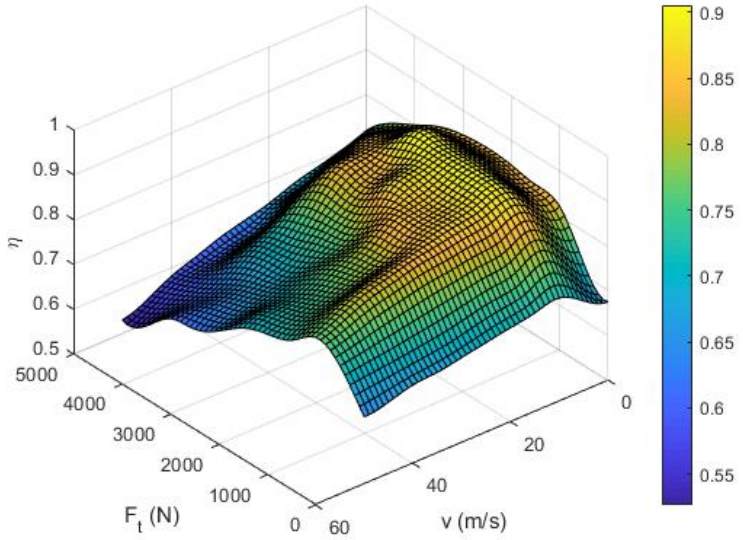
$$t_{i+1} = t_i + \frac{L_{\mathrm{s}}}{v_i} + \Delta t_i$$

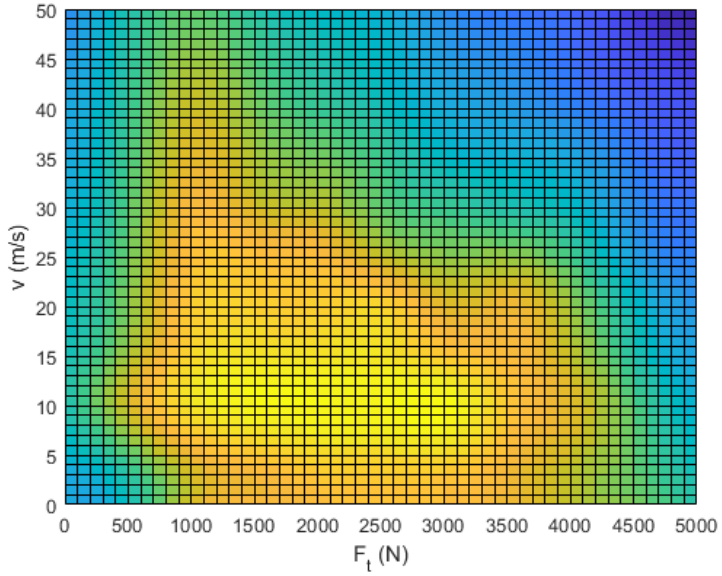
$$\zeta_{i+1} = \zeta_i - \frac{F_{\mathrm{t},i} L_{\mathrm{s}}}{\eta_i E_{\mathrm{cap}}} + \frac{P_{\mathrm{ch},i}}{E_{\mathrm{cap}}} \Delta t_i$$

embotech

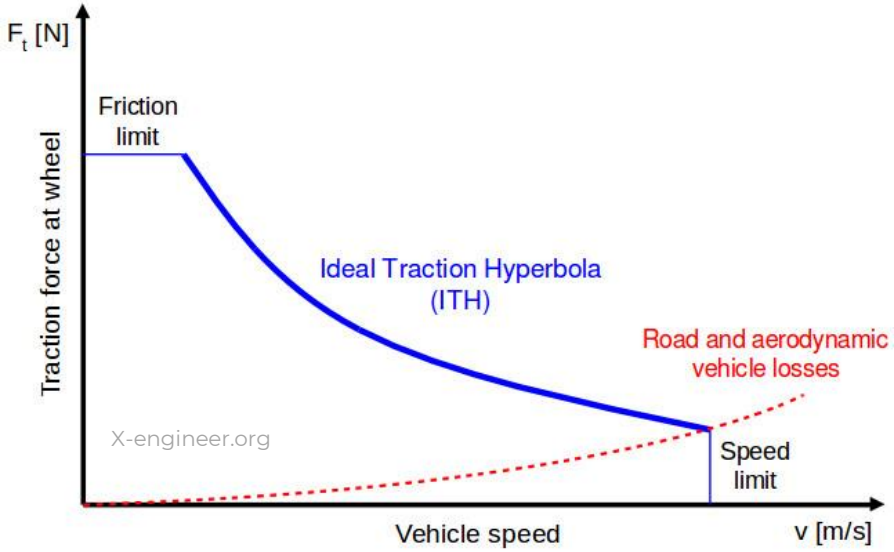
HIGHLY NONLINEAR DYNAMICS!

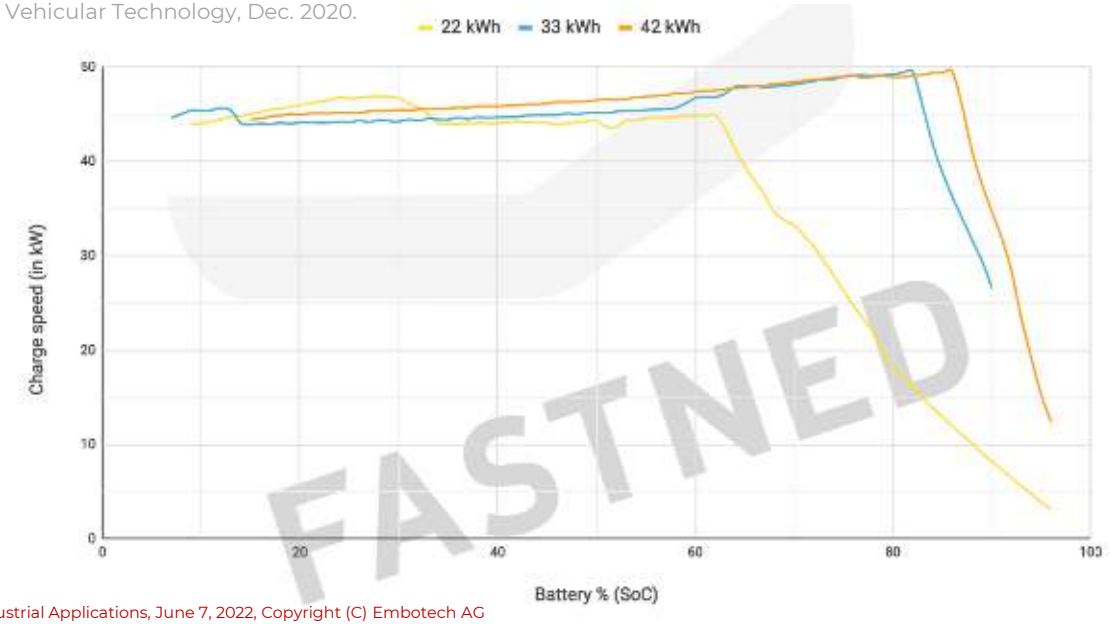




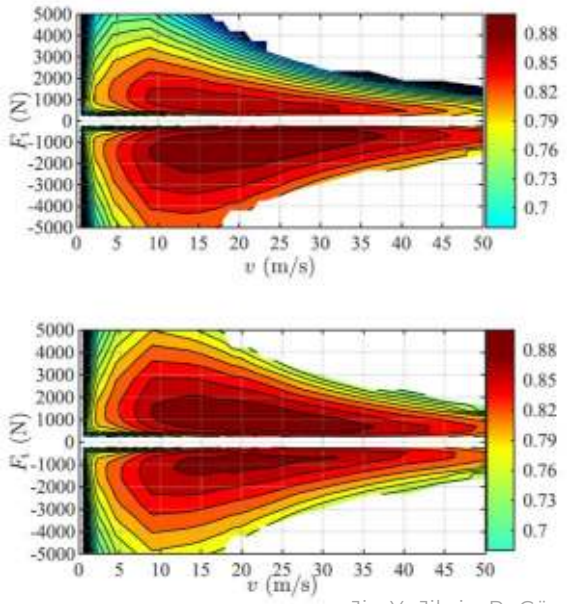


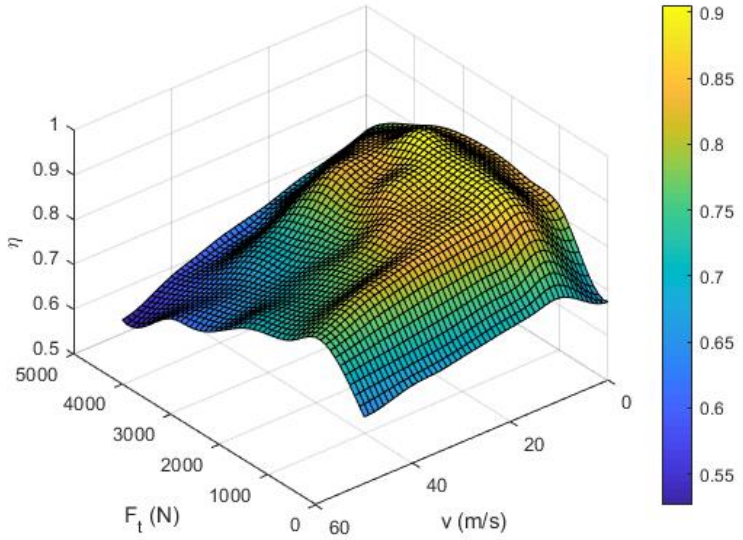
Jia, Y.; Jibrin, R.; Görges, D.: "Energy-Optimal Adaptive Cruise Control for Electric Vehicles Based On Linear and Nonlinear Model Predictive Control", in IEEE Transactions on Vehicular Technology, Dec. 2020.

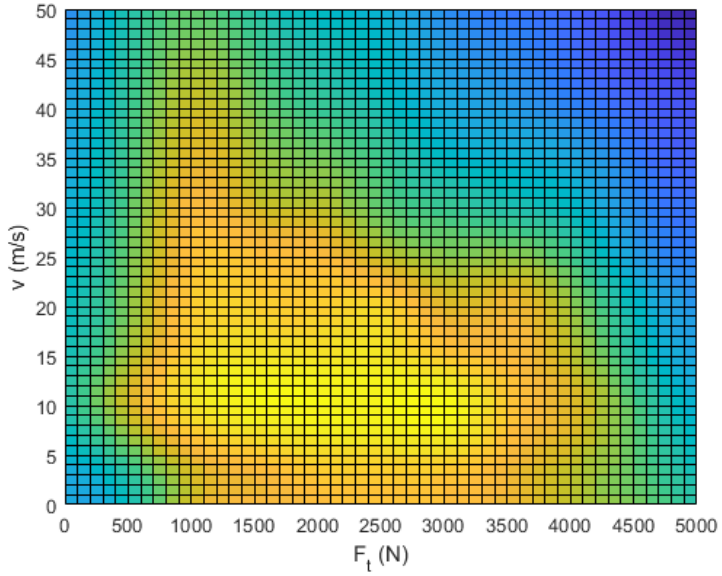




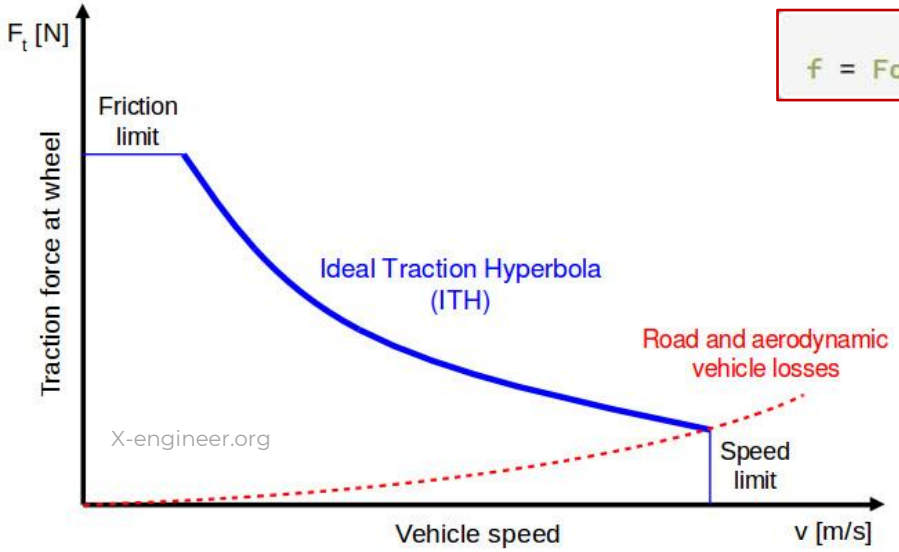
HIGHLY NONLINEAR DYNAMICS!



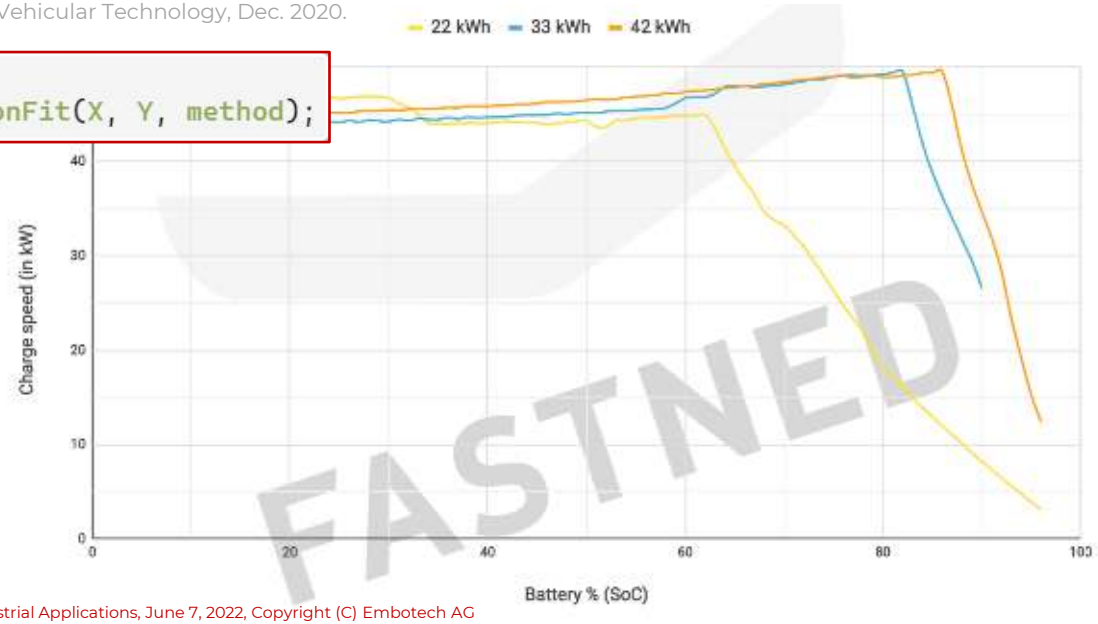




Jia, Y.; Jibrin, R.; Görges, D.: "Energy-Optimal Adaptive Cruise Control for Electric Vehicles Based On Linear and Nonlinear Model Predictive Control", in IEEE Transactions on Vehicular Technology, Dec. 2020.




```
f = ForcesInterpolationFit(X, Y, method);
```



PROBLEM CONSTRAINTS & OBJECTIVES

OBJECTIVE FUNCTION

- **Primary goal:** minimize **total travel time**, i.e. minimize terminal variable t_N at the last stage

$$\text{minimize} \quad t_N$$

PROBLEM CONSTRAINTS & OBJECTIVES

OBJECTIVE FUNCTION

- **Primary goal:** minimize **total travel time**, i.e. minimize terminal variable t_N at the last stage
- **Secondry goal:** minimize the stage cost associated with **slack variable s_i (SoC limits)**

$$\text{minimize} \quad t_N + \sum_{i=0}^{N-1} \omega_s s_i$$

PROBLEM CONSTRAINTS & OBJECTIVES

OBJECTIVE FUNCTION

- **Primary goal:** minimize **total travel time**, i.e. minimize terminal variable t_N at the last stage
- **Secondry goal:** minimize the stage cost associated with **slack variable s_i (SoC limits)**
- **Optional goals:** ensure **comfortable driving** operation and pursue **energy economy**

$$\text{minimize} \quad t_N + \sum_{i=0}^{N-1} \left(\omega_\text{s} s_i + \omega_\text{e} F_{\text{t},i}^2 + \omega_\text{b} F_{\text{b},i}^2 \right)$$

PROBLEM CONSTRAINTS & OBJECTIVES

OBJECTIVE FUNCTION

- **Primary goal:** minimize **total travel time**, i.e. minimize terminal variable t_N at the last stage
- **Secondry goal:** minimize the stage cost associated with **slack variable s_i (SoC limits)**
- **Optional goals:** ensure **comfortable driving** operation and pursue **energy economy**

$$\begin{aligned}
\text{minimize} \quad & t_N + \sum_{i=0}^{N-1} \left(\omega_\mathrm{s} s_i + \omega_\mathrm{e} F_{\mathrm{t},i}^2 + \omega_\mathrm{b} F_{\mathrm{b},i}^2 \right) \\
\text{subject to} \quad & v_{i+1} = \sqrt{\frac{2L_\mathrm{s}}{m_\mathrm{eq}} (F_{\mathrm{t},i} - F_{\mathrm{b},i} - F_{\mathrm{r},i}) + v_i^2} \\
& t_{i+1} = t_i + L_\mathrm{s} v_i^{-1} + \Delta t_i \\
& \zeta_{i+1} = \zeta_i - \frac{L_\mathrm{s}}{\eta_i E_\mathrm{cap}} F_{\mathrm{t},i} + \frac{P_{\mathrm{ch},i}}{E_\mathrm{cap}} \Delta t_i
\end{aligned}$$

CONSTRAINTS

- Equality constraints (**dynamics**)

PROBLEM CONSTRAINTS & OBJECTIVES

OBJECTIVE FUNCTION

- **Primary goal:** minimize **total travel time**, i.e. minimize terminal variable t_N at the last stage
- **Secondry goal:** minimize the stage cost associated with **slack variable s_i (SoC limits)**
- **Optional goals:** ensure **comfortable driving** operation and pursue **energy economy**

CONSTRAINTS

- Equality constraints (**dynamics**)
- State bounds (upper/lower limits on **vehicle's velocity** and **state-of-charge**)

$$
\begin{aligned}
\text{minimize} \quad & t_N + \sum_{i=0}^{N-1} \left(\omega_\mathrm{s} s_i + \omega_\mathrm{e} F_{\mathrm{t},i}^2 + \omega_\mathrm{b} F_{\mathrm{b},i}^2 \right) \\
\text{subject to} \quad & v_{i+1} = \sqrt{\frac{2L_\mathrm{s}}{m_\mathrm{eq}} (F_{\mathrm{t},i} - F_{\mathrm{b},i} - F_{\mathrm{r},i}) + v_i^2} \\
& t_{i+1} = t_i + L_\mathrm{s} v_i^{-1} + \Delta t_i \\
& \zeta_{i+1} = \zeta_i - \frac{L_\mathrm{s}}{\eta_i E_\mathrm{cap}} F_{\mathrm{t},i} + \frac{P_{\mathrm{ch},i}}{E_\mathrm{cap}} \Delta t_i \\
& \underline{v} \leq v_i \leq \bar{v}_i \\
& \underline{\zeta} - s_i \leq \zeta_i \leq \bar{\zeta} + s_i
\end{aligned}
$$

embotech

PROBLEM CONSTRAINTS & OBJECTIVES

OBJECTIVE FUNCTION

- **Primary goal:** minimize **total travel time**, i.e. minimize terminal variable t_N at the last stage
- **Secondry goal:** minimize the stage cost associated with **slack variable s_i (SoC limits)**
- **Optional goals:** ensure **comfortable driving** operation and pursue **energy economy**

CONSTRAINTS

- Equality constraints (**dynamics**)
- State bounds (upper/lower limits on **vehicle's velocity** and **state-of-charge**)
- Control input bounds (upper limits on **traction**, **braking** and permissible **charging time**)

$$
\begin{aligned}
\text{minimize} \quad & t_N + \sum_{i=0}^{N-1} \left(\omega_\mathrm{s} s_i + \omega_\mathrm{e} F_{\mathrm{t},i}^2 + \omega_\mathrm{b} F_{\mathrm{b},i}^2 \right) \\
\text{subject to} \quad & v_{i+1} = \sqrt{\frac{2L_\mathrm{s}}{m_\mathrm{eq}} \left(F_{\mathrm{t},i} - F_{\mathrm{b},i} - F_{\mathrm{r},i} \right) + v_i^2} \\
& t_{i+1} = t_i + L_\mathrm{s} v_i^{-1} + \Delta t_i \\
& \zeta_{i+1} = \zeta_i - \frac{L_\mathrm{s}}{\eta_i E_\mathrm{cap}} F_{\mathrm{t},i} + \frac{P_{\mathrm{ch},i}}{E_\mathrm{cap}} \Delta t_i \\
& \underline{v} \leq v_i \leq \bar{v}_i \\
& \underline{\zeta} - s_i \leq \zeta_i \leq \bar{\zeta} + s_i \\
& 0 \leq \Delta t_i \leq \Delta \bar{t}_i \\
& 0 \leq F_{\mathrm{t},i} \leq \bar{F}_{\mathrm{t},i} \\
& 0 \leq F_{\mathrm{b},i} \leq \bar{F}_\mathrm{b}
\end{aligned}
$$

embotech

PROBLEM CONSTRAINTS & OBJECTIVES

OBJECTIVE FUNCTION

- **Primary goal:** minimize **total travel time**, i.e. minimize terminal variable t_N at the last stage
- **Secondry goal:** minimize the stage cost associated with **slack variable s_i (SoC limits)**
- **Optional goals:** ensure **comfortable driving** operation and pursue **energy economy**

CONSTRAINTS

- Equality constraints (**dynamics**)
- State bounds (upper/lower limits on **vehicle's velocity** and **state-of-charge**)
- Control input bounds (upper limits on **traction**, **braking** and permissible **charging time**)
- Trivial **feasibility terms and initial conditions**

$$
\begin{aligned}
\text{minimize} \quad & t_N + \sum_{i=0}^{N-1} \left(\omega_\mathrm{s} s_i + \omega_\mathrm{e} F_{\mathrm{t},i}^2 + \omega_\mathrm{b} F_{\mathrm{b},i}^2 \right) \\
\text{subject to} \quad & v_{i+1} = \sqrt{\frac{2L_\mathrm{s}}{m_\mathrm{eq}} (F_{\mathrm{t},i} - F_{\mathrm{b},i} - F_{\mathrm{r},i}) + v_i^2} \\
& t_{i+1} = t_i + L_\mathrm{s} v_i^{-1} + \Delta t_i \\
& \zeta_{i+1} = \zeta_i - \frac{L_\mathrm{s}}{\eta_i E_\mathrm{cap}} F_{\mathrm{t},i} + \frac{P_{\mathrm{ch},i}}{E_\mathrm{cap}} \Delta t_i \\
& \underline{v} \leq v_i \leq \bar{v}_i \\
& \underline{\zeta} - s_i \leq \zeta_i \leq \bar{\zeta} + s_i \\
& 0 \leq \Delta t_i \leq \Delta \bar{t}_i \\
& 0 \leq F_{\mathrm{t},i} \leq \bar{F}_{\mathrm{t},i} \\
& 0 \leq F_{\mathrm{b},i} \leq \bar{F}_\mathrm{b} \\
& F_{\mathrm{t},i} F_{\mathrm{b},i} \leq \varepsilon_\mathrm{F} \\
& 0 \leq s_i \\
& x_0 = x_\mathrm{init}
\end{aligned}
$$

embotech

PROBLEM CONSTRAINTS & OBJECTIVES

```
% Smooth maximum: max = ForcesMax(a, b)
sqrt(ForcesMax(2*Ls/meq*(Ft - Fb - Fr) + v^2, 0)
```

- **Primary goal:** minimize **total travel time**, i.e. minimize terminal variable t_N at the last stage
- **Secondry goal:** minimize the stage cost associated with **slack variable s_i (SoC limits)**
- **Optional goals:** ensure **comfortable driving** operation and pursue **energy economy**

$$
\begin{aligned}
\text{minimize} \quad & t_N + \sum_{i=0}^{N-1} \left(\omega_\mathrm{s} s_i + \omega_\mathrm{e} F_{\mathrm{t},i}^2 + \omega_\mathrm{b} F_{\mathrm{b},i}^2 \right) \\
\text{subject to} \quad & v_{i+1} = \sqrt{\frac{2L_\mathrm{s}}{m_\mathrm{eq}} (F_{\mathrm{t},i} - F_{\mathrm{b},i} - F_{\mathrm{r},i}) + v_i^2} \\
& t_{i+1} = t_i + L_\mathrm{s} v_i^{-1} + \Delta t_i \\
& \zeta_{i+1} = \zeta_i - \frac{L_\mathrm{s}}{\eta_i E_\mathrm{cap}} F_{\mathrm{t},i} + \frac{P_{\mathrm{ch},i}}{E_\mathrm{cap}} \Delta t_i \\
& \underline{v} \leq v_i \leq \bar{v}_i \\
& \underline{\zeta} - s_i \leq \zeta_i \leq \bar{\zeta} + s_i \\
& 0 \leq \Delta t_i \leq \Delta \bar{t}_i \\
& 0 \leq F_{\mathrm{t},i} \leq \bar{F}_{\mathrm{t},i} \\
& 0 \leq F_{\mathrm{b},i} \leq \bar{F}_\mathrm{b} \\
& F_{\mathrm{t},i} F_{\mathrm{b},i} \leq \varepsilon_\mathrm{F} \\
& 0 \leq s_i \\
& x_0 = x_\mathrm{init}
\end{aligned}
$$

CONSTRAINTS

- Equality constraints (**dynamics**)
- State bounds (upper/lower limits on **vehicle's velocity** and **state-of-charge**)
- Control input bounds (upper limits on **traction**, **braking** and permissible **charging time**)
- Trivial **feasibility terms and initial conditions**

embotech

MUNICH – COLOGNE TRIP

ROUTE PROFILE

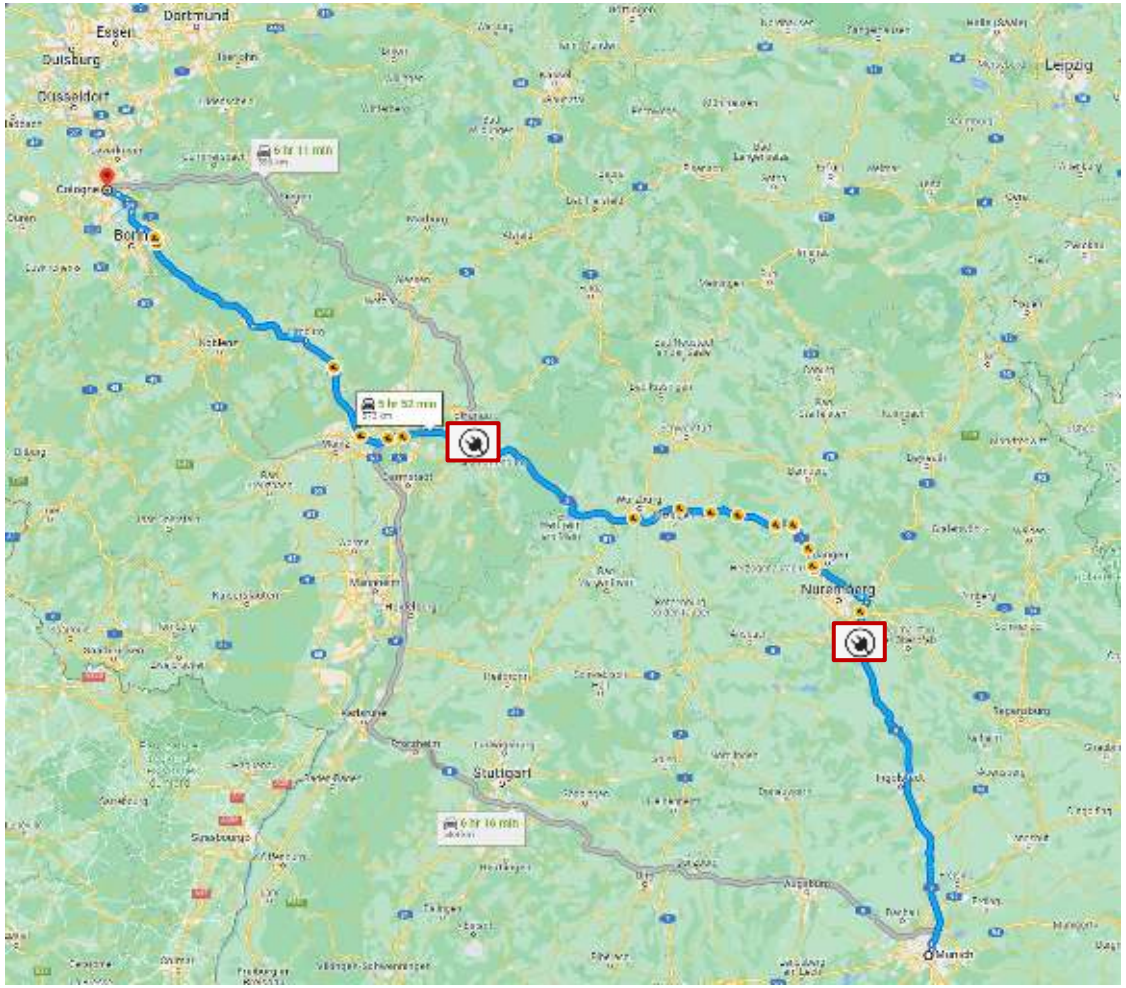
- **573km trip** with **1km discretization steps**
- **50kW DC charging stations** available **at 150km** and **380km** mark

VEHICLE SPECIFICATION

- **BMW i3** (120Ah - 42kWh)
- Max speed: **150km/h**
- Max range: **260km**
- SoC limits: **10% - 90%**

TRIP OBJECTIVE

- Travel **from Munich to Cologne** in the **shortest amount of time** (drive + charge)



embotech

MUNICH – COLOGNE TRIP

ROUTE PROFILE

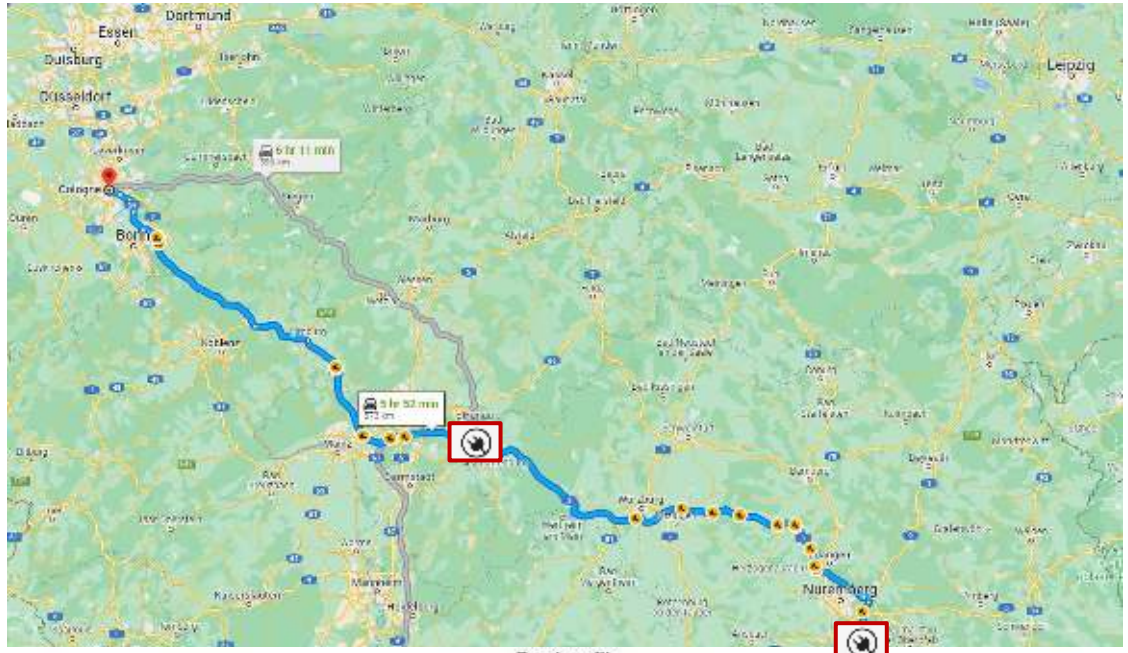
- **573km trip** with **1km discretization steps**
- **50kW DC charging stations** available **at 150km** and **380km** mark

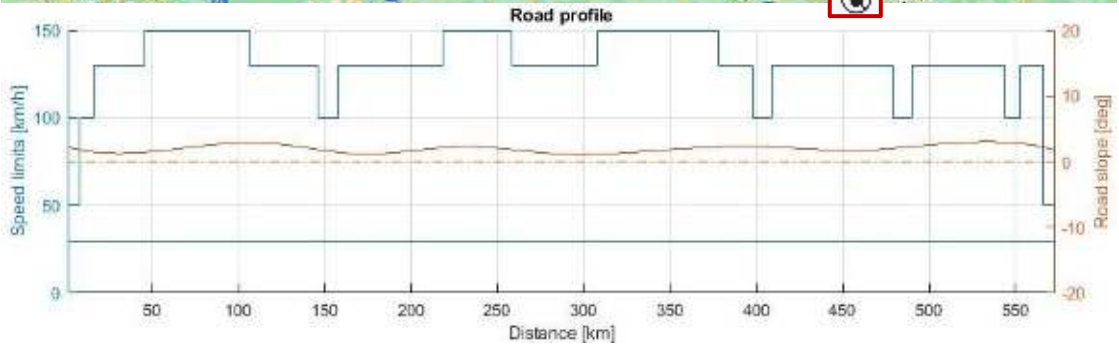
VEHICLE SPECIFICATION

- **BMW i3** (120Ah - 42kWh)
- Max speed: **150km/h**
- Max range: **260km**
- SoC limits: **10% - 90%**

TRIP OBJECTIVE

- Travel **from Munich to Cologne** in the **shortest amount of time** (drive + charge)





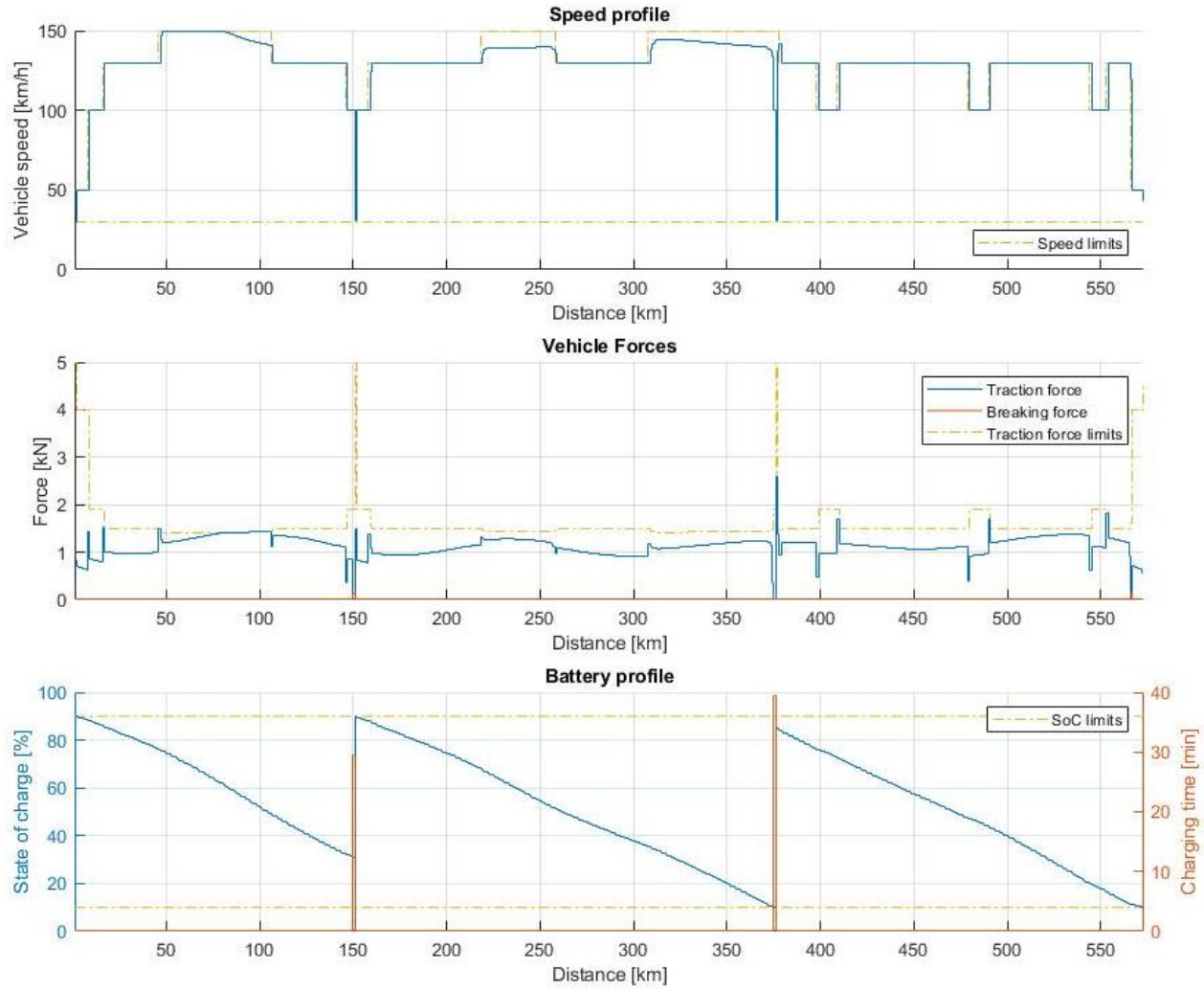
embotech

MUNICH – COLOGNE TRIP

OBSERVATIONS

- Charging is optimally done such that the vehicle arrives with minimum permissible SoC level (thus spending less time charging)

- Vehicle speed is adjusted (sometimes below maximum road speed limit) as a trade-off between the driving and charging time



Total travel time	Total charging time
399.13min	76.19min

MUNICH – COLOGNE TRIP

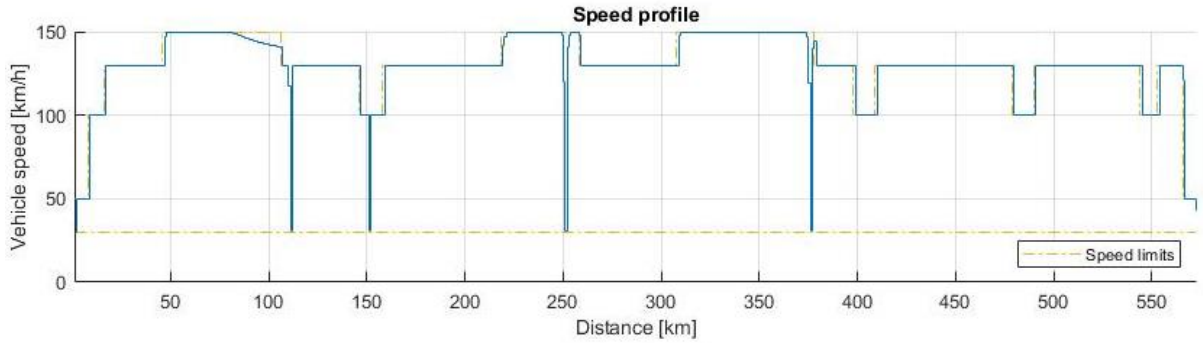
OBSERVATIONS

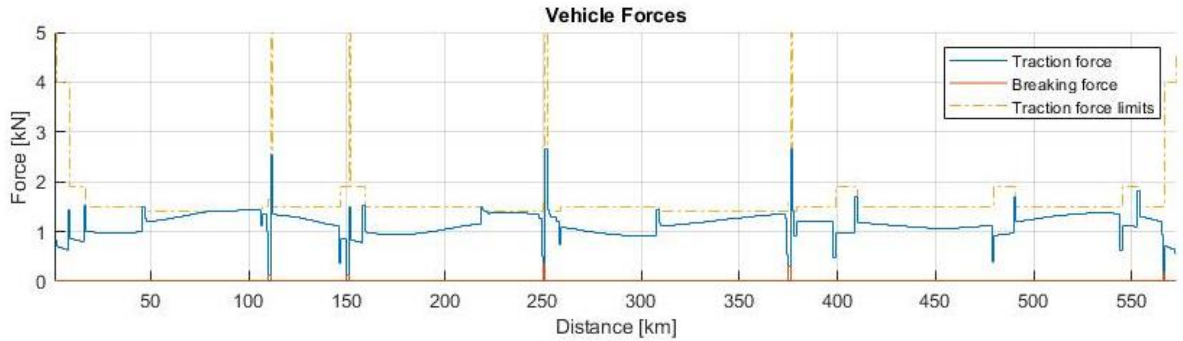
- Charging is optimally done such that the vehicle arrives with minimum permissible SoC level (thus spending less time charging)

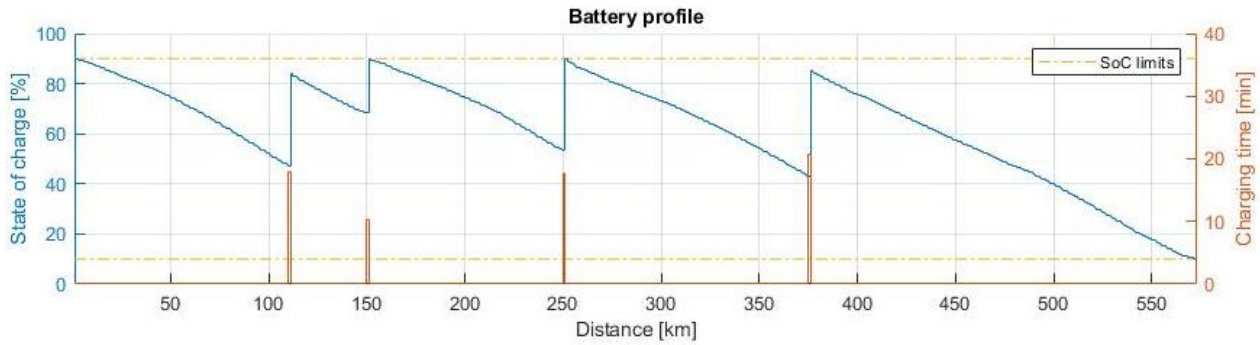
- Vehicle speed is adjusted (sometimes below maximum road speed limit) as a trade-off between the driving and charging time

- Higher charger availability **shortens the trip time by 7.35%**, despite the total **charging time** being **9.82% higher**

Total travel time	Total charging time
371.82min	84.49min





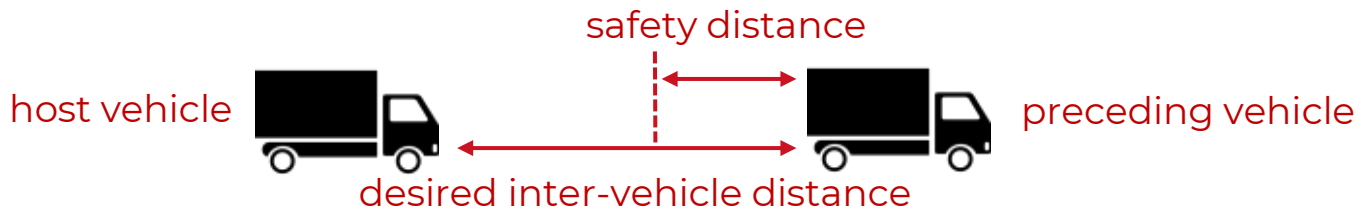


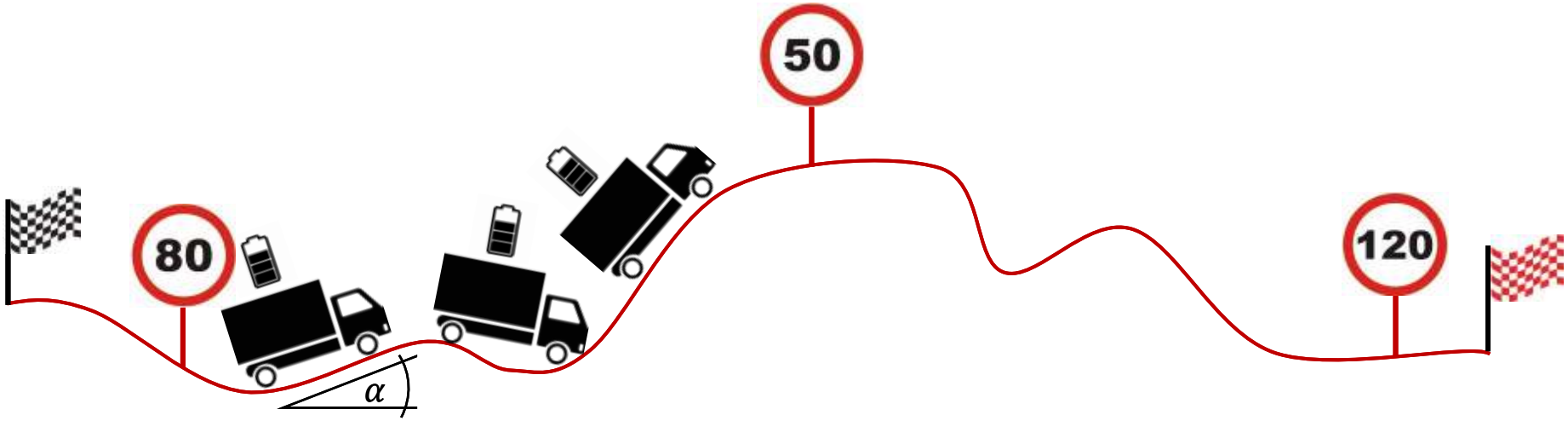
4 PLATOON ADAPTIVE CRUISE CONTROL

PROBLEM OVERVIEW

Goal: Develop **adaptive cruise control** of an EV truck driving **in a platoon**

- Volvo VNR Electric 4x2 Tractors
- Preserving **safety** and **desired inter-vehicle distance**
- Ensure **ride quality**





embotech

PROBLEM FORMULATION

- Penalize **jerking**, excessive **breaking** and **poor inter-vehicle distance tracking**
- Vehicle dynamics implemented in **time domain**
- Host vehicle must **keep a minimum safety distance** from a preceding vehicle

$$
\begin{aligned}
\text{minimize} \quad & \sum_{i=0}^{N-1} \left(\omega_\mathrm{j} \Delta F_{\mathrm{t},i}^2 + \omega_\mathrm{b} F_{\mathrm{b},i}^2 + \omega_\mathrm{d} (d_\mathrm{des} - d_i)^2 \right) \\
\text{subject to} \quad & F_{\mathrm{t},i+1} = F_{\mathrm{t},i} + \Delta F_{\mathrm{t},i} \\
& d_{i+1} = d_i + T_s (v_{\mathrm{p},i} - v_i) \\
& v_{i+1} = v_i + \frac{T_\mathrm{s}}{m_\mathrm{eq}} (F_{\mathrm{t},i} - F_{\mathrm{b},i} - F_{\mathrm{r},i}) \\
& s_{i+1} = s_i + T_\mathrm{s} v_i \\
& \zeta_{i+1} = \zeta_i - \frac{T_\mathrm{s}}{\eta_i E_\mathrm{cap}} v_i F_{\mathrm{t},i} \\
& \underline{v} \leq v_i \leq \bar{v}_i \\
& \underline{\zeta} \leq \zeta_i \leq \bar{\zeta} \\
& 0 \leq F_{\mathrm{t},i} \leq F_{\mathrm{t},i} \\
& 0 \leq F_{\mathrm{b},i} \leq F_\mathrm{b} \\
& F_{\mathrm{t},i} F_{\mathrm{b},i} \leq \varepsilon_\mathrm{F} \\
& d_\mathrm{safe} \leq d_i \\
& x_0 = x_\mathrm{init}
\end{aligned}
$$

CODE WORKFLOW

Trip parameters

Vehicle parameters

Problem formulation & solver generation

Solver runs for each simulation step

Graphical representation of results

```
%% Trip parameters
% Sampling time, i.e. temporal discretization (s)
trip.Ts = 1;
% Prediction horizon lentgh (s)
trip.horizon = 10;
% Trip lentgh (min)
trip.length = 30;
% Initial inter-vehicle distance (m)
trip.initDist = 50;
% Initial host vehicle's SoC
trip.initSoC = 0.75;
% Initial host vehicle's speed
trip.initSpeed = 30;

%% Vehicle parameters
param = defineVehicleParameters();

%% Formulate the problem and generate the solver
model = generateACCProfile(trip, param);

%% Simulate the model
sim = runSimulation(trip, param, model);

%% Plot simulation results
plotResults(trip, param, sim);
```

embotech

CODE WORKFLOW

Trip parameters

Vehicle parameters

Problem formulation & solver generation

Solver runs for each simulation step

Graphical representation of results

```
%% Trip parameters
% Sampling time, i.e. temporal discretization (s)
trip.Ts = 1;
% Prediction horizon lentgh (s)
trip.horizon = 10;
% Trip lentgh (min)
trip.length = 30;
% Initial inter-vehicle distance (m)
trip.initDist = 50;
% Initial host vehicle's SoC
trip.initSoC = 0.75;
% Initial host vehicle's speed
trip.initSpeed = 30;

%% Vehicle parameters
param = defineVehicleParameters();

%% Formulate the problem and generate the solver
model = generateACCProfile(trip, param);

%% Simulate the model
sim = runSimulation(trip, param, model);

%% Plot simulation results
plotResults(trip, param, sim);
```

embotech

VEHICLE PARAMETERS

```
function param = defineVehicleParameters()
    % Gravitational acceleration (m/s^2)
    param.g = 9.81;
    % Vehicle mass (kg)
    param.mv = 8000;
    % Mass factor (-)
    param.eI = 1.06;
    % Equivalent mass (kg)
    param.meq = (1+param.eI)*param.mv;
    % Projected frontal area (m^2)
    param.Af = 8.38;
    % Air density (kg/m^3)
    param.rhoa = 1.206;
    % Maximum traction force (N)
    param.FtMax = 20e3;
    % Maximum breaking force (N)
    param.FbMax = 10e3;
    % Battery's energy capacity (Wh)
    param.Ecap = 264e3;
    % Minimum vehicle velocity (km/h)
    param.vMin = 0;
    % Maximum vehicle velocity (km/h)
    param.vMax = 109;
    % Minimum SoC (-)
    param.SoCmin = 0.1;
    % Maximum SoC (-)
    param.SoCmax = 0.9;
    % Desired distance to preceding vehicle (m)
    param.ddes = 30;
    % Safety distance to preceding vehicle (m)
    param.dsafe = 5;
    ...
```

VEHICLE PARAMETERS

```
function param = defineVehicleParameters()
    % Gravitational acceleration (m/s^2)
    param.g = 9.81;
    % Vehicle mass (kg)
    param.mv = 8000;
    % Mass factor (-)
    param.eI = 1.06;
    % Equivalent mass (kg)
    param.meq = (1+param.eI)*param.mv;
    % Projected frontal area (m^2)
    param.Af = 8.38;
    % Air density (kg/m^3)
    param.rhoa = 1.206;
    % Maximum traction force (N)
    param.FtMax = 20e3;
    % Maximum breaking force (N)
    param.FbMax = 10e3;
    % Battery's energy capacity (Wh)
    param.Ecap = 264e3;
    % Minimum vehicle velocity (km/h)
    param.vMin = 0;
    % Maximum vehicle velocity (km/h)
    param.vMax = 109;
    % Minimum SoC (-)
    param.SoCmin = 0.1;
    % Maximum SoC (-)
    param.SoCmax = 0.9;
    % Desired distance to preceding vehicle (m)
    param.ddes = 30;
    % Safety distance to preceding vehicle (m)
    param.dsafe = 5;
    ...
```

```
    ...
    % Motor efficiency (-)
    param.eta = setupMotorEfficiency(param.FtMax);
    % Traction force hyperbola (upper limit) (N)
    param.FtMaxHyp = setupTractionForceHyperbola(param.FtMax);
    % Air drag coefficient (-)
    param.ca = setupAirDragCoeff();
    % Rolling resistance coefficient (-)
    param.cr = setupRollingResistanceCoeff;
end
```

VEHICLE PARAMETERS

```
function param = defineVehicleParameters()
    % Gravitational acceleration (m/s^2)
    param.g = 9.81;
    % Vehicle mass (kg)
    param.mv = 8000;
    % Mass factor (-)
    param.eI = 1.06;
    % Equivalent mass (kg)
    param.meq = (1+param.eI)*param.mv;
    % Projected frontal area (m^2)
    param.Af = 8.38;
    % Air density (kg/m^3)
    param.rhoa = 1.206;
    % Maximum traction force (N)
    param.FtMax = 20e3;
    % Maximum breaking force (N)
    param.FbMax = 10e3;
    % Battery's energy capacity (Wh)
    param.Ecap = 264e3;
    % Minimum vehicle velocity (km/h)
    param.vMin = 0;
    % Maximum vehicle velocity (km/h)
    param.vMax = 109;
    % Minimum SoC (-)
    param.SoCmin = 0.1;
    % Maximum SoC (-)
    param.SoCmax = 0.9;
    % Desired distance to preceding vehicle (m)
    param.ddes = 30;
    % Safety distance to preceding vehicle (m)
    param.dsafe = 5;
    ...
```

```
...
    % Motor efficiency (-)
    param.eta = setupMotorEfficiency(param.FtMax);
    % Traction force hyperbola (upper limit) (N)
    param.FtMaxHyp = setupTractionForceHyperbola(param.FtMax);
    % Air drag coefficient (-)
    param.ca = setupAirDragCoeff();
    % Rolling resistance coefficient (-)
    param.cr = setupRollingResistanceCoeff;
end
```

```
f = ForcesInterpolationFit(X, Y, method);
```

method	Description
'linear'	Piecewise linear
'nearest'	Piecewise constant, value from nearest data point
'next'	Piecewise constant, value from next data point
'previous'	Piecewise constant, value from previous data point
'spline' (default)	Piecewise cubic spline
'pchip'	Shape-preserving piecewise cubic spline

VEHICLE PARAMETERS - SPLINES

```
    % Motor efficiency (-)
    param.eta = setupMotorEfficiency(param.FtMax);
    % Traction force hyperbola (upper limit) (N)
    param.FtMaxHyp = setupTractionForceHyperbola(param.FtMax);
    % Air drag coefficient (-)
    param.ca = setupAirDragCoeff();
    % Rolling resistance coefficient (-)
    param.cr = setupRollingResistanceCoeff;
end
```

```
function etaMotorSpline = setupMotorEfficiency(FtMax)
    FtSample = 0:0.1*FtMax:FtMax;
    etaSample = [0.835, 0.865, 0.9, 0.85, 0.79, 0.75, 0.72, 0.72, 0.72, 0.72, 0.72];
    etaMotorSpline = ForcesInterpolationFit(FtSample,etaSample);
end

function FtMaxHypSpline = setupTractionForceHyperbola(FtMax)
    vSample = [0.25, 0.4, 0.6, 0.8, 1.0]*kmh2ms(140);
    FtMaxHypSample = [1.0, 0.67, 0.43, 0.32, 0.28]*FtMax;
    FtMaxHypSpline = ForcesInterpolationFit(vSample,FtMaxHypSample);
end
```

embotech

VEHICLE PARAMETERS - SPLINES

```
function caSpline = setupAirDragCoeff()
    dSample = [0, 50, 1e5];
    caSample = [0.15, 0.3, 0.3];
    caSpline = ForcesInterpolationFit(dSample,caSample,'linear');
end

function crSpline = setupRollingResistanceCoeff()
    vSample = [0, 140];
    crSample = [0.01, 0.02];
    crSpline = ForcesInterpolationFit(vSample,crSample);
end
```

```
...
    % Motor efficiency (-)
    param.eta = setupMotorEfficiency(param.FtMax);
    % Traction force hyperbola (upper limit) (N)
    param.FtMaxHyp = setupTractionForceHyperbola(param.FtMax);
    % Air drag coefficient (-)
    param.ca = setupAirDragCoeff();
    % Rolling resistance coefficient (-)
    param.cr = setupRollingResistanceCoeff;
end
```

```
function etaMotorSpline = setupMotorEfficiency(FtMax)
    FtSample = 0:0.1*FtMax:FtMax;
    etaSample = [0.835, 0.865, 0.9, 0.85, 0.79, 0.75, 0.72, 0.72, 0.72, 0.72, 0.72];
    etaMotorSpline = ForcesInterpolationFit(FtSample,etaSample);
end

function FtMaxHypSpline = setupTractionForceHyperbola(FtMax)
    vSample = [0.25, 0.4, 0.6, 0.8, 1.0]*kmh2ms(140);
    FtMaxHypSample = [1.0, 0.67, 0.43, 0.32, 0.28]*FtMax;
    FtMaxHypSpline = ForcesInterpolationFit(vSample,FtMaxHypSample);
end
```

embotech

PROBLEM FORMULATION & SOLVER GENERATION

Trip parameters

Vehicle parameters

Problem formulation & solver generation

Solver runs for each simulation step

Graphical representation of results

```
%% Trip parameters
% Sampling time, i.e. temporal discretization (s)
trip.Ts = 1;
% Prediction horizon lentgh (s)
trip.horizon = 10;
% Trip lentgh (min)
trip.length = 30;
% Initial inter-vehicle distance (m)
trip.initDist = 50;
% Initial host vehicle's SoC
trip.initSoC = 0.75;
% Initial host vehicle's speed
trip.initSpeed = 30;

%% Vehicle parameters
param = defineVehicleParameters();

%% Formulate the problem and generate the solver
model = generateACCProfile(trip, param);

%% Simulate the model
sim = runSimulation(trip, param, model);

%% Plot simulation results
plotResults(trip, param, sim);
```

embotech

PROBLEM FORMULATION

```
function model = generateACCProfile(trip, param)
% Assume variable ordering zi = [du(i); u{i}; x{i}] for i=1 ... N
% zi = [Ft(i+1) - Ft(i); Ft(i); Fb(i) d(i); v(i); s(i); SoC(i)]
% pi = [vp(i); vMax(i); vMin(i); alpha(i)]

    %% Problem dimensions
    nx = 4;
    nu = 2;
    ndu = 1;
    np = 4;
    nh = 4;

    model.N = trip.horizon/trip.Ts;     % horizon length
    model.nvar = nx + nu + ndu;         % number of variables
    model.neq  = nx + ndu;              % number of equality constraints
    model.nh   = nh;                    % number of inequality constraints
    model.npar = np;                    % number of runtime parameters

    %% Objective function
    % R(1,1) penalizes jerking; R(2,2) penalizes braking; Q penalizes poor
    % tracking (i.e. catch up) of the preceding vehicle
    R = [ 0.01,  0;
          0,    0.1];
    Q = 1;

    model.objective = @(z) [z(1);z(3)]'*R*[z(1);z(3)] + Q*(z(4) - param.ddes)^2;   ...
```

PROBLEM FORMULATION

```
...
%% Dynamics, i.e. equality constraints
model.eq = @(z,p) [z(1) + z(2);
                   z(4) + trip.Ts*(p(1) - z(5));
                   z(5) + trip.Ts/param.meq*(z(2) - z(3) - param.cr(z(5))*param.mv*param.g*cos(p(4))
                        - param.mv*param.g*sin(p(4)) - 0.5*param.ca(z(4))*param.Af*param.rhoa*z(5)^2);
                   z(6) + trip.Ts*z(5);
                   z(7) - trip.Ts*z(5)*z(2)/(3600*param.eta(z(2))*param.Ecap)];

% adding a row for Ft(i+1) = Ft(i) + dFt(i)
model.E = [0 1 0 0 0 0 0; zeros(nx,nu+ndu), eye(nx)];

%% Inequality constraints
% Upper/lower variable bounds lb <= z <= ub
%                            inputs                |                    states
%               dFt       Ft           Fb          | d              v          s          SoC
model.lb = [-param.FtMax, 0,           0.,           0.,   kmh2ms(param.vMin), 0.,   param.SoCmin];
model.ub = [param.FtMax,  param.FtMax, param.FbMax, +inf, kmh2ms(param.vMax), +inf, param.SoCmax];

% Nonlinear inequalities hl <= h(z,p) <= hu
model.ineq = @(z,p) [z(5) - p(2);
                     p(3) - z(5);
                     z(2) - param.FtMaxHyp(z(5));
                     param.dsafe - z(4)];

% Upper/lower bounds for inequalities
model.hu = [0,    0,    0,    0];
model.hl = [-inf, -inf, -inf, -inf];

%% Initial and final conditions
model.xinitidx = nu+ndu+1:nu+ndu+nx;  ...
```

SOLVER GENERATION

```
...
    %% Generate FORCESPRO solver
    % Define solver options
    codeoptions = getOptions('FORCESNLPsolver');
    codeoptions.printlevel = 0;
    codeoptions.nlp.compact_code = 1;

    % Generate code
    FORCES_NLP(model, codeoptions);
end
```

SIMULATION

Trip parameters

Vehicle parameters

Problem formulation & solver generation

Solver runs for each simulation step

Graphical representation of results

```
%% Trip parameters
% Sampling time, i.e. temporal discretization (s)
trip.Ts = 1;
% Prediction horizon lentgh (s)
trip.horizon = 10;
% Trip lentgh (min)
trip.length = 30;
% Initial inter-vehicle distance (m)
trip.initDist = 50;
% Initial host vehicle's SoC
trip.initSoC = 0.75;
% Initial host vehicle's speed
trip.initSpeed = 30;

%% Vehicle parameters
param = defineVehicleParameters();

%% Formulate the problem and generate the solver
model = generateACCProfile(trip, param);

%% Simulate the model
sim = runSimulation(trip, param, model);

%% Plot simulation results
plotResults(trip, param, sim);
```

embotech

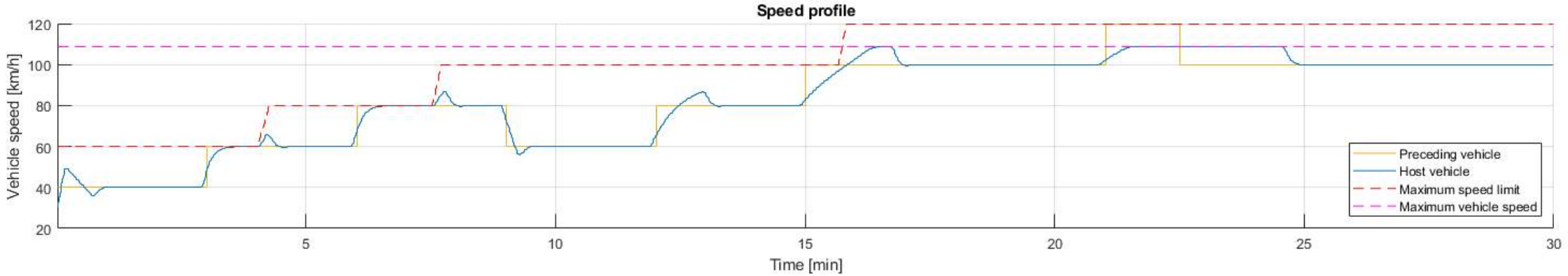
SIMULATION

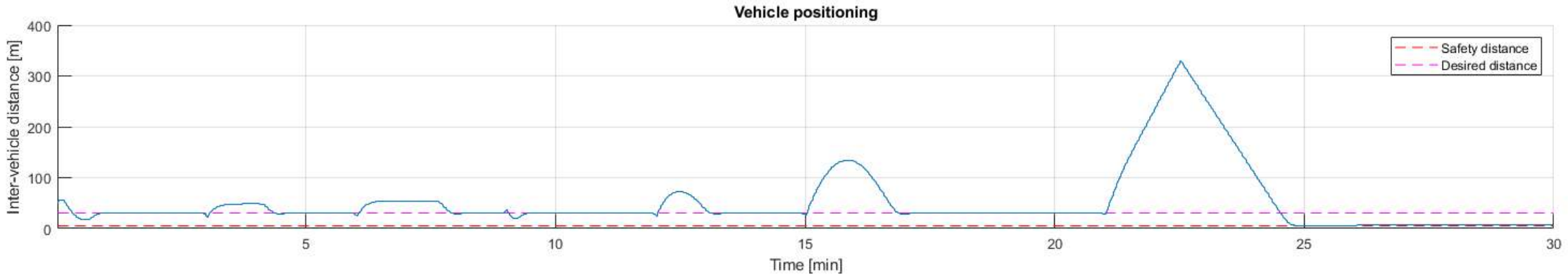
```
function sim = runSimulation(trip, param, model)
    %% Define spatially and temporaly distributed (i.e. stage-dependent) parameters
    % Preceding vehicles's speed profile
    sim.vp = setupPrecedingVehicleSpeed(trip);

    % Road speed limits and slope angles
    [vMaxRoad, vMinRoad, alpha] = setupRoadParameters(trip);

    %% Initialize the problem
    problem.x0 = zeros(model.N*model.nvar,1);
    sim.kMax = trip.length*60/trip.Ts;
    sim.X = zeros(nx,sim.kMax+1);
    sim.U = zeros(nu,sim.kMax);

    % X(1) = [d(1); v(1); s(1); SoC(1)]
    sim.X(:,1) = [trip.initDist; kmh2ms(trip.initSpeed); 0; trip.initSoC];   ...
```

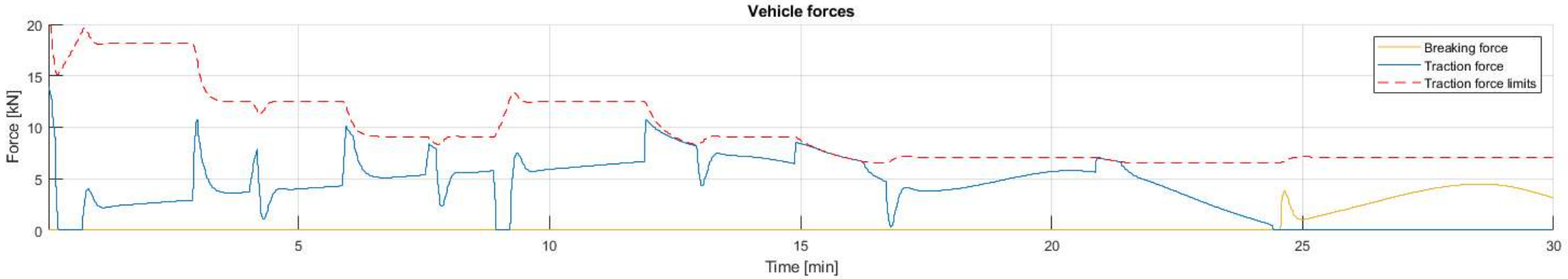
SIMULATION

```
...  %% Simulation
    for k = 1:sim.kMax
        % Set initial condition
        problem.xinit = sim.X(:,k);

        % Set runtime parameters
        problem.all_parameters = zeros(np*model.N,1);
        dist = sim.X(3,k);
        for i = 1:model.N
            problem.all_parameters((i-1)*np+1:i*np) = [kmh2ms(sim.vp(k+i-1)); kmh2ms(vMaxRoad(dist));
                                                       kmh2ms(vMinRoad(dist)); alpha(dist)];
        end

        % Call solver
        [solverout,exitflag,info] = FORCESNLPsolver(problem);

        % Update state and control vector
        if exitflag == 1
            Z = solverout.x01;
            sim.dU(:,k) = Z(1);
            sim.U(:,k) = Z(2:nu+ndu);
            update = model.eq([sim.dU(:,k);sim.U(:,k);sim.X(:,k)], problem.all_parameters(1:np))';
            sim.X(:,k+1) = update(ndu+1:model.neq);
            if k < sim.kMax
                sim.U(:,k+1) = update(1);
            end
        else
            error('Some problem in solver');
        end
    end
end
```

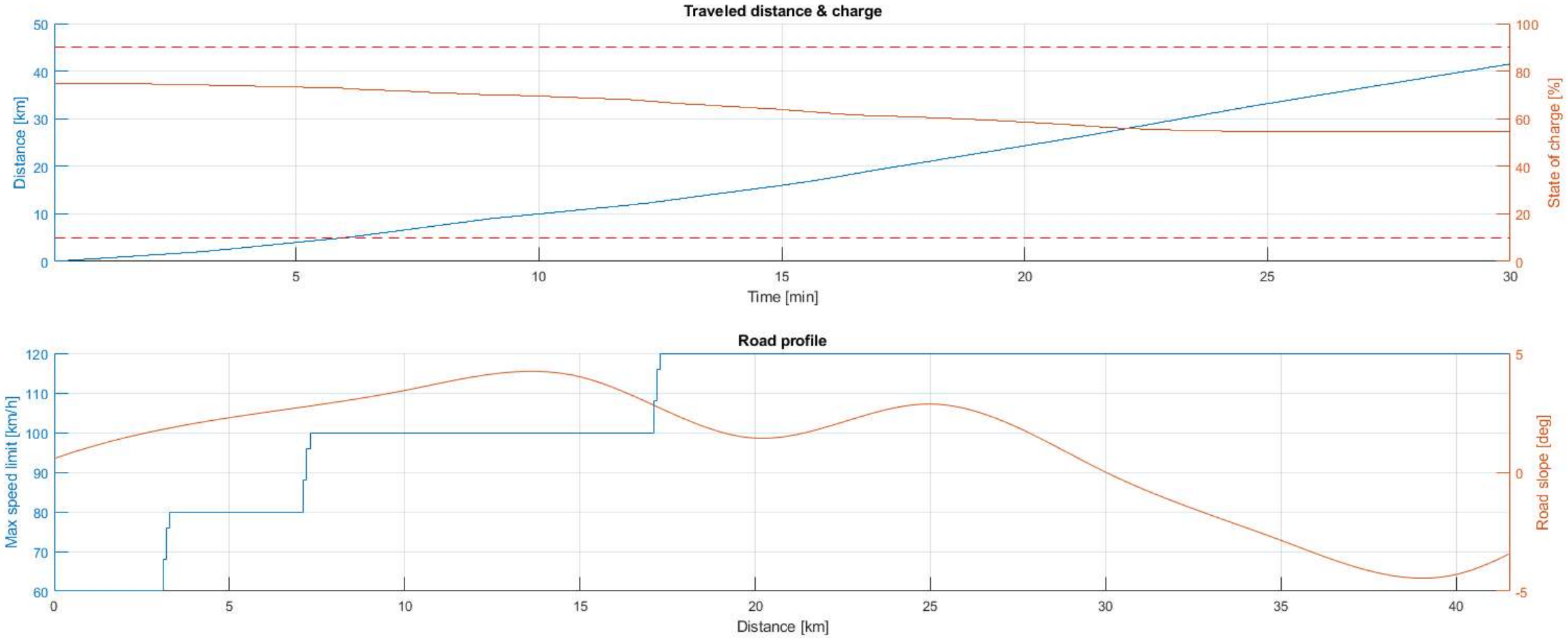
CONTROL PERFORMANCE







CONTROL PERFORMANCE



FORCESPRO takes **42 iterations** and **0.63ms** to solve the problem on average

Simulations performed on x-64-based Intel® Core™ i7-10750H CPU @ 2.60GHz with 16GB RAM

embotech

embotech.com