Dynamic Optimization

Innovate 2022

Day 2 – October 20th , Stockholm

Stéphane Velut

Industry Director, Energy

At Modelon since 2009

PhD in Controls from Lund University, Sweden (2005)

M. Sc. E.E, Grenoble Institute of Technology, France (1999)

Optimization versus Simulation

What question to ask your model?

Simulation

What if… ?

- The valve is closed
- The valve is made smaller
- The upstream pressure is increased
- The controller is tuned differently

Optimization

What is best ?

- In terms of equipment and controls
- With regard to some criteria
- And operational constraints

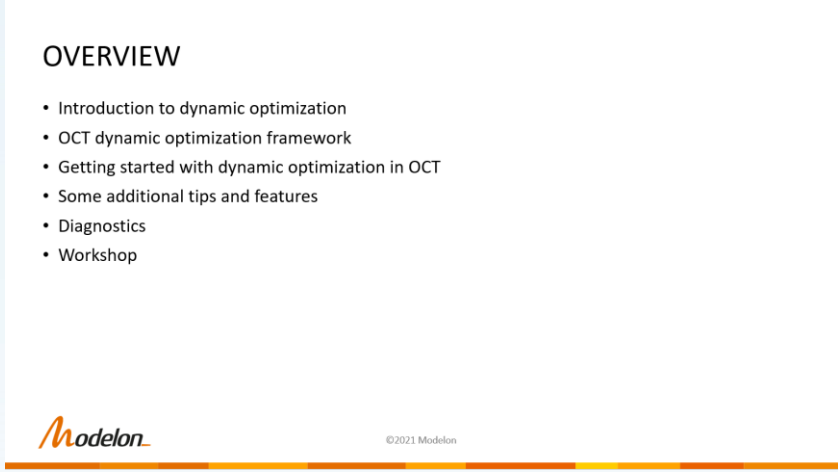
Optimization approaches with Modelon Impact

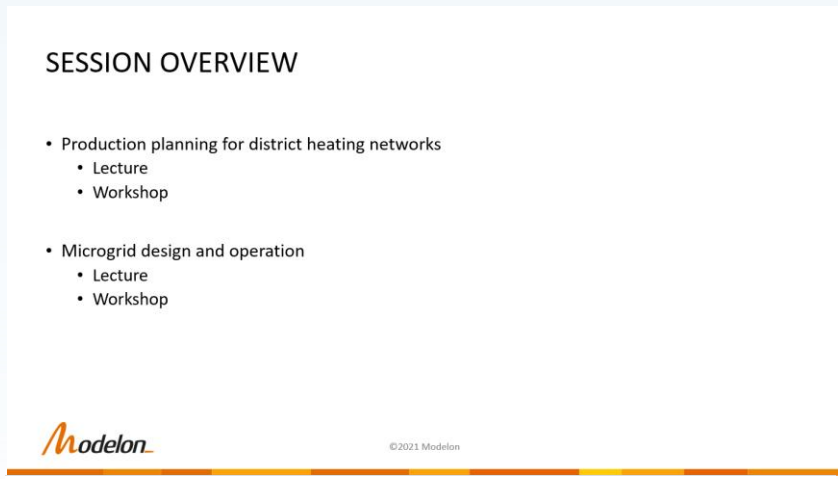
- FMU-based
  - Both derivative-free and gradient-based
    - directional derivatives can be provided by Modelon Impact compiler
  - Works for almost any Modelica model but limited number of degrees of freedom
  - Workflow in Python using PyFMI, OCT API and any optimization package
- Collocation-based
  - Handles a large number of degrees of freedom (trajectory optimization)
  - Computationally efficient and accurate
  - Moderate number of states
  - Requires the model equations to be smooth
  - Accessible from Python and (partially) integrated as Custom Function

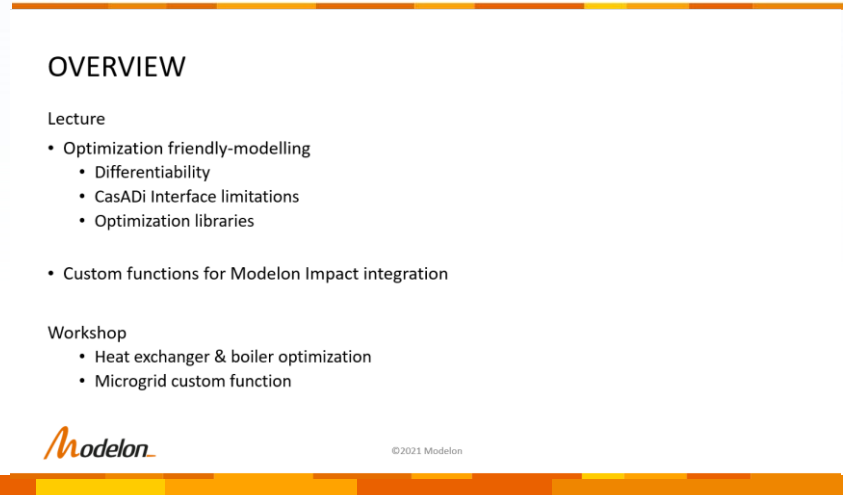
Modelon

Advanced Training

- 1 full day of training
- 3 lectures and 3 workshops covering
  - Basics
  - Applications
  - Modelling







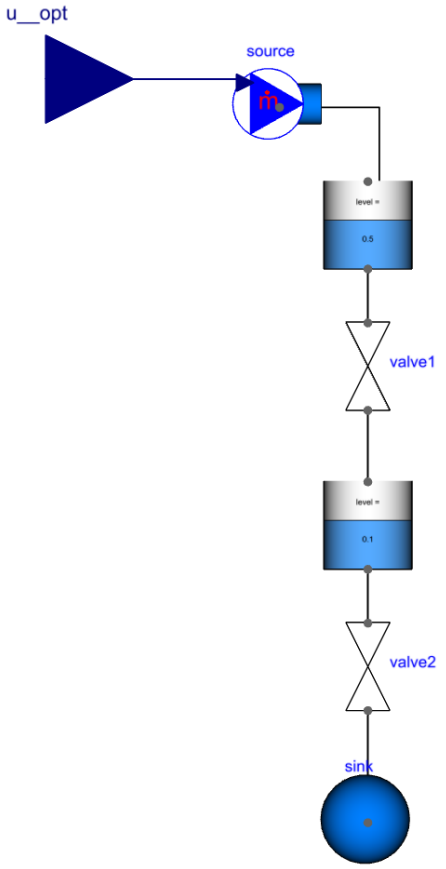
Outline

- A first example
- Under the hood
- Optimization friendly modelling
- Application examples
  - Energy
  - Aerospace
  - Automotive

Modelon

A first example!

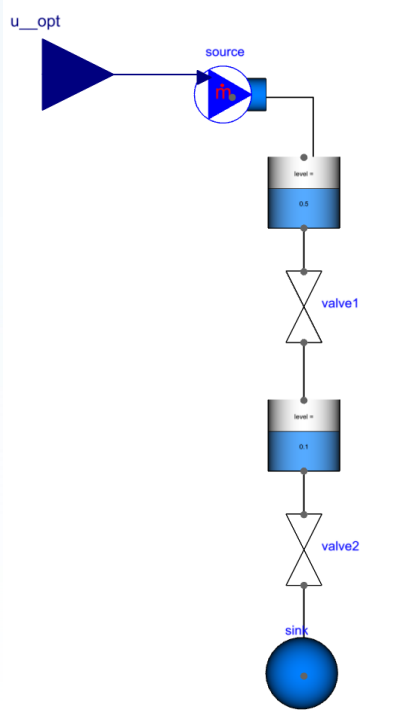
Optimal control of the double tank process



DEMO

Optimal control of the double tank

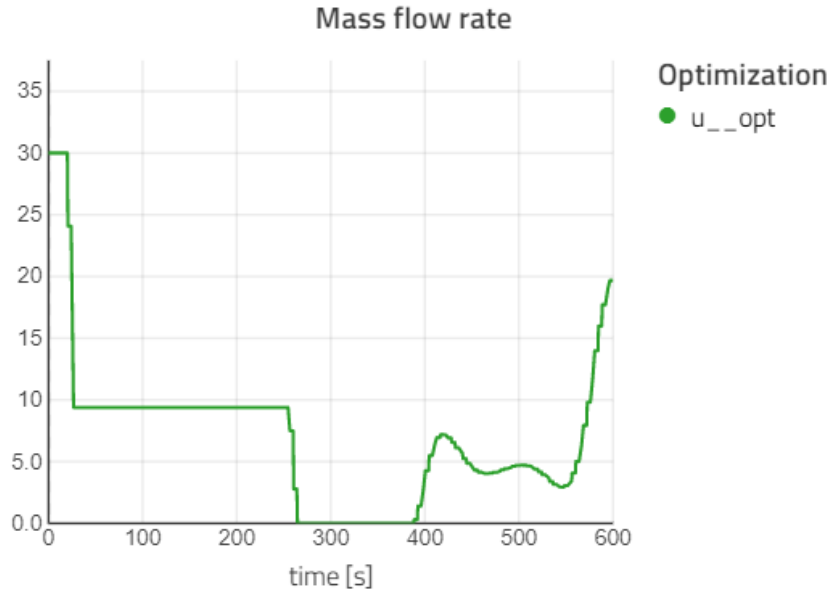
Plant model

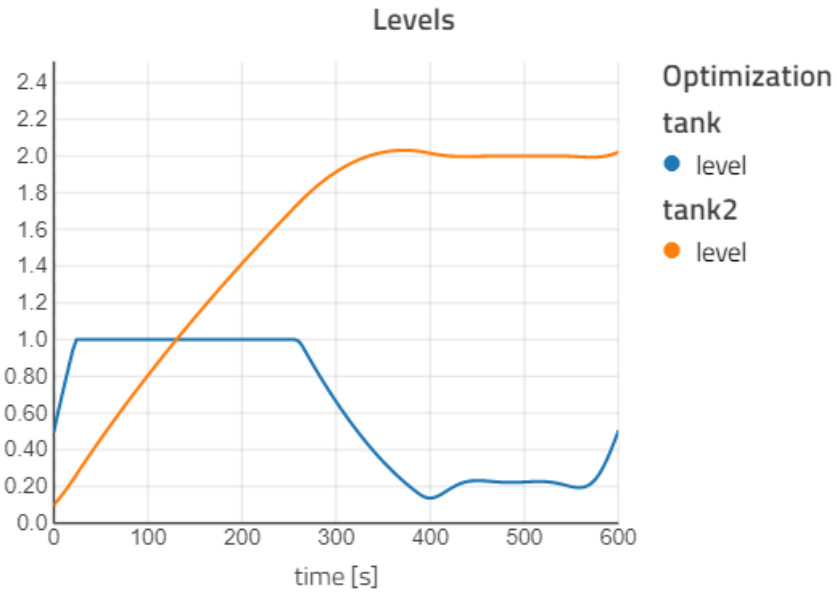


Optimization objective

Use the pump in order to

- Take the level of the lower tank to 2m as quick as possible
- Without getting any spill over in the upper tank of height 1m
- Flow limited to 30 kg/s
- After 10min, the upper tank should be back at initial level





Under the hood

- Problem formulation
- Numerical approach
- Initialization
- Packaging & deployment

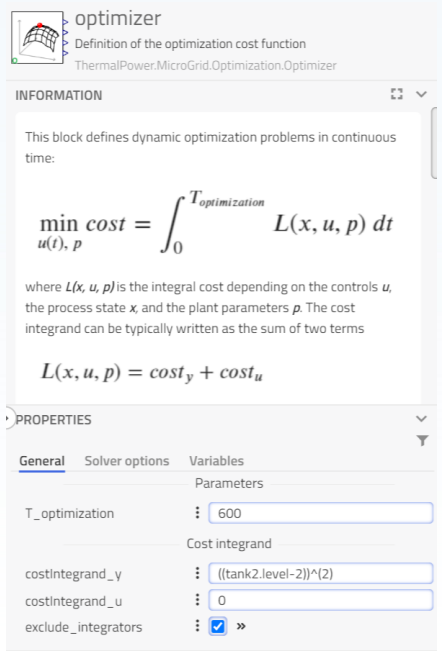
Formulation of the optimization problem

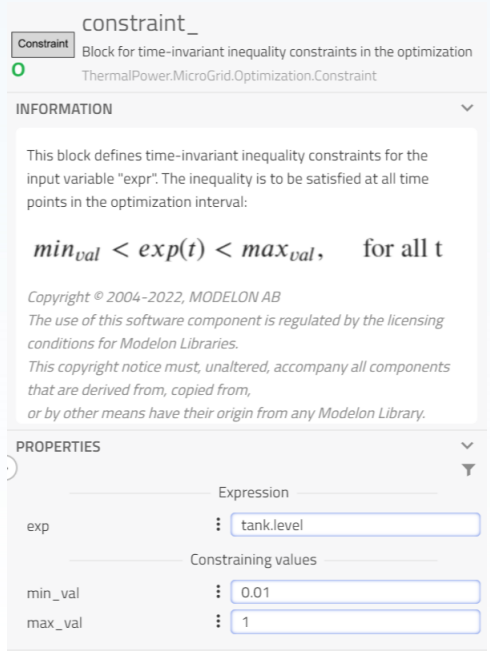
Convenient formulation using optimization specific blocks & Modelica code (min/max attributes)

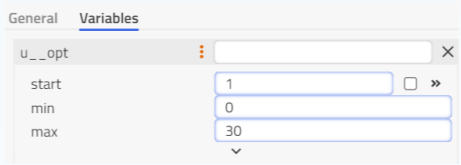
$$\min_{u(t)} \int_0^{600} (\text{tank2.level(t)} - 2)^2 dt$$

$$\text{tank.level(t)} < 1$$

$$u(t) < 30$$

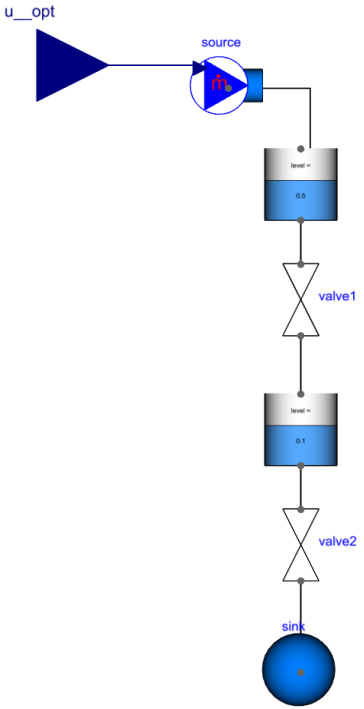






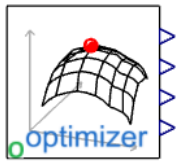
Formulation in Optimica

Plant model



TwoTanks.mo

Optimization formulation



$$\min_{u(t)} \int_0^{1000} (\text{tank2.level(t)} - 2)^2 dt$$

$$u(t) < 30$$

$$tank.level(t) < 1.5$$

$$tank.level(0) = tank.level(1000)$$

Optimica language

```
package OptimicaFormulation
    optimization OptimalTankControl(
                objectiveIntegrand = (tank2.level-2)^2,
                startTime = 0,
                finalTime=1000)
        extends TwoTanks(u__opt(max=30));
        constraint
        tank.level(0)=tank.level(1000);
        tank.level < 1.5;
    end OptimalTankControl;
end OptimicaFormulation;
```

OptimicaFormulation.mop

Modelon

Numerical approach at a glance

Continuous problem → Compiler → CasADi → Collocation → Discrete problem → Nonlinear solver → Solution

```
package OptimicaFormulation
    optimization OptimalTankControl(
                 objectiveIntegrand = (tank2.level-2)^2,
                 startTime = 0,
                 finalTime=1000)
    extends TwoTanks(u__opt(max=30));
```

https://web.casadi.org/

https://coin-or.github.io/Ipopt/

From Optimica to CasADi

Algorithmic differentiation

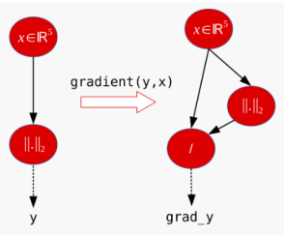
Trajectories as piecewise polynomial

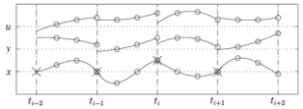
Large and sparse discrete optimization problem

Interior point method

Solution in continuous time

$$\min_{\text{subject to}} f(x,u,p)$$
$$\dot{x} = g(x,u,p)$$
$$x_L \leq x \leq x_U$$
$$u_L \leq u \leq u_U$$
$$h(x,u,p) \leq 0$$



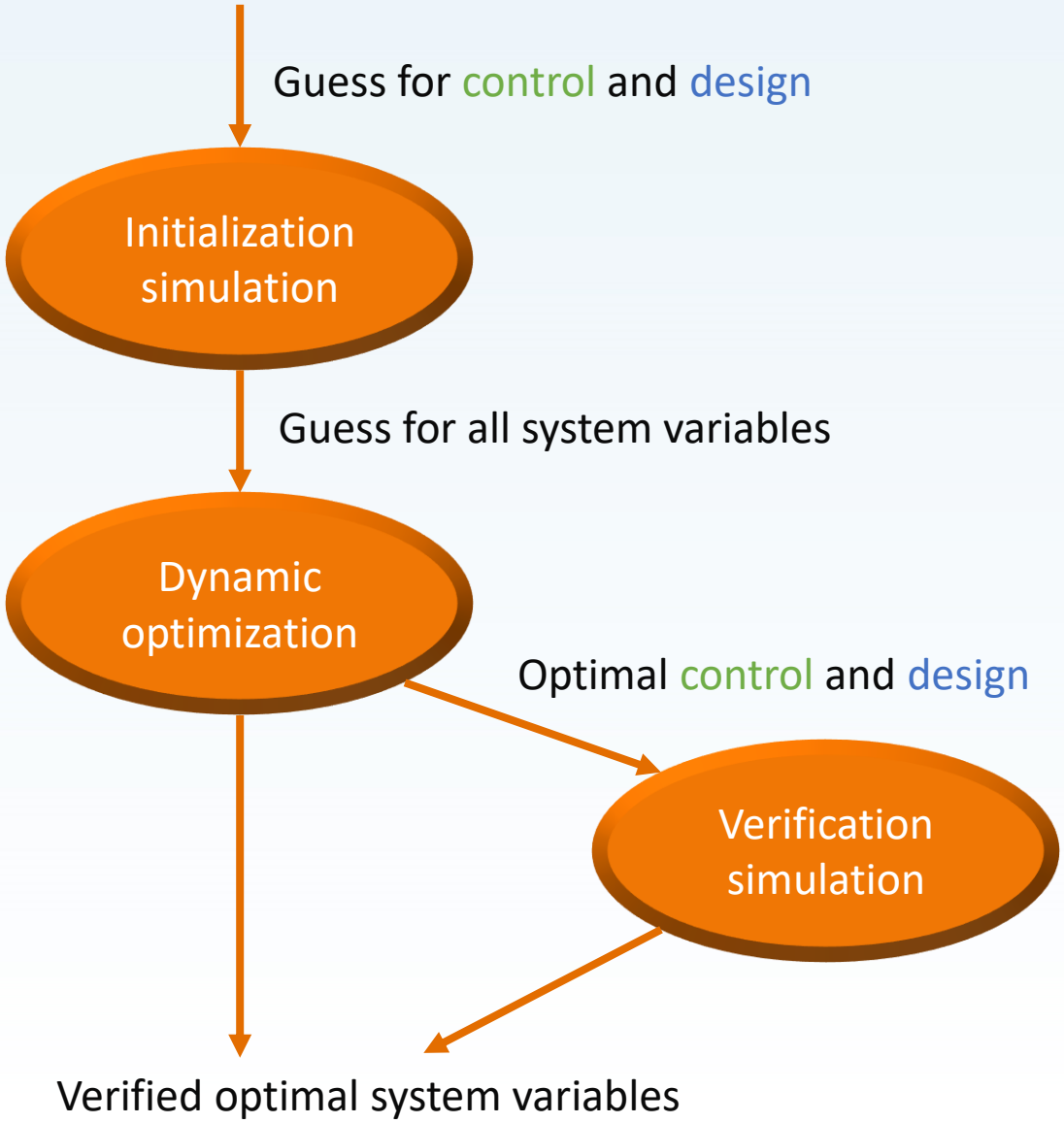


$$\min_{\text{subject to}} f(x)$$
$$0 = g(x)$$
$$x_L \leq x \leq x_U$$
$$h(x) \leq 0$$

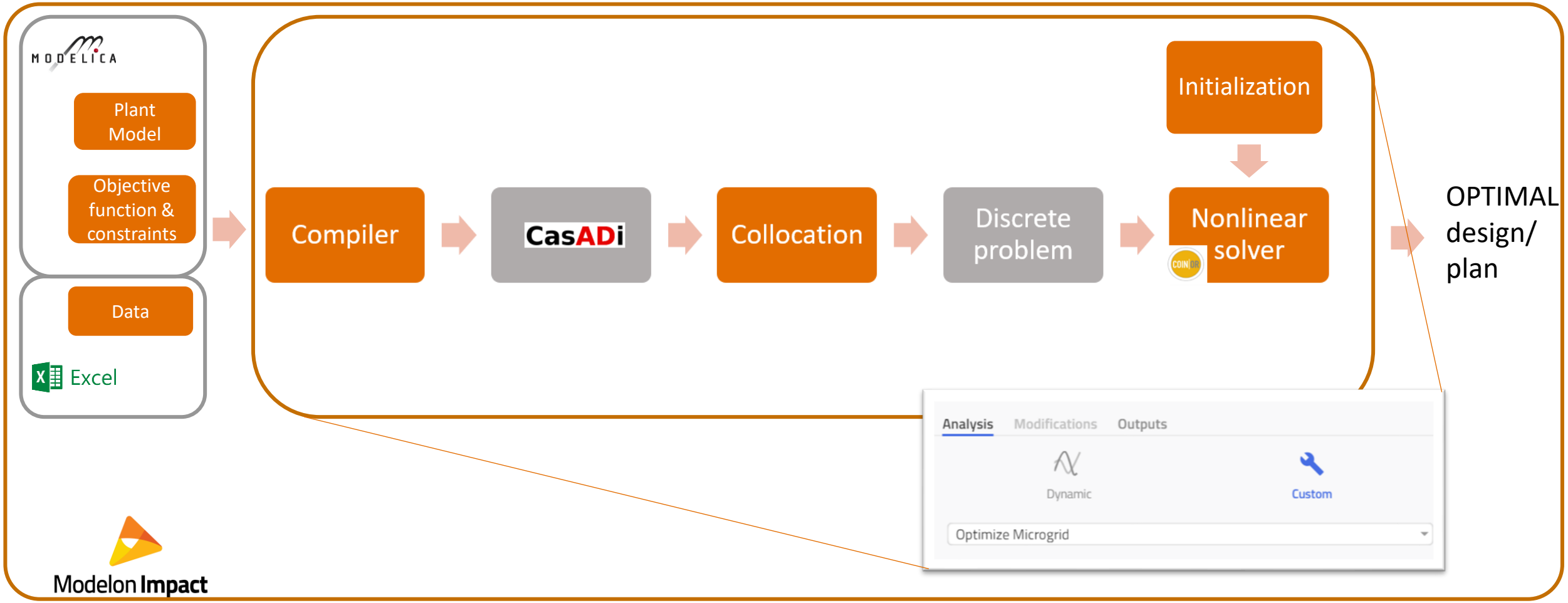
Initialization of the optimization

The nonlinear program requires a reasonable guess for all model variables over the entire time horizon.

1. Construct a reasonable initial guess of the degrees of freedom
   - Parameters: Your best estimate
   - Inputs: Constants or simple controllers
2. Simulate system with this guess to generate a complete initial guess of the system
3. Solve dynamic optimization problem with initialization simulation result as initial guess
4. Simulate system with optimal degrees of freedom, to verify the result



Overall workflow package as a custom function



Python implementation

Import methods for compilation and fmu loading

```
from pymodelica import compile_fmu
from pyfmi import load_fmu
from pyjmi import transfer_optimization_problem
```

Compilation & loading of the model for initialization

```
# Compile the optimization initialization model
init_sim_fmu = compile_fmu("CSTR.CSTR_Init_Optimization", "CSTR.mop")

# Load the model
init_sim_model = load_fmu(init_sim_fmu)
```

Initial simulation

```
# Simulate with constant input Tc
init_res = init_sim_model.simulate(start_time=0., final_time=150.)
```

Compilation of the optimica code

```
# Compile and load optimization problem
op = transfer_optimization_problem("CSTR.CSTR_Opt2", "CSTR.mop")
```

Optimization options

```
# Set options
opt_opts = op.optimize_options()
opt_opts['n_e'] = 19 # Number of elements
opt_opts['init_traj'] = init_res
opt_opts['nominal_traj'] = init_res
opt_opts['IPOPT_options']['tol'] = 1e-10
```

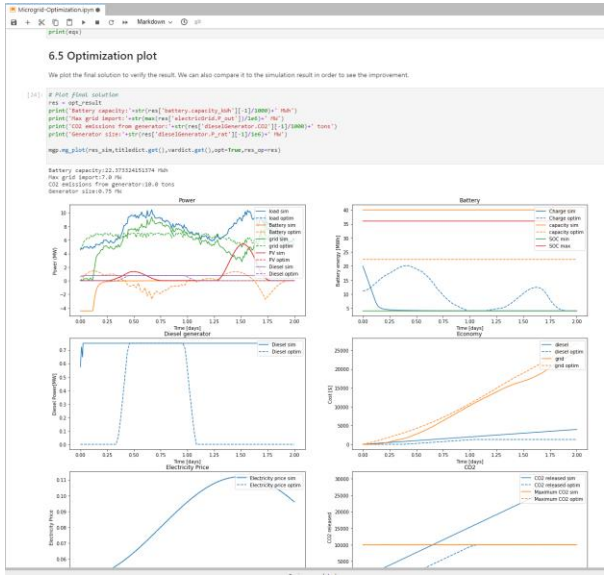
Optimization

```
# Solve the optimal control problem
res = op.optimize(options=opt_opts)
```

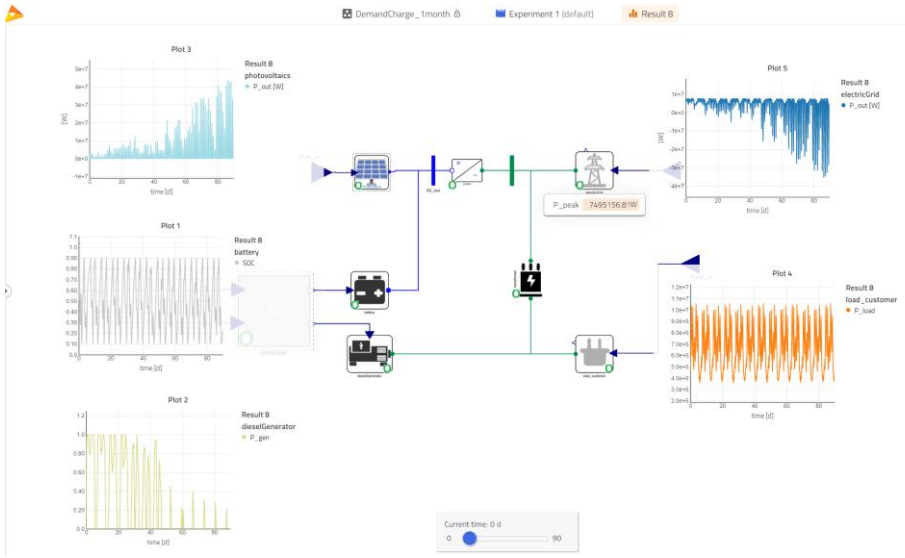

Modelon

OPTIMIZATION WORKFLOW DEPLOYMENT

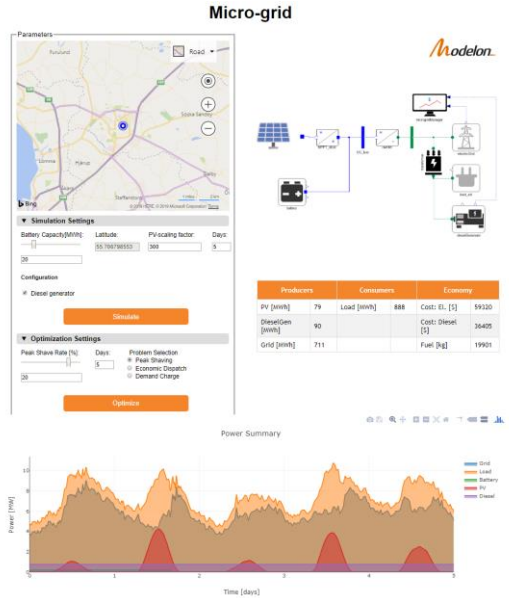
Notebook for workflow design



Model-centric view to test the workflow on new models



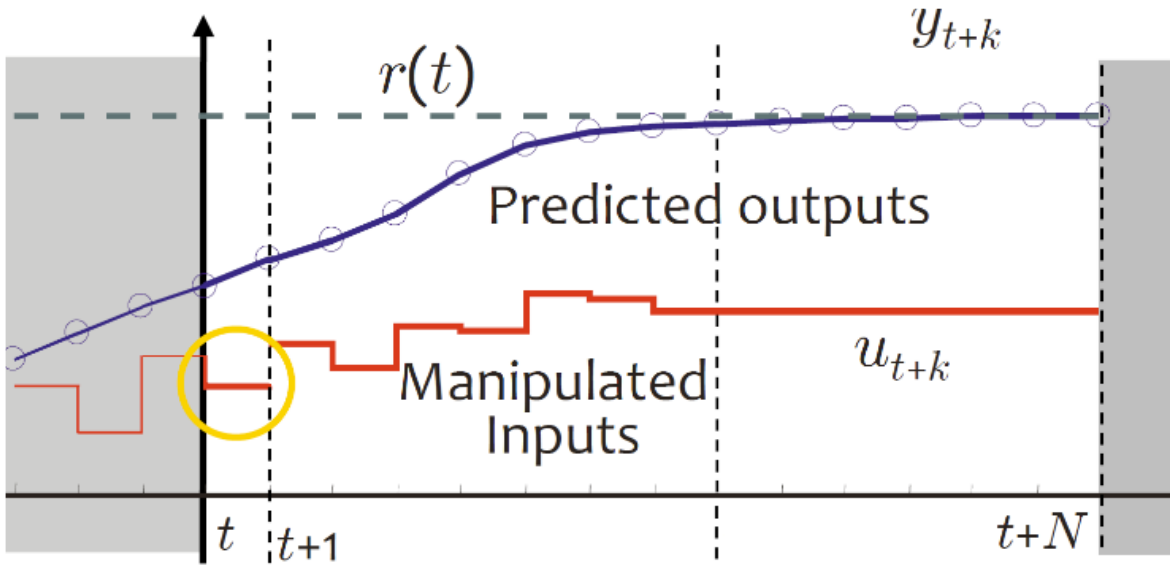
Web-app for deployment



Modelon

MODEL PREDICTIVE CONTROL

- The dynamic optimization framework computes optimal open-loop trajectories
- Feedback is necessary to handle
  - Model-plant mismatch
  - Disturbances



- *Model predictive control* is a framework for introducing feedback:
  - In each sample point, compute optimal control for the coming N samples
  - Until next sample, apply first sample point of optimal control
  - Update state estimate based on measurements, repeat

Optimization-Friendly Modelling

- Basic requirements
- Fundamental limitations
- Building new models

Modelon

Smoothness requirements

- Equation expressions must be twice continuously differentiable, C^2, with respect to all optimization variables $(\dot{x}, x, y, u, p)$
- Consequences:
  - No discrete variables and equations (except parameters)
  - No event-generating expressions
  - Smooth equations
- If requirements not satisfied, the following may happen:
  - Compilation failure
  - Solver failure
    - Often very difficult to diagnose solver failure caused by non-smoothness!
  - Bad solver performance
  - Inaccurate solution
  - If you are lucky, nothing at all!

Modelon

Smoothness checker

- Compiler options to check that smoothness requirements are satisfied:
  - `allow_discrete_variables`: Set to False to detect discrete, non-parametric variables
  - `allow_when_clauses`: Set to False to detect when clauses
  - `allow_discrete_switches`: Set to False to detect event-generating expressions
  - `system_continuity_order`: Set to 2 to detect expressions that the compiler cannot verify are twice continuously differentiable

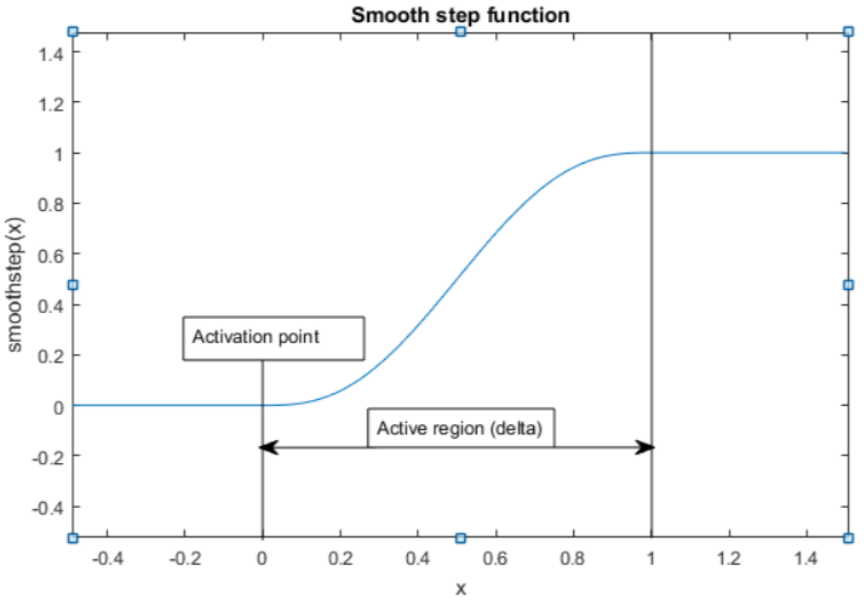
- Recommended to enable these when developing optimization-friendly models

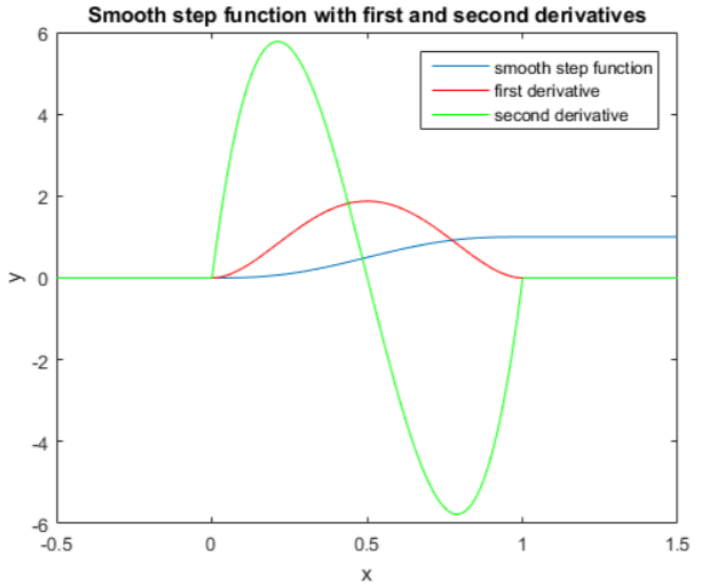
Modelon

SMOOTHING TECHNIQUES

- Typical non-smooth functions that need smooth approximations:
  - Max
  - Min
  - Saturation
  - Step
  - …
- Activation point and region to be specified (see Figure to the right)
  - Inputs should be shifted and scaled: $y_{\text{smooth}} = \text{stepC2}(\frac{x - x_{shift}}{\Delta})$
- Analytic differentiability is not sufficient. Needs to be numerically smooth!
- Important to preserve monotonicity

```
1  function stepC2
2    "Smooth (twice continuous differentiable) step function based on a 5 degree polynomial"
3    input Real x;
4    output Real y;
5
6  algorithm
7    y := if x < 0 then 0 else (if x > 1 then 1 else x^3*(10+x*(-15+x*6)));
8    annotation(•••);
13 end stepC2;
```





LIMITATIONS

- CasADi Interface supports a large (but incomplete) set of the Modelica language

- Two categories of limitations:
  - Fundamental limitations: Some constructs are inherently discrete and should never be used for gradient-based optimization (pre(), edge(), reinit()…)
  - Compiler limitations: Constructs that may be useful for optimization but not yet supported by OCT ex: String, external functions

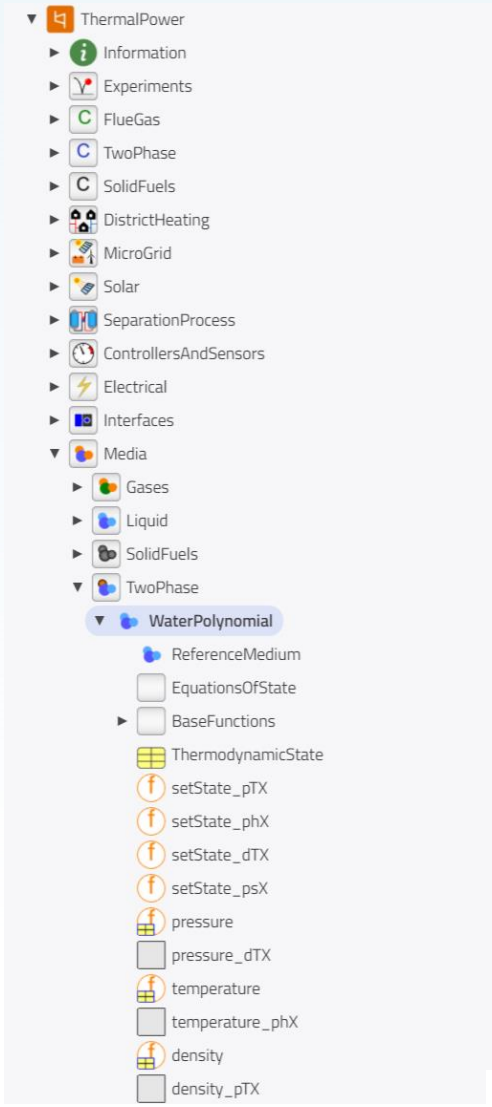
Modelon

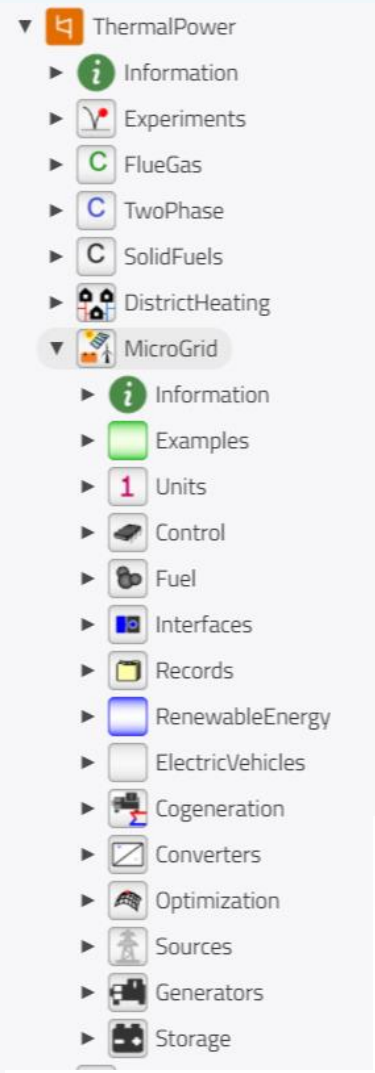
Optimization friendly packages

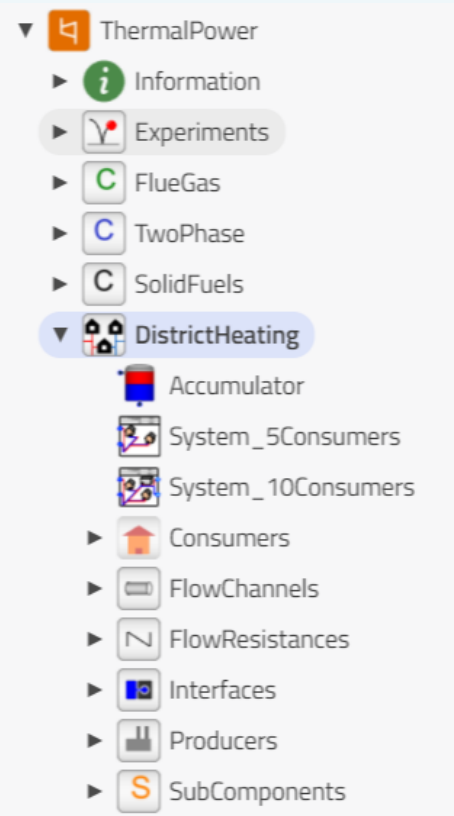
Three packages inside Thermal Power have been derived for dynamic optimization

Additional content to describe boilers can be provided on demand

Good starting point for deriving new models







Recommended workflow for system modelling

- When building models, **start small** and add complexity as you go along

- **Test each step**! Starting with a large, complex, broken model will be very difficult to debug

- Testing steps:
  - FMU compilation, with smoothness check
  - Dynamic simulation, result verification
  - Transfer to CasADi Interface
  - Solve optimization problem

Modelon

Applications

- Energy
- Aerospace
- Automotive

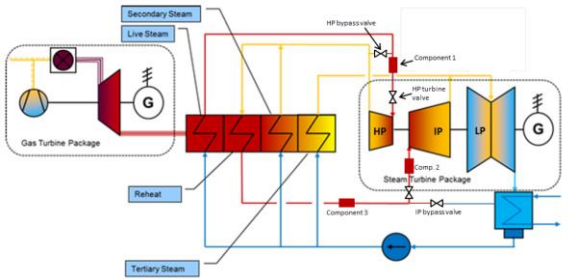
DEMO

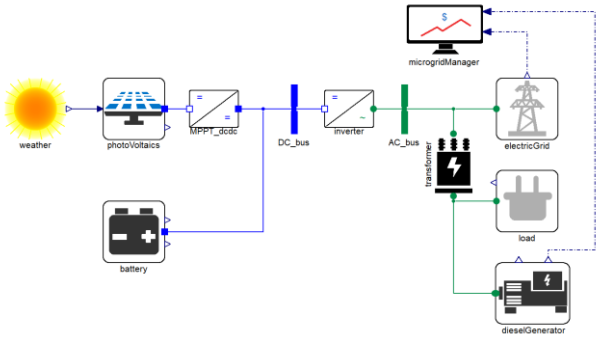
Energy

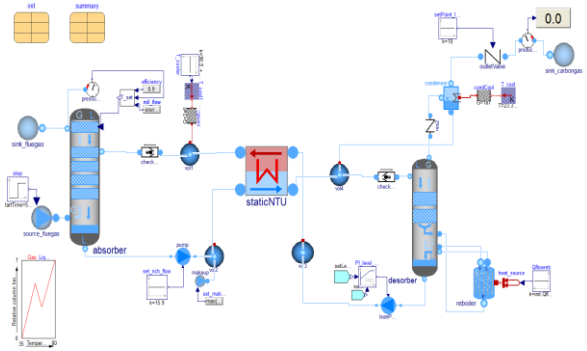
- Microgrid design & operation
- Model Predictive Control for boiler start-up

Modelon

Examples of applications







District Heating

- Production planning
- Network aggregation
- Transport delays

Power plant

- Start-up optimization
- Thermal & mechanical stress
- Offline & online optimization
- Nonlinear predictive control
- Experimental tests by Siemens
- OPC communication

Micro-grid

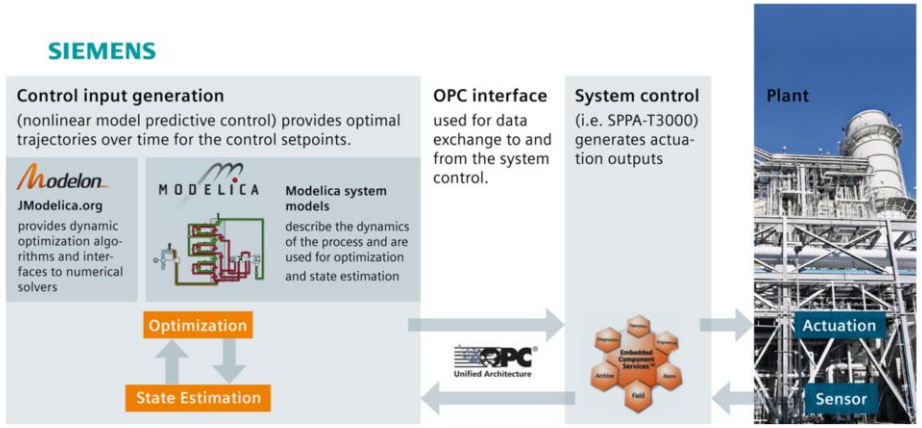
- Optimal design & operation
- Forecast for weather, electricity price, load
- Peak shaving
- Economic dispatch

CO2 capture

- Optimal operation
- DOF: reboiler duty and circulation rate
- Target removal efficiency
- Reboiler pressure constraint

Modelon

SELECTED REFERENCES

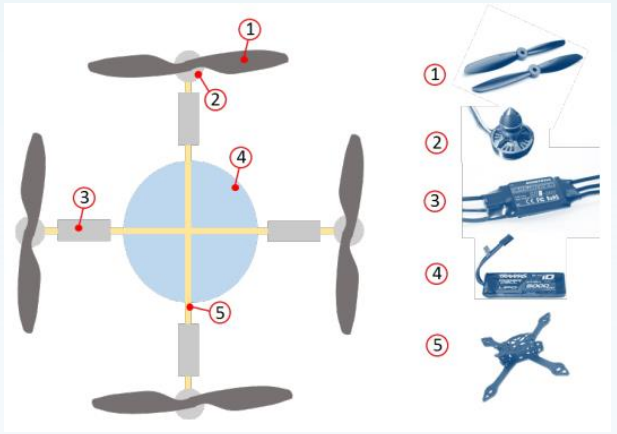


- Dietl et al**.** (2018) **Start up optimization of Combined Cycle Power Plants: Controller development and real plant test results,** *CoDIT*
- Schweiger et al., (2017) **District heating and cooling systems – Framework for Modelica-based simulation and dynamic optimization**, In Energy
- MacRae N et al. (2020) **Micro-grid Design and Cost Optimization using Modelica,** American Control Conference
- Dietl et al., (2014) **Industrial application of optimization with Modelica and Optimica using intelligent Python scripting**. Modelica conference
- Holmqvist A. et al (2016) **Open-loop optimal control of batch chromatographic separation processes using direct collocation** In Journal of Process Control 46. p.55-74
- M. Wetter and C. van Treeck (2017). **New Generation Computational Tools for Building & Community Energy Systems**. Annex 60 Final Report. IEA
- Fouquet M. et al, (2014), **Hybrid dynamic optimization of power plants using Sum-Up Rounding and adaptive mesh refinement**, IEEE Conf on control Applications
- R. De Coninck, L. Helsen (2016). **Practical implementation and evaluation of model predictive control for an office building in Brussels**. Energy and Buildings.
- R. De Coninck et al, (2015). **Toolbox for development and validation of grey-box building models for forecasting and control**. Journal of Building Performance Simulation.
- PhD theses from the Automatic Control department in Lund
    - Fredrik Magnusson (2016), Numerical and Symbolic Methods for Dynamic Optimization
    - Per-Ola Larsson, (2011), Optimization of Low-Level Controllers and High-Level Polymer Grade Changes
    - Staffan Haugwitz (2007), Modeling, Control and Optimization of a Plate Reactor
    - Johan Åkesson (2007), Languages and Tools for Optimization of Large-Scale Systems

Aerospace

Drone sizing and control

Building

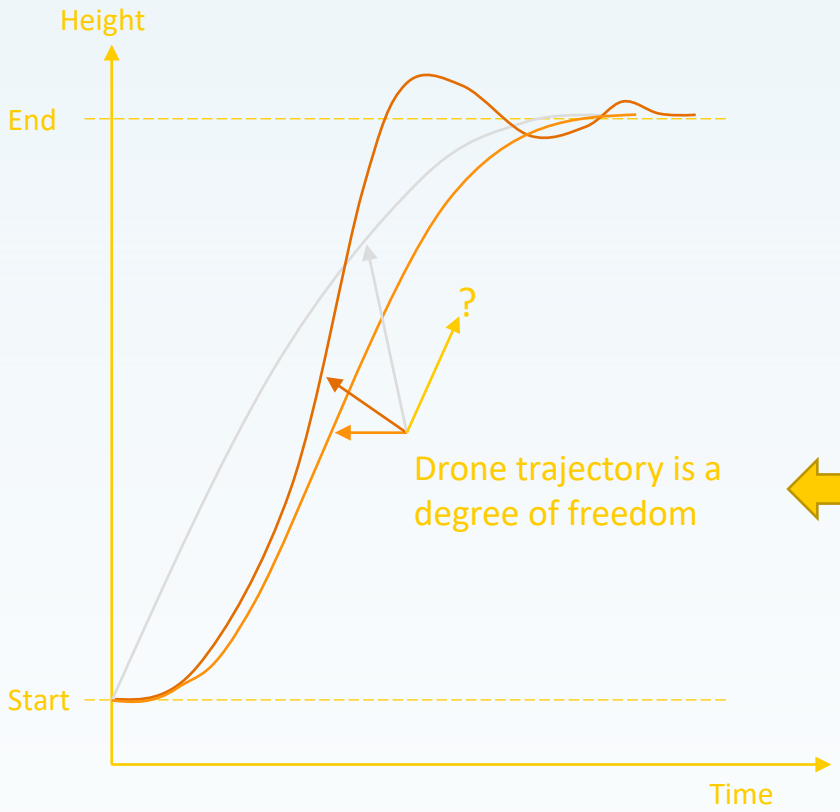


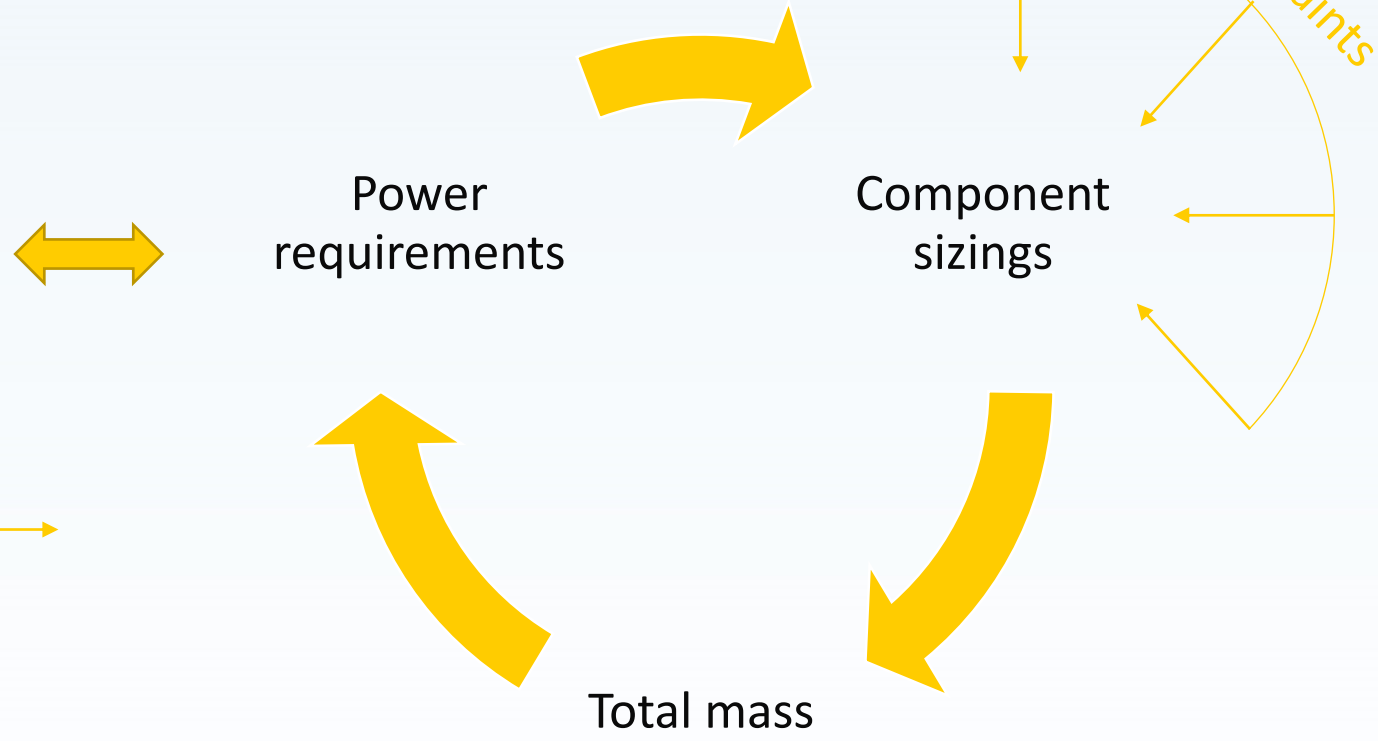
The drone architecture is composed of:

1. Four fixed pitch propellers
2. Four out-runner brushless motors
3. Four electronic speed controllers (ESC) mainly made from MOSFET inverters
4. One battery based on Li-Ion cells
5. One mechanical structure (frame) consisting of four arms and one central body

Modelon

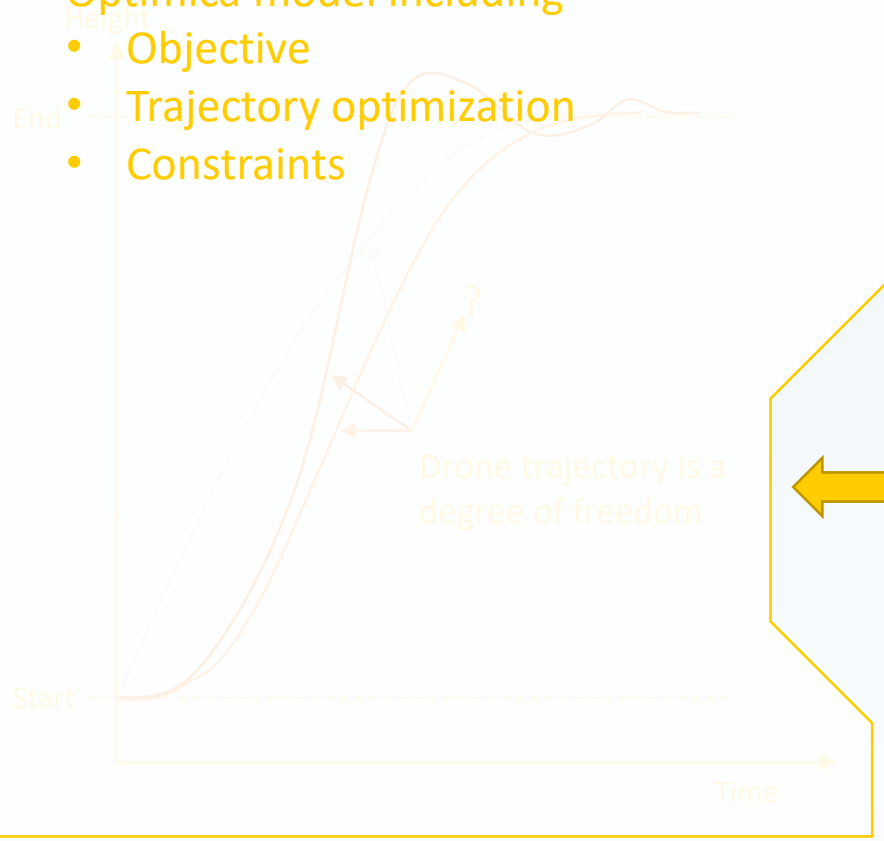
We want to maximize the number of flights, hence our objective is to minimize the energy consumption per flight.

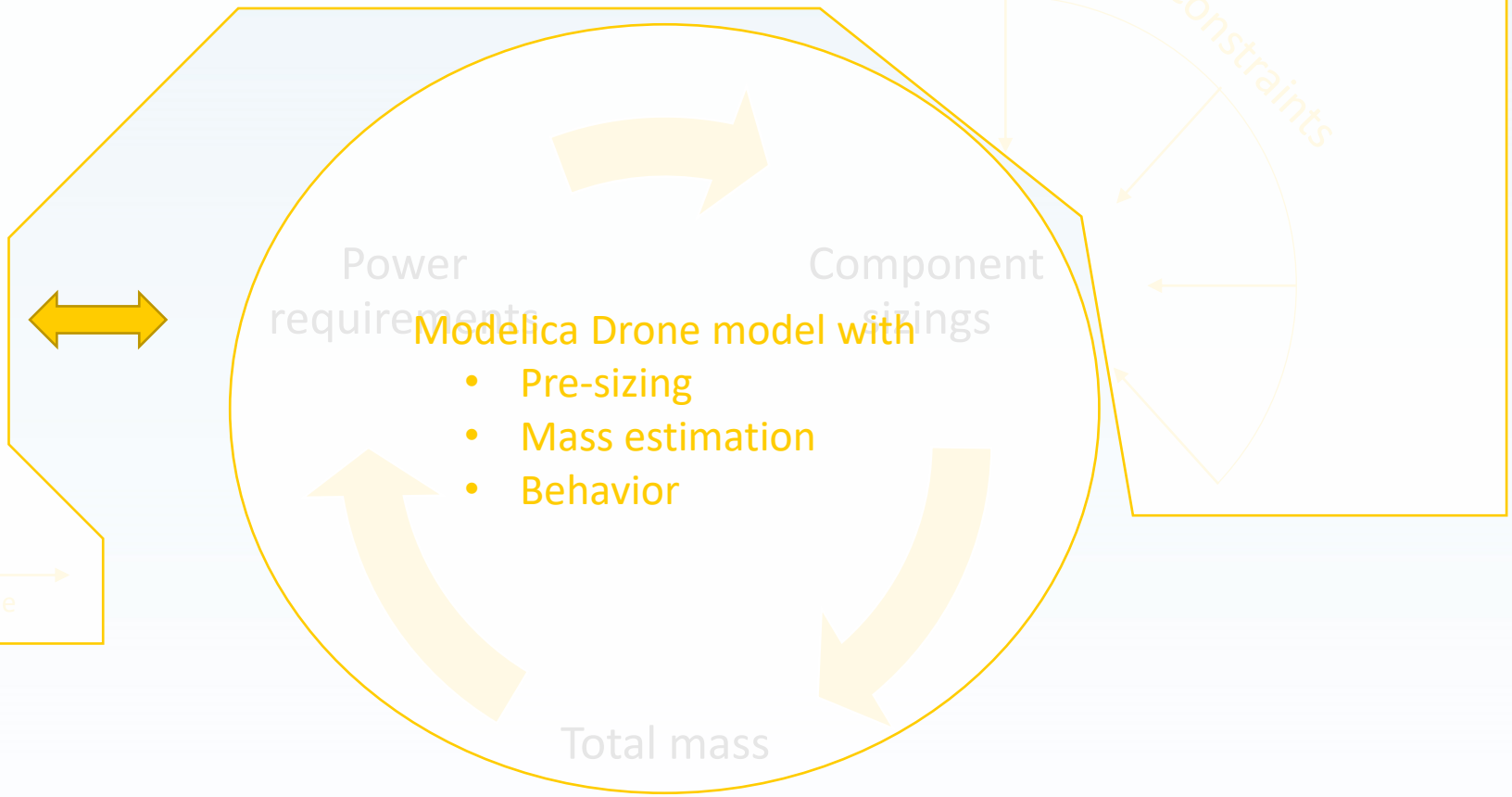




We want to maximize the number of flights, hence our objective is to minimize the energy consumption per flight.

Optimica model including
- Objective
- Trajectory optimization
- Constraints



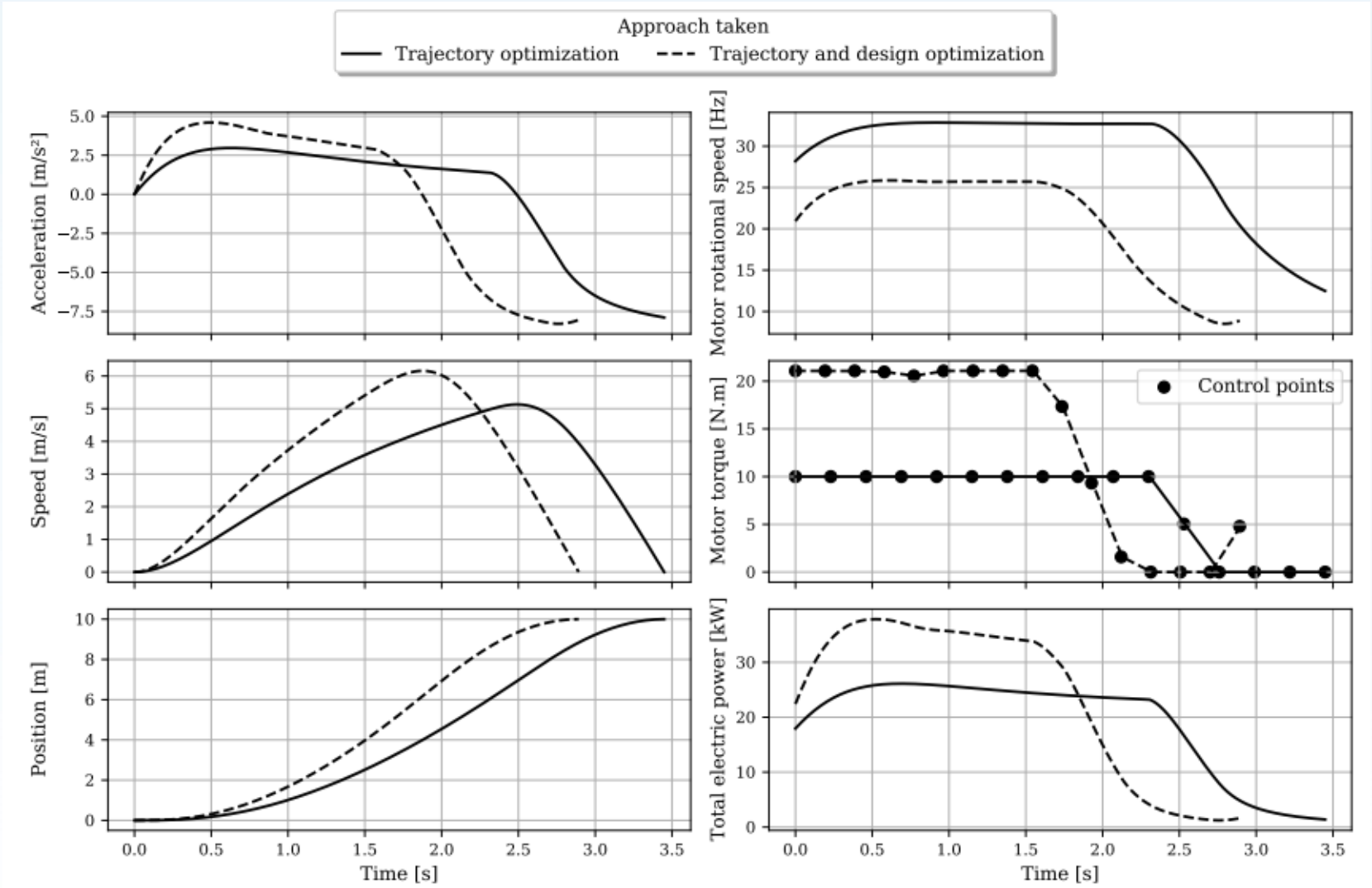


Modelica Drone model with
- Pre-sizing
- Mass estimation
- Behavior

Modelon

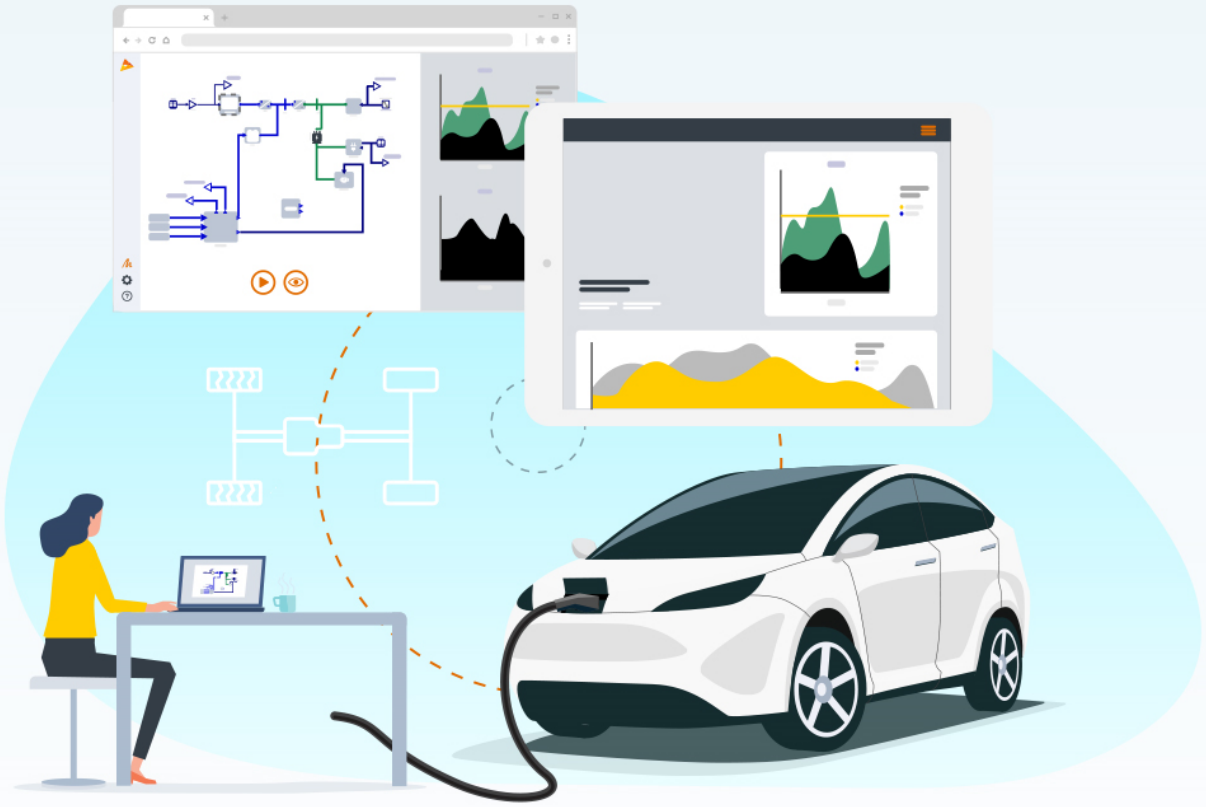
```
optimization SizingAndTrajectoryOptim (
   objective=M_total(startTime),
   finalTime(free=true, min=1, max=10, start=5))
// Minimize the total drone mass and relax the final simulation time within bounds.

   import Modelica.Units.SI.DimensionlessRatio;

   extends Drone(
      x(start = 0, fixed=true),
      xp(start = 0, fixed=true),
      a(start = 0, fixed=true),
      beta(free=true, min=0.3, max=0.6, start=0.4),
      D(free=true, min=0, max=1),
      T_nom_mot(free=true, min=0),
      K_mot(free=true, min=0),
      M_bat(free=true, min=0, max=100),
      P_esc(free=true, min=0),
      k_D(free=true, min=0.01, max=1, start=0.05),
      D_out_arm(free=true, min=0.001, max=1));
// Inherit the Modelica drone model, fix initial conditions and relax design parameters within
bounds.

   Modelica.Blocks.Interfaces.RealInput Traj_in;
// Add input to the trajectory to optimize

...
```

```
...

   DimensionlessRatio n_norm(start=1, fixed=true)=n/n_hover;

   DimensionlessRatio N_norm(min=-1, max=1, nominal=0.8)=ND/ND_max;

   DimensionlessRatio T_hov_norm(min=0, max=1, nominal=0.6) = T_hover/T_nom_mot;

   DimensionlessRatio T_norm(min=-1, max=1, nominal=0.95) = T/T_max_mot;

   DimensionlessRatio U_norm(min=0, max=1, nominal=0.5) = U_mot/V_bat;

   DimensionlessRatio P_norm(min=0, max=1, nominal=0.5) = P_mot/P_esc;

   DimensionlessRatio E_norm(min=0, max=1, nominal=0.25) = E_drone/E_bat;

   DimensionlessRatio sigma_norm(min=-1, max=1, nominal=0.15) = sigma/sigma_max;
// Create additional normalized variables with bounds as inequality constraints

equation

   T=Traj_in; // Bind drone trajectory with optimization input

constraint

   x(finalTime) = 10;

   xp(finalTime) = 0;

   a(finalTime) = 0;
// Define end time constraints.

end SizingAndTrahjectoryOptim;
```


Approach taken
Trajectory optimization
Trajectory and design optimization
Acceleration [m/s²]
Speed [m/s]
Position [m]
Motor rotational speed [Hz]
Motor torque [N.m]
Control points
Total electric power [kW]
Time [s]
Time [s]

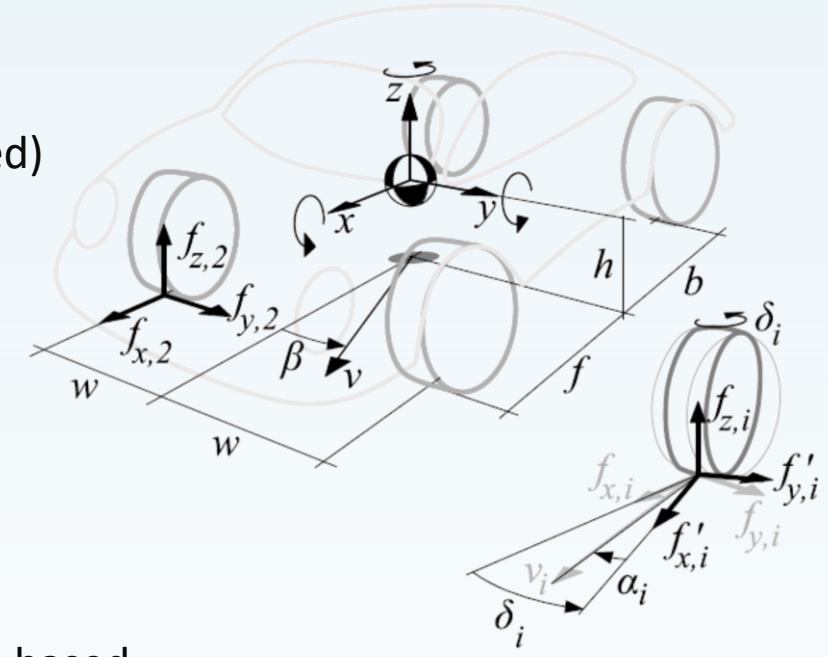
Automotive

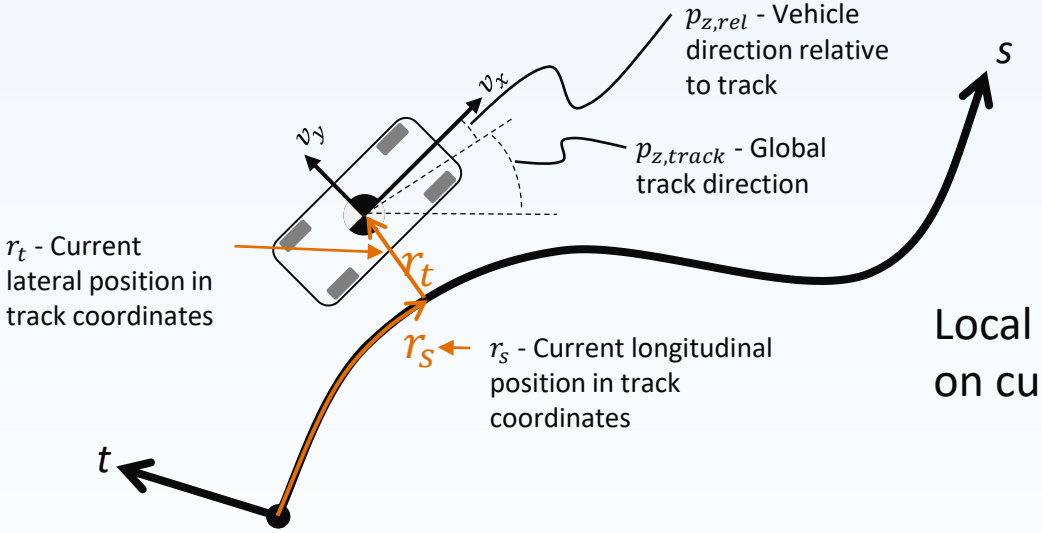
Trajectory optimization of a racing car



Modelon

Two-track vehicle model with combined slip and relaxation dynamics (not multibody based)



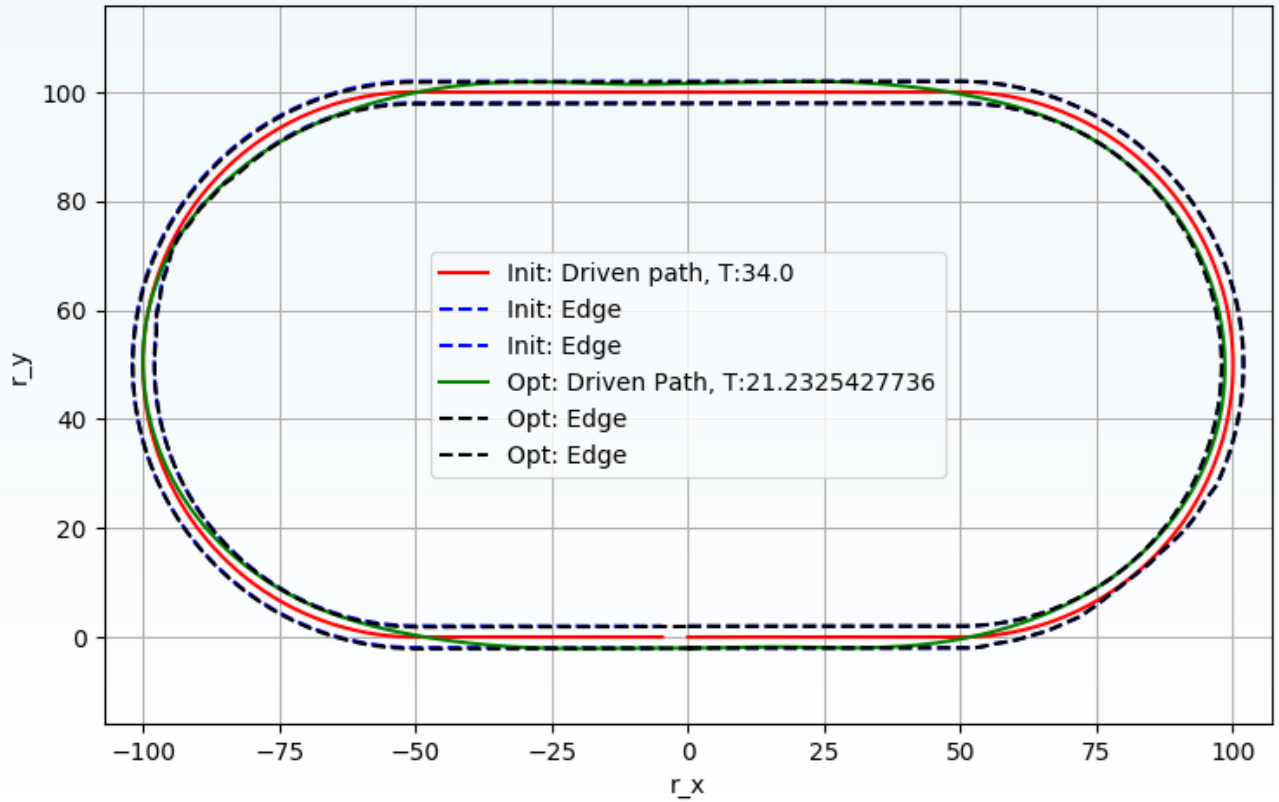


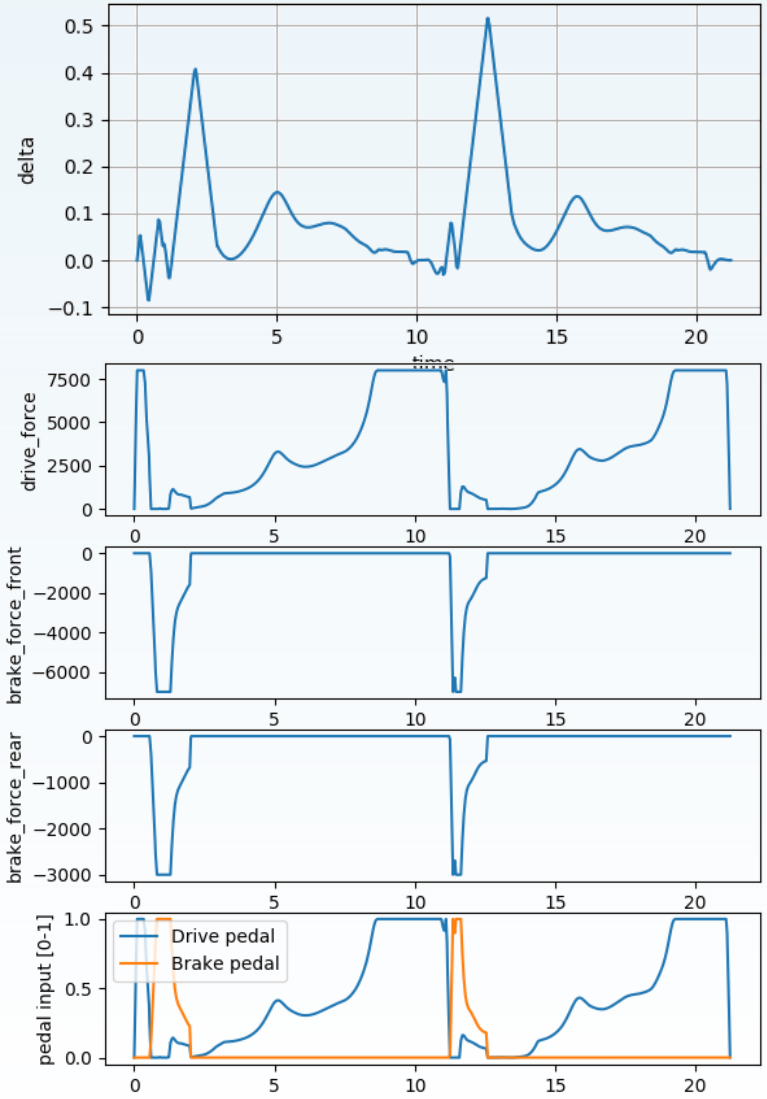
Local track coordinate system based on curvature and distance along track

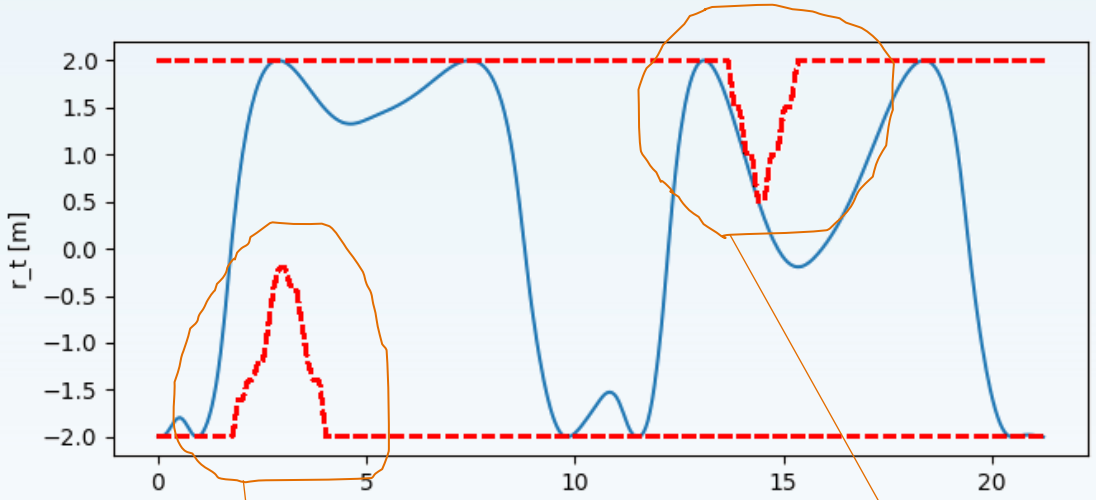
Simple optimization formulation, cost on lap time + small cost on actuator usage

```
optimization LapTimeMinimization(
   objective=100*finalTime+1*icost(finalTime),
   startTime = 0,
   finalTime(free=true,max=160,min=10,initialGuess=122))
```

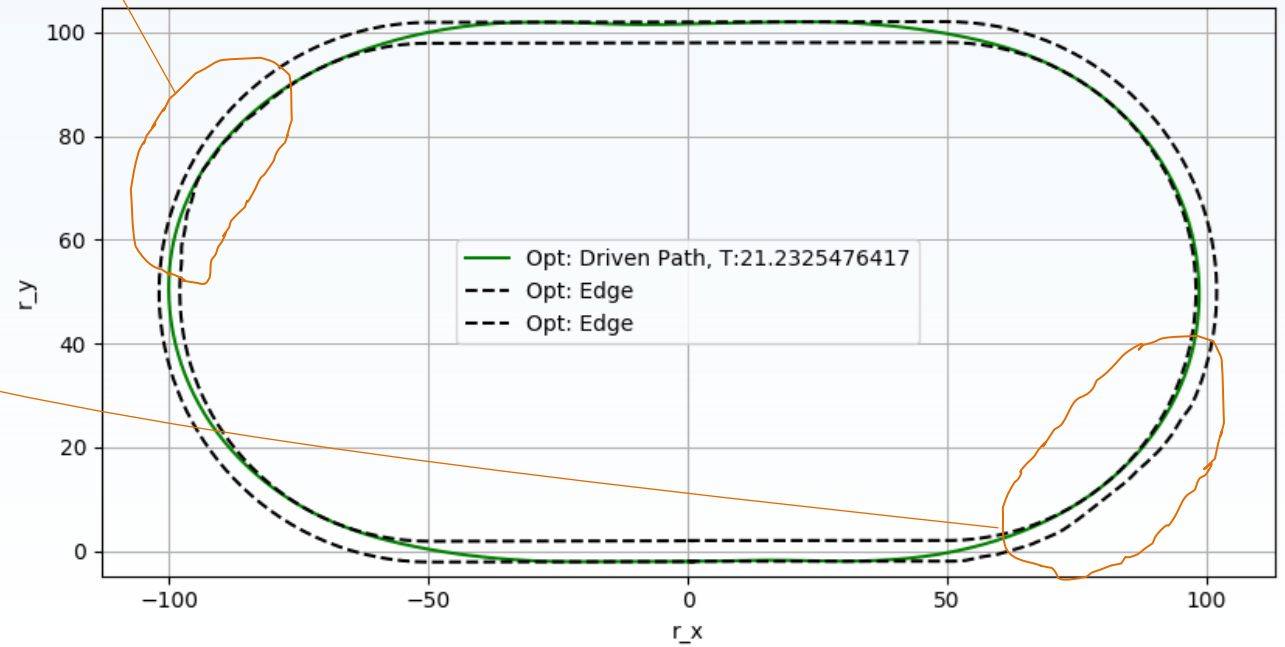
Oval track



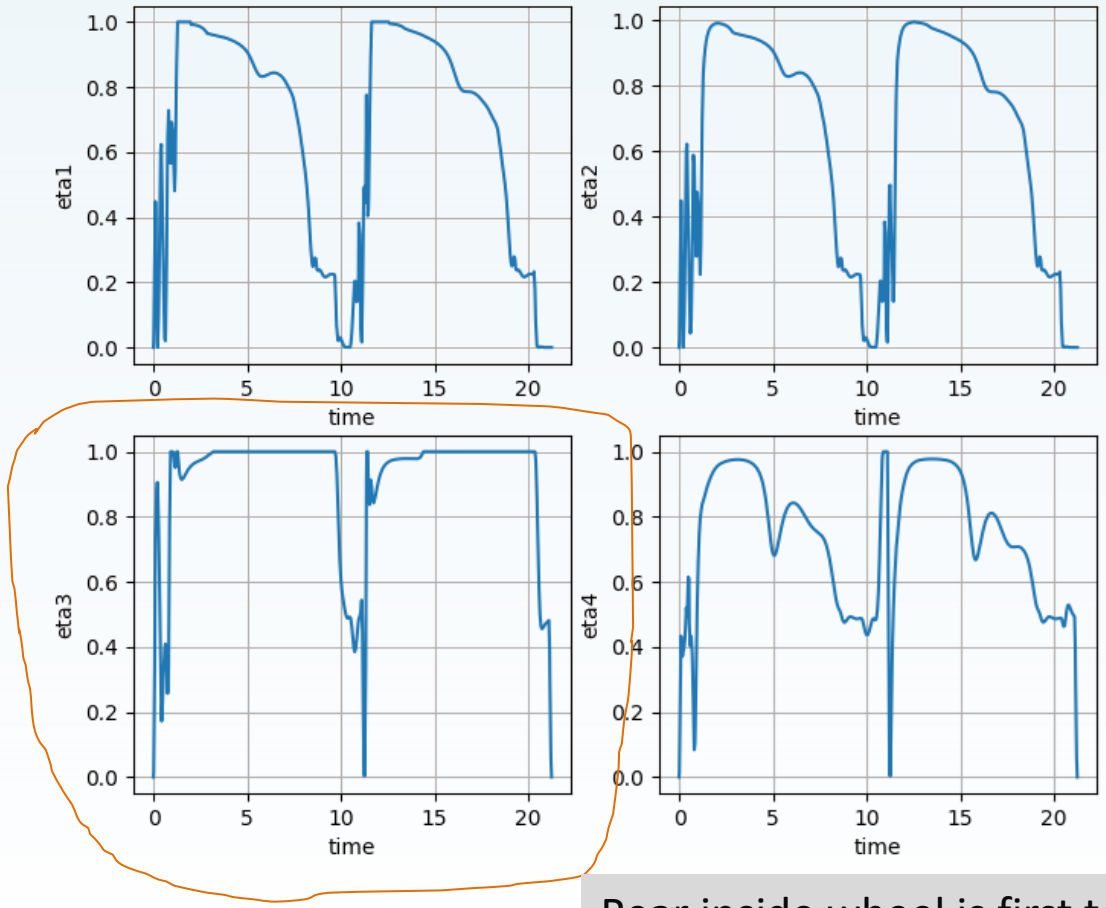




Example of variable track width

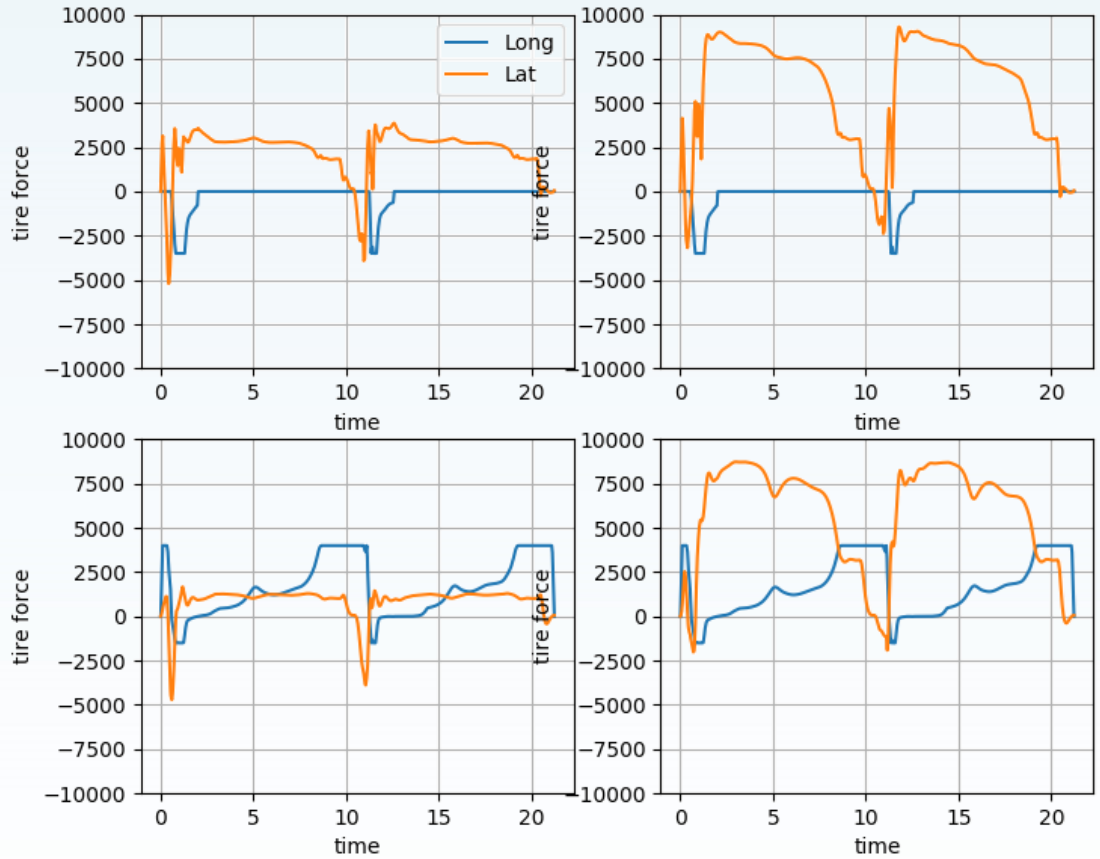


Friction usage

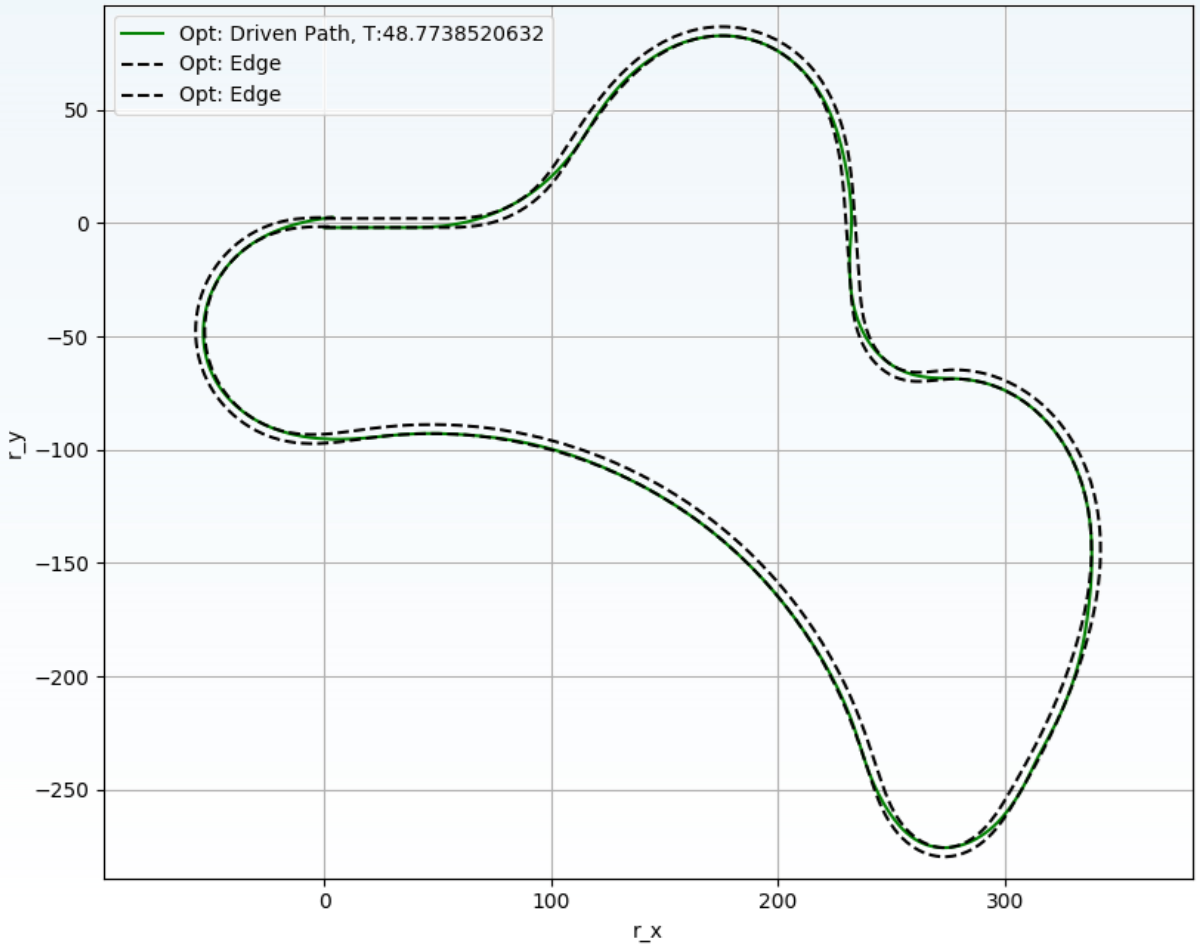


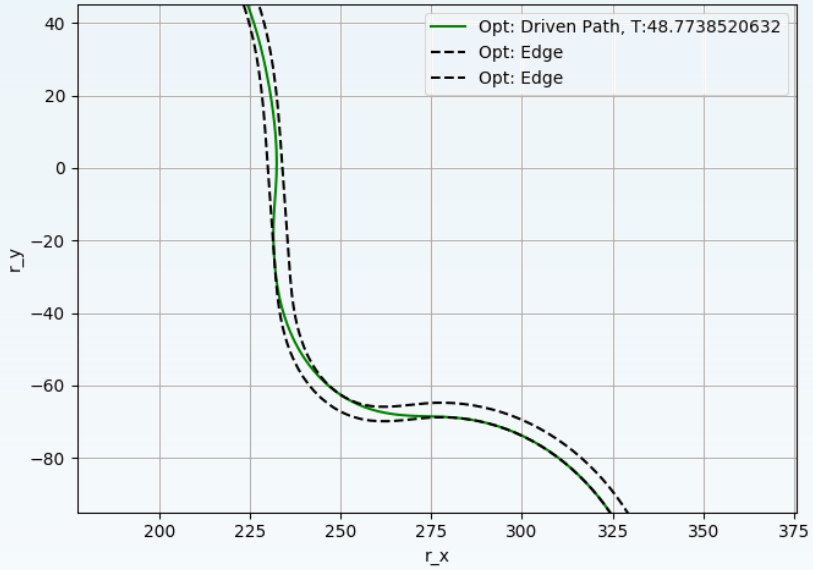
Rear inside wheel is first to be limited by grip

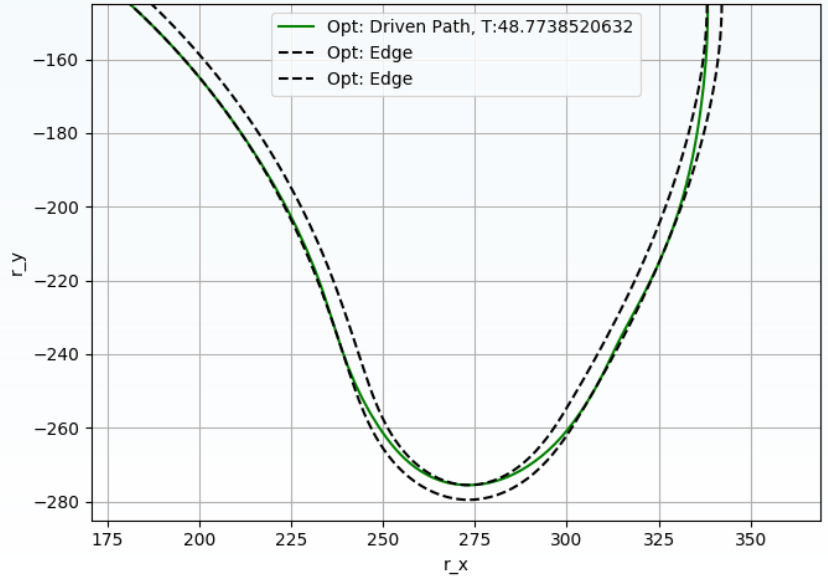
Tire forces



Road course







Conclusion

- Modelon Impact offers powerful dynamic optimization methods
- They set some requirements on the models
  - Smooth
  - No discrete dynamics
- Recent integration of the workflow into the main GUI
- Ideally for optimal control & scheduling problems
- Relatively easy to apply in feasibility studies

Modelon