Reading Instructions

Please study sections I and II of the paper on time-optimal race car driving using nonlinear model predictive control [1].

Technical Prerequisites:

For solving this assignment some additional packages are necessary that are pre-installed on Retina VMs. If you are not working on a Retina machine, install the ROS Gazebo control packages before starting with this lab by executing the following command in a terminal:

```
sudo apt-get install ros-melodic-ros-control ros-melodic-ros-controllers ros-melodic-
    gazebo-ros-control ros-melodic-ackermann-msgs ros-melodic-joy ros-melodic-gazebo-
    ros-pkgs
```

Download the ROS meta-packages `simulator`, `system` and `topics_setup` from the Moodle workspace and paste them into the `src` folder of your `catkin_ws`. The meta-packages contain a lightweight 2D Racecar simulator and some additional messages and packages for running this assignment. Change the permission rights of the python scripts and build the package using

```
chmod +x /path/to/catkin_ws/src/simulator/racecar-simulator/racecar_gazebo/scripts/
  gazebo_odometry.py
chmod +x /path/to/catkin_ws/src/topics_setup/scripts/converter_control_Gazebo.py
chmod +x /path/to/catkin_ws/src/topics_setup/scripts/Collision_detection.py
cd /path/to/catkin_ws
catkin_make
source devel/setup.bash
```

Make sure `catkin_make` finishes without any errors.

Bicycle Kinematic Model

Assume a vehicle with front wheel steering. In the kinematic bicycle model the front wheel represents the left and right front wheel of the actual car, the rear wheel its left and right rear wheel. The bicycle model is depicted in figure 1. The point x, y located on the line connecting the centers of the rear axle x_r, y_r and front axle x_f, y_f denotes the vehicle's center of gravity (COG). Notice that the direction of the velocity vector v at the center of gravity is different from the velocity vectors at the rear wheel v_r (direction ψ) and at the front wheel v_f (direction $\psi + \delta_f$). The angle β, termed (side) slip angle, is the difference between the velocity vector at the COG and the heading of the bicycle ψ.

[1] Source: RST

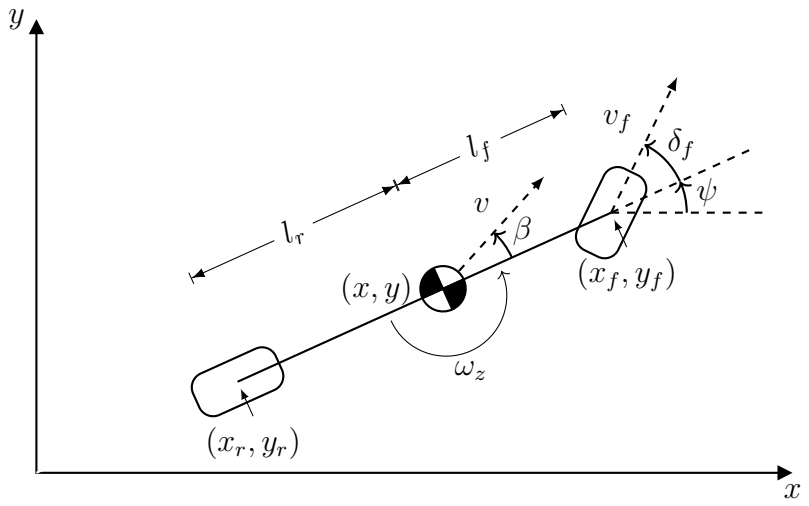

Figure 1: bicycle kinematic model [1]

The bicycle kinematic model is described by

$$\dot{x} = v\cos(\psi+\beta) \tag{1}$$
$$\dot{y} = v\sin(\psi+\beta) \tag{2}$$
$$\dot{\psi} = \frac{v\cos(\beta)\tan(\delta_f)}{l_r+l_f} \tag{3}$$
$$\beta = \text{atan}(\frac{l_r}{l_f+l_r}\tan(\delta_f)) \approx \frac{l_r}{l_f+l_r}\delta_f \tag{4}$$

in which x, y, ψ are the coordinates and orientation of the vehicle center of mass w.r.t. a global frame. v denotes the vehicle translational velocity and δ_f denotes the steering angle. The geometric parameters l_f, l_r represent the distance from the vehicle center to the front and rear axles. The angle β denotes the direction of the current velocity of the center of mass w.r.t. the longitudinal axis of the car. The equation for β is obtained from the no slip condition at the front and rear wheel. For small steering angles $\delta_f \approx 0$ the slip angle β is equal to the ratio of $\frac{l_r}{l_r+l_f}$ times the steering angle δ_f. For the two extreme cases $l_r = 0 \rightarrow \beta = 0$ and $l_f = 0 \rightarrow \beta = \delta_f$. The solution for $\dot{\psi}$ is obtained from the observation $\omega = \dot{\psi} = \frac{v}{R} = \frac{v\cos(\beta)\tan(\delta_f)}{l_r+l_f}$ in which R denotes the distance between the COG and the instantaneous center of rotation O.

The function

```
[xdot] = kinematicsBicycle(x, u, lf, lr)
```

implements the bicycle model to simulate simplified car-like vehicle kinematics and dynamics. The function takes as input arguments the state vector `x`, the control input

vector `u` as well as the geometric parameters `lf` and `lr`. The model utilizes vehicle speed v and steering angle δ_f as input. It computes the time derivative states according to equations (1)-(3).

The following code template generates a bicycle model object which geometric parameters correspond to the F1/10 racecar and simulates an open loop steering at constant velocity $v_0 = 3$ m/s and constant steering angle $\delta_0 = \pi/6$ rad for T_f seconds. The integration time is the actual elapsed time interval between consecutive loop iterations rather than the nominal sample time Ts. That way the simulation provides for overruns of the rate object, in which the computations are not completed within the sample time. The code is given in the template *.mlx* file.

```
% Kinematic bicycle model
lr = 0.2;           % distance from geometric center to front axle
lf = 0.2;           % distance from geometric center to rear axle
kinematicModel = bicycleKinematics('Wheelbase',lr+lf, ...
                                   'MaxSteeringAngle', pi/4, ...
                                   'VehicleSpeedRange', [0 20]);

Ts = 0.1;           % sample time
Tf = 60;            % simulation time
v0 = 3;             % constant velocity in m/s
delta0 = pi/6;      % constant steering angle
u = [v0 delta0];    % v delta_f

X = [];             % history of poses
U = [];             % history of controls

TLast = 0;
rateObj = robotics.Rate(1/Ts);
rateObj.reset;

while (rateObj.TotalElapsedTime < Tf)

  TNow = rateObj.TotalElapsedTime;    % timestamp current iterations
  TSim = TNow - TLast;                % time between consecutive iterations
  TLast = TNow;                       % timestamp for previous iteration

  [t,x] = ode45(@(t,x)kinematicsBicycle(x, u, lf, lr),[0 TSim], x0);
  x0 = x(end,:);

  X = [X; x0];
  U = [U; u];
  plotBicycle(x0, u, lr, lf);
  hold on;
  plot(X(:,1), X(:,2), 'g-');
  hold off;
  axis equal;

  waitfor(rateObj);
end
```

The helper function

```
plotBicycle(x, u, lr, lf);
```

visualizes the bicycle at pose $\mathbf{x} = [x\ y\ \psi]$ with the steering angle δ_f from $\mathbf{u} = [v\ \delta_f]$.

1) Modify the open loop simulation such that the steering angle linearly increases from $-\delta_0$ to $+\delta_0$ over the course of the simulation by manipulating the steering angle d_f (which corresponds to the second element of `u`). The time t corresponds to the variable `TNow`.

$$\delta_f(t) = \delta_0(\frac{2t}{T_f} - 1.0) \tag{5}$$

Matlab / Gazebo Simulation

This section establishes the communication between Gazebo and Matlab. Motion commands in Matlab are published on the corresponding topics for velocity and steering in ROS and passed on to the Gazebo simulation. Gazebo simulates the motion of the F1/10 car in the Gazebo environment and publishes the vehicle pose. Launch the race car simulation using

```
roslaunch racecar_gazebo f1tenth_gazebo_cosim_empty.launch
```

The overall task of controlling and navigating the car along the racetrack is broken down into the following subtasks

- Subscribing to the desired sensors like for example the lidar
- Subscribe to the topics which return the vehicle's current position
- Generating an empty message to be sent through the publisher
- Car parameters
- Pure Pursuit Controller initial Parameters
- While Loop that calculates and publishes through the empty message the velocity commands.

Code for setting up the publisher and subscriber for the Gazebo simulation:

```
% subscriber for vehicle pose either via transforms or via odom
% The topic name is '/vesc/odom'
carPoseSub = ...

% publisher on vehicle speed v and steering angle deltaf.
% The topic name is '/control_command'
```

```
[controlPub, controlMsg] = ...

% set initial vehicle pose in Gazbeo
% The topic name is '/gazebo/set_model_state'
[setPosePub, setPoseMsg] = ...
setPoseMsg = Pose2SetPoseMsg(setPoseMsg, x0);
send(setPosePub,setPoseMsg);
```

The `controlMsg` is of message type `ackermann_msgs/AckermannDriveStamped`. It contains the field `Drive` where the `Speed` and `SteeringAngle` of the vehicle can be set.

Replace the Matlab `ode` function of the bicycle model

```
[t,x] = ode45(@(t,x)kinematicsBicycle(x, u, lf, lr),[0 TSim],x0);
```

with the corresponding interaction with the Gazebo simulation:

```
send(controlPub, controlMsg);
carPoseMsg = receive(carPoseSub);
```

2) Initialize the ROS connection and the ROS publishers and subscribers as described above.

3) Send the constant velocity v_0 and the from $-\delta_0$ to $+\delta_0$ linearly changing steering angle to the Gazebo simulation. Record the vehicle pose and compare the path with the Matlab simulation.

Clothoids

A vehicle that travels along a circular path is subject to a constant lateral acceleration. Assume a road transition from a straight line segment onto a circular arc. Such a path requires a discontinuous change of the steering angle and an abrupt change in lateral acceleration at the tangent point. The first effect requires a complex maneuvering of the driver, the second causes discomfort to the passengers. Therefore road layouts are designed such that the road (path) curvature κ changes in a continuous fashion.

The so called arc length parametrization parameterizes the curvature of a path $\kappa(s)$ by the path length s. For a clothoid curve, also known as Euler spiral or Cornu spiral the curvature

$$\kappa(s) = \kappa_0 + d\kappa_0 s \tag{6}$$

changes linearly with the arc length. The radius of the curve is given by

$$\rho(s) = \frac{1}{\kappa(s)} \tag{7}$$

The straight line is a special case of a clothoid with $\kappa(s) = 0$. The general relationship between the arc length representation $\kappa(s)$ and the Cartesian space is given by

$$x(s) = x_0 + \int_0^s \cos(\theta(s'))ds' \tag{8}$$

$$y(s) = y_0 + \int_0^s \sin(\theta(s'))ds' \tag{9}$$

$$\theta(s) = \theta_0 + \int_0^s \kappa(s'))ds'. \tag{10}$$

The equation for $\theta(s)$ is obtained from the relationship between the arc length s,

$$ds = \rho d\theta \tag{11}$$

of a circular arc $d\theta$ of radius ρ.

Figure 2 illustrates three clothoids with different parameters $\kappa_0, d\kappa_0$.

In general there are no closed form solutions for Eq. 10. In case of a clothoid the relationship between the tangent line orientation $\theta(s)$ and curvature is given by

$$\theta(s) = \kappa_0 s + d\kappa_0 \frac{s^2}{2} \tag{12}$$

The relationship for the position $x(s), y(s)$ of a clothoid is established by the so called Fresnel integrals

Matlab provides an internal function that computes the Fresnel integrals for a clothoid.

```
z = matlabshared.tracking.internal.scenario.fresnelg(s, drho, rho, theta)
```

in which

$$z = \int_0^s e^{i(\kappa_0 s' + d\kappa_0 \frac{s'^2}{2} + \theta)} ds' \tag{13}$$

in which $\kappa_0, d\kappa_0, \theta$ are scalar constants and s is a real array. z is complex number with $x = \mathrm{Re}(z)\,,\ y = \mathrm{Im}(z)$

These calculations are done in the provided `Path` class.

Figure 2 has been generated with the code

```
kappa0=0;
dkappa0=0.3;
theta=0.0;
s=0:0.01:1.5*pi;
z = matlabshared.tracking.internal.scenario.fresnelg(s,kappa0,kappa0,0);
poses=[real(z); imag(z); dtheta];
plot(real(z),imag(z),'b-');
...
```

Path Description with Clothoids

A complete track is defined as a sequence of consecutive clothoids, in which the next clothoid starts at the end of the previous clothoid. The class object Path defined in `Path.m` provides an implementation for track composition from clothoids.

```
classdef Path
  properties
    numClothoids {mustBeInteger}
    initialPose {mustBeNumeric}
    s {mustBeNumeric}  % total path length of i-th clothoid
    cums {mustBeNumeric} % cumulative path length
    poses {mustBeNumeric}   % final pose of i-th clothoid
    kappa0 {mustBeNumeric} % initial curvature of i-th clothoid
    dkappa0 {mustBeNumeric} % initial rate of change of curvature of i-th clothoid
   kappa=kappa0+dkappa0*s;
  end

  methods
    ...
  end
end
```

The path is constructed from a sequence of clothoids defined by the parameters $s_i, \kappa_{0i}, d\kappa_{0i}$, in which s_i denotes the length of the i-th clothoid segment κ_{0i} initial curvature and the optional parameter $d\kappa_{0i}$ (default 0) rate of change of curvature. The optional parameter `initialPose` defines the start pose of the track with the default value [0 0 0].

```
function obj = Path(s, kappa0, dkappa0, initialPose)
```

The following code instantiates and visualizes an oval track composed of two straight line segments (length $s = 10$, $\kappa_0 = 0$) and two semi-circles (length $s = \pi * R$, $\kappa_0 = 1/R$)

```
d = 10;  % length straight segment
R = 6;% radius of turns
track = Path([d pi*R d pi*R],[0 1/R 0 1/R]);
track.plot();
```

The following code instantiates and visualizes a more complex track composed of ten clothoids of variable curvature, including S-shaped segments (change of sign of curvature).

```
% path lengths
s = [25 20 30 30 35 30 45 32 40];
% curvatures
kappa0 = [0 0.15 -0.1 -0.1 0 -0.05 0.08 0.18 0];
% rate of change of curvatures
dkappa0 = [0.0 0.0015 0.002 0.01 0.002 0.0 0.0 -0.015 0];
track = Path(s, kappa0, dkappa0);
track = track.closingClothoid;
track.plot();
```

[2]Source: RST

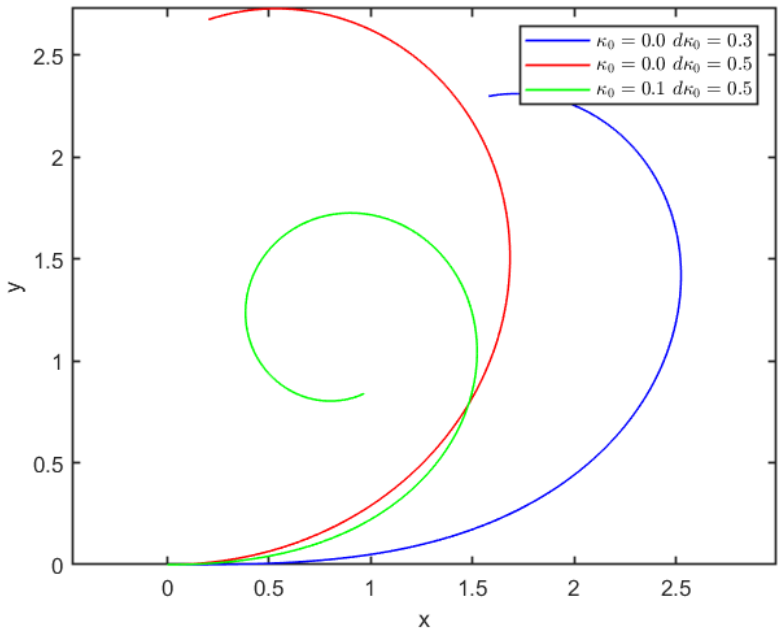

Figure 2: Clothoids: $\kappa_0 = 0.0$ $d\kappa_0 = 0.3$, $\kappa_0 = 0.0$ $d\kappa_0 = 0.5$, $\kappa_0 = 0.1$ $d\kappa_0 = 0.5$ [2]

The circles in the plot mark the start and end of segments. The command

```
track = track.closingClothoid;
```

determines the clothoid that connects the final pose of the last segment with the initial pose of the first segment and adds this segment to the path in order to close the loop.

4) Instantiate the oval track path and visualize it in Matlab. The oval track can be launched in the Gazebo simulation using the command:

```
roslaunch racecar_gazebo f1tenth_gazebo_cosim_oval.launch
```

5) Instantiate the complex track path and visualize it in Matlab. The complex track can be launched in the Gazebo simulation using the command:

```
roslaunch racecar_gazebo f1tenth_gazebo_cosim_complex.launch
```

6) Instantiate your own race track path from a sequence of clothoids, close the track and visualize it in Matlab.

Spatial Formulation

[3]Source: RST

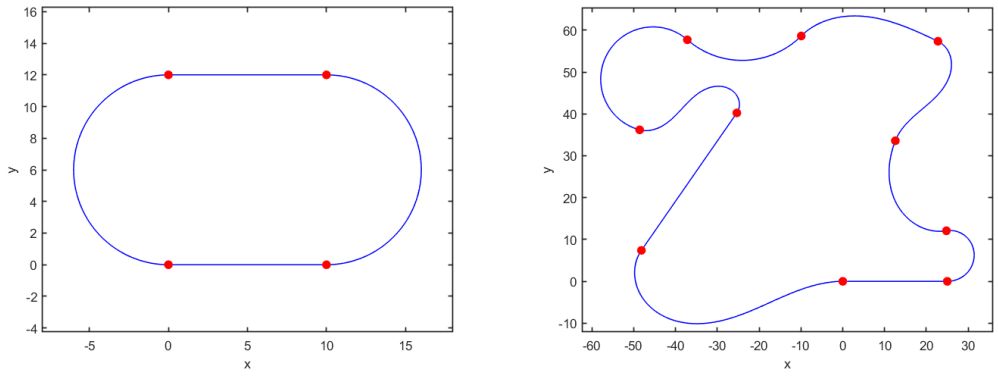

Figure 3: left: oval track, right: complex race track [3]

The next task is concerned with tracking a reference path composed of a sequence of clothoids. The spatial reformulation expresses the error between the vehicle pose and the reference path w.r.t. a path centric frame rather than a global frame.

The curvature of the reference path is given by κ^σ. The curvature $\kappa^\sigma(s)$ and poses $[X^\sigma(s),\ Y^\sigma(s),\ \psi^\sigma(s)]$ of the reference path are parameterized in terms of path length s. The member function

```
function kappas = kappa(obj, s)
```

of the class `Path` maps the path length s onto the local curvature of the reference path $\kappa^\sigma(s)$. The member function

```
function pose = sigma(obj, s)
```

maps the path length s onto the global pose of the reference path $[X^\sigma(s),\ Y^\sigma(s),\ \psi^\sigma(s)]$.

Figure 4 illustrates the spatial relationship between the reference path frame $[X^\sigma,\ Y^\sigma,\ \psi^\sigma]$ and the vehicle frame $[X, Y,\ \psi]$. The lateral error e^y and the orientation error e^ψ (not to be confused with the exponential function) are defined by

$$e^y = \cos(\psi^\sigma)(Y - Y^\sigma) - \sin(\psi^\sigma)(X - X^\sigma) \tag{14}$$
$$e^\psi = \psi - \psi^\sigma \tag{15}$$

where $[X^\sigma,\ Y^\sigma,\ \psi^\sigma]$ are the position and orientation of the reference point on the path at s.

The member function

```
function [ey, epsi] = poseError(obj, pose, s)
```

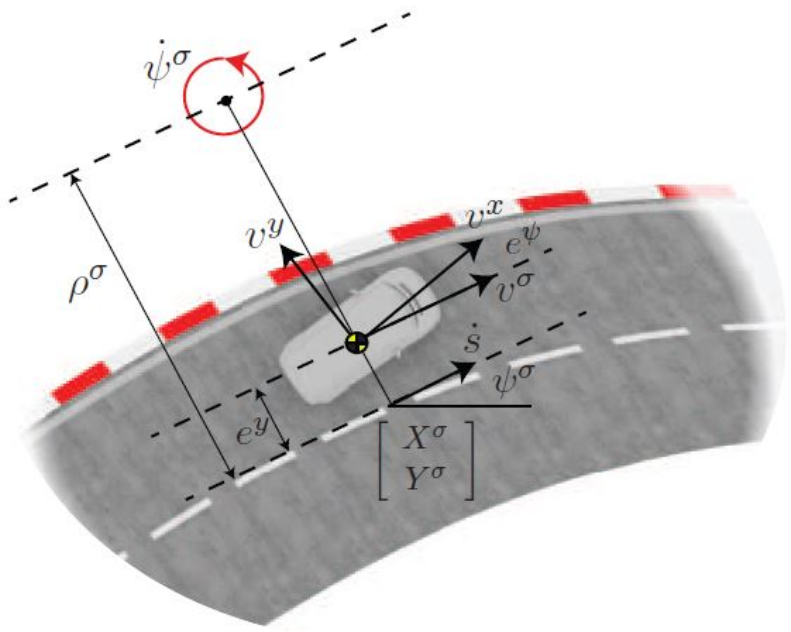

Figure 4: Spatial reformulation of bicycle kinematics [4]

determines the path relative pose error vector $\mathbf{e} = [e^y\ e^\psi]$ between the current robot pose `pose` and the reference pose at pathlength s. The lateral and orientation error provide the input signal for a feedback control law.

The spatial reformulation introduces a new state vector $\Xi = [e^y(s)\ e^\psi(s)\ t(s)]$. The role of time t and path length s are reversed, in the spatial reformulation s becomes the independent variable and time $t(s)$ becomes the dependent state. Applying the spatial formulation to the kinematic equations of the bicycle model in **??** reveals

$$\begin{aligned} e^{y'}(s) &= (v\sin(e^\psi) + v\frac{l_r}{l_f+l_r}\delta_f\cos(e^\psi))/\dot{s} \\ e^{\psi'}(s) &= \frac{\dot{\psi}}{\dot{s}} - \kappa^\sigma(s) \\ t'(s) &= \frac{1}{\dot{s}} \end{aligned} \tag{16}$$

with the inputs $u = [v\ \delta_f]$. The velocity along the path is given by

$$\dot{s} = \kappa^\sigma\dot{\psi} = \frac{1}{1 - e^y/\kappa^\sigma}(v\cos(\beta)\cos(e^\psi) - v\sin(\beta)\sin(e^\psi)) \tag{17}$$

[4]Source: Towards Time-Optimal Race Car Driving using Nonlinear MPC in Real-Time, Robin Verschueren, Stijn De Bruyne, Mario Zanon, Janick V. Frasch, Moritz Diehl, CDC 2014

For small slip angles β the path velocity $\dot{s}$ can be approximated by

$$\dot{s} = \kappa^\sigma \dot{\psi} = \frac{1}{1 - e^y/\kappa^\sigma}(v\cos(e^\psi) - v\frac{l_r}{l_f + l_r}\delta_f \sin(e^\psi)) \tag{18}$$

The vehicle pose in the global frame is recovered from $e^y,\ e^\psi, \psi$ according to

$$\begin{aligned} X &= X^\sigma - e^y \sin(\psi^\sigma) \\ Y &= Y^\sigma - e^y \cos(\psi^\sigma) \\ \psi &= \psi^\sigma + e^\psi \end{aligned} \tag{19}$$

Path Following

In mere path following the longitudinal velocity v is kept constant and the steering angle δ_f regulates the lateral and orientation error w.r.t. lookahead point on the reference path. The objective is to regulate the lateral and orientation error with a proportional controller at a constant velocity. Path following control law with $v = \text{const}$

$$\delta_f = -k_y e^y - k_\psi e^\psi \tag{20}$$

The feedback control in Eq. 20 is embedded into a rated loop, in which the bicycle kinematic model is numerically integrated according to current state x and control u.

```
x0 = [0 1.5 0.2]; % initial pose with lateral and orientation error
s0 = 0;           % initial path length

ky = 0.1;         % gain for lateral error
kpsi = 0.3;       % gain for orientation error
ks = 0.5;         % gain for longitudinal error

TLast = 0;
rateObj.reset;

while (rateObj.TotalElapsedTime < Tf)

  TNow = rateObj.TotalElapsedTime; % timestamp current iterations
  TSim = TNow - TLast;             % time between consecutive iterations
  TLast = TNow;                    % timestamp for previous iteration

  s = track.pose2PathLength(x0, s0); % closest point on path
  s0 = s;

  [ey, epsi] = track.poseError(x0, s + v0 * TSim);  % pose error at lookahead point

  deltaf = ... % lateral control law

  u = [v0 deltaf]; % compose control input


  [t,x] = ode45(@(t,x)derivative(kinematicModel, x, u),[0 TSim], x0);

  x0 = x(end, :); % initial pose next iteration

  clf;
  track.plot();
  hold on;

  X = [X; x0];
  U = [U; u];
  plotBicycle(x0, u, lr, lf);
  hold on;
  plot(X(:,1),X(:,2), 'g-');
  hold off;
  axis equal;

  waitfor(rateObj);
end
```

Here, `track` corresponds to the generated oval or complex (or custom) track respectively.

An advanced control scheme anticipates the track curvature and incorporates the equivalent steering angle as a feedforward action.

$$\delta_f = -k_y e^y - k_\psi e^\psi + \Delta\delta_{ff} \tag{21}$$

$$\Delta\delta_{ff} = \text{sgn}(\kappa(s)) \tan^{-1}\left(\sqrt{\frac{l_f^2\kappa(s)^2}{1 - l_r^2\kappa(s)^2}}\right) \tag{22}$$

The steering angle feedforward action $\Delta\delta_{ff}$ corresponds to a circular path of curvature κ.

7) Implement the path following feedback controller according to Eq. 20 and observe the tracking error.

8) Augment the path following controller with a feedforward action according to Eq. 21. What is the effect on the tracking error?

9) Repeat steps 6) and 7) using Gazebo for simulating the vehicle motion. Record the vehicle pose and compare the path with the Matlab simulation. Test the path following controller for the oval as well as the complex track.

Trajectory Tracking

In trajectory tracking the path description is augmented with a timing law, such that the reference path length $\hat{s}(t)$ becomes a function of time t. The longitudinal velocity remains no longer constant as in path following but regulates the path length error

$$e^s = s - \hat{s} \tag{23}$$

The control law for the longitudinal velocity is given by

$$v = -k_s e^s + \dot{\hat{s}} \tag{24}$$

in which $\dot{\hat{s}}$ denotes the reference path velocity. We assume a constant reference velocity $\hat{v} = 3$ m/s for trajectory following such that

$$\begin{aligned} \dot{\hat{s}} &= \hat{v} \\ \hat{s} &= \hat{v}t \end{aligned} \tag{25}$$

In the Matlab code the constant velocity v_0 is replaced by the control law for the longitudinal motion

```
vref = 3;          % 3 m/s reference velocity
sref = vref*TNow;  % reference path length
```

10) Augment the path following controller to a trajectory tracking controller by replacing `v0` in

```
u = [v0 deltaf]; % proportional control law
```

with the longitudinal motion control law in Eq. 24.

```
v = ...  % longitudinal motion control law
% pose error at lookahead point
[ey, epsi] = track.poseError(x0, s + v * TSim);
u = [v deltaf]; % proportional control law
```

Kinodynamic Constraints

The kinematic bicycle model ignores the vehicle dynamics, in particular the forces required to guide the vehicle along the trajectory. Constraints are categorized into

- input constraints on v, δ_f, in particular limits of absolute steering angle, maximum velocity, maximum steering rate and maximum acceleration and deceleration

$$\begin{aligned} v_{min} &\leq v &&\leq v_{max} \\ a_{min} &\leq \dot{v} &&\leq a_{max} \\ \delta_{min} &\leq \delta_f &&\leq \delta_{max} \\ \dot{\delta}_{min} &\leq \dot{\delta}_f &&\leq \dot{\delta}_{max} \end{aligned} \tag{26}$$

(27)

- geometric constraints, in particular non-violation of track limits $|e^y| < w(s)$ in which $w(s)$ denotes the lanewidth at path lenght s
- dynamics constraints (physics of driving), in particular the limitation of lateral acceleration (forces) in turns that need to be balanced by the tire friction forces. The maximum lateral acceleration compensated by friction is given by

$$|a_{lat_{max}}| = \mu g \tag{28}$$

in which $\mu \in [0, 1]$ denotes the static friction coefficient and $g = 9.81 m/s^2$ the gravity acceleration. The centrifugal acceleration of the car in a curve is related to the path velocity and the curvature by

$$a_{lat} = \frac{v^2}{R} = \frac{v^2 \delta_f}{l_r + l_f} \tag{29}$$

in which R denotes the radius of the curve. Combining Eqs. 28 and 29 imposes a limit on the maximum velocity

$$|v_{max,dyn}| \leq \sqrt{\frac{\mu g (l_r + l_f)}{|\delta_f|}} \tag{30}$$

We merely impose the lateral acceleration constraint. For a more sophisticated approach the kinodynamic and geometric constraints can be considered within a model predictive control framework.

11) Modify the code such that the maximum velocity is limited to the constraint in Eq. 30. Furthermore, assure that the overall maximum velocity v_{max} is limited to 10m/s.

```
v = min(min(v,...), 10); % dynamics constraint for vmaxdyn
[ey, epsi] = track.poseError(x0, s + v * TSim); % pose error at lookahead
point
u = [v deltaf]; % proportional control law
```

12) Increase the reference velocity $\hat{v}$ and observe until which maximum reference velocity the car is able to keep track despite having to slow down in turns.

References

[1] Robin Verschueren, Stijn De Bruyne, Mario Zanon, Janick V. Frasch, Moritz Diehl, Towards Time-Optimal Race Car Driving using Nonlinear MPC in Real-Time, 53rd IEEE Conference on Decision and Control, 2014