Hochschule Konstanz
Fakultät Elektrotechnik
und Informationstechnik

Final Exercise as fulfillment of the exam – Truck/Trailer System realized in MATLAB

Report on the Examination Project

University of Applied Sciences Konstanz

Faculty of Electrical Engineering and Information Technology (EI)

Study Programs “Electrical Engineering and Management”, and “Automotive IT”

Authors:	Niklas Kugler, 298335
	Manuel Reichelt, 298035
Semester:	Summer Semester 2021
Course:	Automotive Control Systems (WPF)
Lecturer:	Stefan Wirtensohn
Location and Date:	Konstanz, 23.07.2021

Content

Figures

1 Introduction and Project Description

The goal of the project is to create a *Matlab* Framework where a vehicle travels together with a trailer, for instance a Truck/Trailer system, from a random initial pose to an arbitrary final pose, given a specified path. This must perform both forward and backwards motion. The desired final pose shall be reached, even if the actual initial pose differs from the one presumed for the path planning (cf. [1], p. 4). Therefore, a controller needs to be designed ensuring that the center point of the Trailer axle follows a predefined trajectory, described by a geometric path and a velocity profile. The derivatives of the path at the start and end point are defined by the Truck/Trailer configuration and the steering angle of the Truck (cf. [2], p. 3).

The project combines two important aspects of autonomous driving or automated driving. On the one hand, there is the planning part, where a route needs to be planned from A to B, and on the other hand, there is the control aspect of controlling the steering wheels. In general, this also includes the acceleration/brake pedal in order, to optimally follow the desired trajectory and move the car from A to B. Hence, these aspects are already playing an important role in the latest development of ADAS and thus are already in execution in the everyday world. Applications that should be mentioned in this context are automatic parking or automatic lane changing on the motorway the vehicle can already perform itself.

To fulfil all requirements of the examination project, it is necessary to integrate a controller responsible for stabilizing feedback control, plus using the flatness approach. Nevertheless, it is important to implement a stable controller. Thus, choosing the corresponding parameters properly is crucial for the control system of this project.

To achieve a functional simulation framework for a Truck/Trailer planning and control system, the framework not only needs the algorithms for the planning and control but also has to perform in an efficient and robust matter for different testing scenarios. The framework should also be well documented and be implemented in a modular and efficient coding style. In addition, the framework should also be user-friendly and intuitive for the user, as well as visualizing the results of the simulation clearly.

The motivation to attend the lecture Automotive Control Systems is mainly based on deepening the theory acquired in the control theory lecture and particularly doing a project with practical orientation. Because we are curious about learning more about autonomous driving the project is an optimal task deepening the knowledge about planning and programming a simulation framework in an automotive purchase.

2 Kinematic Model of the Truck/Trailer

The project relies on a simple car kinematic model, which is shown in figure 1. The simple car model is characterized by a front and rear axle being connected by an axle with the length d_0. The reference point or rather the center of gravity is the middle point of the rear axle. Another main characteristic is the equal steering angle φ for both wheels. This corresponds to kinematic bicycle model where the two front wheels are lumped as well as the two rear wheels are lumped into a unique wheel located at the center of the axle (cf. [3], p. 722 f.).

The equations for the kinematic model are defined as:

$$\dot{x} = v_0 \cos(\theta)$$

$$\dot{y} = v_0 \sin(\theta)$$

$$\dot{\theta} = \frac{v_0}{l_0} \tan(\theta)$$

The kinematic model for a truck with one trailer is defined as follows:

$$u_1 = \dot{\varphi}$$

$$u_2 = \dot{v_0}$$

$$\dot{\theta_1} = \frac{v_0}{d_1} \sin(\theta_0 - \theta_1)$$

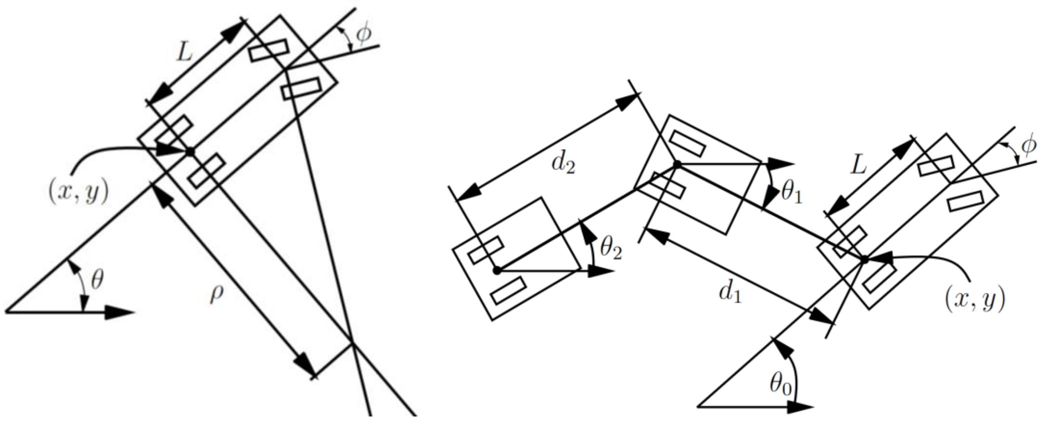


Figure 1: Vehicle Model (left) and Vehicle Pulling two Trailers (right)

An extension to the simple car model, which is also an integral part of the project, can be made by attaching one or more trailer. For the sake of reducing complexity of the project only one trailer is added to the vehicle. Consider a Truck/Trailer system that pulls one or more trailers as shown in figure 1 (right). Each trailer is attached to the center of the rear axle of the body in front of it (cf. [3], p. 730). The center of gravity is the middle point of the trailer's axle. The hitch length is an important parameter for a vehicle/trailer system, which indicates the distance from the center of the rear axle of a trailer to the point at which the trailer is hitched to the next body.

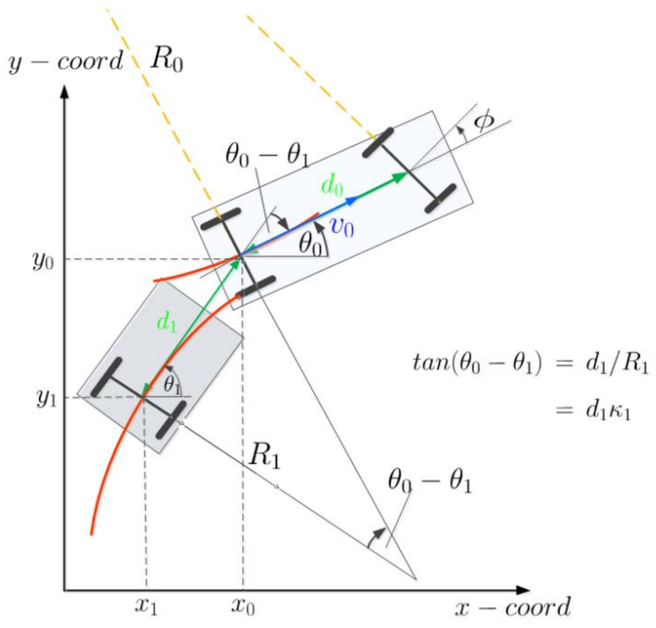


Figure 2: Truck/Trailer System with all Geometrical Dimensions

Figure 2 above shows the Truck/Trailer system meeting the requirements of our project. Thereby, it takes all geometrical dimension into account, such as the coordinates of the center of gravity of truck (x_0, y_0) and trailer (x_1, y_1) as well as their orientation and steering angles (cf. [2], p. 3). The relation between the center of gravity of truck and trailer can be described as:

$$x_1 = x_0 - d_1 * \cos(\theta_1)$$

$$y_1 = y_0 - d_1 * \sin(\theta_1)$$

3 Project Structure – Block Diagram

One of the main functionalities of the code is solving an ordinary differential equation (ODE) regarding stabilizing feedback control. The solution of this ODE returns a matrix of the corresponding *State*.

```
[t, State] = ode45(@ODEFunc, [0,T], odeStartState, [], Parameters);
```

The *State* indicates the configuration of the Truck/Trailer system with its coordinates as well as its orientation and steering angles. With the help of the *State* matrix, it is possible to determine the configuration of the vehicle at discrete points *t*, laying the ground for further analysis and vivid trajectory visualization. Focusing on the approach of the described ODE it becomes clear that this includes the interaction of several function files (*ODEFunc.m*). These function files follow a certain control loop shown below in figure 3. However, the displayed control loop represents the closed loop variant, but it is worth mentioning that there is an open loop variant as well without implementing stabilization control.

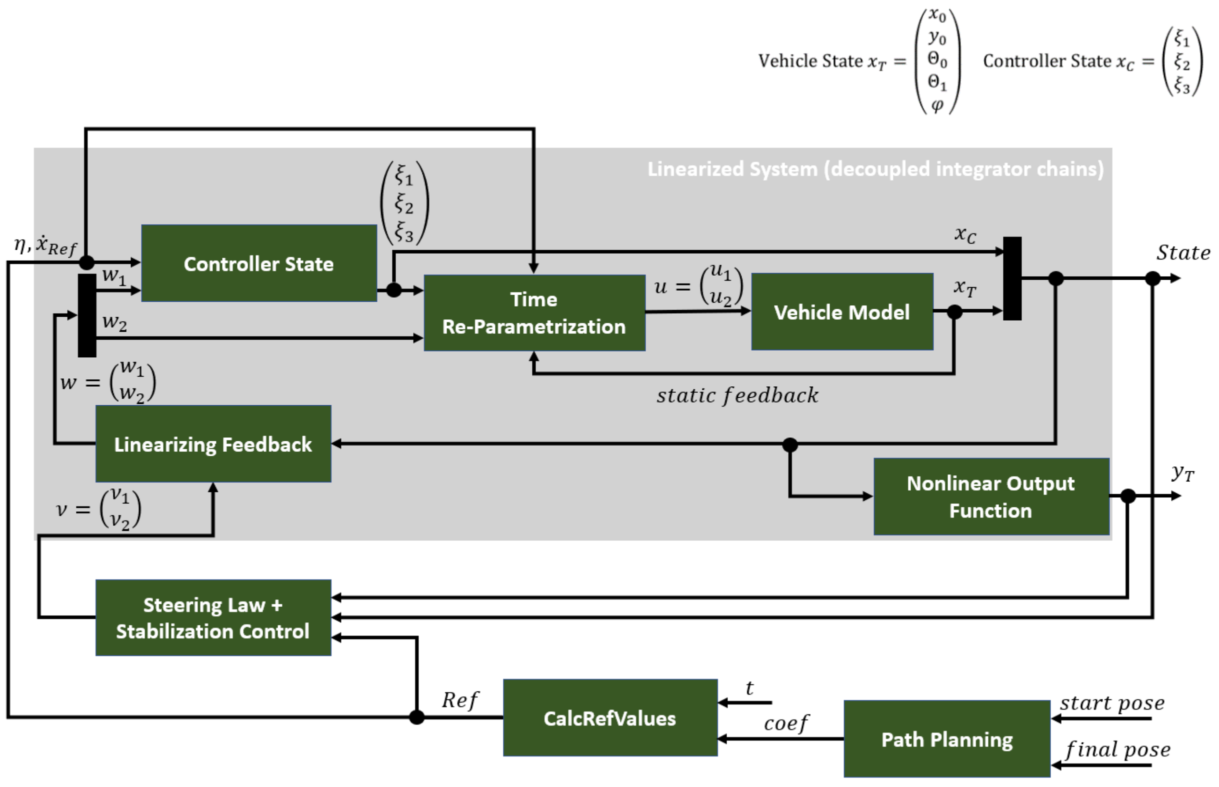

Vehicle State $x_T = \begin{pmatrix} x_0 \\ y_0 \\ \Theta_0 \\ \Theta_1 \\ \varphi \end{pmatrix}$
Controller State $x_C = \begin{pmatrix} \xi_1 \\ \xi_2 \\ \xi_3 \end{pmatrix}$
Linearized System (decoupled integrator chains)
$\begin{pmatrix} \xi_1 \\ \xi_2 \\ \xi_3 \end{pmatrix}$
$\eta, \dot{x}_{Ref}$
w_1
w_2
Controller State
Time Re-Parametrization
$u = \begin{pmatrix} u_1 \\ u_2 \end{pmatrix}$
Vehicle Model
x_C
x_T
$State$
$w = \begin{pmatrix} w_1 \\ w_2 \end{pmatrix}$
$static\ feedback$
Linearizing Feedback
Nonlinear Output Function
y_T
$v = \begin{pmatrix} v_1 \\ v_2 \end{pmatrix}$
Steering Law + Stabilization Control
Ref
CalcRefValues
t
$coef$
Path Planning
$start\ pose$
$final\ pose$

Figure 3: Control Loop of Stabilizing Feedback as Block Diagram (Closed Loop Variant)

The following chapter is all about explaining the single block elements and the corresponding function files of the control loop, respectively.

3.1 Path Planning

Planning is an integral part of autonomous driving which happens after sensing the environment around the vehicle. In general, there are three terms when speaking of planning: Motion Planning, Path Planning and Trajectory Generation. Motion Planning describes the task of determining a feasible way to move a vehicle or a mobile robot from an initial pose to a desired final pose (cf. [4], p. 2). A pose describes the current position and orientation of a vehicle or mobile robot, respectively. In other words, Motion Planning is responsible to find a collision-free path through a configuration space by using geometric or probabilistic algorithms. Figure 4 shows an example of a vehicle parking into a parking lot trying to not collide with any obstacles. The obstacles are apparently indicated by the black area. The red box indicates the initial pose in the same way as the blue box indicates the final pose. This part of the planning process is not relevant for the project, since it is assumed that there is an endless configuration space that has no collision objects and thus is completely collision-free. With the Motion Planning various constraints could be added for the planning like velocity, acceleration limits or steering limitations. After the Motion Planning, the next step throughout the planning process is the Trajectory Generation. When there is a motion plan, a geometric path can be put through the configuration space by generating a trajectory to move from the initial pose to the final pose. Sometimes, it is necessary to execute concrete Path Planning because the vehicle must reach the desired pose in a certain way, for example the shortest path.

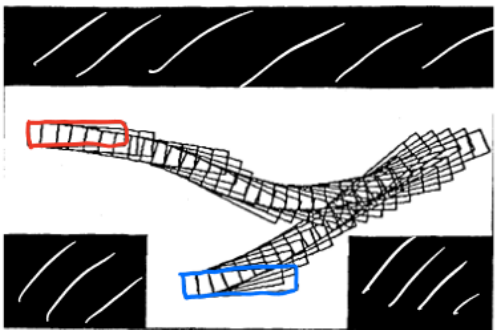


Figure 4:Collision-free Motion Planning into Parking Lot

For the Trajectory Generation based on simple Motion Planning, the path needs to be compliant at least with the following requirements indicating the start and the final pose (cf. [4]):

- Initial Pose: $[x_0,\ y(x_0),\ \theta_0(x_0),\ \theta_1(x_0),\ \varphi(x_0)]$
- Final Pose: $[x_1,\ y(x_1),\ \theta(x_1),\ \theta_1(x_1),\ \varphi(x_1)]$

The parameters must be chosen so that the feature of the curve $y(x)$ match the state configuration at the initial and final pose. Therefore, there are eight conditions that need to be fulfilled, which results into a 7th order polynomial as prototype regarding simple Path Planning:

$$y(x) = ax^7 + bx^6 + cx^5 + dx^4 + dx^4 + ex^3 + fx^2 + gx + h.$$

Simply using a 7th order polynomial for the generation of the trajectory sometimes results in trajectories being not optimal but rather following the curve of a characteristic polynomial function. For some configurations the trajectory runs like a sine function or like a parable which is not the optimal path to reach the desired pose but rather relies on the constraints of such a polynomial for trajectory generation and constraints of the vehicle model based on the starting and final configurations. Some examples can be seen in the attachment.

3.2 Steering Law and Stabilization Control

Function File	**'Steering_StabControl.m'**
Input	Struct with *Ref* values, *State* and y_T (feedback), Struct with *Parameters*
Output	Vector υ (ny)

The Steering Law makes a significant difference between open and closed loop. Whereas the open loop variant only assigns υ to the fourth derivative of x and y reference values, the closed loop variant enables the implementation of stabilization control, by using the feedback of the state and the flat output.

If it is assumed that the Truck/Trailer system meets ideal conditions, choosing the basic setting $\upsilon = y_{Ref}^{(4)}$ would steer the system along y_{Ref} (cf. [5], p. 4). However, it may happen that the system is exposed to environmental interferences. Therefore, certain error dynamics must be considered. The tracking error describes the difference between a reference value and its corresponding actual value, for example $e_x(t) = x_{Ref}(t) - x(t)$ and $e_y(t) = y_{Ref}(t) - y(t)$, respectively. Same is true for their derivatives (cf. [5], p. 6). By realizing these circumstances in this function file, the previously calculated reference values are extracted from the *Ref* struct at first. The corresponding actual values for x and y and their derivatives are calculated with the help of certain Lie Derivatives. The appropriate Lie Derivatives are provided in a separate function file. After the tracking error dynamics $e_x, \dot{e}_x, \ddot{e}_x, \dddot{e}_x, e_y, \dot{e}_y, \ddot{e}_y, \dddot{e}_y$ have been completely determined, it is possible to build the equations of the control law for feedback linearization listed in figure 5.

$$\nu_1 = x_{Ref}^{(4)} + k_3\dddot{e}_x(t) + k_2\ddot{e}_x(t) + k_1\dot{e}_x(t) + k_0 e_x(t)$$

$$\nu_2 = \underbrace{y_{Ref}^{(4)}}_{\text{Feed Forward}} + \underbrace{k_3\dddot{e}_y(t) + k_2\ddot{e}_y(t) + k_1\dot{e}_y(t) + k_0 e_y(t)}_{\text{Structure of PD controller}}$$

Figure 5: Equations of Tracking Control Law Considering Error Dynamics

The resulting error dynamics are stable if the weighting factors k_0, k_1, k_2, k_3 fulfil the Hurwitz criterion. The characteristic polynomial of $e_y^{(4)} + k_3\dddot{e}_y(t) + k_2\ddot{e}_y(t) + k_1\dot{e}_y(t) + k_0 e_y(t)$ must be a Hurwitz polynomial, meaning that all zeros have a negative real part. This will be fulfilled, if there are only strictly positive northwestern sub-determinants D_i of the corresponding $n \times n$ matrix (cf. [6], p. 104 ff.).

$$H = \begin{pmatrix} k_{n-1} & k_{n-3} & k_{n-5} & k_{n-7} \\ k_{n-0} & k_{n-2} & k_{n-4} & k_{n-6} \\ 0 & k_{n-1} & k_{n-3} & k_{n-5} \\ 0 & k_{n-0} & k_{n-2} & k_{n-4} \end{pmatrix} \overset{\boxed{\text{with polynomial degree } n = 4}}{=} \begin{pmatrix} k_3 & k_1 & k_{-1} & k_{-3} \\ k_4 & k_2 & k_0 & k_{-2} \\ 0 & k_3 & k_1 & k_{-1} \\ 0 & k_4 & k_2 & k_0 \end{pmatrix} \overset{\boxed{\text{with } k_0 = 1,\ k_{-\dots} = 0}}{=} \begin{array}{cccc} D_1 & D_2 & D_3 & D_4 \\ \left(\begin{matrix} k_3 & k_1 & 0 & 0 \\ 1 & k_2 & k_0 & 0 \\ 0 & k_3 & k_1 & 0 \\ 0 & 1 & k_2 & k_0 \end{matrix}\right) \end{array}$$

Calculation according to Laplace's development theorem:

$D_1 = k_3$	$\rightarrow \mathbf{k_3 > 0}$ ①
$D_2 = k_3 * k_2 - 1 * k_1$	$\rightarrow k_3 * k_2 - 1 * k_1 > 0$ $\rightarrow \mathbf{k_3 * k_2 > k_1}$ ②
$D_3 = k_3 * \begin{vmatrix} k_2 & k_0 \\ k_3 & k_1 \end{vmatrix} - 1 * \begin{vmatrix} k_1 & 0 \\ k_3 & k_1 \end{vmatrix} = k_3(k_2 k_1 - k_3 k_0) - (k_1)^2$	$\rightarrow k_1 k_2 k_3 - k_0 (k_3)^2 - (k_1)^2 > 0$ $\rightarrow \boldsymbol{k_1 k_2 k_3 > k_0 (k_3)^2 + (k_1)^2}$ ③
$D_4 = k_0 * D_3 = k_0(k_1 k_2 k_3 - k_0 (k_3)^2 - (k_1)^2)$	$\rightarrow k_0 k_1 k_2 k_3 - (k_0 k_3)^2 - k_0 (k_1)^2 > 0$ $\rightarrow \boldsymbol{k_0 k_1 k_2 k_3 > (k_0 k_3)^2 + k_0 (k_1)^2}$ ④

Figure 6: Calculation of Conditions regarding the Hurwitz Criterion

The theoretical calculation leads to four conditions to be fulfilled seen above in figure 6. However, it is more practical to have a closer look at the characteristic polynomial in order, to determine the parameters of the controller. As already mentioned, the characteristic polynomial $\Delta s = s^4 + k_3 s^3 + k_2 s^2 + k_1 s + k_0$ is required to be a Hurwitz polynomial. The approach with $d_0 = 2$ and all zeros chosen $s = -0.5$ (strictly negative real part) results in:

$$\Delta s = \left(s + \frac{1}{d_0}\right)^4 = \left(s + \frac{1}{2}\right)^4 = s^4 + 2s^3 + \frac{3}{2}s^2 + \frac{1}{2}s + \frac{1}{16}.$$

Next, the coefficient comparison with $\Delta s = s^4 + k_3 s^3 + k_2 s^2 + k_1 s + k_0$ leads to the following weighting factors.

```
k0 = 0.0625;
k1 = 0.5;
k2 = 1.5;
k3 = 2;
```

3.3 Linearized System

In general, linear systems are a widely studied classification of differential models, especially regarding control theory (cf. [3], p. 740). There are many helpful techniques from linear algebra implemented within linear systems. This enables to yield proper control laws. At this point it is worth to mention the state transition equation of a linearized system with the real-valued constant matrices A and B:

$$\dot{x} = f(x, u) = Ax + Bu\,.$$

However, one of the crucial characteristics is that the equations resulting from matrix multiplications “are linear in the state and action variables” ([3], p. 740). Considering discrete planning problems, linearized systems provide the possibility to define differential models depending on time (cf. [3], p. 741). Therefore, the corresponding matrix entries may be functions of time too.

In the case of this project the linearized system consists of decoupled integrator chains and includes the linearizing feedback functionality as well as a controller and a vehicle state. Additionally, the flat output approach is realized within the nonlinear output function.

3.3.1 Linearizing Feedback

Function File	**‘LinearizingFeedback.m’**
Input	Vector υ, *State* (feedback), Struct with *Parameters*
Output	Vector w

For implementing feedback linearization there are several steps to be observed. In the first step, the output $y(t)$ must be differentiated until the input appears (cf. [5], p. 2). In this case, $y(t)$ is differentiated four times resulting in a quadruple integrator chain. This work is done in the function file *‘CalcRefValues.m’*. Next, the fourth derivatives of x_{Ref} and y_{Ref} are set to a new input ν as described in chapter 3.2. Now, this function file is responsible for calculating the input w of the controller state using the feedback of *State*. Considering the new input υ, the following linear equation system is set up for feedback linearization (cf. [2], p. 7):

$$\begin{pmatrix} \upsilon_1 \\ \upsilon_2 \end{pmatrix} = \begin{pmatrix} \mathcal{L}_{g1}\mathcal{L}_f^3 h_1 & \mathcal{L}_{g2}\mathcal{L}_f^3 h_1 \\ \mathcal{L}_{g1}\mathcal{L}_f^3 h_2 & \mathcal{L}_{g2}\mathcal{L}_f^3 h_2 \end{pmatrix} \begin{pmatrix} w_1 \\ w_2 \end{pmatrix} + \begin{pmatrix} \mathcal{L}_f^4 h_1 \\ \mathcal{L}_f^4 h_2 \end{pmatrix}.$$

The Lie derivatives were given within a separate file and only required a simple function call. To realize the approach with the ‘*linsolve’* command in *Matlab*, the linear equation system must be transformed to a certain structure:

$$A * w = b \leftrightarrow \begin{pmatrix} \mathcal{L}_{g1}\mathcal{L}_f^3 h_1 & \mathcal{L}_{g2}\mathcal{L}_f^3 h_1 \\ \mathcal{L}_{g1}\mathcal{L}_f^3 h_2 & \mathcal{L}_{g2}\mathcal{L}_f^3 h_2 \end{pmatrix} \begin{pmatrix} w_1 \\ w_2 \end{pmatrix} = \begin{pmatrix} \upsilon_1 \\ \upsilon_2 \end{pmatrix} - \begin{pmatrix} \mathcal{L}_f^4 h_1 \\ \mathcal{L}_f^4 h_2 \end{pmatrix}.$$

Lastly, the command below is executed to receive the vector w.

```
w = linsolve(A, b);
```

3.3.2 Controller State

Function File	**‘ControllerState.m’**
Input	Vector w, η, $\dot{x}_{Ref}$, Controller State x_C
Output	Derived Controller State x_C

This function file determines the states ξ of the decoupling controller with the help of the resulting vector w from the linearizing feedback as well as η and $\dot{x}_{Ref}$ from the calculation of reference values. It also introduces a new parametrization according to the arc length σ.

Due to the time dependency of the system model described in chapter 2, a singularity problem can occur because of division by zero (cf. [5], p. 8 ff.). The original chosen controller state was the velocity of the system v. As a solution one or several controller states ξ are introduced which are also called “pseudo velocity in a new ‘time’ σ” ([2], p. 6). This expression refers to the new parametrization indicated before.

$$v := \xi * \frac{d\sigma}{dt} \text{ with } v = \sqrt{\dot{x}(t)^2 + \dot{y}(t)^2} \text{ and } d\sigma = \sqrt{\dot{x}(t)^2 + \dot{y}(t)^2} * dt$$

The formulas above show the correlation between parametrization regarding time and arc length of the path. Here applies: $\xi = 1$ if the vehicle exactly follows a predefined path, and $\xi \approx 1$ if the vehicle is nearly on the path (cf. [5], p. 13).

In the case of a controller for the Truck/Trailer system, there are three controller states ξ_1, ξ_2, ξ_3 summarized in the vector ξ. For calculation the equations below must be noticed (cf. [2], p. 9).

$$(1)\ \dot{\xi}_1 = \xi_2 * \eta * \dot{x}_{Ref}$$
$$(2)\ \dot{\xi}_2 = \xi_3 * \eta * \dot{x}_{Ref}$$
$$(3)\ \dot{\xi}_3 = w_1 * \eta * \dot{x}_{Ref}$$

However, the re-parametrization takes places in a separate function file implementing the time dependency again. This is needed to calculate the final control output vector $u(t)$.

3.3.3 Vehicle Model

Function File	**‘TruckTrailerModel.m’**
Input	Vector u, Vehicle State x_T, Struct with *Parameters*
Output	Derived Vehicle State x_T

This function file implements the vehicle model consisting of a truck with one trailer. This includes – analogue to the description of the Truck/Trailer system in chapter 2 – the x- and y-coordinates of the vehicle, its orientation angles, and the steering angle regarding the truck´s front axle. The vehicle state can be considered as the plant of the control system and it follows the structure:

$$\dot{x}_T = f(x_T, u).$$

For implementation, the upcoming formulas for the single elements of the Truck/Trailer system must be considered (cf. [2], p. 9). Remark: d_0, d_1 describe the distances between axles.

$$(1)\ \dot{x}_0 = u_1 * \cos(\theta_0)$$
$$(2)\ \dot{y}_0 = u_1 * \sin(\theta_0)$$
$$(3)\ \dot{\theta}_0 = {}^{u_1}/_{d_0} * \tan(\varphi)$$
$$(4)\ \theta_1 = {}^{u_1}/_{d_1} * \sin(\theta_0 - \theta_1)$$
$$(5)\ \dot{\varphi} = u_2$$

3.3.4 Nonlinear Output Function

Function File	**'NonlinearOutputFunction.m'**
Input	*State*, Struct with *Parameters*
Output	Flat output y_T

The nonlinear output function follows the structure $y_T = h(x_T)$. Having a closer look at the vehicle´s geometry, both truck and trailer have a center of gravity at the middle of the rear axle. Based on the coordinates for the truck x_0 and y_0, this function calculates the coordinates of the trailer x_1 and y_1 which provide a flat output.

$$x_1 = x_0 - d_1 * \cos(\theta_1)$$
$$y_1 = y_0 - d_1 * \cos(\theta_1)$$
$$y_T = \binom{y_1}{y_2} = \binom{x_1}{y_1}$$

The differential flatness approach involves several advantages, such as dynamic inversion and fairly easy calculation of the feedforward control law (cf. [7], p. 8 ff.). Hence, three conditions must be fulfilled. Firstly, the elements of the nonlinear output y_T are required to correlate with the components of the state. Secondly, the whole state should be expressed by y_T and a finite number of derivatives. Thirdly, the input should be expressed by y_T and a finite number of derivatives.

4 Visualization and Modular Plotting

The code framework and the results of the examination project is required to be user-friendly and extensible. To design the handling preferably intuitive for the user, the program implements configuration interfaces described in chapter 4.3. However, the main part of the project´s output focuses on visualizing the results of tests and simulation.

This chapter is about explaining how several function files are encapsulated and interact with each other to streamline the main file. After that, the approach of rotation matrices illustrated providing an extensive functionality for our use cases. Then, four plot types are presented, summarizing the potentiality of the code framework, for example simulations.

4.1 Encapsulation of plotting function files

Implementing the visualization of the vehicle with its axles and tiers as well as orientation angles turns out to be complex task. It is important to avoid an overload of code in the main function, and guarantee modularity and extensibility instead. Therefore, some plotting functionalities are relocated in several function files for a better code structure. How these files interact with each other in order, to create vivid plots and simulations, is explained in this chapter.

Function File	**‘MovingPlot.m’**
Input	Matrix *State,* Structs *start, final, Parameters,* Travel Time *T*
Output	Simulation plot with moving Truck/Trailer

In its plotting code section, the main function directly refers to a function call of the ‘moving plot’. It is responsible for drawing the complete state of the vehicle at discrete points in time. This results in a moving plot from the start pose to the predefined final pose. The simulation is realized within a for-loop where the reference points of truck and trailer are determined at first, followed by a graphical representation of the vehicle, calling the *‘PlotState.m’* function. After some configuration regarding plot design, the iterating plot update enables a flowing sequence of moving images. Thereby, the command *‘clf’* indicates to clear the current figure as long, as the final iterating value is not reached. Then the visualization is plotted again due to the next step of iteration. However, beside the simulation of a moving vehicle, the plot also represents the Truck/Trailer trajectories.

Function File	**‘PlotState.m’**
Input	Matrix *State*, refPointTruck, refPointTrailer, State, k
Output	Plot of the vehicle (Truck with one Trailer) with all geometrical data

This function file fulfils the task to create a plot around one reference point. This includes all the geometrical dimensions of the Truck/Trailer system. For instance, the Truck has one main axle, a front and rear axle as well as four tiers. Every displayed element of the vehicle requires a separate function call of *‘SingleElementPlot.m’* and *‘SingleElementPlot_Steering’*, respectively. Beside tiers and axles, the function additionally implements the plotting of the truck´s and the trailer´s shape in form of colored rectangles.

Function File	**‘SingleElementPlot.m’ / ‘SingleElementPlot_Steering.m’**
Input	Rotation angle *theta, refPoint, delta* start (x, y), *delta* end (x, y)
Output	Plot of a single element of the vehicle (axle, tier)

The function file takes over responsibility for the plotting of single elements of the vehicle, e. g. axles and tires, as mentioned above. This happens by calling the rotation matrix for the corresponding start and end point. Thereby, it is possible to take the orientation of the single elements into account. A more detailed description is provided in the following chapter.

Function File	**‘CallRotationMatrix.m’**
Input	Rotation angle *theta, refPoint, delta* (x, y)
Output	Coordinates *x, y*

By calling the rotation matrix, one can implement a rotation around a given viewpoint, for instance the reference point of the vehicle. The functionality behind is described in detail in the next chapter as well.

4.2 Rotation Matrix

For the visualization of the Truck/Trailer system along a desired trajectory, the rotational matrix needs to be introduced.

The Truck/Trailer system is only considered in 2D because we only regard the yaw angle standing orthogonal on a two-dimensional plane. Therefore, the rotation matrix can be simplified as follows

$$R = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix},$$

where θ is the yaw angle. The coordinates of the center of gravity of both trailer and truck are known by the *State* matrix for every timestamp/sampling point. Alternatively, they can be computed with the geometrical dependencies of the Truck/Trailer system shown in chapter 2. The determination of the rotated coordinates can be described as followed. Firstly, there is a matrix vector multiplication, where the rotation matrix R needs to be multiplied with the vector Y. The vector Y consists of the difference from the point that needs to be rotated and the reference point. Secondly, the result (vector X) needs to be added with the reference point to finally get the rotated coordinates.

$$Y = \begin{bmatrix} \Delta x \\ \Delta y \\ 0 \end{bmatrix} \qquad X = R * Y$$

$$x_{rotated} = X(1) + x_{ref}$$

$$y_{rotated} = X(2) + y_{ref}$$

For the visualization of the Truck/Trailer system in a global inertial frame, it is necessary to rotate the body frame of the Truck/Trailer system with the appropriate orientation angle (yaw angle) at any given time around the mentioned center of gravity. The body frame of the truck and the trailer consists of a fixed geometry with a rectangle for the shape and the axles with indicated wheels for the demonstration of the current steering. This vehicle/body frame is shown in figure 7.

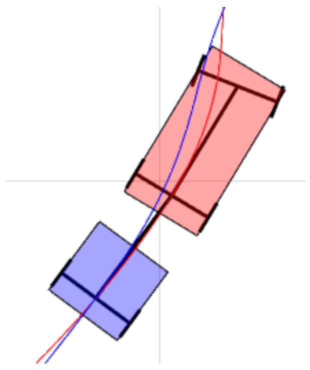
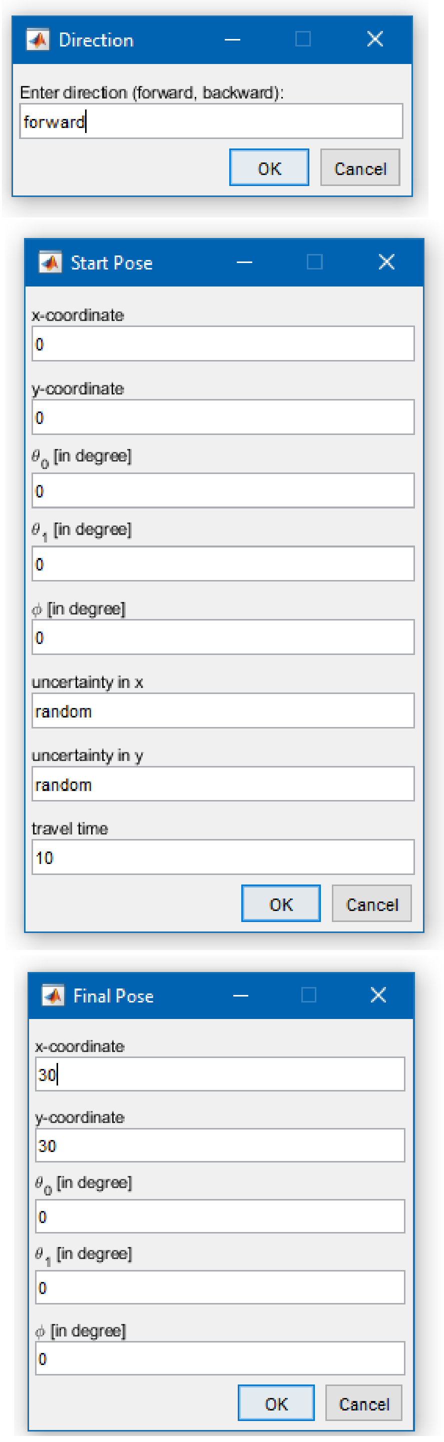


Figure 7: Body of the Truck/Trailer System

The function '*CallRotationMatrix.m'* mainly is responsible for the rotation of the body of the Truck/Trailer system. The function is modular for the sake of being able to use it several times for the rotation of all single axles and the vehicle shape. The function receives as input an angle θ, a coordinate and a distance $\Delta x/\Delta y$. The coordinate represents the reference point, and it corresponds to the center of gravity, when observing the orientation of the Truck/Trailer. Every axle to be rotated according to the orientation angle is characterized by two coordinates that create the axle, when connecting those two coordinates. For the rotation of the axles according to the orientation angle θ, the function needs the distance in the x- and y-direction from the reference point for every point separately. Then this coordinate is rotated around the reference point by the orientation angle θ. For the rotation of the wheels according to the steering angle, the same function can be used. Instead of the orientation angle the steering angle is used as rotation angle, and instead of the center of gravity the middle point of the front axle is used as new reference point. The rotation pipeline starts with calling the *'PlotState'* function. With the iterating parameter k, it is possible to either determine the rotation for every timestamp of the path or just for one single timestamp depending on the configuration of k. Here, the rotation angles get extracted from the state vector at timestamp k. There are the function calls for the axles and the shape of truck and trailer. Their fixed parameters define the Truck/Trailer geometry illustrated in figure 7. For the steering, there is an explicit function file because in the first place, it is necessary to rotate around the center of gravity with the orientation angle in order, to correctly visualize the wheels. The second step requires to rotate around the middle point of the front axle with the steering and the orientation angle.

4.3 Input Interface for Vehicle and Motion Configuration

One main objective of the simulation framework is to be able to do quick changes in the configuration to test different scenarios. Therefore, it is also recommended implementing a user interface to intuitively and simply being able to do the configuration, especially for other people who want to use the framework. When running the simulation of the Truck/Trailer system, there are three user interfaces displayed on the screen. The first UI shown in figure 8 enables the user to enter a predefined direction of motion. The user can decide between forward or backward motion of the Truck/Trailer system by entering the corresponding string. The forward motion is already predefined in the UI which means the user must explicitly change the configuration to 'backward' if he wants the vehicle to execute backward motion. After choosing between the direction of motion, the user is requested to enter the vehicle configuration for both poses starting with the start pose. The start pose interface is shown in figure 8 (middle) where the user can specify the configuration of the starting pose of the Truck/Trailer system in detail. The user can configure the coordinates of the trailer's center of gravity, the orientation for both truck and trailer in degree, the truck´s steering angle in degree as well as the travel time. The user is also able to edit an input for the uncertainty of the position both in x- and y-direction. It is possible to choose either a random position uncertainty or manually editing a specific deviation. A more detailed explanation of the uncertainty aspect follows in section 4.8. The last UI displayed below is the input interface for the vehicle configuration of the final pose. Again, the user can configure the coordinates of the trailer's center of gravity,



Enter direction (forward, backward): forward

Start Pose

x-coordinate	0
y-coordinate	0
θ_0 [in degree]	0
θ_1 [in degree]	0
ϕ [in degree]	0
uncertainty in x	random
uncertainty in y	random
travel time	10

Final Pose

x-coordinate	30
y-coordinate	30
θ_0 [in degree]	0
θ_1 [in degree]	0
ϕ [in degree]	0

Figure 8: Configuration UI

the orientation for both truck and trailer in degree as well as the truck´s steering angle indicating the desired end pose (position and orientation) of the vehicle.

4.4 Plot 1: Forward Motion – Moving Truck/Trailer

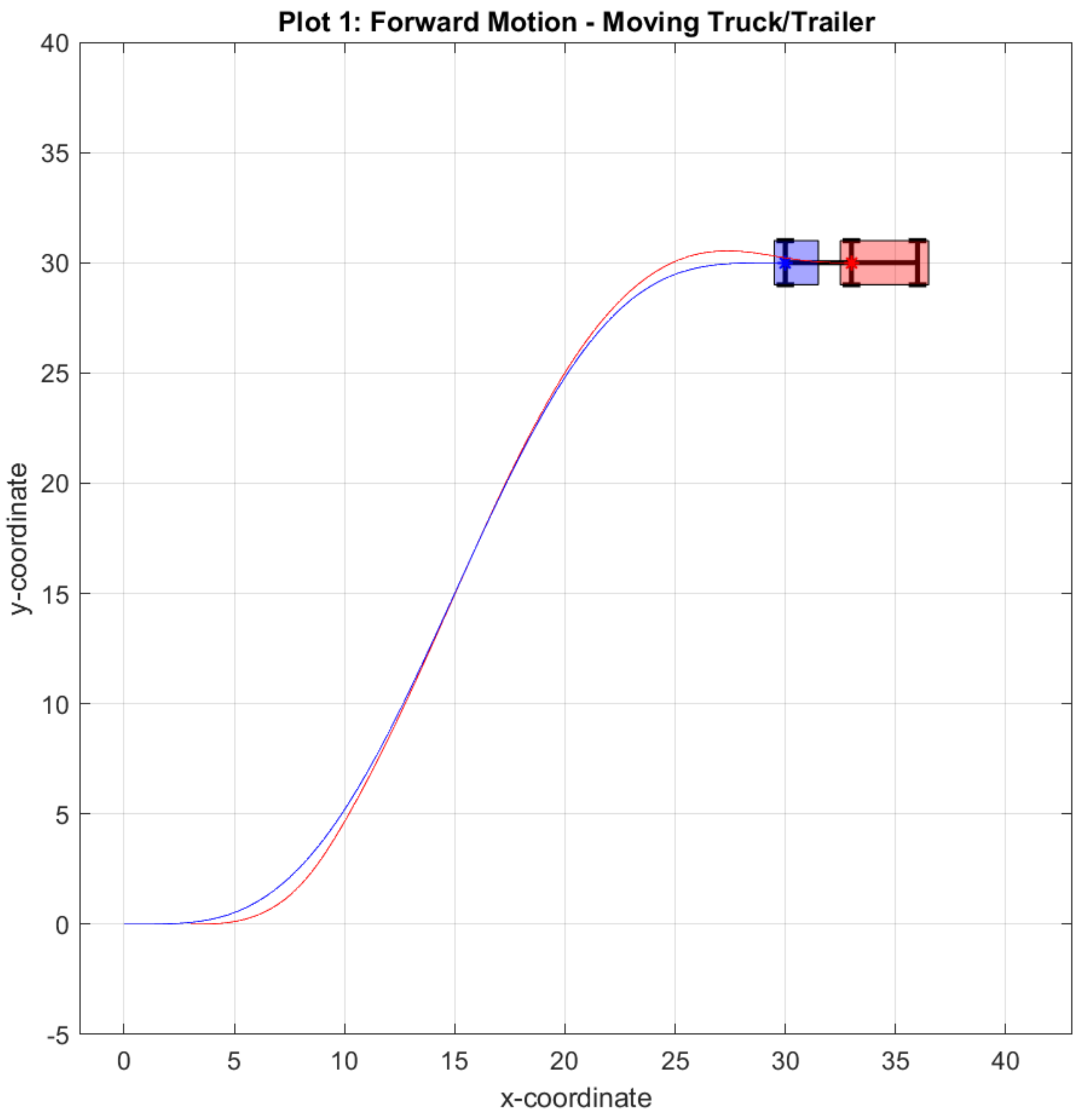


Figure 9: Forward Motion - Moving Truck/Trailer

The first plot creates a simulation of the Truck/Trailer system moving from an initial pose to a predefined final pose which can be edited in the corresponding configuration interfaces described in the previous chapter. For simulation of forward motion, it is crucial to edit the direction 'forward' (flag = 1, default value). Beside the animation of the moving vehicle, the plot additionally shows the resulting trajectories, and highlights the centre of gravity for both truck and trailer. Another feature of the simulation is the possibility to edit a travel Time T indicating a time span in which the final pose should be reached by the vehicle. The route of the vehicle follows a ‘s’ curve. Thereby, the truck needs to adjust its steering due to the alignment of the trailer. This becomes visible especially at the beginning and the end of the trajectories. In comparison the trajectories of truck and trailer overlap when there is a linear course.

4.5 Plot 2: Configuration View – Discrete Points of Truck/Trailer

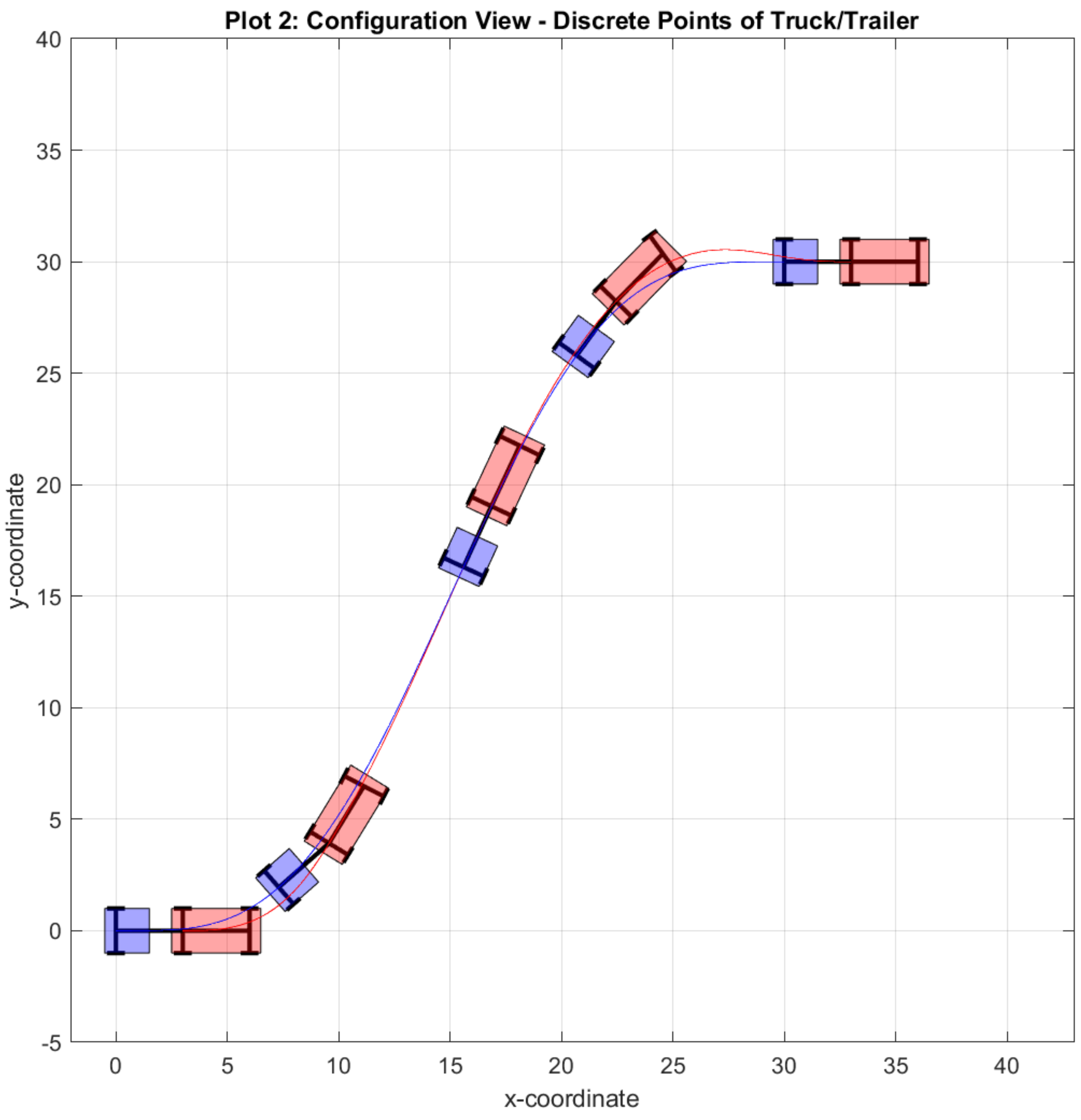


Figure 10: Configuration View - Discrete Points of Truck/Trailer

The second plot can be considered as an addition to the previously described simulation. The graph visualizes the Truck/Trailer system at discrete points of time along the trajectory. This view provides a more detailed look to test the configuration of the vehicle, such as orientation of the Truck/Trailer and the appropriate steering angle. It also gives a better overview in terms of analysis and plotting amount, when there is no possibility to display a moving plot – like in this report.

4.6 Plot 3: Reference Trajectory vs Resulting Trajectories

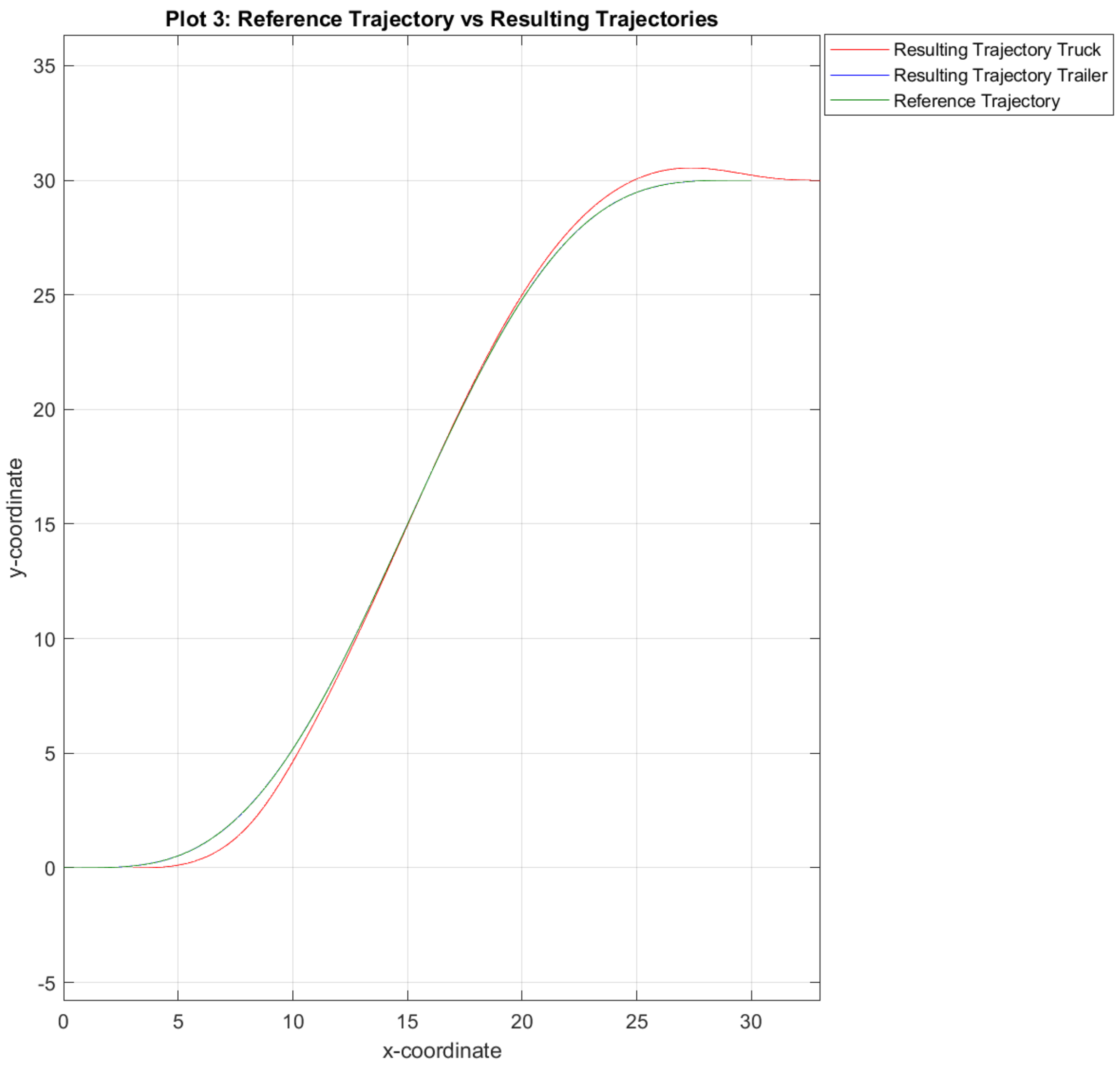
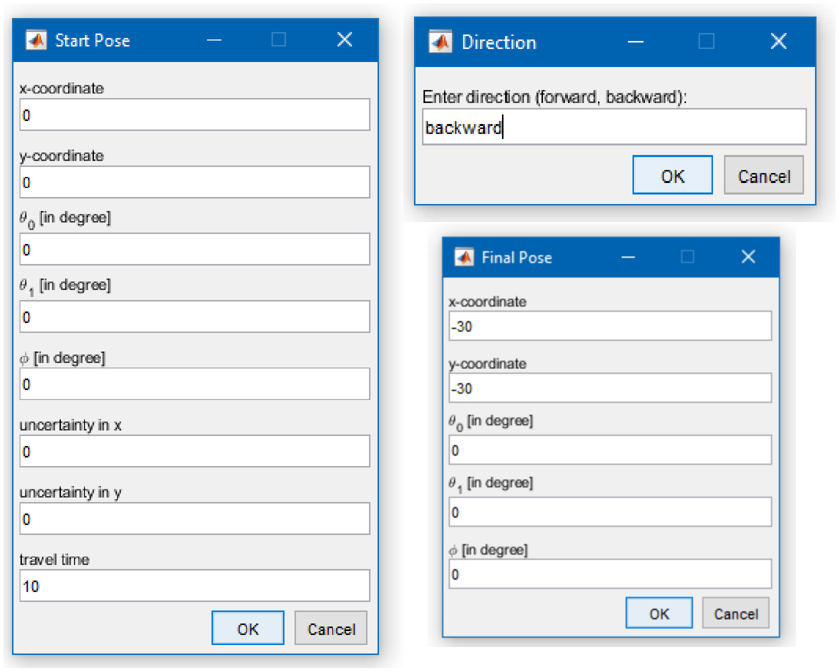


Figure 11: Reference Trajectory vs Resulting Trajectory

The third plot includes a reference trajectory based on the calculated reference values of x-/y-coordinates in the path planning function under ideal conditions. This view enables the observer to compare the reference trajectory with the resulting or the actual trajectories of the trailer, respectively. As can be seen in figure 10, there is maximum overlapping of the trailer´s trajectory and the reference trajectory (100% covered). Thus, the vehicle exactly follows the predefined path.

4.7 Add On: Backward Motion – Moving Truck/Trailer

Another feature of the simulation program is the specification of the direction. This enables the Truck/Trailer system to execute backward motion from an uncertain start pose to a predefined final pose. Therefore, the configuration interfaces must be adapted in order, to modify the first plot responsible for the driving simulation. In the case of backward motion, the interfaces are configured as follows:


Start Pose

x-coordinate
0
y-coordinate
0
θ_0 [in degree]
0
θ_1 [in degree]
0
ϕ [in degree]
0
uncertainty in x
0
uncertainty in y
0
travel time
10

Direction

Enter direction (forward, backward):
backward

Final Pose

x-coordinate
-30
y-coordinate
-30
θ_0 [in degree]
0
θ_1 [in degree]
0
ϕ [in degree]
0

Figure 12: Exemplary Configuration of the Interfaces for Executing Backward Motion

To implement the logic of backward motion in comparison to the execution of forward motion, the coordinates and angles of start and final pose need to be switched. This is because the configuration of the start pose is required to correspond to the parameters of forward motion responsible for backward motion, and vice versa. In concrete terms, this means that the final coordinates equal the start coordinates of the forward plot $(0,0)$. It is also possible to edit negative x and y coordinates regarding the final pose $(-30,-30)$.

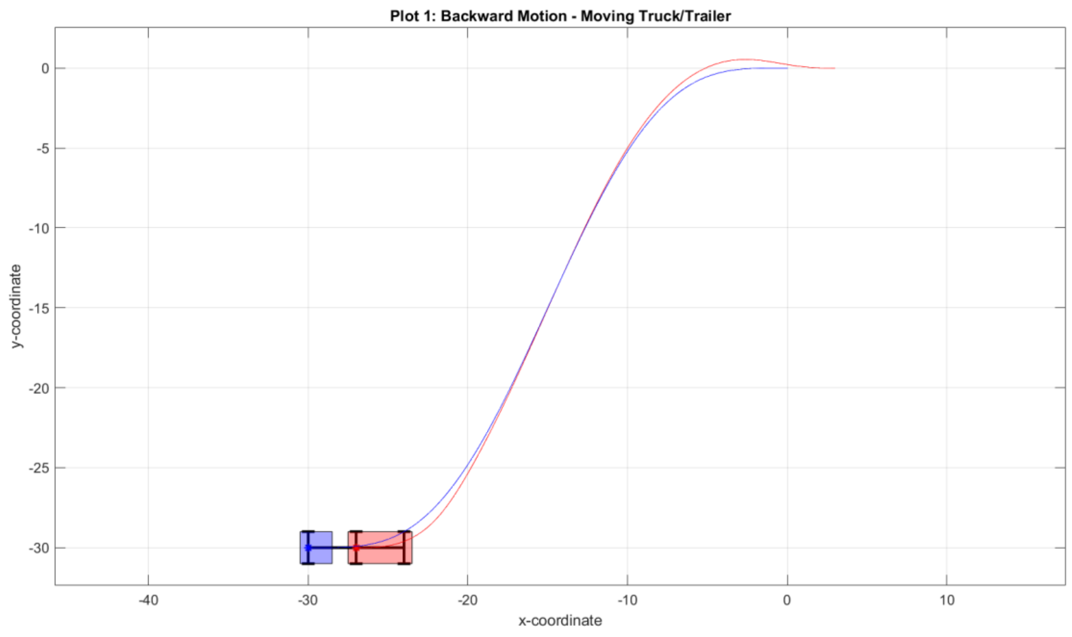


Figure 13: Backward Motion – Moving Truck/Trailer

However, there are more necessary adjustments to enable the backward motion of the vehicle. Firstly, one of the configuration interfaces indicates the direction of motion, resulting in a corresponding flag to be set, depending on the user´s input.

- Forward motion: Flag *'direction'* $= 1$
- Backward motion: Flag *'direction'* $= -1$

This makes it easy to differentiate between the two direction possibilities at the appropriate places of the code, by using if-else-branches.

Additionally, the calculation of the time parameter τ must be considered. The small *Matlab* code section below illustrates the differences.

```
% Time parametrization with helper variable tau
if direction == 1             % forward motion
    tau = t/T;
elseif direction == -1        % backward motion
    tau = (T - t)/T;
```

The next important change for implementing backward motion logic refers to the determination of the controller´s parameters discussed in chapter 3.2. Thereby, the characteristic polynomial $(s^4 + k_3 s^3 + k_2 s^2 + k_1 s + k_0)$ was compared with $\Delta s = \left(s \pm \frac{1}{d_0}\right)$. If the direction of motion is taken into account, the sign of Δs needs to be observed. Whereas '+' indicates forward motion like in the previous chapter, the sign '–' stands for the execution of backward motion. As a result, the coefficients or controller parameters need to be adapted, respectively. This manifests itself in a change of polarity of the parameters k_1 and k_3.

4.8 Uncertainty – Deviation of Start Pose

Recapitulating the crucial requirements of the examination project, it says that the "desired final pose shall be reached, even when the actual initial pose deviates from the one presumed for path planning" ([1], p. 4). It becomes clear that the position uncertainty cannot be handled only using a steering law. For this reason, there is a need for stabilization control in addition to the specified path implemented within a control loop (see chapter 3). Thus, it is possible to regulate the transient behavior of the vehicle due to control. As described before, the interfaces provide the option to edit specific deviations regarding the coordinates of the vehicle´s start pose. Hence, one can test various configurations and visualize the different starting points (viewpoint is the trailer´s center of gravity). These circumstances are shown in figure 14, plus the comparison to the presumed reference trajectory below in figure 15. The main goal is to reach the destination, independent from environmental influences and resulting position uncertainty in the beginning. The control system must guarantee that the trailer´s actual trajectory nestles on the corresponding reference trajectory, despite start pose deviation.

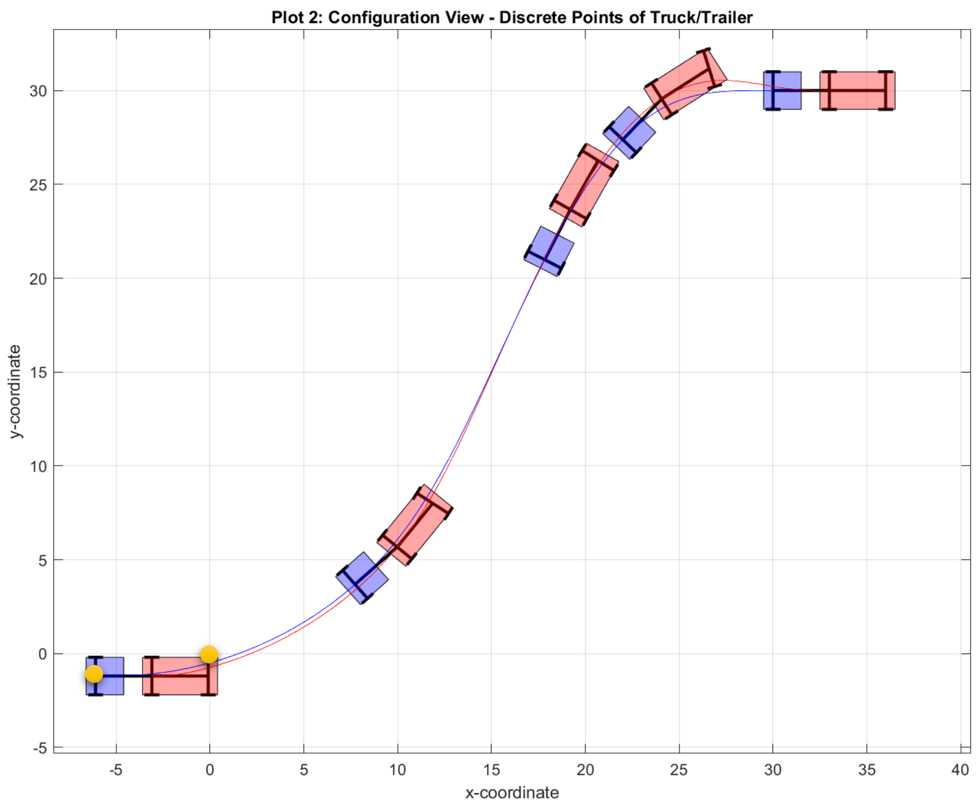


Figure 14: Uncertainty of the vehicle´s start pose

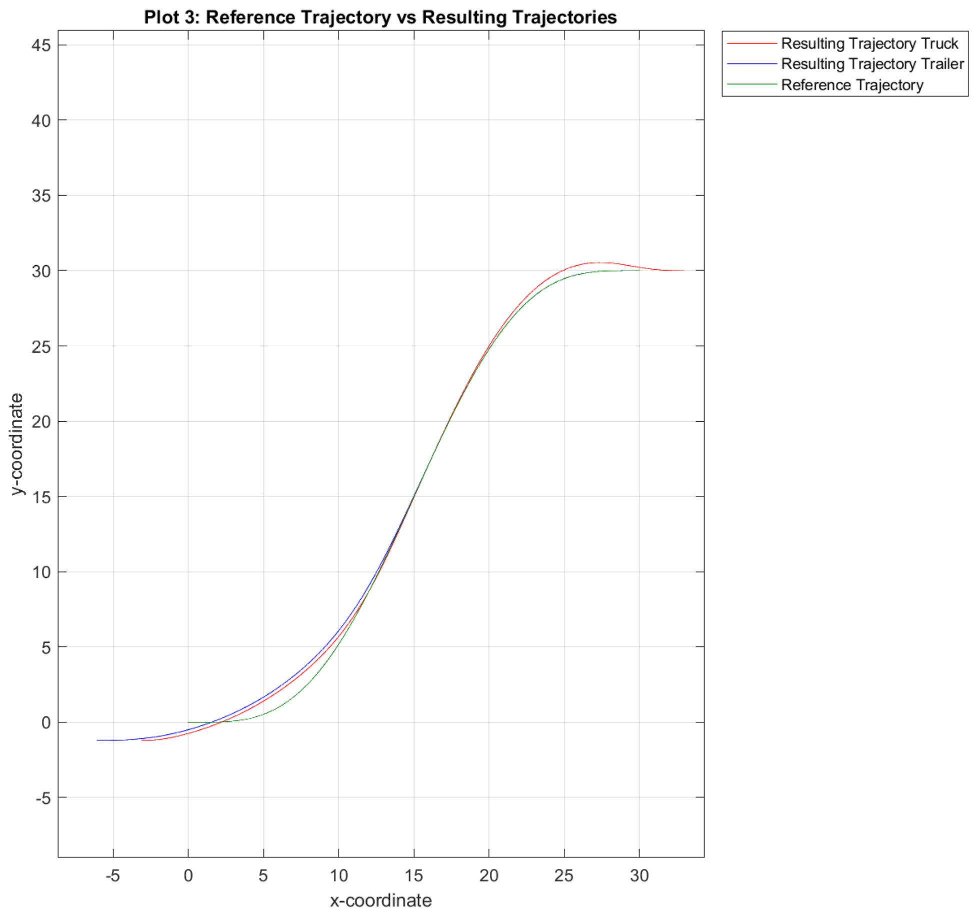


Figure 15: Start Pose Deviation from (0,0) Compared to the Presumed Path Planning

4.9 Further Testing Results

There are various test scenarios listed in the attachments including configuration of the Truck/Trailer System and the resulting plots (all scenarios refer to forward motion):

- Start Orientation Angle + End Orientation Angle of Truck (varying x-/y-coordinates)
  → see figure 16
- Start Orientation Angle + End Orientation Angle of Trailer (varying x-/y-coordinates)
  → see figure 17
- Start Orientation Angle + End Orientation Angle Combined (varying x-/y-coordinates)
  → see figure 18
- Start Steering Angle (of Truck) + possibly orientation angles (varying x-/y-coordinates)
  → see figure 19
- Long Route with big difference between start and final pose
  → see figure 20
- Random deviation of start pose (uncertainty) + varying orientation and steering angles
  → see figure 21

5 Conclusion

To round the examination project and report off, this final chapter will draw a conclusion regarding the results and the experiences gained during project realization. At the same time, it provides an outlook explaining possible extensions to the project.

First of all, the status quo of the project is mentioned here. Every main requirement is fulfilled and the *Matlab* framework for the Truck/Trailer System is completely implemented with helpful features. In addition to the default forward motion, we also managed to implement the possibility for the user to choose the direction, and thus executing backward motion as well. Furthermore, we designed a controller with its parameters to enable stabilization control despite position uncertainty. Simultaneously, we laid our focus on the framework to be extensively automated and modular so that certain changes are adopted at other appropriate places in the code. Consequently, the framework is quite robust against program failures. Moreover, we had the idea to additionally implement a user interface to provide the possibility of intuitive configuration for the sake of testing.

To put our experiences in a nutshell, we faced a couple of challenges during the project realization, for instance visualization, the implementation of backward motion and determining suitable parameters of the controller. However, our group work was remarkable, in terms of communication and cooperating to fulfil certain requirements and pass the milestones of the project. Our motivation and interest in realizing a project with autonomous driving in the context of control theory helped us during the implementation. Thereby, working with Git was quite helpful during the development of the code in terms of work distribution, versioning – for example referring to earlier versions – and many more.

Lastly, we would like to highlight possible extensions regarding path planning and control with constraints. This includes a realistic roadmap and collision avoidance with the integration of obstacles. Furthermore, the vehicle model, path planning and the linear controller could be more complex, as they are quite limited in their current state. For instance, this could lead to algorithms finding the shortest path from start to end pose.

To conclude, the state of the project now fulfils the main functions but can still be expanded with the listed extensions. In addition to this, we significantly expanded our skillset, in terms of motion planning and controlling, Truck/Trailer kinematics and programming an automotive application within a *Matlab* simulation framework.

Literature

[1] S. Wirtensohn, “Automotive Control Systems Lecture 1: Introduction,” Slide Script, HTWG, Konstanz, Summer Semester 2021. Accessed: Aug. 9 2021. [Online]. Available: https://moodle.htwg-konstanz.de/moodle/pluginfile.php/353243/mod_resource/content/1/AutoCtrlSys_Lecture_1_Intro_Kinematics.pdf

[2] S. Wirtensohn, “Automotive Control Systems Lecture 6: Flatness based control of the truck with one trailer,” Slide Scripts, HTWG, Konstanz, Summer Semester 2021. Accessed: Jul. 31 2021. [Online]. Available: https://moodle.htwg-konstanz.de/moodle/pluginfile.php/366393/mod_resource/content/1/AutoCtrlSys_Lecture_6_%20Flat%20Controller%20For%20Truck%20Trailer%20with%20Notes.pdf

[3] S. M. LaValle, “Differential Models,” in *Planning Algorithms*, S. M. LaValle, Ed., Cambridge U.K., 2006, pp. 715–786. Accessed: Jul. 30 2021. [Online]. Available: http://lavalle.pl/planning/ch13.pdf

[4] S. Wirtensohn, “Automotive Control Systems Lecture 2: Path Planning / Trajectory Generation,” Slide Script, HTWG, Konstanz, Summer Semester 2021. Accessed: Aug. 9 2021. [Online]. Available: https://moodle.htwg-konstanz.de/moodle/pluginfile.php/355460/mod_resource/content/1/AutoCtrlSys_Lecture_2_Path%20Planning%20with%20Notes.pdf

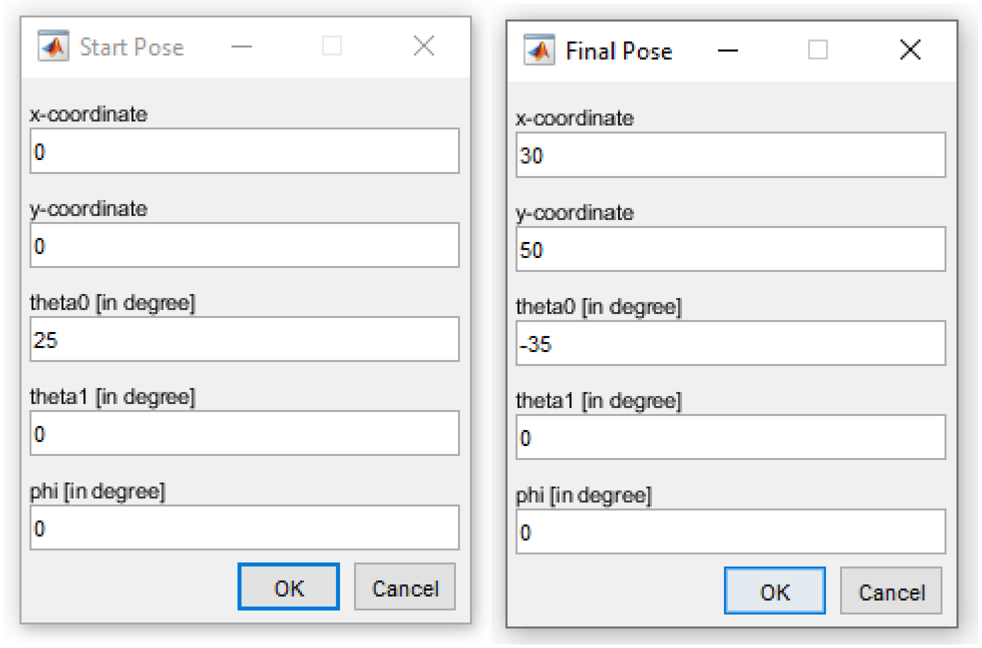
[5] S. Wirtensohn, “Automotive Control Systems Lecture 4: Feedback Linearization,” pp. 1–22, Summer Semester 2021. [Online]. Available: https://moodle.htwg-konstanz.de/moodle/pluginfile.php/361074/mod_resource/content/1/AutoCtrlSys_Lecture_4_Feedback_Linearization%20with%20Notes.pdf

[6] T. Raff, “Vorlesung Regelungstechnik: Kapitel 4: Modellbasierter Reglerentwurf im Bild- und Frequenzbereich,” Folienskript, HTWG, Konstanz, 2020. Accessed: Jul. 30 2021.

[7] S. Wirtensohn, “Automotive Control Systems Lecture 3: Differential Flatness,” Slide Script, HTWG, Konstanz, Summer Semester 2021. Accessed: Jul. 31 2021. [Online]. Available: https://moodle.htwg-konstanz.de/moodle/pluginfile.php/362545/mod_resource/content/1/AutoCtrlSys_Lecture_3_Flat_Systems%20with%20Notes.pdf

Attachment

Figure 16: 1st Test – Varying Start and End Orientation Angle of Truck



Start Pose	
x-coordinate	0
y-coordinate	0
theta0 [in degree]	25
theta1 [in degree]	0
phi [in degree]	0

Final Pose	
x-coordinate	30
y-coordinate	50
theta0 [in degree]	-35
theta1 [in degree]	0
phi [in degree]	0

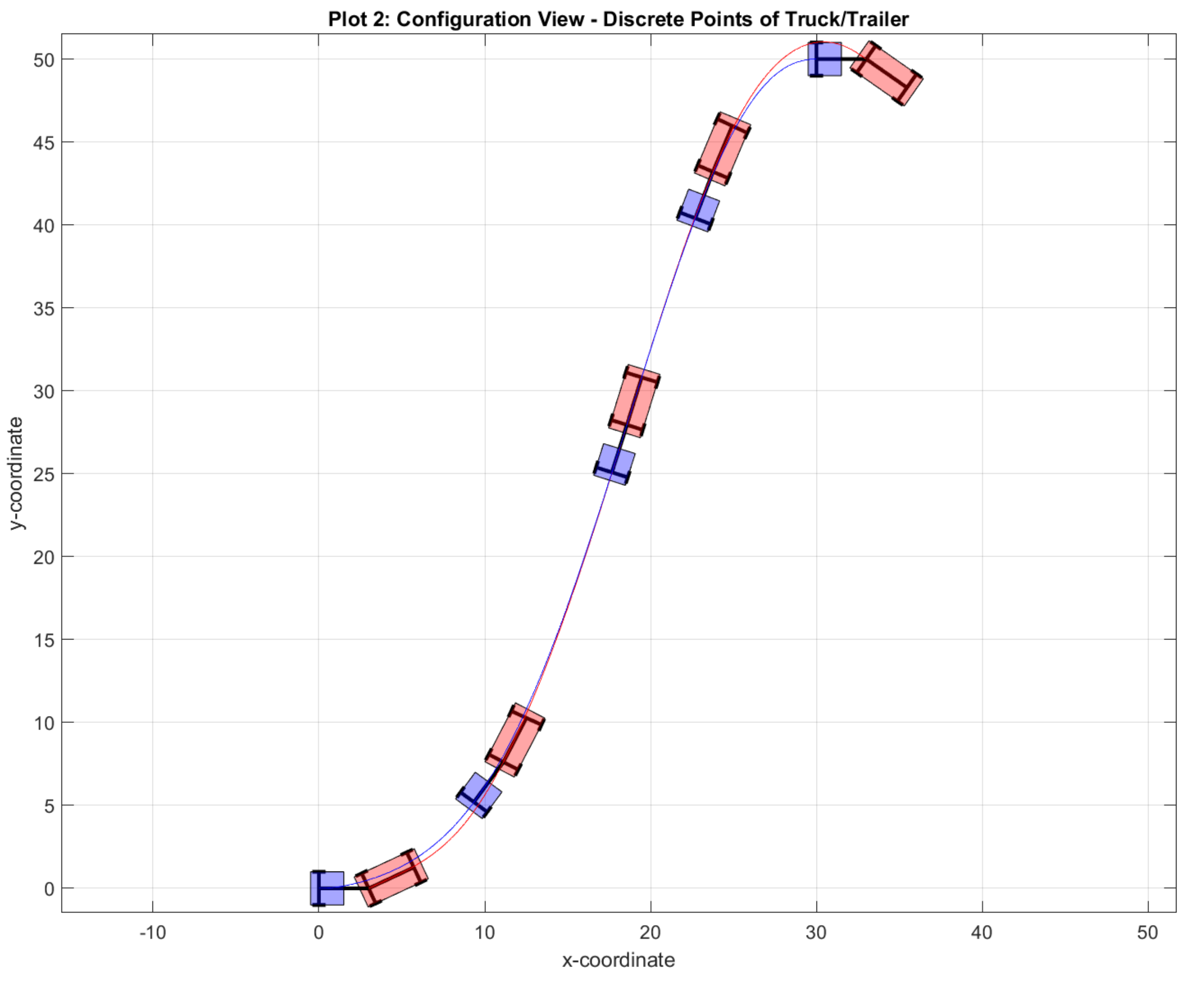
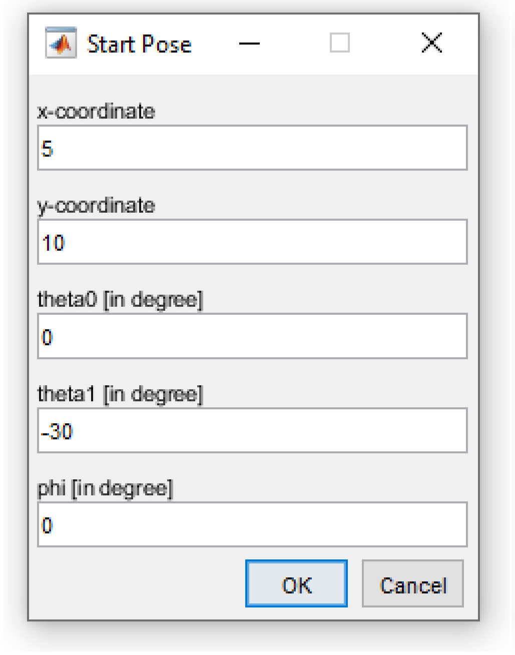


Figure 17: 2nd Test – Varying Start and End Orientation Angle of Trailer


Start Pose

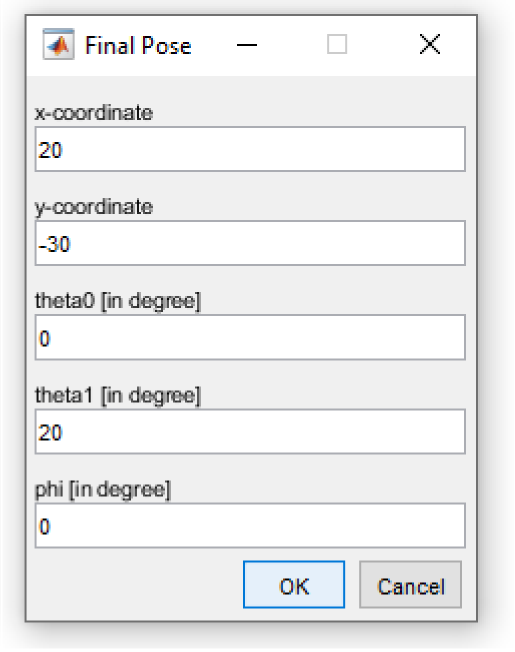
x-coordinate
5

y-coordinate
10

theta0 [in degree]
0

theta1 [in degree]
-30

phi [in degree]
0


Final Pose

x-coordinate
20

y-coordinate
-30

theta0 [in degree]
0

theta1 [in degree]
20

phi [in degree]
0

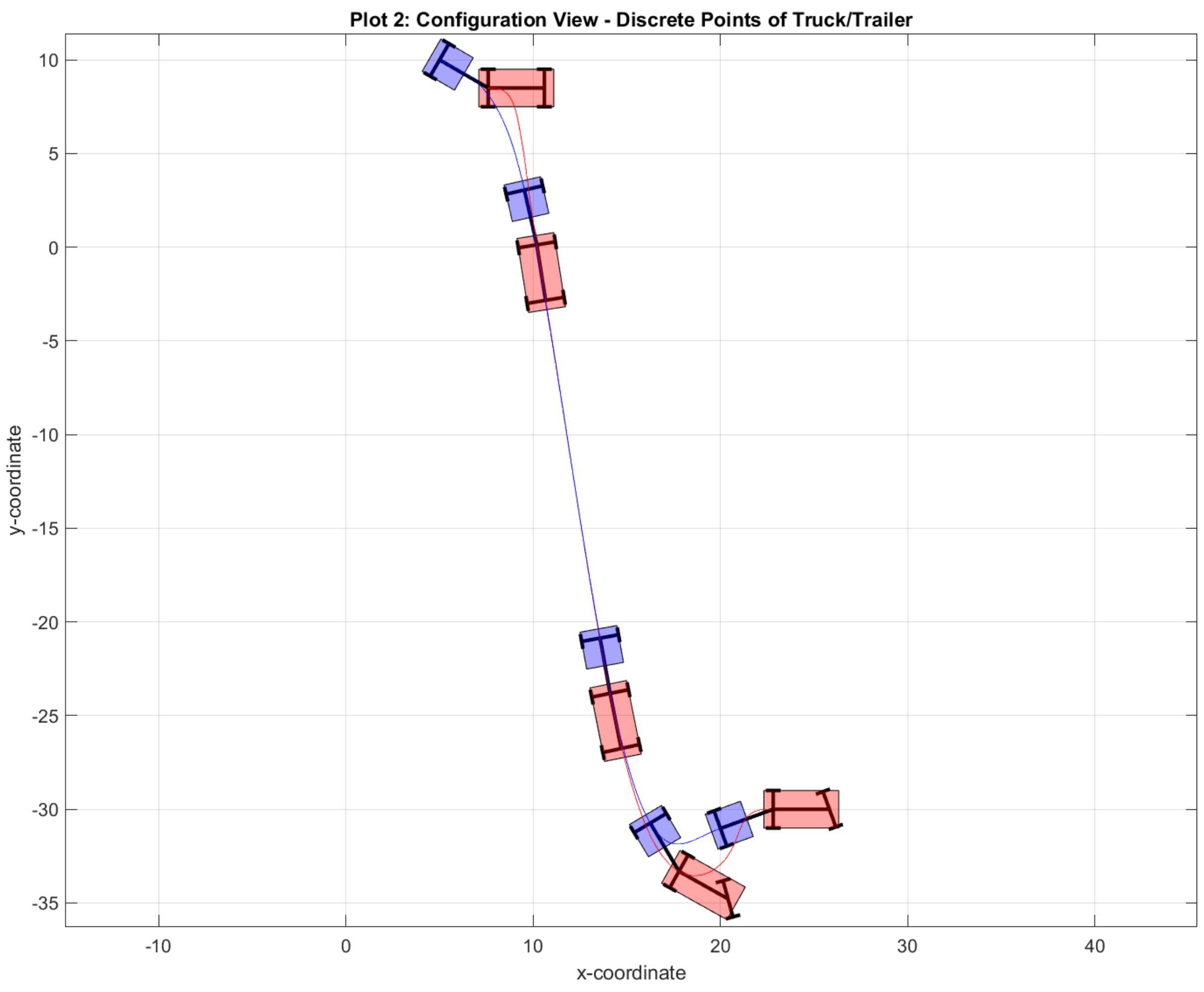
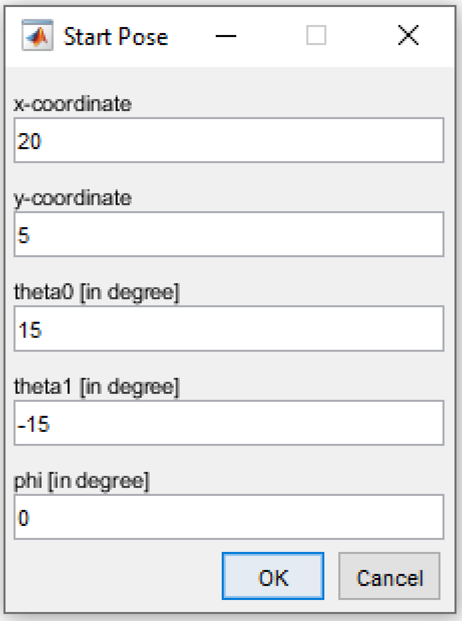
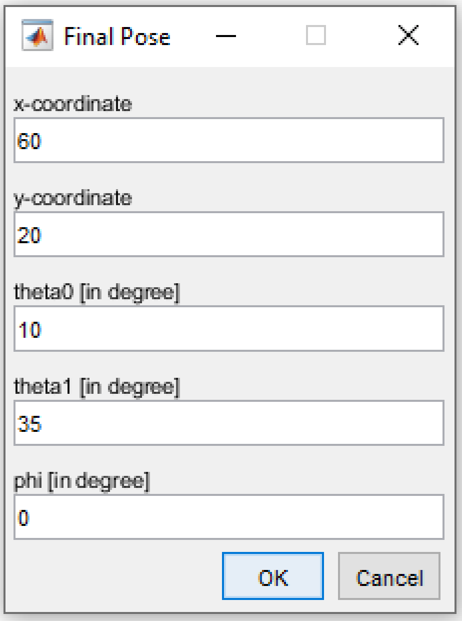


Figure 18: 3rd Test – Varying Start and End Orientation Angle Combined



x-coordinate
20
y-coordinate
5
theta0 [in degree]
15
theta1 [in degree]
-15
phi [in degree]
0



x-coordinate
60
y-coordinate
20
theta0 [in degree]
10
theta1 [in degree]
35
phi [in degree]
0

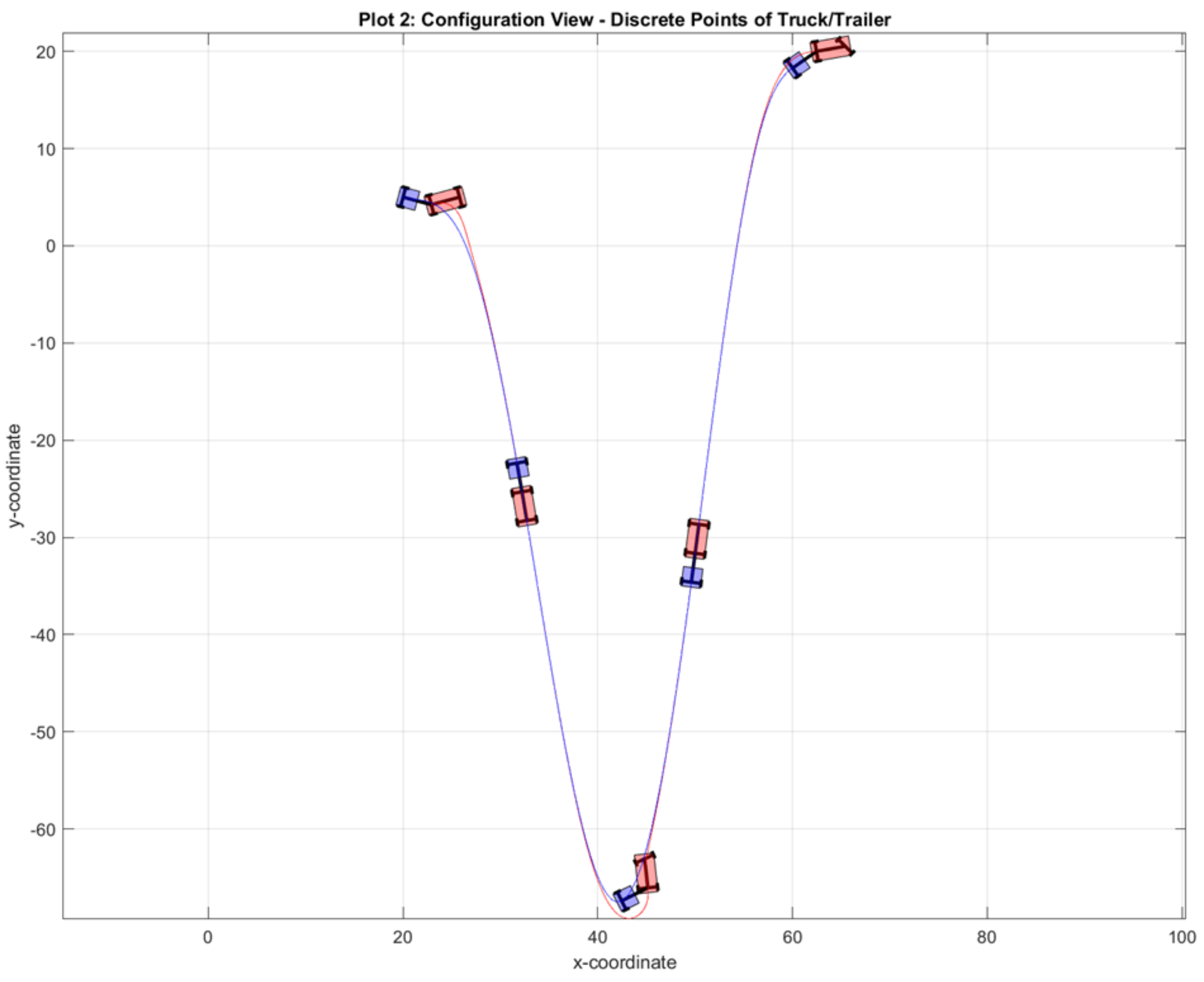
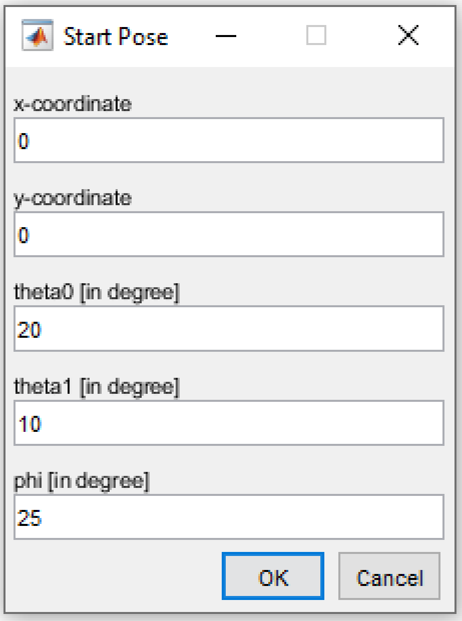
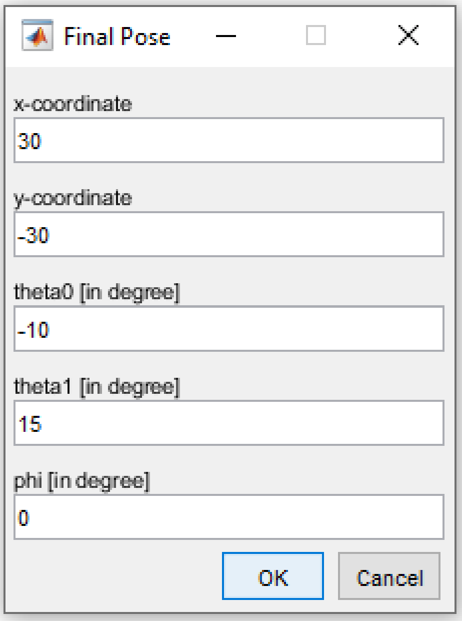


Figure 19: 4th Test – Varying Start Steering Angle and Orientation Angles



Start Pose

x-coordinate
0
y-coordinate
0
theta0 [in degree]
20
theta1 [in degree]
10
phi [in degree]
25



Final Pose

x-coordinate
30
y-coordinate
-30
theta0 [in degree]
-10
theta1 [in degree]
15
phi [in degree]
0

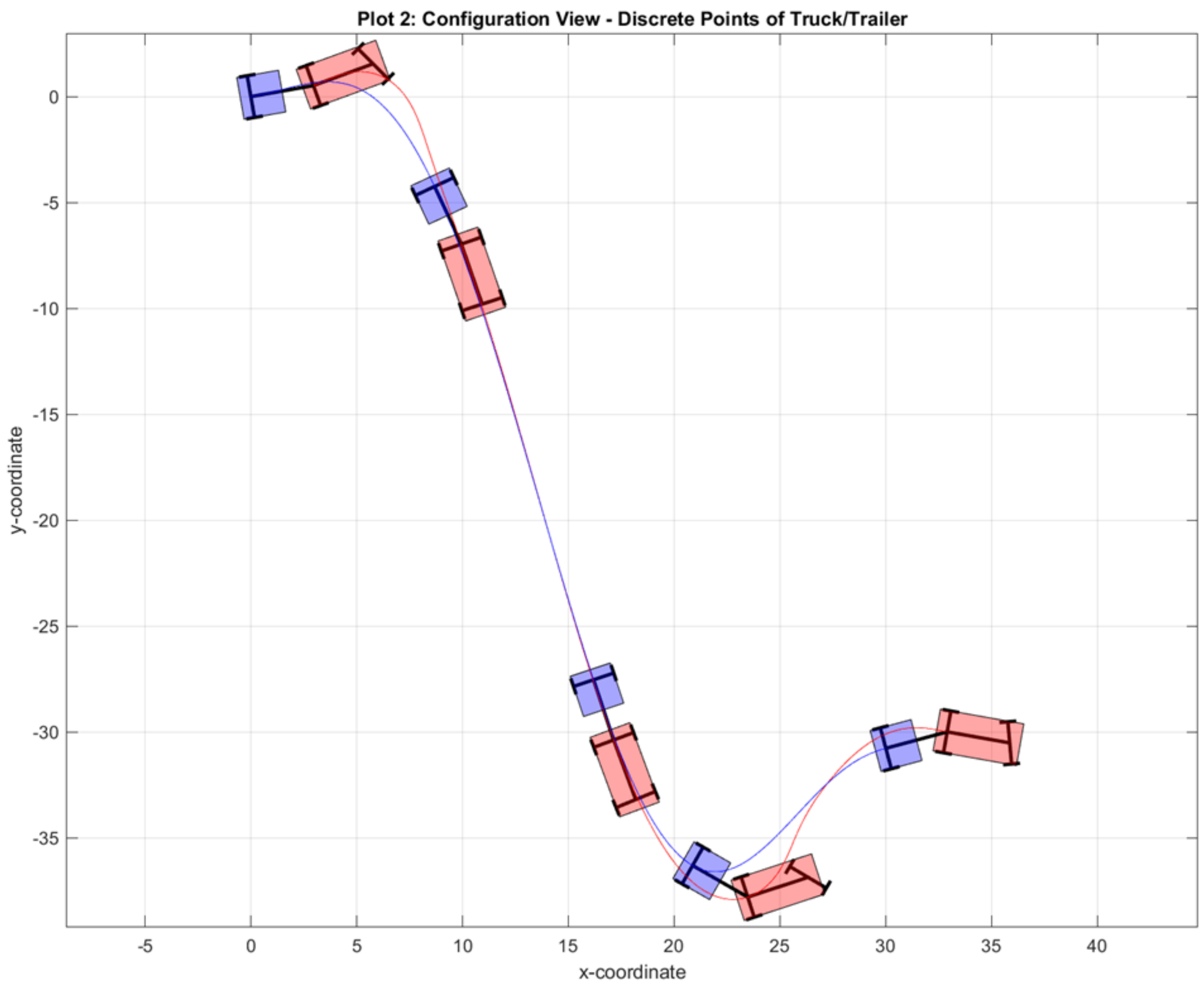
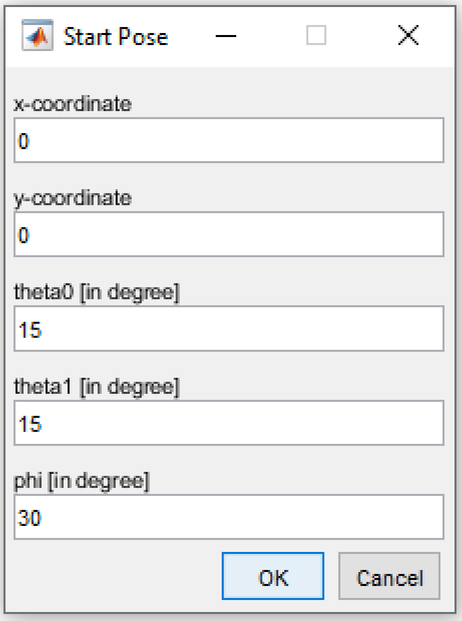


Figure 20: 5th Test – Long Route/Distance between Start and Final Pose



Start Pose

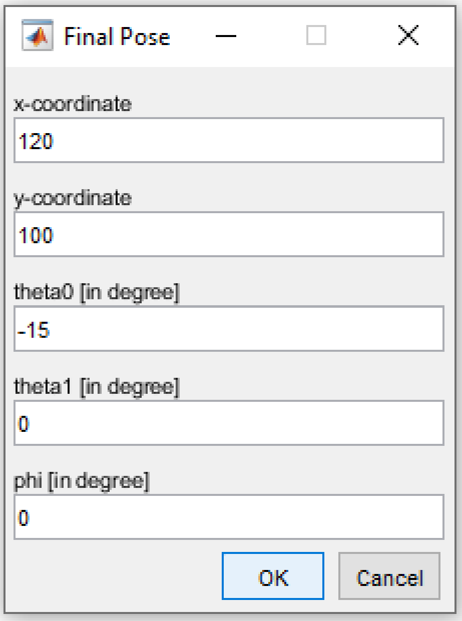
x-coordinate
0

y-coordinate
0

theta0 [in degree]
15

theta1 [in degree]
15

phi [in degree]
30



Final Pose

x-coordinate
120

y-coordinate
100

theta0 [in degree]
-15

theta1 [in degree]
0

phi [in degree]
0

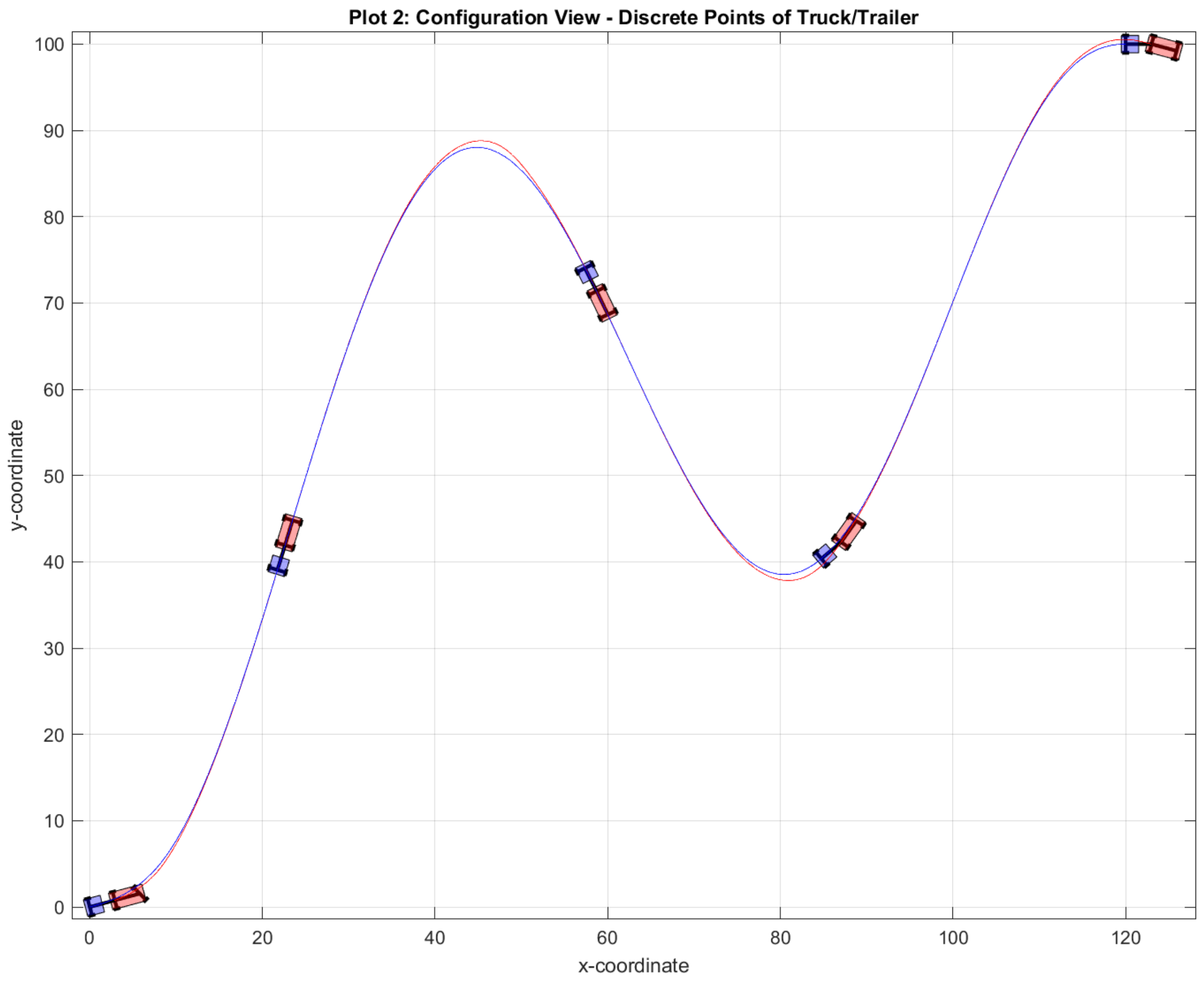
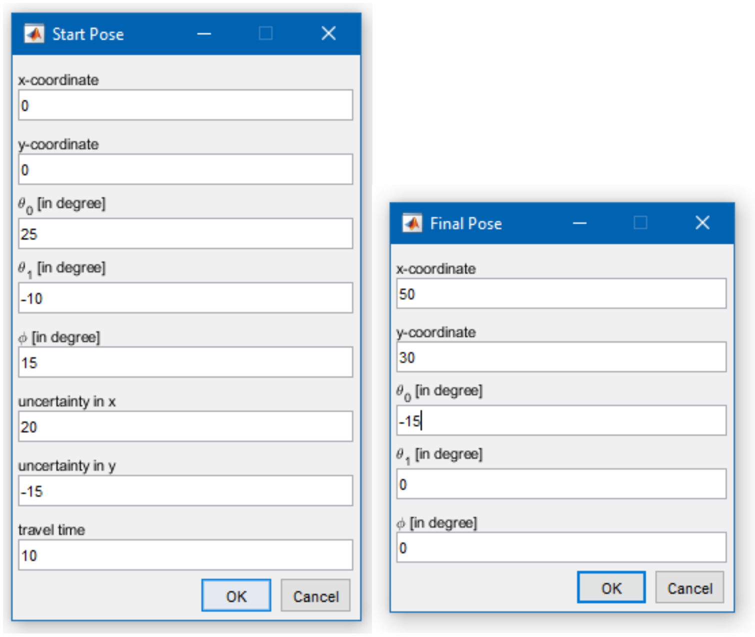


Figure 21: Random Deviation of Start Pose (Position Uncertainty)

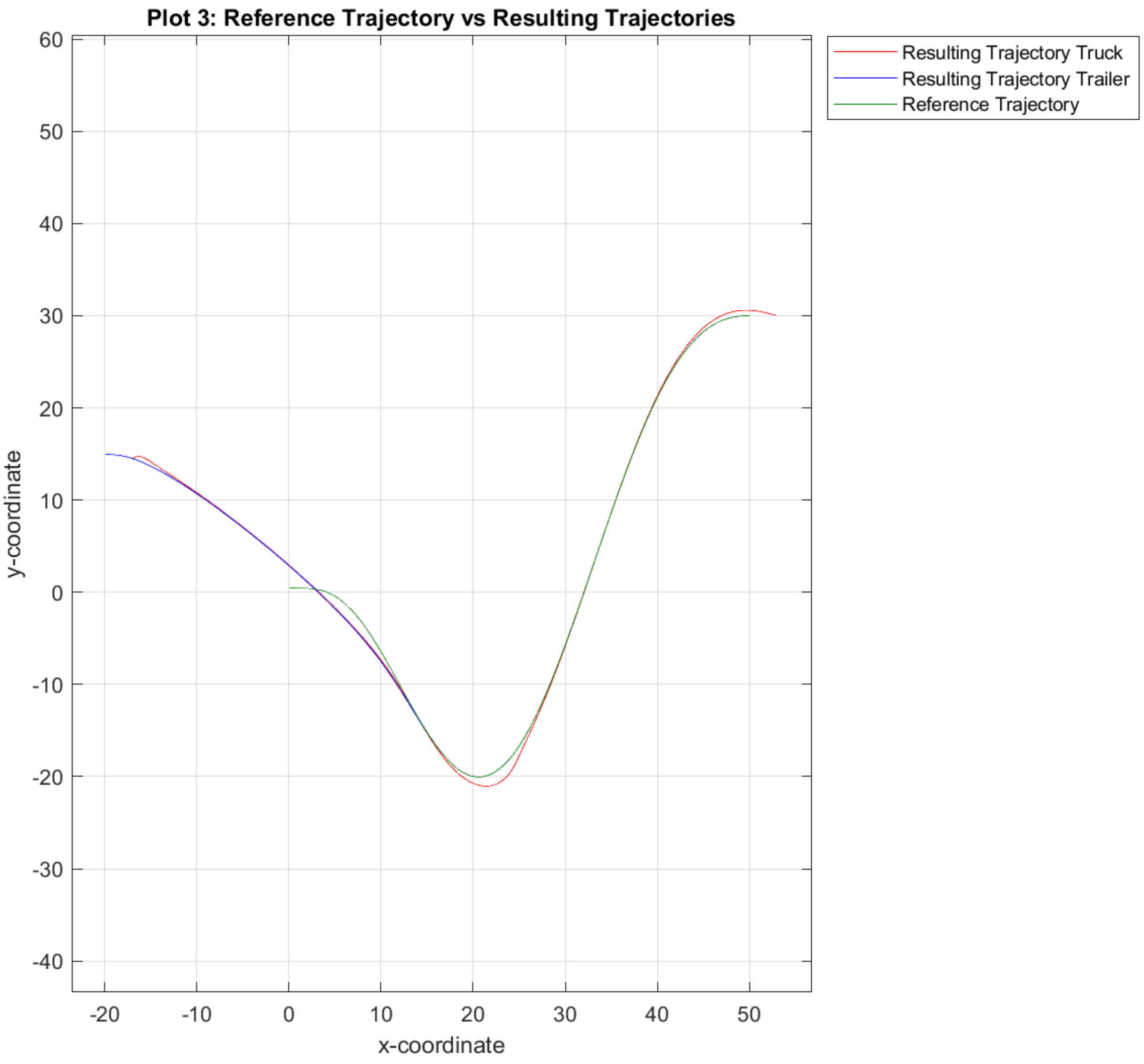


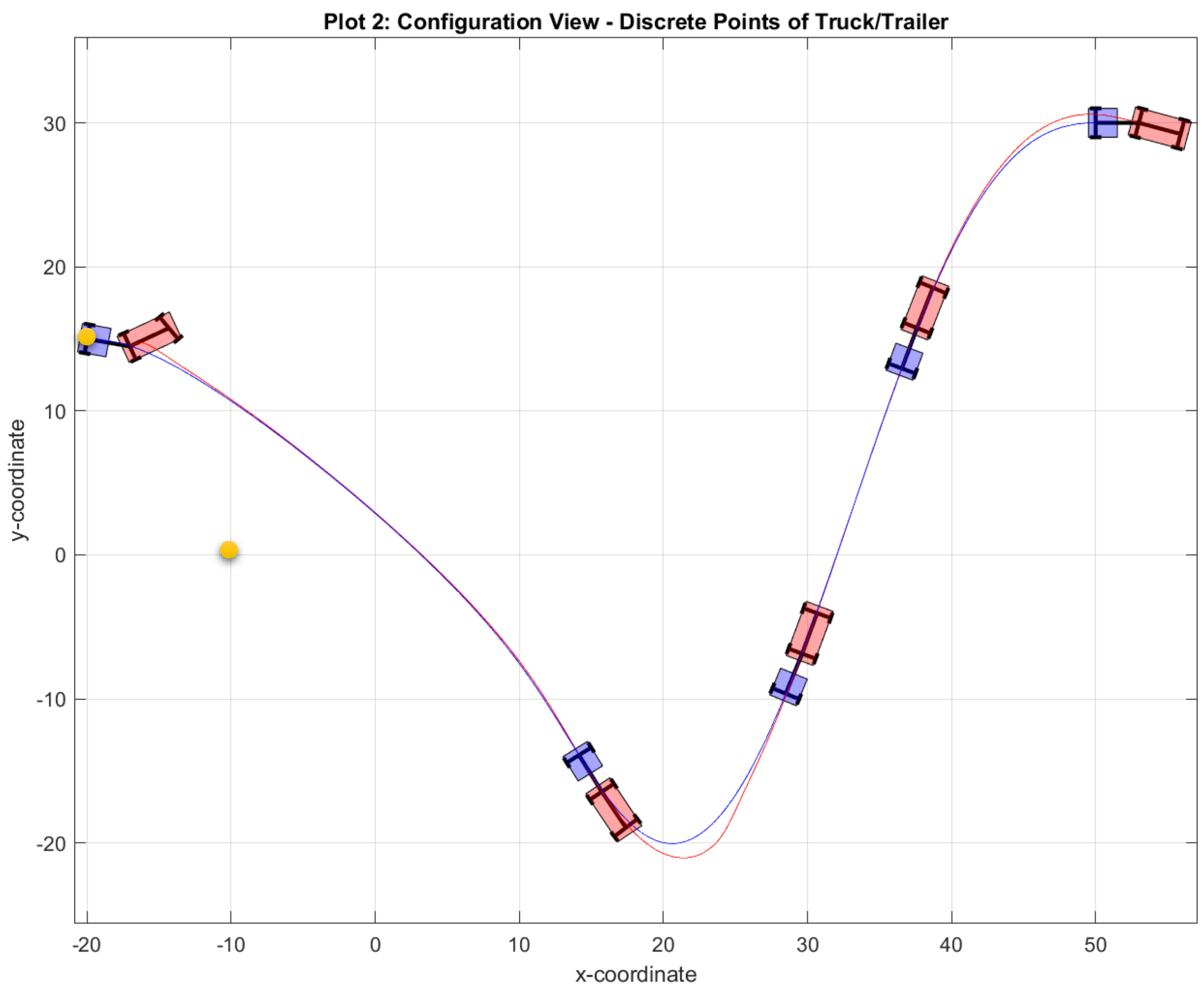
Start Pose

x-coordinate	0
y-coordinate	0
θ_0 [in degree]	25
θ_1 [in degree]	-10
ϕ [in degree]	15
uncertainty in x	20
uncertainty in y	-15
travel time	10

Final Pose

x-coordinate	50
y-coordinate	30
θ_0 [in degree]	-15
θ_1 [in degree]	0
ϕ [in degree]	0




Plot 2: Configuration View - Discrete Points of Truck/Trailer
x-coordinate
y-coordinate
-20
-10
0
10
20
30
40
50
-20
-10
0
10
20
30