Contents

Chapter 1

Trajectory Planning and Control

1.1 Introduction

In the SAE AutoDrive Challenge I [1], Society of Automotive Engineers (SAE) and General Motors (GM) set forth guidelines for vehicle dynamics metrics that should be obeyed by Autonomous Vehicle (AV)s to ensure comfortable and safe urban driving. These guidelines include limits on longitudinal acceleration and jerk, as well as lateral acceleration in the vehicle's body frame. The guidelines are to be adhered to as the AV accomplishes the Dynamic Driving Task (DDT). Generally, the DDT can be seen as progressing towards a goal state in a feasible and efficient manner. In the context of urban driving, feasible means without collision and while obeying traffic rules, and efficient means operating near the speed limit.

1.1.1 Motivation

There are several papers in the literature that address the real-time obstacle avoidance problem using optimal control techniques. However, all have drawbacks that make them ill-suited for the problem presented above. The authors of [11] apply a nonlinear Model Predictive Control (MPC) method, similar to the one presented in Section 1.3, to the obstacle avoidance problem. However, the controller must be instructed by the perception system on which direction to avoid the obstacle (left or right). This assumption is unsatisfactory because the directional decision in itself needs to consider the dynamics of the vehicle to know if a maneuver to the left or right of the obstacle is most optimal.

The authors of [20] take an interesting approach towards integrated obstacle avoidance by representing obstacles as a potential field in the cost formulation. However, this paper, as well as [4], [23], and [13] have the issue that the avoidance maneuver controller (which they separate from the lower level actuator controller) is based on a particle representation of the vehicle. Therefore, these approaches have the same issue as using a higher level planner: the planned states generated by the lower fidelity planning model may not be dynamically feasible and therefore are unsafe.

In [12], a cubic polynomial is utilized to describe the lateral deviation from the reference line. Similarly, the authors of [7] and [14] use a sigmoid function. However, none of these papers provide an analysis of why these functions were chosen in the context of their Optimal Control Problem (OCP) objective function. In contrast, the scheme proposed in this chapter avoids obstacles in a manner that is consistent with its OCP objective function because the obstacles themselves are part of the OCP formulation.

In [10], an MPC-based technique for short-term path planning among multiple moving objects is presented. Experiments are carried out in simulation and present promising results in a variety of scenarios. However, the work depends on a linearized bicycle model assumption, which may not be a valid assumption during avoidance maneuvers that require large road wheel angles. The authors also assume that all obstacle information is known before system start-up and that the operating environment is a straight road, which makes their approach not applicable to a real-world setting.

In [6] the authors take a Dynamic Programming (DP) approach to optimal obstacle avoidance. The paper claims that with a prediction horizon of 50 their method has "high accuracy and fast computing time, which can satisfy the requirements for application of autonomous vehicle driving on real roads." [6]. However, the authors do not present any quantitative data relating to computation time of the DP solution. Since DP solutions are usually prohibitively slower than other optimal control techniques (especially with a large prediction horizon), these claims should be viewed with skepticism until full experimental data is released.

In [19] the authors implement a path following MPC controller similar to the one presented here and deploy it to a VW Golf VII. However, no constraints for static obstacles, dynamic obstacles, nor passenger comfort were designed. This strictly limits the operational domain of the deployment to the simple scenario of following a reference line.

In [9] the authors use hard constraints to avoid moving obstacles, similar to the work presented here. However, the application of [9] is high speed ground vehicles in unstructured environments. The authors realize the benefit of a single-level design where motion planning and reference tracking are combined into a single optimization program. The au-

thors also compare hard-constraint vs. soft-constraint (e.g. potential fields) formulations of the obstacle avoiding optimization problem. The authors find that hard constraints outperform soft constraints in terms of obstacle avoidance performance and optimization time. However, even with the hard constraint formulation the optimization time was 45 seconds, putting the method well out of the domain of real-time implementation. In contrast, our work presented in this chapter operates in real-time.

In [15] the authors operate in the same context of [9], without a reference in an unstructured environment. However, in [15] the obstacles are assumed to be static. Additionally, the authors only consider constraints that enforce vehicle dynamic safety, and not passenger comfort. Additionally, like [7], the controller decides on a speed profile, which then needs to be further regulated into torque commands, which can introduce additional predictive model inaccuracy.

1.1.2 Contributions

This chapter addresses the issues presented above by introducing a novel OCP formulation of the DDT and an accompanying MPC solution. The main areas of improvement in this regard are:

1. The system model is designed to align closely with physical platforms and to allow for computation of comfort metrics and constraints, enabling the OCP to be formulated to abide by the SAE guidelines for passenger comfort.

2. Constraints are added to the OCP formulation that guarantee feasibility of control actions with respect to road boundaries, static obstacles, and dynamic vehicles. In this way, trajectory planning can be executed simultaneously within the controller, and the final control actions are guaranteed to be dynamically feasible.

3. The controller operates in real-time by employing a novel parallel-solver method which uses a warm-started "online" solver for real-time performance, while employing a parallel "exploration" solver which serves to break out of warm-started local minima.

The remainder of this chapter is organized as follows: Section 1.2 presents necessary background information on routing, reference line parameterization, and obstacle representations. Section 1.3 presents the OCP formulation, including the system model, objective function, road boundary constraints, and obstacle constraints. Section 1.4 discusses the

MPC solution to the OCP, including how the OCP is expressed as a Non-Linear Program (NLP), and the novel parallel-solver method for real-time performance. Experimental results are then presented and analyzed in Section 1.5. Conclusions are discussed in Section 1.6.

1.2 Background

In order to properly understand the motion planning scheme in which the proposed controller fits, some background information is necessary. We first note the information given to the motion planning scheme:

1. A Lanelet2 map which describes lane geometry and topology.
2. The desired goal position on the map.
3. A discretized occupancy grid representation of the static environment.
4. Non-ego dynamic vehicle tracks that describe the position and longitudinal velocity of the vehicle.

1.2.1 Reference Spline Creation

In order to transform this information into reference signals that the controller can follow, we first need to find a lane-level route that describes how the vehicle can proceed from its current state to the goal state. This step is called global planning and is done using Dijkstra's search algorithm over the routing graph constructed from the Lanelet2 map [8].

However, a sequence of lanes is not yet an admissible reference signal for the controller. In order to generate a continuous and differentiable reference signal, a spline is fit to the centers of the lanes that make up the global route. In brief, CasADi's Interior Point Optimizer (IPOPT) [22] is used to find the spline coefficients that minimize squared error to the lane centers, see [7] for details.

An important note is that a spline has a limited capacity to express the complex lane geometry which may appear over a long route, incurring high squared error. To solve this problem, a sliding window is applied over the entire lane route, allowing the spline to accurately capture the simpler geometry of the lanes in the local window.

The same method is applied to generate splines for the left and right boundaries of the driveable surface. The lines that describes the driveable surface are determined using the routing rules stored in the Lanelet2 map.

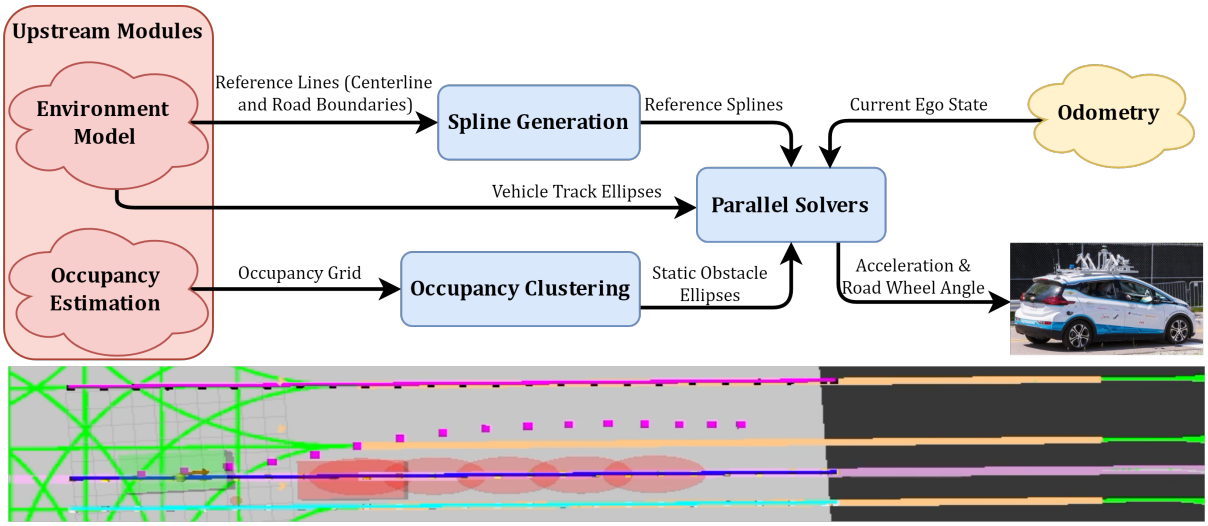

Figure 1.1: Graphical representation of the inputs giving to the motion planning scheme and the controller. The light pink line is the global route constructed from the lane centers. The blue line on top of it is the local window reference spline. The pink line to the left and the aqua blue line to the right are the drivable surface boundaries. The red ellipses show the predicted trajectory of the dynamic vehicle that the ego-vehicle is overtaking.

1.2.2 Occupied Space Representations

Aside from the centerline and road boundary splines, a reference for occupied space over time is also necessary to allow the controller to plan feasible trajectories. Two inputs are used to calculate this reference: An occupancy grid, and localized vehicle tracks from the tracker and environment model. The goal is to have a set of obstacles, and for each obstacle have a state trajectory that covers the MPC predictive horizon. The obstacle state representation used is a 2D ellipse, which has 5 degrees of freedom: X and Y of the centroid, RX (longitudinal radius), RY (lateral radius) and θ (yaw). To transform an occupancy grid into a set of obstacle trajectories, the grid is first filtered to only contain information of obstacles that are on the driveable surface. Then, a region growing algorithm is run with a neighborhood radius of four cells to cluster the obstacles in the grid (see Alg. 1). Lastly,

Algorithm 1 Region Grow

Input: occupancy *grid*, row r, col c, radius *rad*, visited 2D array *visited*
Output: *[N, 2] array*, N is the number of cells in cluster

```
Initialize seedList ← [(r,c)]
Initialize cluster ← [(r,c)]
Initialize neighbors ← [n for c in product(range(-rad, rad + 1), repeat=2]) if n[0] != 0
or c[1] != 0]
while len(seedList) > 0 do
    Initialize currCell ← seedList.pop(0)
    visited[currCell[0]][currCell[1]] = 1
    for neigh ∈ neighbors do
        Initialize tR ← currCell[0] + neigh[0]
        Initialize tC ← currCell[1] + neigh[1]
        if tmpR < 0 or tmpC < 0 or tmpR ≥ grid.height or tmpC ≥ grid.width then
            continue
        end if
        if grid[tR][tC] == 1 and visited[tR][tC] == 0 then
            seedList.append((tR, tC))
            cluster.append((tR, tC))
        end if
    end for
end while
return cluster
```

the Minimum Volume Enclosing Ellipse (MVEE) approximate iterative algorithm [17] [21] is used to fit an ellipse to each cluster found by the region growing algorithm (see Alg. 2). The result of this process is illustrated in Fig. 1.2. Since the occupancy grid contains information about the static portions of the environment only, the occupancy ellipses are the same for each step in the MPC horizon.

Algorithm 2 Grid To Ellipses

Input: occupancy *grid*, neighbor radius *rad*
Output: *[N, 5] array*, *N* is the number of ellipses in grid

```
Initialize ells ← []
Initialize visited ← zeros_like(grid)
for r in range(grid.height) do
    for c in range(grid.width) do
        if visited[r][c] == 0 and grid[r][c] == 1 then
            cluster = RegionGrow(grid, r, c, rad, visited)
            C, rx, ry, theta = MVEE(cluster)
            ells.append(C.x, C.y, rx, ry, theta)
        end if
    end for
end for
return ells
```

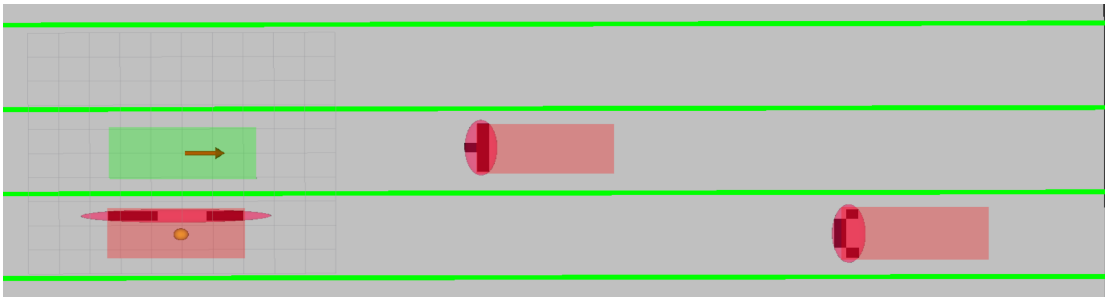

Figure 1.2: Illustration of the result of the occupancy clustering procedure described in Section 1.2.2. The discrete occupied cells are shown in black, the resulting clustered ellipses are shown in red, and the actual physical obstacles are shown in transparent red.

To generate the ellipse trajectories for the dynamic vehicle tracks, the environment model module (see Chapter **??**) first localizes each track in the lanelet it geometrically occupies. Then, based on the estimated longitudinal velocity (assumed constant) from the tracker, and the geometry of the upcoming lanelets, the ellipses are generated following Alg. 3. This simple prediction scheme can be expanded to a more complex learning based technique, but this simple method suffices for controller design.

In summary, the information given to the controller is (as shown in Fig. 1.1):

1. A spline (denoted S_{ref}) that describes a local window of the global route.

Algorithm 3 Predicted Dynamic Vehicle Trajectory

Input: *envModel*, *track*, MPC step S (in seconds), MPC horizon M
Output: *[M, 5] array*, vehicle ellipse trajectory prediction

```
path = envModel.localizeVehicle(track)
startDist = toArcCoordinates(path, track.pos)
traj = []
for i in range(M) do
    lonDist = startDist + i * S * track.lonVel
    p1 = interpPointAtDistance(path, lonDist)
    p2 = interpPointAtDistance(path, lonDist + 0.1)
    theta = atan2(p2.y - p1.y, p2.x - p1.x)
    traj.append({ p1.x, p1.y, track.rx, track.ry, theta })
end for
return traj
```

2. Two splines (denoted S_{left} and S_{right}) which describe a local left and right drivable surface boundary.

3. A set of predicted trajectory ellipses (denoted O, where O_1 is the set of obstacles at step 1, and $O_{1,1}$ is the prediction for obstacle 1 at the 1st step in the MPC horizon).

1.3 Optimal Control Problem Formulation

This section describes the plant system to be controlled, the predictive model used in the proposed MPC controller, the objective function, and the constraints.

1.3.1 System and Predictive Models

Input Variables: The plant speed is controlled by applying a longitudinal acceleration command (denoted u_a), and yaw is controlled via commanded road wheel angle (denoted u_δ). Additionally, an input variable referred to as *path progress* u_ξ controls how far along the reference spline the next state will progress. The usage of this path progress input in the context of path following is explained below.

State Variables: The state vector was chosen in order to maintain the vehicle dynamic metrics that are necessary to compute the constraints presented below. Specifically, the

state vector maintains the vehicle's inertial pose tuple $[x_X, x_Y, x_\psi]^T$ as well as longitudinal and forward body frame velocities (denoted x_{v_x}, and $x_{v_{fwd}}$, respectively). Additionally, a state variable referred to as the *path integral* x_Ξ is defined as the integration of the path progress input mentioned above. The path integral variable keeps track of how far along the reference route the ego is, and its functionality in terms of the objective function is explained below.

Output Variables: The system output variables are the odometry signals generated by the Inertial Navigation System (INS). The INS directly measures inertial pose, as well as longitudinal and lateral velocities and accelerations in the body frame. The current path integral is estimated using a discretized linear search over the reference spline's parameter range.

Predictive Model: To describe the evolution of the system over time, a nonlinear model is necessary due to the nonlinear behavior of vehicular systems under large road wheel angles [18]. In an urban driving setting, maneuvers that require large road wheel angles are common. This is in contrast with a highway driving setting that may only require a few degrees of road wheel angle and hence operate inside of a linear dynamics range. To this end, a nonlinear kinematic bicycle model is employed, as illustrated in Fig. 1.3 and described by Eqs. 1.1, 1.2:

$$\dot{x}_X = x_{v_{fwd}} cos(x_\psi + \beta) \tag{1.1a}$$

$$\dot{x}_Y = x_{v_{fwd}} sin(x_\psi + \beta) \tag{1.1b}$$

$$\dot{x}_\psi = \frac{x_{v_{fwd}} cos(\beta) tan(u_\delta)}{l_f + l_r} \tag{1.1c}$$

$$\dot{x}_{v_x} = u_a \tag{1.1d}$$

$$\dot{x}_\Xi = u_\xi \tag{1.1e}$$

where

$$\beta = \arctan(\frac{l_r tan(u_\delta)}{l_f + l_r}) \tag{1.2}$$

is the slip angle of the vehicle at its CoG.

In [18] the usage of the kinematic bicycle model as a predictive model for MPC is examined in depth, and the authors come to the conclusion that "the kinematic bicycle model is a good modeling for low-speed vehicles but seems to not be precise enough for vehicles at high speed" [18]. Since the target platform has a speed restriction of 25MPH,

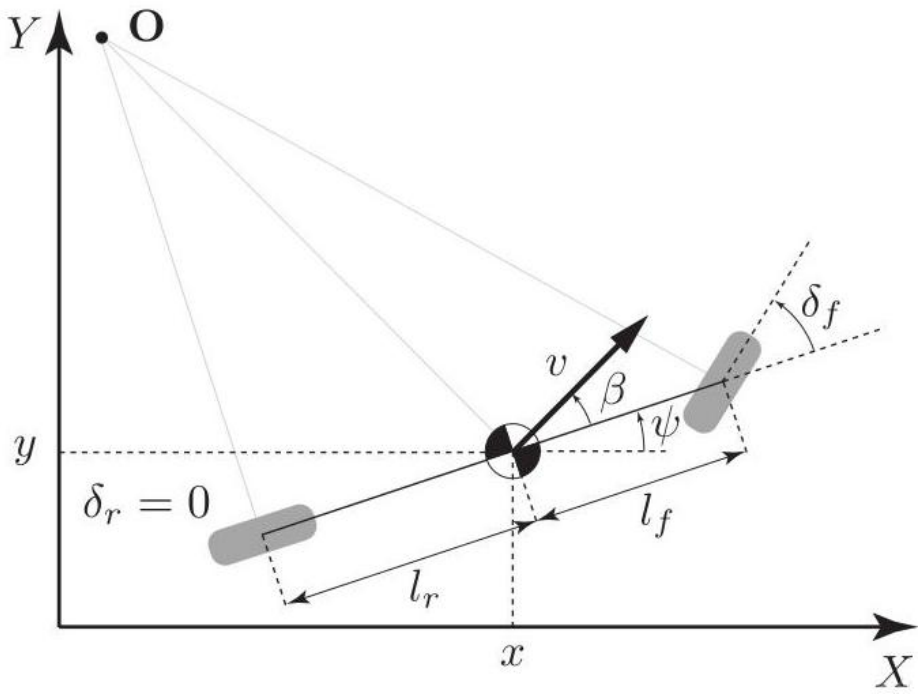

Figure 1.3: Schematic of the kinematic bicycle model representation of a vehicle. In the figure, δ_f is the commanded road wheel angle, l_f and l_r are the front and rear wheel bases, v is the direction of travel, and β is the slip angle of the vehicle at its Center of Gravity (CoG).

the kinematic bicycle model is expected to capture the dynamics of the simulated high fidelity vehicle model well enough for the presented MPC implementation.

In summary, a nonlinear kinematic bicycle model was chosen as it is a simple predictive model that captures the nonlinear system dynamics for low-speed urban navigation [18], and thus is the best trade-off between NLP computation time and accurate prediction.

1.3.2 Objective Function

The terms included in the OCP objective function are:

1. Reference position error. Calculated as squared error between x_X, x_Y and S_{ref} at arclength x_Ξ.

2. The distance of the path parameter to route completion. Calculated as $(1 - x_{\Xi})$.

Giving the final objective function:

$$J = \sum_{k=1}^{N} \quad \left[\begin{bmatrix} x_X^k \\ x_Y^k \end{bmatrix} - \begin{bmatrix} x_{ref}^k \\ y_{ref}^k \end{bmatrix} \right]_2 + Q(1 - x_{\Xi}^k) \tag{1.3}$$

where Q is a positive number and $x_{ref}^k = S_{ref}^x(x_{\Xi}^k)$. Thus, the higher the Q value, the more incentivized the ego will be to complete the route, even at the cost of deviating from the reference route if needed. The outcome of the tuning process of Q was two separate weighting settings, one for nominal path following and another setting for when the controller is avoiding an obstacle. In the path following setting, a $Q = 1$ weight is used which ensures accurate following of the reference spline, whereas in the obstacle avoidance setting (when the state is within 5 meters of an obstacle), $Q = 1000$ is used to encourage the vehicle to deviate from the reference if necessary to avoid the obstacle.

These two simple terms embody the non-safety constraint portions of the DDT: Drive close to the desired route, and make efficient progress towards route completion. The rest of the DDT (the safety constraints) is implemented as hard constraints on the NLP and are covered below.

1.3.3 Constraints

All vehicular systems are non-holonomic and therefore an OCP aiming to control such a system must take the non-holonomic constraints into consideration. These non-holonomic constraints include the system model presented above, as well as further physical constraints like maximum steering angle. There are also constraints enforced for the comfort of the passenger as per the SAE guidelines, which include limits on longitudinal and lateral acceleration as well as jerk. Most importantly, there are the constraints that enforce the feasibility of the predicted vehicle trajectory: The vehicle must stay in free space, defined as inside the road boundaries and not in collision with any obstacle. A complete list of the OCP constraints can be found in Table 1.1.

Road Boundary Enforcement

There are multiple ways to enforce that a 2D point $[x_X, x_Y]'$, i.e. vehicle position, must be inside a set of two splines. The method we found worked best was a "sidedness" test,

Table 1.1: Optimal Control Problem Constraints

Variable(s)	Constraint	Type	Reasoning	Calculation
$\boldsymbol{x}_{k+1}$	$f(\boldsymbol{x}_k, \boldsymbol{u}_k)$	Non-Holonomic	Dynamics of system described via constraints as per multiple shooting [5]	RK4. See Section 1.3.1 for system used to model f.
u_δ	$-\frac{\pi}{4} \leq u_\delta \leq \frac{\pi}{4}$	Non-Holonomic	Physical range of wheel shaft	N/A, obtained from fact sheet
x_{v_x}	$0 \leq x_{v_x} \leq v_{MAX}$	Legal	Vehicle cannot exceed speed limit	N/A, supplied by Lanelet2 map
$\dot{x}_{v_y}$	$-3.5 \leq \dot{x}_{v_y} \leq 3.5$	Comfort	SAE bounds on lateral acceleration	From [16]: $\dot{x}_{v_y} = \frac{x_{v_x}^2 u_\delta}{l_f + l_r}$
u_a	$-3.5 \leq u_a \leq 3.5$	Comfort	SAE bounds on longitudinal acceleration	N/A, input variable
$\dot{u}_a$	$-10 \leq \dot{u}_a \leq 15$	Comfort	SAE bounds on longitudinal jerk	$\dot{u}_a = \frac{u_a^k - u_a^{k-1}}{T}$ Where T is the sample time
$[x_X, x_Y]'$	$[x_X, x_Y]' \in S^{drive}$	Feasibility	The vehicle must be on the driveable surface	See Sec 1.3.3
$[x_X, x_Y]'$	$[x_X, x_Y]' \notin \chi^{occ}$	Feasibility	The vehicle can only occupy free space	See Sec 1.3.3
x_Ξ	$0 \leq x_\Xi \leq 1$	Feasibility	Vehicle required to stop at end of route	N/A, state variable

enforcing that $[x_X, x_Y]'$ is to the right of S_{left}, and to the left of S_{right}. The first step in the formulation is to obtain a 2D bound vector $[b_0, b_1]'$ that we enforce the sidedness constraint with respect to. b_0 is calculated as the boundary spline at arclength x_Ξ and b_1 is calculated as the boundary spline at arclength $x_\Xi + la$ where la is a small look-ahead (e.g. 0.01). Then, the (left) sidedness constraint can be enforced as in Eq. 1.4 (right constraint is < 0).

$$(x_X - b_0.x) * (b_1.y - b_0.y) - (x_Y - b_0.y) * (b_1.x - b_0.x) > 0 \tag{1.4}$$

Obstacle Avoidance Enforcement

For each obstacle, and for each step in the MPC horizon, the vehicle's footprint cannot intersect with any obstacle ellipse. Eq. 1.5 *InEllipse(p, el)* implements the basic operation needed for such a constraint, testing if a 2D point p is inside an ellipse el (assuming that the point and the ellips are in the same reference frame, a detail that is worked out in Alg. 4).

$$\frac{(p.x - el.x)^2}{el.rx} + \frac{(p.y - el.y)^2}{el.ry} < 1 \tag{1.5}$$

Algorithm 4 Obstacle Avoidance

Input: Symbolic set of ellipse trajectories O, symbolic state trajectory S
Output: For each matching O_i, S_i in the horizon, enforce that S_i does not conflict with any object in O_i

for $O_i \in O$, $S_i \in S$ **do**
  for $O_{i,j} \in O_i$ **do**
    th = $O_{i,j}.theta$
    rot = [cos(th), sin(th); -sin(th), cos(th)]
    el = {rot * $O_{i,j}.cent$, $O_{i,j}.rx$, $O_{i,j}.ry$}
    for $S_{i,j} \in expandFootprint(S_i)$ **do**
      ENFORCE(! InEllipse(rot * $S_{i,j}.pos$, el))
    end for
  end for
end for

Now all that is left is to call *InEllipse(p, el)* for each ellipse we want to avoid at each step in the MPC horizon. Alg. 4 implements such a routine, where the $expandFootprint$ subroutine simply returns 6 points around the border of the vehicle's footprint, and $ENFORCE$ enforces the enclosed constraint in the IPOPT solver. Note that in Alg. 4 we are creating $6 * |S| * |O_i|$ constraints ($|S|$ is the MPC horizon, $|O_i|$ is the number of obstacles). This can be on the order of 100s of constraints for a nominal $|S| = 15$ and $|O_i| < 10$.

1.3.4 Conversion to Torque Commands

In the CARLA simulator longitudinal commands can be applied directly, but in production they must be converted to torque commands to send to the Controlled Area Network Bus (CAN Bus). Alg. 5 presents how that conversion is done, where $Cd = 0.3$, $Cr = 0.02$, $A = 2.0$, $\rho = 1.2$, $m = 2500$, $r = 0.4$, $rat = 7.05$, $g = 9.81$.

1.4 Model Predictive Control Solution

To solve the OCP, Nonlinear Model Predictive Control (NMPC) was used. This decision was due to: (1) The high number of constraints used to enforce non-holonomic system dynamics, comfort, and feasibility, and (2) The nonlinear dynamics of the system in an urban setting where large front wheel angles may be necessary to navigate.

Algorithm 5 Longitudinal Acceleration to Torque

Input: Desired longitudinal *accel*, current vehicle *speed*, drag coefficient Cd, rolling resistance Cr, frontal area A, air density ρ, vehicle mass m, wheel radius r, final gear ratio rat, gravity force g

Output: Torque value to command

1: $AeroCons = 0.5 * \rho * A * Cd$
2: $RollCons = m * g * Cr$
3: $ResForce = speed * (AeroCons + RollCons)$
4: $AccForce = accel * m$
5: **return** $\frac{r}{rat} * (AccForce + ResForce)$

In order to transform the OCP into a NLP, temporal discretization was applied to the system dynamics over a finite prediction horizon ($N = 15$). Specifically, the Fourth Order Runge-Kutta (RK4) method was used with a time step of $T = 0.25s$. As proposed by Bock and Plitt in [5], multiple shooting was used to enforce the dynamics of the system using constraints, reducing the nonlinearity of the objective function especially in the latter steps of the prediction horizon.

To solve the NLP, the interior point optimizer (IPOPT) [22] from the CasADi [3] software package was used. Computational concerns regarding real-time IPOPT solutions are discussed below. With the NLP solution obtained, receding horizon control is applied.

1.4.1 Dealing With a Variable Number of Objects

One nuance of obstacle avoidance via optimal control is the fact that the NLP must be specified at startup time. Coupled with the fact that a different number of objects in the scene leads to a different number of constraints and thus a different NLP formulation, it would seem required to know the number of obstacles in the scene *a priori*. To get around this impractical assumption, a collection of NLP solvers are initialized at system startup, each with a different number of constraints according to the number of obstacles the solver expects. Then, at system runtime, a solver is delegated to based on the number of obstacles actually observed in the scene, which may change over time. This procedure is outlined in Alg. 6.

Algorithm 6 Variable Obstacle Solver Delegation

Input: Max number of obstacles max_obs
Output: Delegation to solvers that handles variable numbers of obstacles

```
controller_map = {} {At system startup}
for i ∈ range(max_obs) do
    solver_map[i] = Solver(i)
end for
while true {At runtime} do
    obs_set = PERCEIVE_SCENE()
    sol = solver_map[len(obs_set)](obs_set)
    EXECUTE_SOL(sol)
end while
```

1.4.2 Warm-Starting, Escaping Local Minima via Parallel Solvers

Real-time performance of the controller is defined as the NLP solver operating faster than the sampling time of the controller (0.25s), i.e. greater than 4Hz. In order to achieve real-time performance, warm-starting the iterative IPOPT solver with a "decent" solution is essential, allowing the solver to reach an acceptable solution after only a small number of iterations. The warm-start used at time t is normally the solution obtained by the solver at time $t-1$, following from the assumption that the parameters of the optimization change little from one solve to the next. However, over time this assumption may no longer be true, and the solver may become stuck in a series of warm-started local minima.

An example is the controller attempting to avoid an obstacle. To avoid an obstacle a large change from the previous solution is needed; diverting from the reference line and then re-joining it is a very different solution than just proceeding along the reference. Naive warm-starting does not allow for such a divergence from the previous solution. Given a warm-start that follows the reference line, and a small number of solver iterations, IPOPT will never find a solution that diverges around an obstacle (see Section 1.5.4 and Fig. 1.8 for an illustration of this failure case).

A method of "breaking out" of this local minima is required, a way to search for a more global minima given more IPOPT iterations and a less biased warm-start. Our novel method is to run a parallel solver that consumes the same NLP as the original solver (which we will refer to from now on as the "online" solver), but does not use warm-starting (initial decision variable assignments are all zero) and is allowed 10x the number of IPOPT

Algorithm 7 Warm Start Breakout

Input: Number of online IPOPT iterations o_iters, number of exploration iterations e_iters

Output: A NLP solution scheme that runs in real-time while not getting stuck in local minima

$o\_solver = Solver(o\_iters)$ {At system startup}
$e\_solver = Solver(e\_iters)$ {At system startup}
while true {At runtime} **do**
  $solver\_params = PERCEIVE\_SCENE()$
  $warm\_start = o\_solver.sol$
  if $e\_solver.has\_sol$ **then**
    $e\_sol = e\_solver.sol$
    if $e\_sol.cost < warm\_start.cost$ **then**
      $warm\_start = e\_solver.sol$
    end if
    $e\_solver(solver\_params, \vec{0})$ {Exploration solver runs asynchronously}
  end if
  $sol = o\_solver(solver\_params, warm\_start)$
  $EXECUTE\_SOL(sol)$
end while

iterations. We will refer to this new solver as the "exploration" solver.

These two solvers run asynchronously of each other, the online solver is used to control the vehicle as normal, and the exploration solver is used to simply "suggest" new warm-starts that the online solver could use. Whenever the online solver is about to perform a new solve, it polls the exploration solver for its most recent solve. If the polled solution has a lower cost than the cost of the previous online solution, the exploration solution is used for the online solver's warm-start instead of the previous online solution. This process is outlined in Alg. 7.

The result, viewed from a unified perspective, is a controller than can operate in real-time using a warm-starting scheme but also does not get stuck in local minima (see Section 1.5.4 for results). Of course this comes at the cost doubling the computation load of the controller, we are now running two IPOPT solvers instead of one. However each of those processes only run on a single core, so in total we only take two out of the 64 cores on the WATonomous computation platform for control. Note that this parallelized approach can be scaled up to an arbitrary number of exploring solvers, each running a potentially

different warm-starting scheme and number of IPOPT iterations.

1.5 Experiments

1.5.1 Reference Line Following

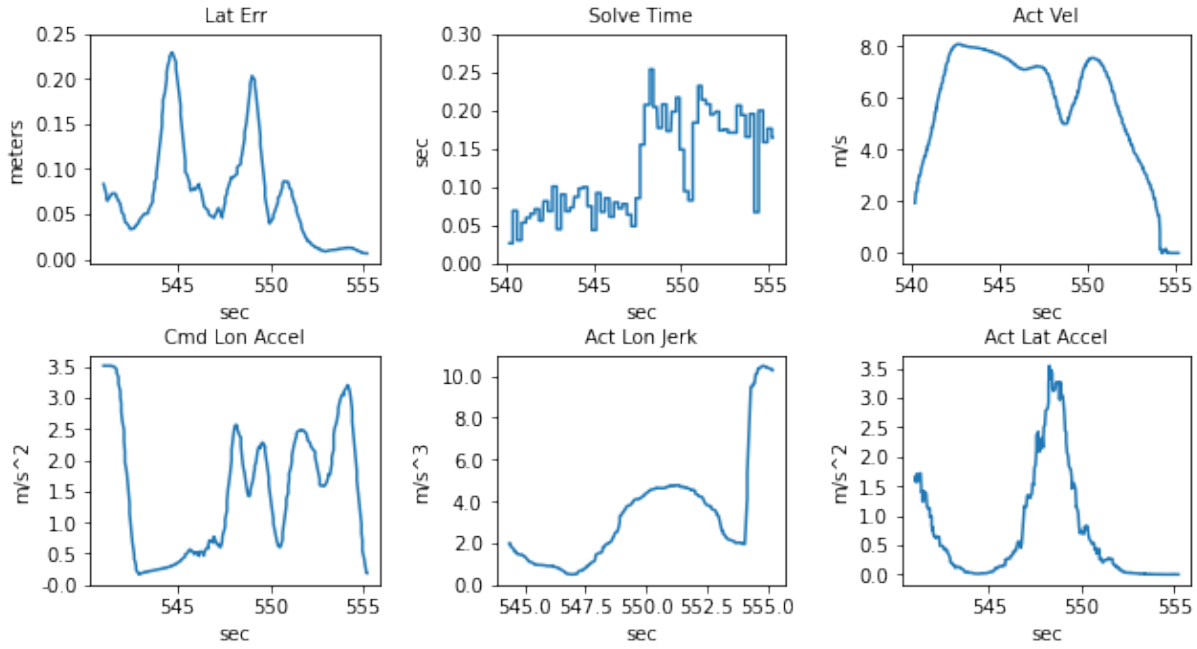

Figure 1.4: Quantitative reference line tracking performance of the nominal controller.

In this experiment, the only reference we have to follow is the lane center as the ego vehicle completes a right turn at a 4-way intersection. The desired behavior is to have low lateral error from S_{ref}, keep the NLP solution time within the 0.25s sampling time, stay as close to the 8m/s speed limit as possible, and obey the legal and comfort constraints from Table 1.1. The results below show two controller settings: The nominal controller described above, and a no-comfort-constraint controller which has the comfort constraints from Table 1.1 disabled. The purpose of the two settings is to prove the effectiveness of the OCP design in regulating passenger comfort.

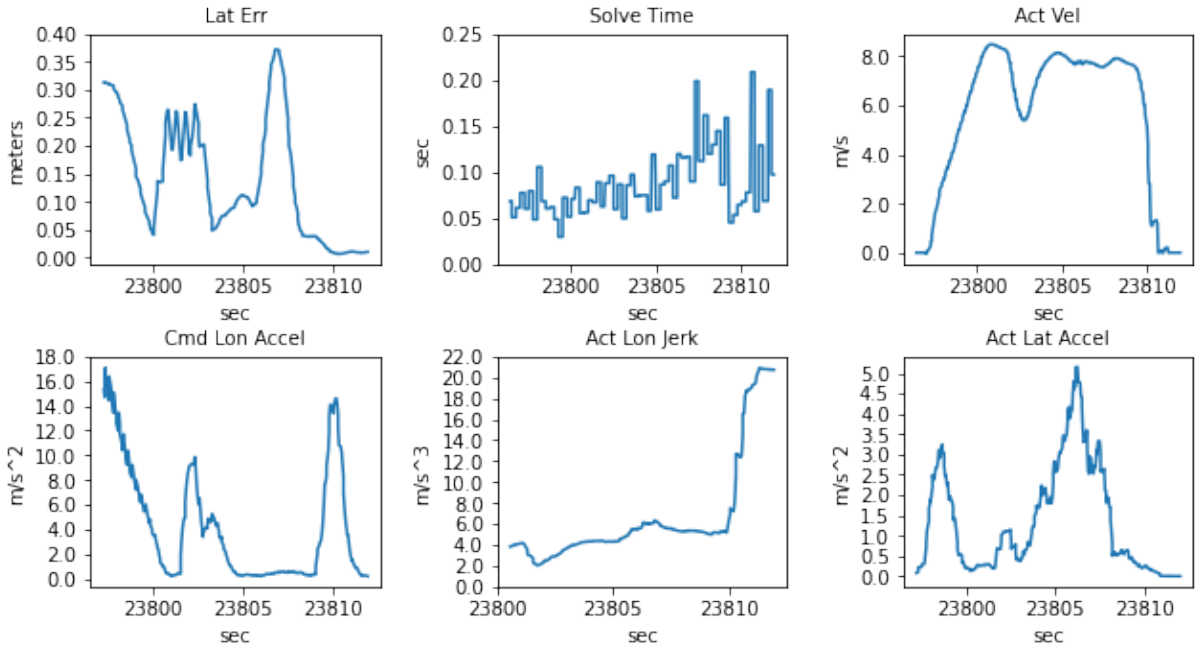

Figure 1.5: Quantitative reference line tracking performance of the no-comfort-constraint controller.

Qualitative Results

The qualitative results are presented via a video of the completion of the test scenario using the nominal controller, which can be found here: https://youtu.be/2Gk3XQKlk38.

Quantitative Results

Fig. 1.4 shows the quantitative results for the nominal controller. As seen in the figure, the lateral deviation stays below 25cm from the reference, well within the lane boundaries. The solution time of the NLP also stays below the sampling time of the controller, and the velocity stays close to 8m/s, except when necessary to slow down while taking the right turn to avoid passenger discomfort around t=548. The comfort metrics are all within the desired comfort range outlined by SAE.

Fig. 1.5 shows the quantitative results for the no-comfort-constraint controller. As seen in the figure, the lateral deviation increases (due to more jerky steering), but is still within the lane boundaries. The solution time and the velocity profile are both comparable to

the nominal controller. However, in the velocity profile there is no drop in velocity around t=23805, in contrast with what we saw in the nominal controller. Taking the corner with such a high speed violates the lateral acceleration constraint, deeming the drive uncomfortable by SAE standards. Furthermore, when first accelerating and decelerating, both the longitudinal acceleration and jerk constraints are violated, which is further evidence of an uncomfortable ride for the passenger.

Overall, it is clear that introducing the comfort constraints to the OCP design forces the controller to slow down for sharp turns to keep lateral acceleration within the desired bounds, and to accelerate and decelerate more smoothly to ensure passenger comfort.

1.5.2 Static Obstacle Avoidance

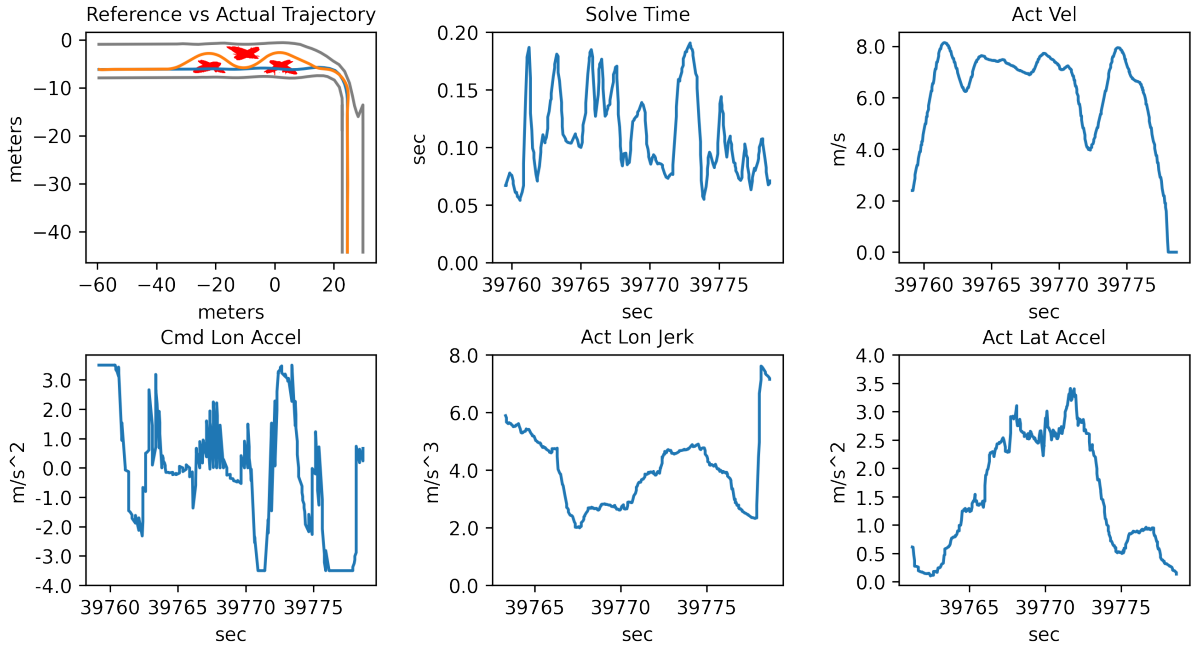

Figure 1.6: Quantitative static obstacle avoidance performance of the nominal controller.

In this experiment, a new reference is introduced: static obstacles. The desired behavior is the same as in Section 1.5.1, with the addition of not hitting any static obstacles. The scenario used to test this behavior is similar to that of Section 1.5.1, a straight road followed by a right turn. However an obstacle course of three static obstacles was added

to the straight road. The results below show the controller in its nominal setting avoiding the course of static obstacles using the formulation discussed in Section 1.3.3.

Qualitative Results

The qualitative results are presented via a video of the completion of the test scenario using the nominal controller, which can be found here: https://youtu.be/T83wHpZDfd0. As desired, there are no collisions with the static obstacles on the course.

Quantitative Results

Fig. 1.6 shows the quantitative results for the nominal controller. As shown in the top left subplot, the reference (in blue) is tracked well when the controller is not performing an obstacle avoidance maneuver. The controller also successfully avoids the static obstacles (in red). Furthermore, the controller performs the avoidance maneuvers while staying inside the driveable surface bounds (shown in grey). Lastly, all the desired driving behaviors from Section 1.5.1 (namely the SAE comfort metrics) are also still obeyed.

1.5.3 Road Boundary

To show the effectiveness of the road boundary constraints discussed in Section 1.3.3, they were removed in this experiment and the resulting vehicle behavior is discussed below. The same static obstacle course testing scenario was used to examine the results. The desired behavior is the same as in Section 1.5.2.

Qualitative Results

The qualitative results are presented via a video of the attempted completion of the static obstacle course over two attempts by the no-road-boundary controller, which can be found here: https://youtu.be/YqpCCIzRw28. As seen in the video, without the constraints the controller may choose to avoid the obstacles in an illegal manner, leaving the road surface completely. This video demonstrate the necessity and effectiveness of the road boundary constraints discussed in Section 1.3.3.

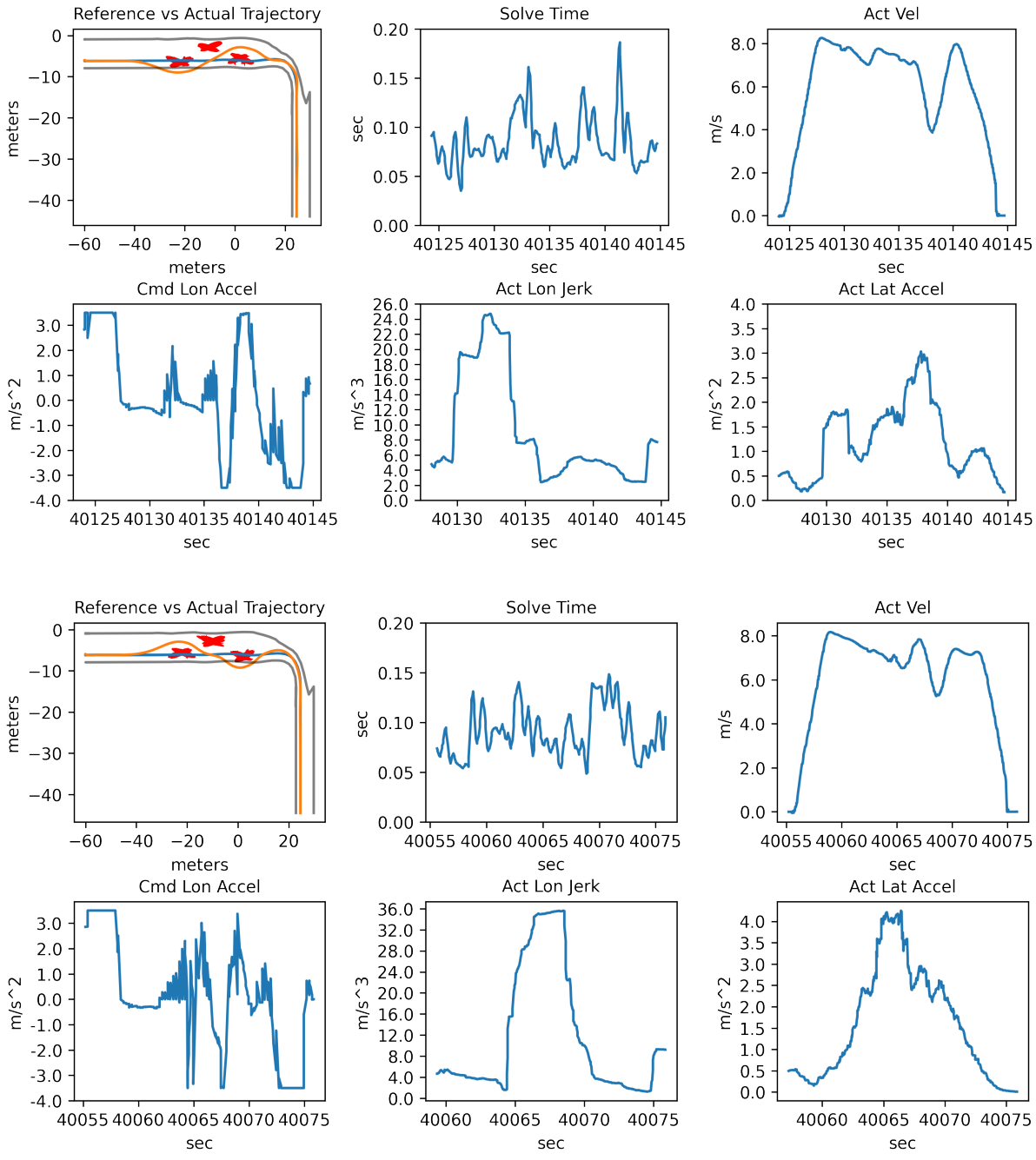

Figure 1.7: Quantitative static obstacle avoidance performance of the no-road-boundary controller over two runs.

Quantitative Results

A quantitative analysis of the no-road-boundary controller is shown in Fig. 1.7. Without the road boundary constraints the controller may arbitrarily decide to avoid the obstacle on either side, possibly violating the road boundary traffic rule (see top left plots, where the orange ego trajectories violate the grey road boundaries). These plots, when compared against the plot in Fig. 1.6, shows that the road boundary constraints specified in Section 1.3.3 are necessary for correct driving behavior.

1.5.4 Parallel Solver

To show the effectiveness of the parallel solver technique discussed in Section 1.4.2, it was removed (only the online solver is used) in this experiment. The same static obstacle course testing scenario was used to examine the results. The desired behavior is the same as in Section 1.5.2. The resulting vehicle behavior is discussed below.

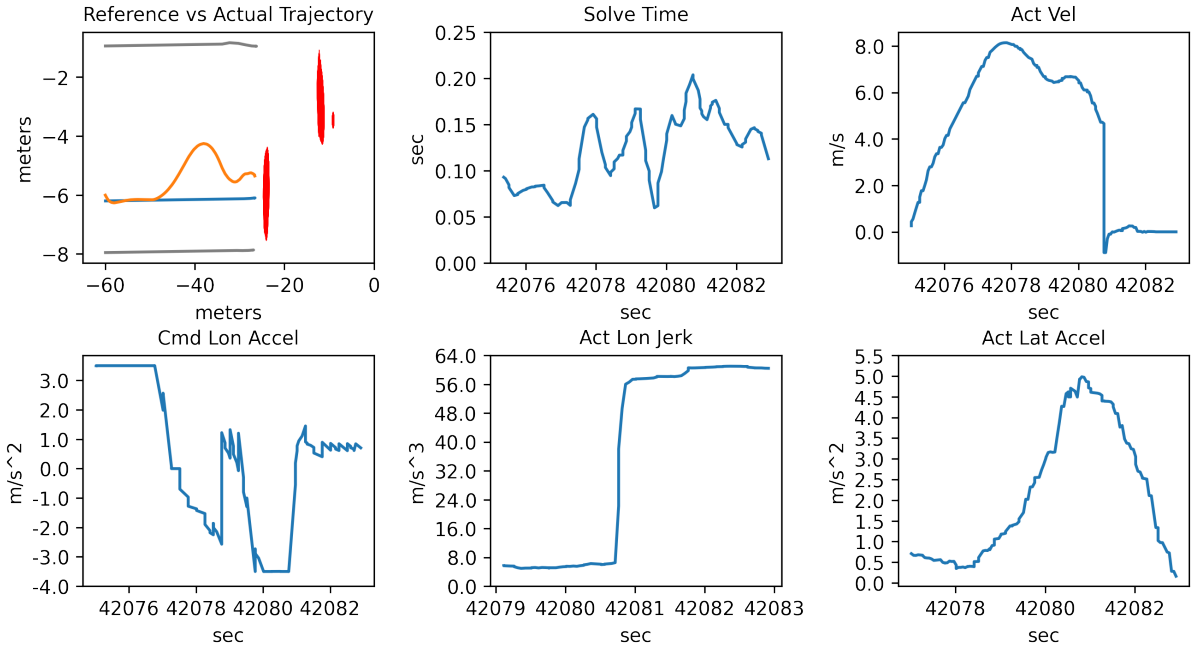

Figure 1.8: Quantitative static obstacle avoidance performance of the no-parallel-solver controller.

Qualitative Results

Qualitative results of the no-parallel-solver controller failing to complete the scenario are shown at: https://youtu.be/cLFCUO3-lMY. As seen in the video, without the parallel solver the controller easily gets stuck in a local warm-starting minima and fails to avoid even the first obstacle. This video demonstrate the necessity and effectiveness of the parallel solver technique discussed in Section 1.4.2 to avoid such local minima traps when using warm-starting for real-time performance.

Quantitative Results

A quantitative analysis of the no-parallel-solver controller is shown in Fig. 1.8. As seen in the reference plot (top left) of the figure, without the parallel solver, the controller cannot find a trajectory around even the first static obstacle on the course. The failure observed here compared to the success in Section 1.5.2 is due to the missing exploration solver discussed in Section 1.4.2. In the no-parallel-solver controller setting the controller lacks the ability to escape local-minima traps caused by the warm-starting method necessary for a real-time implementation.

1.5.5 Dynamic Obstacle Avoidance

In this experiment, a new reference is introduced: dynamic obstacles. The scenario used is a straight road with the ego obeying a speed limit of 15m/s. In front of the ego, there is a target vehicle traveling at 7.5m/s. In order to make efficient progress along the route, the ego vehicle should overtake the target vehicle in a comfortable and legal manner.

Qualitative Results

The qualitative results are presented via a video of the completion of the test scenario using the nominal controller, which can be found here: https://youtu.be/oqjdudBFZ9E. As desired, the ego vehicle overtakes the target vehicle in an efficient and legal manner.

Quantitative Results

Fig. 1.9 shows the quantitative dynamic obstacle avoidance results for the nominal controller. As shown in the top left subplot, the reference (in blue) is tracked well when

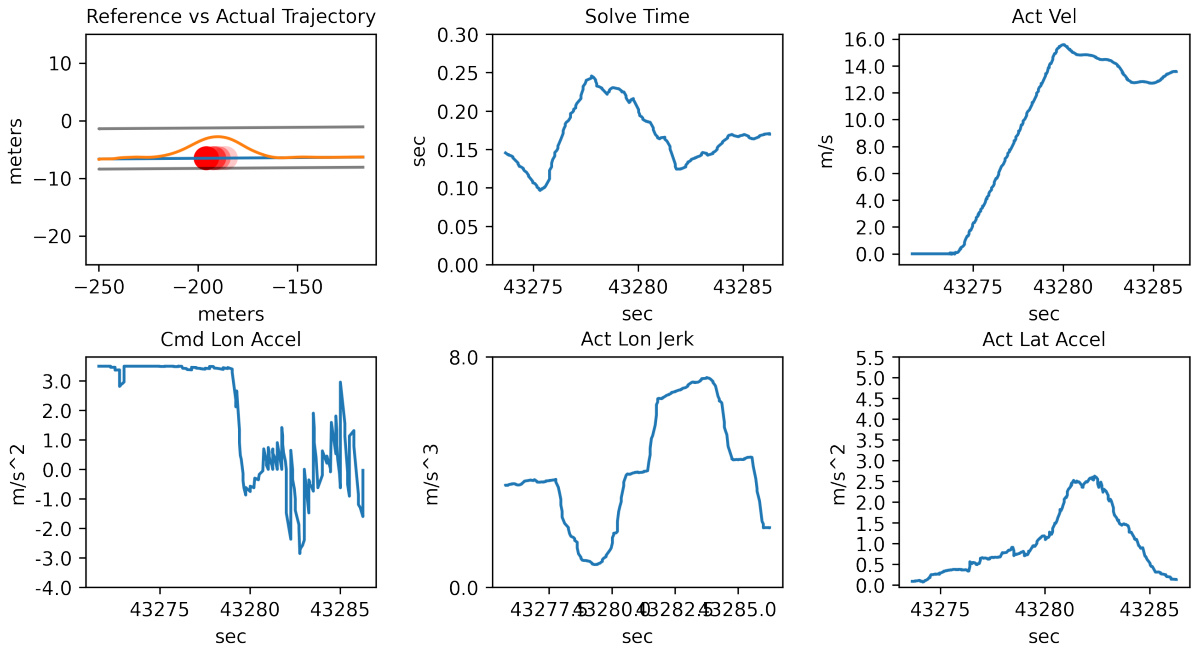

Figure 1.9: Quantitative dynamic obstacle avoidance performance of the nominal controller.

the controller is not overtaking the target vehicle. The controller also successfully avoids the target vehicle (in semi-transparent red), while staying inside the driveable surface bounds (shown in grey). Furthermore, the controller performs the maneuver efficiently (near 15m/s) and comfortably (SAE guidelines obeyed).

1.6 Conclusion

This chapter presented a novel OCP formulation and MPC solution to the DDT that allows for unified trajectory planning and control in a feasibility guaranteed manner. The presented scheme is shown to have key advantages over previous DDT motion planning and execution schemes, including direct adherence to SAE passenger comfort guidelines, as well as free space and road boundary conscious control via the OCP constraints. Promising experimental results were achieved for obstacle avoidance, overtaking, and turning maneuvers.

As for future research directions, incorporating learning techniques for more accu-

rate predictions of non-ego vehicle trajectories will enhance the motion planning scheme[1]. Furthermore, we plan to integrate the proposed motion planning and control framework with learning-based behavioral planners, e.g. [2], for developing feasible decision-making schemes in complex urban environments.

[1]Note that the OCP design will not have to change, only the predictive accuracy of the inputs does.

References

[1] Autodrive challenge - autodrive™ challenge.

[2] Mohammad Al-Sharman, Rowan Dempster, Mohamed A. Daoud, Mahmoud Nasr, Derek Rayside, and William Melek. Self-Learned Autonomous Driving at Unsignalized Intersections: A Hierarchical Reinforced Learning Approach for Feasible Decision-Making. 9 2022.

[3] Joel AE Andersson, Joris Gillis, Greg Horn, James B Rawlings, and Moritz Diehl. Casadi: a software framework for nonlinear optimization and optimal control. *Mathematical Programming Computation*, 11(1):1–36, 2019.

[4] K. Berntorp. Path planning and integrated collision avoidance for autonomous vehicles. In *2017 American Control Conference (ACC)*, pages 4023–4028, 2017.

[5] H.G. Bock and K.J. Plitt. A multiple shooting algorithm for direct solution of optimal control problems*. *IFAC Proceedings Volumes*, 17(2):1603–1608, 1984. 9th IFAC World Congress: A Bridge Between Control Science and Technology, Budapest, Hungary, 2-6 July 1984.

[6] J. Changhao, H. Miaohua, and S. Liyang. An autonomous vehicle motion planning method based on dynamic programming. In *2020 17th International Computer Conference on Wavelet Active Media Technology and Information Processing (ICCWAMTIP)*, pages 394–398, 2020.

[7] Mohamed Ashraf Gameleldin Daoud. Simultaneous local motion planning and control, adjustable driving behavior, and obstacle representation for autonomous driving. Master's thesis, University of Waterloo, 2020.

[8] Rowan Dempster, Mohammad Al-Sharman, Yeshu Jain, Jeffery Li, Derek Rayside, and William Melek. Drg: A dynamic relation graph for unified prior-online environ-

ment modeling in urban autonomous driving. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 8054–8060. IEEE, 2022.

[9] Huckleberry Febbo, Jiechao Liu, Paramsothy Jayakumar, Jeffrey L Stein, and Tulga Ersal. Moving obstacle avoidance for large, high-speed autonomous ground vehicles. In *2017 American Control Conference (ACC)*, pages 5568–5573. IEEE, 2017.

[10] Alberto Franco and Vitor Santos. Short-term path planning with multiple moving obstacle avoidance based on adaptive mpc. In *2019 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)*, pages 1–7. IEEE, 2019.

[11] J. V. Frasch, A. Gray, M. Zanon, H. J. Ferreau, S. Sager, F. Borrelli, and M. Diehl. An auto-generated nonlinear mpc algorithm for real-time obstacle avoidance of ground vehicles. In *2013 European Control Conference (ECC)*, pages 4136–4141, 2013.

[12] H. Guo, C. Shen, H. Zhang, H. Chen, and R. Jia. Simultaneous trajectory planning and tracking using an mpc method for cyber-physical systems: A case study of obstacle avoidance for an intelligent vehicle. *IEEE Transactions on Industrial Informatics*, 14(9):4273–4283, 2018.

[13] Houjie Jiang, Zhuping Wang, Qijun Chen, and Jin Zhu. Obstacle avoidance of autonomous vehicles with cqp-based model predictive control. In *2016 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 001668–001673. IEEE, 2016.

[14] Shaosong Li, Zheng Li, Zhixin Yu, Bangcheng Zhang, and Niaona Zhang. Dynamic trajectory planning and tracking for autonomous vehicle with obstacle avoidance based on model predictive control. *Ieee Access*, 7:132074–132086, 2019.

[15] Jiechao Liu, Paramsothy Jayakumar, Jeffrey L Stein, and Tulga Ersal. Combined speed and steering control in high-speed autonomous ground vehicles for obstacle avoidance using model predictive control. *IEEE Transactions on Vehicular Technology*, 66(10):8746–8763, 2017.

[16] Jose A Matute, Mauricio Marcano, Sergio Diaz, and Joshue Perez. Experimental validation of a kinematic bicycle model predictive control with lateral acceleration consideration. *IFAC-PapersOnLine*, 52(8):289–294, 2019.

[17] Nima Moshtagh. Minimum volume enclosing ellipsoid. https://www.mathworks.com/matlabcentral/fileexchange/9542-minimum-volume-enclosing-ellipsoid, 2022. [Online; accessed August 16, 2022].

[18] Philip Polack, Florent Altché, Brigitte Novel, and Arnaud de La Fortelle. The kinematic bicycle model: A consistent model for planning feasible trajectories for autonomous vehicles? pages 812–818, 06 2017.

[19] Robert Ritschel, Frank Schrödel, Juliane Hädrich, and Jens Jäkel. Nonlinear model predictive path-following control for highly automated driving. *IFAC-PapersOnLine*, 52(8):350–355, 2019.

[20] U. Rosolia, S. De Bruyne, and A. G. Alleyne. Autonomous vehicle control: A nonconvex approach for obstacle avoidance. *IEEE Transactions on Control Systems Technology*, 25(2):469–484, 2017.

[21] Michael J Todd and E Alper Yıldırım. On khachiyan's algorithm for the computation of minimum-volume enclosing ellipsoids. *Discrete Applied Mathematics*, 155(13):1731–1744, 2007.

[22] Andreas Wächter and Lorenz T Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical programming*, 106(1):25–57, 2006.

[23] Qian Wang, Beshah Ayalew, and Thomas Weiskircher. Predictive maneuver planning for an autonomous vehicle in public highway traffic. *IEEE Transactions on Intelligent Transportation Systems*, 20(4):1303–1315, 2018.