KTH Electrical Engineering

Model Predictive Control Course - EL2700

Lab 2 - MPC for autonomous driving and obstacle avoidance

MIKAEL JOHANSSON,
PEDRO F. LIMA, VALERIO TURRI, MARTIN BIEL

Stockholm September 26, 2016

Automatic Control
School of Electrical Engineering
Kungliga Tekniska Högskolan

Contents

Introduction **ii**
1 Model Predictive Control . . . ii
2 The Problem - Autonomous driving and obstacle avoidance . . . ii
3 Modeling . . . iii
4 Getting Started with YALMIP . . . iv
5 Lab overview . . . v
6 MATLAB files . . . v

1 Nonlinear MPC design **1**
1.1 Formulate the nonlinear MPC . . . 1
1.2 Implement the MPC . . . 2
1.3 Introduce the obstacles . . . 4
1.4 Simplify the prediction model . . . 6

2 Linear MPC design **7**
2.1 Linearize the model and constraints . . . 7
2.2 Implement the linear MPC . . . 8

Introduction

The goal of this laboratory exercise is to study Model Predictive Control (MPC). In particular, we will study how MPC can be used for autonomous driving and obstacle avoidance. Because of the nonlinear dynamics inherent in a nonholonomic vehicle, we will design and implement a nonlinear MPC scheme. The aim is to follow a straight road as fast as possible while avoiding some obstacles.

1 Model Predictive Control

MPC differs from most other control strategies in the way the control action is calculated. A finite horizon optimal control problem is solved at each sampling instant. Only the first step in the calculated control sequence is applied to the plant. The calculations are then repeated at the next sampling instant. This principle is known as receding horizon control. A great advantage of solving the optimal control problem on-line is that MIMO plants and constraints can be handled explicitly. Two important considerations are the computation time required to solve the optimal control problem at each sampling instant, and the requirement of a plant model. Traditionally, MPC has been applied to systems with rather slow dynamics, such as process control plants, where the computation time is negligible compared to the sampling time. However, with the emergence of powerful computing hardware, MPC has also been applied to systems with faster dynamics, such as air planes and combustion engines.

2 The Problem - Autonomous driving and obstacle avoidance

Autonomous driving is one of the most active research fields today. One of the most important modules in an autonomous vehicle is the motion controller, i.e., the module responsible for tracking a reference trajectory as precisely as possible. MPC is a common tool in motion control due to its ability to handle non-linear time varying models and constraints in a methodical manner.

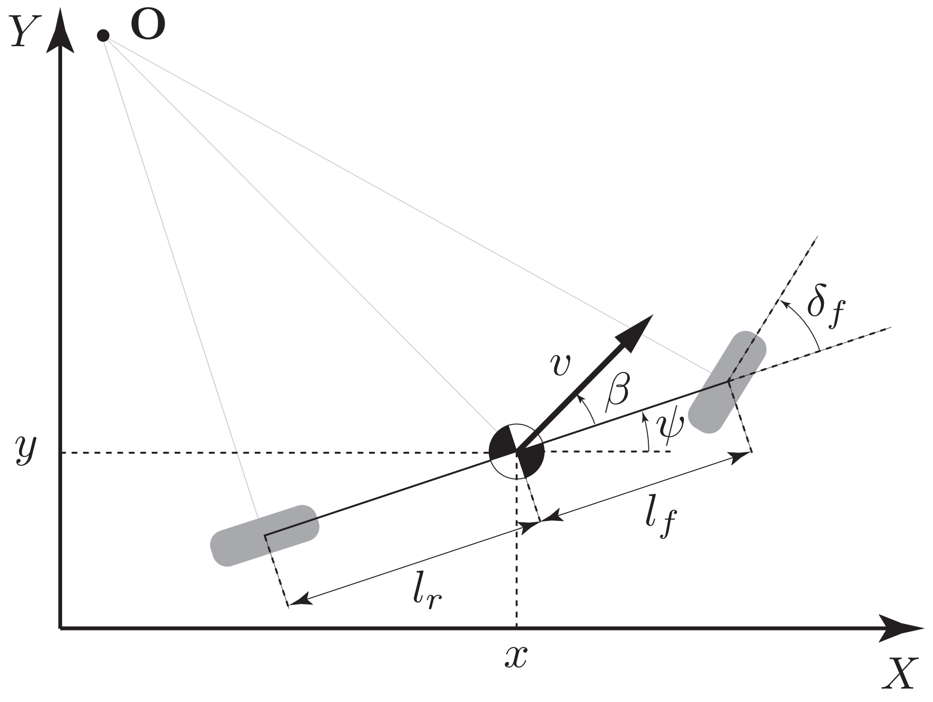
In this lab, we will consider a simplified version of the autonomous driving problem. The vehicle will drive at constant speed and only in a straight road. Also, we will assume that the motion controller has access to fictitious sensor readings, which give a 360 degree view over the environment. This allows the autonomous vehicle to perceive future obstacles and predict corrective trajectories. Also, different MPC tunings and prediction horizons provide different

vehicle behaviors. The final MPC design should be implementable on a real-time autonomous vehicle.

3 Modeling

In the exercise class we have derived the equations for a simplified kinematic bicycle model:

$$
\begin{aligned}
\dot{x}(t) &= v(t)\cos(\psi(t)+\beta(t)), && \text{(1a)}\\
\dot{y}(t) &= v(t)\sin(\psi(t)+\beta(t)), && \text{(1b)}\\
\dot{v}(t) &= a(t), && \text{(1c)}\\
\dot{\psi}(t) &= \frac{v(t)}{l_r}\sin(\beta(t)), && \text{(1d)}\\
\beta(t) &= \tan^{-1}\left(\frac{l_r}{l_f+l_r}\tan(\delta_f(t))\right), && \text{(1e)}\\
& && \text{(1f)}\\
|a(t)| &\leq a_{\max} && \text{(1g)}\\
|\beta(t)| &\leq \beta_{\max} && \text{(1h)}\\
|\dot{\beta}(t)| &\leq \dot{\beta}_{\max} && \text{(1i)}
\end{aligned}
$$



where

$x =$	global x coordinate
$y =$	global y coordinate
$v =$	speed of the vehicle
$\psi =$	global heading angle
$\beta =$	angle of the velocity vector with respect to the longitudinal axis of the car, i.e., sideslip angle
$a =$	acceleration of the center of mass into this direction
$l_r =$	distance from the center of mass of the vehicle to the rear axle
$l_f =$	distance from the center of mass of the vehicle to the front axle
$\delta_f =$	steering angle of the front wheels with respect to the longitudinal axis of the car

Note that there is a direct mapping between β and δ_f. For the lateral control, we use β as control input for the sake of simplicity. The corresponding δ_f can be computed according to (1e). Furthermore, for the longitudinal control, we use the acceleration a as control input.

4 Getting Started with YALMIP

YALMIP is implemented as a free (as in no charge) toolbox for MATLAB. To install YALMIP, visit `https://yalmip.github.io/tutorial/installation/`.

Before starting, visit YALMIP homepage page `https://yalmip.github.io/`. A standard example of MPC implemented in YALMIP is shown here `https://yalmip.github.io/example/standardmpc/`.

5 Lab overview

The lab is composed by two parts:

- **Nonlinear MPC design:** In this part, we will design a nonlinear MPC to solve the problem of motion control and obstacle avoidance. Both the vehicle model and state constraints are nonlinear. The input constraints are linear. We will analyze the influence of weight tuning and prediction horizon in the performance of the controller. Also, we discuss about the advantages and drawbacks of a nonlinear MPC design when deploying the controller in a real vehicle.
- **Linear MPC design:** In this part, we will assume constant speed and linearize the vehicle model around the lane centerline. Also, we simplify the state constraints such that we can have a linear design. Again, we will analyze the influence of weight tuning and prediction horizon in the performance of the controller and discuss about the advantages and drawbacks of a linear MPC design when deploying the controller in a real vehicle.

You should be ready to present your results during the lab session. Saved figures with the plotted results can be helpful. We strongly encourage and suggest that you go beyond the lab tasks to explore and understand the MPC framework. For example, you are free to show a new feature of your MPC controller. We encourage students to work in groups of two or three people to boost discussions and critical thinking.

6 MATLAB files

The files are contained in the folder `EL2700_MPCObsAvoidance` that can be downloaded from the course home page. The main files contained in the folder are:

- `EL2700_Lab2_documentation.pdf`: the current file.
- `obstacle_avoidance.m`: script with a proposed skeleton of the code. In this script, a state-feedback controller is implemented to control the acceleration and the steering of the vehicle.
- `plot_environment.m`: plots the simulation environment. The usage of the function is the following:
  - `plot_environment(z,params)`: plots the environment and the vehicle trajectory;
  - `plot_environment(z,params,z_predicted,u_predicted)`: plots the environment and the vehicle trajectory and plots the predicted states `z_predicted` and inputs `u_predicted`.
- `car_sim.m`: simulates the vehicle for given inputs. Uses a discretized version of the kinematic bicycle model of a vehicle.

Part 1

Nonlinear MPC design

In this part of the lab, we will design a nonlinear MPC for autonomous driving. Also, we will design nonlinear state constraints such that the vehicle avoids the obstacles present in the road. Also, we will discuss the influence of the several tuning parameters in the performance of the controller and its implementability in practice.

Task 1.1 Formulate the nonlinear MPC

Assume that the state vector contains the vehicle position, speed and orientation $z = [x, y, v, \psi]^{\mathrm{T}}$ and that the control input is angle of the current velocity with respect to the longitudinal axis of the car and the acceleration $u = [a, \beta]^T$.

(a) Discretize the vehicle dynamics presented in the introduction using explicit Euler with discretization time T_s. In other words, you should approximate each derivative with

$$\dot{x}_c(t) \approx \frac{x_c(t + T_s) - x_c(t)}{T_s}.$$

You can define the discrete state as $x(k) = x_c(kT_s)$.

(b) Propose a cost function that can be suitable for the MPC autonomous control of the vehicle with a racing objective.

(i) How would you include a reference trajectory in your cost function?

(ii) Is it possible to formulate a MPC controller cost function to solve this task without reference trajectory? How?

(c) Write down the MPC optimization problem to be solved at each sampling time. Include the cost function, vehicle model, state and input constraints, and initial condition.

Task 1.2 Implement the MPC

(a) This is time to get familiar with YALMIP. YALMIP is a high-level framework that allows to define and solve an optimization problem. You will use the YALMIP syntax to define our MPC problem. Check the YALMIP homepage at `https://YALMIP.github.io/` and give a look to the available documentation. We especially suggest to check the MPC example available at `https://yalmip.github.io/example/standardmpc/`.

(b) Find and open the file `obs_avoidance.m`. This file contains a skeleton proposal for the simulation of the closed-loop system. By running it, you will see the closed-loop behavior of the vehicle controlled using a simple state-feedback controller. Give a look to the code and get familiar of the structure of the script.

(c) You will now implement the MPC that you formulated in the previous task. To start, in this subtask we will implement the unconstrained version of the MPC, i.e, only including the cost function and the vehicle model. Change the controller to MPC by setting the variable `params.controller` to

`'MPC'`. Define the cost function and vehicle dynamics using the YALMIP syntax. It can be helpful to give a further look to the MPC example available at `https://yalmip.github.io/example/standardmpc/`. Notice that, in our problem, the vehicle dynamics are nonlinear. Therefore, it is convenient to choose both the control inputs and states as optimization variables as described in the section **Implicit prediction form** of the YALMIP MPC example webpage. Furthermore, you should set YALMIP to use a non-QP solver as `fmincon`. With that end, check the help of `sdpsettings`.

Note: in `sdpsettings`, with `fmincon` use `'usex0'` to make the solver use an initial condition. This is also called a "warm start up". To use as initial condition, the previous MPC solution use the function `assign` (this should be done before the `optimize` and after declaring the optimization variables). This can be of extreme importance, since the problem you are formulating and implementing is non-convex and non-linear. Solver convergence problem can be mitigated if a proper initial condition is provided.

(d) Discuss the influence of different tuning of the MPC parameters.

(e) If you arrived here, you made it. You implemented your first MPC!

It is now time to improve it. In order to do so, include the input constraints defined in (1g) and in (1h).

(Optional) If you want to observe the predicted inputs and states over the prediction horizon, you can use the advanced plotting functionality of `plot_environment.m`. Call it with two additional arguments, the predicted input and the predicted state trajectories, respectively, that you can compute from the YALMIP solver output. In detail, you should use

```
plot_environment(z,params,z_predicted,u_predicted),
```

where `z_predicted` is the predicted state trajectory and `u_predicted` is the predicted input trajectory. See Fig. 1.1 and 1.2 for motivation.

(f) (Optional) Include the constraint on the sideslip rate (1i) in the MPC formulation. First, approximate $\dot{\beta}$ according to the explicit Euler formula presented in the previous task. This should give a constraint on the difference

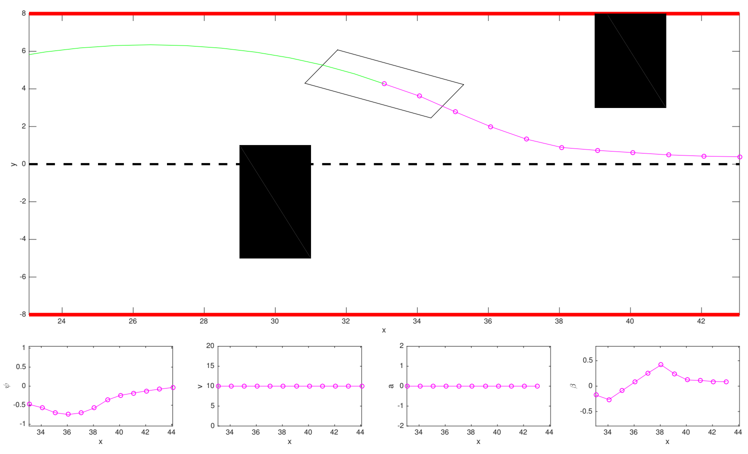


Figure 1.1: Plot example with predicted states and inputs.

of consecutive β, i.e,

$$|\beta(k+1) - \beta(k)| \leq c, \tag{1.1}$$

for a certain c. Include this constraint in the MPC formulation. Notice that, the constraint (1.1) should also hold for the last input $\beta(k-1)$ and the first predicted input $\beta(k|k)$.

Task 1.3 Introduce the obstacles

In this task, we will add some obstacles on the road that the vehicle is expected to avoid. To add the obstacles set the parameter `params.activate_obstacles` to 1.

(a) Does the previous design and implementation avoid the obstacles? Why?

(b) Design state constraints such that the vehicle avoids the obstacles in the road. To that end, we assume total knowledge of the environment. There are 4 rectangular obstacles with size 6×2 m centered at coordinates $(10, -2)$, $(20, 0)$, $(30, -2)$ and $(40, 6)$.

Hint: approximate each rectangle by ellipses. Find the smallest enclosing ellipse for each obstacle maintaining the ratio between the major a and minor axis b equal to the ratio between the width w and length l of the rectangle. Show that $b = \frac{l}{\sqrt{2}}$ and $a = \frac{w}{\sqrt{2}}$,

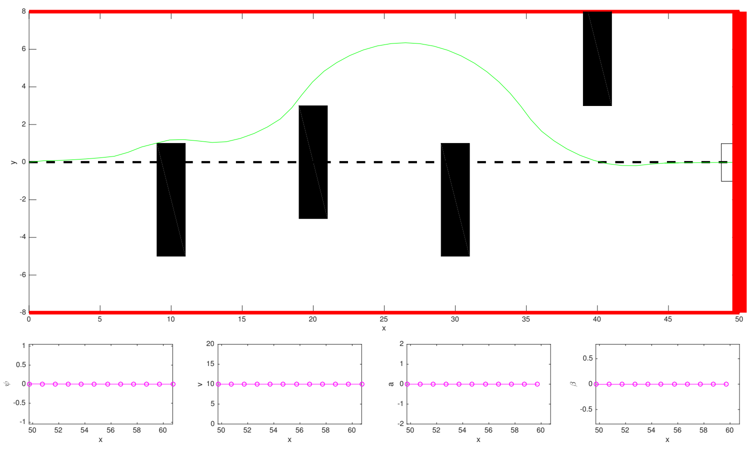


Figure 1.2: Plot example in the end of the simulation.

(c) Add the designed state constraints to the your implementation. Note that, the rectangle already included a safety margin to the real obstacle equivalent to half the width of the vehicle. Therefore, you should guarantee that the center of mass of the vehicle does not hit the obstacles. For such experiment, set the initial position, speed and heading angles to (0, 0) m, 10 m/s and 0 rad, respectively.

(d) Assume that the road semiwidth is of 8 m. How can you update your MPC formulation in order to guarantee that the vehicle does not exit the road bounds? Try it out.

(e) Tune the MPC parameters so that it takes the minimum time to complete the race track. Without obstacles a track of 50 m at 10 m/s is completed in 5 seconds. Can you complete the track with obstacles in less than 6 seconds? This time is **not** the time the solver takes to solve the problem, but yet the number of iterations of the simulation environment times the sampling time.

(f) The execution time of the MPC depends on the type of hardware it is running on. To be able to deploy the MPC in a real vehicle and to run in real-time, the MPC optimization should take less than 100 ms. Investigate the computation time required by your implementation by timing the calls to `optimize` during the simulation. Which MPC parameters influence the execution time? Do you have in mind some other MPC implementation that can reduce the computation time?

    Note: YALMIP incurs some overhead each iteration. The diagnostic struct returned by `optimize` has a field `solvertime` which contains the computation time required to solve the underlying optimization problem, without the overhead. Similar overhead will be present in real-time implementations, although it is possible to reduce it. Nonetheless, it can be interesting to compare these timings. (See provided code skeleton).

Task 1.4 Simplify the prediction model

In this task, we will simplify the model by fixing the acceleration input a to 0. Hence, the speed will be constant over the whole prediction horizon and consequently can be removed from the state vector. The input vector, in the MPC formulation, u will contain 1 variable, i.e, β and the state vector z, 3 variables, i.e, x, y, ψ.

Remove the velocity state from the prediction model and the acceleration constraints. Do you notice any impact in the vehicle performance? How is the execution time affected?

Part 2

Linear MPC design

In this part of the lab, we will turn the MPC implementation more efficient by simplifying the prediction model and state constraints. This is a common solution when the goal is to deploy MPC in a real system. This allows the MPC problem to be cast as a QP problem for which very efficient solvers exist. Also, we will design linear state constraints such that the vehicle still avoids the obstacles.

Task 2.1 Linearize the model and constraints

(a) Assuming constant speed, linearize the model around the lane centerline, i.e., linearize the vehicle model around $\psi = 0$. Write down the resulting state update equations.

(b) What are the advantages and disadvantages of this type of model approximation? Discuss the validity of the approximation.

Task 2.2 Implement the linear MPC

(c) Replace the nonlinear model with the linearized model in your implementation. Confirm that the MPC controller is able to predict the vehicle behaviour and avoid the obstacles. At this point, the non-linear constraints are still present and hence a non-linear solver should still be used.

(d) The ellipses derived as state constraints are nonlinear constraints. Again, we assume total knowledge of the environment. There are 4 rectangular obstacles with size 6×2 m centered at coordinates $(10, -2)$, $(20, 0)$, $(30, -2)$ and $(40, 6)$. Design linear constraints for each obstacle (see Fig. 2.1 for inspiration). For example, when $x_1^{\text{obs}} \leq x \leq x_2^{\text{obs}}$ then $y \leq y_2^{\text{obs}}$.

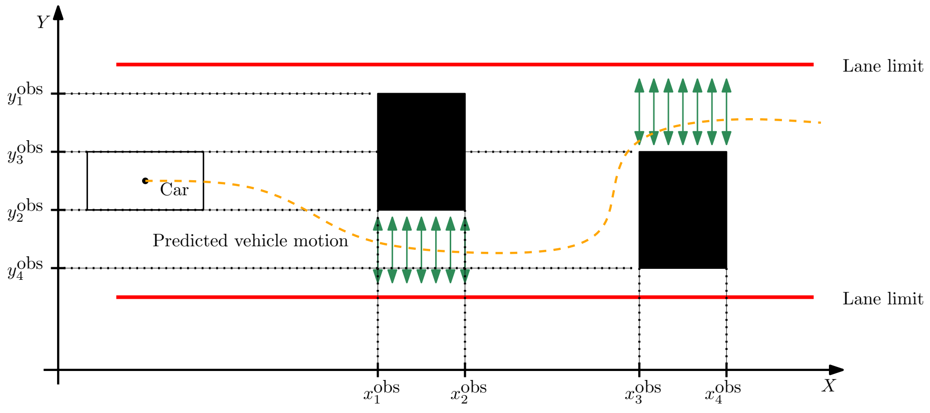


Figure 2.1: Linear obstacle constraints.

(e) What are the advantages and disadvantages of this type of obstacle approximation?

(f) Implement the linear constraints in your previous implementation. The model is now fully linear. Switch to `quadprog` solver (use `sdpsettings` to set the desired solver). Again, set the initial position, speed and heading angles to (0, 0) m, 10 m/s and 0 rad, respectively.

(g) Tune the MPC parameters such that it takes the minimum time to complete the race track. Without obstacles a track of 50 m at 10 m/s is completed in 5 seconds. Can you complete the track in less than 6 seconds?

(h) Again, investigate the required computation times during the simulation. How do they compare to the non-linear implementation?