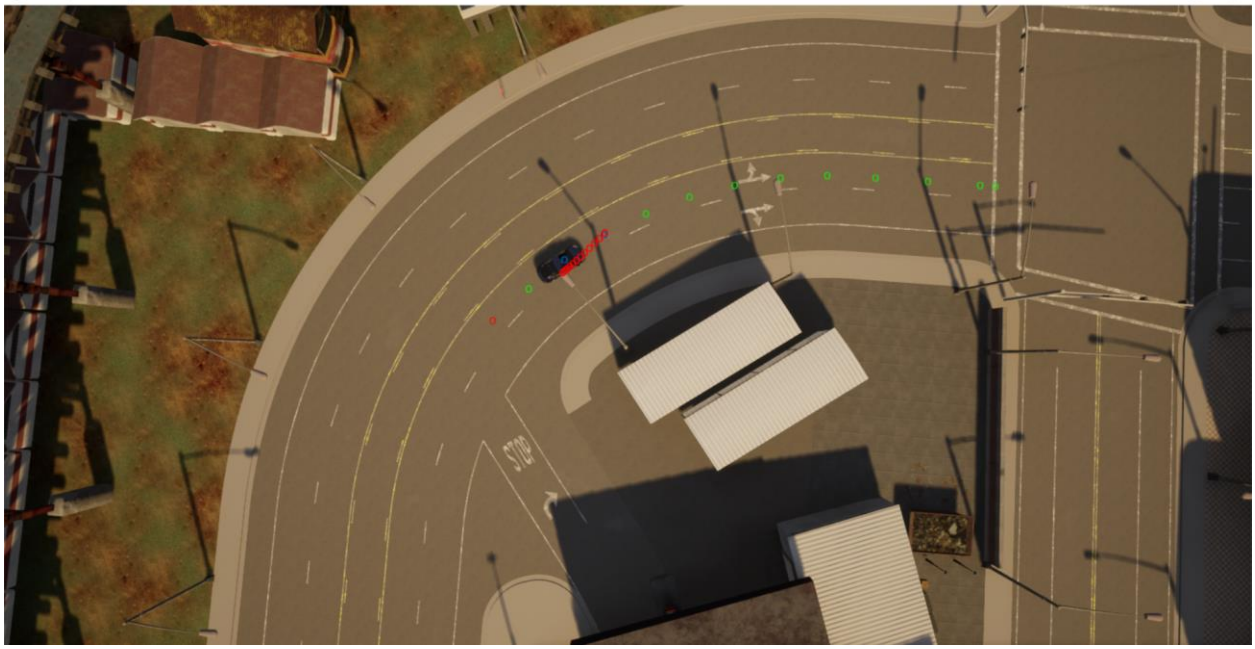
MAE 501 INDIVIDUAL PROBLEMS - PROJECT

# NAVIGATION OF AUTONOMOUS VEHICLE USING MPC

RAJ KALPESHKUMAR SHAH - 50321013




## Overview

Implemented a basic Model Predictive Controller (MPC) to autonomously navigate a car from a start point to a desired goal point in a map by following a trajectory of multiple waypoints. Simulation is done in the CARLA simulator and the optimization tool used for MPC is CasADi.

## CARLA simulator

http://carla.org/

CARLA is an open-source simulator for autonomous driving research. CARLA has been developed from the ground up to support the development, training, and validation of autonomous driving systems. In addition to open-source code and protocols, CARLA provides open digital assets (urban layouts, buildings, vehicles) that were created for this purpose and can be used freely. The simulation platform supports flexible specification of sensor suites, environmental conditions, full control of all static and dynamic actors, maps generation, vehicle models, and much more. Moreover, CARLA is provided with integration with ROS. The various objects in the simulator such as vehicle, sensors, wheels, waypoints, etc. are defined as Python classes having member functions and attributes. We can easily access the methods as well as the parameter attributes of these objects by creating the objects of these classes.

## CasADi

CasADi is an open-source tool for nonlinear optimization and algorithmic differentiation. It facilitates the rapid and efficient implementation of different methods for numerical optimal control, both in an offline context and for nonlinear model predictive control (NMPC). It is compatible with the Python programming language.

## Model Predictive Control (MPC)

MPC also called Receding Horizon Control and Moving Horizon Control is a predictive control technique that involves online optimization.

The idea is to predict the state evolution of the system for N (prediction horizon) time steps in the future and calculate the sequence of optimal control inputs that minimizes the error between the predicted states and the desired state at each time step of the prediction horizon. Then we apply the first control input of the sequence to the system and again calculate the sequence of optimal control inputs at the next time step and repeat the process.

## MPC mathematical formulation

MPC is composed of 3 major steps:

1. Prediction
2. Online optimization
3. Receding horizon implementation

### Prediction

- The state evolution is predicted at each time step of the horizon using the system model.
- The kinematic bicycle model[1] is used to model the car system mathematically

$$\dot{x} = v\,cos(\psi + \beta)$$
$$\dot{y} = v\,sin(\psi + \beta)$$
$$\psi_{dot} = v/lr\,sin(\beta)$$
$$v_{dot} = a$$
$$\beta = arctan(lr/(lr + lf)\,tan(\delta f))$$

Where,

x = x coordinate

y = y coordinate

$\psi$ = inertial heading

$\beta$ = angle of the current velocity of the center of mass with respect to the longitudinal axis of the car

a = throttle

$\delta f$ = steering angle

v = longitudinal velocity

lr = distance from center of gravity to rear wheels

lf = distance from center of gravity to front wheels

- Therefore, it is a 4th order nonlinear system.
- A vehicle system is considered with 4 states namely: x coordinate, y coordinate, angular orientation, and longitudinal velocity
- **State at any time step 't'** is represented as a vector of these quantities:

  $X_t = [x, y, \psi, v]^T$

- Control inputs at any time step 't' are represented as vector of throttle 'a' and steering angle '$\delta f$'

  $u_t = [a, \delta f]^T$

- Using this system model, state evaluation is predicted for the entire horizon as following:

  $X_{t+1} = X_t + T*[\dot{x}, \dot{y}, \psi_{dot}, v_{dot}]^T$

  Where, T = sampling time

- The horizon considered in the project is N= 12 and sampling time is T = 0.8.

## Online optimization

- Once we have the symbolic representation of the predicted states at each time step of the horizon as a function of the current vehicle state (initial condition) and control inputs represented as symbolic variables, the objective function is defined as follows:
- Objective function or the running cost is defined as follows:

$$l(X,U) = \sum_{i=t}^{t+N}((X_i - X_r)^T * Q * (X_i - X_r) + (u_i - u_r)^T * R * (u_i - u_r))$$

- Then the MPC problem is boiled down to Optimal Control Problem (OCP) as follows:

$$\underset{U}{\text{minimize}} \sum_{i=t}^{t+N}((X_i - X_r)^T * Q * (X_i - X_r) + (u_i - u_r)^T * R * (u_i - u_r))$$

$$\text{subject to}$$

$$u(i) = U_c,\ i = t, \ldots, t+N$$

$$X(i) = X_c,\ i = t, \ldots, t+N$$

Where,

*U* is a matrix of size 2 X N. Each column corresponds to control input (throttle and steering angle) at time step 'i'.

X is a matrix of size 4 X N. Each column corresponds to the predicted state at that particular time step in the prediction horizon

$X_r$ is reference state

$U_r$ is reference control input ($[0, 0]^T$ in our case)

N is the length of the prediction horizon

$U_c$ is control bounds which needs to be satisfied throughout the horizon

$X_c$ is state bounds which needs to be satisfied throughout the horizon

- Note here that the state is predicted using the system model and therefore the objective function by default has the system model as another constraint.
- Q and R matrices are set heuristically by tuning the parameters through trial and error.
- The output of this Optimal Control Problem (OCP) is *U* which is nothing but a sequence of optimal control inputs corresponding to each time step of horizon.

## Receding horizon Implementation

The first column of the matrix *U* is applied to the system and then the whole process is repeated until the vehicle reaches its goal point in the map.

At each iteration, the initial state of the system, based on which the states are predicted for the N times steps in the future, is the immediate current state of the vehicle.

## Performance

The following video illustrates the performance of the above described MPC on a vehicle in the CARLA simulator.

https://youtu.be/tBYQyAXvXEg

## Milestones/Challenges

- Managing hardware for the simulation was one of the major challenges faced. CARLA simulator requires GPU. However, after trying different versions of CARLA, I found the one that could run smoothly on my laptop.
- Next challenge was to understand the CARLA simulator and how to spawn, and control vehicle and its attributes using Python programming language
- Then I spent time on understanding LQR and Model Predictive Controller and their implementations
- Next challenge faced was to understand the CasADi optimization tool for non-linear online optimization in MPC and integrating it with Python
- Final milestone was to get the vehicle model parameters from the CARLA built version in unreal engine and then write a python script to successfully implement MPC

## Future work

- In the current controller, the reference state of the system is constant i.e. the next waypoint in the map. As soon as the vehicle reaches within some threshold distance to the current reference waypoint, the reference state changes to the next waypoint in the sequence. Therefore, it is a point stabilization. A Proper trajectory tracking

controller needs to be designed where the reference state varies continuously with time.

- A detailed analysis of the system needs to be done. For example, system's stability and controllability could be studied and accordingly a more robust controller could be designed
- The controller implemented is a single-shoot MPC. Multiple shoot MPC could be implemented to further improve the controller performance.

# Reference

Kinematic and Dynamic Vehicle Models for Autonomous Driving Control Design Jason Kong1 , Mark Pfeiffer2 , Georg Schildbach1 , Francesco Borrelli1