Effectiveness of a Simplified MPC under Different Parking Circumstances

Xiaozhu Fang*, Yide Gu*, Jiawen Shi*, Zhiyu Wang*
*Department of Mechanical Engineering, University of Michigan, MI, 48105, USA

***Abstract*—Model predictive control (MPC) method has been comprehensively used in vehicle control. Because the detectable states are only the position and longitudinal velocity in practice, we propose simplified kinematic equations as the predictive model. This paper performs this MPC under three different kinds of environment: (1)the naive kinetic one to discuss the parameters of MPC; (2)the complex dynamic one to deal with the inconsistency when the predictive model and the system model are different; (3)the authentic noisy one in consideration of the annoying sensor noise. As the solution, we convert the input to solve the consistency and introduce the extended Kalman filter (EKF) to deal with the noise. As a result, the simple predictive model combining the filter achieves great performance, which can be a solution for automatic vehicle parking in reality.**

***Index Terms*—vehicle parking, model predictive control, extended Kalman filter.**

I. Introduction

Self driving has attracted significant attention so far. In this project, we focus on the control of self parking. Our objective is to control the vehicle to move along the designed trajectory, from the initial position to the designed parking space. In addition, the vehicle should not have any collision with obstacles during parking process.

Multiple control strategies have been developed to control the vehicle. For relatively simple models, PID and LQR have been proved very powerful. For vehicle models, however, we need a more complicated controller to control an adaptive non-linear system. MPC is introduced under this circumstance. The operations of MPC can be generalized into three steps: first, find the reference trajectory; second, determine the inputs for the following several time steps and the optimized trajectory under the pre-given constrain; finally, implement the next-one-step input to the model and get the actual trajectory. After every actuation, the procedure repeats over time. In addition, the generation of reference is a motion-planning problem, beyond the scope of this course. Therefore, in this project, we just apply a pre-defined reference and focus on the controller design.

Since parking problem emphasizes the precision and stability, we would discuss the influence of MPC's model and parameters. Two classical models, kinematic and dynamic models, are compared in details. Furthermore, we introduce the noise for sensors, which dramatically strengthens the deviation. To deal with the noise, extended Kalman filter is added to MPC.

II. Background

The schematic of the total system is illustrated in Fig. II. The main idea is to apply MPC to control the vehicle motion. Although there is no noise in our simulation, there would be sensor noise in practice. Therefore, we add a sensor noise and use Extended Kalman Filter to reduce the noise. The results show that EKF are useful.

A. MPC algorithm

We used quadprog function in Matlab to implement the MPC controller. The objective is to minimize the cost function:

$$\frac{1}{2}x^T H x + f^T x \text{ such that } \begin{cases} A \cdot x \leq b \\ Aeq \cdot x = beq \\ lb \leq x \leq ub \end{cases}$$

In our model, the cost function is the quadratic form so that f is set to zero. Argument A_{eq} and b_{eq} are obtained from discretized kinematics system model (3 states and 2 inputs). And argument H is composed of Q and R diagonally. x in the loss function consist of state and inputs. Q and R matrix are similar to LQR controller, where Q and R determine the weight of state and input respectively. For Q, as the problem description, we are more interested in the position of the vehicle and it is critical for collision detection, so we design the first two entries of Q matrix to be greater. For R, for the similar reason, speed is more important than steering angle, so the first entry has a heavier weight. At last, $A \cdot x \leq b$ and $lb \leq x \leq ub$ are the extra constrain for MPC.

For the prediction horizon, we determined it to be 10 as our start point and stack Q and R matrix accordingly to form H matrix required by quadprog function. For each time step, the MPC controller finds the optimal solution to the quadratic problem. After that, ode45 function takes the result from MPC and calculate the next state according to the system model. Finally, we will get an actual trajectory.

B. Extended Kalman filter

Extended Kalman filter is very powerful and comprehensive in modern technology. It can be used to detect the position, speed and acceleration of vehicles. Since the data collected the real sensor might include the noise, the filter could denoise the data. To begin with, the discrete model of system should be provided as follows.

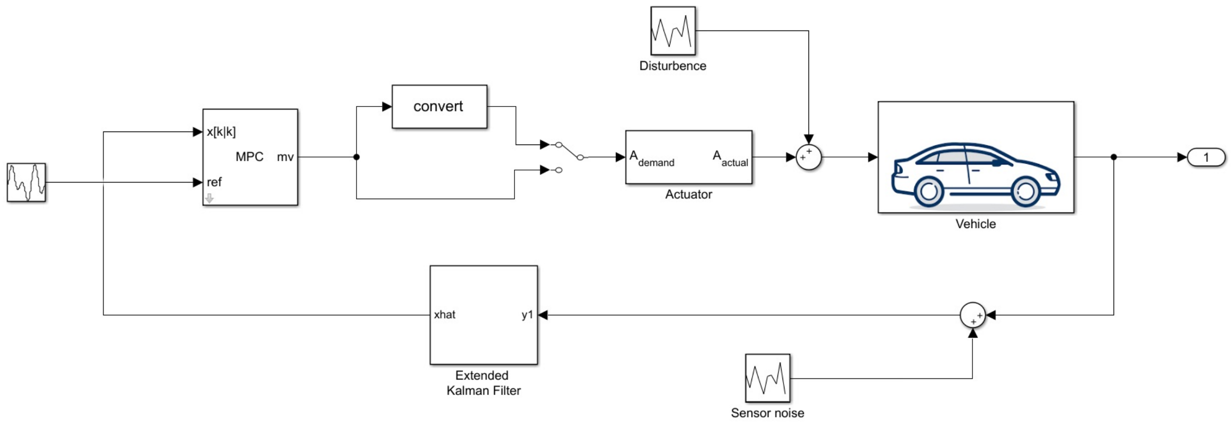
$$x_k = s_k + v_k \tag{1}$$

$$s_k = \mathbf{C}_k^T \xi_k \tag{2}$$

$$\xi_{k+1} = \mathbf{A}_{k+1,k}\underline{\xi}_k + \mathbf{B}_k \underline{w}_k, \quad \underline{\xi}_0 = \underline{\xi}_o \tag{3}$$

where $\xi(k)$ is the state, $s(k)$ is exact output,$x(k)$ is measure value, $v(k)$ is the zero mean noise due to the senor, $w(k)$ is the state disturbance.

The algorithm of the Kalman filter is illustrated in Fig. 1. $\hat{s}(k|k-1)$ is the predicted value of $x(k)$ based on model and previous k-1 measurements. The difference $\eta(k)$ is the error.



Block diagram of the total system.

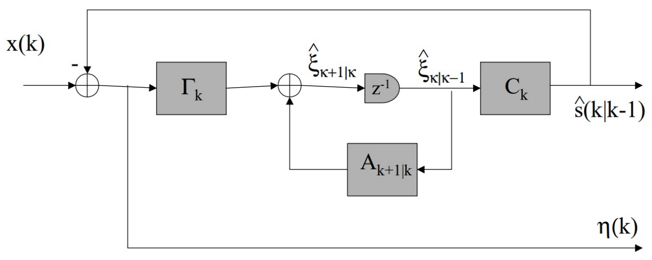


Fig. 1. Block diagram of Kalman filter.

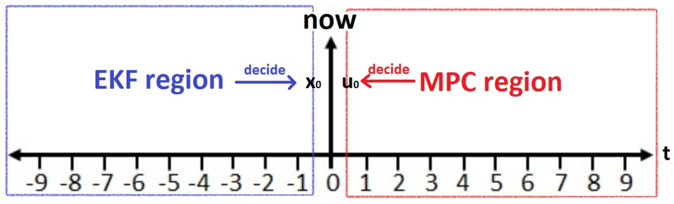


Fig. 2. Coupling of MPC and EKF.

$\mathbf{A}_k$ and $\mathbf{C}_K$ corresponds to the state space model and Γ_k is called Kalman Gain. The complete derivation of Kalman Gain needs the complex recursion, which can refer to [1]

The schematic of combination of EKF and MPC is shown in Fig. 2. EKF uses past states to reduce noise of the current state, getting a better current state x_0, while MPC is designed to obtain the current input u_0. Combining the modified x_0 with u_0, we could get the better trajectory.

III. System Modelling

Due to the symmetry of car, we use the bicycle model to represent the car models in this project, in both kinematic and dynamic models.

A. Dynamic Model

Using the model introduced in [2]. the schematic is shown in Fig. 3. $F_{r,x}$ is the longitudinal force on the rear wheel while the longitudinal force on the front wheel is neglected in this model. $F_{f,y}$ and $F_{r,y}$ are lateral forces on the front and rear wheels, respectively. l_f and l_r are distance from center of gravity to the front and rear wheel, respectively. Six states are used in this model: position X and Y, angle between the car and the inertial frame φ, longitudinal and lateral velocity v_x and v_y, and yaw rate ω. Inputs are PWM duty cycle d of the motor and steering angle δ. The motion equations are listed through Eq.(4) to (9).

$$\dot{X} = v_x \cos(\varphi) - v_y \sin(\varphi) \tag{4}$$
$$\dot{Y} = v_x \sin(\varphi) + v_y \cos(\varphi) \tag{5}$$
$$\dot{\varphi} = \omega \tag{6}$$
$$\dot{v}_x = \frac{1}{m}\left(F_{r,x} - F_{f,y}\sin\delta + m v_y \omega\right) \tag{7}$$
$$\dot{v}_y = \frac{1}{m}\left(F_{r,x} + F_{f,y}\cos\delta - m v_x \omega\right) \tag{8}$$
$$\dot{\omega} = \frac{1}{I_z}\left(F_{f,y} l_f \cos\delta - F_{r,y} l_r\right) \tag{9}$$

The lateral and longitudinal forces are approximated as:

$$\begin{aligned} F_{f,y} &= D_f \sin\left(C_f \arctan\left(B_f \alpha_f\right)\right) \\ F_{r,y} &= D_r \sin\left(C_r \arctan\left(B_r \alpha_r\right)\right) \\ F_{r,x} &= \left(C_{m1} - C_{m2} v_x\right) d - C_r - C_d v_x^2 \end{aligned}$$

where

$$\begin{aligned} \alpha_f &= -\arctan\left(\frac{\dot{\varphi} l_f + v_y}{v_x}\right) + \delta \\ \alpha_r &= \arctan\left(\frac{\dot{\varphi} l_r - v_y}{v_x}\right) \end{aligned}$$

All the parameters used in this model are listed in Table I.

B. Kinematic Model

As introduced before, the detectable states are actually position and longitudinal velocity. So we reduce the six-state dynamic model to the three-state kinematic model: position X and Y, and angle between the car and the inertial frame φ. Inputs are longitudinal velocity u and steering angle δ.

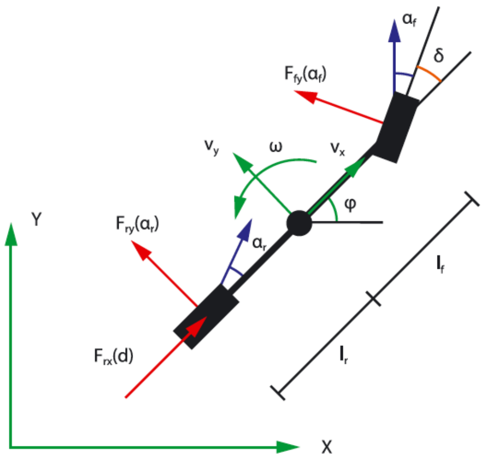


Fig. 3. Schematic of dynamic model.

TABLE I
PARAMETERS IN THE DYNAMIC AND KINEMATIC MODEL.

Attribute	Value	Unit
m	1573	kg
I_z	2873	$kg \cdot m^2$
l_f	1.35	m
l_r	1.35	m
C_{m1}	17303	$kg \cdot m/s^2$
C_{m2}	175	kg/s
C_r	120	$kg \cdot m/s^2$
C_d	0.5359	kg/m
B_f	13	
C_f	2	
D_f	9258.7	
B_r	13	
C_r	2	
D_r	9258.7	

The kinematic model is now:

$$\dot{X} = u\cos(\varphi) - \frac{l_r}{l_r + l_f} u \tan(\delta)\sin(\varphi) \quad (10)$$

$$\dot{Y} = u\sin(\varphi) + \frac{l_r}{l_r + l_f} u \tan(\delta)\cos(\varphi) \quad (11)$$

$$\dot{\varphi} = \frac{u}{l_r + l_f}\tan(\delta) \quad (12)$$

The parameters are the same as in dynamic model.

IV. CONTROL DESCRIPTION

The objective is to park the vehicle into parking space. The parking scenario is illustrated in Fig. 4 (in this figure, the car is backed into parking space, while we use front-parking in our simulation). In this project, we selects a compact car with dimension of 4.45×1.8 m and the dimension of the perpendicular parking space is 2.6×5.8m.

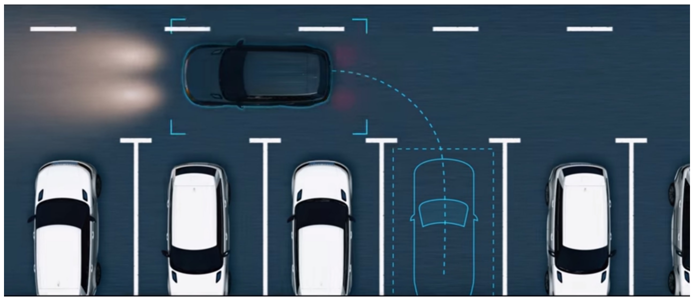


Fig. 4. Illustration of parking. [3]

One control specification is to finish parking within 2 minutes, as stated in proposal. After simulation in Matlab, we find it can finish parking in 18 seconds and get good result. This is reasonable since our simulation is one-time park without adjusting. In addition, to guarantee the precision of parking, we constrain the error to be within 0.1m, as stated in proposal.

To simplify the problem, several **Assumptions** are made as following:

- The continuous movement of the dynamic model is the most authentic one, even though it includes the approximation due to the model and ode45.
- The delay is neglected, the calculated input impacts on the current state.
- The senors without noise are perfect, returning true value of the vehicle's state. In later part, the noise is independent Gaussian white noise.

Although we assume that the dynamic model represents the actual motion of the vehicle, in this project, we apply MPC on the kinematic model because actually the detectable states of the vehicle are only the position and longitude velocity. It is very hard to get the instant lateral velocity (v_y) and yaw rate (ω). However, MPC needs to update all states of the model over time. If we apply MPC to dynamic model, there is no much difference with the kinematic model since we only get the results of three out of six states.

For the controller design, there are several **Constraints**:

- The maximum steering angle is 0.7 rad
- The maximum speed is 2 m/s
- The maximum acceleration is limited by the engine, usually no more than 10 m/s^2.

To verify the performance of this MPC controller, we implement it under three different system and environment.

1) Naive system: The vehicle's behavior follows the kinematic model, consistent with the predictive model. Although the kinematic condition unrealistic, it is worthy for the discussion of parameters' selection.
2) Complex system: The vehicle's behavior follows the dynamic model. The predicted input by kinematic-based MPC is adapted to the dynamic input.
3) Noisy system: The vehicle's behavior follows the dynamic model with noise, resulting in the necessity of Kalman filter.

V. Results

The main part of our controller design and implementation is by using Matlab script. As explained before, we apply MPC to the kinematic model, and obtain the inputs. Use the inputs of kinematic model to compute inputs of dynamic model, then run the dynamic model to further validate the results.

A. Kinematics

We first discretize our model along the reference input and reference trajectory, then determine linear equality constraints according to the discretized model and set input boundary constraints according to the problem description. By using a prediction horizon of 10 as our starting point, we test our controller under various initial conditions and reference trajectory. We also define several ways to evaluate the performance of our controller. The environment parameters are described in Part *Control Description*.

1) Simulation design: We design trajectory to simulate a perpendicular parking process with a left turn. We design the trajectory in two ways. The first way is basically a trial and error method. We first design the input intuitively and run the forward kinematics model to get the final state of vehicle. Then we adjust input speed and steering angle until we get an ideal (no collision and nearly zero final state error) trajectory. The second way is to design the input and trajectory (states) at the same time according to mathematical calculation without using kinematic model. To calculate the error, since potential collision will only happen when the y position is smaller than $6m$, we simply calculate the norm distance between the reference state and actual state when $y \leq 6$ and $x \leq 3$. Fig. 5 shows the difference between reference trajectory and kinematic trajectory, where the two black boxes represent parking spaces adjacent to our target parking spaces. The most significant difference between the reference and calculated trajectory is the initial condition. We can see how well the controller can lead the vehicle close to our reference trajectory and then follow it. For this example, the maximum distance error is $0.072m$, which is considered satisfying since our objective is to park within the error of $0.1m$.

We also design a function to visualize the actual bounding box of a typical compact car and the shaded area in Fig. 6 shows the actual area that the car model would cover during parking process. The two black boxes represent other two parking spaces adjacent to our target parking space. With this function, we can see clearly whether our vehicle will have collision with other obstacles parked in the parking lot. *Script Animation.m* will generate such a plot.

2) Discussion: To test the robustness of our controller, we change our initial condition and test for the maximum state error. The maximum state error should be smaller than 0.5 to make sure there is no collision between our vehicle and adjacent parking spaces. The number 0.5 is not strictly determined, the number is determined by repeatedly testing the initial condition and running animation to see whether there is collision manually. Finally, we find that 0.5 can roughly

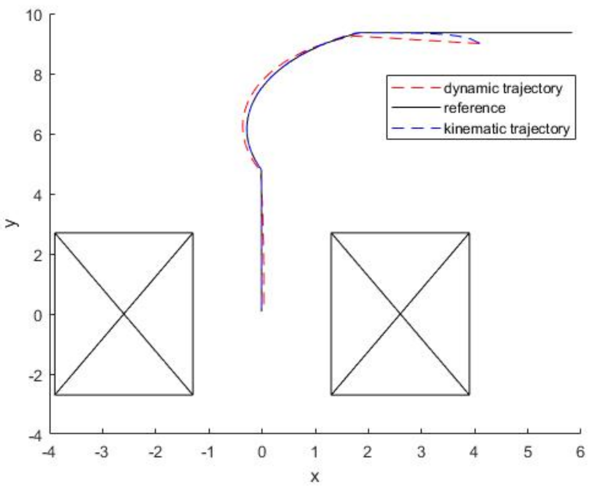


Fig. 5. Comparison between reference and MPC kinematic trajectory. The dynamic trajectory is for discussion in the later part.

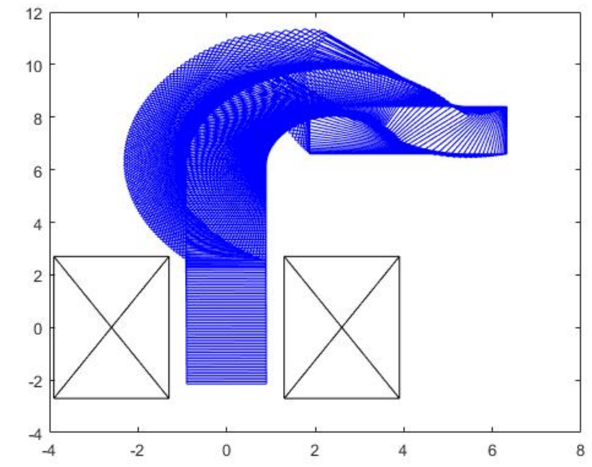


Fig. 6. Actual trajectory of bounding box in 2D plane

satisfy our requirement. we test for various initial conditions and find out a region. If we take point in this region as our initial condition, the vehicle is guaranteed to reach the target position safely. The area was shown in Fig. 7. Note that during the test, we constrain y position in a range of $3 \leq y \leq 9.36$. Since in the real life, a vehicle can neither drive too far away from nor too close to the parking spaces, we consider these constraints reasonable. The initial point of our reference trajectory is actually not so satisfying since some parking lots may not have such large space between two rows of perpendicular parking slots. Since it is tricky and hard to find a better reference trajectory, and the main purpose of our project is not path planning but controller design, we will not focus on getting a better reference.

We also test the controller with different sampling time. By fixing the initial condition ([6 8.7 π]') and increasing the

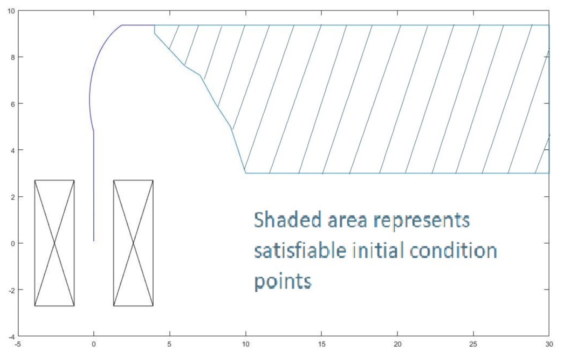


Fig. 7. Shaded area represents the region of satisfying initial condition

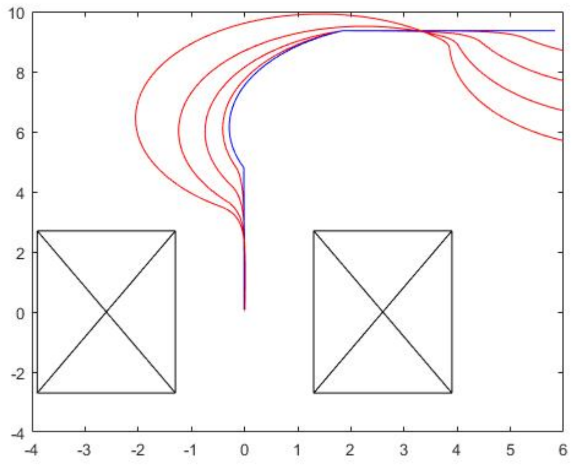


Fig. 8. A bad initial condition (IC) sample shows difference between reference (blue) and actual kinematic (red) trajectory. Four bad IC samples are tested.

sampling time from 0.01s, 0.02s, 0.05s to 0.1s, we observe the maximum distance error varying from $0.1329m$, $0.1512m$, $0.3598m$ and $0.7198m$. This result is quite intuitive since the performance of controller will get worse with a smaller sampling rate.

Overall, the performance of MPC controller is quite well since the final state error is much smaller than 0.1m as purposed in the proposal. The vehicle can finish the parking process in 18 seconds, also satisfies our requirement of no more than 2 minutes.

The estimation of region with satisfying initial conditions is useful since it can help the driver or autonomous driving system to determine when to turn on the MPC controller so that the vehicle can reach the target position safely.

Fig. 8 and Fig. 9 show an example with initial condition outside of this region. We can see that the state error is large and there would be collision during parking process.

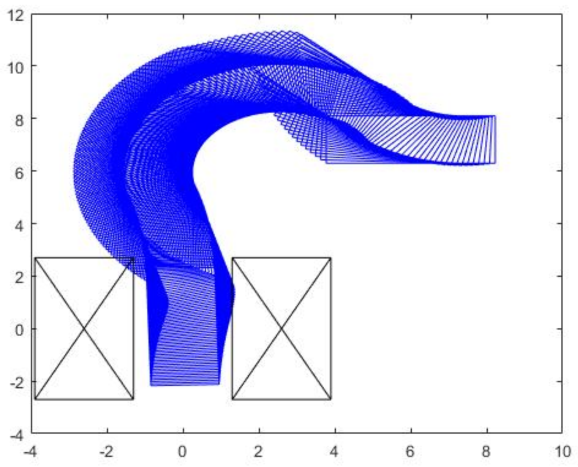


Fig. 9. A bad IC sample shows the collision of vehicle with adjacent parking spaces.

B. Dynamics

The kinematic model is actually a naive model without the consideration about the friction of the tire, which cause the lateral movement of the vehicle. In Control Description part, we have explained why we give up selecting this complex model for MPC: the only 3 observable state in practice makes it meaningless. However, the behavior of the vehicle must conforms the dynamics rather than simple kinetics. Therefore, we propose that we use the result of the simplified kinetic MPC to control the complex dynamic system.

1) Bridge over kinetics and dynamics: The difficulty is that the target input provided by MPC (longitudinal velocity u and steering angle δ)is inconsistent with the system input (PWM duty cycle d and steering angle δ). Fortunately, d can be derived approximately by u.

$$\dot{v}_x(k) = \frac{1}{m}\left(F_{r,x} - F_{f,y}(k)\sin\delta(k) + mv_y(k)\omega(\,k)\right) \quad (13)$$

where $F_{r,x} = (C_{m1} - C_{m2}v_x(k))\,d - C_r - C_d v_x^2(k)$ and $\dot{v}_x(k) \approx (v_x(k) - v_x(k-1))/dt$. Therefore, we translate the kinematic language into dynamic one and manage to use kinematic MPC in the dynamic system. Blue dash lines in Fig5 shows the result of this control.

2) Overshoot: Overshoot deserves the great caution for parking problem because sometimes the overshoot might leads to striking the wall. Although we could add constrain in MPC to ban the region, the vehicles dynamics (inertia) might cause the problem. To make it clear, we plot Fig10 to observe the response of two trajectory. we can see the dynamic situation has the overshoot while the kinematic one does not. This is well explained by the so-called ”dynamics”. The input in the dynamic model could only change the acceleration rather than the velocity directly. Such indirect control makes the response delayed and is more likely to generate overshoot and oscillation.

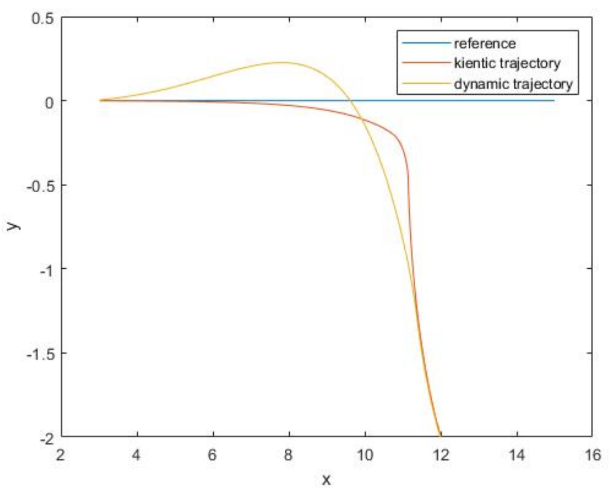


Fig. 10. Response of kinetic trajectory and dynamic trajectory with respect to the reference.

TABLE II
MEAN SQUARE ERROR OF THE TRAJECTORY IN FIG11.

case	input δ	MSE_x	MSE_y	total MSE
#1 kinematic	0.4	7.57E-6	2.12E-6	7.86E-6
#1 dynamic		4.68E-5	6.71E-5	8.18E-5
#2 kinematic	0.6	6.28E-6	3.42E-6	7.15E-6
#2 dynamic		3.46E-4	1.35E-4	3.72E-4
#3 kinematic	0.8	7.21E-6	2.51E-6	7.64E-6
#3 dynamic		3.07E-3	5.17E-3	6.00E-3
#4 kinetic	1.0	6.72E-6	3.09E-6	7.40E-6
#4 dynamic		0.380	0.075	0.387

3) Steering curve: In control task, we limit our steering angle to 0.7, which is recorded in the MPC. In kinetic model, the steering angles leads to the bending curve, so the curvature of the bending can be as large as possible until the steering angle is 90 degree. In dynamics model, however, we can show that the curvature of trajectory would saturate because of the slipping of the tire.

Fig11 illustrates this phenomenon. The reference is given by kinematic MPC. The vehicle is required to accelerate in the straight and instantly turn an angle. We can see when steering angle is small, the kinematic and dynamics system both track the reference very well. When steering angle is as large as 1 rad. we can see that dynamic trajectory has large deviation due to the slipping while the kinematic trajectory still overlaps the reference. To be obvious, we calculate the mean square error of X and Y and total position deviation in following table.

Therefore, it is reasonable that manufacturers constrain the maximum steering to 0.7 to avoid the dangerous slipping of the vehicle.

C. Noise

The appearance of noise is annoying because the controller cannot observe the exact result to make decision. The noise (or the disturbance) can be generalized into two different types. Based on Fig3, one noise impacts the system and the other

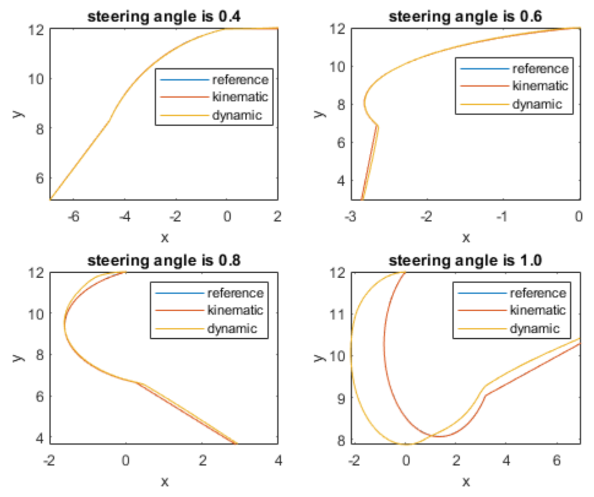


Fig. 11. Comparison between kinetic trajectory and dynamic trajectory with different reference (yellow) generated by the kinetic model with input speed is fixed at 1m/s but input angle is 0.4, 0.6, 0.8, 1.0. We get four reference and implement the kinetic trajectory like previous case(blue) and new dynamic trajectory (red).

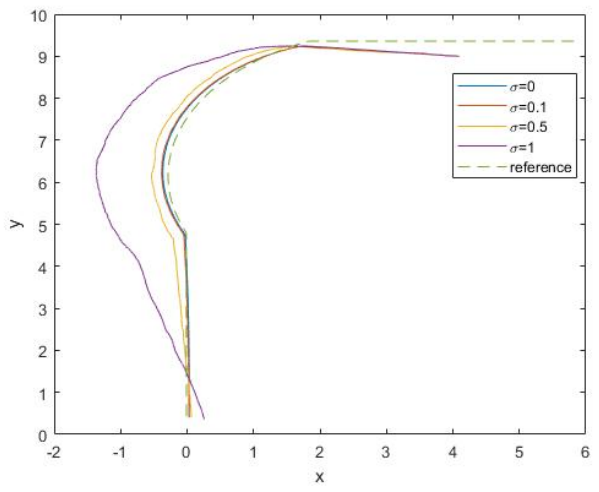


Fig. 12. the severity of the sensor noise's for the MPC control. σ is the standard deviation of the position sensor. The control is harder when the sensor noise is large.

interfere the sensor. In other words, the noise in the system makes difference in future while the sensor noise does not. Although the Kalman filter could solve both noise, we are more interested in the sensor noise rather than the system disturbance in the parking problem.

In fact, the sensor noise is very common in practice. For example, the location provided by GPS system or the radar has the minimum precision. If we use located the vehicle by these sensor, we have to consider the annoying noise. In Fig12, to illustrate the importance of the noise, we use the same MPC controller with different sensors (noise $\sigma = 0, 0.1, 0.5, 1.0$)

We have introduced how EKF works theoretically. Since the noise is random, we can see the trajectory varies significantly

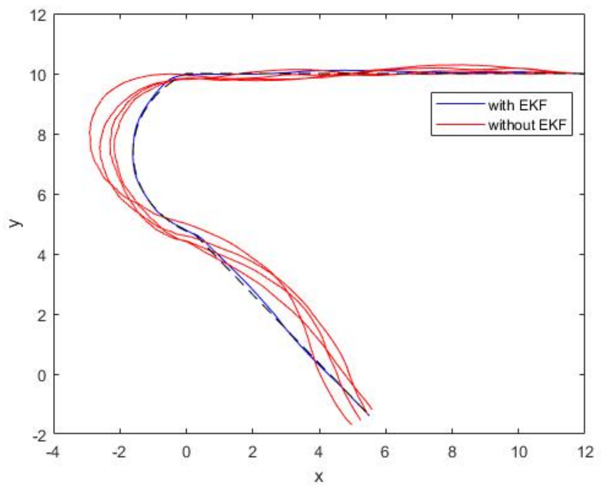


Fig. 13. The comparison between the controller combining MPC and EKF (blue dash line) and ones without EFK(four red dash lines). The noise is set $\sigma = 0.7$.

even under the same condition. For the noise, we generate zero mean Gaussian white noise with given variance. Note the variance of the noise must be known for EKF as covariance matrix as follows. We primarily discuss the position noise due the radar or GPS so we set $\sigma_x^2 = \sigma_y = 0.7$.

$$R_w = \text{diag}(0.49\ 0.49\ 0.01\ 0.01\ 0.0001\ 0.0001) \quad (14)$$
$$R_v = \text{diag}(0\ 0\ 0\ 0.0001\ 0.0001\ 0.0001) \quad (15)$$

For reference, we set the velocity at 2m/s in straight line and 1m/s in the curve line, where the steering angle is 0.8. The parameters of MPC keeps the same as the previous cases. The result in Fig13 shows that EKF is so powerful that only makes little deviation (blue dash line). As the comparison, we generates four trajectories (red dash lines) without EKF. Note the noise is random, so we observe four different red curves even for the same condition.

VI. Simulink

We used MPC control toolbox in Simulink to build car parking model and do simulation. For reference signal, we used Driving Scenario Designer to design the road and generate desired trajectory.

Matrices of time, position and yaw angle is included in the reference input signal. After receiving signals from reference and car plant model, MPC controller outputs steering angle to plant. The detailed structure is shown as following.

However, during the process we found that car plant model is restricted in time invariant system, while in our project the plant model is actually time varying. Therefore, the controlled parameters and flexibility is limited in Simulink. Instead we wrote Matlab function directly to do car parking control.

VII. Conclusion

For this project we developed different approaches to test the effectiveness of MPC controller. Both kinematic and

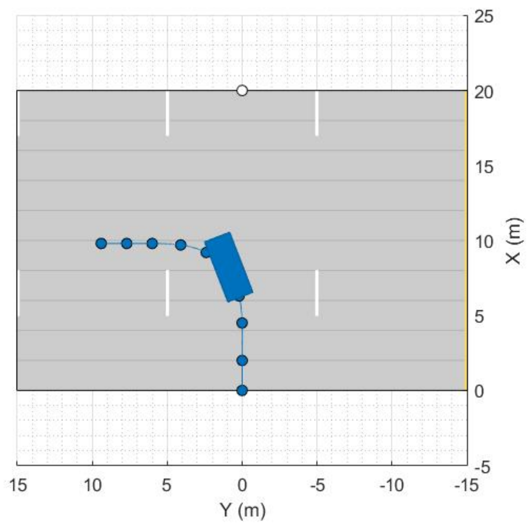


Fig. 14. Parking Scenario Design

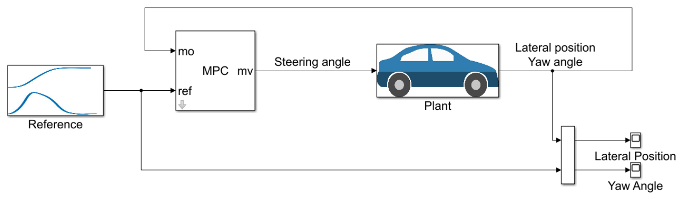


Fig. 15. Simulink Model Structure

dynamic models are compared and discussed. We used Matlab to do system modeling and MPC controller simulation. For kinematic model, the error when vehicle departed from expected region is smaller than 0.1m, which is expected from proposal. Time cost for total process is 18 seconds, which is much smaller than 2 minutes. Based on kinematic model, we create dynamic model and apply noise rejection using Kalman filter. The result is better under multiple conditions for dynamic model.

VIII. Acknowledgment

We thank for values of the parameters for dynamics model provided by Liniger group[2]. We also thank for the course ROB535 Self Driving Cars: Perception and Control, instructed by Professor Matthew Johnson Roberson and Professor Ram Vasudevan. Our code of MPC is based on Homework of ROB535.

References

[1] A. Hero, "Statistical methods for signal processing," 02 2008.
[2] A. Liniger, A. Domahidi, and M. Morari, "Optimization-based autonomous racing of 1: 43 scale rc cars," *Optimal Control Applications and Methods*, vol. 36, no. 5, pp. 628–647, 2015.
[3] "Park assist," 2019, [Online; accessed April 23, 2019]. [Online]. Available: https://www.landrover.com/ownership/incontrol/park-assist.html