CONTROLLER DESIGN

Yin Chheanyun

July 2024

Contents

List of Figures

List of Tables

1 PID CONTROLLER

1.1 What is PID Controller?

A PID-Controller is a widely used feedback control mechanism in engineering and industrial appli- cations. It is employed to regulate processes and systems by continuously adjusting an output based on the difference between a desired setpoint and the measured process variable.

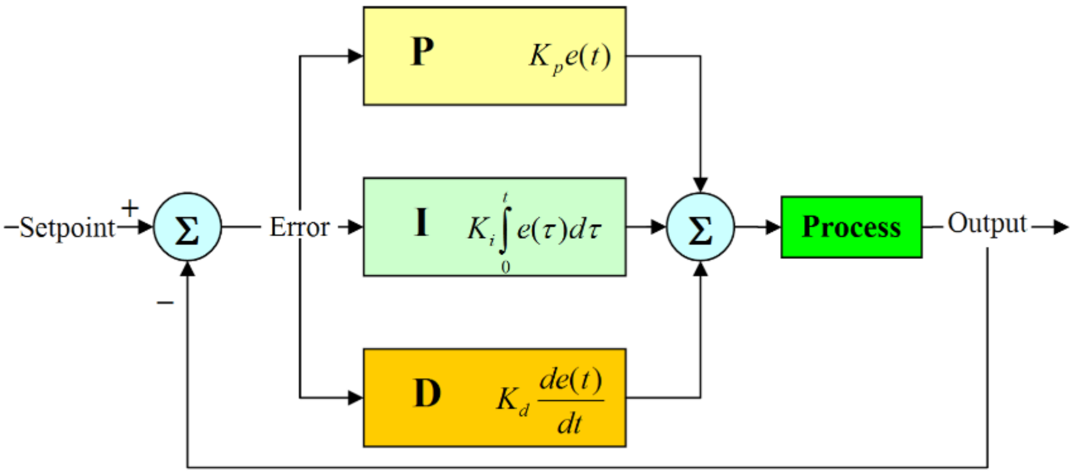

Figure 1: PID Controller Block diagram

1.2 PID Algorithm

- **Step1: Calculate Error**
  Error = Reference - Measurement.
- **Step2:Calculate P,I,D**
  Proportional = Kp*Error_cur
  Integral = Integral_prev + Ki*(Error_cur + Error_prev)*sampling_time
  Derivative = Kd*(Error_cur - Error_prev)/sampling_time
  where kp,ki,kd (you can tuning on Modelling thought MATLAB Simulink or tuning by hand(observe)
- **Step3:Calculate Output**
  Output = Proportional + Integral + Derivative.

1.3 Modifications to the algorithm

1.3.1 Integral Windup

Integrator windup or reset windup, refers to the situation in a PID controller where a large change in setpoint occurs (say a positive change) and the integral term accumulates a significant error during the rise (windup), thus overshooting and continuing to increase as this accumulated error is unwound (offset by errors in the other direction). The specific problem is the excess overshooting.
How to use Integral windup:

- Set Integral Min, Integral Max.
- When you use Integral winup:
  $if(Integral >= IntegralMax)$
  $\{Integral = IntegralMax;$

  $\}$
  $elseif(Integral <= IntegralMin)$
  $\{Integral = IntegralMin;$

  $\}$

```
$else\{Integral = Integral;$
$\}$
```

1.3.2 Low pass filter

A low-pass filter in a PID (Proportional-Integral-Derivative) controller is a component that filters out high-frequency noise from the derivative (D) term of the PID controller's output. The purpose of the low-pass filter is to smooth the derivative term and reduce its sensitivity to rapid, high-frequency changes in the error signal. By doing so, the controller can better distinguish between genuine changes in the process variable and short-term noise, resulting in more stable and accurate control.
How to use Low pass filter in PID:

- Derivative of Positional Error after low pass filter:

$$\boldsymbol{E(t) = (1-\alpha) * E(t-1) + \alpha * \frac{dError(t)}{dt}}$$

- α Smoothing Factor (smaller for greater smoothing) $(0 < \alpha < 1)$

1.3.3 Output Saturation

This objective is achieved by including a saturation block that ensure the output is always in the range $[OUT_{min}, OUT_{max}]$
How to use Output saturation:

- Output = Propotional + Integral + Derivative.

```
$if(Output >= Outputmax)\{$
.  $Output = Outputmax;$
$\}$
$elseif(Output <= Outputmin)\{$
.  $Output = Outputmin;$
$\}$
$else\{$
.  $Output = Output;$
$\}$
```

1.4 PID Control on DC Motor

1.5 PID Control on Position Robot

2 PURE PURSUIT

2.1 What is Pure pursuit?

The pure pursuit controller is an automatic steering method for wheeled mobile robots. It is a steering method, which means it computes the necessary angular velocity for a wheeled robot to stay on pre-computed paths. Linear velocity is assumed to be constant. Therefore, an additional velocity controller of your choice is needed if you wish to slow down the robot as it approaches the target (something as simple as a proportional controller could work).
_**Look Ahead Distance**: It is property is the main tuning property for the controller.

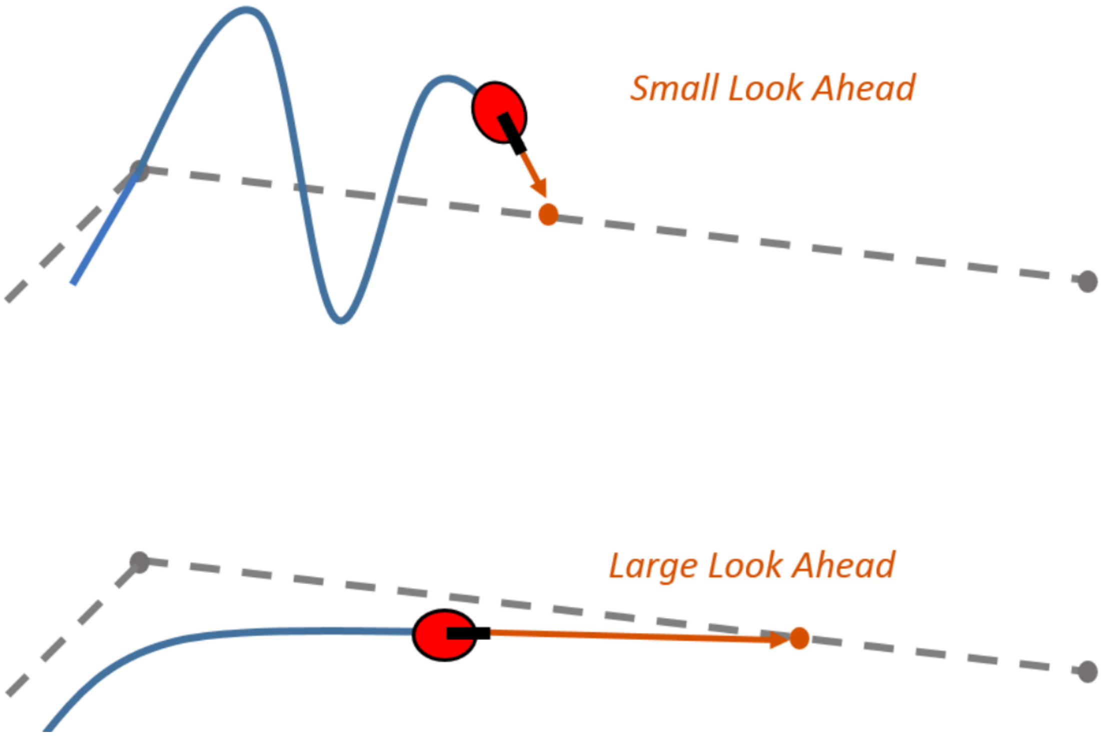

Figure 2: Look Ahead of Pure pursuit

_**Limitations**=There are a few limitations to note about this pure pursuit algorithm:

- As shown above, the controller cannot exactly follow direct paths between way-points. Parameters must be tuned to optimize the performance and to converge to the path over time.
- This pure pursuit algorithm does not stabilize the robot at a point. In your application, a distance threshold for a goal location should be applied to stop the robot near the desired goal.

2.2 Pure pursuit Algorithm

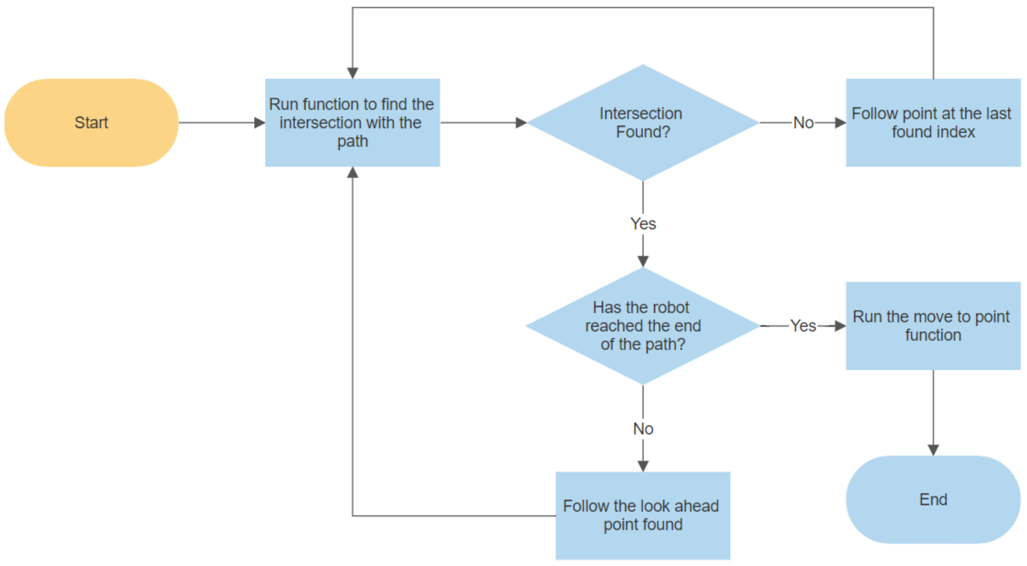
Start
Run function to find the intersection with the path
Intersection Found?
No
Follow point at the last found index
Yes
Has the robot reached the end of the path?
Yes
Run the move to point function
No
Follow the look ahead point found
End

Figure 3: Step work of Pure pursuit

- **Line-Circle Intersection**

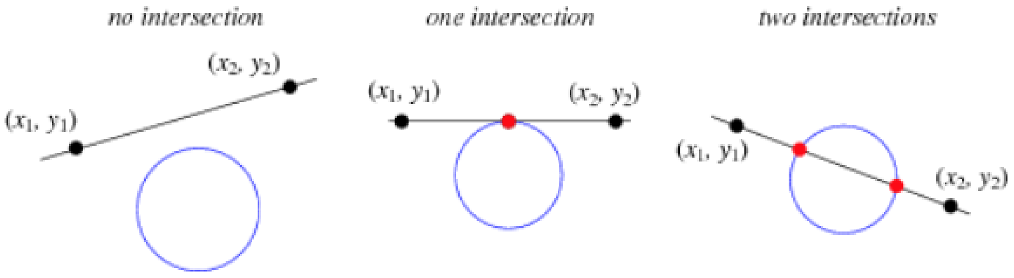

Figure 4: Step work of Pure pursuit

Ingeometry, a line meeting a circle in exactly one point is knows as a tangent line,while a line meeting a circle in exactly two points in known as a section line.

– Defining:

$$\mathbf{d_x = x_2 - x_1,\quad d_y = y_2 - y_1;\quad d_r = \sqrt{d_x^2 + d_y^2}}$$

$$\boldsymbol{D = \begin{vmatrix} x_1 & x_2 \\ y_1 & y_2 \end{vmatrix} = x_1 y_2 - y_1 x_2}$$

– Gives the points of intersection as:

$$\boldsymbol{X = \frac{D d_y \pm sgn * (dy) dx \sqrt{r^2 d_r^2 - D^2}}{d_r^2}}$$

$$\boldsymbol{Y = \frac{D d_x \pm |dy| dx \sqrt{r^2 d_r^2 - D^2}}{d_r^2}}$$

– The function sgn*(x) is defined as:

$$\mathbf{sgn} * (\mathrm{x}) = \begin{cases} -1 & \text{for} \quad x < 0 \\ 1 & otherwise \end{cases}$$

– The discriminant:

$$\boldsymbol{\Delta = l^2 d_r^2 - D^2}$$

Where: l is look ahead.
(**$\boldsymbol{\Delta < 0}$: nonintersection; $\boldsymbol{\Delta = 0}$: tangent; $\boldsymbol{\Delta > 0}$: intersection**

- **Calculate linear**

$$\mathbf{linearError} = \sqrt{(\mathbf{targety} - \mathbf{currenty})^2 + (\mathbf{targetx} - \mathbf{currentx})^2}$$

- **calculate absTargetAngle**

$$\mathbf{absTargetAngle} = \mathbf{atan2}(\mathbf{targety} - \mathbf{currenty}, \mathbf{targetx} - \mathbf{currentx})$$

if absTargetAngle ¡ 0 : absTargetAngle +=2π

- **Compute turn error**

$$\textbf{turnError = absTargetAngle - currnetHeading}$$

```
if $turnError > \pi \quad or \quad turnError < -\pi$:
.    turnError = -1 * sgn(turnError) * (2$\pi$ - abs(turnError)
```

- **Apply propotional controller**

linearVel = Kp_linear * linearError
turnVel = Kp_turn*turnError

3 STANELY CONTROL

3.1 What is Stanely Control?

Stanley control is a path-tracking algorithm widely used in autonomous vehicle navigation. Developed at Stanford University, the Stanley control algorithm is particularly known for its application in the DARPA Grand Challenge, where it successfully guided the Stanford Racing Team's autonomous vehicle to victory.

- Use the center of the front axle as a reference point.
- Look at both the error in heading and the error in position relative to the closet point on the path.
- Define an intuitive steering law to:
  -Correct heading error
  -Correct position error
  -obeymax steering angle bounds

3.2 Stanely Control Algorithm

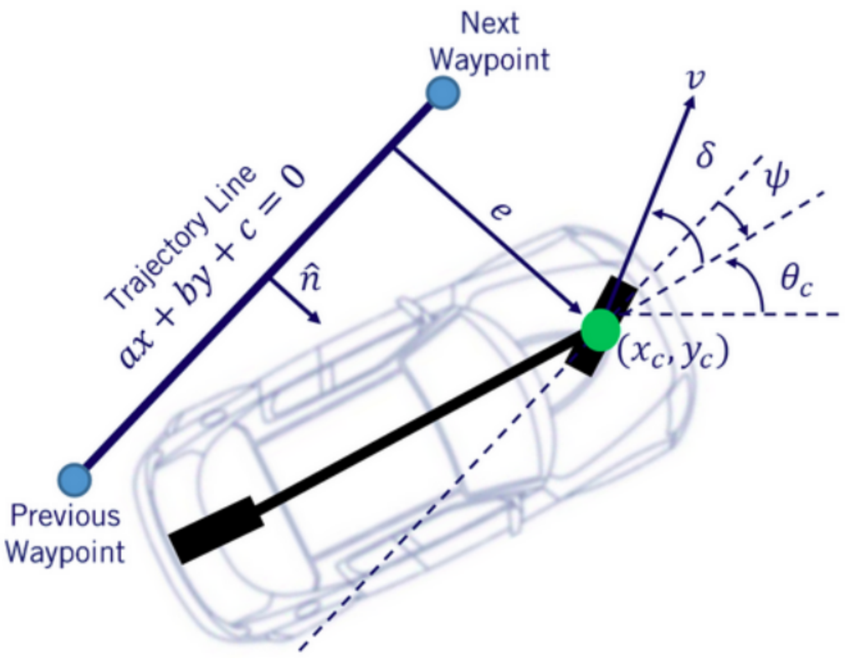

Figure 5: Stanely Control

ψ is the angle between the trajectory heading and the vehicle heading. The steering angle is denoted as δ.There are three intuitive steering laws of Stanley method:

- **Cross track error:**

$$e = \frac{ax_c + by_c + c}{\sqrt{a^2 + b^2}}$$

- **Cross track Steering:**

$$\delta = tan^{-1}(\frac{Ke(t)}{V(t)})$$

- **Heading error:**

$$\psi = tan^{-1}\frac{-a}{b} - \theta_c$$

- **Total steering Input:**

$$\delta = tan^{-1}(\frac{Ke(t)}{V(t)} + \psi)$$

4 LQR CONTROLLER

4.1 What is LQR control?

- LQR is a method that calculates the optimal feedback gain K. The feedback gain can determined by tuning of Q and R. The feedback gain is used to control the system in control signal form.
- The main focus of an LQR control system is to achieve and hold a target configuration for a given linear system using an optimal control law.
- An LQR control system generates the control law using four matrices:
  – A Matrix:physical dynamics
  – B Matrix:control dynamics
  – Q Matrix:state cost
  – R Matrix:control cost
- LQR's goal is to minimize the "cost" of the state error, while minimizing the "cost" of actuator effort.
- LQR determines the point along the cost curve where the "cost function" is minimized. . . thereby balancing both objectives of keeping state error and actuator effort minimized.

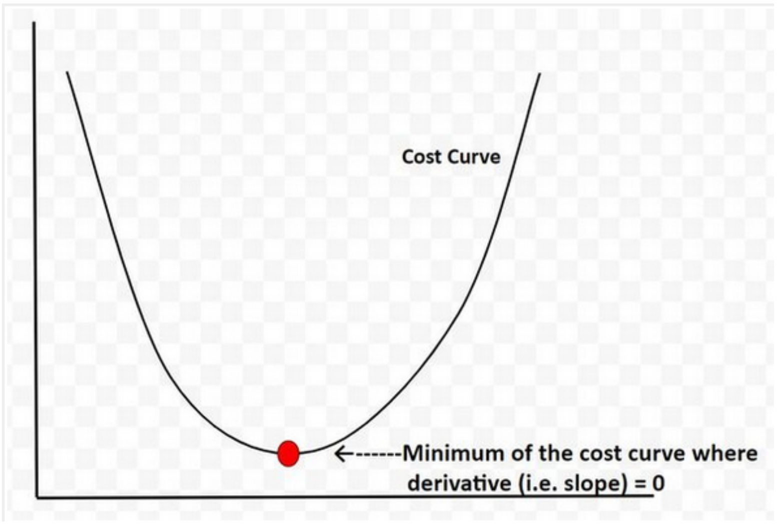

Figure 6: Cost curve of LQR

4.2 LQR Algorithm

- **Setting up the optimization problem**
  - **Positive Semidefinite**

$$\mathbf{X^T Q X >= 0}$$

  - **Positive definite**

$$\mathbf{U^T R U > 0}$$

  - If Q "is bigger" than R => fast regulation of X− >0,U is Large
  - If R "is bigger" than Q => slow regulation of X− >0,U is Small.
- **Solving the optimization problem**
  We are using one of the optimal control method which is Linear Quadratic Regulator. Minimize the cost:

$$\mathbf{J = \frac{1}{2}\sum_{k=0}^{N}(X_{k+1}^T Q X_{k+1} + U_k^T R U_k)} \quad \text{(Discrete-time)}$$

  OR

$$\mathbf{J = \int_0^\infty (X^T Q X + U^T R U) dt} \quad \text{(Continuous-time)}$$

- **Mathematics Model**

$$\mathbf{X_t = A_{t-1}X_{t-1} + B_{t-1}U_{t-1}}$$

The General form of the linearized model Differential Wheel:

$$\mathbf{X_t} = \begin{bmatrix}1 & 0 & 0\\0 & 1 & 0\\0 & 0 & 1\end{bmatrix}\begin{bmatrix}x_{t-1}\\y_{t-1}\\\theta_{t-1}\end{bmatrix} + \begin{bmatrix}cos(\theta_{t-1})dt & 0\\sin(\theta_{t-1})dt & 0\\0 & dt\end{bmatrix}\begin{bmatrix}v_{t-1}\\\omega_{t-1}\end{bmatrix}$$

- **Finite-horizon discrete-time**

$$\mathbf{J = X_{H_p}^T Q_{H_p} X_{H_p} + \sum_{k=0}^{H_p-1}(X_k^T Q X_k + U_k^T R U_k + 2X_k^T N U_k)}$$

Where: H_p: is the time horizon, N: is number step horizon, k: is time current.
from terminal condition $P_{H_p} = Q_{H_p}$.
Note that U_{H_p} is not defined, since x is driven to its final state X_{H_p} by $AX_{H_p-1} + BU_{H_p-1}$.

  - P_k is found iteratively backwards in time by the dynamic Riccati equation:

$$\mathbf{P_k = A^T P_k A - (A^T P_k B + N)(R + B^T P_k B)^{-1}(B^T P_k A + N^T) + Q}$$

  - Optimal feedback gain:

$$\mathbf{F_k = (R + B^T P_{k+1} B)^{-1}(B^T P_{k+1} A + N^T)}$$

  - Optimal control sequence minimizing the performance index is given by:

$$\mathbf{u_k = -F_k X_k}$$

- **Infinite-horizon, discrete-time**

$$\mathbf{J = \sum_{k=0}^{\infty}(X_k^T Q X_k + U_k^T R U_k + 2X_k^T N U_k)}$$

  - P is the unique positive definite solution to the discrete time algebraic Riccati equation (DARE):

$$\mathbf{P = A^T P A - (A^T P B + N)(R + B^T P B)^{-1}(B^T P A + N^T) + Q}$$

  - Optimal feedback gain:

$$\mathbf{F = (R + B^T P B)^{-1}(B^T P A + N^T)}$$

  - Optimal control sequence minimizing the performance index is given by:

$$\mathbf{u_k = -F X_k}$$

- **Note When you use to write in program is using (Finite-horizon discrete-time).**

5 NONLINEAR MODEL PREDICTIVE CONTROL (NMPC) CONTROLLER

5.1 What is MPC and NMPC Controller?

Model Predictive Control is one of the advanced technique control process that can obtained an optimal solution over a period with finite and infinite horizon. Model Predictive Control(MPC) or Receding Horizontal Control (RHC) is also an original framework of feedback control of the nonlinear dynamic system. Nonlinear Model Predicted Control(NMPC) is a variant of model predictive control that is characterized by the use of nonlinear system models in the prediction.

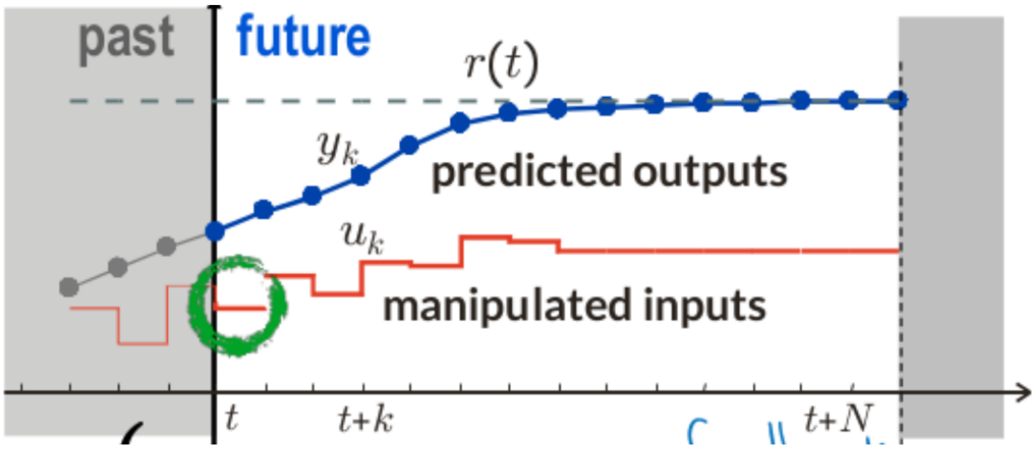

Figure 7: MPC Prdicted output

5.2 NMPC Algorithm Discrete-time

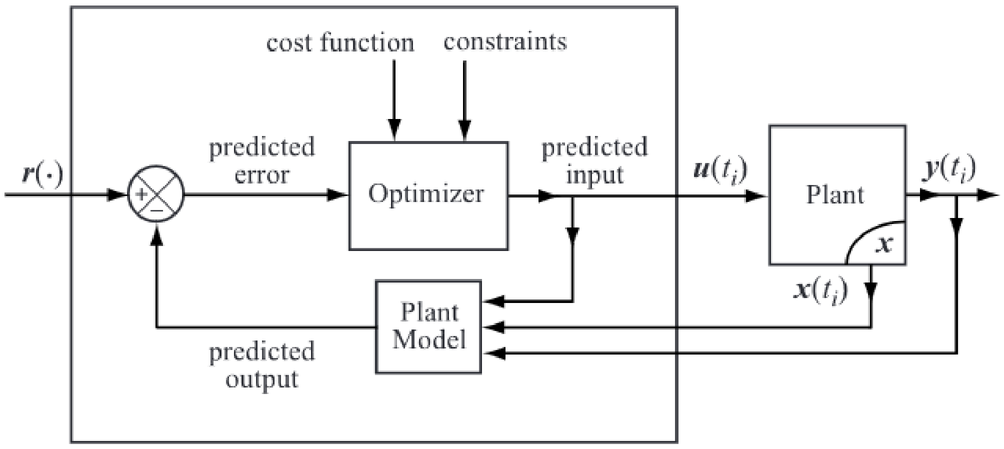

Figure 8: Block diagram of Model Predicted Controller

- **Linear prediction model:**
  - State Space model: $\mathbf{X_{k+1} = AX_k + BU_k}$ (eq.1)
  - Observation model: $\mathbf{Y_k = CX_k}$ (eq.2)
- **The optimal control problem (quadratic performance index):**
  - Backward difference State : $\mathbf{\Delta X_{k+1} = X_{k+1} - X_k}$
  - Backward difference Input : $\mathbf{\Delta U_{k+1} = U_{k+1} - U_k}$
    And given from (eq.1): $\mathbf{\Delta X_{k+1} = A\Delta X_k + B\Delta U_k}$ (eq.3)
  - Backward difference Output : $\mathbf{\Delta Y_k = Y_k - Y_{k-1}}$
    And given from (eq.2): $\mathbf{\Delta Y_{k+1} = C\Delta X_{k+1} = CA\Delta X_k + B\Delta U_k}$
    => $Y_{k+1} = Y_k + CA\Delta X_k + B\Delta U_k$ (eq.4)
    We Combine Equation(3) and (4):

$$\begin{bmatrix}\boldsymbol{\Delta X_{k+1}}\\ \boldsymbol{Y_{k+1}}\end{bmatrix} = \begin{bmatrix}\boldsymbol{A} & \boldsymbol{0}\\ \boldsymbol{CA} & \boldsymbol{I_p}\end{bmatrix}\begin{bmatrix}\boldsymbol{\Delta X_{k+1}}\\ \boldsymbol{Y_k}\end{bmatrix}\begin{bmatrix}\boldsymbol{B}\\ \boldsymbol{CB}\end{bmatrix}\boldsymbol{\Delta U_k}$$

We now define the augmented state vector : $\mathbf{Xa_k} = \begin{bmatrix}\Delta X_k\\ Y_k\end{bmatrix}$

Let: $\boldsymbol{Y_k} = \begin{bmatrix}\mathbf{0} & \boldsymbol{I_p}\end{bmatrix}\begin{bmatrix}\boldsymbol{\Delta X_k}\\ \boldsymbol{Y_k}\end{bmatrix}, \boldsymbol{A_a} = \begin{bmatrix}\boldsymbol{A} & \mathbf{0}\\ \boldsymbol{CA} & \boldsymbol{I_p}\end{bmatrix}, \boldsymbol{B_a} = \begin{bmatrix}\boldsymbol{\Delta X_{k+1}}\\ \boldsymbol{Y_k}\end{bmatrix}, \boldsymbol{C_a} = \begin{bmatrix}\mathbf{0} & \boldsymbol{I_p}\end{bmatrix}$

– **Update Model predicted**:

$$\mathbf{Xa_{k+1}} = \mathbf{A_a Xa_k} + \mathbf{B_a \Delta U_k} \quad (eq.5)$$

$$\mathbf{Y_k} = \mathbf{C_a Xa_k} \quad (eq.6)$$

– **State error:** $\mathbf{E_k} = \mathbf{r_{p_k}} - \mathbf{Y_k}$

$$\min_{\mathbf{z}} \mathbf{E_N^T P E_N} \sum_{\mathbf{k=0}}^{\mathbf{N-1}} \mathbf{E_k^T Q E_k} + \mathbf{\Delta U_k^T R \Delta U_k}$$

Subject to :

$$\mathbf{X_{min}} \le \mathbf{X_k} \le \mathbf{X_{max}}, \quad \mathbf{k} = [\mathbf{0}, \mathbf{N}]$$

$$\mathbf{U_{min}} \le \mathbf{U_k} \le \mathbf{U_{max}}, \quad \mathbf{k} = [\mathbf{0}, \mathbf{N}-\mathbf{1}]$$

Where: $R = R^T > 0; Q = Q^T \ge 0; P = P^T \ge 0; z = \begin{bmatrix}u_0 & u_1 & \dots & u_{N-1}\end{bmatrix}$

– $\mathbf{r_p}$ = the reference state of the system or desired input state.

– $\mathbf{N}$ = number prediction horizon.

- **For update state has two methode:**

$$\dot{\mathbf{X}} = \mathbf{f}(\mathbf{X}, \mathbf{U}(\mathbf{t}))$$

Example model of mecanum wheel:

$$\begin{bmatrix}\boldsymbol{V_x}\\ \boldsymbol{V_y}\\ \boldsymbol{\omega_z}\end{bmatrix} = \frac{\boldsymbol{r}}{\mathbf{4}}\begin{bmatrix}\boldsymbol{cos(\theta)} & \boldsymbol{-sin(\theta)} & \mathbf{0}\\ \boldsymbol{sin(\theta)} & \boldsymbol{cos(\theta)} & \mathbf{0}\\ \mathbf{0} & \mathbf{0} & \mathbf{1}\end{bmatrix}\begin{bmatrix}\mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1}\\ \mathbf{-1} & \mathbf{1} & \mathbf{1} & \mathbf{-1}\\ -\frac{1}{l_x+l_y} & \frac{1}{l_x+l_y} & -\frac{1}{l_x+l_y} & \frac{1}{l_x+l_y}\end{bmatrix}\begin{bmatrix}\boldsymbol{U_1}\\ \boldsymbol{U_2}\\ \boldsymbol{U_3}\\ \boldsymbol{U_4}\end{bmatrix}$$

– Euler Discretization:

$$\begin{bmatrix}\boldsymbol{x_{k+1}}\\ \boldsymbol{y_{k+1}}\\ \boldsymbol{\theta_{k+1}}\end{bmatrix} = \begin{bmatrix}\boldsymbol{x_k}\\ \boldsymbol{y_k}\\ \boldsymbol{\theta_k}\end{bmatrix} + \boldsymbol{\Delta T}\begin{bmatrix}\boldsymbol{V_{x_k}}\\ \boldsymbol{V_{y_k}}\\ \boldsymbol{\omega_{z_k}}\end{bmatrix}$$

– Runge-Kutta 4th (RK4):

$$\boldsymbol{k_1 = f(X, U_k)}$$

$$\mathbf{k_2} = \mathbf{f}(\mathbf{X_k} + \frac{\mathbf{\Delta T}}{\mathbf{2}}\mathbf{k_1}, \mathbf{U_k})$$

$$\mathbf{k_3} = \mathbf{f}(\mathbf{X_k} + \frac{\mathbf{\Delta T}}{\mathbf{2}}\mathbf{k_2}, \mathbf{U_k})$$

$$\mathbf{k_4} = \mathbf{f}(\mathbf{X_k} + \mathbf{\Delta T k_3}, \mathbf{U_k})$$

$$\mathbf{X_{k+1}} = \mathbf{X_k} + \frac{\mathbf{\Delta T}}{\mathbf{6}}(\mathbf{k_1} + \mathbf{2k_2} + \mathbf{2k_3} + \mathbf{k_4})$$

- **Nonlinear Program Problem(NLP):A standard problem formulation in numerical optimization.** Problem decision variables :

  – Single shooting: $\omega = [u_0, u_1, ..., u_{N-1}]$

  – Multiple shooting: $\omega = [x_0, x_1, ..., x_N, u_0, u_1, ..., u_{N-1}]$

$$\min_{\boldsymbol{\omega}} \boldsymbol{\Phi(F(\omega, X_N), \omega)}$$

subject to

$$\boldsymbol{g_1(F(\omega, X_N), \omega) \le 0} \quad \text{Inequality constraints}$$

$$\boldsymbol{g_2(F(\omega, X_N), \omega)} = \begin{bmatrix}\boldsymbol{\bar{X}_0 - X_0}\\ \boldsymbol{f(X_0, U_0) - X_1}\\ \vdots\\ \boldsymbol{f(X_{N-1}, U_{N-1}0 - X_N}\end{bmatrix} = \mathbf{0} \quad \text{Equality constraints}$$

6 MODEL PREDICTIVE PATH INTEGRAL (MPPI) CONTROLLER

6.1 Introductiion

MPPI is a variant of MPC that uses stochastic optimization to compute the control actions. Instead of solving a deterministic optimization problem, At MPPI, we consider two primary features that affect the controller's performance.

- **Trajectory** refers to the sequence of states that a system passes through over time, given a sequence of control inputs. In the context of MPPI, multiple trajectories are sampled to explore different possible future behaviors of the system. In MPPI, multiple such trajectories are sampled to explore the state and control space. The idea is to evaluate the cost associated with each trajectory and then use this information to determine the optimal control action.
- **Horizon** often referred to as the prediction or planning horizon, is the number of time steps over which future trajectories are considered in the optimization problem. It defines how far into the future the controller looks when making decisions.

6.2 MPPI Algorithm

6.2.1 PRIMARY

While our primary focus is on aggressive autonomous driving, the model predictive control algorithm that we develop is applicable to many other tasks.

- Consider a general discrete time, continuous state-action dynamical system of the form:

$$\boldsymbol{X_{t+1} = F(X_t, v_t)} \qquad (eq.1)$$

  Where:

  - X_t is the state of the system at time t
  - v_t is the input to the system at time t

- We assume that we do not have direct control over the input variable, vt, but rather that v_t is a random vector generated by a white noise process with density function:

$$\boldsymbol{v_t \backsim N(u_t, \Sigma)}$$

- Suppose that we are given a sequence of inputs:

$$\boldsymbol{(v_0, v_1, \ldots, v_{T-1}) = V \in R^{m\times T}}$$

- Corresponding sequence of mean input variables:

$$\boldsymbol{(u_0, u_1, \ldots, u_{T-1}) = U \in R^{m\times T}}$$

- We can then define the probability density functions for V as:

$$\boldsymbol{q(V|U,\Sigma) = \prod_{t=0}^{T-1} Z^{-1} exp\Big(-\frac{1}{2}(v_t-u_t)^T\Sigma^{-1}(v_t-u_t)\Big)}$$

$$\boldsymbol{= Z^{-T} exp(\Big(-\frac{1}{2}\sum_{t=0}^{T-1}(v_t-u_t)^T\Sigma^{-1}(v_t-u_t)\Big)} \qquad (eq.2)$$

  Where:$Z = ((2\pi)^m|\Sigma|)^{\frac{1}{2}}$
  So the density q(V —U, Σ) corresponds to the distribution $Q_{u,\Sigma}$.

- Given a running cost function, $L(x_t, u_t)$, and a terminal cost $\phi(x_T)$, we can define the discrete time optimal control problem as:

$$\boldsymbol{U^* = arg \min_{U\in\mu} E_{Q_U,\Sigma}\Big[\phi(x_T) + \sum_{t=0}^{T-1} L(x_t, u_t)\Big]} \qquad (eq.3)$$

  Where: μ is the set of admissible command sequences.

- We assume that the running cost can be split into an arbitrary state-dependent running cost, and a control cost which is a quadratic function of the system noise:

$$\boldsymbol{L(x_t, u_t) = c(x_t) + \frac{\lambda}{2}(u_t^T \Sigma^{-1} u_t + \beta_t^T u_t)} \qquad (eq.4)$$

  The affine term β allows for the location of the minimum control cost to be moved away from zero (although $\beta = 0$ is the standard case).

- Next denote $C(x_0, x_1, \ldots, x_T)$ as the portion of the cost of a trajectory that only depends on the state:

$$\boldsymbol{C(x_0, x_1, \ldots, x_T) = \phi(x_T) + \sum_{t=0}^{T-1} c(x_t)} \qquad (eq.5)$$

- For this we define the operator H which transforms an input sequence (along with an initial condition) into a resulting trajectory:

$$\boldsymbol{H(V; x_0) = (x_0, F(x_0, v_0), F(F(x_0, v_0), v1), \ldots).} \qquad (eq.6)$$

- Then the state-cost of an input sequence is defined as the functional composition:

$$\boldsymbol{S(V; x_0) = C(H(V; x_0))} \qquad (eq.7)$$

- The Free-Energy of a control system as:

$$\boldsymbol{F(S, p, x_0, \lambda) = log\Big(E_p\Big[exp(-\frac{1}{\lambda}S(V))\Big]\Big)} \qquad (eq.8)$$

  Where: $\lambda \in R^+$ is called the inverse temperature, P is some probability density over input sequences which we will refer to as the base probability, and p is the corresponding density.

- Then the KL-Divergence between F and H is:

$$\boldsymbol{D_{KL}(F||H) = E_F\Big[log\Big(\frac{f(V)}{h(V)}\Big)\Big]} \qquad (eq.9)$$

  The KL-Divergence provides a method for comparing distances between probability distributions and is therefore useful for defining optimization objectives.

6.2.2 INFORMATION-THEORETIC MODEL PREDICTIVE CONTROL

In this section we show how the definition of the free energy from (eq.8) can be used to derive a lower bound for the optimal control problem that we defined in (eq.3). This lower bound is subsequently used to create a sampling based model predictive control algorithm.

- We start by making the following observation:

$$\boldsymbol{F(S, p, x_0, \lambda) = log\Big(E_p\Big[exp(-\frac{1}{\lambda}S(V)\Big]\Big)}$$

$$\boldsymbol{= log\Big(E_p\Big[exp(-\frac{1}{\lambda}S(V)\frac{p(V)}{q(V|U, \Sigma)}\Big]\Big)}$$

  where the last equality follows from switching the expectation by using the standard importance sampling trick of multiplying by $1 = \frac{q(V|U,\Sigma)}{q(V|U)}$. Using the concavity of the logarithm, we can apply Jensen's inequality and obtain:

$$\boldsymbol{F(S, p, x_0, \lambda) \leq log\Big(E_p\Big[exp(-\frac{1}{\lambda}S(V)\frac{q(V|U, \Sigma)}{p(v)}\Big]\Big)} \qquad (eq.10)$$

- The right-hand side of this inequality can be simplified, using basic properties of the logarithm and the definition of the KL- Divergence, as:

$$\boldsymbol{= -\frac{1}{\lambda}E_{Q_{U,\Sigma}}\Big[S(V) + \lambda log(\frac{q(V|U, \Sigma)}{p(V)}\Big)}$$

$$\boldsymbol{= -\frac{1}{\lambda}\Big(E_{Q_{U,\Sigma}}[S(V)] + \lambda D_{KL}(Q_{U,\Sigma}||P)\Big)} \qquad (eq.11)$$

- Then multiplying each side by $-\lambda$ results in the following free energy lower bound:

$$-\lambda F(S,p,x_0,\lambda) \leq \Big(E_{Q_{U,\Sigma}}[S(V)] + \lambda D_{KL}(Q_{U,\Sigma}||P)\Big) \qquad (eq.12)$$

- Suppose we assign the base distribution as:

$$p(V) = q(V|\tilde{U},\Sigma) \qquad (eq.13)$$

Where: $\tilde{U}$ represents some nominal control input applied to the system

- Then the KL-Divergence between $Q_{U,\Sigma}$ and $Q_{\tilde{U},\Sigma}$ is

$$D_{KL}(Q_{U,\Sigma}||Q_{\tilde{U},\Sigma}) = \frac{1}{2}\sum_{t=0}^{T-1}(u_t - \tilde{u}_t)^T\Sigma^{-1}(u_t - \tilde{u}_t)$$

$$= \frac{1}{\lambda}\sum_{t=0}^{T-1}(u_t\Sigma^{-1}\tilde{u}_t - \tilde{u}_t^T\Sigma^{-1}u_t + \tilde{u}_t\Sigma^{-1}\tilde{u}_t) \qquad (eq.14)$$

Which if we set $\beta^T = -\tilde{u}_t^T\Sigma^{-1}$ and $c = \tilde{u}_t\Sigma^{-1}\tilde{u}_t$ we get:

$$D_{KL}(Q_{U,\Sigma}||Q_{\tilde{U},\Sigma}) = \frac{1}{\lambda}\sum_{t=0}^{T-1}(u_t\Sigma^{-1}\tilde{u}_t + \beta_t u_t + c_t) \qquad (eq.15)$$

- Substituting (15) and (4) into the RHS of (eq.10) and expanding we obtain

$$E_{Q_{U,\Sigma}}\Big[\phi(t) + \sum_{t=0}^{T-1}c(x_t) + \frac{\lambda}{2}(u_t + \beta_t^T u_t + c_t)\Big]$$

- This is clearly equivalent to the cost function in (3), allowing us to conclude:

$$E_{Q_{U,\Sigma}}\Big[\phi(t) + \sum_{t=0}^{T-1}L(x_t,u_t)\Big] \qquad (eq.16)$$

- **Optimal Distribution**
Define the optimal control density function Q^* as follows:

$$q*(V) = \frac{1}{\eta}exp(-\frac{1}{\lambda}S(V))p(V)$$

$$\eta = \int_{R^{m\times T}} exp(-\frac{1}{\lambda}S(V))p(V)d(V)$$

We will now show that this particular choice of distribution achieves the lower bound. Substituting Q^* into the KL-Divergence term from the RHS of (9) yields

$$D_{KL}(Q^*||P) = E_Q\Big[log\Big(\frac{\frac{1}{\eta}exp(-\frac{1}{\lambda}S(V))p(V)}{p(V)}\Big)\Big]$$

$$= -\frac{1}{\lambda}E_{Q^*}[S(V)] - log(\eta)$$

Substituting this divergence into (eq.10) results in:

$$-\lambda F \leq E_{Q^*}[S(V)] + E_{Q^*}[S(V)] - \lambda log(\eta)$$

Simplifying the RHS and substituting (15) we obtain:

$$-\lambda F \leq -\lambda log\Big(E_P\Big[exp(-\frac{1}{\lambda}S(V))\Big]\Big) \qquad (eq.16)$$

- **KL-Divergence Minimization**
  The goal of aligning the controlled distribution $Q_{U,\Sigma}$ with the optimal distribution Q^* can be achieved by minimizing the KL-Divergence:

$$\boldsymbol{U^* = arg \min_{U \in \mu} [D_{KL}(Q^* || Q_{U,\Sigma})]}$$

  we obtain the following quadratic minimization problem:

$$\boldsymbol{U^* = arg \min_{U \in \mu} \Big(E_{Q^*} \Big[\sum_{t=0}^{T-1} (v_t - u_t)^T \Sigma^{-1} (v_t - u_t) \Big] \Big)}$$

  In the unconstrained case ($\mu = R^m$), we can solve for ut to yield the optimal solution:

$$\boldsymbol{u_t^* = E_{Q^*}[v_t]} \qquad (eq.17)$$

- **Importance Sampling**
  We can use the technique of importance sampling to construct a set of samples that provide an unbiased estimate of the optimal control solution given a current control distribution.Given an initial estimate of the controls, denoted by $\hat{U}$, we have:

$$\boldsymbol{E_{Q^*}[v_t] = \int q^*(V) v_t dV}$$

$$\boldsymbol{= \int \frac{q^*(V)}{q(V|\hat{U}, \Sigma)} q(V|\hat{U}, \Sigma) v_t dV}$$

  This integral expression can be expressed as the following expectation:

$$\boldsymbol{E_{Q_{U,\Sigma}}[w(V) v_t], \quad w(V) = \frac{q^*(V)}{q(V|\hat{U}, \Sigma)}}$$

  This is done by using the base distribution p(V) as follows:

$$\boldsymbol{w(V) = \Big(\frac{q^*(V)}{p(V)}\Big)\Big(\frac{p(V)}{q(V|\hat{U}, \Sigma)}\Big)}$$

$$\boldsymbol{= \frac{1}{\eta} exp\Big(- \frac{1}{\lambda} S(V)\Big)\Big(\frac{p(V)}{q(V|\hat{U}, \Sigma)}\Big)}$$

  In the case that the base distribution takes the form as in (10), we have the following:

$$\boldsymbol{\frac{p(V)}{q(V|\hat{U}, \Sigma)} = exp\Big(- \frac{1}{2} \sum_{t=0}^{T-1} D + 2(\hat{u}_t - \tilde{u}_t)^T \Sigma^{-1} v_t \Big)}$$

$$\boldsymbol{D = \tilde{u}_t^T \Sigma_{-1} \tilde{u}_t - \hat{u}_t^T \Sigma^{-1} \hat{u}_t}$$

  This leaves us with the weight:

$$\boldsymbol{\frac{p(V)}{q(V|\hat{U}, \Sigma)} \propto exp\Big(- \sum_{t=0}^{T-1} (\hat{u}_t - \tilde{u}_t)^T \Sigma^{-1} v_t \Big)}$$

  This term encourages samples to move in the direction of the base distribution, $\tilde{u}_t - \hat{u}_t$. We then have:

$$\boldsymbol{w(V) = \frac{1}{\eta} exp\Big(- \frac{1}{\lambda} \Big(S(V) + \lambda \sum_{t=0}^{T-1} (\hat{u}_t - \tilde{u}_t)^T \Sigma^{-1} v_t \Big)\Big)} \qquad (eq.17)$$

$$\boldsymbol{u_t = E_{Q_{\hat{U},\Sigma}}[w(V) v_t]} \qquad (eq.18)$$

- **Practical Issues**
  However, there are a few practical issues to address before describing the full information theoretic model predictive control algorithm. These are (1) Shifting the range of the trajectory costs, (2) Decoupling the control cost and temperature, (3) Handling control constraints, (4) Smoothing the outputted solution, and (5) Sampling trajectories fast enough for online optimization

  1. **Shifting the range of the trajectory costs:**
     The negative exponentiation required by the importance sampling weight is numerically sensitive to the range of the input values. If the costs are too high then the negative exponentiation results in values numerically equal to zero, and if the costs are not bounded from below then the negative exponentiation can lead to overflow errors. For this reason we shift the range of the costs so that the best trajectory sampled has a value of 0. This simultaneously bounds the costs from below and ensures that at least one trajectory has an importance sampling weight which is not numerically zero. This is done as follows: first expand out the normalizing term η in (22) so that the importance sampling weight is:

$$\frac{exp\Big(-\frac{1}{\lambda}\Big(S(V)+\lambda\sum_{t=0}^{T-1}(\hat{u}_t-\tilde{u}_t)^T\Sigma^{-1}v_t\Big)\Big)}{\int exp\Big(-\frac{1}{\lambda}\Big(S(V)+\lambda\sum_{t=0}^{T-1}(\hat{u}_t-\tilde{u}_t)^T\Sigma^{-1}v_t\Big)\Big)dV}$$

     Now define ρ as the minimum cost (in the Monte-Carlo approximation it is the minimum sampled cost). We then multiply by:

$$1=\frac{exp(\frac{1}{\lambda}\rho}{\frac{1}{\lambda}\rho}$$

     which results in:

$$w(V)=\frac{1}{\tilde{\eta}}exp\Big(-\frac{1}{\lambda}\Big(S(V)+\lambda\sum_{t=0}^{T-1}(\hat{u}_t-\tilde{u}_t)^T\Sigma^{-1}v_t-\rho\Big)\Big)$$

$$\tilde{\eta}=\int exp\Big(-\frac{1}{\lambda}\Big(S(V)+\lambda\sum_{t=0}^{T-1}(\hat{u}_t-\tilde{u}_t)^T\Sigma^{-1}v_t-\rho\Big)\Big)dV$$

  2. **Decoupling control cost and temperature:**
     Consider the form of the importance sampling weight from (22) when we take the uncontrolled dynamics of the system as the base distribution:

$$w(V)=\frac{1}{\tilde{\eta}}exp\Big(-\frac{1}{\lambda}\Big(S(V)+\lambda\sum_{t=0}^{T-1}(\hat{u}_t)^T\Sigma^{-1}v_t\Big)\Big)$$

     Let $\hat{U}$ be the current planned control sequence, and define the new base distribution as:

$$\tilde{p}(U)=p(V|\alpha\hat{U},\Sigma)$$

     where 0 ¡ α ¡ 1. With $\alpha=0$, the base distribution reverts back to the uncontrolled dynamics and pushes U to zero. And with $\alpha=1$, the base distribution is the distribution corresponding to the current planned control law, which keeps U near the distribution corresponding to $\hat{U}$. The case of $\alpha=1$ can be interpreted as placing a cost on how much the new open loop control law is allowed to deviate from the previous one, which is useful for creating smooth motions.
     However, the control cost portion of the importance sampling weight now becomes:

$$\sum_{t=0}^{T-1}(1-\alpha)\hat{u}^T\Sigma^{-1}v_t$$

     We then have $\gamma=\lambda(1-\alpha)$ as the new control cost parameter,resulting in

$$w(V)=\frac{1}{\tilde{\eta}}exp\Big(-\frac{1}{\lambda}\Big(S(V)+\gamma\sum_{t=0}^{T-1}(\hat{u}_t)^T\Sigma^{-1}v_t\Big)\Big)$$

     this the final probability weighting for the algorithm.

3. **Handling Control Constraints:**
Most interesting control systems, including the autonomous vehicle we consider here, have actuator limits that the controller must take into account. There are several ways to do this in our information theoretic framework: rejection sampling for trajectories that violate the control constraints, or by formulating the optimization problem in (19) as a quadratic program and solving the constrained optimization problem
A simpler solution is to make the problem unconstrained by pushing the control constraints into the system dynamics
$$\boldsymbol{x_{t+1} = F(x_t, g(v_t))},$$
Where $g(v_t)$ is a clamping function that restricts vt to remain within an allowable input region. Since the sampling based update law does not require computing gradients or linearizing the dynamics, adding this additional non-linearity (and non-smooth) component into the dynamics is trivial to implement, and it works well in practice.

4. **Control Smoothing**
The stochastic nature of the sampling procedure can lead to significant chattering in the resulting control, which can be removed by smoothing the output control sequence.Consider the quadratic objective (19):
$$\boldsymbol{u_t^* = argmin\Big(E_{Q*}\Big[(v_t - u_t)^T(v_t - u_t)\Big]\Big)}$$
And now consider fitting a local polynomial approximation (at every timestep) so that $u_t = a_0 + a_1t + a_2t^2 + \ldots + a_kt^k = At$ Where $A = (a_0, a_1, \ldots + a_k)$ and $t = (1, t, t^2, \ldots, t^k)$ Our goal is to then find the optimal set of coefficients at each timestep. The optimal coefficients, at timestep j, can be found through the following optimization:
$$\boldsymbol{A_j^* = argmin\Big(E_{Q*}\Big[\sum_{t=(j-k}^{J+k}(v_t - A_t^T)\Sigma^{-1}(v_t - At)}$$
This in turn is equivalent to the minimization:
$$\boldsymbol{A_t = argmin\Big(\sum_{t=j-k}^{j+k}(E_{Q*}[v_t] - At)^T\Sigma^{-1}(E_{q*}[v_t] - At)\Big)}$$

5. **GPU-Based Trajectory Sampling**
The key requirement for applying our information theoretic framework in a model predictive control setting is the ability to generate and evaluate a large number of samples in real time. As in our prior sampling-based MPC methods [24], we perform sampling in parallel on a graphics processing unit (GPU) with Nvidia's CUDA architecture. In our implementation, all of the trajectory samples are processed individually in parallel. In addition to the sample level parallelism, each individual sample uses between 4 and 16 CUDA threads depending on the dynamics model. This is done in order to take advantage of the parallel nature of the linear algebra routines that our vehicle dynamics models rely on. Depending upon the model and cost function, our implementation can achieve control loops from 40-60 HZ using a few thousand samples of 2-3 second long trajectories. Note that sampling 1200, 2.5 second long trajectories at 40 Hz corresponds to making approximately 4.8 million queries to the full non-linear dynamics of the vehicle every second. For the complex vehicle dynamics that we consider, this is only possible using a modern GPU.

6.2.3 Summarize

- We applied the general noise assumption that holds that we could not directly control the system over v_t, but could over the mean u_t of the density function of noise e_t:

$$\boldsymbol{v_t \sim N(u_t, \Sigma)}$$
$$\boldsymbol{(u_0, u_1, \ldots, u_{T-1}) = U \in R^{m\times T}}$$
$$\boldsymbol{(v_0, v_1, \ldots, v_{T-1}) = V \in R^{m\times T}}$$

Noise is simply defined as $\boldsymbol{v_t = u_t + \epsilon_t}$.

- we can define the probability density function q(V) as:

$$\boldsymbol{q(V) = \prod_{t=0}^{T-1} Z^{-1} exp\Big(- \frac{1}{2}(v_t - u_t)^T \Sigma^{-1}(v_t - u_t)} \qquad (eq.1)$$

Where: $Z = ((2\pi)^m|\sigma|)^{\frac{1}{2}}$

- we can also define the uncontrolled density function p(V) where U is usually 0:

$$\boldsymbol{p(V) = \prod_{t=0}^{T-1} Z^{-1} exp\Big(- \frac{1}{2} v_t^T \Sigma^{-1} v_t\Big)} \qquad (eq.2)$$

These two density functions correspond to the distributions of Q and P, respectively.

- Corresponds to the optimal distribution Q^*:

$$\boldsymbol{q^*(V) = \frac{1}{\eta} exp\Big(- \frac{1}{\lambda} S(V)\Big) p(V)} \qquad (eq.3)$$

where η denotes the normalizing constant and S(V) denotes the state-dependent cost.

- The input sequence in state cost is iteratively transformed into state values x through the non affine system dynamics F.

$$\boldsymbol{S(V; x_0) = C(H(V; x_0))}$$

$$\boldsymbol{H(V; x_0) = (X_0, F(x_0, v_0), F(F(x_0, v_0), v_1), \ldots}$$

$$\boldsymbol{C(x_0, x_1, \ldots, x_T) = \phi(x_T) + \sum_{t=0}^{T-1} c(x_t)}$$

$$\boldsymbol{S(V; F) = \phi(x_T) + \sum_{t=0}^{T-1} c(x_t)} \qquad (eq.4)$$

$$\mathbf{x_{t+1} = F(x_t, v_t)}$$

- we can now derive the optimal control input by minimizing the KL divergence between Q and $Q*$:

$$\boldsymbol{E_{Q*}[v_t] = \int q*(V) v_t dV} \qquad (eq.5)$$

- Importance sampling is therefore employed to compute the integral over the known distribution Q:

$$\boldsymbol{E_{Q*}[v_t] = \int w(V) q(V) v_t dV = E_{Q_{U,\Sigma}}[w(V)] v_t} \qquad (eq.6)$$

where the importance weighting term is:

$$\boldsymbol{w(V) = \Big(\frac{q*(V)}{p(V)}\Big)\Big(\frac{p(V)}{q(V)}\Big) = \frac{1}{\eta} exp\Big(- \frac{1}{\lambda}\Big(S(V) + \frac{1}{2}\lambda \sum_{t=0}^{T-1} u_t^T \Sigma^{-1}(u_t + 2\epsilon_t)\Big)\Big)} \qquad (eq.7)$$

Let ε be a noise sequence $\epsilon_0, \epsilon_1, \ldots, \epsilon_{T-1}$ and K be the number of trajectory samples.

- Finally, we have the iterative optimal control update law to compute the weights when the sampled trajectory cost $C(V_0), C(V_1), ...C(V_{K-1})$ is given:

$$\boldsymbol{u_t^{i+1} = u_t^i + \sum_{k=0}^{k-1} w(\varepsilon^k)\epsilon_t^k} \qquad (eq.8)$$

$$\boldsymbol{w(\varepsilon^k) = \frac{1}{\eta} exp\Big(- \frac{1}{\lambda}\Big(C(V^k) - \beta\Big)\Big)} \qquad (eq.9)$$

where we subtract the minimum state cost β to ensure that at least one sample has a numerically non-zero importance sampling weight.

- The trajectory cost in (eq.9) is then simplified as follows:

$$\boldsymbol{C(V^k) = S(V^k) + \lambda \sum_{t=0}^{T-1} u_t^T \Sigma^{-1} \epsilon_t^k} \qquad (eq.10)$$

- calculate weight for each sample sequence:

$$\boldsymbol{w(v) = \frac{1}{\eta} exp\Big(- \frac{1}{\lambda}\Big(S(V) + \lambda(1-\alpha) \sum_{t=0}^{T-1} u_t^T \Sigma^{-1} (u_t + \epsilon_t) - \rho \Big)\Big)}$$

$$\boldsymbol{\eta = \sum_{k=1}^{K} exp\Big(- \frac{1}{\lambda}\Big(S(U + \varepsilon_k) + \lambda(1-\alpha) \sum_{t=0}^{T} u_t^T \Sigma^{-1} (u_t + \epsilon_t) - \rho \Big)\Big)}$$

$$\boldsymbol{\rho = \min_k \Big(S(V_k) + \lambda(1-\alpha) \sum_{t=0}^{T-1} u_t^T \Sigma^{-1} (u_t \epsilon_t^k) \Big)}$$

- Decoupling Control Space and Action Space:
  - Smoothing along the "i-axis": Henceforth, let us decouple the control space and action space by defining action sequence $A = a_0, a_1, ...a_{T-1}$.
  - Smoothing along the "t-axis": In contrast to MPPI, our control distribution corresponds to derivative action, allowing action variables to be treated as augmented state elements and hence making them independent of the control cost.
  - we introduce an extra action cost Ω to smooth the action sequence along the "t-axis" by minimizing the variance of A
  - without violating the information theoretic interpretation of MPPI:

    $$\boldsymbol{\Omega(A) = \sum_{t=1}^{T-1} (a_t - a_{t-1})^T \omega (a_t - a_{t-1})} \qquad (eq.11)$$

    where ω is the weighting parameter in the form of a diagonal matrix.
  - Finally, we obtain the following action sequence update law:

    $$\boldsymbol{u_t^{i+1} = u_t^i + \sum_{k=0}^{k-1} w(\varepsilon^k) \epsilon_t^k} \qquad (eq.12)$$

    $$\boldsymbol{a_t^{i+1} = a_t^i + u_t^{i+1} \Delta t} \qquad (eq.13)$$

  - The trajectory cost (eq.10) now takes the form:

    $$\boldsymbol{C(V^k) = S(V^k) + \Omega(A + V^k \Delta t) + \lambda \sum_{t=0}^{T-1} u_t^T \Sigma^{-1} \epsilon_t^k} \qquad (eq.14)$$

7 Adaptive Control

7.1 What is Adaptive Control?

Adaptive control is a type of control strategy in which the control system can adjust its parameters automatically in real-time to adapt to changes in the system or environment. This is particularly useful in situations where the system being controlled is subject to variations or uncertainties, such as changes in load, temperature, or other external conditions.

7.2 Application of Adaptive Control

- **Robotics:** Adaptive control is widely used in robotic systems to handle uncertainties and improve performance. For example, robotic manipulators can use adaptive control to compensate for payload variations, friction, and other disturbances, resulting in more precise and efficient motion.
- **Aerospace:** Adaptive control is used in aircraft and spacecraft to enhance stability, maintain performance under varying conditions, and accommodate system failures. For instance, adaptive control algorithms can be used to adjust the control surface deflections to maintain stability in the presence of structural damage or control surface failures.
- **Process Control:** In industries such as chemical, pharmaceutical, and food processing, adaptive control is employed to maintain product quality and optimize production processes. By continuously adjusting control parameters in response to changes in operating conditions, adaptive controllers can maintain consistent performance.

7.3 Adaptive Algorithm

Parameter adaptation algorithms generally have the following structure:

$$\begin{bmatrix} New\ estimated \\ parameters \\ (vector) \end{bmatrix} = \begin{bmatrix} Previous\ estimated \\ parameters \\ (vector) \end{bmatrix} + \begin{bmatrix} Adaptation \\ gain \\ (matrix) \end{bmatrix} \times \begin{bmatrix} Measurement \\ function \\ (vector) \end{bmatrix} \times \begin{bmatrix} Predictionerror \\ function \\ (scalar) \end{bmatrix}$$

7.3.1 Parameter Adaptation Algorithms Deterministic Environment

On-line estimation of the parameters of a plant model or of a controller is one of the key steps in building an adaptive control system
Several approaches can be considered for deriving parameter adaptation algorithms:

- heuristic approach;
- gradient technique;
- least squares minimization;
- stability;
- rapprochement with Kalman filter.

1. **Gradient descent algorithm**
   Consider the discrete-time model of a plant described by:

   $$\boldsymbol{y(t+1) = -a_1 y(t) + b_1 u(t) = \theta^T \phi(t)} \qquad (eq.1)$$

   Where:

   - Parameter vector $\theta = \begin{bmatrix} a \\ b \end{bmatrix}$
   - Measurement vector$phi(t) = \begin{bmatrix} -y(t) \\ u(t) \end{bmatrix}$

   The Priori predicted output:

   $$\boldsymbol{y^0(t+1) = \hat{y}(t+1|\hat{\theta}(t)) = \hat{\theta}^T(t)\phi(t)} \qquad (eq.2)$$

The adjustable prediction model will be described in this case by:

$$\hat{y}(t+1) = -\hat{a_1}y(t) + \hat{b_1}u(t) = \hat{\theta}^T(t+1)\phi(t) \qquad (eq.3)$$

Priori prediction error as:

$$e^0(t+1) = y(t+1) - \hat{y}^0(t+1) \qquad (eq.4)$$

Posterior prediction error as:

$$e(t+1) = y(t+1) - \hat{y}(t+1) \qquad (eq.5)$$

The objective is to find a recursive parameter adaptation algorithm with memory.
The structure of such an algorithm is:

$$\hat{\theta}(t+1) = \hat{\theta}(t) + \Delta\hat{\theta}(t+1) = \hat{\theta}(t) + f[\hat{\theta}(t), \phi(t), e(t+1)] \qquad (eq.6)$$

The correction term must enable the following criterion to be minimized at each step:

$$\min_{\hat{\theta}(t)} j(t+1) = [e^0(t+1)]^2 \qquad (eq.7)$$

Corresponding parametric adaptation algorithm will have the form:

$$\hat{\theta}(t+1) = \hat{\theta}(t) - F\frac{\partial J(t+1)}{\delta\hat{\theta}(t)}$$

Where:

- $F = \alpha I)$ $(\alpha > 0)$ is the matrix adaptation gain
- $\frac{\partial J(t+1)}{\delta\hat{\theta}(t)} = -\phi(t)e^0(t+1)$ is he gradient of the criterion

Parameter adaptation algorithm to be come:

$$\hat{\theta}(t+1) = \hat{\theta}(t) + F\phi(t)e^0(t+1)$$

2. **Recursive Least Squares Algorithm**

$$\hat{\theta}(t+1) = \hat{\theta}(t) + F(t)\phi(t)e(t+1)$$

$$\mathbf{F}(\mathbf{t}+\mathbf{1}) = \mathbf{F}(\mathbf{t}) - \frac{\mathbf{F}(\mathbf{t})\phi(\mathbf{t})\phi^{\mathbf{T}}(\mathbf{t})\mathbf{F}^{\mathbf{T}}(\mathbf{t})}{\mathbf{1} + \phi^{\mathbf{T}}(\mathbf{t})\mathbf{F}(\mathbf{t})\phi(\mathbf{t})}$$

$$\mathbf{e}(\mathbf{t}+\mathbf{1}) = \frac{\mathbf{y}(\mathbf{t}+\mathbf{1}) - \hat{\theta}^{\mathbf{T}}(\mathbf{t})\phi(\mathbf{t})}{\mathbf{1} + \phi^{\mathbf{T}}(\mathbf{t})\mathbf{F}(\mathbf{t})\phi(\mathbf{t})}$$

Where: $F(0) = \frac{1}{\delta}I = (GI)I$; (0¡δ¡¡1)
A typical value begin $\theta = 0.0001$ G=1000;

3. **Choice of the Adaptation Gain**

4. **Recursive Least Squares and Kalman Filter**

7.3.2 Parameter Adaptive Algorithm Stochastic Environment