PROJECT-1

ENPM 667: Controls of Robotics System

November 20, 2018

Project Report By: Nakul Patel(116334877)

Contents

1 Abstract

In the area of robotics, as one might have acknowledged, there are many sub-fields or verticals, and hence arise numerous applications. Many robotic industrial applications, such as milling, drilling, gluing, laser-cutting or such high precision measurements, do require the precise following of a predefined geometric(reference) path. So, the idea is to design an appropriate controller which achieves the desired goal and the type of controller plays an important role in this applications. Of all control strategies, model predictive controller helps achieve the trajectory following in the optimal way, by predicting the output states of the system,and using this inherent characteristics, implementation of model predictive path-following control for an industrial robot is done. For this purpose, consider constrained output path following with and without reference speed assignment.Also, during the design stages, stability is included as the important consideration and with the knowledge of optimal control and sequential quadratic programming being applied to the non-linear model predictive controller, and hence defining appropriate controller law, one can address the path-following problem for any system.

2 Introduction

The field of robotics is very vast and includes major verticals like modelling, designing, controls, computer vision, artificial intelligence, neural networks, etc. Of that, the controls of robotic systems plays a crucial role for development of any robot.

Of many control strategies available, namely adaptive control, model predictive control, optimal control, etc. I will elaborate more about model predictive control, since the journal paper is on the same control strategy.In model predictive control also, depending on the type of system the controller is dealing with and type of constraints, we can categorize them as either linear or non-linear.

The practical interest behind the non-linear model predictive controller is driven by the fact that today's real world industrial processes need to be operated under tighter performance specications. At the same time more and more constraints, for example from environmental and safety considerations, need to be satised. Often, one can achieve the same if the process nonlinearities and constraints are explicitly considered in the design stage of the controller.Hence,nonlinear predictive control, considering it as the extension of well established linear predictive control to the nonlinear world, appears to be a well suited approach for this kind of problems.

3 What is Model Predictive Control?

Industrial robots have now emerged as a primary means of contemporary automation due to their potential for increased productivity and product quality improvement. Evidently, a robot should be controlled so as to produce the output as possible per money invested. This in turn naturally leads to the need for minimum-time control of robots.

Model predictive control (MPC) is an advanced method in field of controls of robots and especially process control that is used to control a process while satisfying a set of constraints. They rely on the dynamic models of the process, most often linear empirical models obtained by system identification. The extension to this, can be the MPC which acts on the non-linear plant models. Model based predictive controller, as the name suggests, makes a prediction of future system behavior based on its model(linear or non-linear), the current system state, the input reference trajectory, or a disturbance entering the system. The following figure is the simple block diagram of MPC.

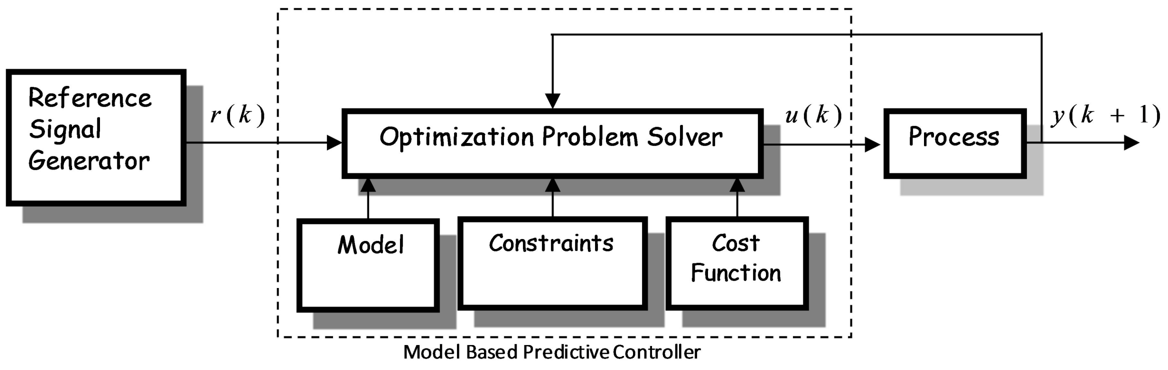
Reference Signal Generator
$r(k)$
Optimization Problem Solver
$u(k)$
Process
$y(k+1)$
Model
Constraints
Cost Function
Model Based Predictive Controller

Fig 3.1

MPC is based on an iterative, finite-horizon optimization of a plant model. As shown in Fig 3.2, at any time 'k' the current plant state is sampled and a cost minimizing control strategy is computed (by minimization algorithm/optimization problem) for a relatively short time horizon in the future :[k, k+p] . Specifically, an online calculation is used to compute the state trajectories that are derived from the current state and then,a cost-minimizing control strategy until time k+p, is calculated. Only the first step of the control strategy is implemented, then the plant state is sampled again and the calculations are repeated starting from the new current state, yielding a new control input and new predicted state path. Since the prediction horizon keeps being shifted forward, the MPC is also referred to as receding horizon control.

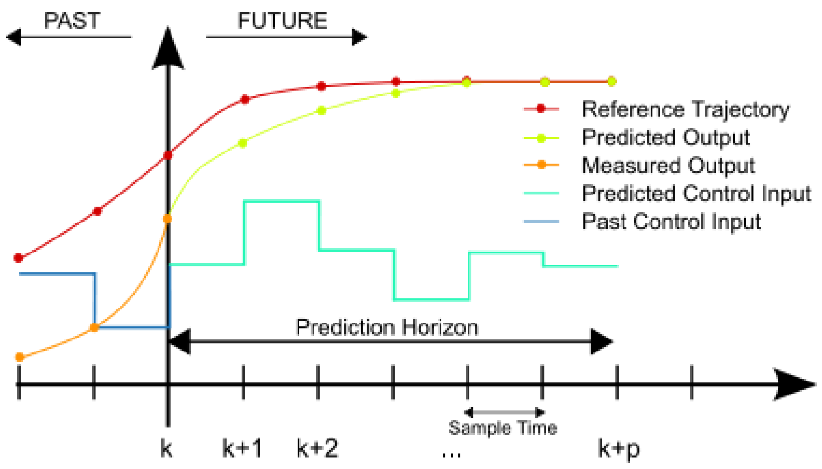

Fig 3.2

3.1 Advantages

1. Handles Multi-Input Multi-Output Systems
2. Multivariable controller
3. Handles linear and non-linear constraints
4. Preview capability and improved performance

3.2 MPC Design Parameters

1. **Sample Time:** It is the time over which the plant model state is sampled for calculating the future output state and the control input for achieving that state
2. **Prediction Horizon:** It is the number of future control intervals that the controller must evaluate by prediction, when doing the optimization of its manipulated variables at a particular time interval 'k'.At each time step, the optimization problem is solved for getting the output trajectory to follow the reference trajectory.In other words, it can be considered as <u>number of predictions</u>.
3. **Control Horizon:** It is the subset of the prediction horizon, which simply denotes the number of moves by the manipulated system variable to be optimized at a particular control interval 'k'.In simple words, it is the <u>number of control moves</u>
4. **Constraints:** The constraints may be linear or non-linear, and the constraints are imposed on output state and the control input to solve the optimization in the way that yields desired output. Furthermore, the constraints can be hard-constraints or soft-constraints.
5. **Weights in the cost function:** These are simply the scalar values by which the system variables gets multiplied, while formulating the cost function. Generally, they are denoted by the positive definite matrices Q and R, where Q is the weighted matrix for predicted outputs and hence errors, and R is the weighted matrix for control moves.

4 Extension to Nonlinear Model Predictive Control

Nonlinear Model Predictive Control, or NMPC, is an extension to model predictive control (MPC) that is characterized by the use of nonlinear system models in the prediction. Like linear MPC, NMPC requires the iterative solution of optimal control problems on a finite prediction horizon. While these problems are convex in linear MPC, they are no more convex in nonlinear MPC. This poses challenges for both NMPC stability theory and numerical solution.

One of the key questions in NMPC is certainly, whether a finite horizon NMPC strategy does lead to stability of the closed-loop. As pointed out, the key problem with a nite prediction and control horizon stems from the fact that the predicted open and the resulting closed-loop behavior is in general different. Ideally one would seek for a NMPC strategy that achieves closed-loop stability independent of the choice of the performance parameters in the cost functional and, if possible, approximates the innite horizon NMPC scheme as good as possible. A NMPC strategy that achieves closed-loop stability independent of the choice of the performance parameters is usually referred to a NMPC approach with guaranteed stability. Different possibilities to achieve closed-loop stability for NMPC using nite horizon length have been proposed.

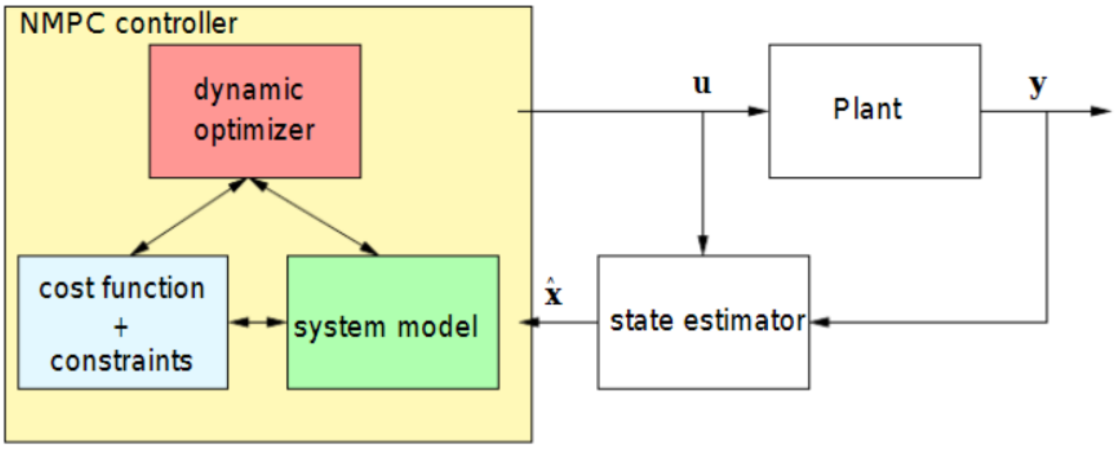

Fig 4.1

5 Keywords and their brief explaination

5.1 Setpoint:

It is the desired value for an essential variable of the system under consideration, or process value of a system.

5.2 Cost Function:

In mathematics,control theory, decision theory and machine learning, a cost function(sometimes known as loss function) is a function that maps an event or values of one or more variables onto a real number, which directly indicates the cost associated with the event. So, in optimization problems, we try to minimize the cost function. In other words, it penalizes the system's state or input.

5.3 Optimization problem:

The optimization problem in controls is the problem of finding the best solution from all the feasible solutions of a given problem. Further division to it is made on the basis of the variables, whether they are discrete or continuous. Hence discrete optimization problem and continuous optimization problem emerge.

The general continuous optimization problem can be considered as:

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & f(x) \\ \text{subject to} \quad & g_i(x) \leq 0, \quad i = 1, \ldots, m \\ & h_j(x) = 0, \quad j = 1, \ldots, n \end{aligned}$$

where

$\cdot f : R^n \rightarrow R$ is the objective function to be minimized over the n -variable vector x,
$\cdot g_i(x) \leq 0$ are inequality constraints
$\cdot h_j(x) = 0$ are called equality constraints, and
$\cdot m \geq 0$ and $n \geq 0$

From this definition, we can infer that if m and n are 0, then it becomes an unconstrained optimization problem.

5.4 Sequential Quadratic Programming:

An extension to the linear programming(wherein the objective function and all the constraints equation are linear functions of decision variables) is quadratic programming(QP).

Quadratic programming (QP) is the process of solving a special type of control optimization problem —specifically, a constrained quadratic optimization problem, that is, the problem of optimizing (minimizing or maximizing) a quadratic function of several system variables subject to linear or non-linear constraints on these variables. Also, it can be considered as a particular type of nonlinear programming.As the name suggests, it has an objective which is a quadratic function of the decision variables, and constraints which are all linear functions of the variables. For example,consider a quadratic function as:

$$ax_1 + b{x_2}^2 + cx_3$$

where x_1,x_2 and x_3 are decision variables, and a,b and c are coefficients. One important property of QP problems is that they have only one feasible region with "flat faces" on its surface (due to the linear constraints), but the optimal solution may be found anywhere within the region or on its surface. The quadratic objective function may be convex -which makes the problem easy to solve, or non-convex- which makes it very difficult to solve.

For the problems that have positive definite Hessians(in a minimization problem) or negative definite Hessians(in a maximization problem),that problems can be considered best for optimization. Also, one can imagine the graph of these functions as having a "round bowl" shape with a single bottom (or top), in other words, they have a form of convex function.

But, if a quadratic problem has an indefinite Hessian and a "saddle" shape function, more intuitively a non-convex function. Its true minimum or maximum can not be found in the "interior" of the function, but on its boundaries with the given constraints, where there are numerous locally optimal points in the co-ordinate space. Hence optimization of such non-convex function is a difficult global optimization problem.

Again extending the concept of QP, is the sequential quadratic programming, in which we utilize the iterative method for constrained non-linear optimization. They have an important property wherein the objective functions and the constraints are twice continuously differentiable. Two methods are widely used for solving the sequence of optimization sub-problems, each of which will lead to linearization of the constraints by optimizing the quadratic model of the objective function. These two methods are Newton-Raphson's method and Karush-Kuhn-Tucker Conditions.

5.5 Hessian:

One of the most important parameters in control theory and its application is the Hessian matrix. The Hessian matrix or Hessian is a square matrix of second-order partial derivatives of any scalar-valued function. Intuitively, it indicates the local curvature of a function of more than one variables.

Mathematically formulating a simple function and the process of obtaining the Hessian, let us consider the simple function $f : R^n \to R$, which takes the input as a n-dimensional vector $x \in R^n$, and gives a scalar output $f(x) \in R$. Now, considering that all the second partial derivatives of function f exists and are continuous over the function's domain, then we can write the Hessian as the square matrix of the form :

$$\mathbf{H} = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix}$$

or simply in terms of indices i,j we can write it as:

$$\mathbf{H}_{i,j} = \frac{\partial^2 f}{\partial x_i \partial x_j}$$

One important point to consider is that the off-diagonal entries in the matrix are the mixed derivatives of f. If assumed that they are continuous, the order of differentiation won't matter.

$$\frac{\partial}{\partial x_i}\left(\frac{\partial f}{\partial x_j}\right) = \frac{\partial}{\partial x_j}\left(\frac{\partial f}{\partial x_i}\right)$$

And, since it is a matrix, it will have a determinant as well, which is called Hessian.Hessian matrices are used often in the optimization problems.

5.6 Locally Lipschitz function:

A function is said to be locally Lipschitz on a particular domain(open and connected set) $D \subset R^n$, if each point of D has a neighbourhood D_0 such that $f(.)$ satisfies

$$\| f(x) - f(y) \| \leq L \| x - y \|, \forall x, y \in D_0$$

(with the same Lipschitz constant L). So, for $f : R \to R$ we have

$$\frac{|f(x) - f(y)|}{|x - y|} \leq L$$

For local existence and uniqueness: Let $f(t, x)$ be piecewise continuous in t and satisfy the Lipschitz condition

$$\| f(t, x) - f(t, y) \| \leq L \| x - y \|$$

$\forall x, y \in B = \{x \in R^n : -x_0 \| \leq r\}, \forall t \in [t_0, t_1]$. Then, there exists some $\delta > 0$ such that the state equation $\dot{x} = f(t, x)$ with $x(t_0) = x_0$ has a unique solution over $[t_0, t_0 + \delta]$.

For global existence and uniqueness: Suppose $f(t, x)$ is piecewise continuous in t and satisfies the Lipschitz condition

$$\| f(t, x) - f(t, y) \| \leq L \| x - y \|$$

$\forall x, y \in R^n, \forall t \in [t_0, t_1]$. Then, the state equation $\dot{x} = f(t, x)$ with $x(t_0) = x_0$ has a unique solution over $[t_0, t_0 + \delta]$.

5.7 Terminal Constraints:

Model Predictive Control is a powerful technique for the control of dynamical systems under state or input constraints that uses an online optimisation to maximise performance in the presence of those constraints. For the control of non-linear systems, recursive feasibility is often ensured through the application of constaints called terminal constraints whereby the final predicted state of the system is placed in a set that is known to be feasible and invariant for some stabilising feedback law. With these constraints defined, it is possible, though difficult to find robustly feasible and invariant sets for uncertain non-linear systems.

5.8 Heaviside step functions:

When working with differential equations, we often come across the signals that changes their sign with respect to time. Thus, we define a Heaviside step function as a discontinuous function, whose value is zero for negative argument, and it is one for positive argument. The simplest definition of this function can be formulated as belows:

$$H(x) := \frac{d}{dx} \max\{x, 0\}$$

Or, in the form of integral, it can be expressed in terms of dirac delta function as:

$$H(x) := \int_{-\infty}^{x} \delta(s) ds$$

5.9 Blockdiagonals:

In the field of mathematics, a block diagonal matrix or a partitioned matrix is simply a matrix that is discerned as having been broken into sections called blocks or submatrices. Intuitively, we can imagine or represent an original matrix as a block matrix which can be partitioned with a collection of horizontal and vertical lines, which break it up into a collection of smaller matrices. For example, consider

$$\mathbf{P} = \begin{bmatrix} 1 & 2 & 5 & 6 \\ 3 & 4 & 7 & 8 \end{bmatrix}$$

can be partitioned into two 2x2 matrices:

$$\mathbf{p_1} = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

$$\mathbf{p_2} = \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix}$$

6 Theory and explaination of the approach

6.1 Introduction to path tracking and trajectory tracking

There are a variety of algorithms available for robot motion control. These algorithms usually consider that the control structure of the robot has been divided into two levels. The lower level is called control or path tracking, and the upper level is called path or trajectory planning. The path tracker drives the robot's actual position and velocity to match desired values of position and velocity; the desired values are provided to the controller by the trajectory planner. The trajectory planner will receive as input some sort of geometric path reference from which it calculates a time history of the desired positions and velocities. The path tracker then tries to minimize the deviation of the actual position and velocity from the desired values.

Of all the control tasks, there are certain applications,where the system should be driven along the pre-specified geometric path in the output space. For example, consider machining applications like drilling, milling or steering an autonomous vehicle along the reference track. Here, one doesn't definitely know the speed to move along the curve. That is where path-following problems are employed.

6.2 Dynamics and Optimal Control Problem(OCP)

For many industrial and commercial robots, these problems are quite often arising from the dynamic motion planning and trajectory generation. As discussed, the division of robot control into path tracker and trajectory planner is required since the dynamics of most robots or processes are non-linear and coupled. However, this leads to inefficiency, since the trajectory planner must be aware of robot's dynamics, which inturn is not the case always owing to the appropriate development in trajectory planning algorithms.

As a consequence, to alleviate this concern, the control strategy of Optimal Control Problem subject to constraints on the geometric path is employed. In addition, since model predictive control itself solves the optimization problem at each time step, this strategy of OCP will yield the minimum-time solution. Hence, the path-following problems are decomposed into trajectory generation and trajectory tracking.

Optimal Control Problem:

Model Predictive Controllers are often solved by the approach of Optimal Control Problem, which basically deals with the problem of deriving a control law for a system in such a way that certain optimality criterion is achieved. We then include a cost functional that is a function of state and control variables.

Most of the optimal control problems are non-linear and therefore, generally do not have analytical solutions.

Moreover, due to the inherent closed-loop characteristics of the MPC and their application to nonlinear dynamic systems, the efforts are made to systematically derive stability and convergence conditions. However, usually the computational demand of the solution of the underlying OCP is rather high. Very often the idea is to refrain from computing the optimal solution at each sampling instant. Rather a suboptimal but feasible solution is searched for, which is improved from one sampling instant to the next.

Ultimately, a closed-loop path-following model predictive controller directly modulates the reference speed along the geometric path to reduce the path-following error and, at the same time, computes inputs to track this reference motion. This is how the OCP is utilized alongside model predictive control for the purpose of desired control of our robot.

In this paper, the design and implementation of a sampled-data nonlinear model predictive path-following control scheme in the presence of input and state constraints is discussed.This has been corroborated by the demonstration of predictive path following.

7 Procedures

7.1 Predictive Path Following

We start the explaination of the predictive path following for non-linear systems by defining some state space equations, path/reference equation, and certain other system-theoretic properties like stability and path-convergence. Also, it has been considered that the reference paths are defined directly in the operational space, i.e the Cartesian space.

Now,let a particular non-linear system be represented as:

$$\dot{x} = f(x) + \sum_{j=1}^{n_u} g_j(x)u_j, x(t_0) = x_0$$

$$y = h(x)$$

The map $h : R^{n_x} \rightarrow R^{n_y}$ defines the output $y \epsilon R^{n_y}$ or the variables of specific interest. We assume that the maps $f : R^{n_x}ßR^{n_x}$, $g_j : R^{n_x} \rightarrow R^{n_x}$, and $h : R^{n_x} \rightarrow R^{n_y}$ are sufficiently often continuously differentiable. Here, $x \in X \subset R^{n_x}$ and $u \in U \subset R^{n_u}$ denote the closed set of state constraints and the compact set of input constraints.

In the simplified form of representation, we can write:

$$\dot{x} = f(x, u), x(t_0) = x_0 \tag{1}$$

$$y = h(x) \tag{2}$$

Here, $x \in R^{n_x}$, $u \in R^{n_u}$ and $y \in R^{n_u}$ are the state, input and the output respectively. Also, the states are being constrained to the closed set, for all $t : x(t) \in X \subseteq n_x$. Here, assume that the maps $f : R^{n_x} \rightarrow R^{n_x}$ and $h : R^{n_x} \rightarrow R^{n_y}$ are often continuously differentiable. And the final assumption made is that the control system has a square input-output structure, thus dim u= dim y= n$_u$.

Consider these notations, which will be used further: The inputs $u : [t_0, \infty] \rightarrow U$ are piecewise continuous and are denoted by $u(t) \in PC(U)$. Also, the solution of equation(1) at time t, which originates at t_0 from x_0, driven by the input u(.), is denoted by $x(t, t_0, x_0|u(t))$.

Having described the state-space equation of the system, we now define the parameters for path-following. Now, the prototypical problem in controls is the stabilization of a set-point along with design of controllers for the tracking of time-varying references. Typically, the highest priority is given to the minimization of the deviation or error between the geometric reference path and the robot's end-effector. The velocity/speed to move along the reference path is of secondary interest and might be tuned in order to achieve better accuracy. Such control problems requiring a system to follow a geometric reference curve, whereby the speed along this reference is a degree of freedom in the controller design are termed path following problems.

7.2 Path-following Problems

Since the direct consideration of the input and the state constraints is difficult for various approaches of path-following, Non-linear Model Predictive Control)(NMPC) is used by defining the path-following problems in the output space. Hence, it is also referred to as the output path-following.

Path-following requires convergence of the states or the outputs of a system to a reference path, where we aim to drive the system to the geometric reference without any pre-specified timing information. Let us define the this reference path to be followed(P) as:

$$P = \{y \in R^{n_u} | \theta \in R \mapsto y = p(\theta)\} \tag{3}$$

where, the scalar variable $\in R$ is called path parameter, and $p(\theta)$ is a parameterization of P.

Parametrization: With this process, one can find a parametrized curve that traces out a given path.

Moreover due to the lenient requirement of when to be where on P, the path parameter is time dependent but its time evolution $t \mapsto (t)$ is not specified beforehand. Instead, we choose the system input u(t) and the timing $\theta(t)$ such that the path is followed as exactly as possible. Furthermore, for finitely long paths, it can be helpful to restrict the path parameter to $[\theta_0, \theta_1]$, where θ_0 is the start point of the path and $\theta_1 \in [\theta_0, \infty)$ denotes the endpoint of the path. Subsequently, path is to be followed in the direction of increasing values of θ.

Now, of many ways, one could solve this problem by choosing a fixed timing $\theta(t)$ and designing a trajectory-tracking controller for $p(\theta(t))$. This way, path following would be reformulated as a trajectory-tracking problem. However, the degree of freedom of adjusting $\theta(t)$ is lost. In order to avoid this problem,we first obtain the system input $u : [t_0 \infty) \to U$ and the reference timing $\theta(t)$ in the controller, and after that the controller determines the input u(t) to converge to reference path as well as the time evolution $\theta(t)$ of the reference. In other words, we consider the following problems:

Problem 1: *Constrained Output-path Following:*
From the given state space equations of the system and the above defined reference path P, we design the controller that calculates u(t) and $\theta(t)$ and satisfies following conditions:
1) Path Convergence: As the name implies, the output signal of the system(robot's end-effector) should follow the path P. So we consider that the output y=h(x) converges as per following definition:

$$lim_{t \to \infty} \parallel h(x) - p(\theta(t)) \parallel = 0$$

2) Convergence on path: Here, the system moves along the path P in forward direction, so we can say that

$$\dot{\theta(t)} \geq 0$$

and

$$lim_{t \to \infty} \parallel \theta(t) - \theta_1 \parallel = 0$$

3)Constraints satisfaction: For all $t \in [t_0, \infty]$, all the constraints on the states x(t) and inputs u(t) are satisfied.
Also, sometimes, there is a requirement to track the speed profile along the path, which gives rise to formulation of other problem as follows:

Problem 2: *Output Path Following with Velocity Assignment:*
We define the condition of velocity convergence here which is supposed to be satisfied, after the above three conditions have been satisified for the same system.

$$lim_{t \to \infty} \parallel \dot{\theta(t)} - \dot{\theta}_{ref}(t) \parallel = 0$$

Non-minimum phase systems: Most of the systems in the real world, are not stable inherently, and hence require an appropriate controller to drive their behaviour to stability. If we consider these systems in the frequency domain, and then obtain their transfer function, then we get atleast one pole/zeroes which lie in the right half plane of the s-plane. Such systems are called non-minimum phase systems.

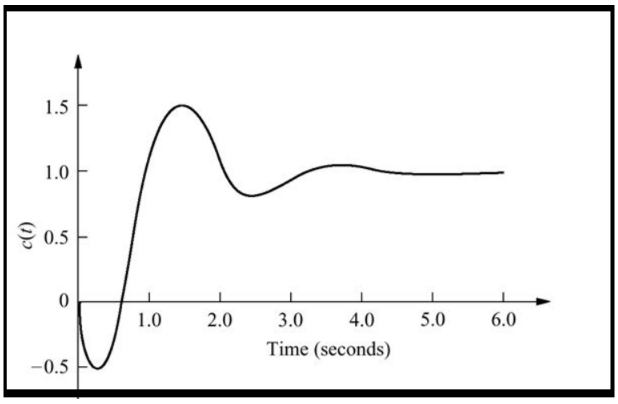

Fig 7.1

Moreover, non-minimum phase systems does not move towards the set point directly. So first the output response will move in opposite direction to gain required energy to reach the set-point. Hence, we require to minimize the tracking error that will be generated from this response, since the reference trajectory will have completely different response.

It is a known fact that in the reference-tracking, for non-minimum phase systems, there exists a fundamental performance limitation in terms of a lower bound on the euclidean-norm of the tracking error, even when the control effort is free. But for lenient path-following problems, where the control objective is to force the output to follow a geometric path without a timing law assigned to it, there will be no limitation described above. Furthermore, the same is true even when an additional desired speed assignment is imposed.

Because many reference trajectories are generated $p(\theta_i(t)), i \in \{1, 2, 3, ...\}$ when we consider path following with the speed assignment, we can say that the approach does not specify a unique output reference $p(\theta(t))$. Also, the speed profile for these trajectories may differ with respect to θ.

Solution to this problem: To overcome the above problems, we apply the concepts of classical design of path-following controllers, wherein we regard the path parameter as the virtual state., which is a very short-lived, unobservable state. The evolution of this state is determined by the additional ordinary differential equation, which is referred to as the timing law. Timing law is basically the adaptive control law as a function of time for time varying control.

Keeping the dynamics of the system in mind, an adaptive control law can be derived, for an n-link manipulator based on Lyapunov Theory for stability. Also, the parameter timing law(control law) is updated by using an exponential function of manipulator's kinematics, inertia parameters and tracking errors.

For this purpose, let us consider a simple dynamics equation of a robot manipulator, neglecting the frictional forces and other disturbances, so that we can write:

$$M(q)\ddot{q}+C(q,\dot{q})\dot{q}+G(q)=T \tag{4}$$

where q is considered to be the vector of joint variables, T is the vector of applied torques,$M(q)$ is the inertia matrix and $C(q, dotq)$ is the Centrifugal and Coriolis components of forces, $G(q)$ is the vector of gravitational forces. So we can rewrite equation as:

$$M(q)\ddot{q}+C(q,\dot{q})\dot{q}+G(q)=Y(q,\dot{q},\ddot{q})\theta \tag{5}$$

where θ is the vector of constant parameters of robot, and we have denoted Y to be the function of joint position, velocity and acceleration.
Now we define a control law(substituting T) with a positive definite matrix K as:

$$T=M(q)\ddot{q}_r+C(q,\dot{q})\dot{q}_r+G(q)+K\sigma \tag{6}$$

We can write other quantities as :

$$\tilde{q}=q_d-q; \quad \dot{\tilde{q}}=\dot{q}_d-\dot{q}; \quad \dot{q}_r=\dot{q}_d+\Lambda\tilde{q}; \quad \ddot{q}_r=\ddot{q}_d+\Lambda\dot{\tilde{q}}$$

in which $\tilde{q}$ is the tracking error between desired and actual position, λ represents the non-linear compensation and decoupling terms in terms of desired velocity and acceleration. Also, $K\lambda$ can be interpreted as equivalent to Proportional Derivative action on the tracking error, so taking σ as:

$$\sigma=\dot{q}_r-\dot{q}=\dot{\tilde{q}}+\Lambda\tilde{q}$$

For this, we have considered system's dynamics, but what if one doesn't know the system parameters, then we write control law(6) as follows:

$$T=\hat{M}(q)\ddot{q}_r+\widehat{C}(q,\dot{q})\dot{q}_r+\widehat{G}+K\sigma=Y\left(q,\dot{q},\dot{q}_r,\ddot{q}_r\right)\hat{\theta}+K\sigma \tag{7}$$

where $\hat{\theta}$ represents the available estimate on the parameters and $\hat{M},\hat{C},\hat{G}$ represent the estimated terms in the dynamic model. So by substitution of (7) in (5), we get

$$M(q)\dot{\sigma}+C(q,\dot{q})\sigma+k\sigma=-\tilde{M}(q)\ddot{q}_r-\tilde{C}(q,\dot{q})\dot{q}_r-\tilde{G}=-Y\left(q,\dot{q},\dot{q}_r,\ddot{q}_r\right)\tilde{\theta} \tag{8}$$

in that $\tilde{\theta}=\hat{\theta}-\theta$ is the parameter error vector.And the modelling error in dynamics is:

$$\tilde{M}=\widehat{M}-M, \quad \tilde{C}=\widehat{C}-C, \quad \tilde{G}=\widehat{G}-G$$

Now, for stability, we define Lyapunov function as follows:

$$V(\sigma,\tilde{q},\tilde{\theta})=\frac{1}{2}\sigma^{\mathrm{T}}M(q)\sigma+\frac{1}{2}\tilde{q}^{\mathrm{T}}B\tilde{q}+\frac{1}{2}\tilde{\theta}^{\mathrm{T}}K_\theta\tilde{\theta}>0 \quad \forall\sigma,q,\tilde{\theta}\neq 0 \tag{9}$$

where B and K_θ are positive definite diagonal matrices. Now we can write time derivative of V (using property $\sigma^{\mathrm{T}}[\dot{M}(q)-2C(q,\dot{q})]\sigma=0\forall\sigma\in R^n$)as:

$$\dot{V}=-\dot{\tilde{q}}^{\mathrm{T}}K\dot{\tilde{q}}-\tilde{q}^{\mathrm{T}}\Lambda K\Lambda\tilde{q}+\tilde{\theta}^{\mathrm{T}}\left(K_\theta\dot{\tilde{\theta}}-Y^{\mathrm{T}}\left(q,\dot{q},\dot{q}_r,\ddot{q}_r\right)\sigma\right)$$

After this, the adaptation law is chosen such that

$$\dot{\hat{\theta}} = K_\theta^{-1} Y^{\mathrm{T}}\left(q, \dot{q}, \dot{q}_r, \ddot{q}_r\right)\sigma$$

Now, since θ is constant,

$$\dot{V} = -\dot{\tilde{q}}^{\mathrm{T}} K \dot{\tilde{q}} - \tilde{q}^{\mathrm{T}} \Lambda K \Lambda \tilde{q} \leq 0$$

So finally, we can write the required timing law as follows:

$$\hat{\theta} = \int K_\theta^{-1} Y^{\mathrm{T}}\left(q, \dot{q}, \dot{q}_r, \ddot{q}_r\right)\sigma \mathrm{d}t + \pi \tag{10}$$

Also, in simple words, one can consider the timing law, as the additional degree of freedom in the controller design.Now, for the derivation of the cost functions and optimization problems to be simpler, we consider the timing law as an integrator chain, wherein the timing of the path parameter θ is specified with the following simple ordinary differential equation:

$$\theta^{(r)} = v, \quad \theta^{(i)}(t_0) = {\theta_0}^{(i)}, \quad i = 0, 1, ..., r-1$$

In the above equation, depending on the value of power r, we can consider the variable v on right hand side as either speed, acceleration or jerk of the the reference path. We defined this timing law so that the virtual input v can be used to control the evolution of the reference $p(\theta(t))$ and the time evolution $\theta(t)$ which generates the reference. Considering v from the initial time to infinite time,we write its domain as $v : [t_0, \infty] \to V$. Also, the value of $r \in N$, will depend on the design method and the system itself, and system dynamics.

Based on the timing law described above, let us define the equations governing the path-following problem.

$$\dot{x} = f(x) + \sum_{j=1}^{n_u} g_j(x) u_j, x(t_0) = x_0 \tag{11}$$

$$\dot{z} = \hat{I}^r z + E^r v \tag{12}$$

$$e = h(x) - p(z_1) \tag{13}$$

$$\theta = z_1 \tag{14}$$

Here the equation(11) includes the dynamics of the system to be controlled, which is already represented in the state space form earlier. Equation(12) is the timing law, wherein $z = (\theta, \theta^1, ...\theta^{r-1})$. Also, when talking about the path-following we definitely consider the tracking error. So equation(13) gives the error as the deviation from from the reference path.And finally, for given time-instant, equation(14) gives the current reference position on the path.

Having done all the basic representation of the system and defined other required system terminologies, we now proceed to formulate a proper control scheme for the problem defined till now. That path-following problem will be tackled with the help of a predictive path-following control scheme.

7.3 Model Predictive Path Following Control Scheme:

As explained earlier, model predictive control is based on the optimization problem wherein along each time instant in the prediction horizon, it solves an online optimization problem to generate an optimized control law that will drive the system to the predicted future value.This strategy indeed will lead to better control of complicated non-linear systems as well.

The above system can be represented in the augmented form as follows:

$$\begin{pmatrix} \dot{x} \\ \dot{z} \end{pmatrix} = \begin{pmatrix} f(x,u) \\ l(z,v) \end{pmatrix},$$

given the initial condition,

$$\begin{pmatrix} x(t_0) \\ z(t_0) \end{pmatrix} = \begin{pmatrix} x_0 \\ z_0 \end{pmatrix}$$

Consequently, an optimization problem in any aspect of controls can be solved by first defining a cost function. And then minimize the cost function with the optimal control problem(OCP).

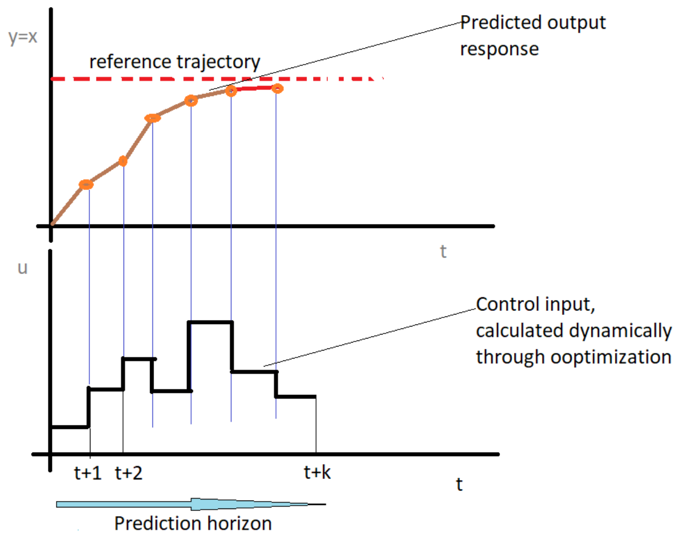
y=x
reference trajectory
Predicted output response
t
u
Control input, calculated dynamically through ooptimization
t+1
t+2
t+k
t
Prediction horizon

Fig 7.2

We solve the following OCP with the cost function as described below:

$$J(x(t_k), \overline{z}(t_k), \overline{u_k}(.), \overline{v_k}(.)) =$$

$$\int_{t_k}^{t_k+T} F(\overline{e}(\tau), \overline{z}(\tau), \overline{u}(\tau), \overline{v}(\tau))dx$$

However, if we are concerned about the conditions that ensures the solution to above problems(Problem 1 and Problem 2), and guarantees path convergence, then we might want to establish those required conditions by adding an end-penalty and terminal constraint in the OCP.

Therefore, after introducing penalty function, we can rewrite the cost function of proposed OCP as:

$$J(x(t_k), \overline{z}(t_k), \overline{u_k}(.), \overline{v_k}(.)) =$$
$$\int_{t_k}^{t_k+T} F(\overline{e}(\tau), \overline{z}(\tau), \overline{u}(\tau), \overline{v}(\tau))dx +$$
$$E(t, \overline{x}(t), z(t))|_{t_k=t_k+T} \tag{15}$$

As usual in non-linear MPC, the function $F : R^{n_y} \times R \times V \times U \rightarrow {R_0}^+$ is termed the cost function, and $E : {R_0}^+ \times R^{n_x} \times R^r \rightarrow {R_0}^+$ is denoted as terminal or end penalty;furthermore the predicted system states and inputs are indicated by the superscript $\overline{(.)}$. The subscript $(.)_k$ highlights that an open-loop input $\overline{u_k}(.)$ is computed at the k_{th} sampling instant t_k. The constant $T \in (\delta, \infty)$ is called the prediction horizon. Hence, the OCP to be solved in a receding horizon fashion at the sampling times t_k reads as follows:

$$\underset{(\overline{u_k}(.), \overline{v_k}(.)) \in PC(U \times V)}{\text{minimize}} J(x(t_k), \overline{z}(t_k), \overline{u_k}(.), \overline{v_k}(.))$$

In the above cost function, all the parameters of the cost function are in themselves the function of time τ which is defined for

$$\forall \tau \in [t_k, t_k + T]$$

Moreover, following constraints hold for the above defined minimization:

$$\frac{d}{d\tau}\overline{x}(\tau) = f(\overline{x}(\tau)) + \sum_{j=1}^{n_u} g_j(\overline{x}(\tau))\overline{u}_{k,j}(\tau), \overline{x}(t_k) = x(t_k) \tag{16}$$

$$\frac{d\overline{z}(\tau)}{d\tau} = \hat{I}^r z(\tau) + E^r v(\tau) \tag{17}$$

$$\overline{z}(t_k) = \overline{z}(t_k, t_{k-1}, \overline{z}(t_{k-1})|\overline{v}^*_{k-1}(.)) \tag{18}$$

$$\overline{e}(\tau) = h(\overline{x}(\tau)) - p(\overline{z_1}(\tau)) \tag{19}$$

$$\overline{\theta}(\tau) = \overline{z_1}(\tau) \tag{20}$$

$$\overline{x}(\tau) \in X, \overline{u_k}(\tau) \in U \tag{21}$$

$$\overline{v}(\tau) \in V, \overline{z_k}(\tau) \in Z \tag{22}$$

$$(\overline{x}(t_k + T), \overline{z}(t_k + T))^T \in \epsilon \subset X \times Z \tag{23}$$

In the above cost function, the decision variables are the real system input $u(.) \in PC(U)$ as well as the virtual path parameter input $v(.) \in PC(V)$. In other words, by solving the system of equations((16) to (23)) we obtain the system input and the reference evolution at the same time.

We know by now that, at each sampling instant t_k, the current measured state value given by $x(t_k)$ and it can be used as an initial condition for equation(16). Also, the initial condition(18) of the timing law(17) will be dependent on the last predicted trajectory. However, for the first sampling instant, we have no initial conditions of the system , so we have k=0. So we can obtain the timing law by following optimization problem:

$$\overline{z}(t_0) = ((t_0), 0, ..., 0)^T \tag{24}$$

$$\theta(t_0) = \underset{\theta \in [\theta_0, \theta_1]}{\arg\min} \parallel h(x_0) - p(\theta) \parallel \tag{25}$$

There might be multiple optimal solutions to above problem, and hence we simply choose one of them. Furthermore, the state and input constraints of the system to be controlled, denoted by $\overline{x}(\tau)$ and $\overline{u_k}(\tau)$ respectively are enforced by (21). The path parameter dynamics(17) are subject to the state and input constraints (14), whereby the state constraint Z is defined as: $Z := [\theta_0, \theta_1] \times R_0^+ \times R^{r-2} \subset R^r$

Basically, this constraint ensures that $\overline{\theta}(\tau) = \overline{z_1}(\tau) \in [\theta_0, \theta_1]$ as well as its derivative $\dot{\overline{\theta}} \geq 0$. In this way, we can enforce monotonous forward motion along the path.

In general, given initial state $x(t_0)$, $\overline{z}(t_0)$, sampling rate δ and sampling time of prediction horizon T, we can write the OCP problem for MPC in following steps:

Step 0: Initialize the constant k=0.
Step 1: Obtain the system's state information $x(t_k)$.
Step 2: Solve the OCP (16) to (23) with initial condition $x(t_k)$,$\overline{z}(t_k)$, which will yield optimized control law.
Step 3: Now apply the optimal input from the control law,

$$\forall t \in [t_k, t_k + \delta) : u(t) = \overline{u}_k^*(t).$$

Step 4: We will now assign $\overline{z}(t_{k+1}) = \overline{z}(t_{k+1}, t_k, \overline{z}(t_k)|\overline{v}_k^*(.))$.
Step 5: $k \to k + 1$
Step 6: Goto Step 1.

In the above process, the terminal constraint (23) enforces that at the end of each optimization sequence, the predicted augmented state $(\overline{x}(t_k+T), \overline{z}(t_k+T))^T$ lies inside a terminal region $\epsilon \subseteq X \times Z$.From the definition of the cost function, it is clear that we have defined only outputs and inputs in the cost function F, which have been penalized. Also, the terminal constraint is represented in the state space. We have done so because under suitable assumptions , our problem of output path following can be reformulated as a manifold stabilization problem in the state space.

Also note that the terminal penalty E will be used to obtain an upper bound on the cost associated to solutions originating inside the terminal region $\epsilon \subseteq X \times Z$. Thus, E is stated as a function of the augmented state $(x, z)^T$.

Hence, the optimal solution of overall OCP problem can be denoted as optimized cost function $J^*(x(t_k), \overline{z}(t_k), \overline{u}_k^*(.), \overline{v}_k^*(.))$. And lastly, it is specified by the optimal input trajectories $\overline{u}_k^* : [t_k, t_k + T] \to U$ and $\overline{v}_k^* : [t_k, t_k + T] \to V$.

Thus, solving these OCP equations at each time t_k with finite horizon $[t_k, t_k + T]$, one can plan a reference motion $t \mapsto p(\theta(t)) \in P$ and, at the same time, computes the system inputs to track these trajectories. However, since the MPFC scheme described above doesn't aim at time-optimal motion along the path P, we particularly aim at the feedback strategy ensuring that the system output converges to the path and moves along the path in forward direction.

Following remarks can be summarized from the discussion of the control strategy described above:

Remark 1: We can infer the dynamic nature of the model predictive control from the fact that the solution of optimization obtained at each time instant t_k depends on the solution of previous sampling instant t_{k-1}.

Remark 2: Computational Demand: The OCP we wrote is based on the terminal constraints and the terminal penalties, and we have increased dimensions of state and input variables. Since we have focused on path-following, the numerical implementation of the same is computative and complex as compared to the model predictive control scheme for set-point stabilization.

7.4 Sufficient Convergence Conditions:

After the formulation of the system equations, its control with optimal control problem, and obtained predicted states, it now becomes necessary to validate the conditions that ensure that the proposed MPFC scheme easily solves the Problem 1(Output path-following with constraints) and Problem 2(Output path-following with velocity assignment).We know by now that the receding horizon application of optimal open-loop inputs does not necessarily lead to stability nor to convergence of the closed-loop output to the reference path. Let us define these conditions, which will assure path convergence and feasibility in the presence of state constraints.

Assumption 1 (System Dynamics): The vector fields $f, g_j, j = 1, ..., n_u$, and h from simplifies state-space equations(1) are continuous and locally Lipschitz for any pair $(x, u)^T \in X \times U$.

Assumption 2 (Continuity of System Trajectories): For any given initial state of the system $x_0 \in X$ and any input function $u(.) \in PC(U)$, the system represented by equation(1) and (2) has an absolutely continuous solution.

Assumption 3 (Consistency of Path and State Constraints): The path P defined earlier is contained in the interior of the point-wise image of the state constraints X under the output map $h : R^{n_x} \rightarrow R^{n_y}$ from (2), i.e.,$P \subset int(h(X))$.

Assumption 4 (Cost Function): The cost function $F : R^{n_y} \times R \times V \times U \rightarrow R_0^+$ is a continuous function. Furthermore, we assume that the cost function F is lower-bounded by a class K function, i.e., $\psi(\| (e, \theta - \theta_1)^T \|) \leq F(e, \theta, u, v)^3$.

Convergence of MPFC: Under the above assumptions, we can assure path convergence and since we are working around cost functional and its optimization,we derive the proof of the following results, that exists given the terminal region of states and virtual path-parameter input $\epsilon \subset X \times Z$ and terminal penalty E(t,x,z).
i) The set ϵ is compact. E(t,x,z) is positive definite with respect to (t,x,z).

ii) For all $t \in [t_0, \infty)$ and all $(\tilde{x}, \tilde{z})^T \in \epsilon$, there exists a scalar $\epsilon \geq \delta > 0$ and admissible inputs $(u_\epsilon(.), v_\epsilon(.)) \in PC(U \times V)$ such that for all $\tau \in [t, t+\delta]$,

$$\frac{d}{d\tau}E(\tau, x(\tau), z(\tau))) + F(e(\tau), \theta(\tau), u_\epsilon(\tau), v_\epsilon(\tau)) \leq 0 \tag{26}$$

and the solution $x(\tau)$ and $z(\tau)$ remain in the terminal region ϵ for all $\tau \in [t, t+\delta]$

iii) The solution of OCP is feasible for initial time t_0.

Now, let us see the proof.

Step1: Recursive feasibility: Here, we show recursive feasibility recursive feasibility via the usual concatenation of optimal inputs $(u_k^*(.), v_k^*(.))$ with the terminal controls $(u_\epsilon(.), v_\epsilon(.))$.Since these concatenated inputs ensure positive invariance of the terminal constraint E, it immediately follows that the proposed MPFC scheme is recursively feasible.

Step2: Constraint satisfaction and forward motion: We know that, for all $(x, z)^T \in \epsilon$, we have $z \in Z$, which implies that the forward motion requirement $\dot{(\theta)} = z_2 \geq 0$ holds. Hence, the forward motion aspect in part (2) of Problem 1(formulated above) is ensured. Furthermore, the terminal constraint set is contained in the state constraints, i.e.,$\epsilon \subset X \times Z$. Thus part (3) of Problem 1 is also satisfied.

Step3: Path convergence and convergence on path: After we have shown that the part (2) and (3) of the Problem 1 are satisfied, now we are still to prove that part (1) is also satisfied. For this, we consider the value function of the OCP as follows:

$$V(t_k, x(t_k), \overline{z}(t_k)) := J(x(t_k), \overline{z}(t_k), u_k^*(.), v_k^*(.))$$

Now, we can use the invariance condition (18) to show that for all sampling instants $\delta \in (0, \epsilon]$, we have

$$V(t_{k+1}, x(t_{k+1}), \overline{z}(t_{k+1})) - V(t_k, x(t_k), \overline{z}(t_k)) \leq$$

$$-\int_{t_k}^{t_{k+1}} \psi(\| \ (e(t), \theta(t) - \theta_1(t))^T \ \|)dt$$

Now, let us consider the MPC value function:

$$V^\delta(t, x(t), \overline{z}(t)) := V(t_k, x(t_k), z(t_k)) - \int_{t_k}^{t} F(e(\tau), \theta(\tau), u_k^*(\tau), v_k^*(\tau))d\tau$$

which is the remainder of $V(t_k, x(t_k), \overline{z}(t_k))$ for $x(t) = x(t, t_k, x(t_k)|u_k^*(.))$ and $z(t) = z(t, t_k, z(t_k)|v_k^*(.))$. Here, we have considered the time instant to be $t_k = k\delta$ with

$k = max_{k \in N}\{k|t_k \leq t\}$, i.e. the closest previous sampling instant. So, we can show that for all $t \geq t_0$,

$$V^{\delta}(t, x(t), \overline{z}(t)) + \int_{t_0}^{t} \psi(\| \ (e(t), \theta(t) - \theta_1(t))^T \ \|) d\tau \leq V^{\delta}(t_0, x(t_0), \overline{z}(t_0))$$

Note that the proposed control scheme aims on establishing the convergence of the output y = h(x) to the path.

7.5 Problems with and without speed assignment:

As discussed till now, we have seen the formulation of general model predictive control scheme, wherein we have been considering the problem of just path-following. This is to say that we had not considered other parameters like speed and acceleration during the process of the path-following by the robot. Now let us consider the speed assignment to the problem formulation.

As seen, we considered θ as the path parameter and defined some conditions for its stability when we formulated speed-assigned path following. In that we considered $\dot{\theta}_{ref}$ as the predefined reference speed.

So, in new optimal control problem(OCP) formulation of the speed-assignment, we consider new quadratic cost function as follows:

$$F(e, z, u, v) = \left\| \left(e, z_1 - \theta_1, z_2 - \dot{\theta}_{\text{ref}} \right) \right\|_Q^2 + \left| (u, v) \right\|_R^2 \tag{27}$$

which will simplify and make the computation of the Hessian of OCP easier. Also, Q is a positive semi-definite matrix and R is a positive definite matrix. The common property they possess is that they are diagonal matrices. Another interpretation for Q is that if you choose large value of Q, that means you are trying to stabilize the system with least possible changes in the states, and vice-versa. Similarly, choosing a larger value of R will stabilize the system with less weighted energy. Hence, together these parameters can be considered as the design parameters that penalize the state variables and the control signals.
So, we consider Q of the form $Q = diag(w_e, w_\theta, w_{\dot{\theta}})$ and $R = diag(r_u, r_v)$, where e, θ,$\dot{\theta}$, u and v are the varibles which we have considered in the cost function. Hence, if we are considering only the path-following, then we will require $z_1 = \theta_1$, leading to penalizing $z_1 - \theta_1$, and finally we choose $w_\theta > 0$ and $w_{\dot{\theta}} = 0$. And now, considering the other case of path-following with speed assignment, we want to achieve $z_2 = \dot{\theta}_{ref}$, hence penalizing $z_2 - \dot{\theta}_{ref}$, and equating it to zero. And we select the diagonal entries as $w_{\dot{\theta}} > 0$ and $w_\theta = 0$.

7.6 System's Dynamics and MPFC Design:

Here, the experiment for the validity of the proposed model has been done with the help of KUKA LWR IV robot, which is a light-weight robot, basically and depending on the application, few joints have been actuated, while other are kept fixed. So now consider the dynamic model of the robot, which can be described as follows:

$$B(q)\ddot{q} + C(q, \dot{q})\dot{q} + \tau_F(\dot{q}) + g(q) = \tau \tag{28}$$

Here, we have considered the joint variables as q. For revolute joints, it is the joint angle, and for the prismatic joint it is the link length. Keeping the simplest configuration of robot in mind, only the joints 1,2 and 4 have been considered for calculating the equation of motion. Hence, we can write the joint variables vector as $q = (q_1, q_2, q_4)^T$. Also, the time derivative of q are considered as $\dot{q}$(joint velocities) and $\ddot{q}$(joint accelerations). The vector $\tau = (\tau_1, \tau_2, \tau_4)^T$ indicates the actuation torques applied to the corresponding joints.

Also, in the above equation the term $B(q)$ is the inertia matrix; and the term $C(q, \dot{q})$ denotes the centrifugal and coriolis components of the velocities. Moreover, the term $\tau_F(\dot{q})$ indicates the dynamics due to joint friction. Also, the gravity will definitely act on any system, hence here also we have included it as the term $g(q)$. Here, since the contact forces with the interacting environment have not been considered, we can say that the model is describing the free moving robot. However, we can consider the contact forces as disturbances, and correspondingly include the disturbance terms and variables as well in the controller design.

Now, writing the above dynamics in the state-space form, we can consider the states x_1 and x_2, inputs u as follow:

$$x_1 = q$$
$$x_2 = \dot{q}$$
$$u = \tau$$

Thus, we can write the system in the augmented state-space representational form as :

$$E\begin{pmatrix}\dot{x_1}\\ \dot{x_2}\end{pmatrix} = \begin{pmatrix} x_2 \\ u - C(x_1, x_2)x_2 - \tau_F(x_2) + g(x_1)\end{pmatrix} \tag{29}$$

$$y = h_{cart}(x_1) \tag{30}$$

where, E is the blockdiagonal.Since the first state's derivative is the second state itself, we multiply it by identity matrix I. Also, in the equation of motion, the second state($\dot{q}$)'s derivate $\ddot{q}$ is multiplied with the inertia matrix B(q), we will consider it in blockdiagonal matrix E. Hence we can write block-diagonal matrix as $E = diag(I, B(x_1))$. Also, $y = h_{cart}(x_1)$ denotes the position of the end-effector's tip. All the matrices and co-ordinates have been considered in the Cartesian coordinate system. Here, note attentively that the output equation has not been written directly as $y = x_1$, for a specific reason, which is mentioned in the next paragraph. We have considered the cartesian coordinates in the operational space of the robot.

Generalized coordinates vs. Operational space:
The term generalized coordinates refers to a representation of the system that uniquely defines its configuration in the real-time. For example, if our robot has 6 degrees of freedom, then there are 6 state variables, such that when all of these variables are specified with some value, we can easily calculate for the position and orientation of

the robot with the help of forward kinematics.

The problem with generalized coordinates, however, is that for planning trajectories in this space for desired tasks we intend to perform seems not to be straight forward. Consider for example, if we have a robotic arm, and we want to control the position of the end-effector, it's not obvious how to control the end-effector position by specifying a trajectory for each of the joints to follow through joint space.

Hence, the idea behind introducing operational space control is to overcome the limitations of the generalized coordinates of the system and plan a trajectory in a coordinate system that is directly relevant to the desired task needed to be accomplished. Therefore, our objective is to design a controller system that allows us to specify a trajectory or path in the operational space and will transform that trajectory signal into generalized coordinates> Finally,the controller can then send out these converted signals to the system for execution.

Finally, we denote the parameterization of $p(\theta)$ as the set of polynomial splines. And we can describe the path P by

$$p(\theta) = \sum_{i=1}^{N_p} H(\overline{\theta}_i - \theta)H(\theta - \overline{\theta}_{i-1}) \sum_{j=0}^{N_o} a_{i,j}\theta^j$$

where $H : R \rightarrow \{0, 1\}$ denotes the Heaviside step function and the polynomial coefficients are denoted as $a_{i,j}$. Also, as seen from the upper limits of the summation, N_p and N_o denote the number of path segments and the order of each polynomial respectively.

8 Simulations and results

8.1 Real-time interfacing and laboratory simulation:

The KUKA robot arm was interfaced with the designed model predictive control scheme via Fast Research Interface and an external ethernet connection. With the help of interface, torques can be superposed on each joint. Furthermore, the faster calculations were achieved by relying on the internal functionalities of the robot, which allowed for gravity compensation, hence neglecting the same in OCP.

Then, for the MPC controller parameters, the variables were set to following values: prediction horizon of 100ms, sampling time of 1ms. One important note is that in continuous time formulations of NMPC, the choice of input parameterization is dependant to some extent on the chosen sampling period. But, in discrete-time formulation of the same, input parameterization is independant of the sampling period. However, due to the use of research interface, the writers of journal paper could only obtain the joint angles x_1 from the magnetoresistive encoders but not the the joint velocities x_2. Hence the state $z = (\theta, \dot{\theta})^T$ has been considered as the internal state of the controller, and not included in the estimation.All this laboratory simulation was done on the basis of sampling rate of 1 kHz, which corresponds to the fastest sampling rate available via the research interface.

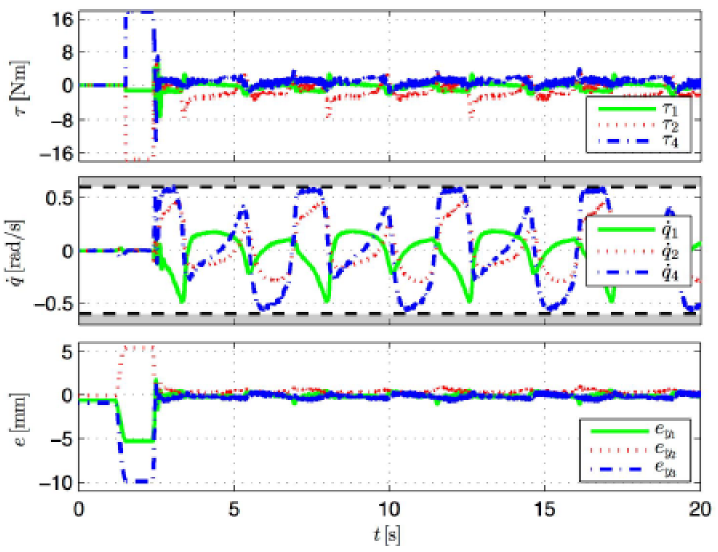

Fig 8.1

Since the MPC has been designed for path-following trajectories, the behaviour of inputs and outputs will greatly depend on the type of the reference trajectory. So if the trajectory is closed path then we will get periodic behavior of the inputs and outputs due to the speed assignment.This Fig 8.1 is one of the results for the path-following by robot on the three-leaved cloverpath. As seen in the third part of the figure, the MPFC scheme reduces the path error due to disturbance rapidly and allows the robot tip to follow the path accurately, thenafter. Also, the implementation of this MPFC scheme will lead to acceleration of robot tip along the straight parts/segments of the path, while slowing down the speed at the sharp corners.

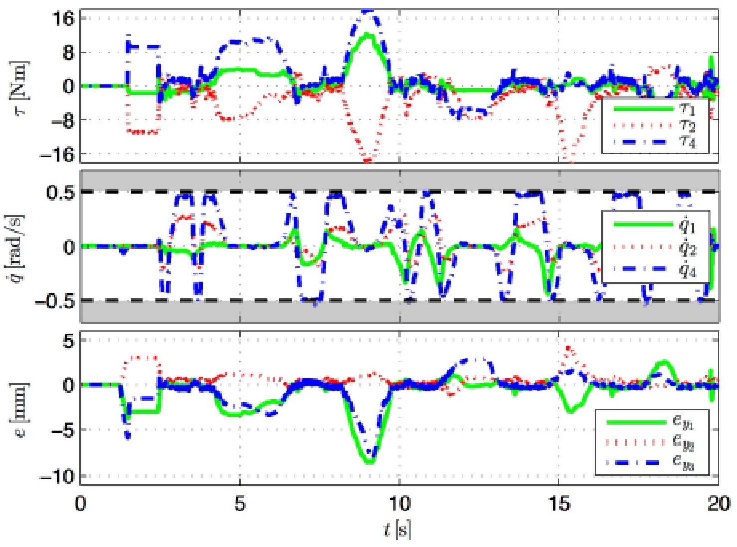

Fig 8.2

Another trial of the implementation was done by considering a random path, say in this case word 'Hello', wherein again the responses of the output path-following were corroborated by applying additional disturbances to the robot arm at intermittent intervals. Corresponding to the instant when the disturbances are applied, the robot either stops or moves away from the path(Fig 8.2). But the MPFC controller takes care of it by steering the robot tip back to the path, because when disturbances are applied, the controller is not switched OFF, rather reacts by slowing down or stopping the reference motion along the path.

8.2 Implementation of non-linear system in MATLAB and controlling it with MPC:

The system, whether it's linear or non-linear can be implemented with a MPC controller, by designing appropriate closed loop systems, which mainly consists of the plant(system itself) and the controller. Also, there needs to be some reference inputs againts which the system's response can be compared to.

For the purpose of designing the MPC, we need to take into account various factors like measured and unmeasured disturbances, manipulated variables, and measured and unmeasured outputs. These variables are essential for consideration in the overall architecture of the MPC scheme.The basic MPC modelling will include following models:

1. **Plant model:** It describes the system in either the form of transfer function, state-space representation. While specifying the plant model, MPC controller will perform all estimations and optimization calculations using a discrete-time, delay-free, state-space system.

2. **Input Disturbance model:** It is important in the sense that it influences following controller performance attributes:

   (a) Dynamic response to apparent disturbances

   (b) Rejection of sustained disturbances

   If we don't provide the input disturbance model, then the controller uses a default model, which uses integrators with unity gain.

3. **Output disturbance model:**Here the output disturbance is directly added to the plant output, thereby not affecting the plant states.Again if we don't specify this model, controller selects the default model with integrators.

4. **Measurement noise model:** One of the important design objective for controller design is to distinguish the disturbances from the measurement noise, which should be ignored.

MATLAB Code:

```matlab
%% Control of a Nonlinear Plant with Model predictive control


%% To describe the model and linearize it

linear_plant = linearize('mpc_nlm1');


%% Design MPC Controller
sampling_period = 0.1;
pred_horizon = 15;
control_horizon = 3;
 %creating the object
object1_mpc = mpc(linear_plant,sampling_period,pred_horizon,
control_horizon);

%% Define the input and output parameters for the linearized system(model).
 % Assign names to I/O variables.
linear_plant.InputName = {'Position';'Velocity';'Torque'};
linear_plant.OutputName = {'Joint angle';'joint velocity'};
linear_plant.InputUnit = {'m' 'm/s' 'Nm'};
linear_plant.OutputUnit = {'deg' 'm/s'};


%% Now we impose constraints on the variables of MPC

object1_mpc.MV = struct('Min',{-3;-2;-1},'Max',{4;2;1},'RateMin',{-1000;
                 -1000;-1000});
object1_mpc.Weights = struct('MV',[0 0 0],'MVRate',[.2 .3 .2],'OV',[1 2]);


%% Design the closed-loop MPC for the obtained linearized model,
  % by adding the MPC block from the Model Predictive Control toolbox
closed_loop_1= 'mpc_lin1';
sim(closed_loop_1);


%% Modify the MPC design to track the ramp signals
% In order to track a ramp signal, use a triple integrator as an output
% disturbance model on both outputs.

disturbance_model = tf({1 0;0 1},{[1 0 0 0],1;1,[1 0 0 0]});
setoutdist(object1_mpc,'model',disturbance_model);

%% Modify the previous model'mpc_lin1' by using ramp signal block
% instead of step signal, and recreate the simulink model.

closed_loop_2 = 'mpc_lin_ramp1';
sim(closed_loop_2);

%% Remove the constraints to see the behaviour of MPC without constraints

object1_mpc.MV = [];
```

```
 % Reset output disturbance model to default.
setoutdist(object1_mpc,'integrators');

%% After removing costraints, MPC can be considered as Linear controller,
 % so we define analogous linear controller from the new MPC object
 % 'object1_mpc', which is set for default disturbances.
linear_contrllr = ss(object1_mpc,'r');

%% Provide the ouput reference signals to both MPC and Linear controller
 % simultaneously to see the behaviour of controller, and graphs of
 % generated trajectories.

refs = [1;1];                   % output references are step signals
cl_loop_3 = 'mpc_comp1';
open_system(closed_loop_1);
open_system(closed_loop_2);
sim(cl_loop_3);                 % starts simulation
open_system(cl_loop_3);         % Opens simulink model for viewing scopes

%% End of Code
```

In the implementation of above code, first a simple non-linear model was made with a basic transfer function and a non-linear function of the input. But, since controlling the non-linear plant and system is difficult due to complexity of resulting OCP, we first linearize the plant about its default operating point with the help of in-built MATLAB function.
Then we create the object of MPC. By creating object, we can assign the controller attributes, parameters and functions.Also, the manipulated variables are imposed with the constraints, along with the weights being assigned to each of them. This MPC object represents the block diagram of MPC when we use it in the Simulink models. Then, we run the simulation model of the closed loop control strategy obtained by including the MPC block along with the plant. At this stage of design, step input is given as the reference signal, which the outputs of the controller are supposed to follow.
Also, to verify the design for reference tracking, we change one of the reference input as the ramp signal, and observe the change in the control inputs and system outputs. And, one important fact to consider is that when we remove the constraints imposed earlier on the variables, the MPC will start behaving like the linear controller.Along with that, we set the output disturbance model to default, implying that there are no disturbances being added to the output signal trajectory.

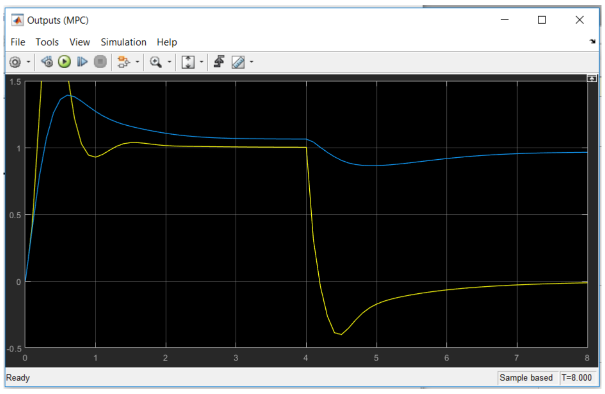

Fig 8.3

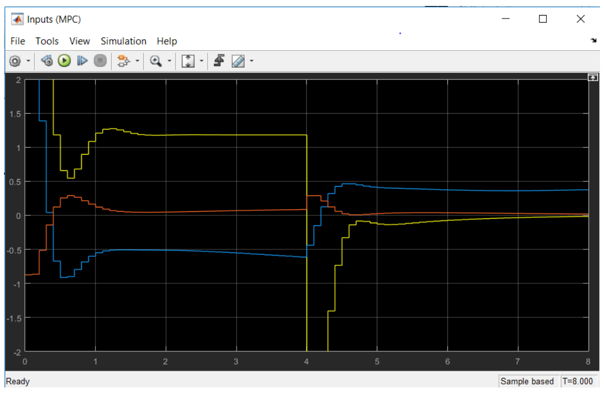

Fig 8.4

As seen from above two figures, we can see that the MPC controller design leads to control of non-linear system by leading the output trajectories to follow the desired trajectory. However, if we remove the constraints from the manipulated variables, then MPC behaves as the linear controller.
Also, the graph of the inputs(Fig. 8.4) shows the control inputs that generate the predicted state of the output. This computation is done by the model predictive controller on the basis of constraints being given on the system variables.

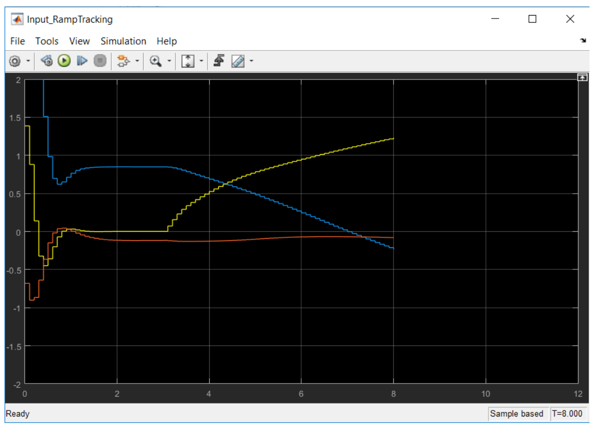

Fig 8.5

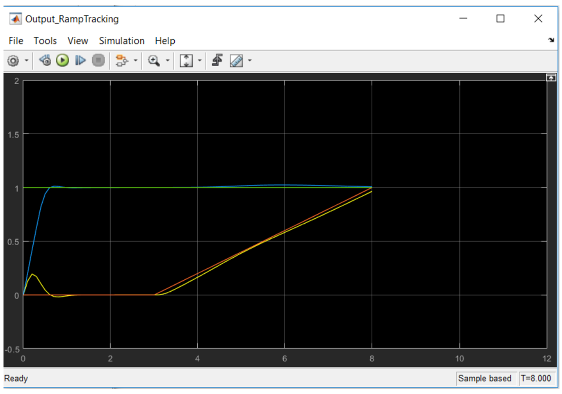

Fig 8.6

From above two figures, we can see that the path-following of the output for desired trajectory(ramp signal, in this case) has been implemented, and the output signals can be easily seen to follow the same.Thus, we can say that the MPC acts dynamically against the disturbances, and reduces the path error, to follow the reference trajectory.

9 Conclusions

Though there are many systems on which MPC can be used as the control strategy, we considered here the effectiveness of the proposed MPC design when applied to the industrial robot.

In this process described in the report, we defined some important parameters of the Model predictive control and the optimization problem, by considering the advantages of MPC over conventional controllers like proportional derivative(PD) or proportional integral derivative(PID) controller. As seen from the characteristics, one can employ MPC when one wants the output to follow the desired trajectory. Also, since there is the boom of the industrial applications like drilling, milling, painting, and other machining and fabrication process in this modern times, the use of MPC best meets the requirements, as it enhances the performance of the system, can deal with linear and non-linear systems easily, and solves the optimization problem at each time step in its algorithm to yield the control input that will achieve the desired trajectory.

Hence, we defined the path-following problem for an industrial robot, made some assumptions based on whether velocity is assigned as the parameter in design. Then, the design and implementation of procedures for continuous time non-linear model predictive control(NMPC) has been discussed. It is evident that while formulating the dynamics of the system, the design of the controller has to be done keeping the constraints into consideration, which again can be imposed either on present states of the system, control input or output of the system. This led to the formulation of constrained path-following problem with and without speed(velocity) assignment. For the purpose of starting with solution to this problem, the adaptive control law(timing law) has been derived , and then its simplified version is used for further calculations.

Finally, the strategy to achieve the desired output through MPC is formulated with the help of OCP. The cost function is defined and minimized along with the constraints being solved iteratively, to generate the optimal sequence of the control inputs. And, the augmented system was described for the path-following problem so as to allow the path and its parameter to be described in the operational space, as compared to generalized coordinates. Thus, with all these assumptions, remarks discussed, and conditions for convergence, we have come to conclusion that MPC can be used for real-time feasible and high-performance control of the industrial robot.

10 References and Bibliography

1. A. P. Aguiar, J. P. Hespanha, and P. V. Kokotovi´c, “Pathfollowing for nonminimum phase systems removes performance limitations,” IEEE Trans. Autom. Control, vol. 50, no. 2, pp. 234–239, Feb. 2005.

2. T. Faulwasser and R. Findeisen, “Nonlinear model predictive control for constrained output path following,” IEEE Trans. Autom. Control, vol. 61,no. 4, pp. 1026–1039, Apr. 2016.

3. V. Kumar, M. Žefran, and J. P. Ostrowski, “Motion planning and control of robots,” in Handbook of Industrial Robotics. New York, NY, USA:Wiley, 1999, pp. 295–315.

4. T. Faulwasser, J. Matschek, P. Zometa, and R. Findeisen, “Predictive path-following control: Concept and implementation for an industrial robot,” in Proc. IEEE Conf. Control Appl. (CCA), Hyderabad, India, Aug. 2013, pp. 128–133.

5. R. Skjetne, T. I. Fossen, and P. V. Kokotovi´c, “Robust output maneuvering for a class of nonlinear systems,” Automatica, vol. 40, no. 3, pp. 373–383, Mar. 2004.

6. Fernando Fontes, "A General Framework to Design Stabilizing Nonlinear Model Predictive Controllers" .

7. Recep Burkan, Ibrahim Uzmay, "A model of parameter adaptive law with time varying function for robot control", Elsevier, 17 Nov. 2004.

8. B. Houska, H. J. Ferreau, and M. Diehl, “An auto-generated realtime iteration algorithm for nonlinear MPC in the microsecond range,” Automatica, vol. 47, no. 10, pp. 2279–2285, 2011.