"Preparation of CAD software for design and analysis of base frames of standard industrial gearboxes "

A Major Project Report

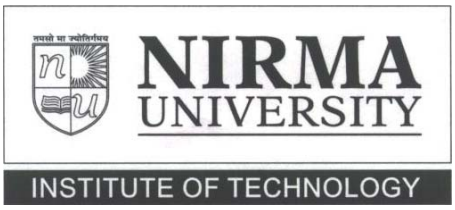
Submitted in Partial Fulfillment of the Requirements for the Degree of

MASTER OF TECHNOLOGY

IN

MECHANICAL ENGINEERING (CAD / CAM)

By
Narahari Sreenadh
(Roll No.03MME008)



Department of Mechanical Engineering
INSTITUTE OF TECHNOLOGY
NIRMA UNIVERSITY OF SCIENCE & TECHNOLOGY,
AHMEDABAD 382 481

MAY 2005

Certificate

This is to certify that the Major Project Report entitled " Preparation of CAD software for design and analysis of base frames of standard industrial gearboxes " submitted by Mr./Ms. Narahari Sreenadh (Roll No.03MME008), towards the partial fulfillment of the requirements for the award of Degree of Master of Technology in Mechanical Engineering (CAD/CAM) of Nirma University of Science and Technology is the record of work carried out by him/her under my/our supervision and guidance. The work submitted has in my/our opinion reached a level required for being accepted for examination. The results embodied in this major project work to the best of my/our knowledge have not been submitted to any other University or Institution for award of any degree or diploma.

Project Guide/Guides **i.Mr.V.B.Kalyankar, GM, Design, Gear, ELECON**

ii. Mr.S.S.Das, Assistant Prof., NIT

(Prof. A.B. Patel)
Head of Dept.
Mechanical Engg. Dept.

(Dr. H.V. Trivedi)
Director
Institute of Technology
Nirma University

Examiners : **i.**

ii.

iii.

iv.

(Name and Signature)

Acknowledgement

First and for most I would like to mention my sincere thanks to Mr.V.B.Kalyankar, GM, design, gear division, ELECON for giving me this opportunity to do this project and his guidance in difficult times.

I would also like to use this opportunity to thank my guide Mr.K.K.Das, Mechanical, NIT for his guidance through out this project.

I would like to thank Mr.A.B.Patel, HOD, Mechanical, NIT for his continuous encouragement and guidance for completing this project.

I would also like to thank Mr.P.B.Popat, course coordinator, M.Tech (CAD/CAM) for giving me his advice.

I am also thankful to Mr. Sandeep Mitra, Sr.Manager, ELECON for his support and guidance

I would also like mentioning my sincere thanks to the staff of ELECON like Mr.N.G.Pandya, Mr. Vaju sanjalia, Mr. Samir.J.

I would also like to thank all the faculty of mechanical department for their support and guidance.

Narahari Sreenadh

Abstract

The objective of this project is to develop CAD software to automatically design and analyze the base frames for standard type of industrial gearboxes. This project can be divided into two main parts. In the first part, the objective is to develop CAD software for drawing base frames of standard industrial gearboxes using Auto CAD 2000. Second part of the project deals with the stress analysis of the base frame using ANSYS program.

For development of CAD software Auto LISP language was used. Design of these base frames depends on the gearbox dimensions, which are taken from the Elecon catalogues. This software is developed keeping in mind the user friendliness and flexibility in designing. Here in this project the method employed for software development was explained. The algorithm used for development was also mentioned. This software can be run using Auto CAD 2000. While execution of this software it will give option to select gearbox type and size. By selecting gearbox type and size we can generate drawings of base frame using this software. This software was also provided with a facility to draw new designs. For this we need to supply the parameters of the gearbox first time and these can be utilized again and again to draw base frames of new gearboxes as well.

Stress analysis of the base frame can be performed using a macro that is developed in APDL (Ansys parametric design language). This macro is used to develop the model of the base frame for the stress analysis. This macro takes the parameters of base frame as input. Using this input it generates the model. Force analysis is taken up to find the forces that are coming on the base frame from gearbox to use in analysis. ANSYS workbench software is used for stress analysis. From the analysis equivalent stress, maximum shear stress, maximum deflection, safety factor and critical stress zones were found out. Modal analysis of the base frame was also done. From the modal analysis natural frequencies for different modes are found.

List of figures

List of tables

Nomenclature

L1	length of base frame in front view
TL	total length of base frame in front view
W1	width of base frame in top view
TW	total width of base frame in top view
TH	total height of the base frame in side view
P	pitch of the stiffeners
ST	stiffener thickness
PTB	bottom plate thickness
PLB	bottom plate length
PWB	bottom plate width
PTT	top plate thickness
PLT	top plate length
PWT	top plate width
HFB	height of the foundation bolt
SFB	size of the foundation bolt
Combined length	total length of base frame for motor and gearbox

CONTENTS

Chapter 1

Introduction

1.1 Company profile

This project is an in plant live project. This project is done at ELECON ENGG. CO. LTD., vallabh vidhyanagar. This company manufactures industrial reduction gearboxes. It supplies reduction gearboxes to various industries like cement, paper mills, steel plants, sugar mills, mines, rolling mills, and etc. this company is stared at vidhyanagar in 1961. In the beg-inning it is making mining equipment as well as material handling equipment for industries. In 1971 it stared manufacturing of industrial reduction gearboxes.

The product range for the gear division of the company include [1.1]

Helical gearboxes

Bevel helical gearboxes

Planetary gearboxes

Worm gears

Couplings and etc.

The infrastructure of the company is one of the best for gear making in India. It includes latest CNC machines like

CNC machines used in elecon

Sinumeric (siemens) machining centre

Fanuc india series

Features

Two tables A,B

Both horizantal and vertical machining jobs

automatic tool changer (total 5 tool magzines)

Mazak power centreH-20

Automatic work table changer

At atime we can have 5 works in Q

For medium and low size jobs

Automatic tool changer

Juaristi

Sr no 9-150-2
Type MDR 165
Spindle dia 165mm
Table size 1700X2000mm
Spindle axial travel 950mm
Weight on table 8000kgs
Vertical/cross travel 3000mm
Long travel 2600mm
Installation 1982

Pfuater p1200nc

Hobing machine
Spur, helical gear cutting
Magnetic chip conveyor
6-axis machine

Mazak integrex 50 and 30

Turning centre
Automatic chip conveyor

Klingelnberg kcn60

8 axis spirel,bevel gear cutting machine
For rough and hard cut two different tools
For different modules different cutters are used
Cutter head radius depends on o.d of job

In addition to above mention manufacturing equipment this company is having very well equipped design department with drafting software like Auto CAD, modeling software like Pro-E. In design department they are designing all types of gearboxes and couplings using above-mentioned software. Mostly they are using Auto CAD for drafting of designs.

1.2 About the project

The objective of this project is to develop CAD software to automatically design and analyze the base frames for standard type of industrial gearboxes. This project can be divided into two main parts. In the first part, the objective is to develop CAD software for drawing base frames of standard industrial gearboxes using Auto CAD 2000. Second part of the project deals with the stress analysis of the base frame using ANSYS program.

For development of CAD software Auto LISP language was used. Design of these base frames depends on the gearbox dimensions, which are taken from the Elecon catalogues. This software is developed keeping in mind the user friendliness and flexibility in designing. Here in this project the method employed for software development was explained. The algorithm used for development was also mentioned. This software can be run using Auto CAD 2000. While execution of this software it will give option to select gearbox type and size. By selecting gearbox type and size we can generate drawings of base frame using this software. This software was also provided with a facility to draw new designs. For this we need to supply the parameters of the gearbox first time and these can be utilized again and again to draw base frames of new gearboxes as well.

Stress analysis of the base frame can be performed using a macro that is developed in APDL (Ansys parametric design language). This macro is used to develop the model of the base frame for the stress analysis. This macro takes the parameters of base frame as input. Using this input it generates the model. Force analysis is taken up to find the forces that are coming on the base frame from gearbox to use in analysis. ANSYS workbench software is used for stress analysis. From the analysis equivalent stress, maximum shear stress, maximum deflection, safety

factor and critical stress zones were found out. Modal analysis of the base frame was also done. From the modal analysis natural frequencies for different modes are found.

First chapter of this project gives an introduction to company profile as well as about the project. Second chapter consist of the articles of literature review that are carried out during this project. Third chapter of the project is consisting of data collection during the project, programs that are used in the software, process of using the software for base frame generation, macros developed in APDL for modeling, stress analysis of base frame using ANSYS workbench.

Fourth chapter is results and discussion. In this chapter results obtained from ANSYS were given. Fifth chapter suggest some future developments for this project. Sixth chapter consists of summery of project.

At the end of there are references used in this project work as well as appendixes.

Chapter 2

Literature review

2.1 Design of structures

A great number of members in a typical steel structure are beam columns. Beam columns are defined as members subject to combined bending and compression. They are therefore elements, which comprise the special cases of beams and columns. In principle all members in frame structures are beam columns with the particular cases of beams (where N=0) and columns (where M=0). To treat a frame several possibilities may be used. The development of large capacity computers for a reasonable prize has made it possible to investigate the structures as a whole. This is very easy if theory of elasticity is used. But it is much more complicated if the theory of plasticity is used and especially the spread of plastic zones should be taken into account, simplified rules are still needed [2.1.1].

In general the concept of non-linear analysis is applied to stability design of frames with columns subjected to flexural buckling. Code specifications require in this case that the structure is stable and maximum stresses are less than ultimate allowable stresses under the action of design loads. For structures, not subjected to fatigue loading, load carrying capacity is increased by limit states design based on ultimate cross section capacity. While structure is in stable equilibrium one or more plastic hinges are allowed to form. The condition of stable equilibrium is proved by a geometrically nonlinear analysis taking into account equilibrium in deformed configuration. Assuming that displacements are small, this leads to theory second order analysis [2.1.2].

Local buckling, lateral tensional buckling and member buckling including local stability failure however is extensively analyzed by considering single beams and columns, mentally cut out of the entire structure. Applied forces and internal forces at both ends load those members. Though common to all design codes this approach neglects the real interaction between different elements in a structural system. Limit states design in this case is based on interaction equations in which maximum internal forces are compared with reduced cross section capabilities. Capacity reduction forces depend both on strength of materials and slenderness, derived from buckling loads or critical buckling length. While numerous design aids are available to calculate slenderness parameters for lateral tensional buckling and local plate buckling, only few designs oriented publications offer explicit support for solving the combined problem. An extensive review of lateral-distortional buckling with respect to steel I-sections in given in [2.1.3].

Load carrying capacity of thin-walled steel box columns is influenced by overall buckling of column or by local buckling of component plates or by the interaction between overall column buckling and local buckling. Researchers have proposed a number of empirical or semi-empirical

methods using average stress-strain curves or effective width concept at different stages. The purpose of this paper [2.1.4] is to present brief review of some of these methods.

Generalized functions are indispensable to solve the problems when there is singularity, discontinuity or change in expression. The use of generalized functions in solving different types of problems in structural engineering was presented in [2.1.5]. This included the solution of problems to determine critical loads for columns with one singularity either in flexural rigidity or in the axial load. Generalized functions made it possible to solve all such problems for the columns with four standard conditions.

A typical column may have step variations of flexural rigidity at a number of sections. Such problems have been solved by numerical methods or by some approximate methods. The accuracy of these solutions is not known. Besides such solutions are useful for the numerical data employed and not for a class of similar problems. In this paper [2.1.6] generalized functions are used to solve such problems. Eigenvalue conditions are represented in the form, which is convenient to notice the trend of the terms for more singularities in the flexural rigidity.

2.2 Finite element analysis

A Brief History

R. Courant, who utilized the Ritz method of numerical analysis and minimization of variational calculus to obtain approximate solutions to vibration systems, first developed finite Element Analysis (FEA) in 1943. Shortly thereafter, a paper published in 1956 by M. J. Turner, R. W. Clough, H. C. Martin, and L. J. Topp established a broader definition of numerical analysis. The paper centered on the "stiffness and deflection of complex structures".

By the early 70's, FEA was limited to expensive mainframe computers generally owned by the aeronautics, automotive, defense, and nuclear industries. Since the rapid decline in the cost of computers and the phenomenal increase in computing power, FEA has been developed to an incredible precision. Present day supercomputers are now able to produce accurate results for all kinds of parameters [2.2.1].

Finite Element Analysis

FEA consists of a computer model of a material or design that is stressed and analyzed for specific results. It is used in new product design, and existing product refinement. A company is able to verify a proposed design will be able to perform to the client's specifications prior to manufacturing or construction. Modifying an existing product or structure is utilized to qualify the product or structure for a new service condition. In case of structural failure, FEA may be used to help determine the design modifications to meet the new condition.

There are generally two types of analysis that are used in industry: 2-D modeling, and 3-D modeling. While 2-D modeling conserves simplicity and allows the analysis to be run on a relatively normal computer, it tends to yield less accurate results. 3-D modeling, however, produces more accurate results while sacrificing the ability to run on all but the fastest computers effectively. Within each of these modeling schemes, the programmer can insert numerous algorithms (functions) which may make the system behave linearly or non-linearly. Linear systems are far less complex and generally do not take into account plastic deformation. Non-linear systems do account for plastic deformation, and many also are capable of testing a material all the way to fracture.

FEA uses a complex system of points called nodes, which make a grid called a mesh. This mesh is programmed to contain the material and structural properties, which define how the structure will react to certain loading conditions. Nodes are assigned at a certain density throughout the material depending on the anticipated stress levels of a particular area. Regions that will receive large amounts of stress usually have a higher node density than those, which experience little or no stress. Points of interest may consist of: fracture point of previously tested material, fillets, corners, complex detail, and high stress areas. The mesh acts like a spider web in that from each node, there extends a mesh element to each of the adjacent nodes. This web of vectors is what carries the material properties to the object, creating many elements.

Wide ranges of objective functions (variables within the system) are available for minimization or maximization:

- Mass, volume, temperature
- Strain energy, stress strain
- Force, displacement, velocity, acceleration

- Synthetic (User defined)

There are multiple loading conditions, which may be applied to a system. Some examples are shown:

- Point, pressure, thermal, gravity, and centrifugal static loads
- Thermal loads from solution of heat transfer analysis
- Enforced displacements
- Heat flux and convection
- Point, pressure and gravity dynamic loads

Each FEA program may come with an element library, or one is constructed over time. Some sample elements are:

- Rod elements
- Beam elements
- Plate/Shell/Composite elements
- Shear panel
- Solid elements
- Spring elements
- Mass elements
- Rigid elements
- Viscous damping elements

Types of Engineering Analysis

Structural analysis consists of linear and non-linear models. Linear models use simple parameters and assume that the material is not plastically deformed. Non-linear models consist of

stressing the material past its elastic capabilities. The stresses in the material then vary with the amount of deformation.

Vibrational analysis is used to test a material against random vibrations, shock, and impact. Each of these incidences may act on the natural vibrational frequency of the material, which, in turn, may cause resonance and subsequent failure.

Heat Transfer analysis models the conductivity or thermal fluid dynamics of the material or structure. This may consist of a steady state or transient transfer. Steady-state transfer refers to constant thermoproperties in the material that yield linear heat diffusion.

2.3 The stiffness method

Matrix procedures used in the analysis of framed structures and other finite element structures are described. The plane truss is used as the principle vehicle for the discussion.
Certain matrix procedures of structural mechanics are described. These methods are also used in finite element analysis of many other physical problems. These procedures include assembly of elements to form a structure, imposition of boundary or support conditions, solution of simultaneous equations to obtain nodal quantities, and processing of elements to obtain quantities as stresses or flows.
A structure has n d.o.f. if n independent quantities are needed to uniquely define configuration of the structure.

Structural stiffness equation

We begin by generating the structure stiffness matrix [K] of a plane truss by a direct attack on the structure as a whole. Later we will show how [K] can be built by assembly of element matrices, which is the process actually used in computer programs.

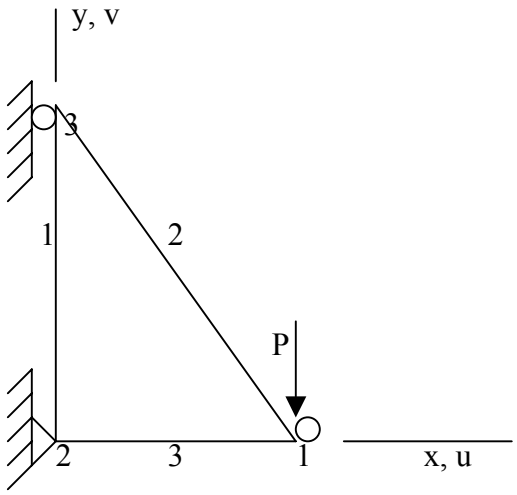


Fig2.1 : Three bar truss

Consider, for example, the three bar truss of fig. Nodes and elements are numbered arbitrarily. For element i, where i = 1, 2, 3 in this example. Let A_i = cross-sectional area, E_i = elastic modulus, and L_i = length. From the elementary mechanics of materials, axial force F_i and change in length e_i have the relation

$$e_i = \frac{F_i L_i}{A_i E_i}$$

Stiffness is defined as the ratio of force to displacement and is by custom given the symbol k. Thus, the axial stiffness of any uniform bar of a truss is

$$k_i = \frac{F_i}{e_i} = \frac{A_i E_i}{L_i}$$

In our standard abbreviation, the structure stiffness equations are

$$[K]\ \{D\} = \{R\}$$

Where [K] is the structure stiffness matrix.

The physical meaning of [K], as well as procedure for formulating [K], are contained in the fallowing statement. The jth column of [K] is the vector of loads that must be applied to nodal d.o.f. in order to maintain the deformation state associated with unit value of d.o.f. j while all other nodal d.o.f. are zero. For a frame, “loads” include moments as well as forces. By this

procedure-activating one d.o.f. at a time-we can generate the stiffness matrix of any truss of frame, regardless of the number of bars or the degree of state indeterminacy. The stiffness matrix is square; that is, there are as many equations as there are d.o.f. {D} is the displacement vector; {R} is vector representing the forces at nodes.

Properties of [K]

In general, for any structure, no diagonal coefficient K_{ii} is negative or zero unless the structure is unstable.

[K] is symmetric. This is true of any structure that displays a linear relationship between applied loads and the resulting displacements.

For an unsupported structure, (a) [K]{D} = {0} when {D} represents rigid body motion and (b) each column of [K] represents a set of nodal forces and/or moments in static equilibrium.

Solution for unknowns

The stiffness matrix is singular. One must remove the singularity of [K] in order to solve for the unknown d.o.f. in {D}. We now show a formal procedure by which this may be done. Let $\{D_c\}$ and $\{R_c\}$ be known d.o.f. and known loads, and $\{D_x\}$ and $\{R_x\}$ be as yet unknown d.o.f. and loads. By partitioning, accompanied by such rearrangement of matrix coefficients as may be necessary, the structural equations [K]{D} = {R} can be written in the form

$$\begin{bmatrix} K_{11} & K_{12} \\ K_{21} & K_{22} \end{bmatrix} \begin{Bmatrix} D_x \\ D_c \end{Bmatrix} = \begin{Bmatrix} R_c \\ R_x \end{Bmatrix}$$

Or, in a more expanded form.

$$\begin{aligned} [K_{11}]\{D_x\} + [K_{12}]\{D_c\} &= \{R_c\} \\ [K_{21}]\{D_x\} + [K_{22}]\{D_c\} &= \{R_x\} \end{aligned}$$

$[k_{11}]$ is nonsingular if the prescribed d.o.f. $\{D_c\}$ are sufficient in arrangement and number to prevent rigid body motion. Therefore, the unknown d.o.f. $\{D_x\}$ can be found from

$$\{D_x\} = [K_{11}]^{-1}(\{R_c\} - [K_{12}]\{D_c\})$$

Finally, unknown loads $\{R_x\}$ can be found after substituting of d.o.f. $\{D_x\}$, which are now known. In structural mechanics, $\{R_x\}$ usually represents support reactions.

Element stiffness equations

In practice, [K] is built by summation of coefficients from element stiffness matrices [k]. The summation process easily computerized.

We will show the element stiffness equation of the element shown below

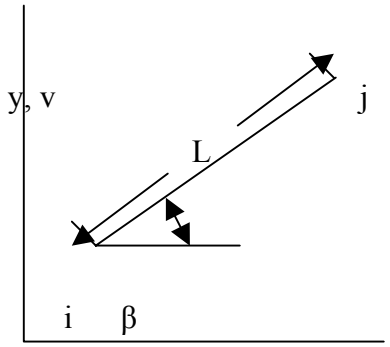


Fig 2.2 : Simple bar element

$$\frac{AE}{L}\begin{bmatrix} c^2 & cs & -c^2 & -cs \\ cs & s^2 & -cs & -s^2 \\ -c^2 & -cs & c^2 & cs \\ -cs & -s^2 & cs & s^2 \end{bmatrix}\begin{Bmatrix} u_i \\ v_i \\ u_j \\ v_j \end{Bmatrix} = \begin{Bmatrix} p_i \\ q_i \\ p_j \\ q_j \end{Bmatrix}$$

Where c = cos β and s = sin β

$L=[(x_j-x_i)^2 + (y_j-y_i)^2]^{1/2}$

The square matrix, including the factor AE/L, is the element stiffness matrix. We abbriviate

[k]{d} = {r}

Node numbering that exploits matrix sparsity

A finite element structure with many d.o.f. has a sparse coefiicient matrix [K]. That is, most of the individual coefficients K_{ij} are zero. Sparsity should be exploited in order to economize on

computr storage space and running time. Sparsity may be exploited by various schemes. In this present section we emphasize bandness, which is among the simpler schemes.

The number of nonzero coefficients in [K], and thier numerical values, are independent of how structure nodes are numbered. A change in structure node numbers change only the arrangement of nonzero K_{ij}. The semibandwidth (also callded the half bandwidth) is given the symbol b. Matrix [K] is symetric and has a total bandwidth 2b-1. bandwidth 2b-1 indicates the horizantal span of the zone in which all nonzero K_{ij} reside.this zone lies along the principal diagonal of [K]. Some zeros may appeare within the band, but only zeros appeare outside of it. A small semibandwidth is usually achieved by placing consecutive node numbers along the shorter dimension of a structure. By this way we can reduce the space and run ile on the computer. For example, if n_{eq} = 10b, then the time needed to solve for d.o.f. {D} is reduced by a factor of about 30.

Stress computation

After solving the global equations [K]{D} = {R} for {D}, all nodal d.o.f. of the structure are known. To compute stress in a given element we extract nodal d.o.f. {d} of that element from {D}, compute element strains from {d}, and finally compute stresses from strains .

Elongation e of a plane truss bar is computed from components of nodal d.o.f. parallel to the bar

$$e = (u_j - u_i)\cos\beta + \sin\beta$$

Axial strain is $\varepsilon = e/L$. The bar is in uniaxial stress. Therfore, the axial stress caused by strain is

$$\sigma = E\varepsilon = E\frac{e}{L}.$$

Support reactions

A particular reaction R_i in the list $\{R_x\}$ can be computed as

$$\sum_j K_{ij} D_j = R_i \; or \sum_m \left(\sum_j k_{ij} d_j \right) = R_i$$

2.4 Structural Mechanics

Introduction

From ancient times humans have been preoccupied with the planning, design, and construction of structures.

Planning a structure involves the selection of the most suitable type of structure and the choice of its general layout and overall dimensions on the basis of economic, aesthetic, functional, and other criteria.

Designing a structure entails determining the disturbances (external forces, change of temperature, etc.) to which it is expected to exposed during its lifetime and then choosing the dimensions of its members as well as the details of their connections. The structure is then analyzed, that is, the internal forces and moments in its members and the displacements of some of its cross sections are computed. The components of stress acting at any point on a cross section of a member of a structure are established from the internal forces and moments acting on this cross section. The members of a structure must have sufficient strength and rigidity so that when the structure is subjected to the disturbances to which it is expected to be exposed, the components of the stress and displacement at any of its points do not exceed the maximum allowable values given in the appropriate design codes. Moreover, the members of a structure must be such that the structure, or any of its parts, does not reach a state of in stability (buckling) when subjected to the disturbances, which are expected to act on it. If the results of the analysis show that the members of a structure do not have sufficient strength and rigidity to satisfy the aforementioned requirements, the structure is redesigned. That is, new dimensions of the cross section are chosen for some of its members, and the resulting structure is reanalyzed. The process is repeated until a structure is obtained which satisfies all the aforementioned requirements. Moreover, when the analysis of a structure indicates that some of its members are not stressed sufficiently, the structure is redesigned and reanalyzed.

On the basis of forgoing, it is apparent that structural analysis is an integral part of the design of structures; consequently, every competent structural designer must be well versed in structural analysis.

Idealizations in structural analysis

A structure made of line members joined together is referred to as a frame structure.

We limit our attention to straight-line members, which either have constant cross section or cross section s whose geometry changes so that the direction of their principle centroidal axes remains constant throughout their length. Moreover, we limit our attention to curved members whose axis lies in one plane, whereas one of the principle centroidal axes of their cross sections is normal to this plane.

Framed structures may be classified as

Planer

Space

The joints of planar frame structure are usually idealized as rigid joints or pinned joints.

The supports of framed structure are idealized as:

Roller

Hinged

Fixed

Loads on structures

We refer to the disturbances, which cause internal forces and moments in the members of a structure and/or displacements of its points as the loads on the structure. These disturbances may be classified as

1. External actions
2. Displacements of the supports
3. Changes of environment conditions (usually change of temperature)
4. Initial stresses

The establishment of the loads acting on a structure is one of the most important steps in the process of designing this structure because the accuracy of the results of its analysis depends on the accuracy of the loads used.

The loads acting on the structure may be classified as

Static

Quasi-static

Dynamic

Classification of framed structures

Framed structures may be classified on the basis of how their members resist the applied loads as:

1.structures whose members are primarily subjected to an axial force, which induces mainly uniformly, distributed axial tensile or compressive components of stress. These structures include cables, cable structures, and trusses (planar or space).

2.strucures whose members are subjected to shearing forces, axial forces, bending moments, and possibly torsional moments, including a nonuniform distribution of the axial component of stress on their cross sections. These structures include beams, arches, planar frames, grids, and space frames.

3.structures with some members subjected only to uniform distribution of axial stress and others subjected to a nonuniform distribution of the axial stress.

Frames

Frames are most general type of framed structures. Their members are subjected to axial and shear forces, bending moments, and, possibly, torsional moments. They can have both rigid and nonrigid joints and can be loaded in any way. Usually, frames are space structures. Frequently, however they can be broken down into parts, which can be considered as planar frame or grid. The members of grid also lie in one plane. However, the external forces are normal to the plane of the grid and the vector of the external moments lies in this plane.

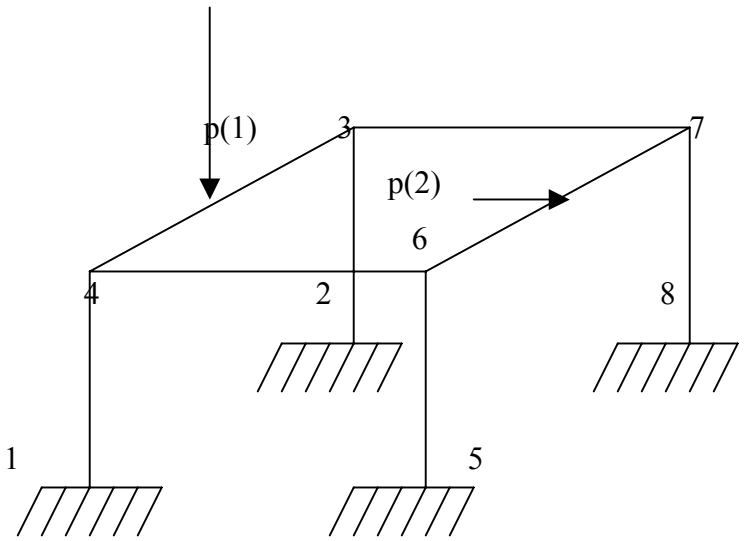


Fig 2.3 : space frame

Free body diagrams

A free-body diagram is a sketch of a part of a structure showing all external actions and all possible reactions acting on it, as well all internal actions acting on the cross sections where the part under consideration is cut from the structure. Usually, in a free-body diagram either the local or the global components of the internal actions are shown as assumed positive.

Restrictions in the analysis of framed structures

We only consider framed structures, which have the fallowing attributes:

1. They are in equilibrium under the influence of the external loads. This implies that if we isolate any part of the structure and draw its free-body diagram, the sum of the forces acting on the part, as well as the sum of the moments about any conveniently chosen point of the actions acting on the part, must vanish.
2. The ratio of the thickness to the radius of curvature of curved line members is very small compared to unity. In general, the distribution of the normal component of stress on the cross section of curved members subjected to forces acting in their plane is hyperbolic.

However, this distribution can be approximated by a linear distribution when the ratio of the thickness to the radius of curvature of the members is small compared to unity.

3. The structures or any group of their members cannot move without deforming, under the influence of any loading.
4. The deformation of the members of the structures is within the range of validity of the theory of small deformation. This theory is based on the assumption that the deformation of a structure is such that
    a. The change of length par unit length of any infinitesimal material line of the structure (unit elongation) is negligible compared to unity
    b. The change of angle between any two mutually perpendicular infinitesimal material lines of the structure (unit shear) is negligible compared to unity
    c. The angles of rotation of the partials of the structure are negligible compared to unity and not of a higher order of magnitude than the unit elongations and the unit shears

As a result of these assumptions, the fallowing approximations are valid for structures whose deformation is within the range of validity of the theory of small deformations:

1. The deformation of a particle is completely specified by its six components of strain which referred to the rectangular system of axes x_1, x_2, x_3 are denoted by e_{11}, e_{22}, e_{33}, $e_{12} = e_{21}$, $e_{13} = e_{31}$, and $e_{23} = e_{32}$. The components of strain of a particle are related to its components of displacement u1, u_2, and u_3 by the fallowing linear relations:

$$e_{11} = \frac{\partial \overline{u}_1}{\partial x_1} e_{22} = \frac{\partial \overline{u}_2}{\partial x_2} e_{33} = \frac{\partial \overline{u}_3}{\partial x_3}$$

$$e_{21} = e_{12} = \frac{1}{2}\left(\frac{\partial \overline{u}_2}{\partial x_1} + \frac{\partial \overline{u}_2}{\partial x_1} \right)$$

$$e_{31} = e_{13} = \frac{1}{2}\left(\frac{\partial \overline{u}_3}{\partial x_1} + \frac{\partial \overline{u}_1}{\partial x_3} \right)$$

$$e_{32} = e_{23} = \frac{1}{2}\left(\frac{\partial \overline{u}_2}{\partial x_3} + \frac{\partial \overline{u}_3}{\partial x_2} \right)$$

The components of strain e_{ii} (i = 1, 2, 3) of a particle is referred to as the normal component of strain at the particle in the direction x_i. It represents the change of length per unit length of an infinitesimal line segment of length dx_i at the particle. The component of strain e_{ij} (i=1, 2, 3; $i \neq j$) of a particle is referred to as the shearing component of strain at the particle in the direction of the x_i and x_j axes. It represents half

the change due to the deformation of the angle between two mutually perpendicular infinitesimal line segments dx_i and dx_j at the particle.

2. The effect of the change due to the deformation of the dimensions of the cross sections of the members of the structure is disregarded when computing the components of stress from the internal actions.
3. The effect of the change of the geometry of the structure on the internal actions of its members negligible.
4. The angle between the tangent line of the deformed axis of a member and its undeformed axis is so small that its cosine is approximately equal to unity, while it sine is approximately equal to the angle in radians.
5. When the members of a structure are made of linearly elastic materials, the relation between the loading acting on the structure and the resulting internal actions in its members and the relations between the loading acting on the structure and the displacement of its cross section are linear.

Notice that we can distinguish two types of nonlinear behavior of structures:

1. Material nonlinearity. That is, the relations between the components of stress and strain are nonlinear.
2. Geometric nonlinearity. That is, the deformation of the structure is not in the range of validity of the theory of small deformation.

The theory of small deformations cannot be used in analyzing certain structures of interest to the structural designer when they are subjected to certain types of loading. For example, the theory of small deformation cannot be employed in the fallowing cases:

1. In analyzing beams subjected to transverse and axial forces when the effect of the axial forces on their bending moment cannot be neglected.
2. In establishing the loading under which a structure or a group of its members reaches a state of unstable equilibrium.
3. In analyzing kinematically unstable structures.

Kinematically unstable structures

The geometry of some of the structures can be such that for certain types of loading, a number of their members can move instantaneously without deforming. These structures are termed

kinematically unstable. The internal actions acting on the members of kinematically unstable structures cannot be computed if the change of their geometry due to their deformation is not taken into account. That is the internal actions in the members if kinematically unstable structure cannot be established using the theory of small deformation.

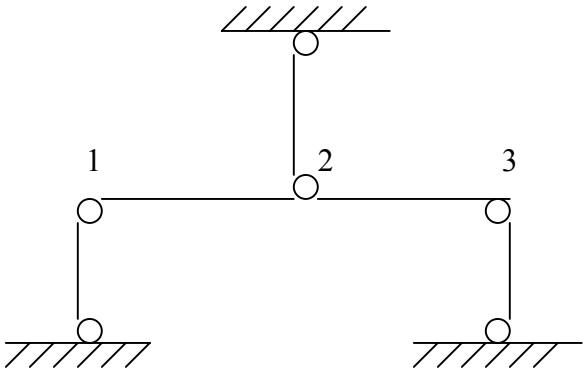


Fig 2.4 : kinematically unstable simple structures

The structural designer is not concerned with the analysis of kinematically unstable structures but rather with the detection of kinematically unstable structures whose configuration approaches a kinematically unstable configuration. Stiffness matrix of kinematically unstable structures is singular. Simple trusses whose supports are equivalent to three nonparallel and nonconcurrent links are kinematically stable.

Compound structures

Statically determinate structures may be classified as simple or compound. Compound trusses are made of two or more simple trusses connected to each other or to the supporting body by hinges or links. Compound beams, frames, and arches are built with internal release mechanisms, such as hinges, links, rollers, etc.

Chapter 3

Software developing/Modeling

3.1 Introduction

Data collection stage consists of collecting data required to program the application software for preparation of 2D drawings of base frames for gearbox. For different gearbox types the design of base frames are different. The design of base frame depends on the size of gearbox. For a particular gearbox all the dimensions of base frame are standardized. The selection of the gearbox fixes the majority of dimensions of the base frame for that gearbox.

This stage considered above-mentioned points and gathered the data that is required. As mentioned in earlier chapters there will be mainly two types of gearboxes

1. S'type (helical gearbox) [3.1]
2. K'type (bevel helical gearbox) [3.1]

Data for the two types of gearboxes are different depending on the layout of the base frame. For collecting the data all the existing drawings of base frames are studied and certain parameters that will decide the dimension of base frames are collected.

In the fallowing pages there are the typical drawings of two types one each, which will show the parameters considered for data collection.

In the first stage of programming the following data, that is collected from the available drawings s used in programming. Later on there is change in programming; the data available from the catalogues of the ELECON is used. This data is also presented later on in this chapter.

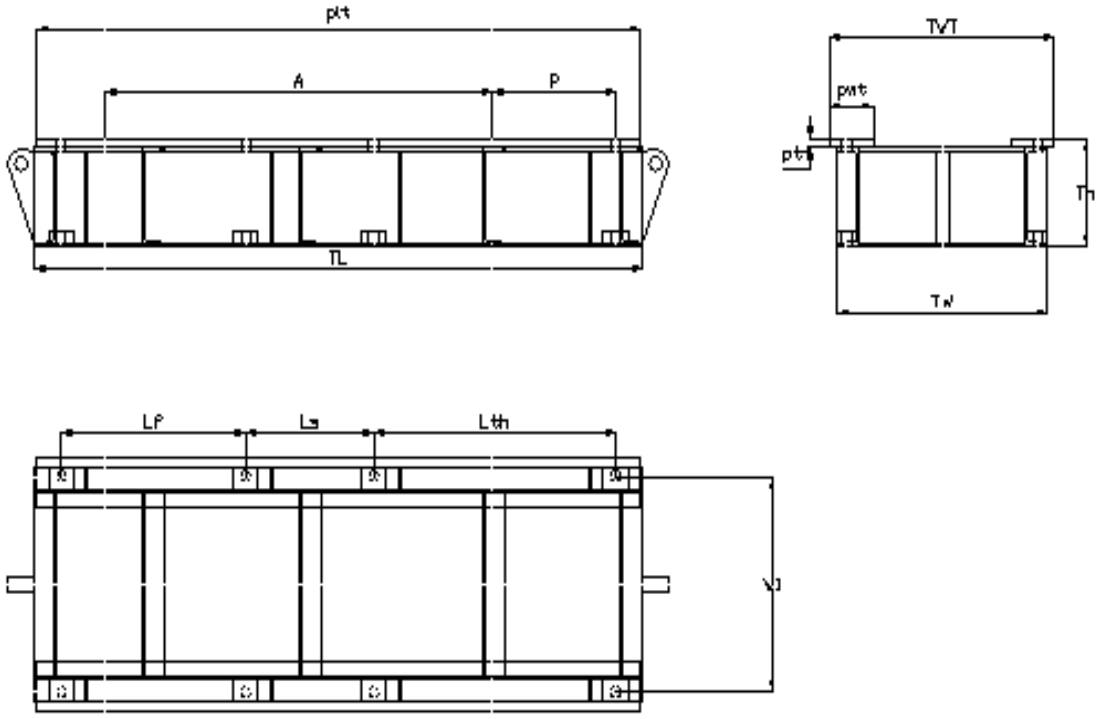


Fig 3.1: parameters used in data collection for S'type and K'type of design [3.2]

In the fallowing tables the data collected from existing designs is displaye

gear box type	section of major component	L1	TL	W1	TW	TH	n	LF
SAN 400/SO	PLATE-10	910	1030	350	610	250	3	370
SAN-100	Channel-100	220	280	125	169	185	2	220
SAN-250	Channel-200	570	660	265	335	210	2	570
SAN-315/SO	Plate-10	715	825	380	480	200	3	285
SAN-355	Channel-200	770	920	325	395	210	3	330
SAN-400	Channel-300	910	1050	350	430	310	3	370
SBN-355	Plate-10	975	110	410	480	400	3	525
SBN-500	Channel-300	1385	1555	560	660	310	4	335
SBN-630	Plate-90	1755	2450	690	1060	90	4	450
SBN-630	Plate-20	1755	2410	690	1440	150	4	800
SBN-710	Channel-400	1870	2190	770	850	435	4	900
SBN-800	Channel-400	2340	2480	870	1020	435	4	660
SCN-160	Channel-125	495	565	210	270	145	3	285
SCN-200	200	615	735	250	320	210	3	360
SCN-450	Plate-10	940	1540	510	610	250	4	575
SCN-560	Channel-400	1735	1970	640	800	400	4	570
SCN-630/SO	Channel-400	1985	2160	690	820	435	4	665
SCN-710	Channel-400	2220	2420	770	920	435	4	730
SCN-800	Channel-400	2520	2760	870	1020	435	4	840

gear box type	LS	LTh	P	ST	PTB	PLB	PWB	PTT	PLT	PWT
SAN 400/SO	540	0	*	*	10	1030	190	20	1030	135
SAN-100	0	0	100	10	50	280	*	25	55	60
SAN-250	0	0	250	10	*	*	*	10	660	100
SAN-315/SO	420	0	315	10	*	*	*	25	825	120
SAN-355	440	0	355	10	*	*	*	10	910	125
SAN-400	540	0	400	*	*	*	*	10	1030	140
SBN-355	450	0	*	*	10	110	*	10	1080	*
SBN-500	405	645	*	*	*	*	*	10	1535	160
SBN-630	505	800	*	*	*	*	*	*	*	*
SBN-630	505	450	*	20	*	*	*	20	1935	285
SBN-710	590	480	*	*	*	*	*	35	2170	210
SBN-800	580	1100	*	*	*	*	*	35	2460	220
SCN-160	210	0	*	*	*	*	*	20	495	65
SCN-200	255	0	*	*		*	*	10	685	85
SCN-450	365	460	*	*	*	*	*	10	1540	150
SCN-560	450	715	*	*	*	*	*	20	1970	200
SCN-630/SO	520	800	*	*	*	*	*	35	2160	170
SCN-710	590	900	*	*	38	80	90	35	2420	210
SCN-800	580	1100	*	*	*	*	*	35	2710	200

Table 3.1 : Data collected from old drawings of base frames

3.2 Standardization of design using CAD

3.2.1 Introduction

This chapter is about using of Auto LISP language to draw the designs of the base frames for the gearbox and motor combinations. As there are var-ious designs of base frames available using the data collected in the previous task is utilized for the programming of the base frames. The programming part will go in stages like,

1.preparation of program that will be useful in main program to draw the channel, which is the main component of any base frame design

2.preparation of main program to draw three vies of the base frames for various gearbox and motor combinations

The main aim of the project is to make the software a user-friendly one. For this purpose dialog boxes are utilized for user input. Dialog boxes are the best way of user in put types. User will have various choices to choose form.

Here in this chapter there are programs for the required purpose and how to utilize them. The code of the program is presented here, various functions used are described in the Appendix A: About Auto LISP language of this report.

Before going into the main base frame program here is the program that will draw an IS channel, which is the main part of the any base frame. The standard dimensions that are used for program are taken from Indian Standards. These standards are shown in Appendix C: Indian Standard channel.

3.2.2 Main programs of base frames

These programs can be made in to an application to draw three views of the base frames.

A complete application will be able to draw the three views of the base frame with dimensions and notes. There are three programs in this application

1. Program of channel
2. Program of dialog boxes

3. Main base frame program

3.2.2.1 Program of channel

These programs are able to take the input from user about the size of the channel, placement point of the channel in the drawing, orientation of the channel in terms of angle, length of the channel and the view of the channel i.e. front, side and top views. By using this information these programs can draw the required channel.

The code for this program is given below, it consists of one main program and two sub programs. Main program was used to store the standard specifications of the IS-MC channel, and to draw the different views of the channel according to the requirement in the main baseframe program. The code is given below

```
(defun Degrees->Radians       (numberOfDegrees)
  (* pi (/ numberOfDegrees 180.0))
) ;_ end of defun
(defun tan (a)
  (/ (sin a) (cos a))
)
(defun ch-mc (ps    ang           height     l        view  /        width height
              th_flange    th_web      r_root       r_toe  w     p
              p1    p2       p3      p4    p5    p6      p7      p8    p9
              p10   p11     p12    p13   p14   p15           p16    p17   p18
              tf    tw       rr       rt    tf1   tf2       a       x     y
              ang
             )

  (setq  width (list (cons '75 40)
                     (cons '100 50)
                     (cons '125 65)
                     (cons '150 75)
                     (cons '175 75)
                     (cons '200 75)
                     (cons '225 80)
                     (cons '250 80)
                     (cons '300 90)
                     (cons '350 100)
                     (cons '400 100)
               )
  )
  (setq  th_flange (list  (cons '75 7.3)
                          (cons '100 7.5)
```

```
                    (cons '125 8.1)
                    (cons '150 9.0)
                    (cons '175 10.2)
                    (cons '200 11.4)
                    (cons '225 12.4)
                    (cons '250 14.1)
                    (cons '300 13.6)
                    (cons '350 13.5)
                    (cons '400 15.3)
                )
)

(setq  th_web (list (cons '75 4.4)
                    (cons '100 4.7)
                    (cons '125 5.0)
                    (cons '150 5.4)
                    (cons '175 5.7)
                    (cons '200 6.1)
                    (cons '225 6.4)
                    (cons '250 7.1)
                    (cons '300 7.6)
                    (cons '350 8.1)
                    (cons '400 8.6)
            )
)
(setq  r_root (list (cons '75 8.5)
                    (cons '100 9.0)
                    (cons '125 9.5)
                    (cons '150 10.0)
                    (cons '175 10.5)
                    (cons '200 11.0)
                    (cons '225 12.0)
                    (cons '250 12.0)
                    (cons '300 13.0)
                    (cons '350 14.0)
                    (cons '400 15.0)
            )
)
(setq  r_toe (list (cons '75 4.5)
                   (cons '100 4.5)
                   (cons '125 5.0)
                   (cons '150 5.0)
                   (cons '175 5.5)
                   (cons '200 5.5)
                   (cons '225 6.0)
                   (cons '250 6.0)
                   (cons '300 6.5)
```

```
                (cons '350 7.0)
                (cons '400 7.5)
            )
)
(setq w (cdr (assoc height width)))
(setq tf (cdr (assoc height th_flange)))
(setq tw (cdr (assoc height th_web)))
(setq rr (cdr (assoc height r_root)))
(setq rt (cdr (assoc height r_toe)))

(setq a (* (/ pi 180) 6))
(setq tf1 (- tf (+ rt (* (tan a) (- (/ (- w tw) 2) rt)))))
(setq y (* (cos a) (- w (+ tw rr rt))))
(setq tf2 (+ tf (* (tan a) (/ (- w tw) 2))))
(setq x (- height (* 2 (+ rr tf2))))
(if (= view 0)
 (progn
  (command
      "line"
      (setq p1 ps)
      (setq p2 (polar p1 (degrees->radians (- 180 ang)) w))
      (setq p3 (polar p2 (degrees->radians (- 90 ang)) height))
      (setq p4 (polar p3 (degrees->radians (- 0 ang)) w))
      (setq p5 (polar p4 (degrees->radians (- 270 ang)) tf1))
      ""
  )
  (command "arc"
           "ce"
           (setq p6 (polar p5 (degrees->radians (- 180 ang)) rt))
           p5
           "a"
           -90
  )
  (command "line"
           (setq p7 (polar p6 (degrees->radians (- 270 ang)) rt))
           (setq p8 (polar p7 (degrees->radians (- 186 ang)) y))
           ""
  )
  (command "arc"
           "ce"
           (setq p9 (polar p8 (degrees->radians (- 270 ang)) rr))
           p8
           "a"
           90
  )
  (command "line"
```

```
            (setq p10 (polar p9 (degrees->radians (- 180 ang)) rr))
            (setq p11 (polar    p10
                                (degrees->radians (- 270 ang))
                                x
                      )
            )
            ""
    )
    (command "arc"
            "ce"
            (setq p12 (polar p11 (degrees->radians (- 0 ang)) rr))
            p11
            "a"
            90
    )
    (command "line"
            (setq p13 (polar p12 (degrees->radians (- 270 ang)) rr))
            (setq p14 (polar p13 (degrees->radians (- -6 ang)) y))
            ""
    )
    (command "arc"
            "ce"
            (setq p15 (polar p14 (degrees->radians (- 270 ang)) rt))
            p14
            "a"
            -90
    )
    (command "line"
            (setq p16 (polar p15 (degrees->radians (- 0 ang)) rt))
            ps

            ""
    )
  )
)

(if (= view 1)
  (progn
    (command
        "line"
        (setq p1 ps)
        (setq p2 (polar p1 (degrees->radians (- 90 ang)) l))
        (setq p3 (polar p2 (degrees->radians (- 0 ang)) height))
        (setq p4 (polar p3 (degrees->radians (- 270 ang)) l))
        (setq p5 (polar p4 (degrees->radians (- 180 ang)) height))

        (setq
```

```
          p6 (polar p5 (degrees->radians (- 0 ang)) tf)
        )
        (setq p7 (polar p6 (degrees->radians (- 90 ang)) l))
        (setq p8
              (polar p7 (degrees->radians (- 0 ang)) (- height (* 2 tf)))
        )
        (setq p9 (polar p8 (degrees->radians (- 270 ang)) l))
        ""
    )

  )
 )

 (if (= view 2)
  (progn
    (command
        "line"
        (setq p1 ps)
        (setq p2 (polar p1 (degrees->radians (- 90 ang)) w))
        (setq p3 (polar p2 (degrees->radians (- 0 ang)) l))
        (setq p4 (polar p3 (degrees->radians (- 270 ang)) w))
        (setq p5 (polar p4 (degrees->radians (- 180 ang)) l))
        (setq p6 (polar p5 (degrees->radians (- 90 ang)) (- w tw)))

        ""
    )
    (command "linetype" "load" "dashed" "acad" "" "")
    (command "setvar" "celtype" "dashed")
    (command "setvar" "ltscale" 30)

    (command "line"
             p6
             (setq p7 (polar p6 (degrees->radians (- 0 ang)) l))
             ""
    )
    (command "setvar" "celtype" "bylayer")

  )
 )


 (setq res (list (cons 1 w) (cons 2 tf) (cons 3 tw)))
 res

)
```

Using the above given codes it is possible to draw the required channel. Using this program we can draw three views of the IS-MC channel as shown below.

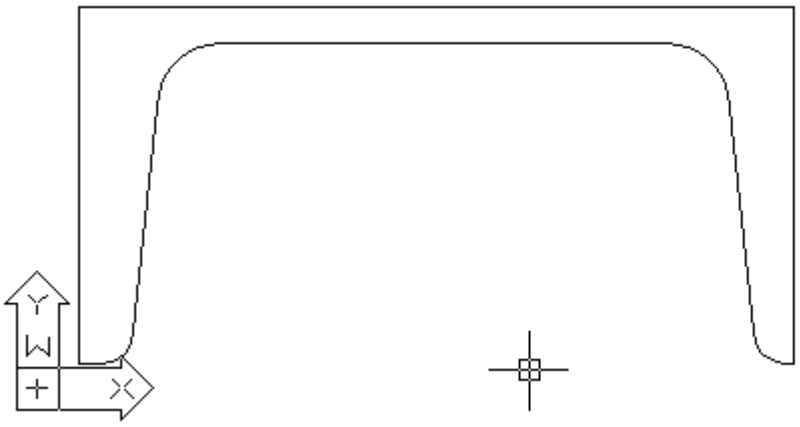


Fig 3.2: channel of size=100, angle=90, side view

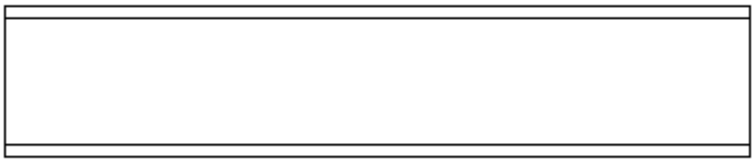


Fig 3.3: channel of size=100, angle=90, front view.

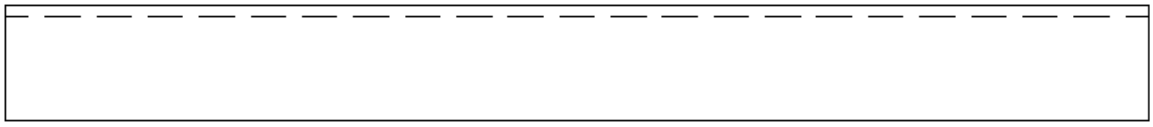


Fig 3.4: channel of size=100, angle=90, top view.

3.2.2.2 Program of dialog boxes

This program is able to take the input as type of the gearbox and output of the program is the data needed to draw the base frame for that type of gearbox. The code for this program is given below,

```
(defun getdialoginput (a          b          /          dcl_id
                       dialogloaded          dialogshow userclick
                       height     ang        result
                      )
 (setq  dialogloaded
                       T
        dialogshow
                       T
 )
 (setq newdialog nil)
 (if (= -1 (setq dcl_id (load_dialog "sam1.dcl")))
  (progn
   (princ "\nERROR: dialog box can't be loaded")
   (setq dialogloaded nil)
  )
 )
 (if (and dialogloaded
          (not (new_dialog "ch_dialog" dcl_id))
   )
  (progn
   (princ "\nERROR: can't show the dialog box")
   (setq dialogshow nil)
  )
 )
 (start_list "gbtype" 2)
 (mapcar 'add_list a)
 (end_list)
 (start_list "gbsize" 2)
 (mapcar 'add_list b)
 (end_list)

 (if (and dialogloaded dialogshow)
  (progn
   (action_tile
       "cancel"
       (strcat
         "(progn (done_dialog)"
         "(setq userclick nil)
         (setq result nil))"
```

```
        )
    )

    (action_tile
        "ok"
        (strcat
          "(progn (setq type1 (atof (get_tile \"gbtype\")))"
          "(setq size (atof (get_tile \"gbsize\")))"
          "(done_dialog) (setq userclick T))"

        )
    )
    (action_tile
        "new"
        (strcat
          "(progn(setq new (newdesign))
          (setq newdialog T))"


        )
    )


    (start_dialog)

    (unload_dialog dcl_id)
    (if newdialog
        (setq result (list (cons '1 new)
                           (cons '4 newdialog)
                     )
        )
        (if userclick
          (setq  result
                 (list

                   (cons '1 type1)
                   (cons '2 size)
                   (cons '4 newdialog)
                 )
          )
        )
    )
   )

 )
 result
)
```

```
(defun dialogout (/ gb gbtype)
  (setq gb (addlist))
  (setq b (cdr (assoc 1 gb)))
  (setq nb (cdr (assoc 2 gb)))
  (setq gbdata (cdr (assoc 3 gb)))
  (setq newdialog (cdr (assoc 4 gbdata)))
  (if (= newdialog nil)
    (progn
      (setq gbtype (cdr (assoc 1 gbdata)))
      (setq gbsize (cdr (assoc 2 gbdata)))
      (setq ct 0)
      (if (/= gb nil)
          (progn
            (if (= gbtype 0)
              (progn
                (setq gbtype1 "SAN")
                (while (<= ct nb)
                     (if (= gbsize ct)
                       (setq gbsize1 (nth ct b))
                     )
                     (setq ct (1+ ct))
                )
              )
            )
            (setq ct 0)
            (if (= gbtype 1)
              (progn
                (setq gbtype1 "SBN")
                (while (<= ct nb)
                     (if (= gbsize ct)
                       (setq gbsize1 (nth ct b))
                     )
                     (setq ct (1+ ct))
                )
              )
            )
            (setq ct 0)
            (if (= gbtype 2)
              (progn
                (setq gbtype1 "SCN")
                (while (<= ct nb)
                     (if (= gbsize ct)
                       (setq gbsize1 (nth ct b))
                     )
                     (setq ct (1+ ct))
                )
              )
```

```
            )
            (setq ct 0)
            (if (= gbtype 3)
              (progn
                (setq gbtype1 "SDN")
                (while (<= ct nb)
                     (if (= gbsize ct)
                       (setq gbsize1 (nth ct b))
                     )
                     (setq ct (1+ ct))
                )
              )
            )

          )
      )
      (setq gbtype (strcat gbtype1 "-" gbsize1))

    )
    (progn
      (setq gbtype (cdr (assoc 1 gbdata)))

    )
  )
  (setq  result (list (cons '1 newdialog)
                      (cons '2 gbtype)
                )
  )
  result
)
(defun addlist (/)
  (setq f (open "E:/Program Files/ACAD2000/geartype.txt" "r"))
  (setq  gt1 (read-line f)
         gt2 (read-line f)
         gt3 (read-line f)
         gt4 (read-line f)
  )
  (close f)
  (setq a (list gt1 gt2 gt3 gt4))
  (setq f1 (open "E:/Program Files/ACAD2000/gearsize.txt" "r"))

  (setq b (list (read-line f1)))
  (while (setq gs (read-line f1))
    (setq temb (list gs))
    (setq b (append b temb))
  )
  (close f1)
```

```
  (setq nb (length b))

  (setq data (getdialoginput a b))
  (setq  result (list (cons '1 b)
                      (cons '2 nb)
                      (cons '3 data)
                )

  )

  result
)

(defun newdesign (/)
  (setq  dialogloaded
                      T
         dialogshow
                      T
  )

  (if (= -1 (setq dcl_id (load_dialog "sam3.dcl")))
    (progn
      (princ "\nERROR: dialog box can't be loaded")
      (setq dialogloaded nil)
    )
  )
  (if (and dialogloaded
           (not (new_dialog "new_dialog" dcl_id))
      )
    (progn
      (princ "\nERROR: can't show the dialog box")
      (setq dialogshow nil)
    )
  )
  (if (and dialogloaded dialogshow)
    (progn
      (action_tile
        "cancel"
        (strcat
          "(progn (done_dialog)"
          "(setq userclick nil)
          (setq result nil))"
        )
      )

      (action_tile
        "ok"
```

```
          (strcat
            "(progn (setq size (get_tile \"size\"))"
            "(setq type1 (get_tile \"type\"))"
            "(setq channel (get_tile \"channel\"))"
            "(setq T1 (get_tile \"T\"))"
            "(setq B (get_tile \"B\"))"
            "(setq F  (get_tile \"F\"))"
            "(setq E (get_tile \"E\"))"

            "(setq R (get_tile \"R\"))"
            "(setq K (get_tile \"K\"))"
            "(setq PTT (get_tile \"PTT\"))"
            "(setq O (get_tile \"O\"))"
            "(setq h (get_tile \"h\"))"
            "(setq A (get_tile \"A\"))"
            "(setq P (get_tile \"P\"))"
            "(setq G (get_tile \"G\"))"
            "(setq N (get_tile \"N\"))"
            "(done_dialog) (setq userclick T))"
           )
      )


      (start_dialog)

      (unload_dialog dcl_id)
      (if userclick
          (setq result (list type1 size channel T1 B F E  R K PTT O h A P
                             G N)
          )
      )
     )

   )
  result
)

(defun filehandling (/)
  (setq dialog (dialogout))
  (setq newdialog (cdr (assoc 1 dialog)))
  (if newdialog
    (progn
      (setq newdata (cdr (assoc 2 dialog)))
      (setq nn (length newdata))
      (setq type1 (nth 0 newdata))
      (setq size (nth 1 newdata))
      (setq ct 0)
```

```
    (setq f (open (strcat "E:/Program Files/ACAD2000/basedata1/"
                          type1
                          "-"
                          size
                          ".txt"
                         )
                  "w"
            )
    )
    (while (< ct nn)
        (write-line (nth ct newdata) f)
        (setq ct (1+ ct))
    )
    (close f)
    (setq result (strcat type1 "-" size))
   )
   (setq result (cdr (assoc 2 dialog)))
  )
  result
)
```

The purpose of this program is to find the file that is containing the data for drawing the base frame according to the type of the gearbox. Initial dialog box of this program looks as shown below,

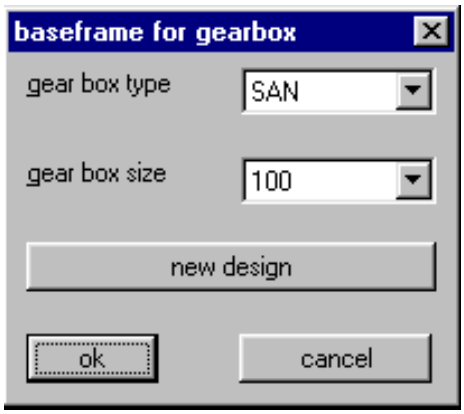


Fig3.5 : dialog box for gearbox selection

By selecting the required gearbox for which we need to design the base frame. This program will give the name of the file, which contains the data for that base frame.

For example for the selection of SAN 100 type gearbox the file name is SAN-100.

This program also has the provision to make new designs of base frames. By selecting the **new design** button, it will display another dialog box like shown below.

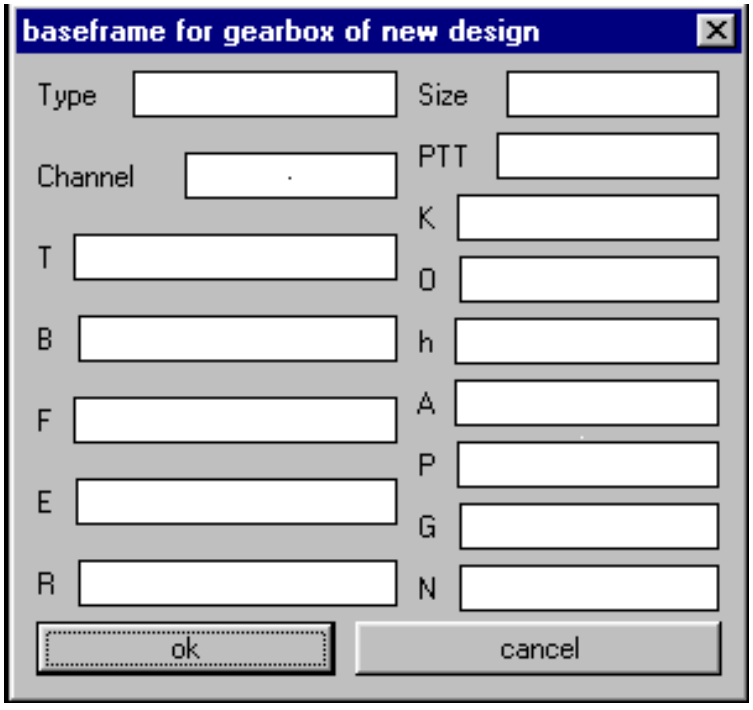


Fig 3.6 : dialog box for new design

This dialog box asks the data that is needed for the catalogues of ELECON to draw the base frame. These parameters of gearbox are described in he chapter **data collection.**

3.2.2.3 Program for main base frame design

This program is going to use above-mentioned two programs for the purpose of designing the base frame. This program is going to call the dialog box program to get the name of the file where the required data of the base frame can be found out. It calls the program to draw the channel repeatedly when ever the need arise. The code for this program is given below ,

```
;foundation detail
(defun loop (n lf ls lth sfb p1 tw1 / test n p1 step p2 e1)
  (setq test 1)
  (setq points (list p1))
  (while (<= test n)
    (if    (= test 1)
      (progn
           (setq step lf)
      )
    )
    (if    (= test 2)
      (progn
           (setq step ls)
      )
    )
    (if    (= test 3)
      (progn
           (setq step lth)
      )
    )

    (command "circle" p1 sfb)
    (setq e1 (entlast))
    (command "setvar" "CECOLOR" "red")
    (command "setvar" "celtype" "center")
    (setq p3 (polar p1 (Degrees->Radians 90) 50))
    (setq p4 (polar p3 (Degrees->Radians 270) 100))
    (setq p5 (polar p1 (Degrees->Radians 0) 150))
    (setq p6 (polar p5 (Degrees->Radians 180) 300))
    (command "line" p3 p4 "")
    (setq e2 (entlast))
    (command "line" p5 p6 "")
    (setq e3 (entlast))
    (setq p2 (polar p1 (Degrees->Radians 270) tw1))
```

```
    (command "copy" e1 e2 e3 "" p1 p2)
    (command "setvar" "cecolor" 11)
    (command "setvar" "celtype" "bylayer")
    (setq p1 (polar p1 (Degrees->Radians 0) step))
    (setq temp (list p1))
    (setq points (append points temp))
    (setq test (1+ test))
   )
   (setq  result (list (cons '1 points)
                       (cons '2 p2)
                 )
   )
   result
)
;foundation detail
(defun loop1 (n lf ls lth sfb p3 tf ptt h / test n p1 step p2 e1)
   (setq test 1)
   (while (<= test n)
     (if    (= test 1)
       (progn
            (setq step lf)
       )
     )
     (if    (= test 2)
       (progn
            (setq step ls)
       )
     )
     (if    (= test 3)
       (progn
            (setq step lth)
       )
     )
     (command "setvar" "celtype" "dashed")
     (setq p4 (polar p3 (Degrees->Radians 180) (/ sfb 2)))
     (command "line"
              p4
              (setq p5 (polar p4 (Degrees->Radians 270) (+ ptt tf)))
              ""
     )
     (setq e1 (entlast))
     (setq p6 (polar p5 (Degrees->Radians 270) (- h tf)))
     (command "line"
              p6
              (setq p7 (polar p6 (Degrees->Radians 90) (* sfb 1.25)))
              ""
     )
```

```
    (setq e2 (entlast))
    (setq p8 (polar p3 (Degrees->Radians 0) (/ sfb 2)))
    (command "copy" e1 e2 "" p4 p8)
    (command "setvar" "CECOLOR" "red")
    (command "setvar" "celtype" "center")
    (setq p9 (polar p3 (Degrees->Radians 90) 50))
    (setq p10 (polar p9 (Degrees->Radians 270) 100))
    (command "line"
             p9
             p10
             ""
    )
    (setq e3 (entlast))
    (setq p11 (polar p9 (Degrees->Radians 270) h))
    (command "copy" e3 "" p9 p11)
    (command "setvar" "cecolor" 11)
    (command "setvar" "celtype" "bylayer")
    (setq p3 (polar p3 (Degrees->Radians 0) step))
    (setq test (1+ test))
  )
)


;channels
(defun loop2 (n tl scfact tw w h pf ls lf lth tw1 / p1 temp l test)
  (setq l (- tw (* w 2)))
  (if (= n 2)
    (setq temp n)
    (if    (= n 3)
      (progn
          (setq p1 (list (- (- 0 (* scfact 20))
                            (/ tl 2)
                            (/ w 2)
                         )
                         (- (- 0 (* scfact 20)) w)
                         0
                   )
          )

          (ch-mc p1 90 h l 2)
          (setq p2 (list
                     (- (- 0 (* scfact 20))
                       (/ tl 2)
                       (/ w 2)
                     )
                     h
                     0
```

```
            )
        )
        (command "setvar" "CECOLOR" "magenta")
        (setq pdim1
            (list (- (- 0 (* scfact 20)) w) (- 0 (* scfact 20)) 0)
        )
        (setq pdim2 (polar p1 (Degrees->Radians 0) w))
        (command "dimlinear"
                pdim1
                pdim2
                "h"
                (strcat "@0," (rtos (- 0 (* scfact 4))))
        )
        (command "setvar" "celtype" "dashed")
        (command "setvar" "CECOLOR" 11)
        (ch-mc p2 180 h l 0)
        (command "setvar" "CECOLOR" "green")
        (command "setvar" "celtype" "bylayer")

)
(if (= n 4)
    (progn

     (setq  ptemp (polar pf
                    (Degrees->Radians 270)
                    (-   w
                         (/ (- tw tw1) 2)
                    )
              )
     )
     (setq  pf1 (polar ptemp
                    (Degrees->Radians 0)
                    (/ (+
                          lf
                          w
                       )
                       2
                    )
              )
     )
     (setq  pf2 (polar pf1
                    (Degrees->Radians 270)
                    l
              )
     )
     (ch-mc pf2 270 h l 2)
     (setq  ps1 (polar pf2
```

```
                (Degrees->Radians 0)
                (/ (+ lf ls) 2)
            )
)
(ch-mc ps1 270 h l 2)
(setq pth1 (polar ps1 (Degrees->Radians 0) (/ (+ ls lth) 2)))
(ch-mc pth1 270 h l 2)
(command "setvar" "CECOLOR" "magenta")
(setq  pdim1
        (list (- (- 0 (* scfact 20)) w) (- 0 (* scfact 20)) 0)
)
(setq pdim2 (polar pth1 (Degrees->Radians 180) w))
(command "dimlinear"
          pdim1
          pdim2
          "h"
          (strcat "@0," (rtos (* scfact 4)))
)
(command "dimlinear"
          pdim2
          (setq pdim3 (polar pdim2
                             (Degrees->Radians 180)
                             (/ (+ ls lth) 2)
                      )
          )
          "h"
          (strcat "@0," (rtos (* scfact 4)))
)
(command "dimlinear"
          pdim3
          (setq pdim4
                (polar pdim3 (Degrees->Radians 180) (/ (+ lf ls) 2))
          )
          "h"
          (strcat "@0," (rtos (* scfact 4)))
)
(command "setvar" "celtype" "dashed")
(command "setvar" "CECOLOR" 11)
(setq  pth2 (polar pth1
                   (Degrees->Radians 90)
                   (+ l w (* scfact 20))
            )
)
(ch-mc pth2 0 h l 0)
(setq
  ps2  (polar pth2 (Degrees->Radians 180) (/ (+ ls lth) 2))
)
```

```
            (ch-mc ps2 0 h l 0)
            (setq pf3 (polar ps2 (Degrees->Radians 180) (/ (+ lf ls) 2)))
            (ch-mc pf3 0 h l 0)
            (command "setvar" "CECOLOR" "green")
            (command "setvar" "celtype" "bylayer")

          )
      )
    )
  )
)
;stiffners
(defun loop3 (n l1 scfact tw w h pf ls lf lth stfthk sfb tweb tw1 tf /)
  (setq x (/ (- tw tw1) 2))
  (setq p1 (polar pf (Degrees->Radians 0) (* sfb 2.5)))
  (setq p2 (polar p1 (Degrees->Radians 90) x))
  (command "setvar" "celtype" "dashed")
  (command "setvar" "CECOLOR" 11)
  (command "line"
            p2
            (setq p3 (polar p2 (Degrees->Radians 270) (- w tweb))
            )
            ""
  )
  (setq e1 (entlast))
  (setq p4 (polar p3 (Degrees->Radians 0) stfthk))
  (command "line"
            p4
            (setq p5 (polar p4
                            (Degrees->Radians 90)
                            (- w tweb)
                     )
            )
            ""
  )
  (setq e2 (entlast))

  (if (= n 2)
    (progn
      (command
          "copy"
          e1
          e2
          ""
          "M"
          p2
          (setq p6 (polar p2 (Degrees->Radians 270) (- (+ tw tweb) w)))
```

```
      (setq p7 (polar p6
                      (Degrees->Radians 0)
                      (- l1 (* (* sfb 2.5) 2) stfthk)
               )
      )
      (setq p8 (polar p7 (Degrees->Radians 90) (- (+ tw tweb) w)))
      ""
  )
)
(if   (= n 3)
  (progn
      (command
       "copy"
       e1
       e2
       ""
       "M"
       p2
       (setq p6 (polar p2 (Degrees->Radians 270) (- (+ tw tweb) w)))
       (setq p7 (polar p6 (Degrees->Radians 0) lf))
       (setq p8 (polar p7 (Degrees->Radians 90) (- (+ tw tweb) w)))
       (setq  p9 (polar p8
                        (Degrees->Radians 0)
                        (- ls (* (* sfb 2.5) 2) stfthk)
                 )
       )
       (setq p10 (polar p9 (Degrees->Radians 270) (- (+ tw tweb) w)))
       ""
      )
  )

  (if (= n 4)
      (progn
       (command
        "copy"
        e1
        e2
        ""
        "M"
        p2
        (setq
          p6 (polar p2 (Degrees->Radians 270) (- (+ tw tweb) w))
        )
        (setq p7 (polar p6 (Degrees->Radians 0) lf))
        (setq p8 (polar p7 (Degrees->Radians 90) (- (+ tw tweb) w)))
        (setq p9 (polar p8 (Degrees->Radians 0) ls))
        (setq
```

```
            p10 (polar p9 (Degrees->Radians 270) (- (+ tw tweb) w))
           )
           (setq p11 (polar p10
                            (Degrees->Radians 0)
                            (- lth (* (* sfb 2.5) 2) stfthk)
                     )
           )
           (setq
            p12 (polar p11 (Degrees->Radians 90) (- (+ tw tweb) w))
           )

           ""
          )
         )
    )
  )
)
(command "setvar" "CECOLOR" "green")
(command "setvar" "celtype" "bylayer")
(setq pf1 (polar p2 (Degrees->Radians 90) (+ (* scfact 20) tf)))
(command "line"
         pf1
         (setq pf2 (polar pf1 (Degrees->Radians 90) (- h (* tf 2))))
         ""
)
(setq e3 (entlast))
(setq pf3 (polar pf2 (Degrees->Radians 0) stfthk))
(command "line"
         pf3
         (setq pf4 (polar pf3 (Degrees->Radians 270) (- h (* tf 2))))
         ""
)
(setq e4 (entlast))
(command "setvar" "CECOLOR" "magenta")
(command "dimlinear"
         pf1
         pf3
         "h"
         (strcat "@0," (rtos (- 0 (* scfact 2)))))
(command "setvar" "CECOLOR" "green")
(if (= n 2)
  (progn
    (command "copy"
             e3
             e4
             ""
             pf1
```

```
            (setq pf5
                  (polar pf1 (Degrees->Radians 0) (- lf (* (* sfb 2.5) 2)))
            )
  )
  (command "setvar" "CECOLOR" "magenta")
  (command "dimlinear"
            (setq pdim1 (list (- (- 0 (* scfact 20)) tl) 0 0)
                  )
            pf1
            "h"
            (strcat "@0," (rtos (- 0 (* scfact 2))))
            )
  (command "dimlinear"
            pf4
            pf5
            "h"
            (strcat "@0," (rtos (- 0 (* scfact 2))))
            )
  (command "dimlinear"
            (setq pdim2 (polar pf5 (Degrees->Radians 0) stfthk))
            (setq pdim3 (list (- 0 (* scfact 20)) 0 0))
            "h"
            (strcat "@0," (rtos (+ (- 0 (* scfact 2)) tf)))
            )
)
(if    (= n 3)
  (progn
      (command "copy"
                  e3
                  e4
                  ""
                  "M"
                  pf1
                  (setq pf5 (polar pf1
                                    (Degrees->Radians 0)
                                    lf
                              )
                  )
                  (setq pf6 (polar pf5
                                    (Degrees->Radians 0)
                                    (- ls (* (* sfb 2.5) 2) stfthk)
                              )
                  )
                  ""
      )
      (command "setvar" "CECOLOR" "magenta")
  (command "dimlinear"
```

```
        (setq pdim1 (list (- (- 0 (* scfact 20)) tl) 0 0)
              )
        pf1
        "h"
        (strcat "@0," (rtos (- 0 (* scfact 2))))
        )
(command "dimlinear"
        pf4
        pf5
        "h"
        (strcat "@0," (rtos (- 0 (* scfact 2))))
        )
    (command "dimlinear"
        (setq pdim2 (polar pf5 (Degrees->Radians 0) stfthk))
        pf6
        "h"
        (strcat "@0," (rtos (- 0 (* scfact 2))))
        )

(command "dimlinear"
        (setq pdim2 (polar pf6 (Degrees->Radians 0) stfthk))
        (setq pdim3 (list (- 0 (* scfact 20)) 0 0))
        "h"
        (strcat "@0," (rtos (+ (- 0 (* scfact 2)) tf)))
        )

)
(if (= n 4)
    (progn
      (command "copy"
               e3
               e4
               ""
               "M"
               pf1
               (setq pf5 (polar pf1
                                (Degrees->Radians 0)
                                lf
                         )
               )
               (setq pf6 (polar pf5
                                (Degrees->Radians 0)
                                ls
                         )
               )
               (setq pf7 (polar pf6
                                (Degrees->Radians 0)
```

```
                                    (- lth (* (* sfb 2.5) 2) stfthk)
                                )
                        )
                        ""
                )
                (command "setvar" "CECOLOR" "magenta")
        (command "dimlinear"
                (setq pdim1 (list (- (- 0 (* scfact 20)) tl) 0 0)
                        )
                pf1
                "h"
                (strcat "@0," (rtos (- 0 (* scfact 2))))
                )
        (command "dimlinear"
                pf4
                pf5
                "h"
                (strcat "@0," (rtos (- 0 (* scfact 2))))
                )
            (command "dimlinear"
                (setq pdim2 (polar pf5 (Degrees->Radians 0) stfthk))
                pf6
                "h"
                (strcat "@0," (rtos (- 0 (* scfact 2))))
                )
            (command "dimlinear"
                (setq pdim2 (polar pf6 (Degrees->Radians 0) stfthk))
                pf7
                "h"
                (strcat "@0," (rtos (- 0 (* scfact 2))))
                )

        (command "dimlinear"
                (setq pdim2 (polar pf7 (Degrees->Radians 0) stfthk))
                (setq pdim3 (list (- 0 (* scfact 20)) 0 0))
                "h"
                (strcat "@0," (rtos (+ (- 0 (* scfact 2)) tf)))
                )

            )
        )
    )
  )
)
;side view of bottom plate
(defun loop4 (sfb tw w tweb / pb1 pb2 pb3 e1 pbt)
  (setq pbt (* sfb 1.25))
```

```
  (setq pb1 (list 0 pbt 0))
  (setq pb2 (list (- w tweb) pbt 0))
  (command "line" (list 0 0 0) pb1 "")
  (setq e1 (entlast))
  (command "line" pb1 pb2 "")
  (setq e2 (entlast))
  (setq pb3 (list tw 0 0))
  (command "copy" e1 "" (list 0 0 0) pb3)
  (command "copy" e2 "" pb2 (list tw pbt 0))
)
; bottom plate
(defun loop5 (pf lf ls lth sfb w tweb tw tw1 n /)
  (setq x (/ (- tw tw1) 2))
  (setq pbt (* sfb 1.25))
  (setq p1 (polar pf (Degrees->Radians 90) x))
  (setq p2 (polar p1 (Degrees->Radians 180) (* sfb 1.25)))
  (setq p3 (polar p2 (Degrees->Radians 270) (- w tweb)))
  (setq p4 (polar p3 (Degrees->Radians 0) (* sfb 2.5)))
  (command "setvar" "celtype" "dashed")
  (command "setvar" "CECOLOR" 11)
  (command "rectangle" p2 p4)
  (setq e1 (entlast))
  (if (= n 2)
    (progn
      (command
          "copy"
          e1
          ""
          "M"
          p2
          (setq p5 (polar p2 (Degrees->Radians 270) (- (+ tw tweb) w)))

          (setq p6 (polar p5 (Degrees->Radians 0) lf))
          (setq p7 (polar p6 (Degrees->Radians 90) (- (+ tw tweb) w)))
          ""
      )
    )
    (if    (= n 3)
      (progn
          (command
            "copy"
            e1
            ""
            "M"
            p2
            (setq p5 (polar p2 (Degrees->Radians 270) (- (+ tw tweb) w)))
            (setq p6 (polar p5 (Degrees->Radians 0) lf))
```

```
        (setq p7 (polar p6 (Degrees->Radians 90) (- (+ tw tweb) w)))
        (setq p8 (polar p7 (Degrees->Radians 0) ls))
        (setq p9 (polar p8 (Degrees->Radians 270) (- (+ tw tweb) w)))
        ""
      )
    )
    (if (= n 4)
      (progn
        (command
          "copy"
          e1
          ""
          "M"
          p2
          (setq
            p5 (polar p2 (Degrees->Radians 270) (- (+ tw tweb) w))
          )
          (setq p6 (polar p5 (Degrees->Radians 0) lf))
          (setq p7 (polar p6 (Degrees->Radians 90) (- (+ tw tweb) w)))
          (setq p8 (polar p7 (Degrees->Radians 0) ls))
          (setq
            p9 (polar p8 (Degrees->Radians 270) (- (+ tw tweb) w))
          )
          (setq p10 (polar p9 (Degrees->Radians 0) lth))
          (setq
            p11 (polar p10 (Degrees->Radians 90) (- (+ tw tweb) w))
          )
          ""
        )
      )
    )
  )
)
(command "setvar" "CECOLOR" "green")
(command "setvar" "celtype" "bylayer")
(setq pf1 (polar p2 (Degrees->Radians 90) (* scfact 20)))
(setq pf2 (polar pf1 (Degrees->Radians 0) (* sfb 2.5)))
(setq pf3 (polar pf2 (Degrees->Radians 90) pbt))
(command "rectangle" pf1 pf3)
(setq e1 (entlast))
(command "setvar" "CECOLOR" "magenta")
(command "dimlinear"
         pf3
         pf1
         "h"
         (strcat "@0," (rtos (* scfact 3))))
(command "dimlinear"
```

```
            pf3
            pf1
            "v"
            (strcat "@" (rtos (- 0 (* scfact 1))) ",0"))
   (command "setvar" "CECOLOR" "green")
  (if (= n 2)
   (progn
    (command "copy"
              e1
              ""
              pf1
              (setq pf4 (polar pf1 (Degrees->Radians 0) lf))
    )
   )
   (if    (= n 3)
    (progn
        (command "copy"
                  e1
                  ""
                  "M"
                  pf1
                  (setq pf4 (polar pf1 (Degrees->Radians 0) lf))
                  (setq pf5 (polar pf4 (Degrees->Radians 0) ls))
                  ""
        )
    )
    (if (= n 4)
        (progn
          (command "copy"
                    e1
                    ""
                    "M"
                    pf1
                    (setq pf4 (polar pf1 (Degrees->Radians 0) lf))
                    (setq pf5 (polar pf4 (Degrees->Radians 0) ls))
                    (setq pf6 (polar pf5 (Degrees->Radians 0) lth))
                    ""
          )
        )
    )
   )
  )
)


(defun liftdetail (scfact dialift w h /)
  (setq p1 (list (* scfact 15) (- 0 (* scfact 20)) 0))
```

```
(command "circle" p1 "d" dialift)
(setq p2 (polar p1 (Degrees->Radians 180) dialift))
(command "arc" "ce" p1 p2 "a" -90)
(setq p3 (polar p1 (Degrees->Radians 90) dialift))
(command "line"
        p3
        (setq p4 (polar p3
                        (Degrees->Radians 354)
                        (/ (+ w dialift) (cos (Degrees->Radians 6)))
                )
        )
        ""
)
(setq e1 (entlast))
(setq  p5 (polar p3
                (Degrees->Radians 354)
                (/ dialift (cos (Degrees->Radians 6)))
        )
)
(setq p6 (polar p5 (Degrees->Radians 270) (- h (* tf 2))))
(command "line"
        p6
        (setq p7 (polar p6
                        (Degrees->Radians 6)
                        (/ w (cos (Degrees->Radians 6)))
                )
        )
        ""
)
(setq e2 (entlast))
(command "line" p2 p6 "")
(command "line" p7 p4 "")
(setq e3 (entlast))
(command "chamfer" "d" 10 10)
(command "chamfer" e1 e3)
(command "chamfer" e2 e3)
(command "setvar" "CECOLOR" "magenta")
(command "dimasz" (/ scfact 1.2))
(command "dimtxt" (/ scfact 2))

(setq p8 (polar p2 (Degrees->Radians 0) (/ dialift 2)))
(command "qleader"
        p8
        (setq p9 (polar p8 (Degrees->Radians 135) (* scfact 5)))
        ""
        ""
        (strcat "%%c" (rtos dialift 2 0))
```

```
        ""
)
(setq p10 (polar p1 (Degrees->Radians 120) dialift))
(command "qleader"
        p10
        (setq p11 (polar p10 (Degrees->Radians 135) (* scfact 5)))
        ""
        ""
        (strcat "%%c"
                (rtos (* dialift 2)
                        2
                        0
                )
        )
        ""
)
(command "dimlinear"
        p2
        p7
        "h"
        (strcat "@0," (rtos (* scfact -6)))
)
(command "dimlinear"
        p1
        p2
        "h"
        (strcat "@" (rtos (* scfact -4)) "," (rtos (* scfact -10)))
)
(command "dimlinear"
        p6
        p7
        "h"
        (strcat "@0," (rtos (* scfact -4)))
)
(command "dimlinear"
        p6
        p3
        "v"
        (strcat "@" (rtos (* scfact 3)) ",0")
)
(command "line"
        p3
        (setq p10 (polar p3 (Degrees->Radians 0) scfact))
        ""
)
(setq e5 (entlast))
(setq  ptemp (polar p4
```

```
                    (Degrees->Radians 174)
                    (/    (/ (+ w dialift) (cos (Degrees->Radians 6)))
                          2
                    )
               )
  )
  (command "dimangular"
           ""
           p3
           p10
           ptemp
           (strcat "@" (rtos (* scfact 4)) ",0")
  )
  (setq ptemp1 (polar p7 (Degrees->Radians 90) 10))
  (command "dimlinear"
           (setq p11 (polar p7 (Degrees->Radians 180) 10))
           ptemp1
           "h"
           "t"
           "10x45%%d"
           (strcat "@0," (rtos (- 0 (* scfact 2))))
  )
  (command "dimlinear"
           p1
           p3
           "v"
           (strcat "@" (rtos scfact) ",0")
  )
  (command "dimasz" scfact)
  (command "dimtxt" scfact)

)
;top plates
(defun topplate (pf lf ls lth sfb w tweb tw tw1 n x ptw tw2 scact ptt /)
  (command "setvar" "CECOLOR" "cyan")
  (setq p1 (polar pf (Degrees->Radians 180) (* sfb 3)))
  (setq p2 (polar p1 (Degrees->Radians 90) (+ x (/ (- tw2 tw) 2))))
  (setq p3 (polar p2 (Degrees->Radians 0) (* sfb 6)))
  (setq p4 (polar p3 (Degrees->Radians 270) ptw))
  (command "rectangle"
           p2
           p4)
  (setq e1 (entlast))
  (if (= n 2)
    (progn
      (command "copy"
               e1
```

```
        ""
        "M"
        p2
        (setq p5 (polar p2 (Degrees->Radians 270) (- tw2 ptw)))
        (setq p6 (polar p5 (Degrees->Radians 0) lf))
        (setq p7 (polar p6 (Degrees->Radians 90) (- tw2 ptw)))
        "")
(setq p8 (list (- (- 0 (* scfact 20)) (/ (+ tl l1) 2) ) h 0))
(setq p9 (polar p8 (Degrees->Radians 180) (* sfb 3)))
(setq p10 (polar p9 (Degrees->Radians 90) ptt))
(setq p11 (polar p10 (Degrees->Radians 0) (* sfb 6)))
(setq p12 (polar p11 (Degrees->Radians 270) ptt))
(command "rectangle"
        p10
        p12)
(setq e2 (entlast))
(command "copy"
        e2
        ""
        p8
        (setq p13 (polar p8 (Degrees->Radians 0) lf))
        )
(command "setvar" "CECOLOR" "magenta")
(command "dimlinear"
        (setq p15 (list (- (- 0 (* scfact 20)) tl) h 0))
        p10
        "h"
        (strcat "@0," (rtos (* scfact 1))))
(command "dimlinear"
        p10
        p12
        "h"
        (strcat "@0," (rtos (* scfact 2)))
        )
(command "dimlinear"
        p12
        (setq p14 (polar p13 (Degrees->Radians 180) (* sfb 3)))
        "h"
        (strcat "@0," (rtos (* scfact 2))))
(command "dimlinear"
        p14
        (setq p16 (polar p13 (Degrees->Radians 0) (* sfb 3))
              )
        "h"
        (strcat "@0," (rtos (* scfact 2))))

(command "dimlinear"
```

```
            p16
            (setq p17 (list (- 0 (* scfact 20)) h 0))
            "h"
            (strcat "@0," (rtos (* scfact 2)))
            )
  )
(if (= n 3)
   (progn
   (command "copy"
            e1
            ""
            "M"
            p2
            (setq p5 (polar p2 (Degrees->Radians 270) (- tw2 ptw)))
            (setq p6 (polar p5 (Degrees->Radians 0) lf))
            (setq p7 (polar p6 (Degrees->Radians 90) (- tw2 ptw)))
            (setq p8 (polar p7 (Degrees->Radians 0) ls))
            (setq p9 (polar p8 (Degrees->Radians 270) (- tw2 ptw)))
            "")
   (setq p10 (list (- (- 0 (* scfact 20)) (/ (+ tl l1) 2) ) h 0))
   (setq p11 (polar p10 (Degrees->Radians 180) (* sfb 3)))
   (setq p12 (polar p11 (Degrees->Radians 90) ptt))
   (setq p13 (polar p12 (Degrees->Radians 0) (* sfb 6)))
   (setq p14 (polar p13 (Degrees->Radians 270) ptt))
   (command "rectangle"
            p12
            p14)
   (setq e2 (entlast))
   (command "copy"
            e2
            ""
            "M"
            p10
            (setq p15 (polar p10 (Degrees->Radians 0) lf))
            (setq p16 (polar p15 (Degrees->Radians 0) ls))
            ""
            )
   (command "setvar" "CECOLOR" "magenta")
   (command "dimlinear"
            p12
            (setq p17 (list (- (- 0 (* scfact 20)) tl) h 0))
            "h"
            (strcat "@" (rtos (- 0 (* scfact 2)))"," (rtos (* scfact 2) )))
   (command "dimlinear"
            p12
            p14
            "h"
```

```
        (strcat "@0," (rtos (* scfact 2)))
        )
(command "dimlinear"
        p14
        (setq p18 (polar p15 (Degrees->Radians 180) (* sfb 3)))
        "h"
        (strcat "@0," (rtos (* scfact 2))))
(command "dimlinear"
        p18
        (setq p19 (polar p15 (Degrees->Radians 0) (* sfb 3))
              )
        "h"
        (strcat "@0," (rtos (* scfact 2))))

(command "dimlinear"
        p19
        (setq p20 (polar p16 (Degrees->Radians 180) (* sfb 3)))
        "h"
        (strcat "@0," (rtos (* scfact 2)))
        )
(command "dimlinear"
        p20
        (setq p21 (polar p16 (Degrees->Radians 0) (* sfb 3)))
        "h"
        (strcat "@0," (rtos (* scfact 2)))
        )
(command "dimlinear"
        p21
        (setq p22 (list (- 0 (* scfact 20)) h 0))
        "h"
        (strcat "@0," (rtos (* scfact 2))))
)

(if (= n 4)
    (progn
(command "copy"
        e1
        ""
        "M"
        p2
        (setq p5 (polar p2 (Degrees->Radians 270) (- tw2 ptw)))
        (setq p6 (polar p5 (Degrees->Radians 0) lf))
        (setq p7 (polar p6 (Degrees->Radians 90) (- tw2 ptw)))
        (setq p8 (polar p7 (Degrees->Radians 0) ls))
        (setq p9 (polar p8 (Degrees->Radians 270) (- tw2 ptw)))
        (setq p10 (polar p9 (Degrees->Radians 0) lth))
        (setq p11 (polar p10 (Degrees->Radians 90) (- tw2 ptw)))
```

```
        "")
(setq p12 (list (- (- 0 (* scfact 20)) (/ (+ tl l1) 2) ) h 0))
(setq p13 (polar p12 (Degrees->Radians 180) (* sfb 3)))
(setq p14 (polar p13 (Degrees->Radians 90) ptt))
(setq p15 (polar p14 (Degrees->Radians 0) (* sfb 6)))
(setq p16 (polar p15 (Degrees->Radians 270) ptt))
(command "rectangle"
        p14
        p16)
(setq e2 (entlast))
(command "copy"
        e2
        ""
        "M"
        p12
        (setq p17 (polar p12 (Degrees->Radians 0) lf))
        (setq p18 (polar p17 (Degrees->Radians 0) ls))
        (setq p19 (polar p18 (Degrees->Radians 0) lth))

        ""
        )
(command "setvar" "CECOLOR" "magenta")
(command "dimlinear"
        p14
        (setq p20 (list (- (- 0 (* scfact 20)) tl) h 0))
        "h"
        (strcat "@" (rtos (- 0 (* scfact 2)))"," (rtos (* scfact 2) )))
(command "dimlinear"
        p14
        p16
        "h"
        (strcat "@0," (rtos (* scfact 2)))
        )
(command "dimlinear"
        p16
        (setq p21 (polar p17 (Degrees->Radians 180) (* sfb 3)))
        "h"
        (strcat "@0," (rtos (* scfact 2))))
(command "dimlinear"
        p21
        (setq p22 (polar p17 (Degrees->Radians 0) (* sfb 3))
              )
        "h"
        (strcat "@0," (rtos (* scfact 2))))

(command "dimlinear"
        p22
```

```
            (setq p23 (polar p18 (Degrees->Radians 180) (* sfb 3)))
            "h"
            (strcat "@0," (rtos (* scfact 2)))
            )
    (command "dimlinear"
            p23
            (setq p24 (polar p18 (Degrees->Radians 0) (* sfb 3)))
            "h"
            (strcat "@0," (rtos (* scfact 2)))
            )
    (command "dimlinear"
            p24
            (setq p25 (polar p19 (Degrees->Radians 180) (* sfb 3)))
            "h"
            (strcat "@0," (rtos (* scfact 2)))
            )
    (command "dimlinear"
            p25
            (setq p26 (polar p19 (Degrees->Radians 0) (* sfb 3)))
            "h"
            (strcat "@0," (rtos (* scfact 2)))
            )
    (command "dimlinear"
            p26
            (setq p27 (list (- 0 (* scfact 20)) h 0))
            "h"
            (strcat "@0," (rtos (* scfact 2))))
    )
        )
    )
   )
  (command "setvar" "CECOLOR" "green"))
(defun c:baseframe (/  tw  h   p     p1  p2  p3  p4  p5   p6
                       p7   p8  p9  p10  p11 p12 x   y   tw1  tw2
                       ptt   ptw gbdata   pp
                      )
  (setq gbtype (filehandling))
  (setq  f
         (open (strcat "E:/Program Files/ACAD2000/basedata1/"
                       gbtype
                       ".txt"
            )
            "r"
         )
  )
  (setq ct 0)
  (setq gbdata (list ""))
```

```
(while (< ct 17)
  (setq gbd (list (read-line f)))
  (setq gbdata (append gbdata gbd)
  )
  (setq ct (1+ ct))
)
(close f)

(princ (nth 1 gbdata))
(setq type1 (nth 1 gbdata))
(setq size (atof (nth 2 gbdata)))
(setq h (atof (nth 3 gbdata)))
(setq T1 (atof (nth 4 gbdata)))
(setq B (atof (nth 5 gbdata)))
(setq F (atof (nth 6 gbdata)))
(setq E (atof (nth 7 gbdata)))
(setq R (atof (nth 8 gbdata)))
(setq K (atof (nth 9 gbdata)))
(setq PTT (atof (nth 10 gbdata)))
(setq O (atof (nth 11 gbdata)))
(setq h1 (atof (nth 12 gbdata)))
(setq A (atof (nth 13 gbdata)))
(setq P (atof (nth 14 gbdata)))
(setq G (atof (nth 15 gbdata)))
(setq N (atof (nth 16 gbdata)))
(if (<= size 250)
  (progn
    (setq div 10)
    (setq stfthk 6)
    (setq wf1 30)
    (setq wf2 20)
    (setq tempfact 4.7)
  )
  (if    (<= size 630)
    (progn
         (setq div 15)
         (setq stfthk 8)
         (setq wf1 70)
         (setq wf2 40)
         (setq tempfact 4.5)
    )
    (progn
         (setq div 18)
         (setq stfthk 10)
         (setq wf1 75)
         (setq wf2 40)
         (setq tempfact 4.2)
```

```
    )
   )
  )
  (setq scfact (/ size div))
  (setq tw (+ F (* wf2 2)))
  (setq tw1 F)
  (if (= type1 "SAN")
   (progn
    (setq l1 K)
    (if (= R 0)
        (progn
          (setq lf K)
          (setq ls 0)
          (setq lth 0)
          (setq n 2)
        )
        (progn
          (setq lf (- K R))
          (setq ls R)
          (setq lth 0)
          (setq n 3)
        )
    )
   )
   (progn
    (setq l1 T1)
    (if (= K 0)
        (progn
          (setq  lf (- T1 R)
          )
          (setq ls R)
          (setq lth 0)
          (setq n 3)
        )
        (progn
          (setq lf (- T1 K))
          (setq ls (- K R))
          (setq lth R)
          (setq n 4)
        )

    )
   )
  )
(setq tl (+ l1 (* O 6) 20))
  (setq ptt PTT)
  (setq plt (- tl 20))
```

```
(setq ptw (+ G 20))
(setq sfb O)
(setq tw2 (+ F (* wf1 2)))
(princ n)



(setq x (/ (- tw tw1) 2))
(setq y (/ (- tw2 tw1) 2))
(setq pc (list 0 h 0))
(setq p1 (list tw 0 0))
(setq lift (/ (* sfb 4) 3))
(if (<= lift 20)
  (progn (setq dialift 20) (setq pthlift 20))
  (if    (<= lift 25)
    (progn (setq dialift 25) (setq pthlift 25))
    (if (<= lift 30)
        (progn (setq dialift 30) (setq pthlift 32))
        (if (<= lift 35)
          (progn (setq dialift 35) (setq pthlift 36))
          (if (<= lift 40)
            (progn (setq dialift 40) (setq pthlift 40))
            (if    (<= lift 45)
              (progn (setq dialift 45) (setq pthlift 50))
              (if (<= lift 50)
                  (progn (setq dialift 50) (setq pthlift 50))
                  (if (<= lift 55)
                    (progn (setq dialift 55) (setq pthlift 56))
                    (if (<= lift 60)
                      (progn (setq dialift 60) (setq pthlift 63))
                    )
                  )
              )
            )
          )
        )
    )
  )
)

(command "setvar" "CECOLOR" "green")
(ch-mc pc 180 h 100 0)
(setq res (ch-mc p1 0 h 100 0))
(setq w (cdr (assoc 1 res)))
(setq tf (cdr (assoc 2 res)))
(setq tweb (cdr (assoc 3 res)))
(setq l (- tw (* w 2)))
```

```
(setq p2 (list (- x y) (+ h ptt) 0))
(setq p3 (list (+ x (- ptw y)) h 0))
(setq p4 (list (+ x y tw1) h 0))
(setq p5 (list (- (+ x y tw1) ptw) (+ h ptt) 0))
(setq z (/ (- tw2 tw) 2))
(setq m (/ (- tl plt) 2))
(setq p12 (list (- (- 0 (* scfact 20)) m) h 0))
(setq p13 (list (- (- (- 0 (* scfact 20)) m) plt) (+ h ptt) 0))
(setq
  p14  (list (- (- 0 (* scfact 20)) m) (+ (- 0 (* scfact 20)) z) 0)
)
(setq  p15 (list (- (- (- 0 (* scfact 20)) m) plt)
                 (- (+ (- 0 (* scfact 20)) z) ptw)
                 0
           )
)
(command "linetype" "load" "center" "acad" "" "")
(command "setvar" "CECOLOR" "cyan")
(command "rectangle" p2 p3)
(command "rectangle" p4 p5)
(command "setvar" "CECOLOR" "magenta")
(command "dimtvp" 1.0)
(command "dimtad" 0)
(command "dimdec" 0)
(command "dimasz" (/ scfact 1.2))
(command "dimtxt" (/ scfact 2))
(setq pv (list tw (+ h ptt) 0))
(command "dimlinear"
         p13
         (polar p13 (Degrees->Radians 0) plt)
         "h"
         (strcat "@0," (rtos (* 8 scfact)))
)
(command "dimlinear"
         p13
         (polar p13 (Degrees->Radians 90) h1)
         "v"
         (strcat "@-" (rtos (* 5 scfact)) ",0")
)

(command "dimlinear"
         (list 0 0)
         (list tw 0)
         "h"
         (strcat "@0,-" (rtos (* 6 scfact)))
)
```

```
(command "dimlinear"
        (setq ptemp (list (- 0 (* scfact 20)) 0))
        (polar ptemp (Degrees->Radians 180) tl)
        "h"
        (strcat "@0,-" (rtos (* 8 scfact)))
)
(command "dimlinear"
        (list x 0)
        (list (+ x tw1) 0)
        "h"
        (strcat "@0,-" (rtos (* 4 scfact)))
)
(command "dimlinear"
        p1
        pv
        "v"
        (strcat "@" (rtos (* 4 scfact)) ",0")
)


(command "dimlinear"
        p2
        (polar p2 (Degrees->Radians 270) ptt)
        "v"
        (strcat "@-" (rtos (* 2 scfact)) ",0")
)
(command "dimlinear"
        p2
        p3
        "h"
        (strcat "@0," (rtos (* 6 scfact)))
)
(command "dimlinear"
        p2
        p4
        "h"
        (strcat "@0," (rtos (* 10 scfact)))
)

(command "setvar" "CECOLOR" "red")
(command "setvar" "celtype" "center")
(command "ltscale" "10" "")
(command "line" (list x (- 0 (* scfact 2))) (list x (* scfact 2)) "")
(command "line" (list x (- h (* scfact 3))) (setq pdim1 (list x (+ h (* scfact 3)))) "")
(command "line" (list (+ x tw1) (- 0 (* scfact 2))) (list (+ x tw1) (* scfact 2)) "")
(command "line"
        (list (+ x tw1) (- h (* scfact 3)))
```

```
        (setq pdim2 (list (+ x tw1) (+ h (* scfact 3))))
        ""
)
(command "setvar" "CECOLOR" "magenta")
(command "setvar" "celtype" "bylayer")
(command "dimlinear"
        pdim1
        pdim2
        "h"
        "@0,0")
(command "setvar" "dimdec" 1)
(command "dimlinear"
        pdim2
        (setq pdim3 (polar pdim2 (Degrees->Radians 180) (/ tw1 2)))
        "h"
        (strcat "@0," (rtos (* scfact 2))))
(command "setvar" "dimdec" 0)
(command "linetype" "load" "dashed" "acad" "" "")
(command "setvar" "celtype" "dashed")
(command "setvar" "CECOLOR" 11)
(setq p6 (list (- 0 (* scfact 20)) 0 0))
(ch-mc p6 0 h l 0)
(setq p7 (list (- (- 0 (* scfact 20)) tl) h 0))
(ch-mc p7 180 h l 0)
(command "setvar" "CECOLOR" "green")
(command "setvar" "celtype" "bylayer")
(setq pp (list w h 0))
(ch-mc pp 90 h l 1)
(ch-mc p7 90 h tl 1)
(setq p8 (list (- 0 (* scfact 20)) (- 0 (* scfact 20)) 0))
(ch-mc p8 180 h tl 2)
(setq  p9 (list (- (- 0 (* scfact 20)) tl)
                (- (- 0 (* scfact 20)) tw)
                0
        )
)
(ch-mc p9 0 h tl 2)
(setq  p10 (list (- 0 (* scfact 20))
                 (- (- 0 (* scfact 20)) (- tw w))
                 0
         )
)
(ch-mc p10 270 h l 2)
(setq  p11 (list (- (- 0 (* scfact 20)) tl)
                 (- (- 0 (* scfact 20)) w)
                 0
         )
```

```
)
(ch-mc p11 90 h l 2)
(command "setvar" "CECOLOR" 11)
(command "line"
        (list (- x (/ sfb 2)) (* sfb 1.25) 0)
        (list (- x (/ sfb 2)) 0 0)
        ""
)
(setq e1 (entlast))
(command "line"
        (list (+ x (/ sfb 2)) (* sfb 1.25) 0)
        (list (+ x (/ sfb 2)) 0 0)
        ""
)
(setq e2 (entlast))
(setq pc (list tw1 0 0))
(command "copy" e1 e2 "" '(0 0 0) pc)
(command "line"
        (setq p15 (list (- x (/ sfb 2)) (- h tf) 0))
        (polar p15 (Degrees->Radians 90) (+ tf ptt))
        ""
)
(setq e1 (entlast))
(command "line"
        (setq p16 (list (+ x (/ sfb 2)) (- h tf) 0))
        (polar p16 (Degrees->Radians 90) (+ tf ptt))
        ""
)
(setq e2 (entlast))
(command "copy" e1 e2 "" '(0 0 0) pc)
(setq  p1 (list (- (- 0 (* scfact 20)) (/ (+ tl l1) 2))
                (- (- 0 (* scfact 20)) (/ (- tw tw1) 2))
                0
        )
)
(setq result (loop n lf ls lth (/ sfb 2) p1 tw1))
(setq p1 (list 0 0 0))
(setq p2 (polar p1 (Degrees->Radians 90) (+ h ptt)))
(setq  p3 (polar p2
                 (Degrees->Radians 0)
                 (- (- 0 (* scfact 20)) (/ (+ tl l1) 2))
        )
)
(loop1 n lf ls lth sfb p3 tf ptt h)
(setq points (cdr (assoc 1 result)))
(setq pr (cdr (assoc 2 result)))
(setq test 0)
```

```
(while (< test n)
  (if    (= test 0)
    (setq pf (nth test points))
  )
  (if    (= test 1)
    (setq ps (nth test points))

  )
  (if    (= test 2)
    (setq pth (nth test points))

  )
  (if    (= test 3)
    (setq pft (nth test points))

  )
  (setq test (1+ test))
)
(if (= n 2)
  (setq ptemp ps)
)
(if (= n 3)
  (setq ptemp pth)
)
(if (= n 4)
  (setq ptemp pft)
)
(command "setvar" "CECOLOR" "red")
(command "setvar" "celtype" "center")
(setq ph1 (list 0 (+ h ptt h1) 0))
(setq ph2 (list (- (- 0 (* scfact 20)) tl) (+ h ptt h1) 0))
(command "line" ph1 ph2 "")
(setq pth1 (list (/ (- (- 0 (* scfact 20)) tl) 3) (+ h ptt h1) 0))
(command "text" "j" "bc" pth1         scfact "" "C.L. of Gear Box" "")
(setq ptemp1 (polar ptemp (Degrees->Radians 180) P))
(setq  pocl1 (polar ptemp1
                    (Degrees->Radians 90)
                    (+   h
                         ptt
                         h1
                         (* scfact 30)
                    )
             )
)
(setq  pocl2 (polar pocl1
                    (Degrees->Radians 270)
                    (+ h ptt h1 (* scfact 30) tw2 50)
```

```
            )
)
(setq picl1 (polar pocl2 (Degrees->Radians 180) A))
(setq
  picl2 (polar picl1
                (Degrees->Radians 90)
                (+ h ptt h1 (* scfact 30) tw2 50)
          )
)
(setq  pistp (polar picl2 (Degrees->Radians 270) (* scfact 10))
)
(setq postp (polar pocl1 (Degrees->Radians 270) (* scfact 10)))
(command "line" pocl1 pocl2 "")
(command "line" picl1 picl2 "")
(command "text" "j" "bc" pistp scfact 90 "C.L. of Input" "")
(command "text" "j" "bc" postp scfact         90 "C.L. of Output" "")
(command "setvar" "celtype" "bylayer")
(setq  pgbdim1          (polar ptemp
                      (Degrees->Radians 90)
                      (+ h ptt (* scfact 20) 50)
                  )
)
(setq pgbdim2 (polar pgbdim1 (Degrees->Radians 180) P))
(setq pgbdim3 (polar pgbdim2 (Degrees->Radians 180) A))
(command "setvar" "CECOLOR" "magenta")
(command "dimtvp" 1.0)
(command "dimtad" 0)
(command "dimdec" 0)
(command "dimasz" (/ scfact 1.2))
(command "dimtxt" (/ scfact 2))
(command "dimaligned"
          pgbdim1
          pgbdim2
          (strcat "@0," (rtos (* 3 scfact)))
)

(command "dimaligned"
          pgbdim2
          pgbdim3
          (strcat "@0," (rtos (* 3 scfact)))
)

(setq test 1)
(while (< test n)
  (if    (= test 1)
    (command "dimaligned"
              pf
```

```
            ps
            (strcat "@0," (rtos (* 4 scfact)))
   )
  )
  (if    (= test 2)
   (command "dimaligned"
            ps
            pth
            (strcat "@0," (rtos (* 4 scfact)))
   )
  )
  (if    (= test 3)
   (command "dimaligned"
            pth
            pft
            (strcat "@0," (rtos (* 4 scfact)))
   )
  )
  (setq test (1+ test))
)

(command "setvar" "CECOLOR" "bylayer")
(command "insert"
         "E:/Program Files/ACAD2000/Template/GA2"
         (list (- 0 (+ (* scfact 20) tl (* scfact 20)))
               (- 0 (+ (* scfact 20) tw (* scfact 15)))
         )
         (/ scfact tempfact)
         (/ scfact tempfact)
         ""
         "Base frame for gear box"
         gbtype
)
(command "setvar" "CECOLOR" "green")
(loop2 n tl scfact tw w h pf ls lf lth tw1)
(loop3 n l1 scfact tw w h pf ls lf lth stfthk    sfb tweb tw1 tf)
(setq pstfsv1 (list 0 h 0))
(setq pstfsv2 (list tw h 0))
(setq pstfsv3 (list tw 0 0))
(command "line" (list 0 0 0) pstfsv1 "")
(command "line" pstfsv2 pstfsv3 "")
(loop4 sfb tw w tweb)
(loop5 pf lf ls lth sfb w tweb tw tw1 n)
(topplate pf lf ls lth sfb w tweb tw tw1 n x ptw tw2 scfact ptt)
(command "zoom" "extents")
(command "ucsicon" "off")
(setq plift1 (list (- 0 (* scfact 20)) (- h tf) 0))
```

```
(command "line"
        plift1
        (setq plift2   (polar plift1
                            (Degrees->Radians 6)
                            (/ dialift (cos (Degrees->Radians 6)))
                       )
        )
        ""
)
(command "arc"
        "ce"
        (setq plift3 (polar plift2 (Degrees->Radians 270) dialift))
        (setq plift4 (polar plift3 (Degrees->Radians 90) dialift))
        "a"
        -90
)
(setq plift5 (polar plift3 (Degrees->Radians 0) dialift))
(command "line"
        plift5
        (setq plift6 (list (- 0 (* scfact 20)) tf 0))
        ""
)
(command "circle" plift3 "d" dialift)
(command "setvar" "CECOLOR" "red")
(command "setvar" "celtype" "center")
(command "line"
        (setq pliftcl1 (polar plift3 (Degrees->Radians 0) (+ dialift (* scfact 2))))
        (setq pliftcl2 (polar plift3 (Degrees->Radians 180) (+ dialift (* scfact 2))))
        "")
(setq ecl1 (entlast))
(command "line"
        (setq pliftcl3 (polar plift3 (Degrees->Radians 90) (+ dialift (* scfact 2))))
        (setq pliftcl4 (polar plift3 (Degrees->Radians 270) (+ dialift (* scfact 2))))
        "")
(setq ecl2 (entlast))
(command "setvar" "CECOLOR" "green")
(command "setvar" "celtype" "bylayer")
(setq plift7 (list (- (- 0 (* scfact 20)) tl) (- h tf) 0))
(command "line"
        plift7
        (setq plift8   (polar plift7
                            (Degrees->Radians 174)
                            (/ dialift (cos (Degrees->Radians 6)))
                       )
        )
        ""
)
```

```
  (command "arc"
           "ce"
           (setq plift9 (polar plift8 (Degrees->Radians 270) dialift))
           (setq plift10 (polar plift9 (Degrees->Radians 90) dialift))
           "a"
           90
  )
  (setq plift11 (polar plift9 (Degrees->Radians 180) dialift))
  (command "line"
           plift11
           (setq plift12 (list (- (- 0 (* scfact 20)) tl) tf 0))
           ""
  )
  (command "circle" plift9 "d" dialift)
  (command "copy"
           ecl1
           ecl2
           ""
           plift3
           plift9
           )
  (setq pliftcl5 (polar plift3 (Degrees->Radians 0) (+ (/ tw 2)(- (* scfact 20) dialift))))
  (command "copy"
           ecl1
           ""
           plift3
           pliftcl5)
(command "setvar" "CECOLOR" "magenta")
  (command "dimlinear"
           (setq pdim4 (polar pliftcl5 (Degrees->Radians 180) (/ pthlift 2)))
           (setq pdim5 (polar pdim4 (Degrees->Radians 0) pthlift))
           "h"
           (strcat "@0," (rtos (- 0 (* scfact 2)))))
  (command "setvar" "CECOLOR" "green")
  (setq plifts1 (list (/ tw 2) tf 0))
  (setq plifts2 (polar plifts1 (Degrees->Radians 180) (/ pthlift 2)))
  (command "line"
           plifts2
           (setq plifts3 (polar plifts2 (Degrees->Radians 90) (- h (* tf 2))))
           ""
  )
  (setq plifts4 (polar plifts3 (Degrees->Radians 0) pthlift))
  (command "line"
           plifts4
           (setq plifts5 (polar   plifts4
                                  (Degrees->Radians 270)
                                  (- h (* tf 2))
```

```
                        )
        )
        ""
)

(setq pliftt1 (list (- 0 (* scfact 20)) (- 0 (* scfact 20)) 0))
(setq pliftt2 (polar pliftt1 (Degrees->Radians 270) (/ tw 2)))
(setq pliftt3 (polar pliftt2 (Degrees->Radians 90) (/ pthlift 2)))
(command "line"
        pliftt3
        (setq pliftt4 (polar pliftt3 (Degrees->Radians 0) (* dialift 2)))
        (setq pliftt5 (polar pliftt4 (Degrees->Radians 270) pthlift))
        (setq pliftt6 (polar pliftt5 (Degrees->Radians 180) (* dialift 2)))
        ""
)
(setq pliftt7 (list (- (- 0 (* scfact 20)) tl) (- 0 (* scfact 20)) 0))
(setq pliftt8 (polar pliftt7 (Degrees->Radians 270) (/ tw 2)))
(setq pliftt9 (polar pliftt8 (Degrees->Radians 90) (/ pthlift 2)))
(command
 "line"
 pliftt9
 (setq pliftt10 (polar pliftt9 (Degrees->Radians 180) (* dialift 2)))
 (setq pliftt11 (polar pliftt10 (Degrees->Radians 270) pthlift))
 (setq pliftt12 (polar pliftt11 (Degrees->Radians 0) (* dialift 2)))
 ""
)
(setq pliftcl6 (polar pliftt8 (Degrees->Radians 180) dialift))
(setq pliftcl7 (polar pliftt2 (Degrees->Radians 0) dialift))
(command "copy"
        ecl2
        ""
        plift3
        pliftcl6)
(command "copy"
        ecl2
        ""
        plift3
        pliftcl7)

(command "setvar" "CECOLOR" "red")
(command "setvar" "celtype" "center")
(setq pliftt13 (polar pliftt8 (Degrees->Radians 180) (* scfact 6)))
(setq pliftt14 (polar pliftt2 (Degrees->Radians 0) (* scfact 6)))
(command "line"
        pliftt13
        pliftt14
        ""
```

```
)
(setq plifts6 (polar plifts1 (Degrees->Radians 270) (* scfact 6)))
(setq
  plifts7 (polar plifts1 (Degrees->Radians 90) (+ (* scfact 6) h))
)
(command "line"
          plifts6
          plifts7
          ""
)
(command "setvar" "CECOLOR" "green")
(command "setvar" "celtype" "bylayer")
(liftdetail scfact dialift w h)
(command "setvar" "CECOLOR" "yellow")
(setq pleader1 (list (- 10 (* scfact 20)) (/ h 2) 0))
(command
  "qleader"
  pleader1
  (setq peader2 (polar pleader1 (Degrees->Radians 315) (* scfact 10)))
  ""
  ""
  "4"
  ""
)
(command "text"
          (setq ptext1  (list (* scfact 17)
                              (- 0 (+ (* scfact 20) (* scfact 15)))
                              0
                        )
          )
          ""
          0
          "Item No 4"
)
(setq plead1 (list (- (- 0 (* scfact 20)) (/ (+ l1 tl) 2)) (- (- 0 (* scfact 20)) (/ (+ tw tw1) 2)) 0))
(setq plead2 (polar plead1 (Degrees->Radians 225) (/ sfb 2)))
(command "qleader"
          plead2
          (setq plead3 (polar plead2 (Degrees->Radians 225) (* scfact 10)))
          (setq plead4 (polar plead3 (Degrees->Radians 0) (* scfact 5)))
          ""
          (strcat (rtos (* n 2)) "-%%c" (rtos sfb) "Holes")
          "")
(setq plead5 (list 5 5 0))
(command "qleader"
          plead5
          (setq plead6 (polar plead5 (Degrees->Radians 225) (* scfact 12)))
```

```
(setq plead7 (polar plead6 (Degrees->Radians 0) (* scfact 5)))
""
(strcat "Channel")
(strcat "IS MC" (rtos h))
"")
```

)

For the selected gearbox type this program will draw the drawing of the base frame automatically. For example for the selection of gearbox SCN-800 the drawing of the base frame will be as shown below,

3.3 Macro for modeling of base frame

For modeling base frame APDL language is used. APDL is Ansys parametric design language used for preparing macros to speed up the process of modeling as well as solution of mechanical components in Ansys. Macros are basically Ansys commands run through a Mac file.

Base frames are mainly constructed out of channels so a macro was developed for making the model of IS-MC channel. This is used in main macro to construct the base frame using different length of cannels.

3.3.1 Macro for channel

This macro takes size of the channel, length of the channel as the input. The code for this macro is given below.

```
/prep7
parres,change,arg1,,
numstr,kp,arg3
k,,0,0,0
k,,b,0,0
k,,b,h,0
k,,0,h,0
k,,r2,h-r2,0
*afun,deg
x=((b-tw)/2-r1)*tan(6)
y1=h-(tf+x)
k,,b-(tw+r1),y1,0
y2=y1-r1
k,,b-tw,y2,0
y3=h-y2
k,,b-tw,y3,0
y4=y3-r1
x4=b-(tw+r1)
k,,x4,y4,0
k,,r2,r2,0
ar=arg3-1
numstr,line,arg3
l,ar+1,ar+2
l,ar+2,ar+3
l,ar+3,ar+4
larc,ar+4,ar+5,ar+3,r2,
```

```
l,ar+5,ar+6
larc,ar+6,ar+7,ar+1,r1
l,ar+7,ar+8
larc,ar+8,ar+9,ar+5,r1
l,ar+9,ar+10
larc,ar+10,ar+1,ar+2,r2
k,,0,0,arg2
l,ar+1,ar+11
numstr,area,arg3
a,ar+1,ar+2,ar+3,ar+4,ar+5,ar+6,ar+7,ar+8,ar+9,ar+10
vdrag,ar+1,,,,,,ar+11,,,,,,
```

This macro uses an input file, which consists of parameters of particular channel. For example for a channel of size 400, Input file will be

```
/NOPR
*SET,B       ,  100.0000000000
*SET,H       ,  400.0000000000
*SET,R1      ,  15.00000000000
*SET,R2      ,  7.500000000000
*SET,TF      ,  15.30000000000
*SET,TW      ,  8.100000000000
*SET,_BUTTON ,  0.000000000000
*SET,_RETURN ,  0.000000000000
*SET,_STATUS ,  1.000000000000
*SET,_UIQR   ,  155.0000000000
/GO
```

3.3.2 Macro for base frame

As mentioned earlier this macro repeatedly calls channel macro to develop base frame for particular gearbox. The code for this macro is,

```
/prep7
parres,change,arg2,,
channel,arg1,tl,50
local,11,0,twtb,h,0,180,0,0,,
wpcsys,,11
csys,11
channel,arg1,tl,150
local,12,0,b,h,0,0,180,90,,
wpcsys,,12
csys,12
channel,arg1,twtb-(2*b),250
local,13,0,twtb-b,h,tl,180,0,90,,
```

```
wpcsys,,13
csys,13
channel,arg1,twtb-(2*b),350
csys,0
wpcsys,,
vplot
xb1=(twtt-twtb)/2
zb1=(tl-plt)/2
zb2=
block,0-xb1,pwt-xb1,h,ptt+h,zb1,(sfb*6)+zb1
*do,n,1,3,1
*if,n,eq,1,then
lg=lf
*elseif,n,eq,2,
lg=ls
*else
lg=lth
*endif
vgen,2,4+n,,,,,lg,100,1,0
*enddo
vsel,,,,5,8,,0
cm,vol2,volu
vgen,2,vol2,,,,twtt-pwt,,,500,1,0
zb2=(tl-l1)/2
wb1=sfb*2.5
block,0,b-tw,0,(sfb*1.25),zb2-(wb1/2),zb2+(wb1/2)
*do,n,1,3,1
*if,n,eq,1,then
lg=lf
*elseif,n,eq,2,
lg=ls
*else
lg=lth
*endif
vgen,2,12+n,,,,,lg,1000,1,0
*enddo
vsel,,,,13,16,,0
cm,vol3,volu
vgen,2,vol3,,,,twtb+tw-b,,,1500,1,0
allsel,all
vplot
local,14,0,0,0,0,0,0,90,0,,
wpcsys,,14
csys,14
xc1=(twtb-w1)/2
yc1=zb2
cyl4,xc1,yc1,(sfb/2),,,,(0-h-ptt)
```

```
csys,0
wpcsys,,
vplot
*do,n,1,3,1
*if,n,eq,1,then
lg=lf
*elseif,n,eq,2,
lg=ls
*else
lg=lth
*endif
vgen,2,20+n,,,,,lg,100,1,0
*enddo
vsel,,,,21,24,,0
cm,vol4,volu
vgen,2,vol4,,,w1,,,1000,1,0
allsel,all
vplot
vsel,s,,,1,20,,0
cm,vol5,volu
vsel,s,,,21,28,,0
cm,vol6,volu
allsel,all
vplot
numstr,volu,100
vsbv,vol5,vol6,,,,
zbstf1=yc1+2.5*sfb
block,0,b,0,h,zbstf1,zbstf1+10
vgen,2,118,,,,,lf,,,,,
vgen,2,119,,,,,ls,,,,,
vgen,2,120,,,,,lth-(5*sfb),,,,,
vsel,s,,,118,121,,,
cm,vol7,volu
allsel,all
vplot
vgen,2,vol7,,,twtb-b,,,,,,,
wpcsys,,13
wpx1=zb2+(lth/2)-(b/2)
wpoffs,wpx1,,
csys,4
channel,arg1,twtb-(2*b),3000
wpx2=(lth+ls)/2
wpoffs,wpx2,,
channel,arg1,twtb-(2*b),4000
wpx3=(lf+ls)/2
wpoffs,wpx3,,
channel,arg1,twtb-(2*b),5000
```

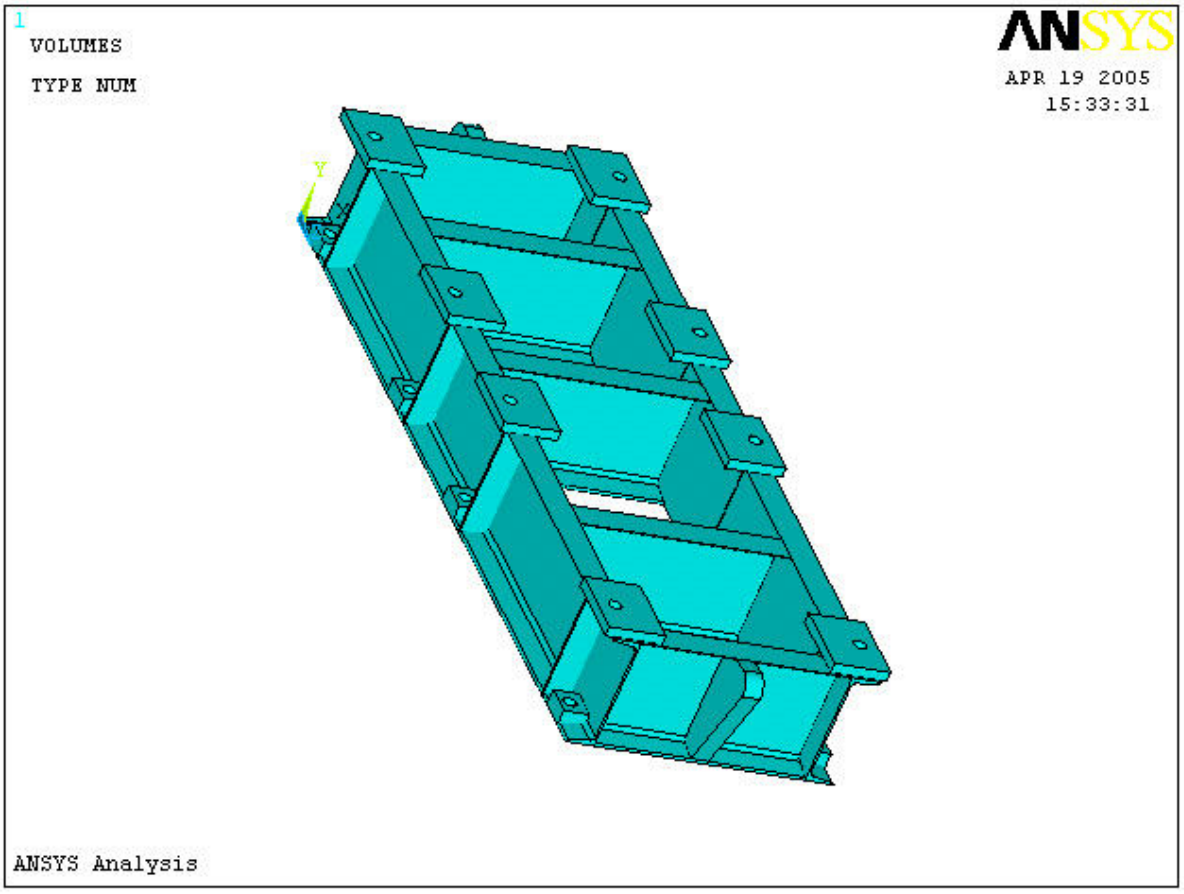
```
csys,0
wpcsys,,
local,15,0,(twtb/2)-(pthk/2),0,b,0,0,90,,
wpcsys,,15
csys,15
numstr,kp,10000
numstr,line,10000
numstr,area,10000
numstr,volu,150
k,,0,0,0
k,,b,0,0
k,,(b-r2),r2,0
ytemp1=(((b-(r2+tw))*tan(6))+r2)
k,,tw,ytemp1,0
ytemp2=(h-ytemp1)
k,,tw,ytemp2,0
ytemp3=ytemp2+((b+lifdia)-tw)*tan(6)
k,,(b+lifdia),ytemp3,0
k,,(b+(lifdia*2)),(ytemp3-lifdia),0
k,,(b+lifdia),(ytemp3-lifdia),0
l,10002,10003
l,10003,10004
l,10004,10005
l,10006,10002
larc,10005,10006,10007,lifdia
k,,0,0,pthk
l,10000,10008
a,10002,10003,10004,10005,10006
vdrag,10000,,,,,,10005,,,,,,
local,16,0,(twtb/2)+(pthk/2),0,(tl-b),0,0,-90,,
wpcsys,,16
csys,16
numstr,kp,20000
numstr,line,20000
numstr,area,20000
numstr,volu,250
k,,0,0,0
k,,b,0,0
k,,(b-r2),r2,0
ytemp1=(((b-(r2+tw))*tan(6))+r2)
k,,tw,ytemp1,0
ytemp2=(h-ytemp1)
k,,tw,ytemp2,0
ytemp3=ytemp2+((b+lifdia)-tw)*tan(6)
k,,(b+lifdia),ytemp3,0
k,,(b+(lifdia*2)),(ytemp3-lifdia),0
k,,(b+lifdia),(ytemp3-lifdia),0
```

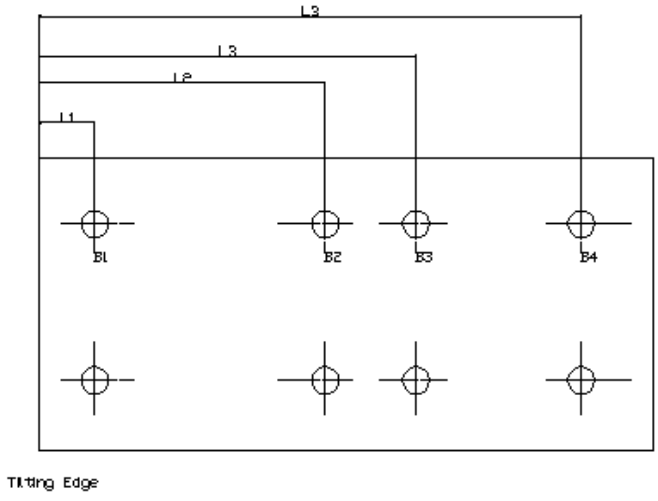
```
l,20002,20003
l,20003,20004
l,20004,20005
l,20006,20002
larc,20005,20006,20007,lifdia
k,,0,0,pthk
l,20000,20008
a,20002,20003,20004,20005,20006
vdrag,20000,,,,,,20005,,,,,,
```

Parameters needed by this macro can be supplied using an input file as,

```
/NOPR
*SET,L1      ,  2520.0000000000
*SET,LF      ,  840.0000000000
SET,LS      ,  580.0000000000
*SET,LTH     ,  1100.000000000
*SET,PLT     ,  2790.000000000
*SET,PTT     ,  35.00000000000
*SET,PWT     ,  200.0000000000
*SET,SFB     ,  45.00000000000
*SET,TL      ,  2810.000000000
*SET,TWTB     ,  950.0000000000
*SET,TWTT     ,  1000.0000000000
*SET,W1      ,  870.0000000000
*SET,pthk      ,  63.0000000000
*SET,lifdia      ,  60.0000000000
*SET,_BUTTON ,  0.000000000000
*SET,_RETURN ,  0.000000000000
*SET,_STATUS ,  1.000000000000
*SET,_UIQR   ,  0.000000000000
/GO
```

For this input i.e. for SCN-800 gear box the model for base frame looks as shown below.


1
VOLUMES
TYPE NUM
ANSYS
APR 19 2005
15:33:31
Y
ANSYS Analysis


L3
L3
L2
L1
B1
B2
B3
B4
Tilting Edge

Chapter 4

Results and discussion

4.1 Force analysis

The object of force analysis is to find the forces that are coming on base frame from gearbox due to out put torque. In this project we had chosen a gearbox with fallowing specifications

Type of gearbox: SCN

Size of gearbox: 800

Speed ratio i_N: 50

Input speed n_1: 1000 rpm

For these specifications we first find out the power rating P_N form catalogues

Power rating P_N: 1480 KW

From this we calculate output torque using the equation

Output torque: P_N/N

Where $N = (2*Pi*n_1)/(I_N*60)$

$= 2.094$ rad/sec

Form this output torque is 706.36 N-mm

From this we fallow to calculate the load at each bolt in the foundation of the gearbox.

In this gearbox there are four foundation bolts at each side, we call them B1, B2, B3, B4.

L1=length of B1 form tilting edge= 90

L2=length of B2 from tilting edge= 930

L3=length of B3 from tilting edge= 1510

L4=length of B4 from tilting edge= 2610

Let w=load in a bolt per unit length, then

Total momentum on all the bolts is

2(L1*L1+L2*L2+L3*L3+L4*L4)*w

Which is equal to the out put torque

Form this we can find w = 35.4415 N/mm

After finding we calculate load at each bolt hat is going to act on base frame as

Load at B1 = w*L1 = 3189.7 N

Load at B2 = w*L2 = 32960.6 N

Load at B3 = w*L3 = 53516.68 N

Load at B4 = w*L4 = 92502.34 N

4.2 Boundary conditions

All the bottom holes of the base frame are fixed

All the forces on top holes of base frame are applied vertically upwards

When all the boundary conditions are applied the base frame model looks as shown below

4.3 Material properties

Material properties of structural steel used in this analysis are

Young's Modules = 2.0e11 Pa

Poison's ratio = 0.3

4.4 Meshing

Meshing of the model used in this analysis is adaptive meshing that is generated automatically b the Ansys program.

4.5 Results from stress analysis

This stress analysis shown results given below

Equivalent (von-mises) stress = 0.979 Pa (min)

$1.732X10^{5}$ Pa (max)

Maximum shear stress = 0.5548 Pa (min)

$9.243X10^{4}$ Pa (max)

Total deformation = 0 m (min)

$0.114X10^{-4}$ m (max)

4.6 Results from model analysis

The analysis was done for six modes.

The frequency for each mode is given below.

1st Frequency 7.7595 Hz

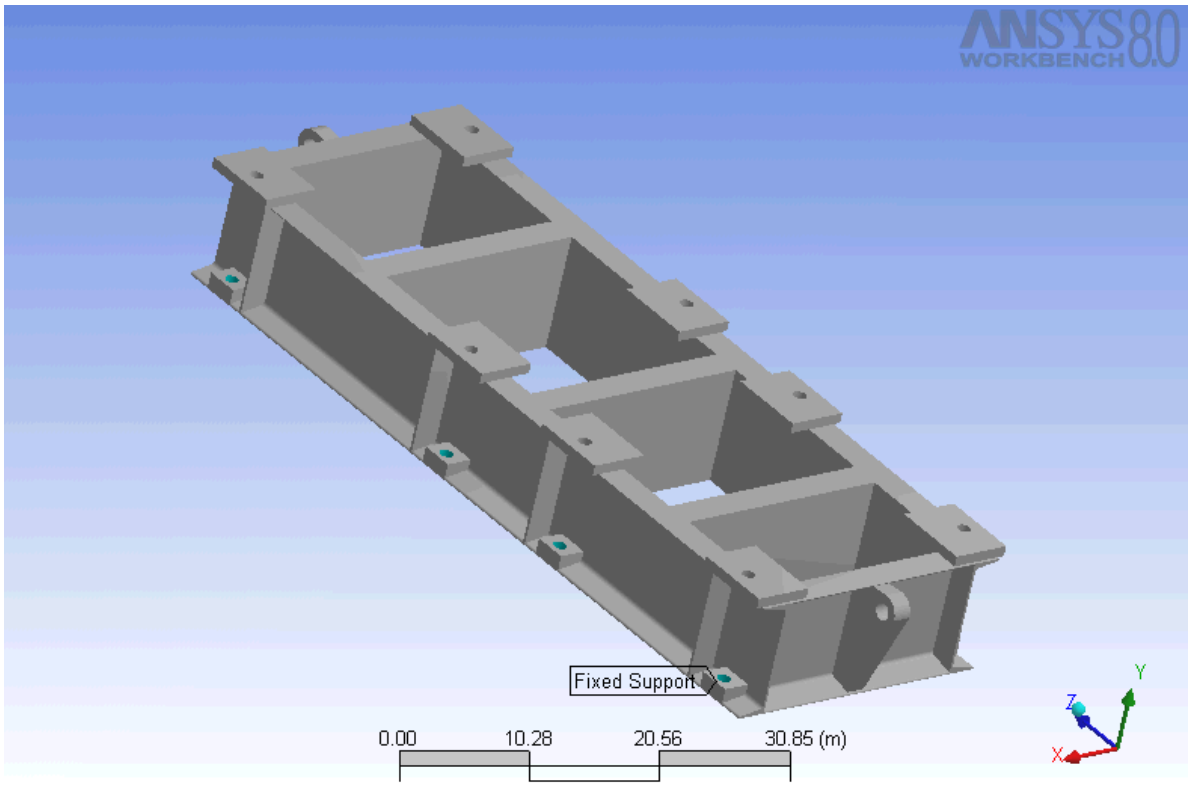
2nd Frequency 10.726 Hz

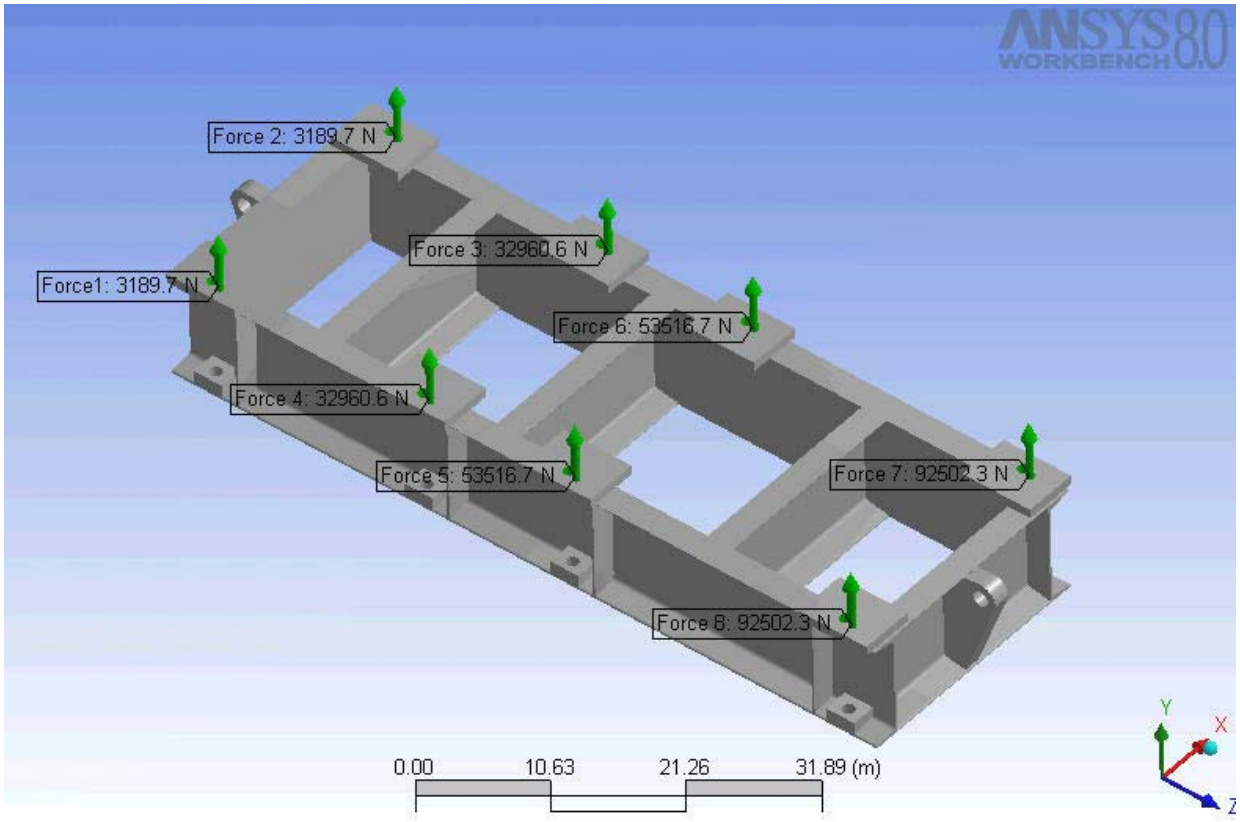
3rd Frequency 11.022 Hz

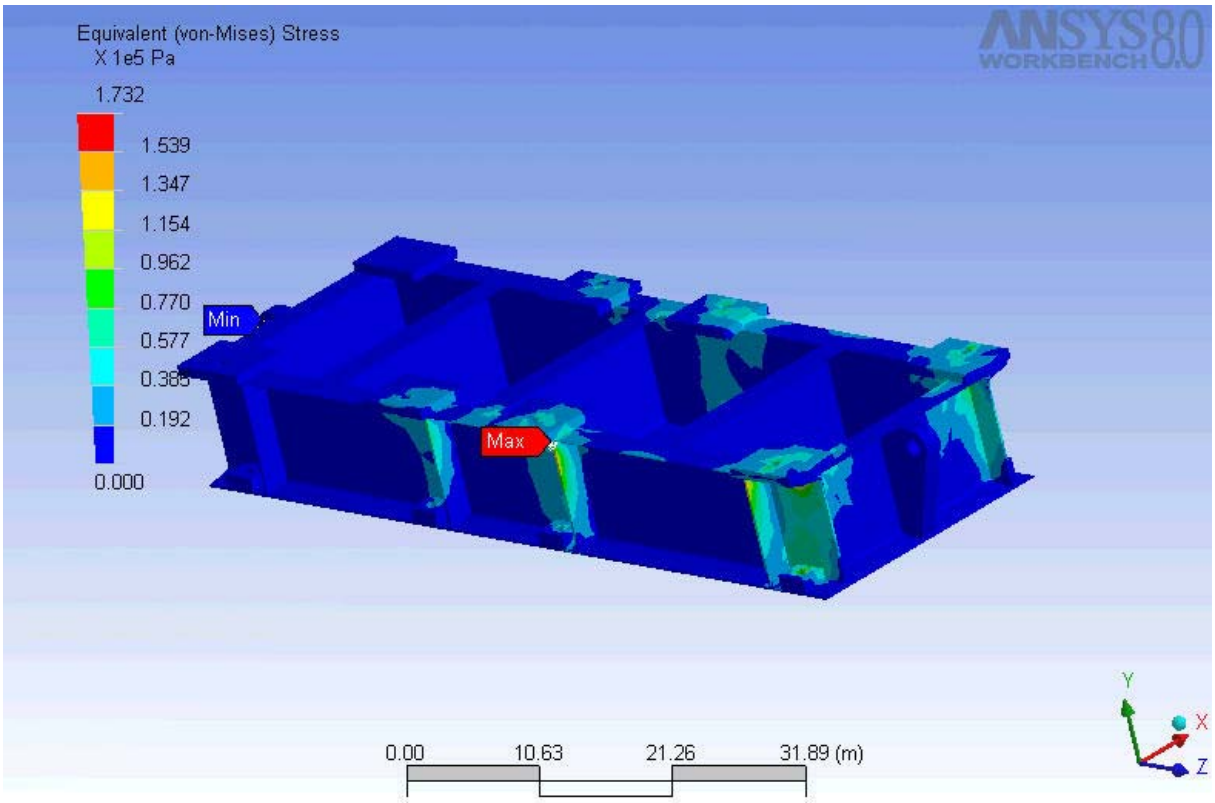
4th Frequency 12.113 Hz

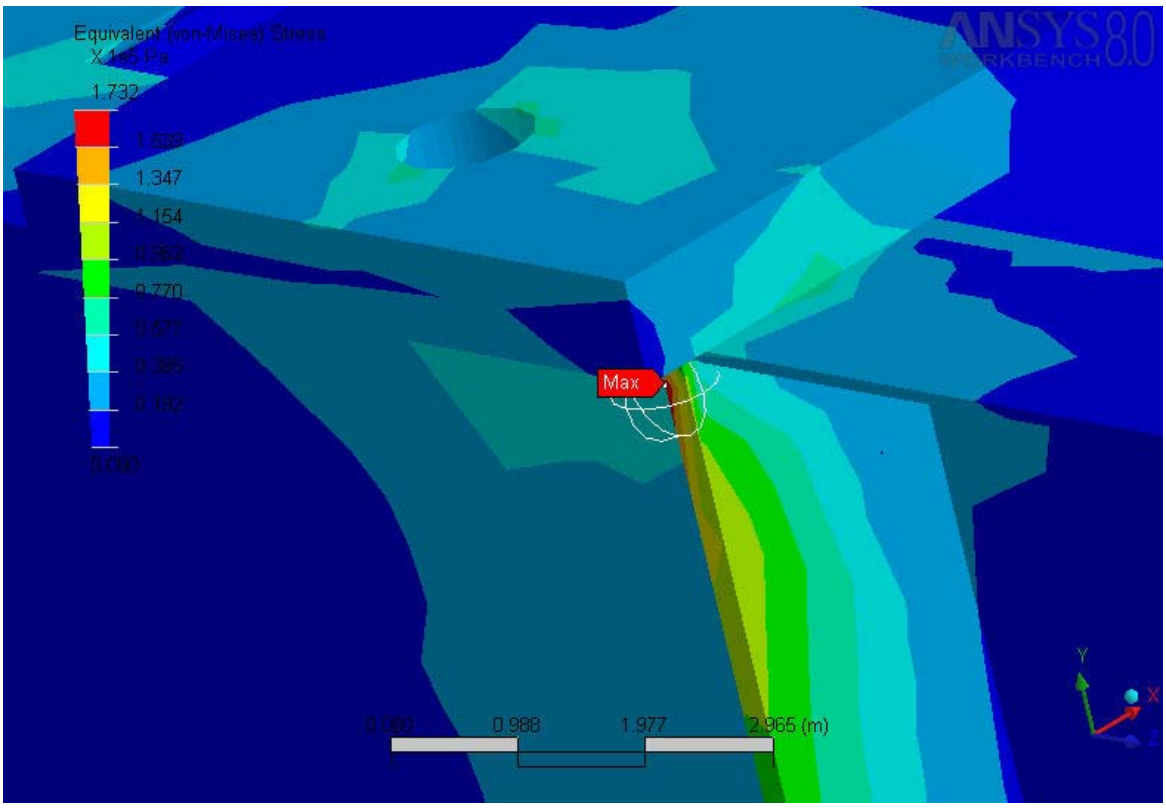
5th Frequency 12.357 Hz

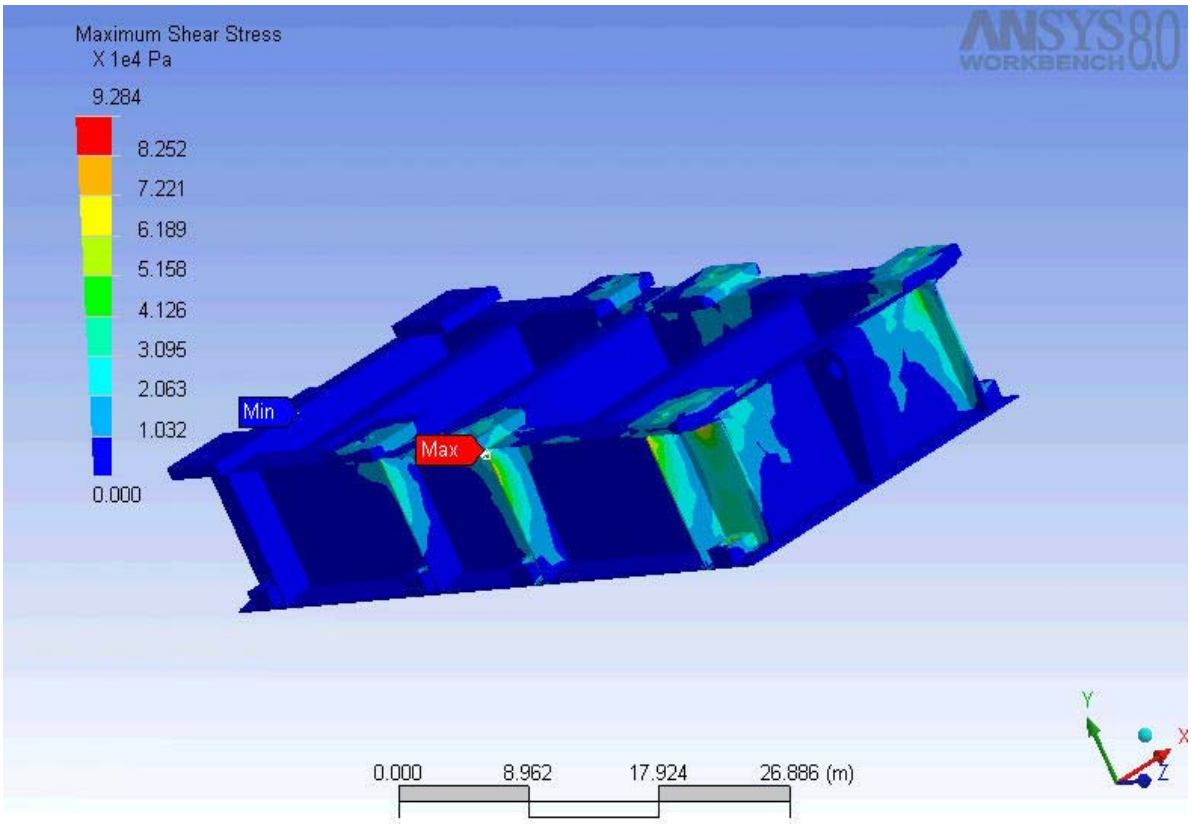
6th Frequency 12.434 Hz

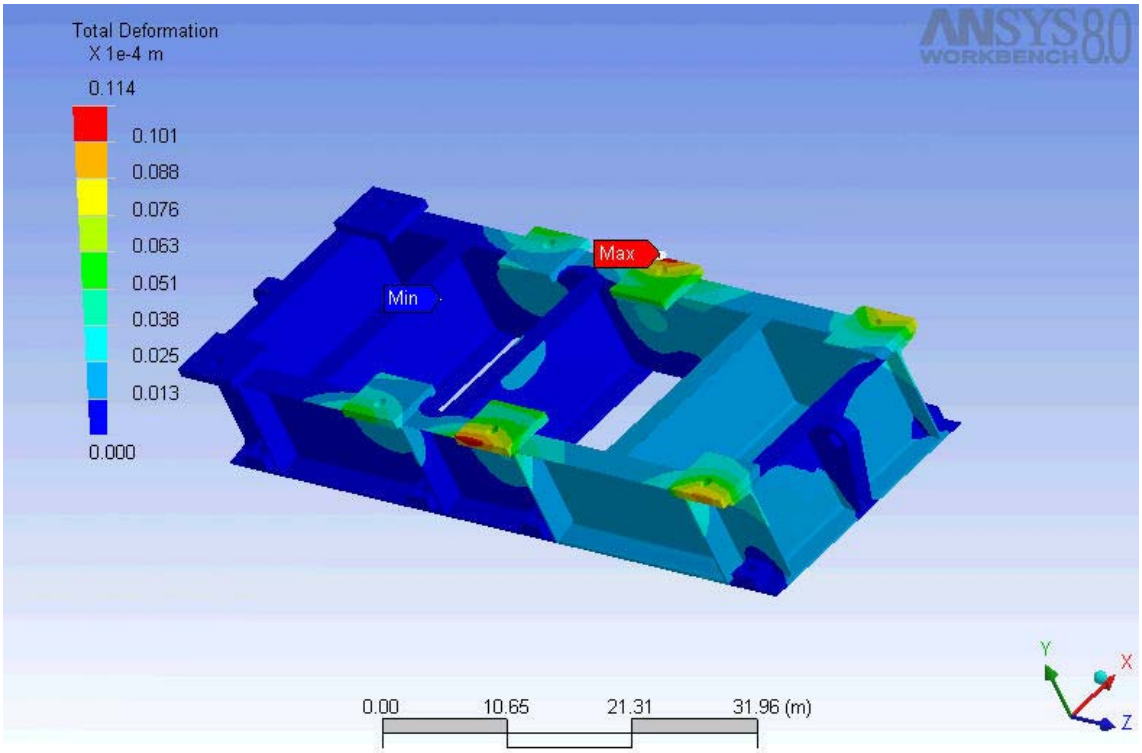

ANSYS 8.0
WORKBENCH
Fixed Support
0.00
10.28
20.56
30.85 (m)
Y
Z
X

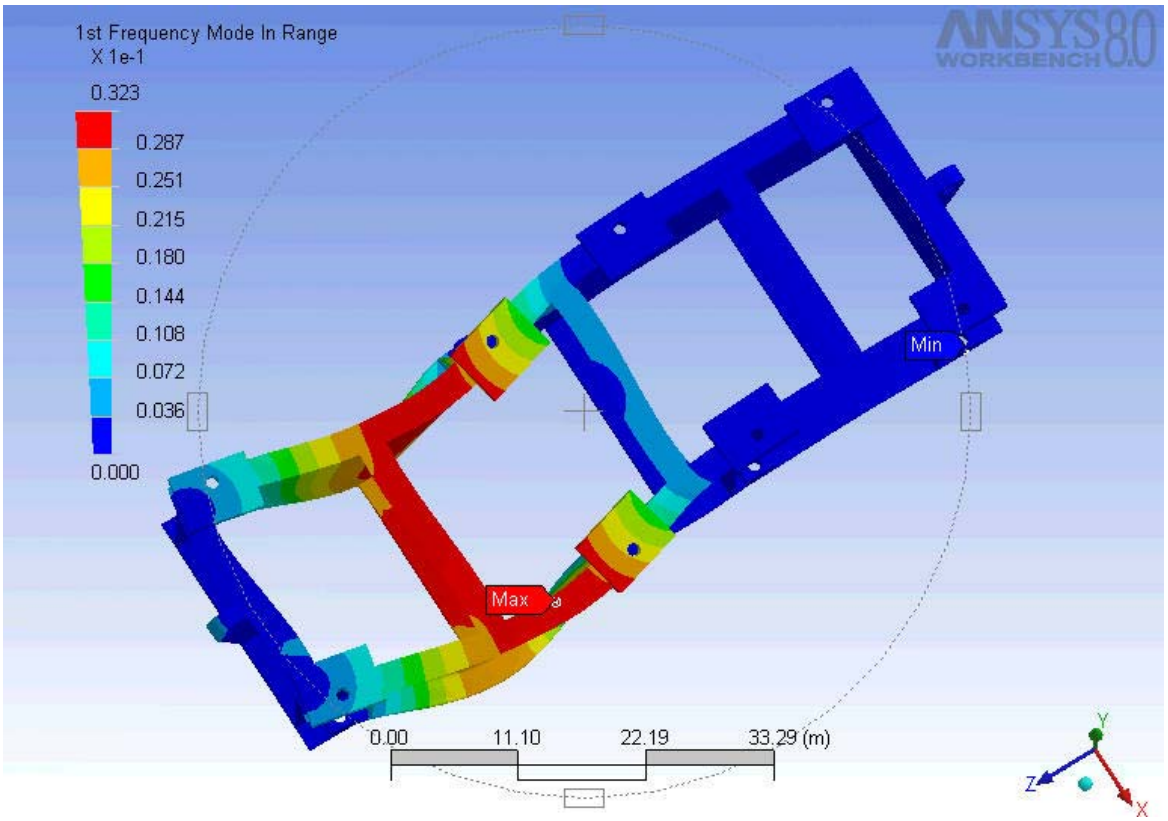

ANSYS 8.0
WORKBENCH
Force 2: 3189.7 N
Force 3: 32960.6 N
Force1: 3189.7 N
Force 6: 53516.7 N
Force 4: 32960.6 N
Force 5: 53516.7 N
Force 7: 92502.3 N
Force 8: 92502.3 N
0.00
10.63
21.26
31.89 (m)
Y
X
Z

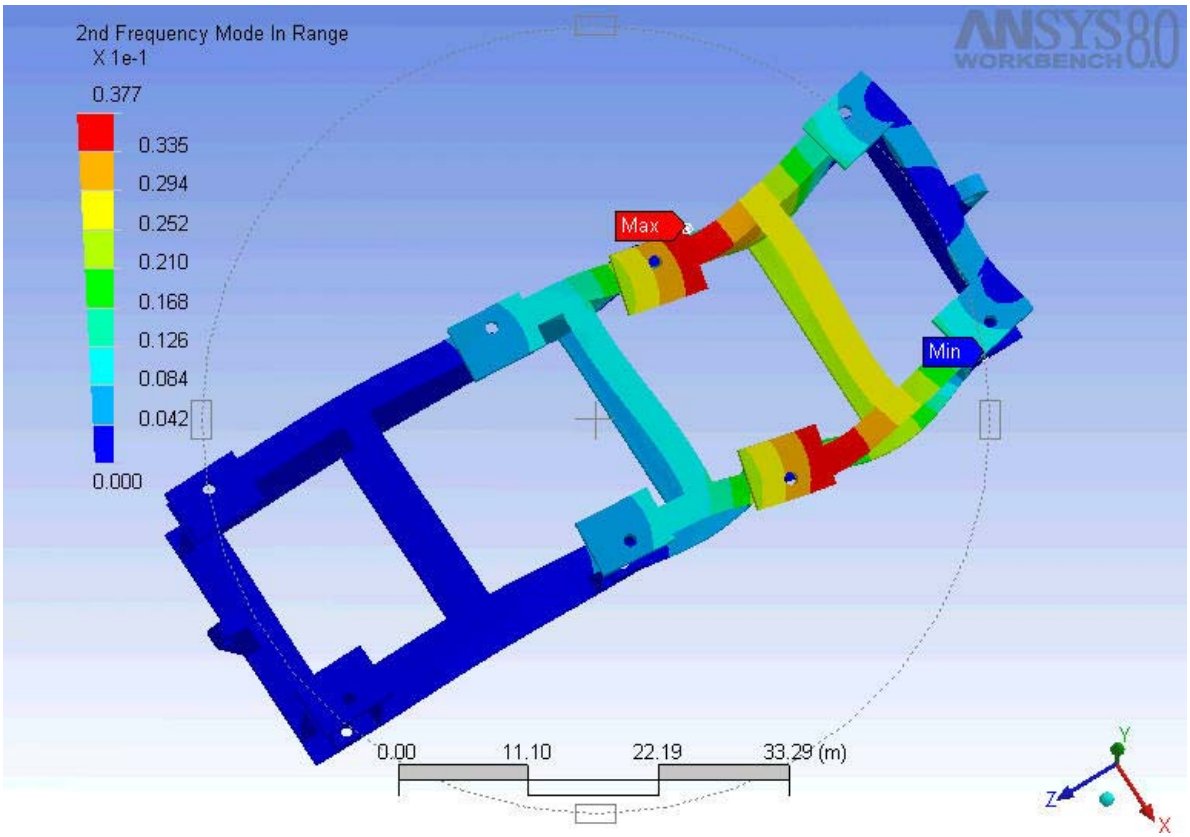

Equivalent (von-Mises) Stress
X 1e5 Pa
1.732
1.539
1.347
1.154
0.962
0.770
0.577
0.385
0.192
0.000
Min
Max
ANSYS 8.0
WORKBENCH
0.00
10.63
21.26
31.89 (m)
Y
X
Z

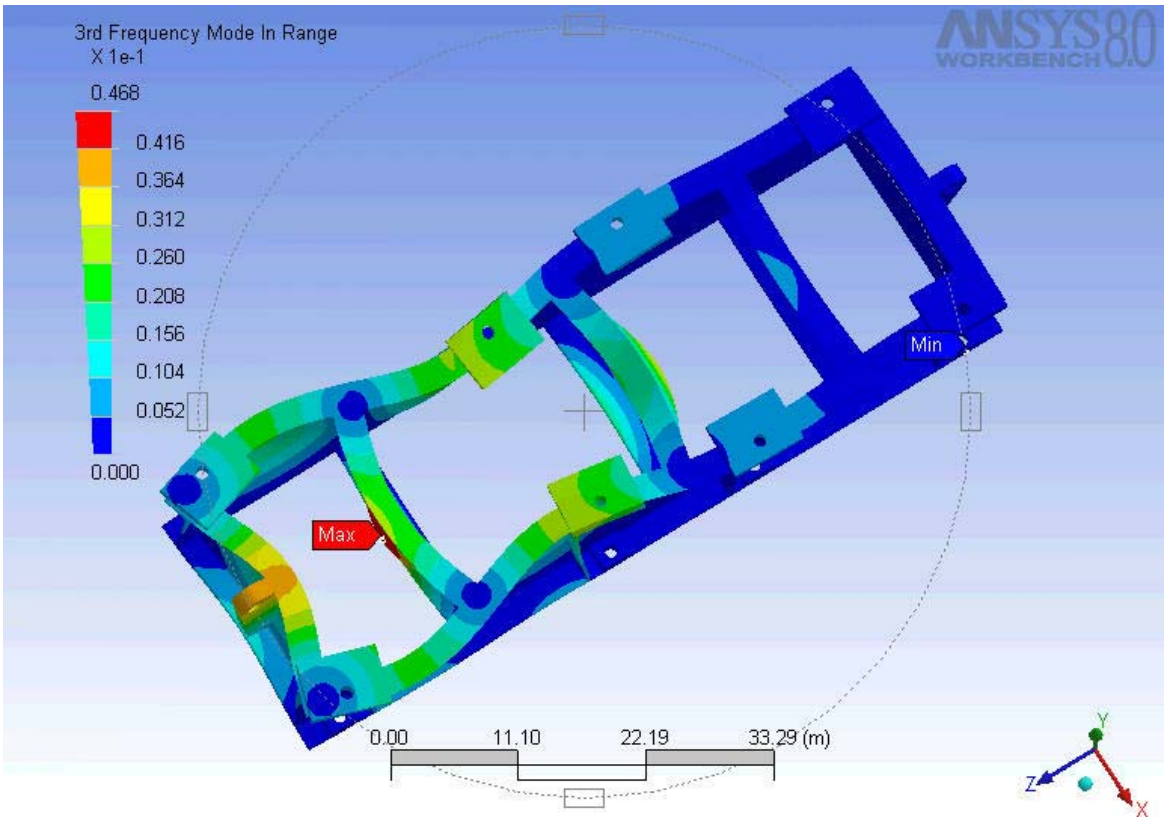

Equivalent (von-Mises) Stress
1.732
1.347
Max
ANSYS 8.0
WORKBENCH
0.988
1.977
2.965 (m)
Y
X

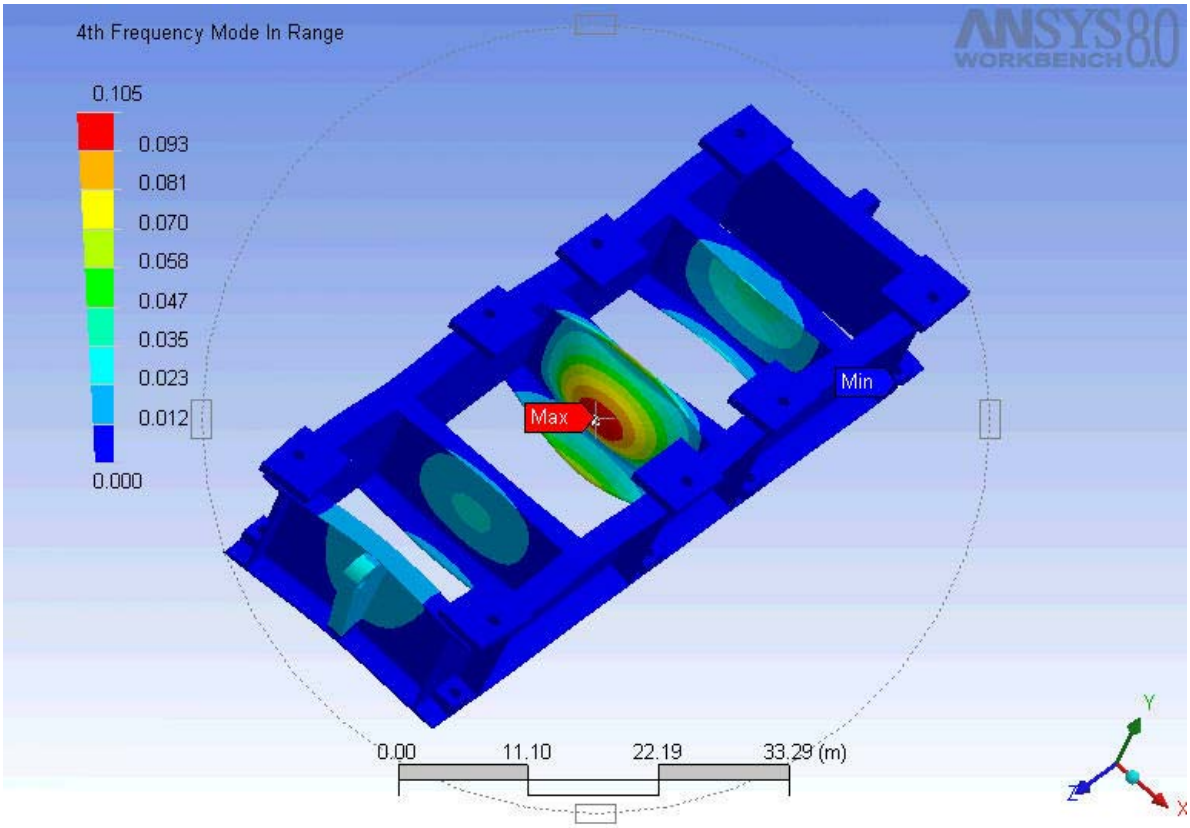

Maximum Shear Stress
X 1e4 Pa
9.284
8.252
7.221
6.189
5.158
4.126
3.095
2.063
1.032
0.000
Min
Max
ANSYS 8.0
WORKBENCH
0.000
8.962
17.924
26.886 (m)
Y
X
Z

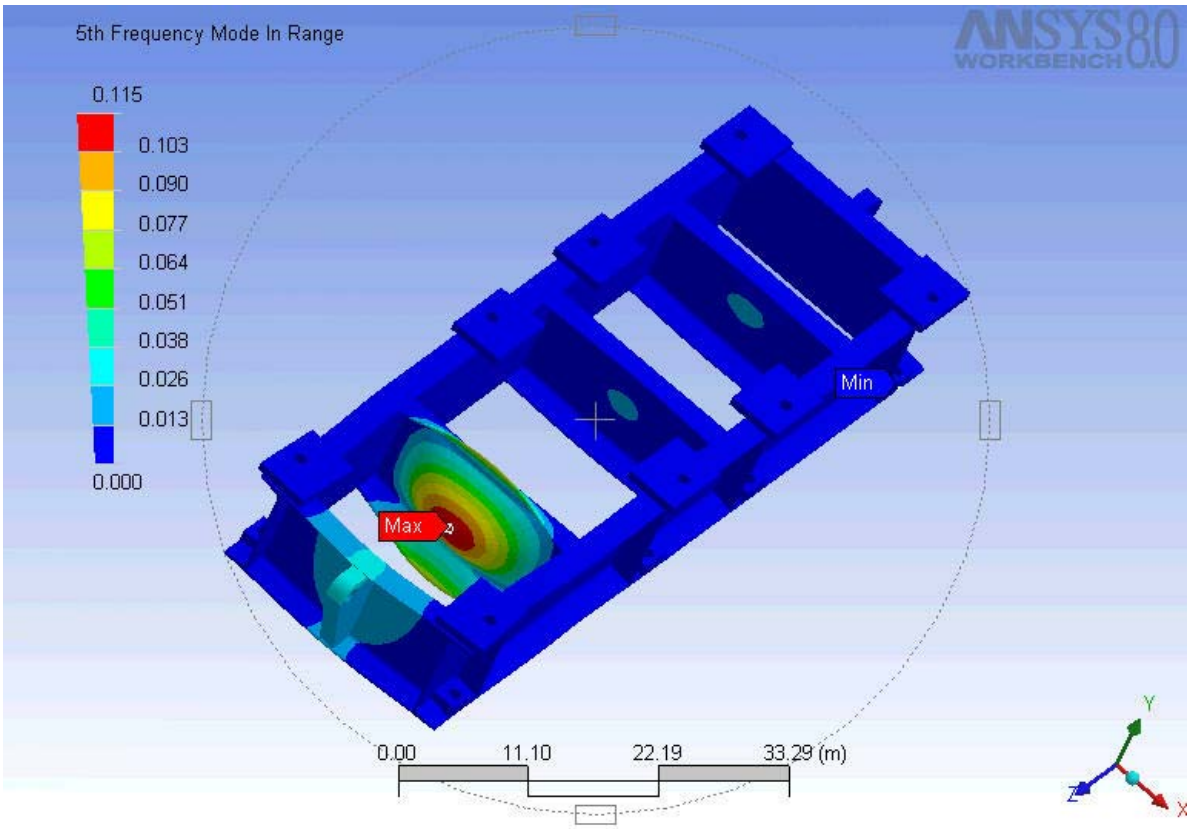

Total Deformation
X 1e-4 m
0.114
0.101
0.088
0.076
0.063
0.051
0.038
0.025
0.013
0.000
Max
Min
ANSYS 8.0
WORKBENCH
0.00
10.65
21.31
31.96 (m)
Y
X
Z

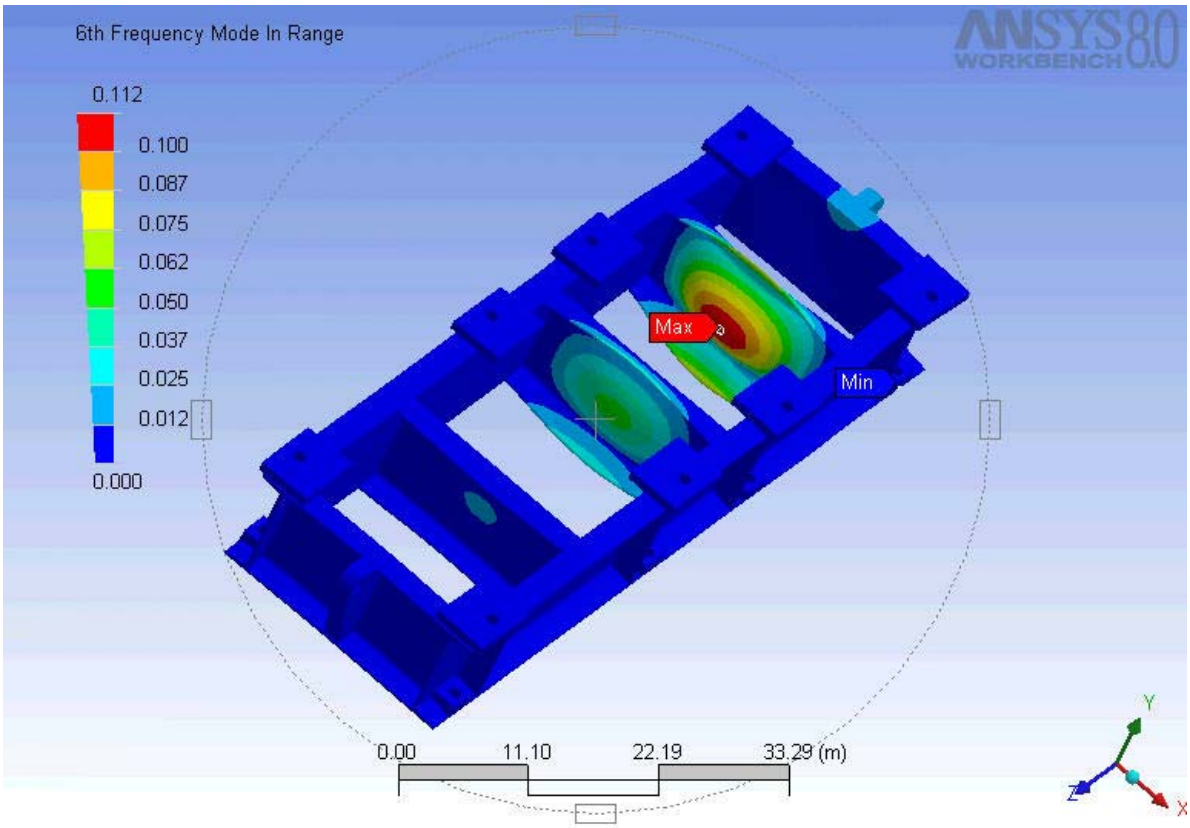

1st Frequency Mode In Range
X 1e-1
0.323
0.287
0.251
0.215
0.180
0.144
0.108
0.072
0.036
0.000
Max
Min
0.00
11.10
22.19
33.29 (m)
Y
Z
X


2nd Frequency Mode In Range
X 1e-1
0.377
0.335
0.294
0.252
0.210
0.168
0.126
0.084
0.042
0.000
Max
Min
0.00
11.10
22.19
33.29 (m)
Y
Z
X


3rd Frequency Mode In Range
X 1e-1
0.468
0.416
0.364
0.312
0.260
0.208
0.156
0.104
0.052
0.000
Max
Min
0.00
11.10
22.19
33.29 (m)
Y
Z
X


4th Frequency Mode In Range
0.105
0.093
0.081
0.070
0.058
0.047
0.035
0.023
0.012
0.000
Max
Min
0.00
11.10
22.19
33.29 (m)
Y
Z
X


5th Frequency Mode In Range
ANSYS 8.0
WORKBENCH
0.115
0.103
0.090
0.077
0.064
0.051
0.038
0.026
0.013
0.000
Max
Min
0.00
11.10
22.19
33.29 (m)
Y
Z
X


6th Frequency Mode In Range
ANSYS 8.0
WORKBENCH
0.112
0.100
0.087
0.075
0.062
0.050
0.037
0.025
0.012
0.000
Max
Min
0.00
11.10
22.19
33.29 (m)
Y
Z
X

Chapter 5

Scope for future work

This project is taken in to consideration only standard type of gearboxes. In the same way software can be developed to draw base frames for special type of gearboxes with out much difficulty. For doing that we need to do lot of data collection as well as lot of programming.

Chapter 6

Conclusions

The software developed for generating the base frames for standard type of industrial gearboxes is working satisfactorily. The software can also be used for any new designs conveniently with out much difficulty.

The results obtained from stress analysis and model analyses of base frame for SCN-800 gearbox are all safe. The stresses developed are with in the limits. The total deformation the base frame is also with in the permitted range.

References

1.1 Design department, catalogue, ELECON Engg. Co. Ltd.

2.1.1 Dr.ING.J.LINDER, design of steel beams and beam columns, Technical University Berlin, Germany, ICSS 95,PSG college of technology, Coimbatore, INDIA, 103-117.

2.1.2 PETER OSTERRIEDER and FRANCK WERNER[1] , interaction of local and global buckling in thin-walled beams, BTU Cottbus Germany, [1] HAB Weimar Germany, ICSS 95, 247-261.

2.1.3 BRADFORD M.A., lateral-distorsional buckling of steel I-section members, journal of construction steel research, 23 (1-3), 97-116.

2.1.4 N.E.SHANMUGAM, strength of thin walled steel box beam columns, department of civil engineering, National University of Singapore, ICSS 95, 291-311.

2.1.5 INAMDAR.V.M., solution of structural engineering problems by theory of distributions, Ph.D thesis, 1987, south Gujarat University, Surat, Gujarat, 395007, INDIA.

2.1.6 INAMDAR.V.M and BALMUKHUND P. PARIKH, use of generalized functions for buckling columns with variable moment of inertia, ICSS 95, 311-321.

2.2.1 PETER WIDAS, Introduction to Finite Element Analysis, Virginia Tech Material Science and Engineering.

3.1 DESIGN DEPARTMENT, ELECON Engg. Co. Ltd.

3.2 REFERENCE DRAWINGS, ELECON Engg. Co. Ltd.

Appendix

Auto LISP language

Introduction

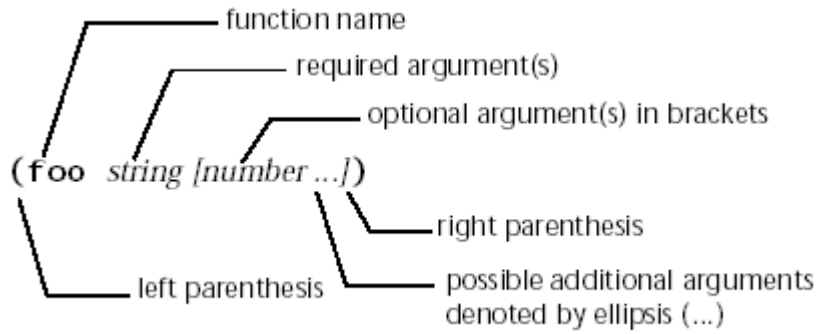
AutoLISP, an implementation of the LISP programming language, is an integral part of the AutoCAD package. With AutoLISP, you can write macro programs and functions in a powerful, high-level language suited to graphics applications. AutoLISP is easy to use and very flexible
The AutoLISP evaluator processes expressions according to the order and data type of the code within the parentheses. Before you can fully utilize AutoLISP, you must understand the differences between the data types and how to use them.

AutoLISP Expressions

(fun1 (fun2 arguments)(fun3 arguments)) [A.1]

The first function, fun1, has two arguments, and the other functions, fun2 and fun3, have one argument each. The functions fun2 and fun3 are surrounded by function fun1, so their return values are passed to fun1 as arguments. Function fun1 evaluates those two arguments and returns the value to the command line.
Example:
_$ (* 2 (+ 5 10))
30

AutoLISP Function Syntax



AutoLISP Data Types

1. Integers
2. Real
3. Strings
4. Lists
5. Selection Sets
6. Entity Names
7. VLA-objects
8. File Descriptors
9. Symbols and Variables

Functions used in Programs

Defun:

Defines a function

(defun *sym ([arguments] [/ variables...]) expr...*)

Arguments

sym

A symbol naming the function.

arguments

The names of arguments expected by the function.

/ variables

The names of one or more local variables for the function.

The slash preceding the variable names must be separated from the first local name and from the last argument, if any, by at least one space.

expr

Any number of AutoLISP expressions to be evaluated when the function executes.

If you do not declare any arguments or local symbols, you must supply an empty set of parentheses after the function name.
If duplicate argument or symbol names are specified, AutoLISP uses the first occurrence of each name and ignores the following occurrences.

Return Values

The result of the last expression evaluated.
WARNING! Never use the name of a built-in function or symbol for the sym argument to defun. This overwrites the original definition and makes the built-in function or symbol inaccessible. To get a list of built-in and previously defined functions, use the atoms-family function.

Examples

(defun myfunc (x y) ...) Function takes two arguments
(defun myfunc (/ a b) ...) Function has two local variables
(defun myfunc (x / temp) ...) One argument, one local variable
(defun myfunc () ...) No arguments or local variables

setq:

Sets the value of a symbol or symbols to associated expressions

(setq *sym expr [sym expr]...*)

This is the basic assignment function in AutoLISP. The setq function can assign multiple symbols in one call to the function.

Arguments

sym

A symbol. This argument is not evaluated.

expr

An expression.

Return Values

The result of the last expr evaluated.

Examples

The following function call set variable a to 5.0:

Command: (setq a 5.0)

5.0

List:

Takes any number of expressions, and combines them into one list

(list *[expr...]*)

This function is frequently used to define a 2D or 3D point variable (a list of two or three reals).

Arguments

expr

An AutoLISP expression.

Return Values

A list, unless no expressions are supplied, in which case list returns nil.

Examples

Command: (list 'a 'b 'c)

(A B C)

Command: (list 'a '(b c) 'd)

(A (B C) D)

Cons:

Adds an element to the beginning of a list, or constructs a dotted list

(cons *new-first-element list-or-atom*)

Arguments

new-first-element

Element to be added to the beginning of a list. This element can be an atom or a list.

list-or-atom

A list or an atom.

Return Values

The value returned depends on the data type of list-or-atom. If list-or-atom is a list, cons returns that list with new-first-element added as the first item in the list. If list-or-atom is an atom, cons returns a dotted pair consisting of new-first-element and list-or-atom.

Examples

Command: (cons 'a '(b c d))

(A B C D)

Command: (cons '(a) '(b c d))

((A) B C D)

Princ:

Prints an expression to the command line, or writes an expression to an open file

(princ *[expr [file-desc]]*)

This function is the same as prin1, except control characters in expr are printed without expansion. In general, prin1 is designed to print expressions in a way that is compatible with load, while princ prints them in a way that is readable by functions such as read-line.

Arguments

expr

A string or AutoLISP expression. Only the specified expr is printed; no newline or space is included.

file-desc

A file descriptor for a file opened for writing.

Return Values

The value of the evaluated expr. If called with no arguments, princ returns a null symbol.

If:

Conditionally evaluates expressions

(if *testexpr thenexpr [elseexpr*])

Arguments

testexpr

Expression to be tested.

thenexpr

Expression evaluated if testexpr is not nil.

elseexpr

Expression evaluated if testexpr is nil.

Return Values

The if function returns the value of the selected expression. If elseexpr is missing and testexpr is nil, then if returns nil.

Examples

Command: (if (= 1 3) "YES!!" "no.")

"no."

Command: (if (= 2 (+ 1 1)) "YES!!")

"YES!!"

Load_dialog:

Loads a DCL file

(load_dialog *dclfile*)

The load_dialog function searches for files according to the AutoCAD library search path.
This function is the complement of unload_dialog. An application can load multiple DCL files with multiple load_dialog calls.

Arguments

dclfile

A string that specifies the DCL file to load. If the dclfile argument does not specify a file extension, .dcl is assumed.

Return Values

A positive integer value (dcl_id) if successful, or a negative integer if load_dialog can't open the file. The dcl_id is used as a handle in subsequent new_dialog and unload_dialog calls.

New_dialog:

Begins a new dialog box and displays it, and can also specify a default action

(new_dialog *dlgname dcl_id [action [screen-pt]]*)

Arguments

dlgname

A string that specifies the dialog box.

dcl_id

The DCL file identifier obtained by load_dialog.

action

A string that contains an AutoLISP expression to use as the default action. If you don't want to define a default action, specify an empty string (""). The action argument is required if you specify screen-pt.
The default action is evaluated when the user picks an active tile that doesn't have an action or callback explicitly assigned to it by action_tile or in DCL.

screen-pt

A 2D point list that specifies the X,Y location of the dialog box on the screen. The point specifies the upper-left corner of the dialog box. If you pass the point as'(-1 -1), the dialog box is

opened in the default position (the center of the AutoCAD graphics screen).

Return Values

T, if successful, otherwise nil.

Action_tile:

Assigns an action to evaluate when the user selects the specified tile in a dialog box

(action_tile *key action-expression*)

The action assigned by action_tile supersedes the dialog box's default action (assigned by new_dialog) or the tile's action attribute, if these are specified. The expression can refer to the tile's current value as $value, its name as $key, its application-specific data (as set by client_data_tile) as $data, its callback reason as $reason, and its image coordinates (if the tile is an image button) as $x and $y.

Arguments

key

A string that names the tile that triggers the action (specified as its key attribute). This argument is case-sensitive.

action-expression

A string naming the expression evaluated when the tile is selected.

NOTE You cannot call the AutoLISP command function from the action_tile function.

Return Values

T

Strcat:

Returns a string that is the concatenation of multiple strings

(strcat *[string [string]...]*)

Arguments

string

A string.

Return Values

A string. If no arguments are supplied, strcat returns a zero-length string.

Examples

Command: (strcat "a" "bout")

"about"

Command: (strcat "a" "b" "c")

"abc"

Prong:

Evaluates each expression sequentially and returns the value of the last expression

(progn *[expr]...*)

You can use progn to evaluate several expressions where only one expression is expected.

Arguments

expr

One or more AutoLISP expressions.

Return Values

The result of the last evaluated expression.

Examples

The if function normally evaluates one then expression if the test expression evaluates to anything but nil. The following example uses progn to evaluate two expressions following if:

```
(if (= a b)
  (progn
    (princ "\nA = B ")
    (setq a (+ a 10) b (- b 10))
  )

)
```

Done_dialog:

Terminates a dialog box

(done_dialog *[status]*)

Arguments

status

A positive integer that start_dialog will return instead of returning 1 for OK or 0 for Cancel. The meaning of any status value greater than 1 is determined by your application.

You must call done_dialog from within an action expression or callback function (see "action_tile").

Return Values

A two-dimensional point list that is the (X,Y) location of the dialog box when the user exited it.

Start_dialog:

Displays a dialog box and begins accepting user input

(start_dialog)

You must first initialize the dialog box by a previous new_dialog call. The dialog box remains active until an action expression or callback function calls done_dialog. Usually done_dialog is associated with the tile whose key is "accept" (typically the OK button) and the tile whose key is "cancel" (typically the Cancel button).
The start_dialog function has no arguments.

Return Values

The start_dialog function returns the optional status passed to done_dialog. The default value is 1 if the user presses OK, 0 if the user presses Cancel, or -1 if all dialog boxes are terminated with term_dialog. If done_dialog is passed an integer status greater than 1, start_dialog returns this value, whose meaning is determined by the application.

Unload_dialog:

Unloads a DCL file

(unload_dialog *dcl_id*)

Unloads the DCL file associated with dcl_id (obtained from a previous new_dialog call) from memory.
It is generally not necessary to unload a DCL definition from memory, unless you are running low on memory or need to update the DCL dialog definition from a new file.

Arguments

dcl_id

A DCL file identifier obtained from a previous load_dialog call.

Return Values

The unload_dialog function always returns nil.

Cdr:

Returns a list containing all but the first element of the specified list

(cdr *list*)

Arguments

list

A list.

Return Values

A list containing all the elements of list, except the first element (but see Note below). If the list is empty, cdr returns nil.
NOTE When the list argument is a dotted pair, cdr returns the second element without enclosing it in a list.

Examples

Command: (cdr '(a b c))

(B C)

Command: (cdr '((a b) c))

(C)

Assoc:

Searches an association list for an element and returns that association list entry

(assoc *element alist*)

Arguments

element

Key of an element in an association list.

alist

An association list to be searched.

Return Values

The alist entry, if successful. If assoc does not find element as a key in alist, it returns nil.

Examples

Command: (setq al '((name box) (width 3) (size 4.7263) (depth 5)))

((NAME BOX) (WIDTH 3) (SIZE 4.7263) (DEPTH 5))

Command: (assoc 'size al)

(SIZE 4.7263)

Command:

Executes an AutoCAD command

(command *[arguments]* ...)

Arguments

arguments

AutoCAD commands and their options.

The arguments to the command function can be strings, reals, integers, or points, as expected by the prompt sequence of the executed command. A null string ("") is equivalent to pressing ENTER on the keyboard. Invoking command with no argument is equivalent to pressing ESC and cancels most AutoCAD commands.

The command function evaluates each argument and sends it to AutoCAD in response to successive prompts. It submits command names and options as strings, 2D points as lists of two reals, and 3D points as lists of three reals. AutoCAD recognizes command names only when it issues a Command prompt.

Note that if you issue command from Visual LISP, focus does not change to the AutoCAD window. If the command requires user input, you'll see the return value (nil) in the Console window, but AutoCAD will be waiting for input. You must manually activate the AutoCAD window and respond to the prompts. Until you do so, any subsequent commands will fail.

Return Values

nil

Examples

The following example sets two variables pt1 and pt2 equal to two point values 1,1 and 1,5. It then uses the command function to issue the LINE command and pass the two point values.

Command: (setq pt1 '(1 1) pt2 '(1 5))

(1 5)

Command: (command "line" pt1 pt2 "")
line From point:
To point:
To point:

Command: nil

Polar:

Returns the UCS 3D point at a specified angle and distance from a point

(polar *pt ang dist*)

Arguments

pt

A 2D or 3D point.

ang

An angle expressed in radians relative to the X axis, with respect to the current construction plane. Angles increase in the counterclockwise direction.

dist

Distance from the specified pt.

Return Values

A 2D or 3D point, depending on the type of point specified by pt.

Examples

Supplying a 3D point to polar:

Command: (polar '(1 1 3.5) 0.785398 1.414214)

(2.0 2.0 3.5)

Supplying a 2D point to polar:

Command: (polar '(1 1) 0.785398 1.414214)

(2.0 2.0)

B. Dialog Control Language (DCL)

Dialog boxes are defined by ASCII files written in dialog control language (DCL). The elements in a dialog box, such as buttons and edit boxes, are known as tiles. The size and functionality of each tile is controlled by the tile's attributes. The size of the dialog box and the layout of its parts are set automatically with a minimum of positioning information. VLISP provides a tool for viewing dialog boxes, and provides functions for controlling dialog boxes from application programs.

A dialog box consists of the box and the tiles within it. The basic tile types are predefined by the programmable dialog box (PDB) facility [B.1].

You can create complex tiles, called subassemblies, by grouping tiles into rows and columns, with or without an enclosing box or border. A row or column of tiles is referred to as a cluster. Subassemblies define groups of tiles or clusters that can be used in many dialog boxes. For example, the OK, Cancel, and Help buttons are grouped into a subassembly, defined as a row (cluster) of three button tiles and some spacing separating the buttons.

Subassemblies are treated as single tiles. The tiles within a subassembly are called children. DCL files are organized in a tree structure. At the top of the tree is a (dialog) tile that defines the dialog box itself. The following diagram shows a DCL file structure:

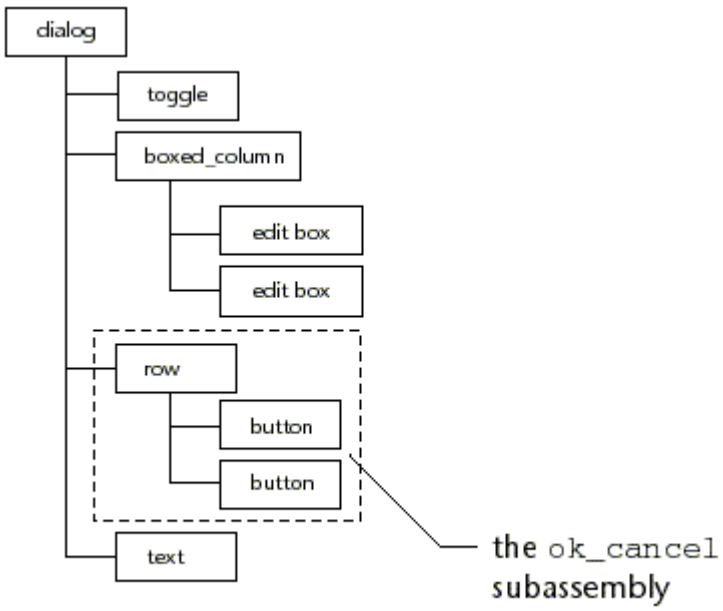


Fig B.1: DCL file structure.

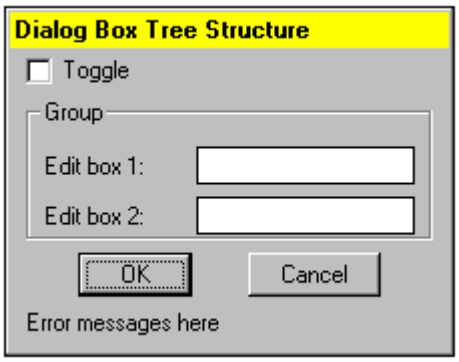


Fig B.2: dialog box tree structure.

The layout, appearance, and behavior of a tile or subassembly are specified in DCL by the tile's attributes. For example, the dialog itself, and most predefined tile types, has a label attribute that specifies the text associated with the tile. The label of a dialog box defines the caption at the top of the dialog box, the label of a button specifies the text inside the button, and so on.

DCL also enables you to define new tiles, called prototypes, that are not necessarily associated with a specific dialog box. This is useful when you want to use the same component in several dialog boxes. You can reference prototype tiles from other DCL files and change their attributes the same way you change predefined tiles.
Before you program a dialog box, plan both the dialog box and the application in detail before you code and debug. The sequence in which the data is entered will vary with each user. The need to anticipate a variety of user actions imposes a program structure that is less linear than conventional programming, but is more reflective of the way users work.

A single DCL file can contain the description of one or more dialog boxes, or it can contain only prototype tiles and subassemblies for use by other DCL files. A DCL file consists of the following three parts, which can appear in any order. Depending on your application, only one or more of these parts is required.

a. References to other DCL files
b. Prototype tile and subassembly definitions

c. These are tile definitions you can refer to in subsequent tile definitions (including dialog box definitions).
d. Dialog box definitions
e. These define the attributes of tiles or override the attributes defined in prototype tiles and subassemblies.

When you create dialog boxes, you must create a new, application-specific DCL file. All DCL files can use the tiles defined in the base.dcl file. A DCL file can also use tiles defined in another DCL file by naming the other file in what is called an include directive. You can create your own hierarchy of DCL files, as shown in the following figure:

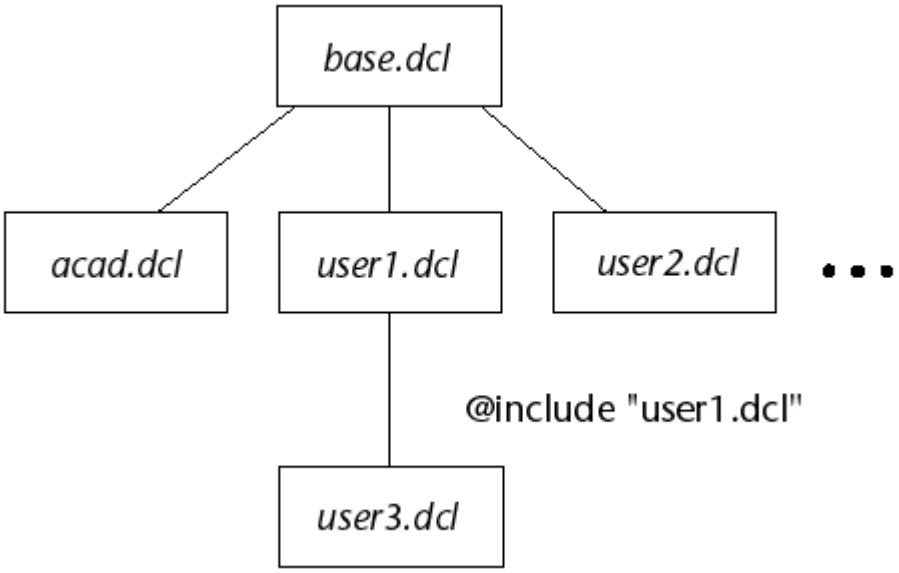


Fig B.3: hierarchy of DCL files

In this figure, the user1.dcl and user2.dcl files are independent of each other, but user3.dcl uses tiles defined in user1.dcl. The include directive has the form:

@include filename

Where filename is a quoted string containing the full name of the other DCL file. For example, the following directive includes a file named usercore.dcl:

@include "usercore.dcl"

New tiles are created by tile definitions. If a tile definition appears outside a dialog box definition, it is a prototype or a subassembly. Prototypes and subassemblies can be used in dialog box definitions by tile references. Each reference to a definition inherits the attributes of the original tile. When referring to prototypes, you can change the values of the inherited attributes or add new attributes. When referring to subassemblies, you cannot change or add attributes.
If you need multiple instances of a tile with some attributes in common, it is easiest to define and name a prototype that contains only the common attributes. Then, in each reference to the prototype, you can change attributes or add new ones, but you do not have to list all the common attributes each time you reference the tile. Because attributes are inherited, you will more often need to create tile references especially references to the predefined tiles—than to define new tiles.

Tile definitions have the following form:

```
name : item1 [ : item2 : item3 ... ] {
        attribute = value;
        ...

}
```

Tile references have one of the following forms:

```
name;
or
: name {
    attribute = value;
    . . .
}
```

where name is the name of a previously defined tile. Tile names are case sensitive. In the first instance, all the attributes defined in name are incorporated into the reference. In the second instance, the attribute definitions within the curly braces either supplement or replace the definitions inherited from name. Because this is a tile reference, as opposed to a definition, the attribute changes apply only to this instance of the tile.

Within the curly braces of a tile definition or reference, you specify attributes and assign them values using the following form:

attribute = value ;

where attribute is a valid keyword and value is the value assigned to the attribute. An equal sign (=) separates the attribute from the value, and a semicolon (;) ends the assignment statement.

```
hello : dialog {
   label = "Sample Dialog Box";
   : text {
     label = "Hello, world";
   }
   : button {
     key = "accept";
     label = "OK";
     is_default = true;
   }
}
```

C. INDIAN STANDARD CHANNEL

This appendix is containing the standard type of channels used in the chapter 10.2. There are basically two types of channels, MC and MCP [].

Designation - Medium weight channel shall be designated MC if the flanges are sloping. If the flanges are parallel the channel shall be designated MCP.

Here there is only the nominal dimensions of the channels, which are used in the programming. There are no details here on weight and etc.

The fallowing fig shows the typical channel cross section with symbols used as a standard to describe the channel.

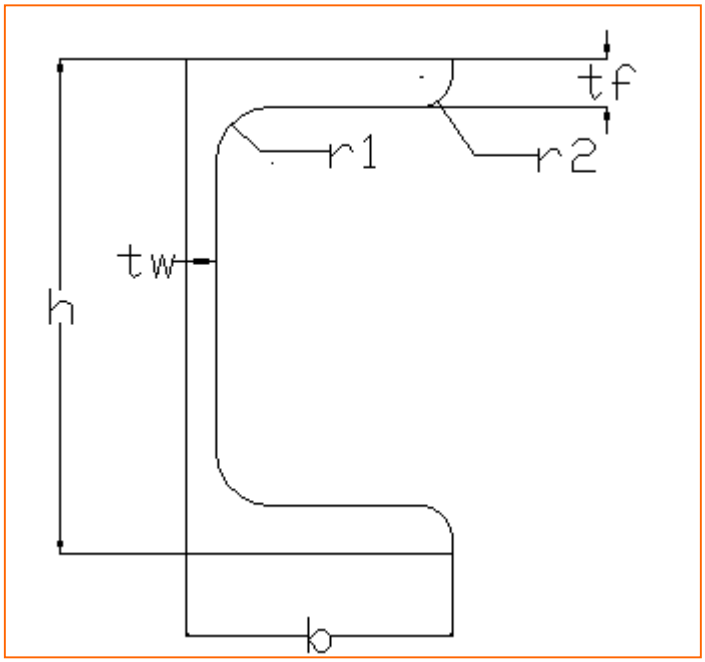


Fig C.1: Indian Standard MC type channel

h	75	100	125	150	175	200	225	250	300	350	400
b	40	50	65	75	75	75	80	80	90	100	100
tf	7.3	7.5	8.1	9.0	10.2	11.4	12.4	14.1	13.6	13.5	15.3
tw	4.4	4.7	5.0	5.4	5.7	6.1	6.4	7.1	7.6	8.1	8.6
r1	8.5	9.0	9.5	10.0	10.5	11.0	12.0	12.0	13.0	14.0	15.0
r2	4.5	4.5	5.0	5.0	5.5	5.5	6.0	6.0	6.5	7.0	7.5

Table C.1: Nominal dimensions of the channel

Above table is taken from IS: 808 (part-III) – 1979.

The fallowing table display the nomenclature used in this standard.

h	Height of the channel
b	Width of the channel
tf	Flange thickness
tw	Web thickness
r1	Radius of root
r2	Radius of toe

Table C.2: Nomenclature of the standard channel.