The Physics behind the Powertrain Mounting System model

Calculating the centre of mass displacement

For the purposes of this project, the powertrain is assumed to be a 6 degree of freedom rigid body. This rigid body is attached to a rigid frame/support structure by a variable number of mounts (3 incase of our project). For the project, we assume the frame to also be a rigid body.

The mounts and the engine body may or may not have the same axes. For generality, we assume the mounts and the powertrain to have different axes. The origin of the coordinate system is located at the centre of mass of the powertrain. Let us define the coordinate system directions as follows:

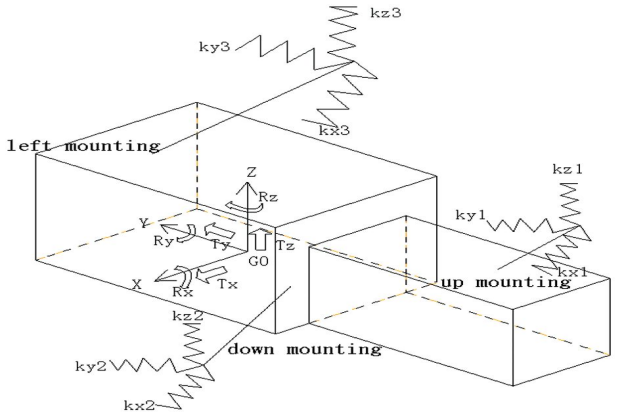
x-axis → Engine to the transmission in the crankshaft direction

y-axis → Directed by the tail of the car

z-axis → upward along the vertical direction

Let's assume the coordinates of a mount to be x', y' ,z'.

In the local coordinates of the mount, the stiffness matrix is defined as:



$$[K]'_i = \begin{bmatrix} k'_x(1+i\eta'_x) & 0 & 0 \\ 0 & k'_y(1+i\eta'_y) & 0 \\ 0 & 0 & k'_z(1+i\eta'_z) \end{bmatrix}$$

Please note that the stiffness matrix is complex in nature. Here, η represents the loss factors in the local coordinate system and $k'_{x,y,z}$ represents the spring rates. This is in accordance with the Kelvin-Voigt model, where an elastoviscous substance can be represented by a purely viscous damping force and a purely elastic string force in parallel.

For further computation, we need to transform the stiffness matrix from the mount-local coordinate system to the global coordinate system. This is done using the linear transformation matrix which is written as:

$$[T]_i = \begin{bmatrix} \cos(x'x) & \cos(y'x) & \cos(z'x) \\ \cos(x'y) & \cos(y'y) & \cos(z'y) \\ \cos(x'z) & \cos(y'z) & \cos(z'z) \end{bmatrix}$$

Where i represents the mount number, since, each mount has its own coordinate system.

The global stiffness matrix for the i^{th} mount is represented as:

$$[K]_i = [T]_i^T [K]'_i [T]_i$$

Where the matrices have been defined above.

Next, we need to map the displacements of the mounts to the displacement and rotation of the engine. Assuming the displacements to be small, the displacement of the mount is represented as $\{u\}_i$, the transformation matrix is given by:

$$[G]_i = \begin{bmatrix} 1 & 0 & 0 & 0 & z & -y \\ 0 & 1 & 0 & -z & 0 & x \\ 0 & 0 & 1 & y & -x & 0 \end{bmatrix}$$

Then the transformation is given by:

$$\{u\}_i = \begin{Bmatrix} \Delta x \\ \Delta y \\ \Delta z \end{Bmatrix}_i$$

$$= \begin{bmatrix} 1 & 0 & 0 & 0 & z & -y \\ 0 & 1 & 0 & -z & 0 & x \\ 0 & 0 & 1 & y & -x & 0 \end{bmatrix}_i \begin{Bmatrix} \Delta x \\ \Delta y \\ \Delta z \\ \Delta\theta_x \\ \Delta\theta_y \\ \Delta\theta_z \end{Bmatrix}_g$$

$$= [G]_i \{U\}$$

Here, {U} represents the displacement and rotation of the engine's centre of mass.

The engine mounts are considered to be non-perfect elastic bodies. Hence, the normal force on the engine due to displacement of the mounts is given by:

$$\{F\}_i = -\left[[G]_i^T [K]_i [G]_i\right] \{U\}$$

Disregarding the second-order terms in the rigid body equation of motion, the force balance equation for the engine becomes:

$$[M]\{\ddot{U}\} = \{F\}$$

Where {F} is the external force & [M] is the mass matrix, which takes into account both mass and moment of inertia of the powertrain in order to account for translational and rotational displacements.

$$[M] = \begin{bmatrix} m & 0 & 0 & 0 & 0 & 0 \\ 0 & m & 0 & 0 & 0 & 0 \\ 0 & 0 & m & 0 & 0 & 0 \\ 0 & 0 & 0 & I_{xx} & I_{xy} & I_{xz} \\ 0 & 0 & 0 & I_{xy} & I_{yy} & I_{yz} \\ 0 & 0 & 0 & I_{xz} & I_{yz} & I_{zz} \end{bmatrix}$$

Here, m is the engine mass and I_{kl} is the moments and product of inertia expressed in terms of the global coordinate system.

But for the force balance equation of the engine should also include the normal force due to the mounts, that is,

$$\sum_{i=1}^{n}\{F\}_i = -\left[\sum_{i=1}^{n}[G]_i^T[K]_i[G]_i\right]\{U\} = -[K]\{U\}$$

Hence, the final force balance equation becomes,

$$[M]\{\ddot{U}\} + [K]\{U\} = \{F\}_e$$

Here, $\{F\}_e$ represents the external forces, not including the mount reactions.

Since we are examining the model in the frequency domain. The external forces are considered to be periodic excitations using a Fourier series approximation. Assuming {F} to comprise a series of m sinusoidal force phasors $\{F\}_j$ with frequencies wj and corresponding phase angles α_j, j = 1,2, . ..m. it follows that:

$$\{\overline{U}\} = \sum_{j=1}^{m}\left[[K] - \omega_j^2[M]\right]^{-1}\{\overline{F}\}_j$$

Where {Ū} represents the displacement of the engine.

This modelling allows to get a final displacement metric for the engine, and how the excitation forces of various amplitudes and frequencies affect
a particular configuration of the design variable.

Finding the objective function

The objective function is given by the following equation:

$$\varphi = \sqrt{\sum_{i=1}^{n}\{f_{xi}^2(\{X\}) + f_{yi}^2(\{X\}) + f_{zi}^2(\{X\})\}}$$

{X} is a vector of design variables typically comprising spring rates or mount positions and orientations. $\bar{f}_{xi}, \bar{f}_{yi}$ and $\bar{f}_{zi}$ are the maximum values of forces transmitted to the support structure and are functions of {r(X)].

It is necessary to minimize objective function with respect to variables {X), subject to the inequality constraints

$$\{g\} = |\{U\}| - \{U\}_c \leq 0$$

where |{U}| is the vector of displacement amplitudes (only translations in this implementation) for a particular design configuration {X}. {(Ilc is a vector of specified maximum acceptable translational displacement amplitudes. This implies that

$$\max(\{g\}) = \max\{g_{x1}, g_{y1}, g_{z1}, g_{x2}, \cdots, g_{xi}, \cdots, g_{zn}\} \leq 0$$

The force transmitted to the support structure is basically the reaction forces between the engine and the engine mounts which act as the support structure. This can be done easily by converting the displacement of COM to the individual displacement of mounts. This is done using the following equation:

$$\begin{aligned}
\{u\}_i &= \begin{Bmatrix} \Delta x \\ \Delta y \\ \Delta z \end{Bmatrix}_i \\
&= \begin{bmatrix} 1 & 0 & 0 & 0 & z & -y \\ 0 & 1 & 0 & -z & 0 & x \\ 0 & 0 & 1 & y & -x & 0 \end{bmatrix}_i \begin{Bmatrix} \Delta x \\ \Delta y \\ \Delta z \\ \Delta\theta_x \\ \Delta\theta_y \\ \Delta\theta_z \end{Bmatrix}_g \\
&= [G]_i \{U\}
\end{aligned}$$

After we find these for individual mount displacements, we use these to find the maximum displacements.

Challenge faced and solution:
Here, we run into an issue that these displacements are time-dependent phasors. To solve this problem we find the value of displacement and in that, the value of the objective function for multiple values of time for a given configuration. After we get a list of values of the objective function, we simply pick the maximum value. This sampling approach was suggested by Dr Sabareesh Geetha Rajasekharan.

Coming back to calculating the mount reaction force. Since, we know that the mount is defined to be a non-perfect elastic body, we can find the reaction forces by simply using the Hooke's law in matrix form. The local stiffness matrix is given by:

$$[K]_i' = \begin{bmatrix} k_x'(1+i\eta_x') & 0 & 0 \\ & k_y'(1 \quad i\eta_y') & \\ 0 & 0 & k_z'(1+i\eta_z') \end{bmatrix}$$

Hence, the mount force is given by:

$$\mathrm{f_i} = \begin{bmatrix} k_{\mathrm{xi}}(1+\iota\eta_{\mathrm{xi}})\Delta x \\ k_{\mathrm{yi}}(1+\iota\eta_{\mathrm{yi}})\Delta y \\ k_{\mathrm{zi}}(1+\iota\eta_{\mathrm{zi}})\Delta z \end{bmatrix} = \begin{bmatrix} f_{\mathrm{xi}} \\ f_{\mathrm{yi}} \\ f_{\mathrm{zi}} \end{bmatrix}$$

The mount force that we calculate is complex in nature. In order to calculate the objective function we only utilize the real part.

On using the values calculated above in the objective function formula defined earlier we get the value of the objective function.

We use the sampling approach defined earlier to find the maximum value of the objective function.

We created a MatLab function for this model, which outputs the objective function value for a given configuration.

Mathematics behind Bayesian Optimization

When faced with an expensive-to-evaluate black-box function, whose analytical expression as well as derivatives are unknown, Bayesian optimization is used. In such a case it becomes important to minimize the number of samples drawn from the black box function. Bayesian Optimization techniques attempt to find the global optimum in the minimum possible number of steps.

Bayesian optimization incorporates prior belief about the function and updates the prior with samples drawn from the function to get a posterior that better approximates it.

Bayesian optimization uses an acquisition function that directs sampling to areas where an improvement to the current best observation is likely.

The Bayesian optimization procedure is as follows. For a given objective function f & t=1,2,3… repeat:

1. Find the next sampling point xt by optimizing the acquisition function over the Gaussian Process.
2. Obtain a possibly noisy sample $yt = f(xt) + \varepsilon t$ from the objective function f.
3. Add the sample to previous samples $D1 : t = D1 : t - 1, (xt, yt)$.

Expected improvement is defined as:

$$\mathrm{EI}(\mathbf{x}) = \mathbb{E}\max(f(\mathbf{x}) - f(\mathbf{x}^+), 0)$$

Where $f(x^+)$is the value of the best sample so far and x^+ is the location of that sample. The expected improvement can be evaluated analytically under the GP model:

$$\mathrm{EI}(\mathbf{x}) = \begin{cases} (\mu(\mathbf{x}) - f(\mathbf{x}^+) - \xi)\Phi(Z) + \sigma(\mathbf{x})\phi(Z) & \text{if } \sigma(\mathbf{x}) > 0 \\ 0 & \text{if } \sigma(\mathbf{x}) = 0 \end{cases}$$

Where $\mu(x)$ and $\sigma(x)$ are the mean and the standard deviation of the GP posterior predictive at x, respectively. Φ and ϕ are the CDF and PDF of the standard normal distribution, respectively. Parameter ξ determines the amount of exploration during optimization and higher ξ values lead to more exploration.

Implementation of Bayesian Optimization

In order to simplify the implementation of Bayesian optimization, during our research we found out about an open-source python package called bayesian-optimization. This helped us implement Bayesian optimization on our model using the python programming language.

Challenge faced and solution:
It is very easy to notice that we are creating our model in MATLAB and the optimization in python. Then how to use both simultaneously? This issue was solved by using the Matlab-engine for python provided by Mathworks for running .m files in python.