SIMULADOR DE ALTO DESEMPENHO DO MODELO INTEGRADO DO TELESCÓPIO GIGANTE DE MAGALHÃES

Felipe Brandão Ippolito [1]; Tiago Sanches da Silva [2]

[1] Aluno de Iniciação Científica da Escola de Engenharia Mauá (EEM/CEUN-IMT);
[2] Professor da Escola de Engenharia Mauá (EEM/CEUN-IMT).

Resumo. *O simulador do Telescópio Gigante de Magalhães é responsável por simular diversos subsistemas como ópticos, mecânicos e estruturais do telescópio. A simulação requer um alto poder computacional devido sua alta complexidade e dimensão. Esse trabalho propõe resolver dois problemas que até então impediam a simulação do modelo integrado do GMT. O primeiro, é a impossibilidade de integrar subsistemas desenvolvidos no Simulink com o simulador de modelo integrado em python, a solução desse vem por meio da criação de duas ferramentas pyCreate e pySimulate que permitem a integração de forma automática. Outro problema endereçado neste trabalho é o interfaceamento entre os subsistemas óptico e estrutural do telescópio, utilizando técnicas de otimização matricial. Por meio da decomposição de valor singular foi possível reduzir a informação da translação, rotação e flexão das superfícies dos espelhos primários, possibilitando a transmissão das informações das superfícies dos espelhos para o subsistema óptico com alto desempenho. Esse trabalho possibilitou a simulação do modelo integrado do GMT em malha fechada com alta eficiência computacional e em um ambiente escalável.*

Introdução

O Telescópio Gigante de Magalhães (GMT) pertence a nova geração de Telescópios Extremamente Grandes, projetados para fornecer clareza e sensibilidade sem precedentes para a observação de fenômenos astronômicos. Uma das principais ferramentas de análise dos requisitos técnicos dos diversos subsistemas é o modelo integrado do GMT que combina em uma única estrutura computacional os modelos óptico, estrutural, térmico e de controle do GMT.

Diversos subsistemas do telescópio são desenvolvidos no ambiente Simulink, uma vez que essa é uma plataforma consolidada e de fácil uso para o desenvolvimento de subsistemas de controle e mecânicos. Porém todo o subsistema óptico (CEO — *CUDA Engine Optics)* do telescópio foi desenvolvido em C++ com a utilização da programação CUDA (COOK, 2012), a qual permite a aceleração em *hardware* com a utilização de placa de vídeo (GPU, do inglês g*raphics processing unit*). A simulação do modelo integrado do GMT em malha fechada só é possível com todos os subsistemas mecânicos, ópticos e estruturais comunicando entre si. Esse trabalho tem o objetivo de realizar a integração dos diversos subsistemas em um ambiente escalável e com alto desempenho, e para isso, foram desenvolvidas duas ferramentas que possibilitam a integração dos subsistemas Simulink no simulador de modelo integrado do GMT (desenvolvido em python). Esse trabalho também introduz o conceito dos modos de flexão, essa é uma mudança de base utilizando a decomposição de valor singular que permite representar a superfície dos espelhos primários de maneira mais eficiente.

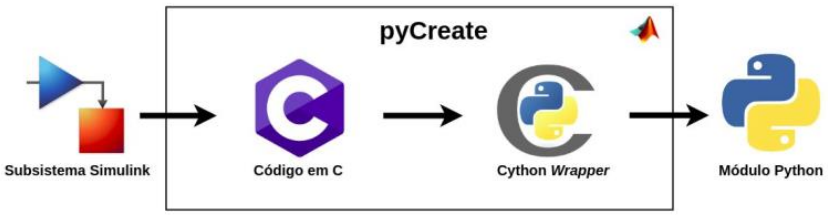
Material e Métodos

pyCreate

O pyCreate é uma ferramenta desenvolvida em MATLAB® que permite a integração dos modelos gerados no Simulink com o simulador de modelo integrado. O pyCreate gera módulos python por meio da compilação de subsistemas do Simulink na linguagem de

programação C (KERNIGHAN e RITCHIE, 1978), códigos de integração gerados em cython (DALCIN et al., 2011) permitem a importação e utilização das bibliotecas compiladas em C dentro do ambiente python — representado pela Figura 1.

Figura 1- Diagrama pyCreate



A compilação do subsistema Simulink é feita através da ferramenta proprietária *rtwbuild* (MATHWORKS, 2020), qual gera código C a partir de um modelo selecionado. Para isso é necessário que as variáveis deste modelo estejam pré-carregadas no ambiente do MATLAB®. O subsistema compilado tem que ser um sistema com amostragem uniforme e não pode conter blocos que utilizam tempo contínuo. Com isso preestabelecido, o pyCreate é capaz de trabalhar com todos os blocos que podem ser compilados em C, incluindo não linearidades e tempos de amostragem diferentes dentro do subsistema.

A partir dos arquivos produzidos para a biblioteca C é gerado um arquivo denominado de *wrapper*, esse faz o envelopamento do código C para que o subsistema possa ser executado no python. O arquivo *header* (arquivo com extensão .h gerado na compilação do código C) contém as informações necessárias para a construção do *wrapper* em cython, o pyCreate faz uma varredura do texto do *header* procurando informações importantes, como número total de entradas e saídas do subsistema, nome das classes e métodos internos, dimensões e nomes de cada entrada e saída. Dadas as informações coletadas o pyCreate gera o *wrapper* específico para o subsistema com base em suas propriedades. O *wrapper* possui duas funções importantes, permitir a utilização do código C dentro do ambiente python e padronizar as entradas, saídas e métodos do subsistema para que possam ser utilizadas dentro do simulador de modelo integrado do GMT.

Os *wrapper* gerados contém quatro métodos que permitem a integração com o simulador integrado: *init( )* — método de inicialização do subsistema — permite o carregamento de variáveis e arquivos externos, as dimensões das entradas e saídas são definidas e a função de inicialização do código C é inicializada. O método *update( )* — método de atualização — recebe uma entrada e faz a atualização de seus estados internos, o *output( )* retorna as saídas do sistema e o *terminate( )* realiza quaisquer pós-processamento necessário e executa a função de término em C.

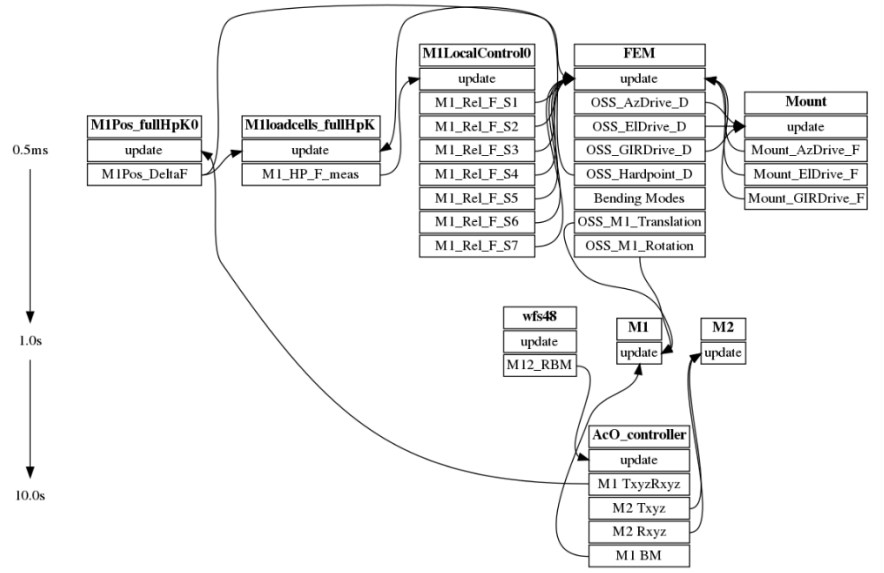
pySimulate

O pyCreate é uma ferramenta que permite a compilação e inclusão de novos subsistemas (*drivers)* no simulador de modelo integrado, porém a integração dos *drivers* dentro do simulador e a construção da simulação ainda é responsabilidade do usuário. Toda a preparação da simulação é realizada através dos arquivos de configurações — o arquivo *dos.yaml*, é responsável pelo gerenciamento da simulação por completo, incluindo a taxa de atualização da simulação e de cada *driver*, suas entradas, saídas, dimensões e conexões com outros *drivers*. Além disso, cada subsistema também possui seu arquivo de configuração específico, esse pode incluir dependências externas e configurações pré-definidas.

Com o aumento da complexidade das simulações devido a inclusão de novos subsistemas o gerenciamento de todos os *drivers*, conexões e taxas de atualizações se torna uma tarefa complexa para o usuário — a Figura 2 mostra o diagrama de bloco de um cenário de simulação do GMT. Realizar modificações nos subsistemas do Simulink é uma tarefa

árdua, porque, é necessário primeiramente realizar as modificações no Simulink, utilizar o pyCreate para compilar o novo subsistema e por fim incluir as alterações nos arquivos de configuração. O pySimulate é um *framework* desenvolvido neste trabalho que torna automático o processo de gerar simulações para o simulador de modelo integrado por meio do arquivo Simulink — Figura 3 — e permite que o usuário realize testes e simulações dentro do Simulink de maneira mais eficiente.

Figura 2- Diagrama da simulação



O *framework* desenvolvido inclui uma biblioteca de blocos Simulink específica do simulador do modelo integrado do GMT, esses blocos podem ser divididos em dois grupos com funções diferentes. O primeiro grupo são blocos do modelo estrutural de elementos finitos do telescópio (FEM, do inglês *finit element model*). O segundo grupo de blocos são subsistemas auxiliares para a função principal do pySimulate: compilar um cenário de simulação do Simulink para o simulador de modelo integrado de forma automática.

A biblioteca FEM é um conjunto de blocos do Simulink, com diversas versões do FEM, esses blocos contém o modelo de estado de espaços do telescópio [1] — sendo esse o subsistema mais pesado computacionalmente, o que torna simulações no Simulink muito demoradas, logo, inviáveis. A versão 4 (versão consolidada) do FEM é uma representação de espaço de estados com as seguintes dimensões [1]:

$$\begin{aligned} \dot{x} &= Ax + Bu \\ y &= Cx + Du \end{aligned} \qquad \begin{aligned} \dim[A(\cdot)] &= 11000 \times 11000 \\ \dim[B(\cdot)] &= 11000 \times 8021 \\ \dim[C(\cdot)] &= 16133 \times 11000 \\ \dim[D(\cdot)] &= 0 \end{aligned} \qquad (1)$$

A biblioteca de blocos do FEM permite reduzir significativamente o tempo de processamento das simulações dentro do Simulink com redução das matrizes B e C do sistema de espaço de estados para apenas as entradas e saídas necessárias para o atual cenário de simulação — as matrizes B_R e C_R representam as matrizes reduzidas — equação [2], suas dimensões reduzidas dependem das entradas (dimensão j) e saídas (dimensão k) do modelo de espaço de estados selecionadas para o determinado cenário de simulação.

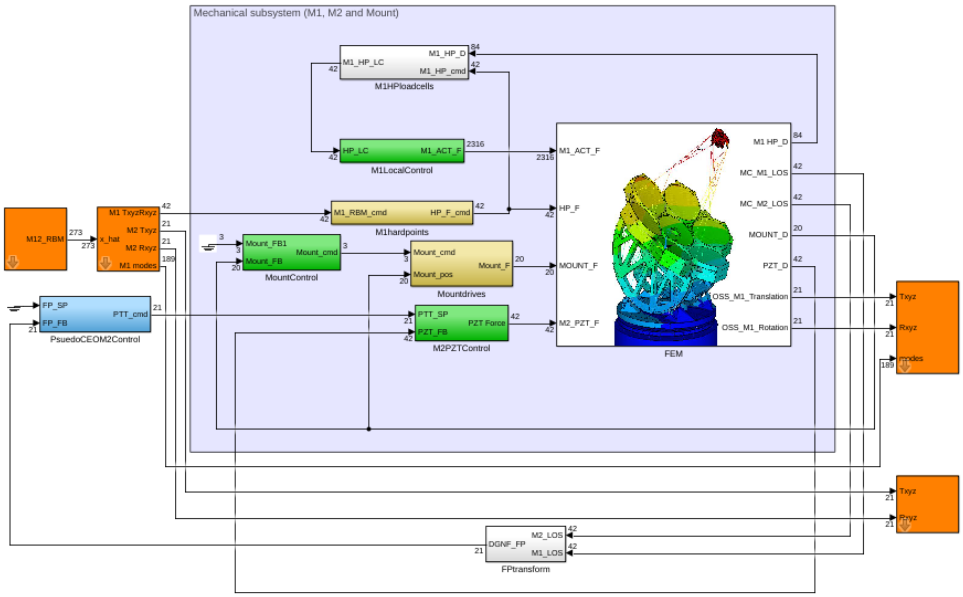
$$\begin{aligned} \dot{x} &= Ax + B_R u \\ y &= C_R x + Du \end{aligned} \qquad \begin{aligned} \dim[A(\cdot)] &= 11000 \times 11000 \\ \dim[B_R(\cdot)] &= 11000 \times j \\ \dim[C_R(\cdot)] &= k \times 11000 \\ \dim[D(\cdot)] &= 0 \end{aligned} \qquad (2)$$

Uma vez com as matrizes de espaço de estados reduzidas torna-se necessário reconectar as entradas e saídas internas do bloco FEM, para isso foi necessário a criação da biblioteca Simulink. Blocos Simulink possuem funções específicas para certos eventos, essas funções são denominadas de *callbacks*. Na biblioteca FEM foi utilizado o *callback* de inicialização, isso é, antes da simulação começar essa função é executada. A função de inicialização do FEM faz a leitura dos parâmetros do bloco contendo informações das matrizes do espaço de estados com os nomes e dimensões das saídas e entradas do sistema (*Input Table* e *Output Table)*. Durante a inicialização também é realizada a leitura das entradas e saídas conectadas ao bloco FEM, deste modo somente as entradas e saídas utilizadas são conectadas durante a inicialização de forma dinâmica.

Esse processo de reconstrução das conexões internas do FEM é realizado para cada nova simulação, evitando assim o conflito de conexões das entradas e saídas com as dimensões das matrizes reduzidas B_R e C_R. A Figura 3 ilustra um cenário de simulação com entradas com dimensão 2420 e saídas com 272, isso representa uma redução do número de entradas e saídas respectivamente de 85% e 96% em relação ao modelo original — equação [2].

Além de permitir a simulação mais eficiente dentro do Simulink, outra funcionalidade do pySimulate é gerar simulações para o simulador de modelo integrado a partir de um modelo Simulink (Figura 3). O pySimulate além da biblioteca de blocos FEM possuí uma biblioteca de blocos virtuais, esses blocos não realizam operações durante as simulações no Simulink, porém eles contêm parâmetros que podem ser definidos no Simulink e são utilizados para a construção da simulação do modelo integrado. Deste modo é possível construir simulações do modelo integrado dentro do ambiente Simulink com a inclusão da biblioteca de blocos do CEO é possível integrar módulos CEO com o resto dos subsistemas Simulink e executa-los dentro do simulador de modelo integrado.

Figura 3- Modelo Simulink



A simulação de modelo integrado é gerada pelo pySimulate através da varredura do arquivo Simulink (*.slx*) em que as informações das conexões dos subsistemas, suas dimensões e parâmetros são coletadas, a partir destas informações o pySimulate prepare o arquivo *dos.yaml*, reduz as matrizes de espaço de estados do FEM e gera os arquivos de configuração para cada subsistema. Utilizando o pyCreate também são compilados todos os subsistemas Simulink para o uso no simulador integrado. A execução do pySimulate se dá através de um comando no MATLAB®, no qual o usuário especifica alguns parâmetros de simulação, como

taxa de atualização, qual arquivo *.slx* será compilado e onde os arquivos de simulação serão criados; uma vez com o pySimulate executado toda a simulação está pronta para ser executada dentro do simulador de modelo integrado.

Modos de Flexão

Os modos de flexão são formas de representar as superfícies dos espelhos sem a necessidade de informar as posições x, y e z de todos os nós. Os espelhos primários do GMT no modo de baixa resolução são formados por 742 nós nos espelhos externos e 675 nós no segmento central; para o modo de alta resolução esses valores aumentam para 27685 e 25794 nós respectivamente. Isso torna a simulação, incluindo as superfícies dos espelhos, computacionalmente inviável. Sabendo que, para realizar uma simulação realista o telescópio deve ser simulado em uma frequência de 2kHz, isso é, para simular 1 segundo é necessário atualizar as posições x, y e z dos 191904 nós dos espelhos primários 2000 vezes, resultando 383,8 milhões de atualizações por segundo, somente para as superfícies dos espelhos primários. Como muitos cenários de simulações devem simular horas ou até dias da dinâmica do GMT o tempo para realizar as simulações se torna muito grande e, portanto, proibitivo.

A representação utilizando os modos de flexão permite reduzir a informação da superfície dos espelhos para o número de modos de flexão desejado p [19]. A compressão da informação da posição das superfícies dos espelhos é feita utilizando uma mudança de base através da decomposição em valor singular da matriz Ψ permite representar as superfícies dos espelhos utilizando os modos de flexão ao invés do descolamento em x, y e z.

A matriz Ψ [4] é a matriz de ganho DC do sistema de espaço de estados [1] da estrutura do telescópio calculada apenas para as entradas dos atuadores dos espelhos primários e a saída do deslocamento em z das superfícies destes espelhos. É possível obter a matriz de ganho dc do sistema de espaço de estados Ψ_{SS} calculando o ganho do sistema em regime permanente:

$$\Psi_{SS} = D - C(A)^{-1} B \qquad (3)$$

Para os modos de flexão a matriz de ganho em regime permanente, Ψ, é calculada utilizando apenas as entradas referente às forças dos atuadores dos espelhos primários e às saídas referentes ao descolamento em z dos nós dos espelhos

$$\Psi = C_D(A)^{-1} B_F \qquad (4)$$

sendo a matriz C_D a matriz C com apenas as linhas referente ao deslocamento em z dos espelhos primários e a matriz B_F é a matriz B somente com as colunas referente às forças aplicadas pelos atuadores dos espelhos primários. A matriz de ganho transforma as forças aplicadas no espelho, F_z, no módulo do deslocamento perpendicular à superfície, z

$$\Psi F_Z = z \qquad (5)$$

os modos de flexão são obtidos a partir da decomposição de valor singular (svd, do inglês *singular value decomposition*) (ANDERSEN, T, 2011) da matriz Ψ,

$$svd(\Psi) = U\Sigma V^{-1} = \Psi \qquad (6)$$

sendo as matrizes U e V da decomposição de valores singular, matrizes unitárias. Os modos de flexão (B_M, do inglês *Bending Modes*) é definido por:

$$B_M \triangleq U^T z \qquad (7)$$

As equações [5] e [6] podem ser reescritas da seguinte forma:

$$U\Sigma V^{-1} F_Z = z \qquad (8)$$

Como $U^T U = I$, multiplicando a equação [8] por U^T e através da definição dos modos de torção da equação [7], obtém-se [9]:

$$\Sigma V^{-1} F_Z = U^T z \triangleq B_M \tag{9}$$

A partir da equação [9] é possível escrever as forças em função dos modos de torção [10]:

$$\begin{aligned} \Sigma V^{-1} F_Z &= B_M \\ V^{-1} F_Z &= \Sigma^{-1} B_M \\ F_Z &= (V^{-1})^{-1} \Sigma^{-1} B_M \\ F_Z &= V \Sigma^{-1} B_M \end{aligned} \tag{10}$$

O deslocamento em função dos modos de torção é dado diretamente pela equação [11]:

$$z = U B_M \tag{11}$$

A redução de dimensionalidade é realizada pela inclusão dos modos de flexão na matriz C, isso permite a obtenção dos modos de flexão sem a necessidade de calcular a posição de todos os nós dos espelhos:

$$y_z^{m\times 1} = C_D^{m\times r} x^{r\times 1} \tag{12}$$

A matriz de ganho do sistema é definida pelo ganho das forças nos atuadores para o deslocamento da superfície do espelho em z, sua dimensão é:

$$\dim[\Psi(\cdot)] = m \times n \tag{13}$$

Sendo m e n o número de nós e o número de atuadores dos espelhos primários respectivamente. Com a decomposição das matrizes Ψ, obtém-se as seguintes dimensões para as matrizes U, Σ e V:

$$svd(\Psi^{m\times n}) = U^{m\times p} \Sigma^{p\times p} V^{p\times n} \tag{14}$$

sendo p é o posto da matriz Ψ.

É possível reduzir o sistema de estado de espaços com a inclusão dos modos de flexão na matriz C.

$$\begin{aligned} C_D^{m\times r} x^{r\times 1} &= z^{m\times 1} \\ (U^T)^{p\times m} C_D^{m\times r} x^{r\times 1} &= (U^T)^{p\times m} z^{m\times 1} \end{aligned} \tag{15}$$

Pela definição dos modos de flexão [7]:

$$U^T C_D x = B_M \tag{16}$$

Desse modo é possível reescrever o sistema de espaço de estados com as novas saídas com os modos de flexão — equação [17]:

$$\begin{aligned} \dot{x} &= Ax + B_F u \\ y_{B_M} &= U^T C_D x \end{aligned} \tag{17}$$

$$C_{B_M} \triangleq U^T C_D \tag{18}$$

Onde C_{B_M} — definido na equação [18] — é a matriz de saída do subsistema de espaço de estados em que saídas y_{B_M} são os modos de flexão.

$$\begin{aligned} \dot{x} &= Ax + B_F u \\ y_{B_M} &= C_{B_M} x \end{aligned} \tag{19}$$

É importante notar a redução das dimensões de C_{B_M} e y_{B_M} em relação à C_D e y_z:

$$\dim[C_D(\cdot)] = m \times r \qquad \dim[C_{B_M}(\cdot)] = p \times r$$
$$\dim[y_z(\cdot)] = m \times 1 \qquad \dim[y_{B_M}(\cdot)] = p \times 1 \tag{20}$$

em que p é o número de atuadores dos espelhos primários $p = 1144$, r é o número de estados $r = 11000$ e m, o número de nós dos segmentos primários $m = 191904$.

O novo sistema de espaço de estados:

$$\dot{x} = Ax + Bu$$
$$y' = C'x \tag{21}$$

sendo C' e y':

$$C' = \begin{bmatrix} C_O \\ C_{B_M} \end{bmatrix};\ y' = \begin{bmatrix} y_O \\ y_{B_M} \end{bmatrix}; \tag{22}$$

sendo, C_O a matriz de saída do sistema de espaço da equação [1] sem as linhas referentes à informação da superfície dos espelhos primários e y_O referente às saídas de C_O. Através de simulações, mostrou-se possível reduzir o número de modos de flexões para 27 modos para cada espelho, reduzindo ainda mais o valor de C_{B_M} e y_{B_M} para $189 \times r$ e 189×1 respectivamente. Para a simulação do telescópio na frequência de 2kHz, utilizando os modos de flexões, são necessárias 378 mil atualizações para cada segundo de simulação para a representação das superfícies dos espelhos primários. Esse é um ganho de mais mil vezes comparado com a utilização das posições x,y e z de cada nó (383,8 milhões de atualizações para cada segundo).

A Figura 4 ilustra os primeiros 20 modos de flexão nos espelhos externos. A combinação desses modos de flexão é capaz de representar a superfície dos espelhos primários durante a simulação. A Figura 5 ilustra a superfície do espelho primário à esquerda e a combinação dos diferentes modos de flexão e suas magnitudes que condizem com essa superfície.

Figura 4- Modos de flexão

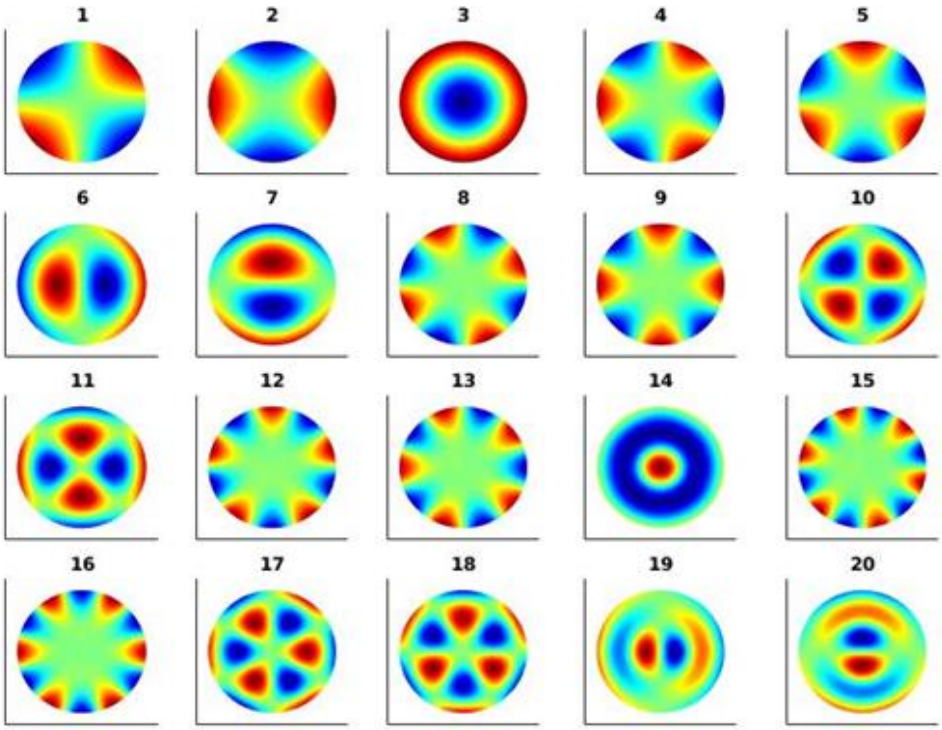
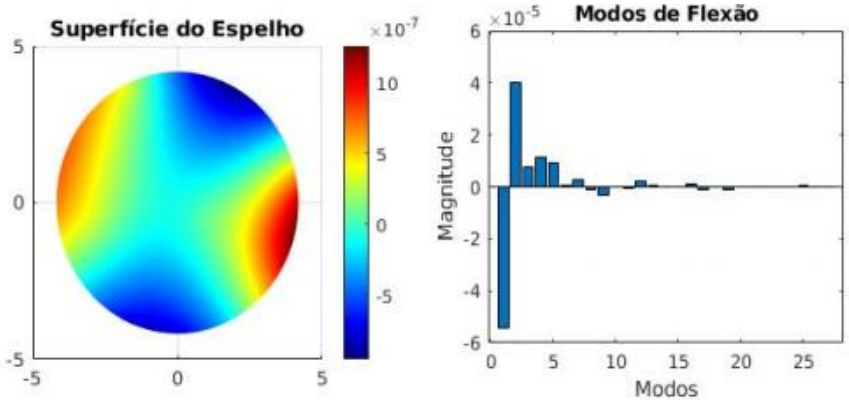


Figura 5 - Representação da superfície do espelha através dos modos de flexão



Resultados e Discussão

As ferramentas desenvolvidas neste trabalho possibilitaram simulações mais realistas do modelo integrado do GMT através da integração de modelos desenvolvidos no Simulink com bibliotecas desenvolvidas em python, C e C++ (e.g. simulação óptica — CEO). Junto com a otimização da representação das superfícies dos espelhos foi possível realizar simulações em malha fechada com a inclusão da dinâmica dos espelhos, permitindo estudos mais aprofundados de como essa dinâmica afeta o erro de frente de onda no sensor *Shack-Hartmann*.

As Figura 6, Figura 7 e Figura 8 ilustram o resultado da simulação em malha fechada do modelo integrado do telescópio; a Figura 6 ilustra — em azul (AcO) — o *setpoint* do movimento de corpo rígido dos espelhos primários (translação e rotação em x, y e z) junto com o movimento dos espelhos, em rosa (FEM). Além do movimento de corpo rígido é possível visualizar as flexões nos espelhos através dos modos de flexão ao longo do tempo — Figura 7.

Figura 6 - Simulação: Movimento de corpo rígido

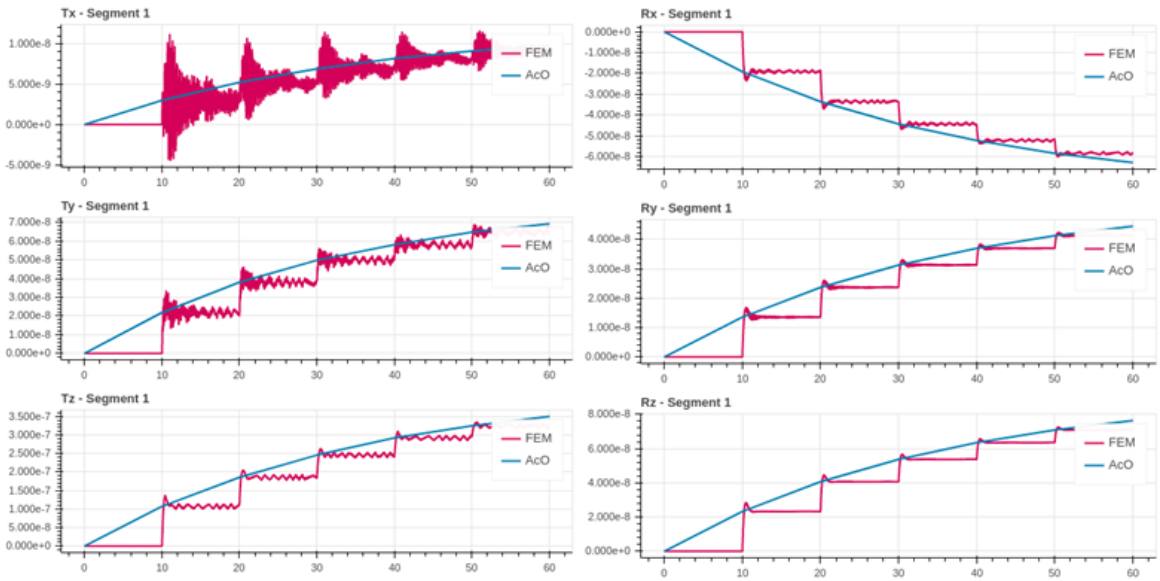
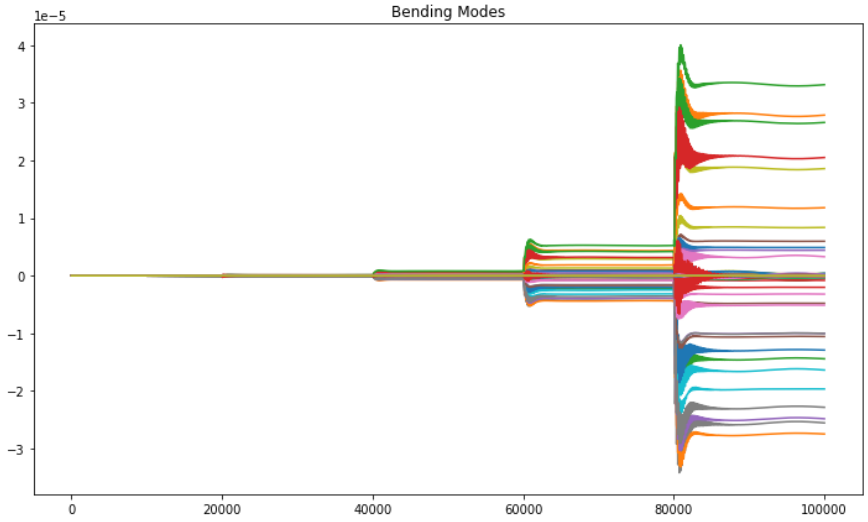
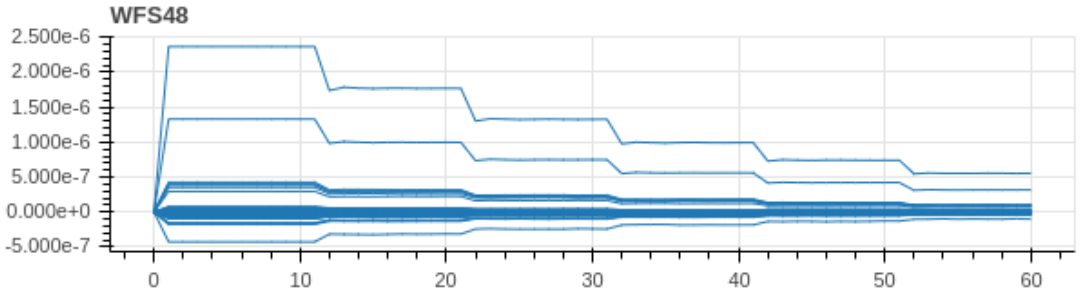


Figura 7 - Simulação: Modos de flexão



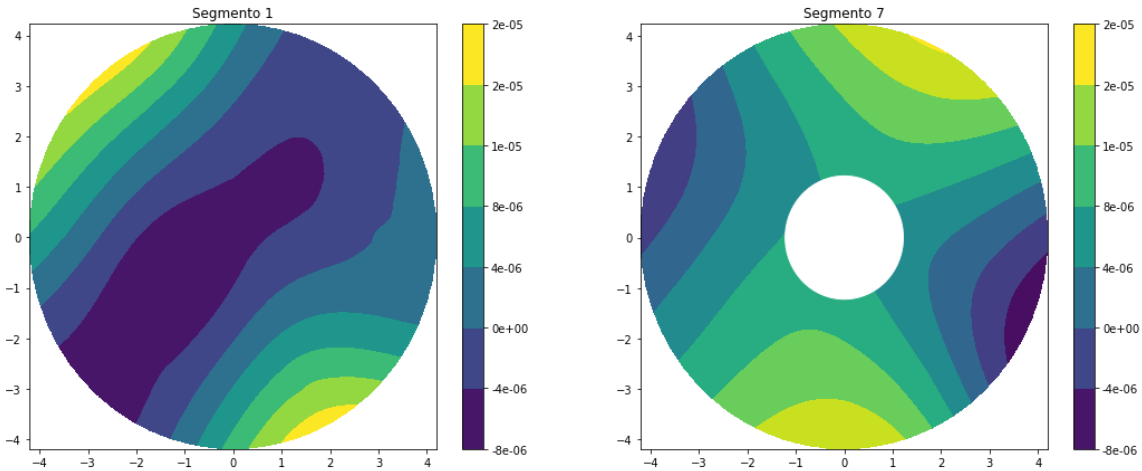
As movimentações dos espelhos representados nas figuras anterior é em resposta à uma carga inicial aplicada nos espelhos, como a simulação está em malha fechada, os controlados movimentam o espelho a fim de minimizar o erro de frente de onda. A Figura 8 ilustra o erro de frente de onda ao longo do tempo, nota-se que o erro converge para zero.

Figura 8 - Simulação: Erro de frente de onda



Dado os modos de flexão ao longo do tempo durante a simulação (Figura 7) é possível reconstruir as superfícies dos espelhos [11] para uma melhor entendimento de como os espelhos reagem à um determinado erro de frente de onda.

Figura 9 - Simulação: Superfícies dos espelhos



Conclusões

Esse trabalho permitiu uma mudança significativa no simulador do Telescópio Gigante de Magalhães, em que o MATLAB e Simulink deixaram de ser a principal plataforma de simulação do modelo integrado; essas são agora plataformas de desenvolvimento e testes. Simulações podem ser realizada dentro do ambiente python com integração de outras linguagens de programação, e.g. C, C++, Rust; permitindo, não somente simulações mais rápidas em ambientes gratuitos e *open-source*, mas também torna o simulador do GMT uma plataforma escalável, uma vez que é possível sua implementação utilizando computação em nuvem com recursos mais avançados e maior poder de processamento.

Referências Bibliográficas

ANDERSEN, T.; ENMARK, A. (2011) *Integrated Modeling of Telescopes*. Springer. ISBN 9781461401490

DALCIN, L.; BRADSHAW, R.; SMITH, K.; CITRO, C. (2011) BEHNEL, S.; SELJEBOTN, D. *Cython: The best of both worlds*. Computing in Science & Engineering, IEEE Computer Society, Los Alamitos, CA, USA, v. 13, n. 02, p. 31–39.

KERNIGHAN, B. W.; RITCHIE, D. M. (1978) *The C Programming Language*. USA: Prentice-Hall, Inc. ISBN 0131101633.

MATHWORKS. rtwbuild. (2020). Disponível em:<https://www.mathworks.com/help/rtw/ref-/rtwbuild.html>. Acesso em: 14 nov. 2020.

COOK, S. (2012*) CUDA Programming: A Developer's Guide to Parallel Computing with GPUs*: Morgan Kaufmann Publishers Inc. IBSN 9780124159334