

Modelagem de sistemas físicos, lógicos e cyber-físicos em OpenModelica

Gabriel de Vargas Coelho (20200400)

João Pedro Lopes de Camargo (21100134)

Florianópolis
13 de outubro de 2023

Sumário

Introdução.....	3
1. Modelo de controle de temperatura de um ambiente com aquecedor.....	4
2. Modelo de veículo e motorista.....	7
3. Modelo de oscilações de glicose em leveduras.....	12
4. Modelo de impressão 2D.....	14
Considerações finais.....	18

Introdução

O presente trabalho tem como objetivo apresentar 4 exemplos de modelagem física atrelados a diferentes engenharias, utilizando a ferramenta OMEdit, um editor gráfico e open source de Modelica. Nas seções adiantes, os exemplos são descritos e são apresentados modelagens – e como produzi-las – para simulá-los.

Vale ressaltar que esse trabalho faz parte de um projeto maior, em que pretende-se fazer o desenvolvimento de modelagens lógicas para os 4 exemplos apresentados, e integrar os modelos físicos e lógicos para obter sistemas cyber-físicos.

1. Modelo de controle de temperatura de um ambiente com aquecedor

Este primeiro exemplo se trata do controle de temperatura de um ambiente que é resfriado naturalmente, e que pode aquecer por meio de um aquecedor, e foi previamente descrito na apostila da disciplina de Modelagem e Simulação ofertada pela UFSC, e escrita pelo Prof. Dr. Eng. Rafael Luiz Cancian. Para a modelagem física desse sistema, denotamos T como a temperatura do ambiente, e a variação dessa temperatura ao longo do tempo é definida pela equação diferencial: $\frac{dT}{dt} = -aT + Tr + \mu Ta$, em que a representa o fator de resfriamento natural do ambiente, Tr expressa a influência externa para a temperatura do ambiente, Ta expressa a influência do aquecedor no ambiente, e $\mu \in \{0, 1\}$ informa se o aquecedor está ligado ou não.

Para realizar essa modelagem em OpenModelica, acessamos o OMEdit (Editor de OpenModelica) e criamos um novo modelo através da barra de opções superior em *File > New > New Modelica Class*, dê um nome para a classe, e confirme. Com o espaço de modelagem criado, usaremos os blocos de *realExpression* e *integrator* – que podem ser encontrados pelo ferramenta de busca *Libraries Browser*, ou dentro da seção *Libraries* em *Modelica > Blocks > Sources* e *Modelica > Blocks > Continuous* respectivamente – e conectamos os blocos seguindo a imagem 1.0.

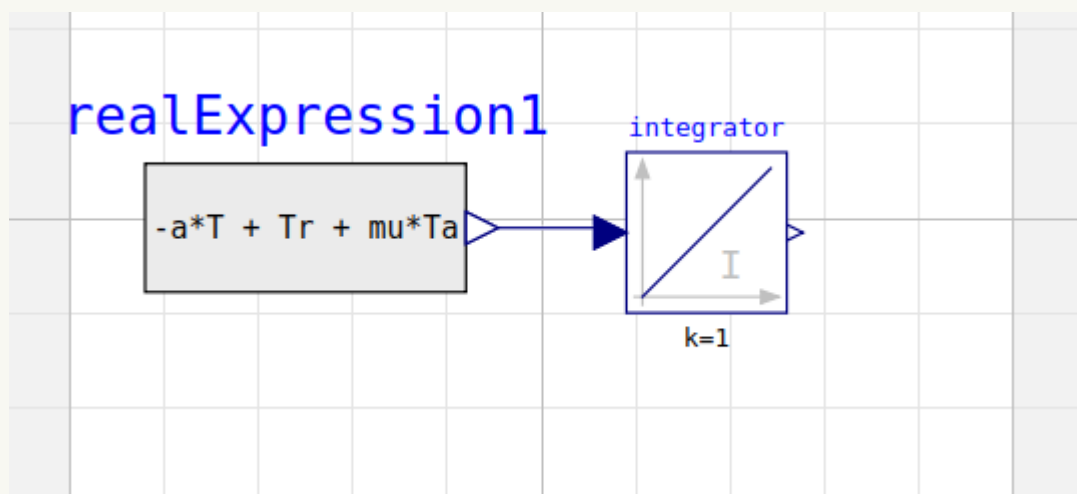
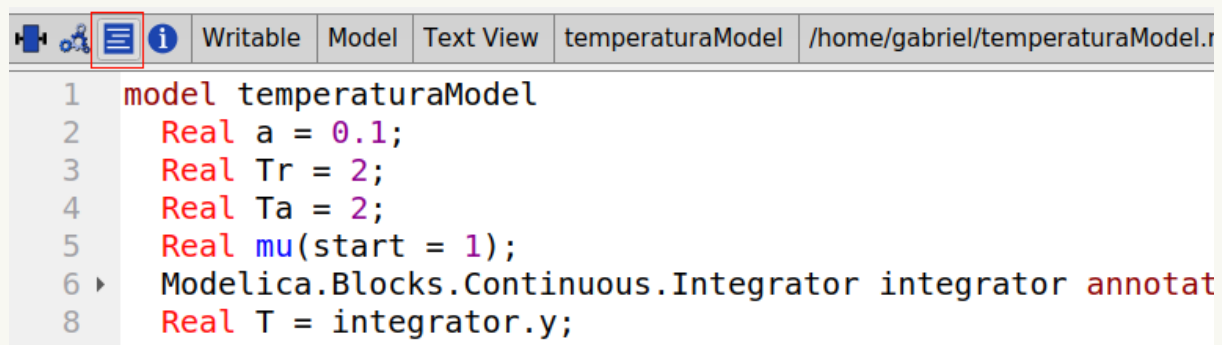


Imagem 1.0: Modelo físico do controle de temperatura de um ambiente

Para inserir a expressão no bloco *realExpression* basta clicar com o botão esquerdo do *mouse* duas vezes sobre o bloco, e no campo de input *y* e adicionar a expressão. Só isso não é suficiente para que a simulação possa ser executada, precisamos ainda definir o que são esses termos presentes na expressão, e para isso utilizamos a aba *Text View* e modificamos o texto adicionando algumas linhas, como pode ser observado na imagem 1.1. Note que a definição “*Real T = integrator.y*” precisa necessariamente estar abaixo da definição do *integrator*, pois é ela que permite sabermos qual é a temperatura do ambiente em dado instante.



```

1  model temperaturaModel
2    Real a = 0.1;
3    Real Tr = 2;
4    Real Ta = 2;
5    Real mu(start = 1);
6    Modelica.Blocks.Continuous.Integrator integrator annotated
8    Real T = integrator.y;

```

Imagem 1.1: Variáveis do modelo

Antes de executarmos a simulação iremos configurá-la antes, e para isso acessamos na barra de opções superior *Simulation > Simulation Setup* e mudamos o tempo de parada (*Stop Time*) para 100 segundos. O resultado da simulação pode ser visto no gráfico gerado ao final da simulação, e ao selecionar a variável *T* para visualização.

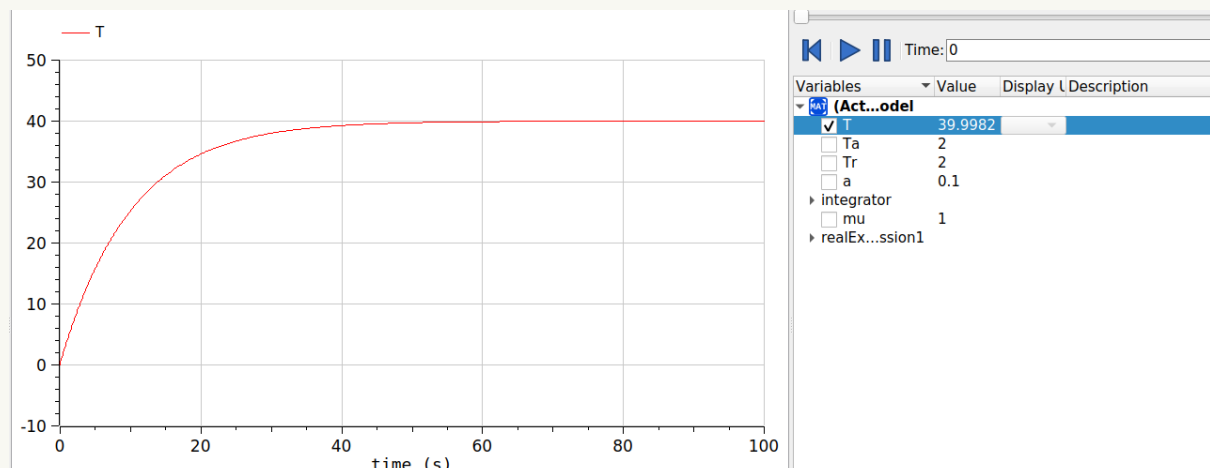


Imagem 1.2: Resultado da simulação

Podemos ainda adicionar um acionador do aquecedor, que desliga ele após 50 segundos, para visualizarmos como o ambiente se comporta nessa situação.

Para isso, basta adicionarmos uma pequena verificação ao código, acompanhe a imagem 1.3.

```
11 equation
12 ▶ connect(realExpression1.y, integrator.u) annotation( ...);
14 ▶ annotation( ...);
17 algorithm
18   if time >= 50 then
19     mu := 0;
20   end if;
21 end temperaturaModel;
```

Imagem 1.3: Algoritmo para desligar aquecedor após 50 segundos

Ao executarmos o modelo após essa alteração, obtemos o gráfico da imagem 1.4.

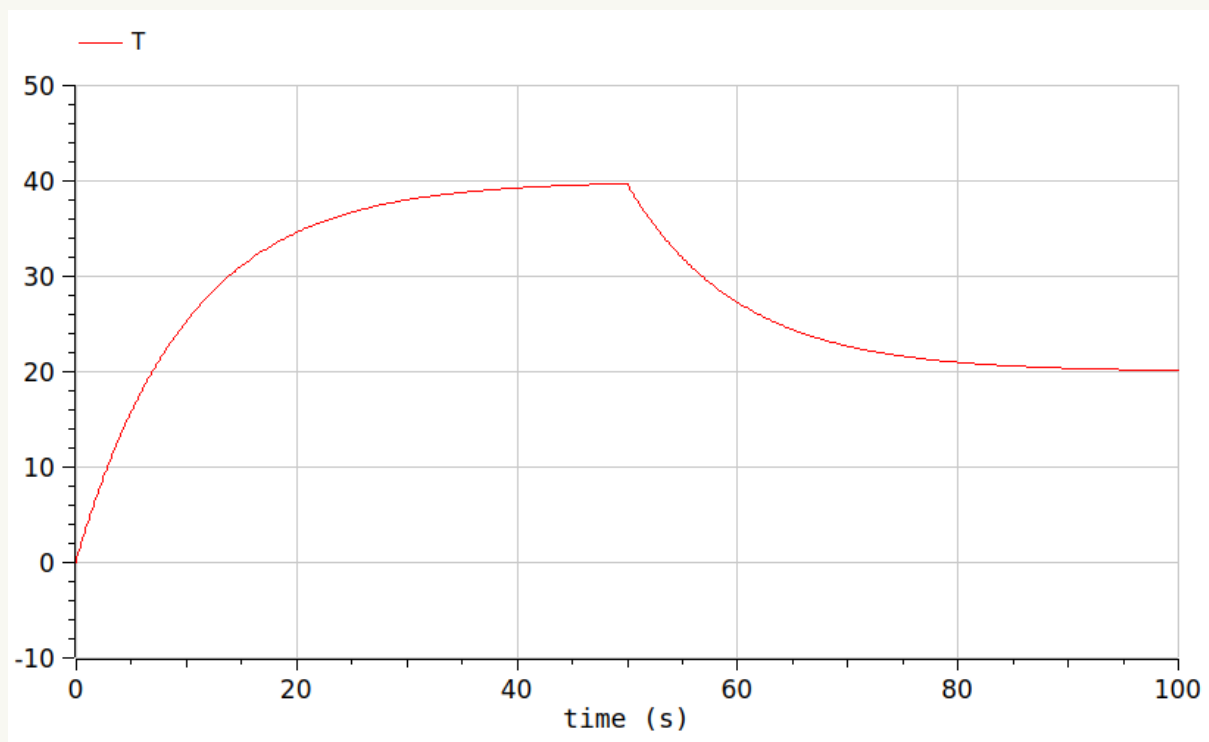


Imagem 1.4: Resultado da simulação com estímulo ao aquecedor

2. Modelo de veículo e motorista

Essa modelagem se trata de um veículo que é controlado por um motorista que possui duas formas de interagir com o carro, acelerando e mudando sua direção, e é inspirada pela modelagem presente na apostila da disciplina. A modelagem física do modelo se dá pelo veículo, que possui duas entradas: a aceleração, e o ângulo do veículo, e uma saída que é sua posição, que pode ser decomposta em posição no eixo x e posição no eixo y de um plano cartesiano. Sendo s um estado com 4 componentes – posição x e y, velocidade e mudança angular –, a variação de s é dada pelo seguinte sistema de equações diferenciais:

$$\frac{ds_1}{dt} = s_3(t) * \sin(s_4(t))$$

$$\frac{ds_2}{dt} = s_3(t) * \cos(s_4(t))$$

$$\frac{ds_3}{dt} = acc(t)$$

$$\frac{ds_4}{dt} = ang(t),$$

em que s_1 e s_2 representam respectivamente a posição do veículo no eixo x, e no eixo y de um plano cartesiano, s_3 expressa a velocidade do veículo, s_4 representa a mudança angular, e acc e ang são respectivamente aceleração e ângulo do veículo em dado instante.

Para modelar esse sistema, usaremos os blocos: *realExpression*, *integrator*, *cos*, *sin*, e *multiProduct*, além de um bloco que iremos criar para auxiliar na transformação de um ângulo em graus para radianos, *degreesToRadians*. Conectamos os blocos seguindo a imagem 2.0 e definimos as variáveis do sistema como mostrado na imagem 2.1.

Para criarmos o block *degreesToRadians*, criaremos um novo modelo usando os blocos: *realExpression*, *multiProduct*, *realInput* e *realOutput*, e conectamos eles como mostrado na imagem 2.2, e definimos pi como uma variável do sistema, como ilustra a imagem 2.3.

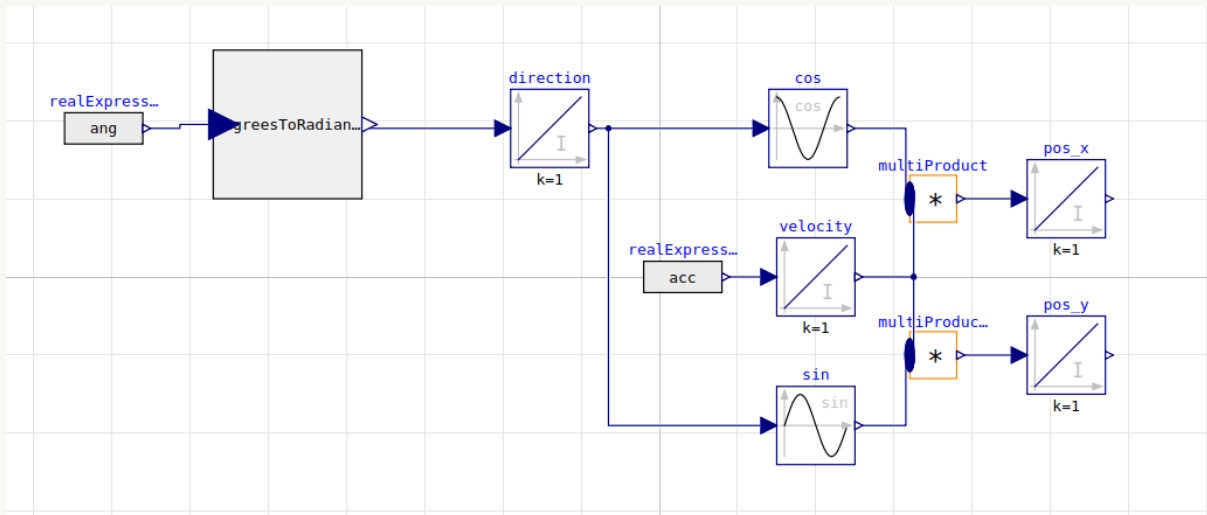


Imagem 2.0: Modelo físico do veículo

```

1  model vehicle
2    Real ang(start = 10);
3    Real acc(start = 1);
4    Modelica.Blocks.Continuous.Integrator veloc

```

Imagem 2.1: Variáveis do modelo

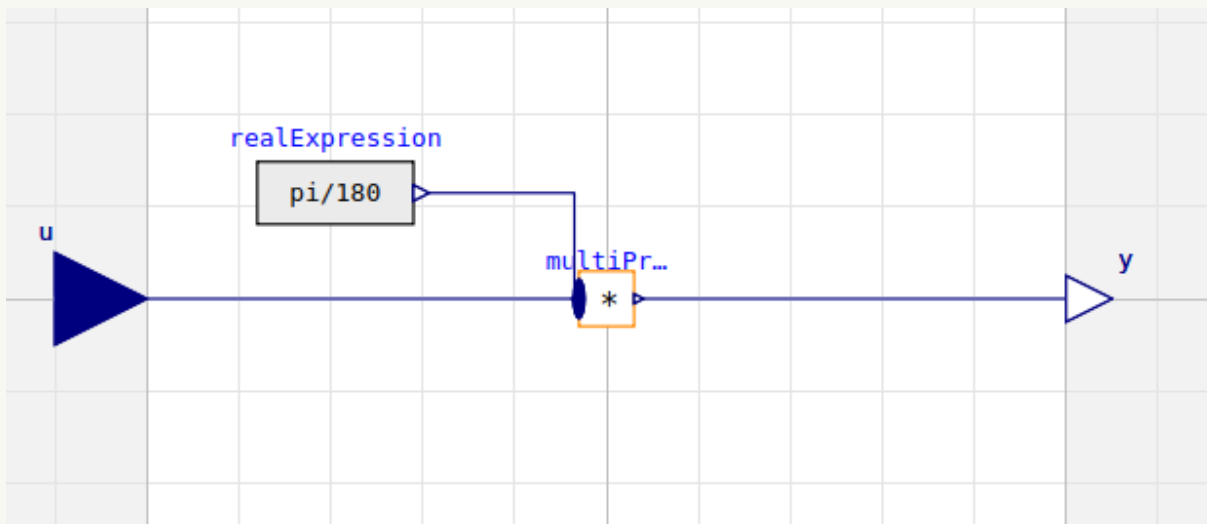


Imagem 2.2: Bloco de conversão de ângulo em graus para radianos

```

1  model degreesToRadians
2    Real pi = 3.1415926535;
3    Modelica.Blocks.Math.MultiProduct multiProduct

```

Imagem: 2.3: Variáveis do modelo do bloco de conversão

Antes de executarmos a simulação, configuramos o tempo da simulação para parar após 100 segundos, como mostrado no primeiro exemplo de modelagem. O resultado é o gráfico da imagem 2.4.

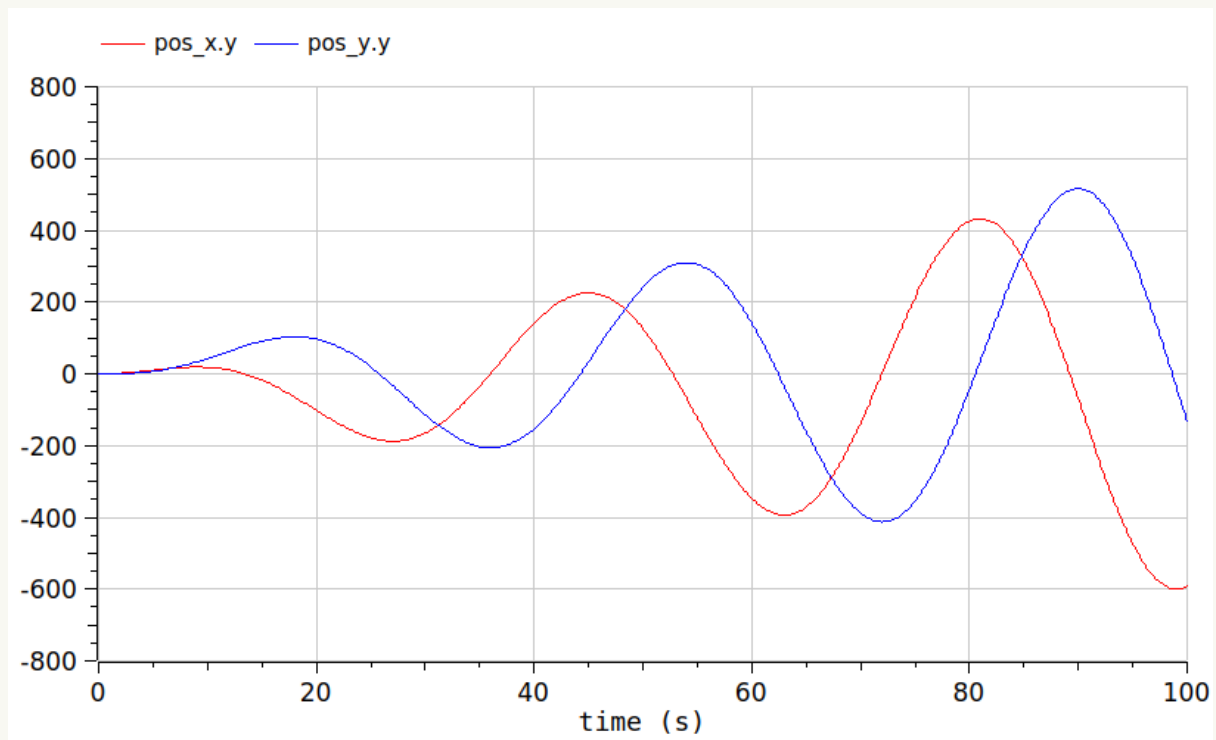


Imagem 2.4: Resultado da simulação

Podemos ainda gerar um gráfico mais interessante, que mostra o percurso do veículo, e para isso usaremos um outro tipo de plotagem chamada de *New Parametric Plot Window*. Para escolher que variáveis representam cada eixo, primeiro selecionamos a variável do eixo x utilizando o menu lateral, que apresenta todas as variáveis do sistema, enquanto pressionamos a tecla *shift*, e para selecionar a variável do eixo y basta selecionar uma variável, mas sem o *shift* estar pressionado.

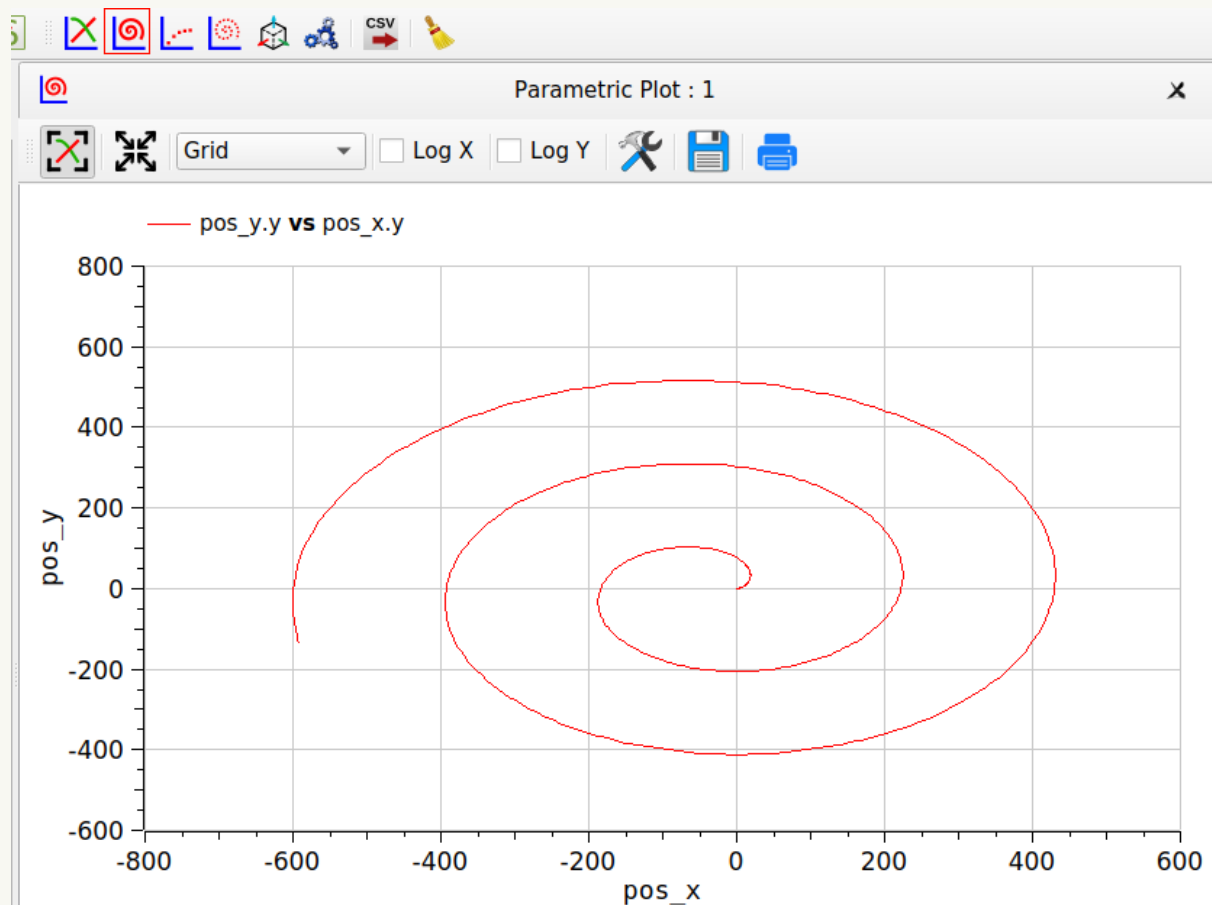


Imagem 2.5: Gráfico da trajetória do veículo durante a simulação

Com pequenas alterações no nosso código podemos adicionar uma mudança nas entradas do modelo para visualizarmos como o sistema se comporta. A mudança no algoritmo pode ser visto na imagem 2.6, e o gráfico da trajetória do veículo na imagem 2.7.

```
47 ▶ connect(multiProduct1.y  
49 algorithm  
50 if time >= 50 then  
51   ang := 30;  
52   acc := -0.5;  
53 end if;  
54 ▶ annotation( [ ] )
```

Imagem 2.6: Algoritmo para mudar ângulo e aceleração do veículo após 50 segundos

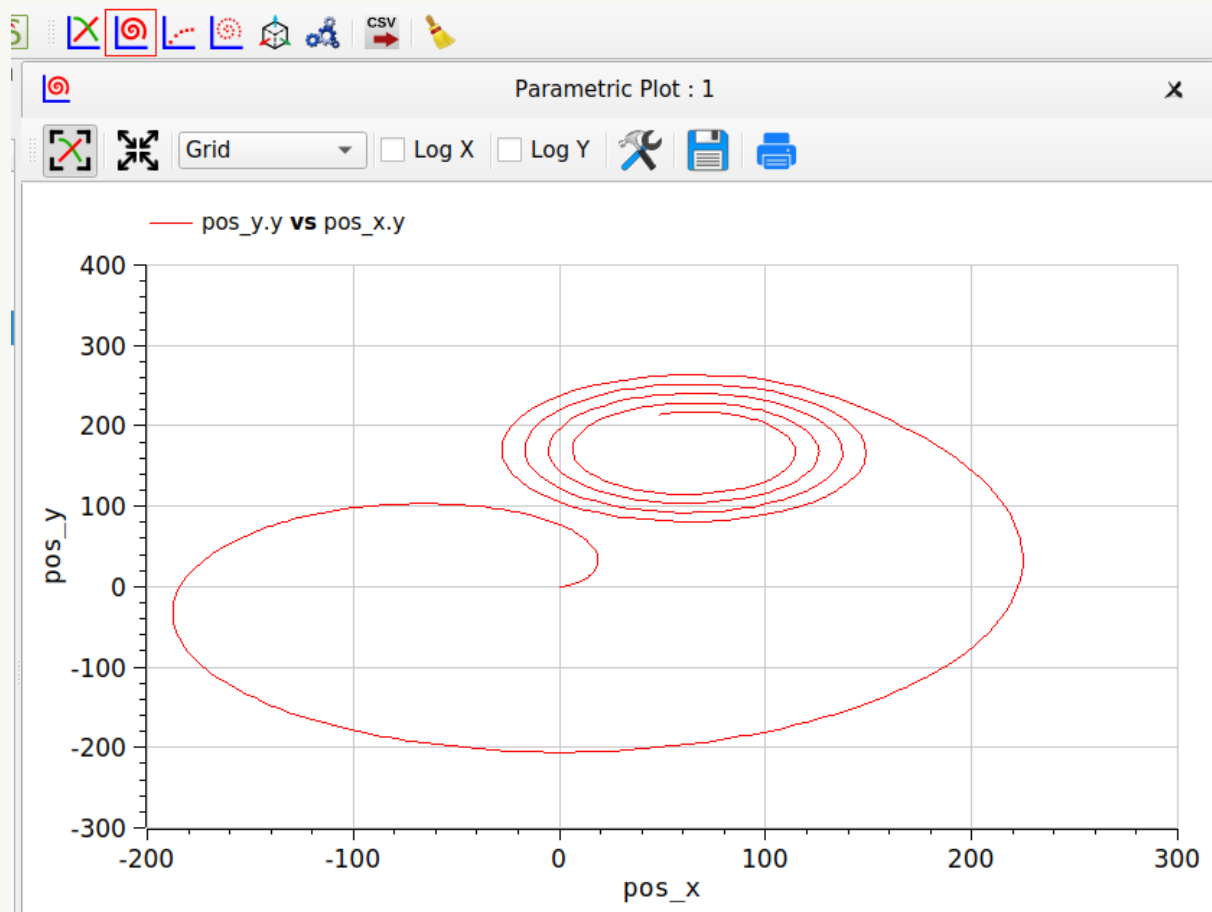
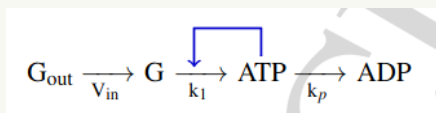


Imagem 2.7: Gráfico da trajetória do veículo com mudança no ângulo e aceleração

3. Modelo de oscilações de glicose em leveduras

Este modelo, também apresentado previamente na apostila da disciplina, tem como intuito simular as oscilações glicolíticas em leveduras, por meio do ensaio do transporte de glicose de de fora para dentro da célula seguindo a equação química abaixo:



Na realidade existem mais etapas e enzimas presentes, mas para simplificar esta equação foi escolhida. Para simular as variações de G e ATP em função do tempo foram utilizadas as seguintes equações diferenciais:

$$\frac{d[ATP]}{dt} = 2k_1[G][ATP] - \frac{k_p[ATP]}{[ATP] + K_m}$$

$$\frac{d[G]}{dt} = V_{in} - k_1[G][ATP]$$

Uma vez que o modelo é composto em apenas duas equações, para sua criação basta adicionar integradores as suas respectivas expressões

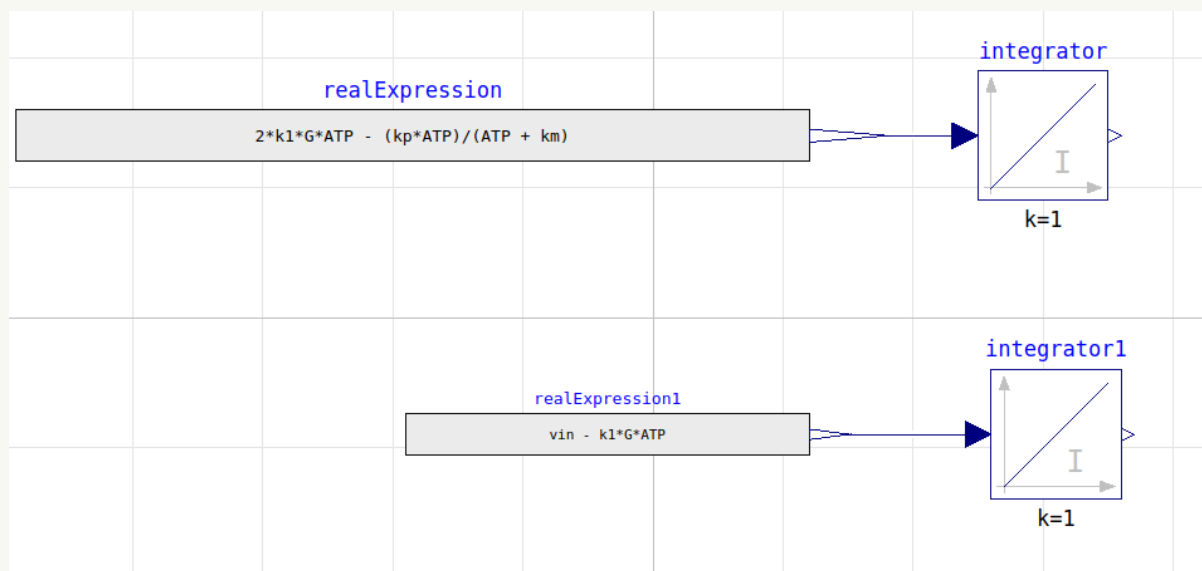


Imagem 3.1: Modelo de rede metabólica em openmodelica

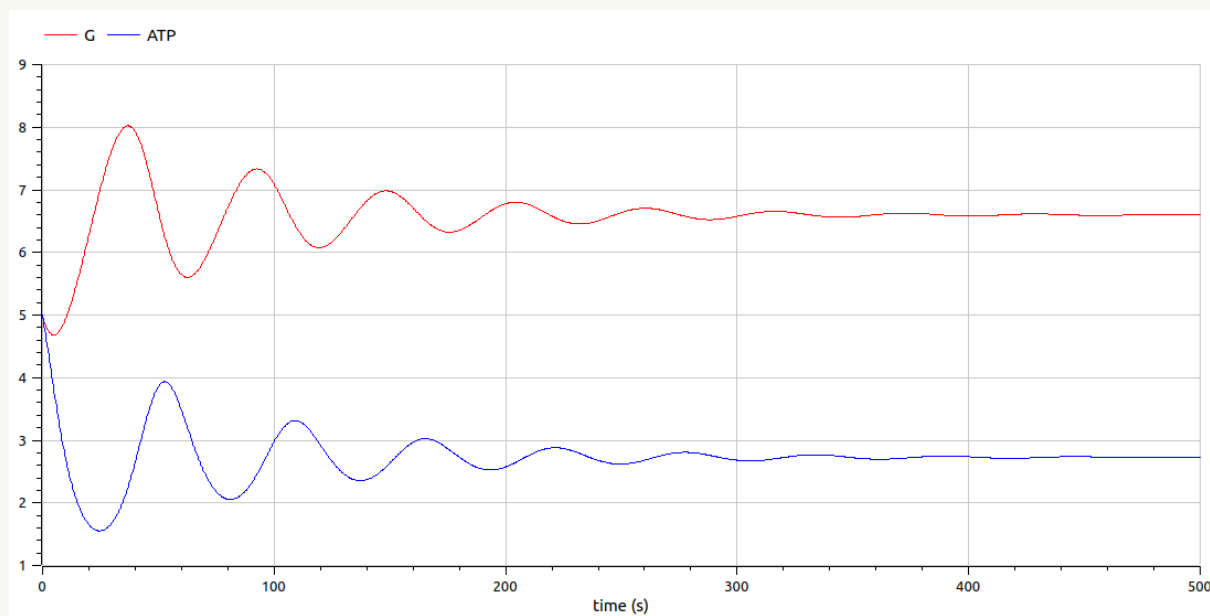


Imagem 3.2: Gráfico resultante da simulação

4. Modelo de impressão 2D

Esse modelo tem como objetivo simular os motores que movimentam o cabeçote de uma impressora 2D entre coordenadas. A parte física implementada dessa simulação possui como *input* duas coordenadas 2D, representando os pontos de origem e destino do cabeçote da impressora, e como output a posição dos motores ao decorrer do tempo, que pode ser projetada em um plano para fornecer a trajetória do cabeçote entre os pontos fornecidos.

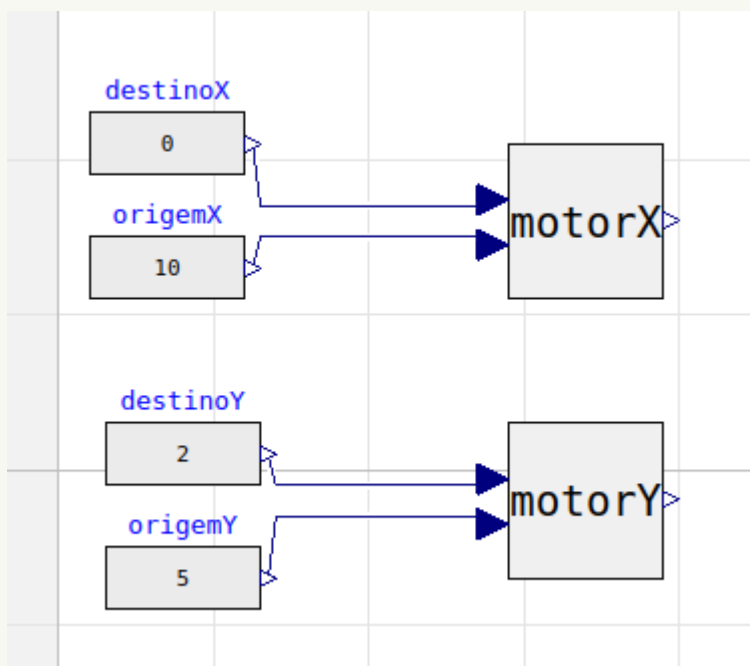


Imagem 4.1: Modelo da físico da impressora 2D

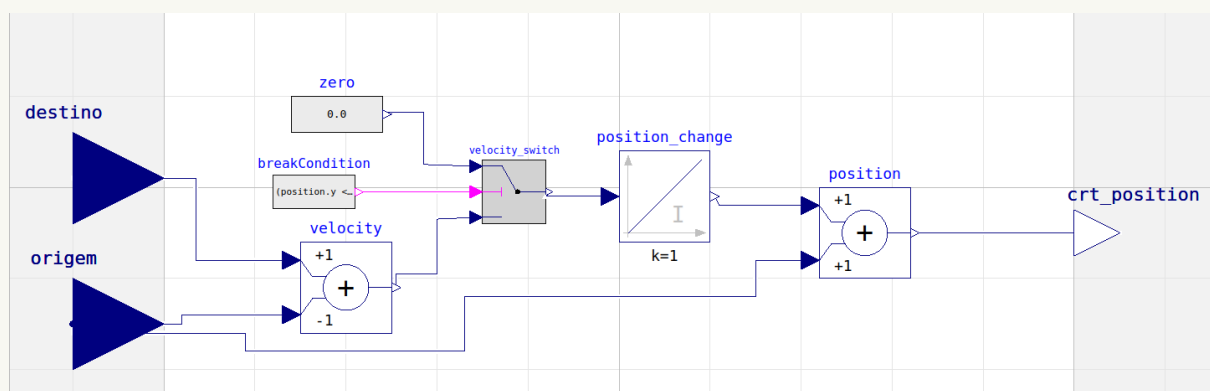


Imagem 4.2: Modelagem dos submotores da impressora

Como demonstrado na imagem 4.1, o modelo físico da impressora consiste apenas em fornecer as coordenadas para seus respectivos motores, futuramente ele também implementar a parte discreta do modelo, fornecendo novas coordenadas aos motores ao decorrer do tempo.

Para modelar os motores da impressora foram usados os blocos *add*, *realExpression*, *booleanExpression*, *switch* e *integrator*. A velocidade do motor é calculada pela diferença entre os pontos destino e origem, ela é consequentemente integrada com o intuito de fornecer a mudança de posição ao decorrer do tempo. Optamos por não modelar a aceleração do motor nesta entrega, por necessidade de um componente discreto para controlar a aceleração.

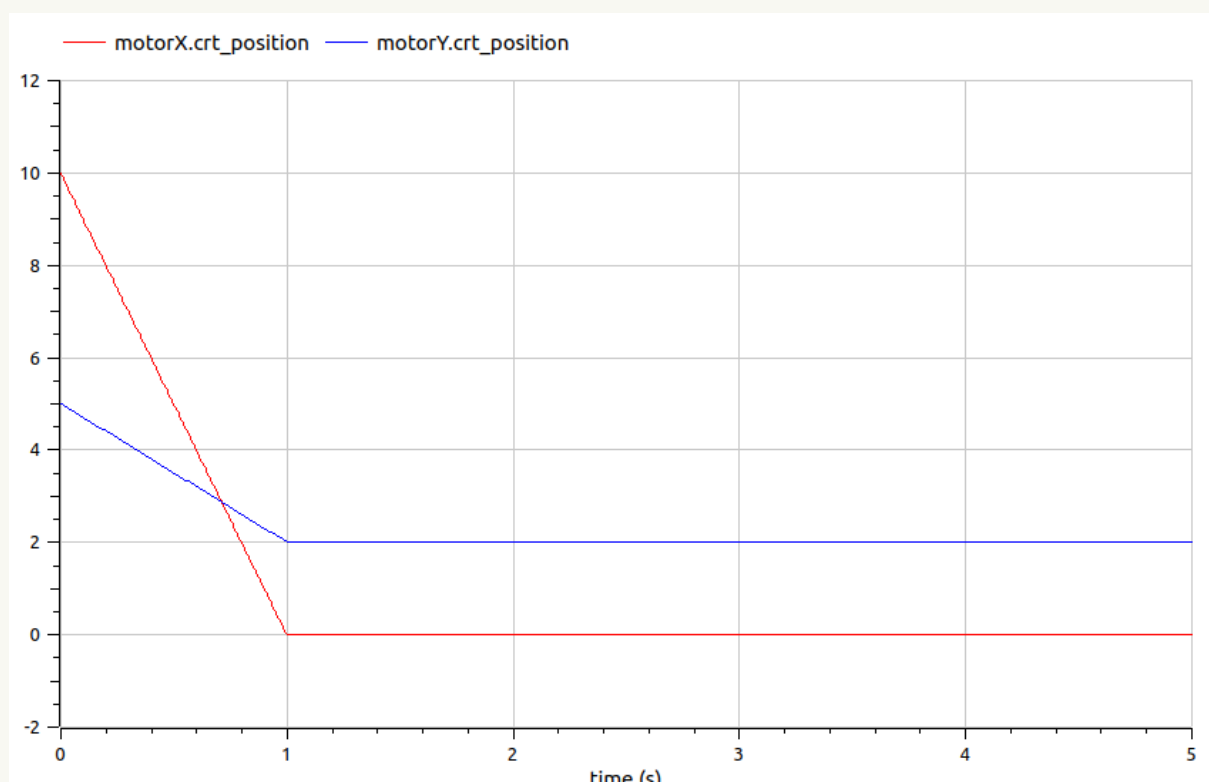


Imagem 4.3: Gráfico das posições dos motores em função do tempo, entre os pontos (10, 5) e (0, 2).

Uma pequena parte discreta foi introduzida com o intuito de parar os motores quando atingirem seus destinos, composta pelos blocos *booleanExpression* e *switch*, de maneira a zerar a velocidade caso o destino tenha sido atingido.

Também foi implementado um protótipo do modelo que também simula a aceleração e velocidade máxima, no entanto este ainda apresenta falhas na lógica de seus componentes discretos, para testar sua execução basta mudar os motores utilizados na modelagem da impressora.

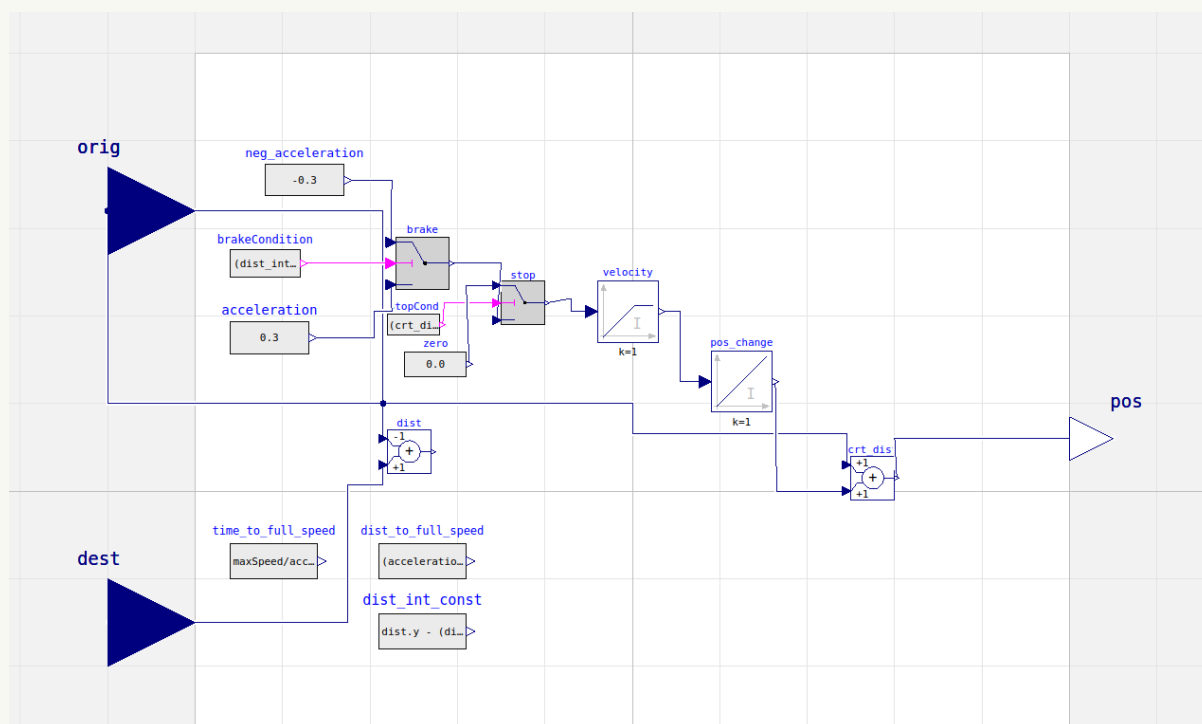


Imagem 4.4: Modelo que simula aceleração

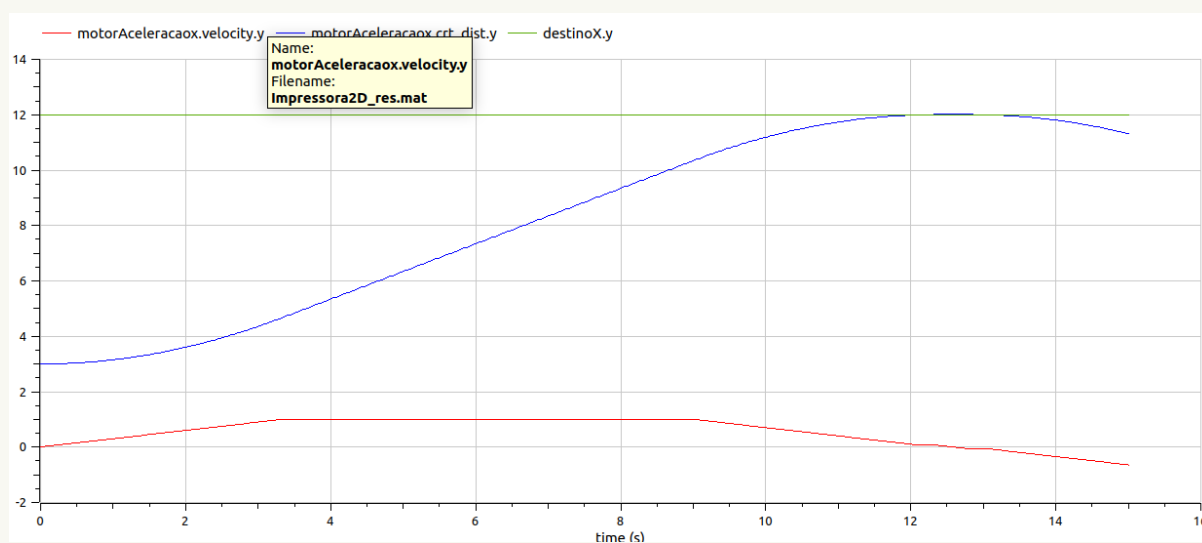


Imagem 4.5: Gráfico resultante da simulação com aceleração

A simulação representada na figura 4.5 demonstra o motor acelerando até sua velocidade máxima, e desacelerando antes de atingir seu destino. Para que o motor fique com sua velocidade igual à zero ao atingir o destino, é calculada a distância percorrida até que atinja sua velocidade máxima. No entanto, como pode ser visto na imagem 4.5, a aceleração não foi igualada a zero no momento que o destino foi alcançado.

Em comparação com o modelo mais simples, sua maior diferença é um cálculo de integral a mais, com intuito de transformar a aceleração do motor em sua velocidade.

Considerações finais

Como uma primeira parte de um trabalho mais extenso, essa etapa teve êxito em completar as modelagens físicas dos quatro exemplos propostos. Como próximo pretende-se trabalhar em modelagens lógicas para cada um dos exemplos, fazendo uso de máquinas de estado finitos (FSM - *Finite State Machine*).

Para o primeiro exemplo a modelagem lógica agirá como controlador do aquecedor do ambiente, o ligando e desligando em momentos oportunos; no segundo exemplo, a FSM terá o objetivo de simular o motorista do veículo, controlando sua aceleração e direção; para o terceiro exemplo a modelagem lógica variará os termos das equações, a fim de observar como o sistema se comporta frente a mudanças no seu ambiente; e para o último exemplo a FSM irá controlar as entradas das coordenadas para impressão 2D, além de ser desenvolvido um controle mais completo sobre a aceleração causada pelos motores.

Finalmente, com as modelagens físicas e lógicas completas, ainda numa segunda etapa do trabalho, a junção dessas duas partes será realizada, tendo como objetivo final chegar a quatro modelos cyber-físicos, um para cada exemplo apresentado anteriormente.