

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

Gabriel de Vargas Coelho (20200400)

João Pedro Lopes de Camargo (21100134)

Florianópolis

24 de novembro de 2023

# Sumário

<b>Introdução.....</b>	<b>3</b>
<b>1. Modelo de controle de temperatura de um ambiente com aquecedor.....</b>	<b>4</b>
<b>2. Modelo de veículo e motorista.....</b>	<b>10</b>
<b>3. Modelo de oscilações de glicose em leveduras.....</b>	<b>18</b>
<b>4. Modelo de impressão 2D.....</b>	<b>23</b>
<b>Considerações finais.....</b>	<b>31</b>

# Introdução

O presente trabalho tem como objetivo apresentar 4 exemplos de modelagem física, lógica e cyber-físicas 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.

# 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: *realExpression* e *integrator* – que podem ser encontrados pela 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.

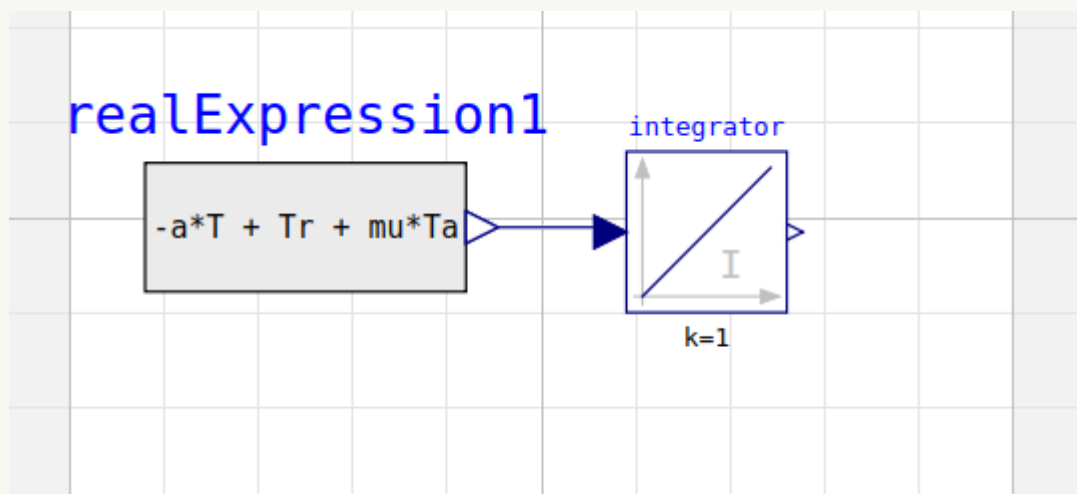
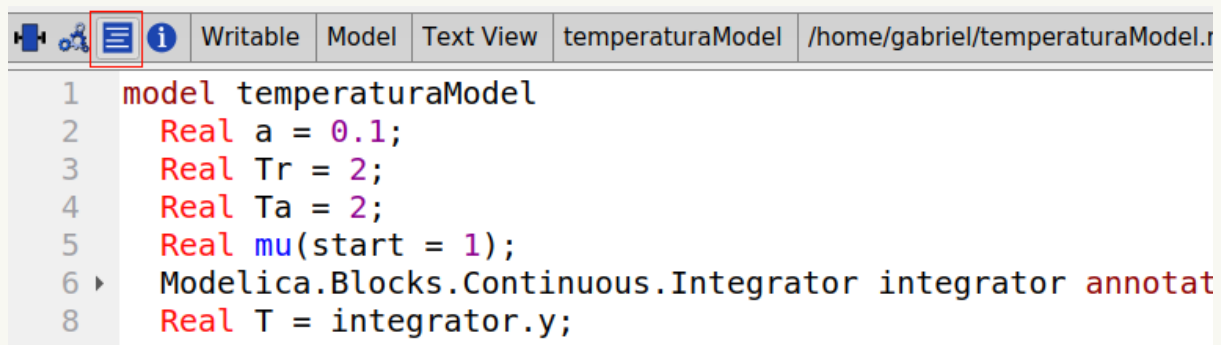


Imagem 1.0: Modelo físico do controle de temperatura de um ambiente

Para inserir a expressão no bloco *realExpression* basta clicar com o botão esquerdo do *mouse* duas vezes sobre o bloco, e no campo de entrada *y* adicionar a expressão  $-a \cdot T + Tr + \mu \cdot Ta$ . 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, e para isso acessamos na barra de opções superior *Simulation > Simulation Setup* e mudamos o tempo de parada (*Stop Time*) para 100 segundos. Agora para simularmos nosso modelo é só selecionar *Simulation > Simulate*, que se encontra na barra de opções superior. O resultado da simulação pode ser visto no gráfico gerado ao final da simulação, e ao selecionar a variável *T* para visualização.

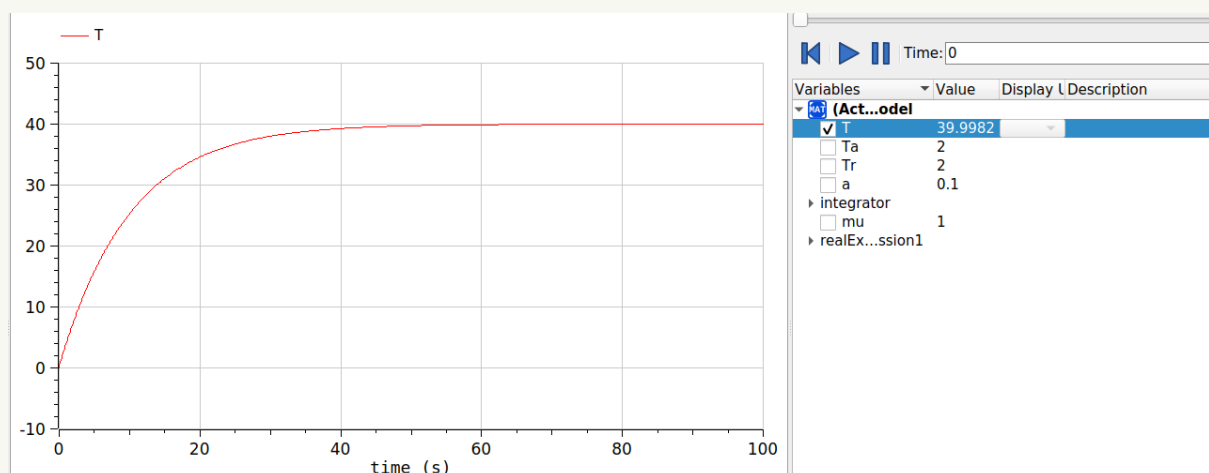


Imagem 1.2: Resultado da simulação

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.

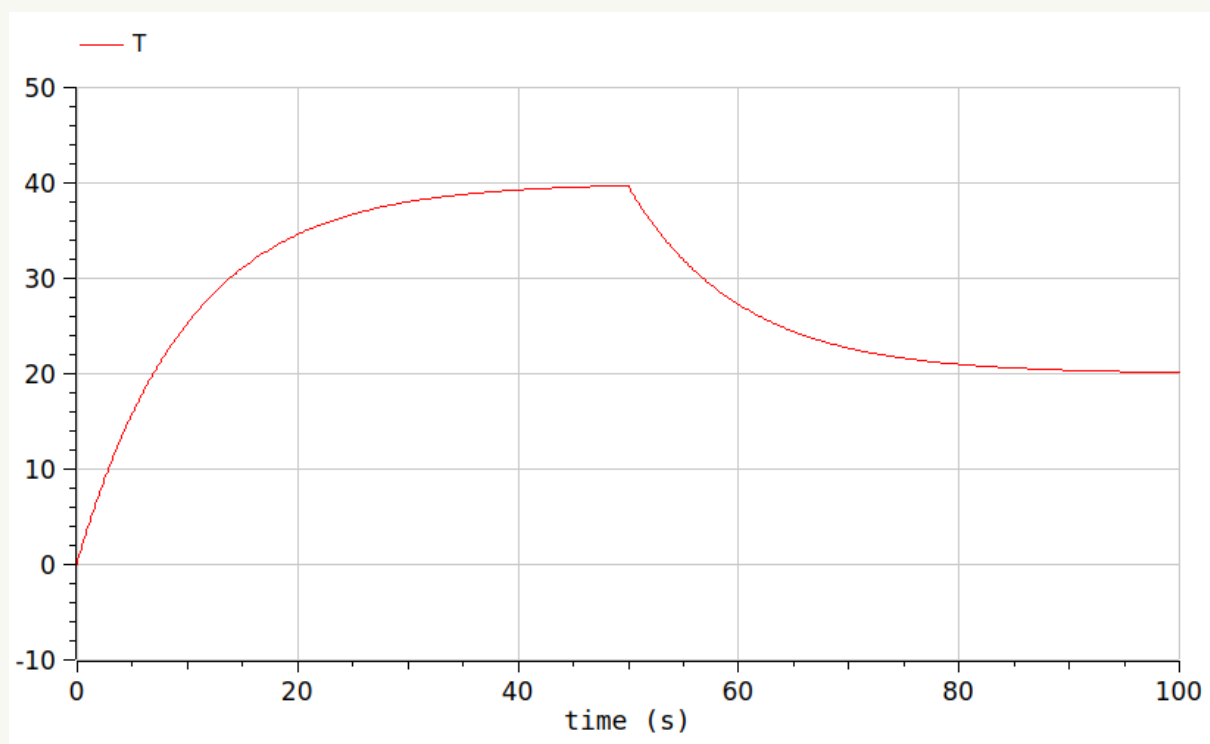


Imagem 1.4: Resultado da simulação com estímulo ao aquecedor

Para um controle de temperatura verdadeiro é necessário que o aquecedor seja ligado e desligado quando a temperatura do ambiente chegar a determinados valores, e para atingir esse objetivo precisaremos de um modelo lógico. Antes de construirmos nosso modelo lógico, primeiro iremos excluir o código da imagem 1.3, retirando a seção *algorithm*, que serviu apenas para testes, e segundo: vamos

construir uma máquina de estados no mesmo arquivo do nosso modelo físico. Na construção da máquina de estados usaremos os seguintes blocos: *stateGraphRoot*, *initStep*, *step*, *transitionWithSignal* e *booleanExpression*, da seguinte forma:

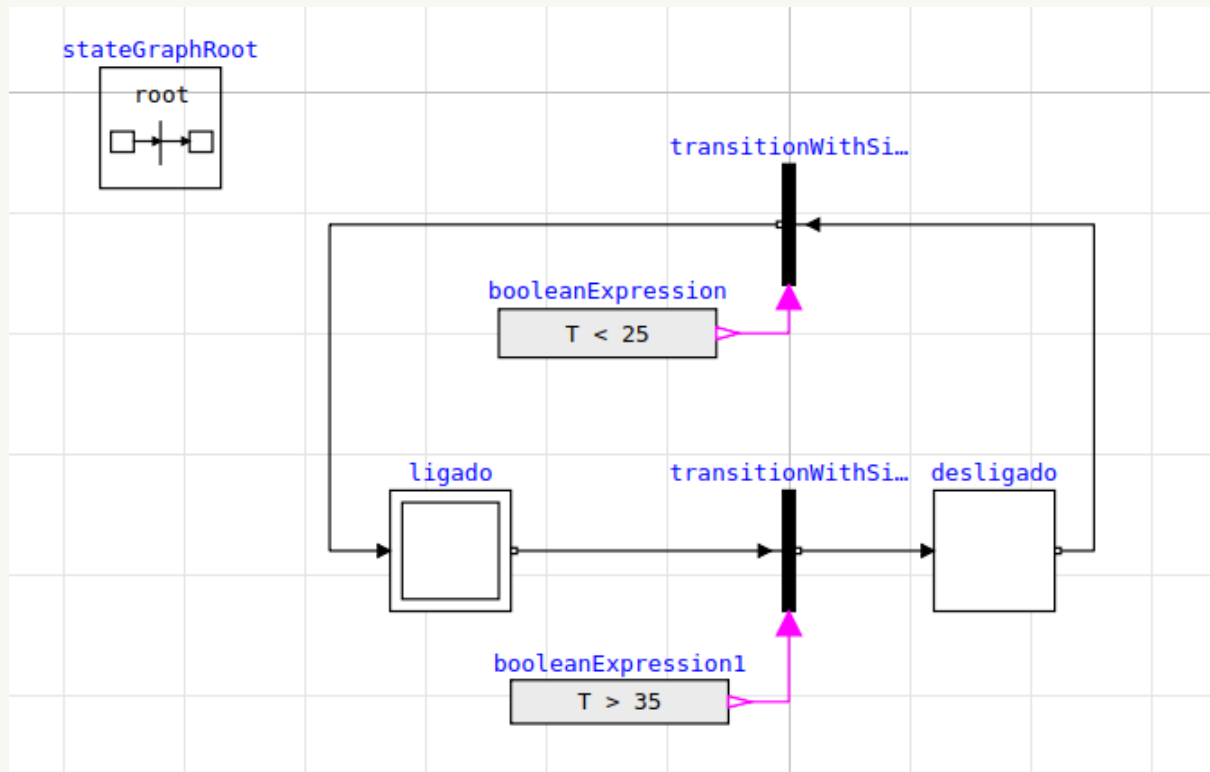


Imagem 1.5: Máquina de estados para controle de temperatura

Observe que na imagem os blocos *initStep* e *step* foram renomeados para *ligado* e *desligado* respectivamente. Para fazer isso é necessário clicar com o botão direito do mouse sobre o bloco, e selecionar a opção *Attributes*. No formulário que se mostrará altere o campo chamado *Name* para o novo nome do bloco.

Note como as transições usam os sinais das expressões booleanas como forma de saber quando a transição entre os estados está ativa. Contudo, ainda é preciso configurar o que acontece quando um estado está ativo, e para isso voltaremos a visualização do documento em texto, na visualização *Text View*, e adicionaremos o seguinte código dentro da seção *equation*:

```
equation
  when ligado.active then
    mu = 1;
  elseif desligado.active then
    mu = 0;
  end when;
```

Imagem 1.6: Ação nos estados da máquina de estado

Observa-se que quando o estado ativo da máquina de estados é o ligado (bloco *initStep*) o aquecedor está ligado, e quando o estado ativo é o desligado (bloco *step*) o aquecedor está desligado. Como os modelos se encontram no mesmo arquivo, e usam as mesmas variáveis, sendo elas  $T$  e  $\mu$ , já obtemos nosso primeiro sistema cyber-físico. Simulando novamente nosso sistema, obtemos agora o seguinte gráfico:

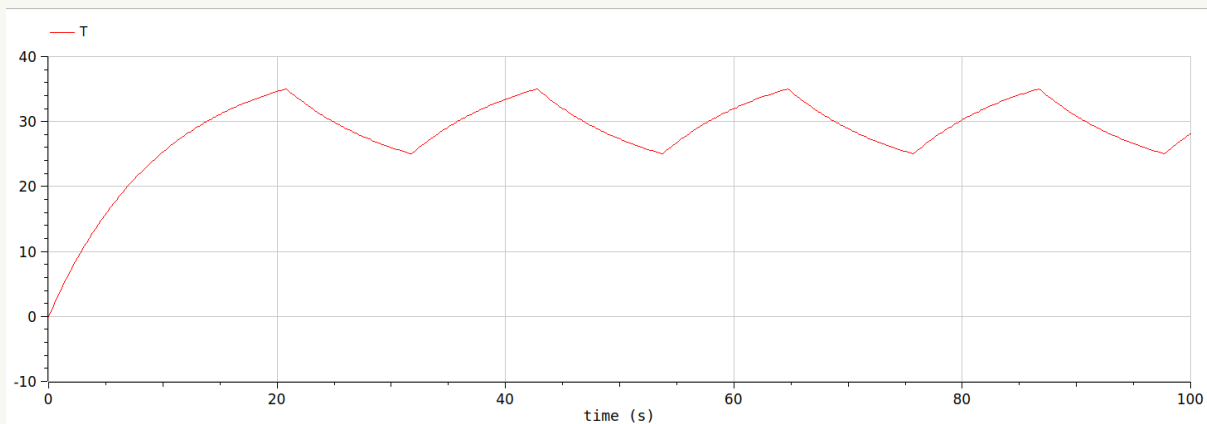


Imagem 1.7: Resultado da simulação do sistema cyber-físico



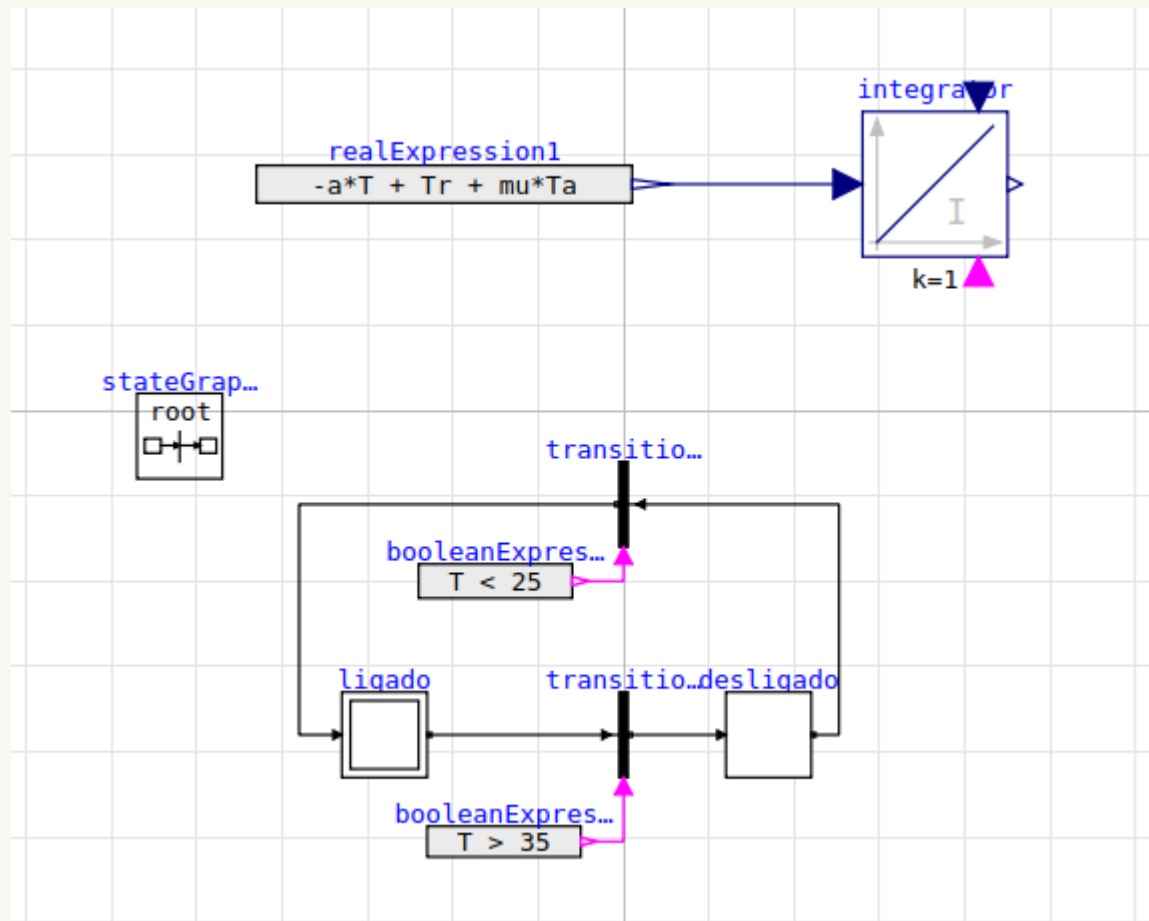


Imagem 1.8: Modelo final em blocos

## 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 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. Usaremos os blocos: *realExpression*, *integrator*, *cos*, *sin*, *multiProduct*, e *from\_deg*, que podem ser encontrados pela ferramenta de busca *Libraries Browser*. Utilizaremos os blocos da seguinte forma:

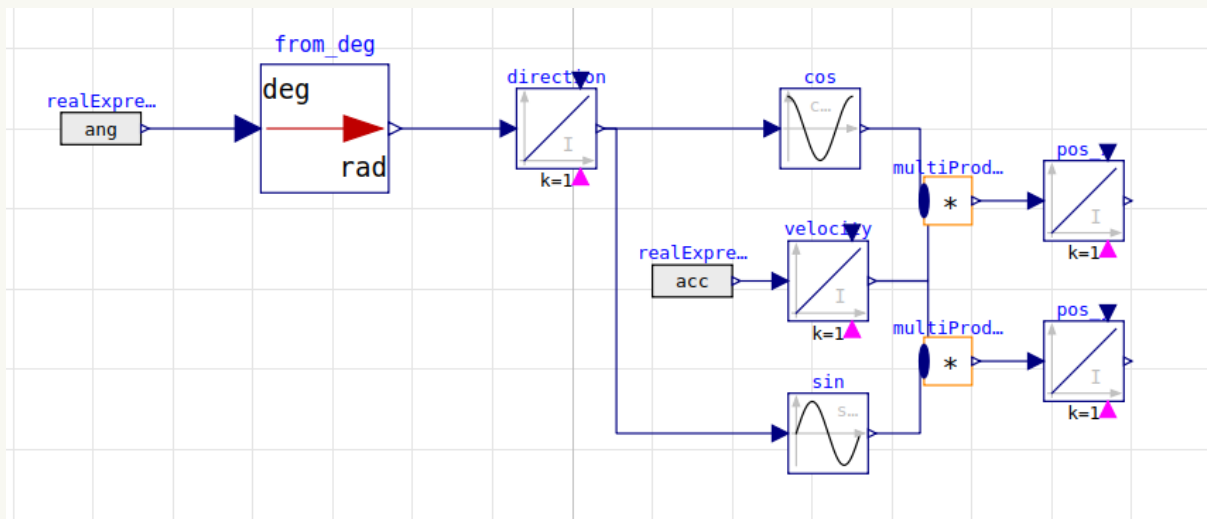


Imagem 2.0: Modelo físico do veículo

Observe que na imagem alguns blocos foram renomeados. Para renomear um bloco é necessário clicar com o botão direito do mouse sobre o bloco, e selecionar a opção *Attributes*. No formulário que se mostrará altere o campo chamado *Name* para o novo nome do bloco.

Para que nossa simulação funcione adequadamente precisamos também definir o que *ang* e *acc* significam, e para isso utilizamos a aba *Text View* e modificamos o texto adicionando algumas linhas, como pode ser observado na imagem abaixo.

```

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

Antes de executarmos a simulação, configuramos o tempo da simulação para parar após 100 segundos, para isso acessamos na barra de opções superior *Simulation > Simulation Setup* e mudamos o tempo de parada (*Stop Time*) para 100 segundos. Agora para simularmos nosso modelo é só selecionar *Simulation > Simulate*, que se encontra na barra de opções superior. Na janela de resultado da simulação, selecionando as variáveis de interesse do modelo o resultado é o gráfico da imagem 2.2.

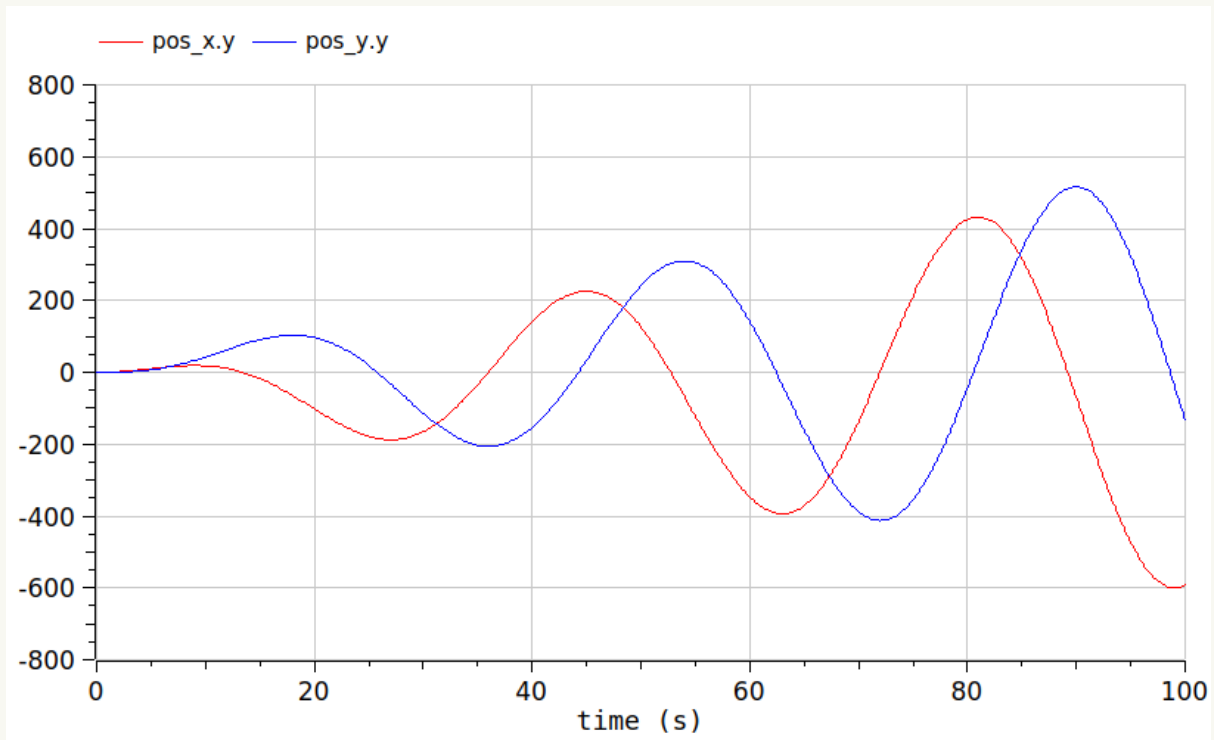


Imagem 2.2: Resultado da simulação

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 *shit* estar pressionado.

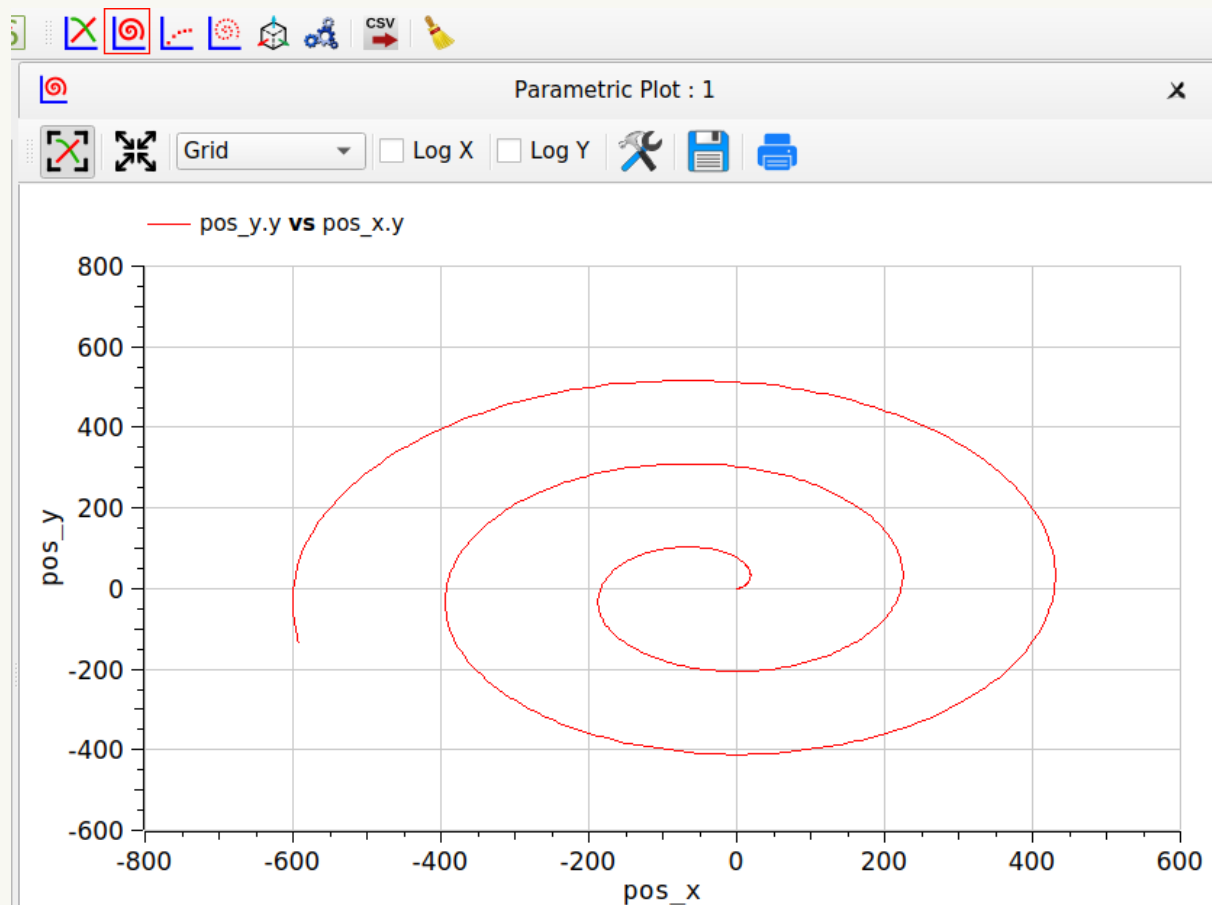


Imagem 2.3: Gráfico da trajetória do veículo durante a simulação

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 2.4, e o gráfico da trajetória do veículo na imagem 2.5.

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

Imagem 2.4: Algoritmo para mudar ângulo e aceleração do veículo após 50 segundos

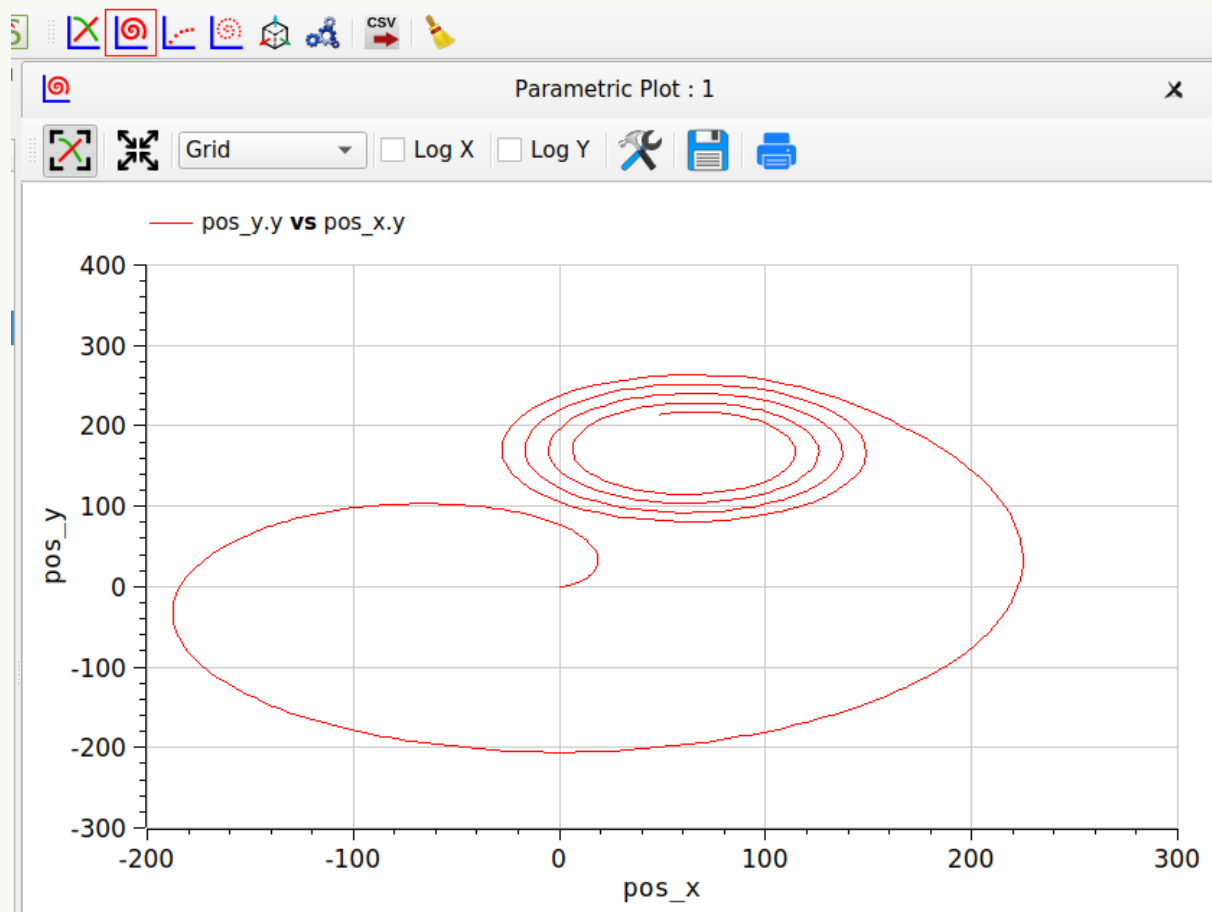


Imagem 2.5: Gráfico da trajetória do veículo com mudança no ângulo e aceleração

Nosso modelo até agora não apresenta um grande controle do motorista sobre o veículo, a única mudança no sistema é feita no momento 50. Imagine agora o seguinte cenário: o veículo inicia parado e com direção reta; depois de 1 segundo o motorista acelera a  $2\text{m/s}^2$ ; quando o veículo atinge a velocidade de  $16,5\text{ m/s}$  o motorista para de acelerar e vira o volante  $30^\circ$  à esquerda; depois de mais 5 segundos o motorista volta o volante para a posição inicial; depois de mais 5 segundos ele acelera a  $3\text{ m/s}^2$  com o volante virado  $25^\circ$  à direita; quando chega a velocidade de  $33\text{ m/s}$  o motorista volta o volante a posição neutra e começa a desacelerar a  $0,5\text{ m/s}^2$ ; quando o veículo atinge a velocidade 0 o motorista para de acelerar o carro.

Para que possamos modelar o cenário descrito precisaremos de um modelo lógico para nosso sistema, e esse modelo será representado por meio de uma máquina de estados finito. Mas antes de implementarmos nossa máquina de estados precisaremos mudar um pouco nosso modelo físico, primeiro configuramos os valores iniciais de *ang* e *acc* para zero, e após isso, excluiremos a seção

*algorithm* que adicionamos na imagem 2.4. Feito isso, no mesmo arquivo montaremos a máquina de estados utilizando os blocos: *stateGraphRoot*, *initialStepWithSignal*, *transitionWithSignal*, *step*, *stepWithSignal*, *timer*, *greaterThreshold*, e *booleanExpression*. Os blocos podem ser encontrados utilizando a ferramenta de busca apresentada anteriormente. A imagem abaixo ilustra como conectarmos todos esses blocos:

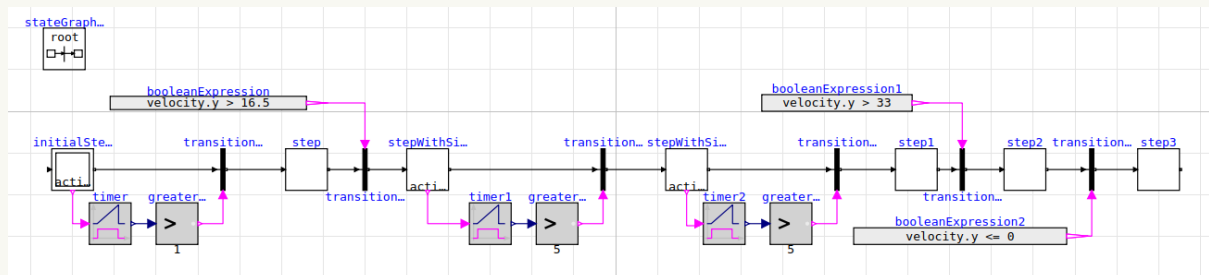


Imagem 2.6: Máquina de estados que representa o motorista

No último estado da nossa máquina de estados, que nesse caso é chamado de *step3*, é importante sinalizar que se trata de um estado final, isso é feito com um clique direito do mouse sobre o bloco, depois acessando a opção de atributos e marcando a opção *Final*.

No arquivo de texto do modelo, adicionaremos as ações que acontecem em cada estado, para isso adicionamos código à seção *equation*, seguindo a imagem 2.7:

```

equation
  when initialStepWithSignal.active then
    acc = 0;
    ang = 0;
  elseif step.active then
    acc = 2;
    ang = 0;
  elseif stepWithSignal.active then
    acc = 0;
    ang = 30;
  elseif stepWithSignal1.active then
    acc = 0;
    ang = 0;
  elseif step1.active then
    acc = 3;
    ang = -25;
  elseif step2.active then
    acc = -0.5;
    ang = 0;
  elseif step3.active then
    acc = 0;
    ang = 0;
  end when;

```

Imagem 2.7: Ação nos estados da máquina de estados

Com a máquina de estados pronta temos agora o seguinte modelo cyber-físico:

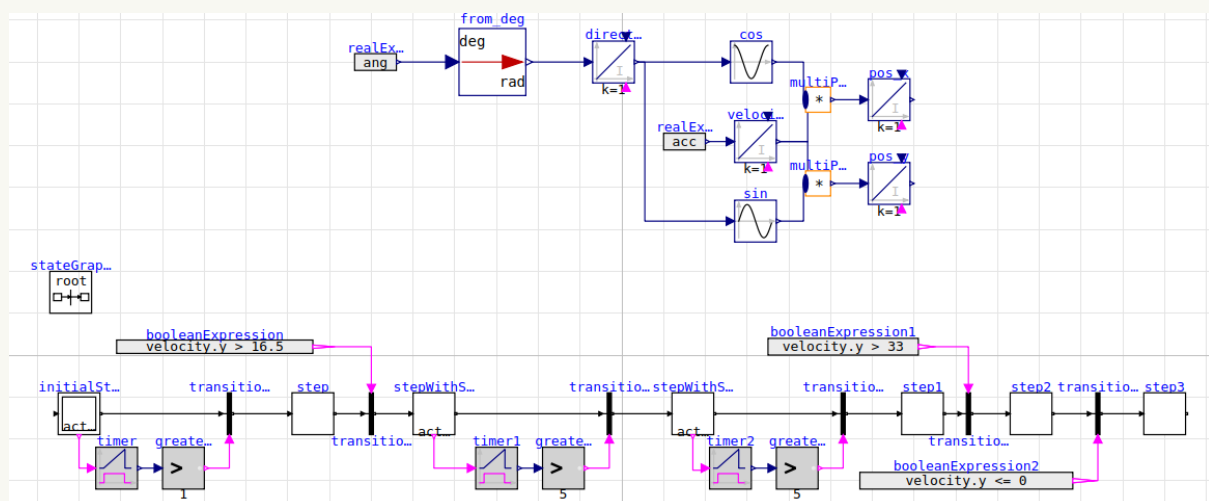


Imagem 2.8: Modelo final em blocos



Executando agora o modelo cyber-físico, e selecionando as variáveis de interesse para o gráfico, obtemos os seguintes resultados:

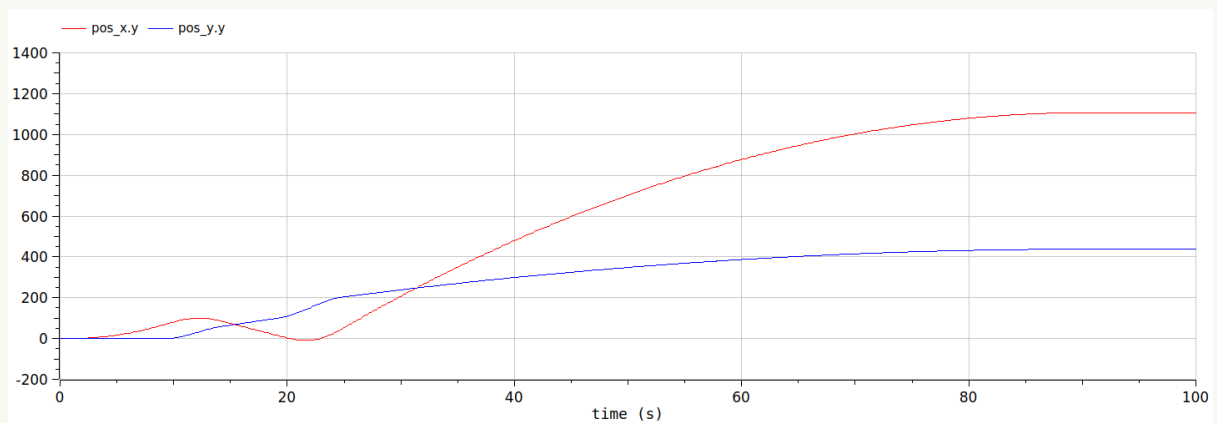


Imagem 2.9: Resultado da simulação com a posição x e y do veículo pelo tempo

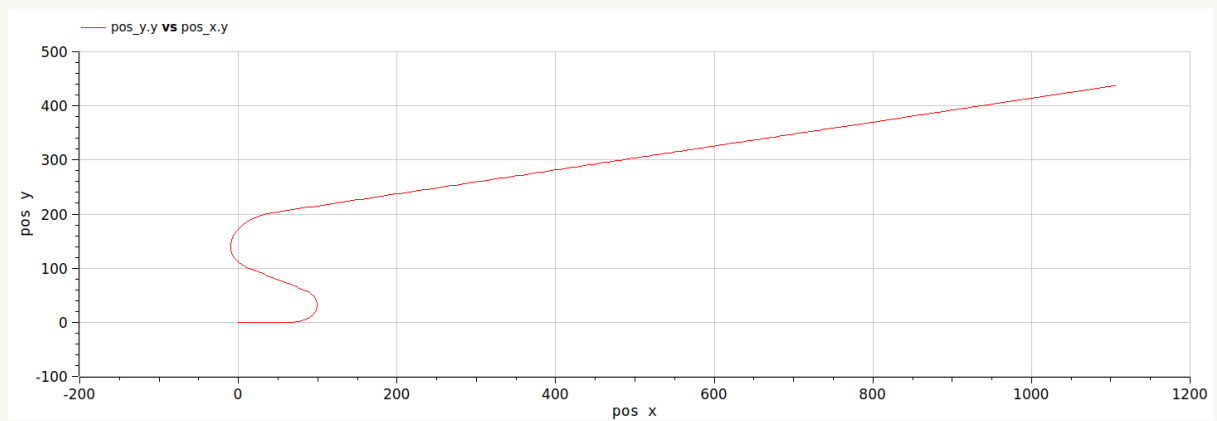
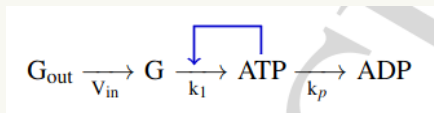


Imagem 2.10: Trajetória do veículo

### 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:

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

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. Usaremos os blocos: *realExpression*, e *integrator*, que podem ser encontrados pela ferramenta de busca *Libraries Browser*. Utilizaremos os blocos da seguinte forma:

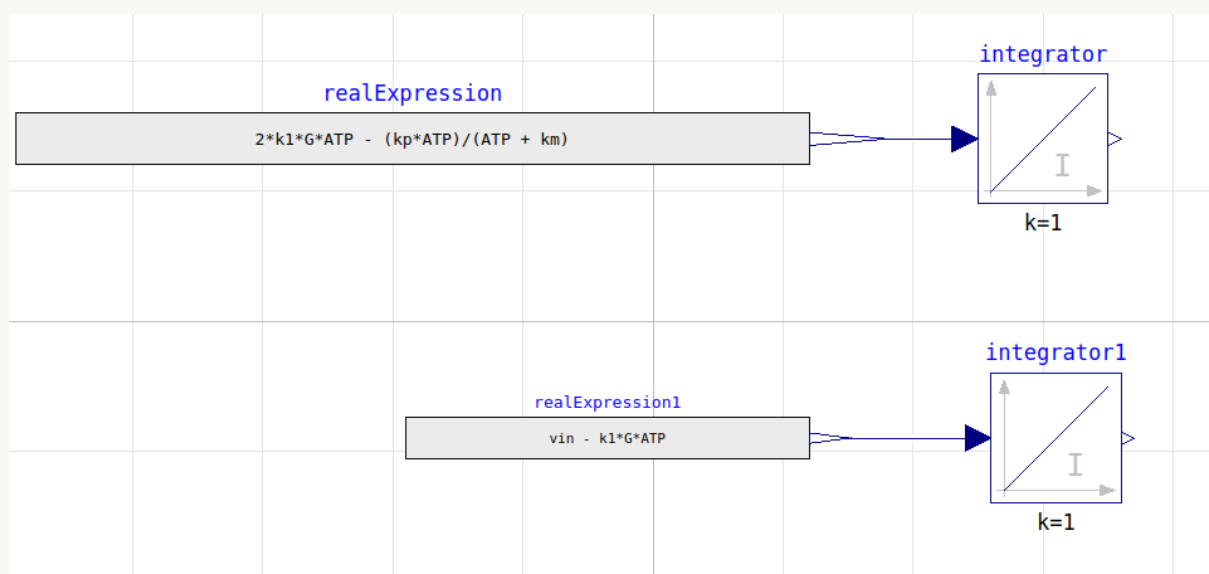


Imagem 3.1: Modelo de rede metabólica em openmodelica

Note que a saída do bloco *integrator* será o responsável por nos fornecer o valor numérico de ATP num dado instante, e por sua vez o bloco *integrator1* fornecerá o valor numérico para a glicose num dado instante.

Para definir os parâmetros e constantes das fórmulas matemáticas é necessário ainda modificar o código da modelagem, para isso é necessário acessar a visualização de texto do modelo, no *Text View* do modelo adicione:

```
model glucose
  Real k1 = 0.02;
  Real kp = 6;
  Real km(start = 20);
  Real vin(start = 0.36);

  Modelica.Blocks.Continuous.Integrator integrator(in
true), y_start = 5) annotation( ...);
  Modelica.Blocks.Continuous.Integrator integrator1(i
= true), y_start = 5) annotation( ...);

  Real ATP = integrator.y;
  Real G = integrator1.y;
```

Imagem 3.2: Variáveis do modelo

Antes de executarmos a simulação, configuramos o tempo da simulação para parar após 500 segundos, para isso acessamos na barra de opções superior *Simulation > Simulation Setup* e mudamos o tempo de parada (*Stop Time*) para 500 segundos. Agora para simularmos nosso modelo é só selecionar *Simulation > Simulate*, que se encontra na barra de opções superior. Na janela de resultado da simulação, selecionando as variáveis de interesse do modelo o resultado é o gráfico da imagem a seguir:

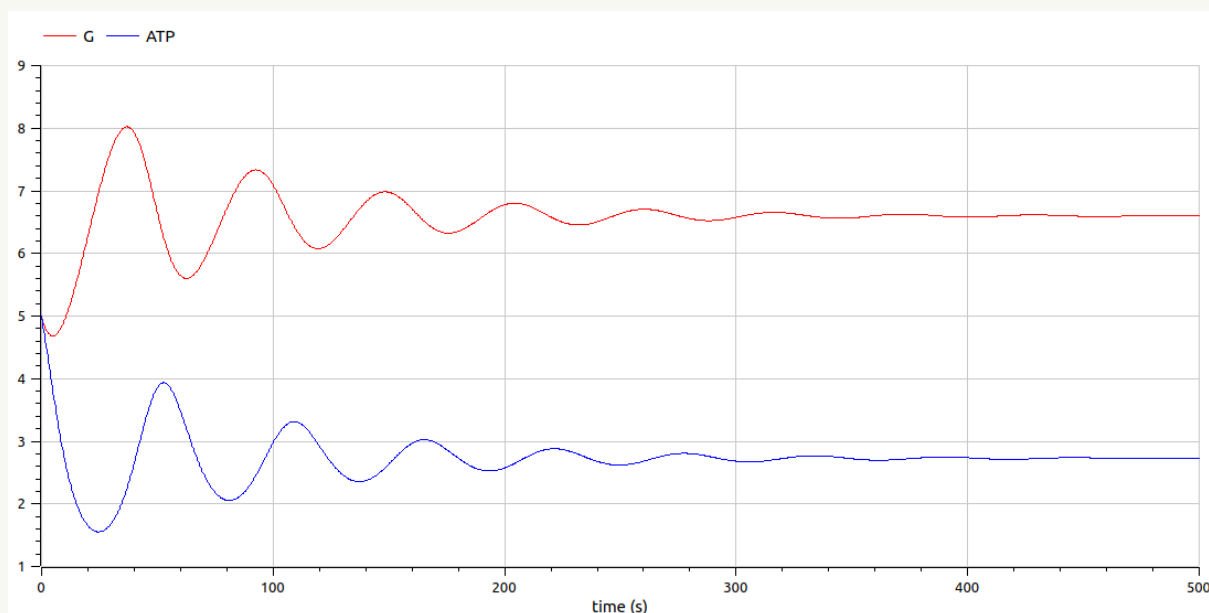


Imagem 3.3: Gráfico resultante da simulação

Imagine agora que por causa das características extrínsecas à levedura, sua taxa de absorção de glicose e também o parâmetro  $km$  da variância de ATP sofrem mudanças após um certo período de tempo. Considere que os valores de  $V_{in}$  e  $km$  começam com os valores 0,36 e 20 respectivamente; após 250 segundos o valor de  $V_{in}$  dobra; e após mais 250 segundos o parâmetro  $km$  muda para 13.

Para modelar o cenário descrito usaremos a máquina de estados mostrada a seguir:

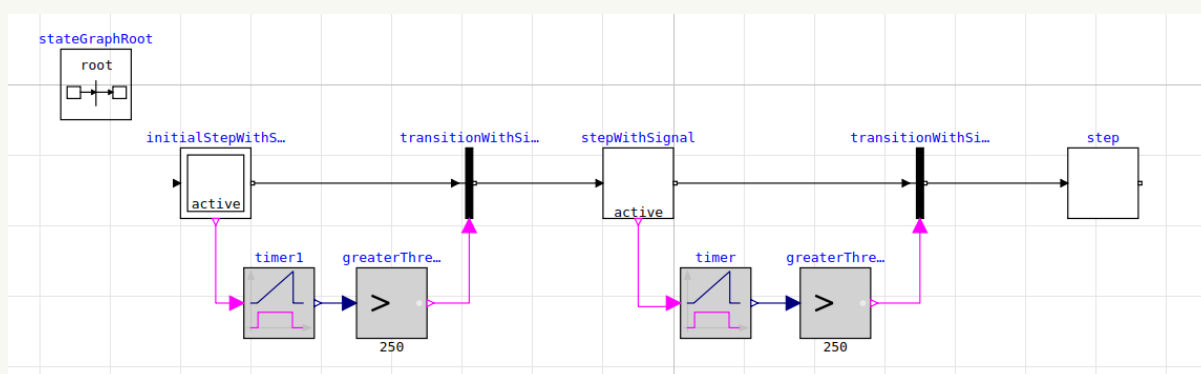


Imagem 3.4: Máquina de estados do modelo

Se atente aos valores mostrados abaixo dos blocos *greaterThreshold*, que devem ser o valor 250. Para configurar esse valor basta dar um duplo clique com o mouse sobre o bloco, e editar o campo *threshold* no formulário.

A ação realizada em cada estado é descrita no arquivo de texto do modelo, que pode ser acessado pelo *Text View*, e deve ser como mostrado:

```

equation
  when initialStepWithSignal.active then
    km = 20;
    vin = 0.36;
  elseif stepWithSignal.active then
    km = 20;
    vin = 0.62;
  elseif step.active then
    km = 13;
    vin = 0.62;
  end when;

```

Imagem 3.5: Ações de cada estado

Com a adição do modelo lógico, representado pela máquina de estados, temos como resultado o seguinte modelo:

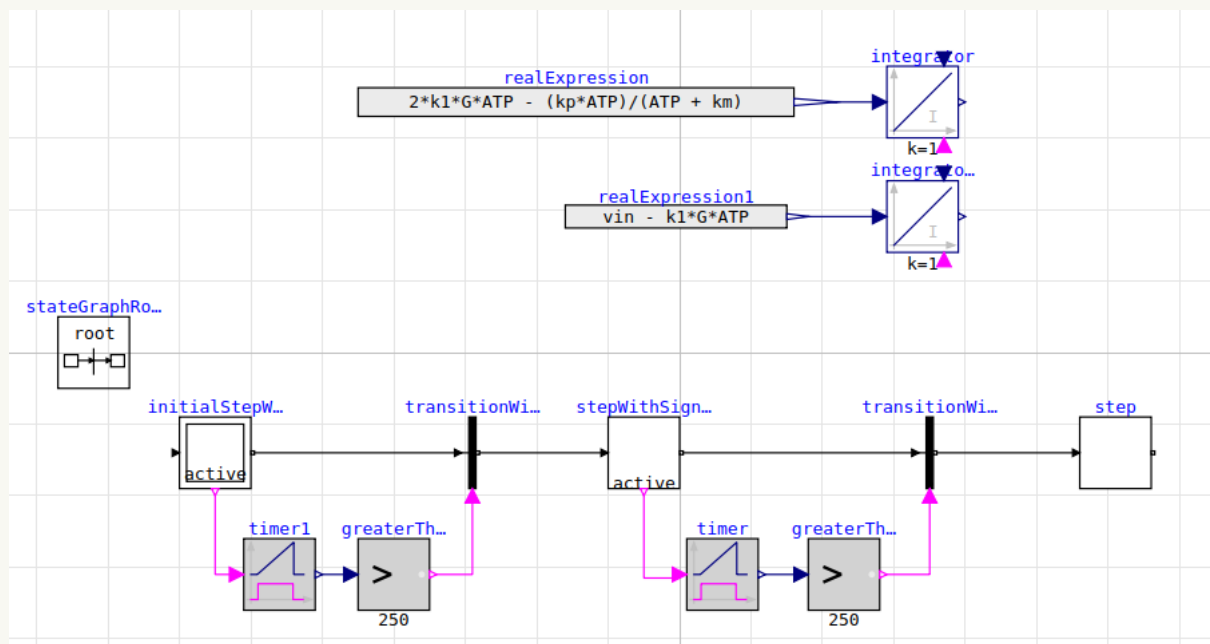


Imagem 3.6: Modelo final em blocos

Por fim, executamos o modelo cyber-físico resultante por 800 segundos – configurando novamente o tempo de simulação como já mostrado – obtendo o seguinte gráfico como resultado:



Imagem 3.7: Resultado 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. O sistema cyber-físico implementado nessa simulação possui como *input* duas coordenadas 2D, representando os pontos de origem e destino do cabeçote da impressora, além de dois pontos para representar a posição inicial dos cabeçotes, 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.

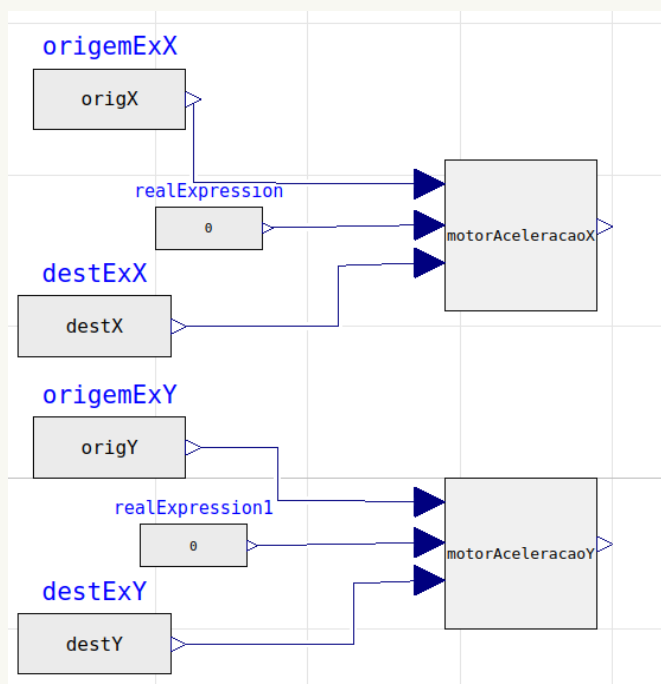


Imagem 4.1: Parte física do modelo da impressora 2D

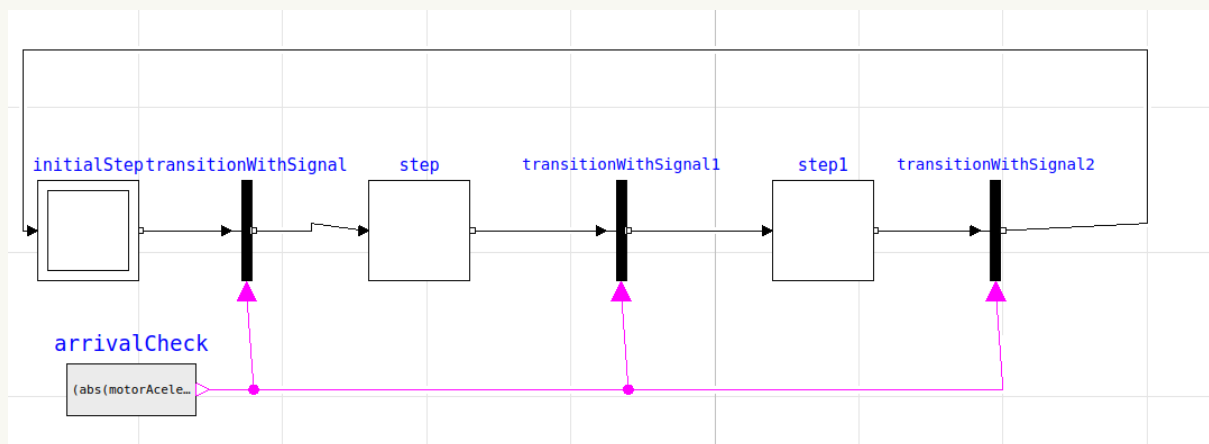


Imagem 4.2: Parte discreta do modelo da impressora 2D

Como demonstrado na imagem 4.1, o modelo físico da impressora consiste apenas em fornecer as coordenadas para seus respectivos motores, enquanto seu modelo discreto, representado na figura 4.2 consiste de uma máquina de estados cuja cada passo atualiza os pontos de origem e destino passados para os motores, por meio da utilização de equações utilizadas dentro do bloco de código *when*.

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 o bloco *realExpression*, que pode ser encontrado pela ferramenta de busca *Libraries Browser*, ou dentro da seção *Libraries* em *Modelica > Blocks > Sources*, e um novo sub-bloco a ser criado chamado *motorAceleracao*, inicialmente vazio, ao clicar com o botão direito do mouse sobre o modelo da Impressora2d recém criado e escolher a opção *new modelica class, respectivamente* – conectamos os blocos seguindo a imagem 4.1.

Para modelagem da parte discreta do modelo, representada na imagem 4.2, são necessários os blocos *initialStep*, *step* e *transition with signal*, disponíveis em *modelica > StateGraph*, um bloco *BooleanExpression* também é necessário e pode ser encontrado em *modelica > Blocks > Sources*, conectar esses componentes como representado na figura 4.2. O fator que aciona a transição de estados é a chegada do cabeçote ao seus pontos de destino, essa checagem é feita no bloco *arrivalCheck* que checa se ambos os cabeçotes estão nos seus pontos de destinos, dentro de uma margem de erro. Para realizar a implementação da lógica dessa máquina de estados, usamos as variáveis declaradas no início do código do modelo (imagem 4.3) e especificamos o output das transições por meio do bloco *when*, demonstrado na imagem 4.4.

```
Real destX(start = 12);  
Real destY(start = -6);  
Real origX(start = 0);  
Real origY(start = 0);
```

Imagem 4.3: Declaração de variáveis no início do modelo



```

equation
when initialStep.active then
    destX = 12.0;
    destY = -6.0;
    origX = 0;
    origY = 0;
elsewhen step.active then
    destX = 15;
    destY = 10;
    origX = 12;
    origY = -6;
elsewhen step1.active then
    destX = 0;
    destY = 0;
    origX = 15;
    origY = 10;
end when;

```

Imagem 4.4: Código utilizado na parte discreta do modelo para controlar as entradas dos motores.

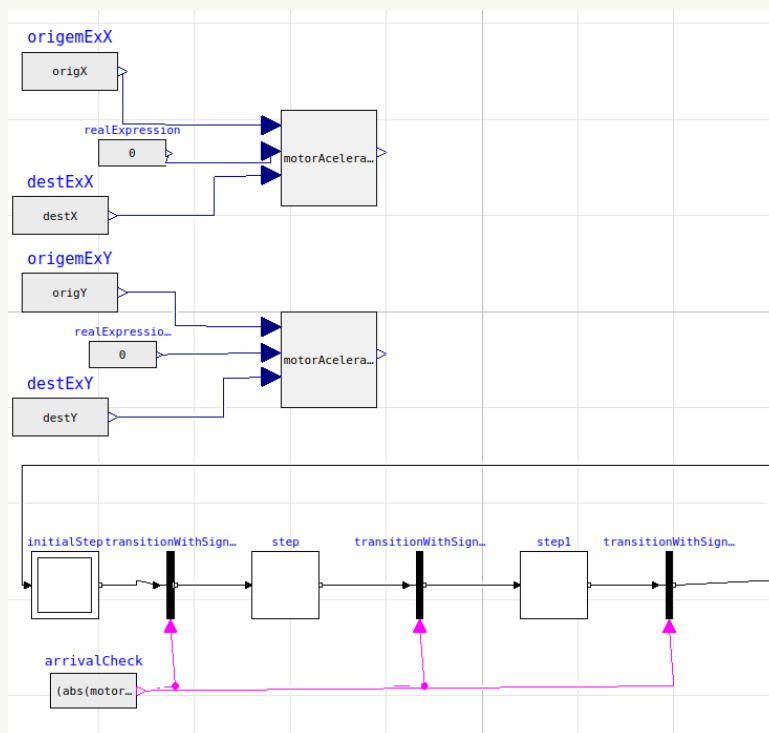


Imagem 4.4: Visão completa do modelo de alto nível da impressora

Para modelar os motores da impressora foram usados os blocos *add*, *realExpression*, *booleanExpression* e *integrator* (o bloco integrator pode ser encontrado em *modelica > blocks > continuous*, enquanto o bloco *add* em *modelica > Blocks > Math*, além de um novo sub-modelo *discreteController* para o controle da aceleração do seu cabeçote, para criar os inputs e outputs deste submodelo, os blocos *realInput* e *realOutput* são utilizados (disponíveis em *modelica > Blocks > interface*), esses blocos devem serem conectados segundo a figura 4.6. A velocidade do motor é obtida pela integração da aceleração atual retirada da saída

do sub-bloco *discreteController*, ela é consequentemente integrada com o intuito de fornecer a mudança de posição do cabeçote ao decorrer do tempo, que é consequentemente adicionada à posição inicial do cabeçote, obtendo sua nova posição.

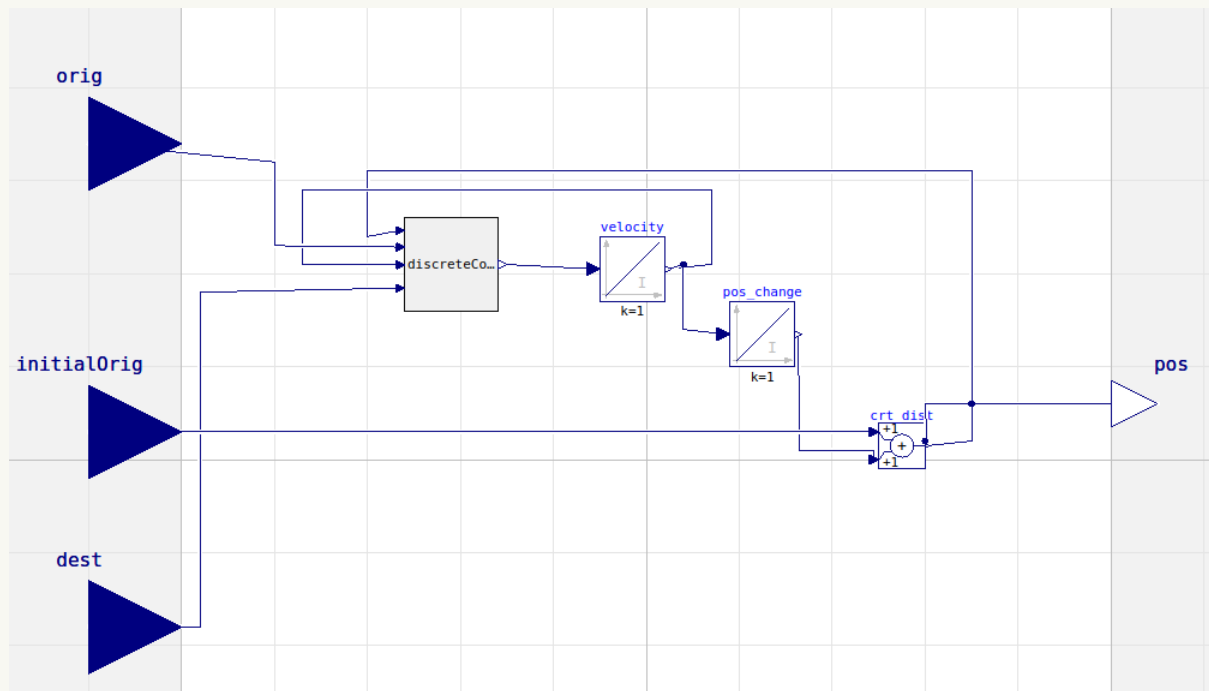


Imagem 4.6: Modelo do motor da impressora 2D.

A parte mais complexa deste modelo reside dentro do sub-bloco discreto *discreteController* que controla quando o motor deve acelerar, frear ou manter sua velocidade atual (imagem 4.7). O caso base do controlador passa por quatro estados distintos:

1. Esperar por uma nova entrada (aceleração = 0)
2. Acelerar até sua velocidade máxima (aceleração = Max)
3. Manter essa velocidade máxima (aceleração = 0)
4. Frear em tempo o suficiente para atingir velocidade igual a zero ao atingir seu destino (aceleração = -Max)

Desse modo, para controlar as transições entre os estados, as seguintes expressões foram utilizadas:

- Transição 1->2: Checa se o destino atual é diferente do destino utilizado no último ciclo do motor.
- Transição 2->3: Checa se a velocidade máxima foi atingida.

- Transição 3->4: Checa se a distância que falta para atingir o destino é menor ou igual a distância necessária para frear completamente o cabeçote.

Essa distância é obtida pela fórmula:

$$tempoParaVelocidadeMáxima = \frac{velocidadeMáxima}{aceleraçãoMáxima}$$

$$distânciaParaVelocidadeMáxima = \frac{tempoParaVelocidadeMáxima^2 * aceleraçãoMáxima}{2}$$

- Transição 4->1: Checa se o cabeçote chegou ao seu destino.

Assim como na máquina de estado presente no modelo de alto nível da impressora 2D, o controle do output de cada estado também é controlado por um bloco de código *when* em conjunto com os blocos *step initialStep* e *transitionWithSignal*. Uma checagem também é feita utilizando o bloco *Switch* (disponível em *modelica > Blocks > Logical*) para mudar o sinal da aceleração caso o destino seja menor que a origem. Para implementar a lógica de cada transição, blocos *booleanExpression* são utilizados para cada expressão descrita.

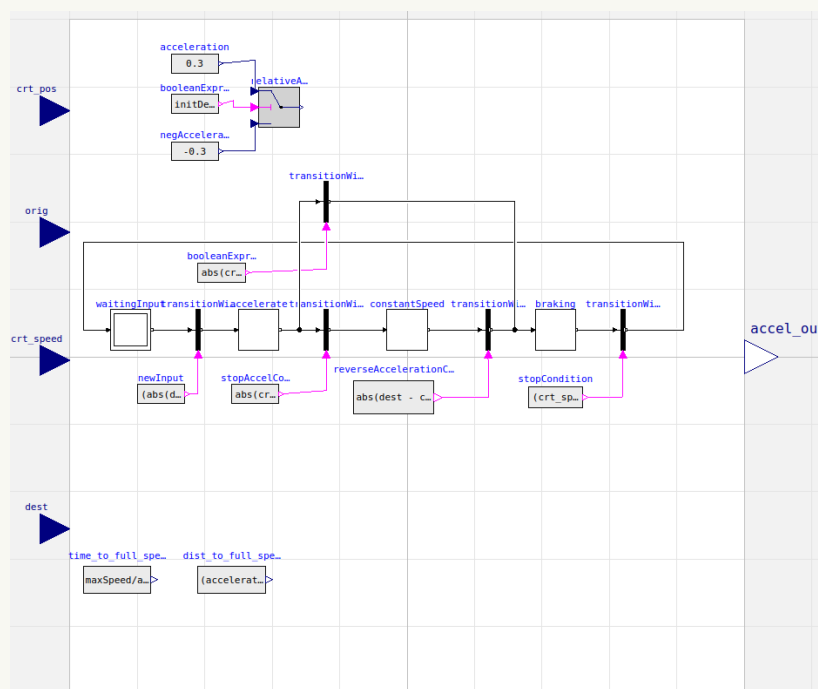


Imagem 4.7: Modelo discreto que controla a aceleração do cabeçote.

```

equation
  when accelerate.active then
    accel_out = relativeAccel.y;
    initDest = dest;
    initOrig = orig;
  elseif constantSpeed.active then
    accel_out = 0.0;
    initDest = initDest;
    initOrig = initOrig;
  elseif braking.active then
    accel_out = -1*relativeAccel.y;
    initDest = initDest;
    initOrig = initOrig;
  elseif waitingInput.active then
    accel_out = 0.0;
    initDest = initDest;
    initOrig = initOrig;
  end when;

```

```

parameter Real Error = 0.001;
parameter Real maxSpeed = 1.0;
Real initDest(start = 19999.0);
Real initOrig(start = 19999.0);

```

Imagem 4.8: Código de controle do bloco “when” utilizado na máquina de estados e declaração das variáveis utilizadas.

Para controlar casos em que a distância necessária para atingir a velocidade máxima e logo depois desacelerar até zero é maior do que a distância entre a origem e o destino do cabeçote, uma outra transição entre os estados 2->4 é necessária. Ela possui como gatilho para sua ativação uma expressão que checa caso o cabeçote já tenha atravessado pelo menos metade do percurso entre sua origem e destino e ainda esteja acelerando, quando ativada o cabeçote começa imediatamente a desacelerar, atingindo seu destino quando sua velocidade chega a zero.

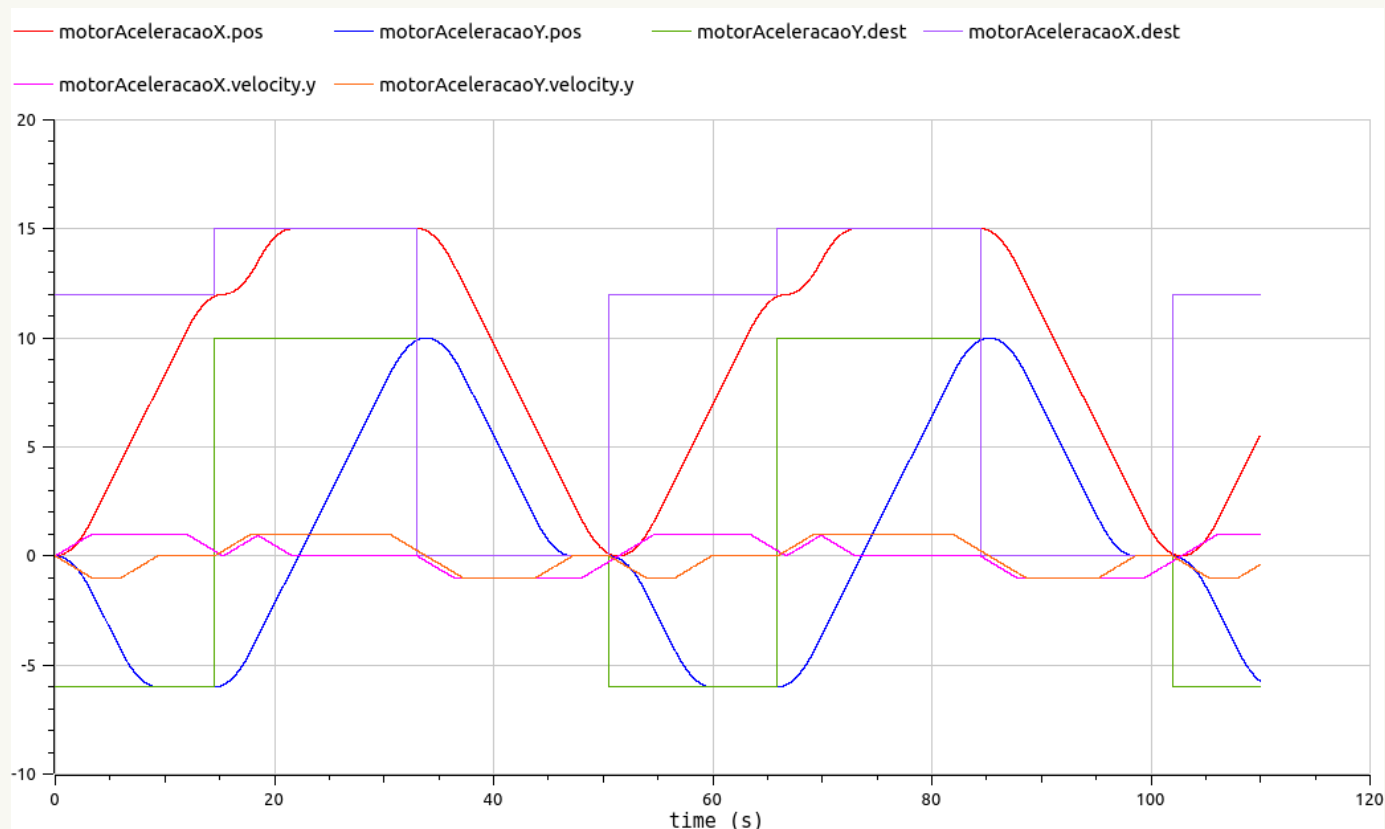


Imagem 4.8: Gráfico resultante da simulação da impressão

A simulação representada na figura 4.8 demonstra a posição dos cabeçotes, suas velocidades e destinos ao decorrer de 120 segundos, com velocidade máxima sendo igual a 1cm/s e aceleração igual a 0.3 cm/s<sup>2</sup>, a simulação começa no ponto (0,0) e atinge os pontos (12,-6) e (15,10), antes de voltar aos pontos (0, 0) e refazer o percurso novamente. Para realizar essa simulação, as opções de simulação, disponíveis em *simulation->simulation setup* foram:

- Start time = 0 secs
- End time = 110 secs
- Interval = 0.001 secs

O intervalo teve de ser diminuído para ser compatível com a constante de erro de 0.001 colocada no modelo.

O gráfico deixa evidente o funcionamento correto do modelo, com os cabeçotes acelerando, mantendo sua velocidade máxima e freando, atingindo seus destinos quando suas velocidades atingem zero. Também é possível verificar o funcionamento de casos onde a velocidade máxima não pode ser atingida, como o cabeçote se movendo no eixo x entre os pontos 12 e 15.

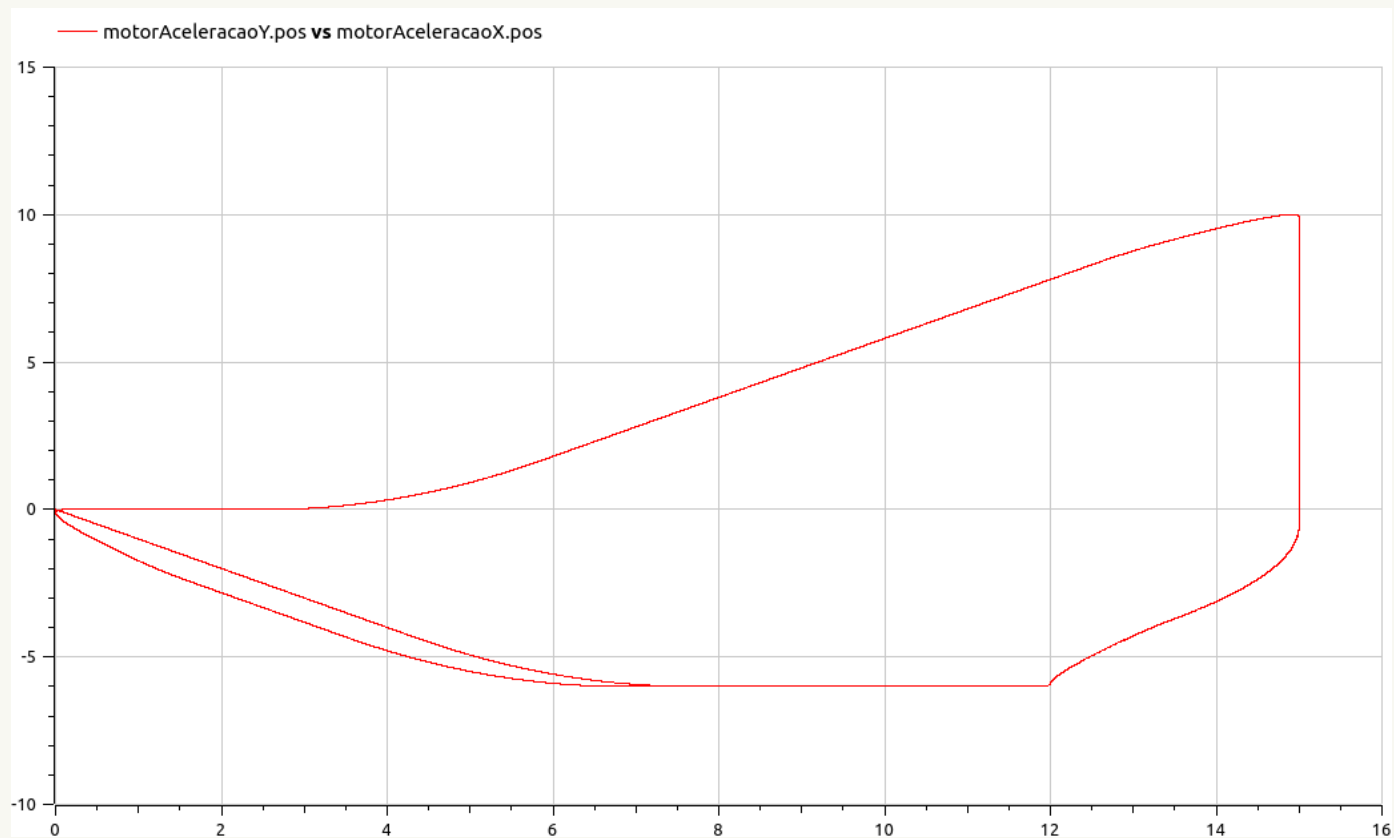


Imagem 4.9: Caminho traçado pelo cabeçote entre os pontos (0, 0),(12,-6),(15,10),(0,0) completando um pouco mais de dois ciclos em 110 segundos.

## Considerações finais

Como primeira parte do trabalho foram produzidos os modelos físicos de cada um dos exemplos, isso possibilitou a melhor compreensão de como o OpenModelica funciona e de como usá-lo. Além disso, modelar sistemas físicos também trouxe um maior entendimento de como esses sistemas funcionam e como representá-los matematicamente.

Já num segundo momento foram produzidos os modelos lógicos, e foi feita a integração dos modelos lógicos e físicos de cada exemplo, produzindo ao final os modelos cyber-físicos. Mais familiarizados com o OpenModelica, a segunda fase do trabalho foi mais simples, e mostrou o quão poderosa a ferramenta é, pois a integração de diferentes sistemas se fez muito simples e intuitiva.