INSTITUTO FEDERAL DE EDUCAÇÃO CIÊNCIA E TECNOLOGIA DE MINAS GERAIS - CAMPUS FORMIGA BACHARELADO EM ENGENHARIA ELÉTRICA

Pedro Ivo de Oliveira Tironi

DESIGN DE SISTEMAS REATIVOS COM STATECHARTS: um tutorial à ferramenta Stateflow

Formiga - MG

2021

PEDRO IVO DE OLIVEIRA TIRONI

DESIGN DE SISTEMAS REATIVOS COM STATECHARTS: um tutorial à ferramenta Stateflow

Trabalho de conclusão de curso apresentado ao Curso Bacharelado em Engenharia Elétrica do Instituto Federal de Minas Gerais - *Campus* Formiga como requisito para obtenção do título de bacharel em Engenharia Elétrica.

Orientador: Dr. Gustavo Lobato Campos

Coorientador: Dr. Patrick Santos de Oliveira

Formiga - MG

2021

Tironi, Pedro Ivo de Oliveira.
T597d Design de sistemas reativos com statecharts: um tutorial à ferramenta Stateflow / Pedro Ivo de Oliveira Tironi -- Formiga : IFMG, 2021.
124p. : il.

Orientador: Prof. Dr. Gustavo Lobato Campos
Coorientador: Prof. Dr. Patrick Santos de Oliveira
Trabalho de Conclusão de Curso – Instituto Federal de Educação, Ciência e Tecnologia de Minas Gerais – *Campus* Formiga.

1. Sistemas reativos. 2. Stateflow. 3. Statecharts. 4. Tutorial. 5. Aprendizagem ativa. I. Campos, Gustavo Lobato. II. Oliveira, Patrick Santos de. III. Título.

CDD 621.3

Ficha catalográfica elaborada pela Bibliotecária Msc. Simoni Júlia da Silveira

PEDRO IVO DE OLIVEIRA TIRONI

DESIGN DE SISTEMAS REATIVOS COM STATECHARTS: UM TUTORIAL A FERRAMENTA STATEFLOW

Trabalho de Conclusão de Curso apresentado ao Curso de Engenharia Elétrica do Instituto Federal de Minas Gerais como requisito para obtenção do Título de Bacharel em Engenharia Elétrica.

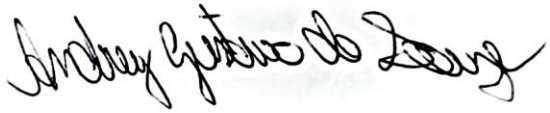
Avaliado em: 07 de outubro de 2021.

Nota: 95 pontos

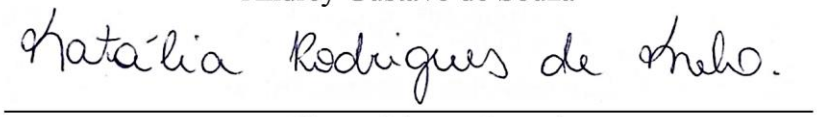
BANCA EXAMINADORA

Prof. Dr. Gustavo Lobato Campos

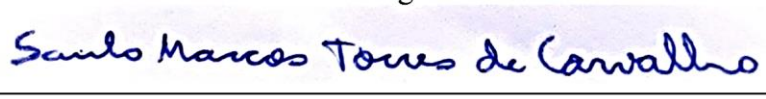
Prof. Dr. Patrick Santos de Oliveira



Andrey Gustavo de Souza



Natália Rodrigues de Melo



Saulo Marcos Torres de Carvalho

AGRADECIMENTOS

Aos meus pais, Marcelino e Elcineia, e minha irmã, Anna Paula, por serem os maiores motivadores e apoiadores desta trajetória.

Ao meu avô Zezinho e minha avó Cilinha, por sua forma especial de mostrar carinho.

Ao meu orientador e amigo Gustavo, por toda paciência e ensinamentos passados nestes últimos anos.

Aos meus amigos conterrâneos Claudienses, por sempre estarem presentes e suprirem meu coração de energia, e em especial, Mariana, pelo auxílio e correção deste trabalho.

A todos que moraram comigo em Formiga, por serem minha nova família ao mudar de cidade, em especial: Guilherme, Paulo, João Henrique e Otoniel.

Aos meus amigos de graduação, pela amizade e companheirismo nesta jornada.

Aos participantes da banca: Andrey, Natália, Saulo e aos professores: Patrick e Mário Luiz, por todas as sugestões e dicas apresentadas.

A todos que colaboraram em minha jornada.

“Seja você quem for, ..., tenha sempre como meta:
muita força, muita determinação e sempre faça tudo
com muito amor e com muita fé em Deus, que um dia
você chega lá. De alguma maneira você chega lá.”

Ayrton Senna

RESUMO

O desenvolvimento tecnológico promove a inserção gradativa de dispositivos inteligentes nos ambientes industriais. Estes, usualmente são sistemas embarcados com objetivo de automatizar processos, realizar a comunicação entre máquinas e fornecer dados essenciais para o monitoramento dos operadores. Logo, o presente trabalho tem como finalidade apresentar uma ferramenta computacional para auxílio ou alternativa ao desenvolvimento de sistemas embarcados. Fundamentado em trazer uma nova visão ao modo de programação convencional (*bottom-up*) pela programação por "*statecharts*" (forma gráfica/visual) aliado aos sistemas reativos. Apresenta-se este trabalho em forma de tutorial, a fim de capacitar o leitor por meio de Aprendizagem Ativa com exemplos didáticos pelo software MATLAB Simulink com toolbox Stateflow da MathWorks. Para isto, apresenta-se o Stateflow com seus menus e configurações. Fazem parte deste trabalho, exemplos educacionais para exercitar conceitos e demonstrar a aplicação das "*statecharts*" dentro deste princípio. Por fim, são feitas considerações de tecnologias que fazem uso dos conceitos apresentados, a fim de justificar a importância no tema inserido no desenvolvimento industrial, como por exemplo no setor automotivo, vinculado à elaboração de funções veiculares. Além disso, há uma pesquisa realizada com alunos de diferentes níveis sobre os tutoriais apresentados neste trabalho, como forma a validar os procedimentos expostos. Portanto, o texto apresentado busca oferecer ao leitor uma alternativa ao desenvolvimento convencional de sistemas reativos (com foco em sistemas embarcados) no intuito de contornar as principais dificuldades existentes na programação de microcontroladores por meio da Aprendizagem Ativa do software Stateflow, assim como complementar sua formação acadêmica.

Palavras chave: Sistemas reativos. Stateflow. Statecharts. Tutorial. Aprendizagem Ativa.

ABSTRACT

Technological development promotes the gradual insertion of smart devices in industrial environments. These are usually embedded systems in order to automate processes, carry out communication between machines and provide essential data for monitoring operators. Therefore, this work aims to present a computational tool to aid or alternative to the development of embedded systems. Based on bringing a new vision to the conventional programming mode (bottom-up) by programming by "statecharts" (graphic/visual form) combined with reactive systems. This work is presented in the form of a tutorial, in order to enable the reader through Active Learning with didactic examples by MATLAB Simulink software with MathWorks' Stateflow toolbox. For this, Stateflow is presented with its menus and configurations. This work includes educational examples to exercise concepts and demonstrate the application of "statecharts" within this principle. Finally, considerations are made of technologies that make use of the concepts presented, in order to justify the importance of the theme inserted in industrial development, such as in the automotive sector, linked to the development of vehicle functions. In addition, there is a survey carried out with students of different levels about the tutorials presented in this work, as a way to validate the exposed procedures. Therefore, the text presented seeks to offer the reader an alternative to the conventional development of reactive systems (focusing on embedded systems) in order to overcome the main difficulties in microcontroller programming through Active Learning of Stateflow software, as well as to complement their training academic.

Keywords: *Reactive systems. Stateflow. Statecharts. Tutorial. Active learning.*

LISTA DE ILUSTRAÇÕES

SUMÁRIO

1. INTRODUÇÃO

O desenvolvimento tecnológico no setor industrial está aliado principalmente a evolução de controladores e dispositivos inteligentes inseridos em seu ambiente (NEME, 2017). Atualmente diversos trabalhos manuais e repetitivos vêm sendo substituídos por máquinas automatizadas, fato que não se restringe mais as grandes empresas (EHL, 2019).

Aliado a isto, a programação de sistemas embarcados torna-se uma habilidade essencial para alunos e entusiastas das áreas de engenharia elétrica, eletrônica, assim como da ciência da computação, dentre outras. Visto que, para o desenvolvimento de dispositivos inteligentes, é necessário domínio da programação destes controladores. E no processo de aprendizagem de programação, item essencial para o desenvolvimento destes sistemas automatizados, é necessário desenvolvimento de competências e habilidades (técnicas e lógicas) (CILLIERS, CALITZ e GREYLING, 2005). Dentre estas competências destacam-se principalmente um elevado nível de abstração e organização de ideias, generalização e pensamento crítico (GOMES, AREIAS, *et al.*, 2008), (CILLIERS, CALITZ e GREYLING, 2005), (BROWN, 1988), (MORAIS, MENDES e OSÓRIO, 2020). Porém, demonstra-se em estudos como de (GOMES, AREIAS, *et al.*, 2008) que durante a fase de aprendizagem inicial surgem problemas focados nestes tópicos citados. Portanto, os principais obstáculos encontrados pelos alunos são: conceitos abstratos de programação, estruturas de controle e lógica, e na resolução de problemas reais (CILLIERS, CALITZ e GREYLING, 2005).

De modo geral as linguagens de programação para um usuário não familiarizado detêm uma sintaxe complexa (BROWN, 1988). Uma vez que são desenvolvidas para uso profissional com detalhes sintáticos e características especificas, onde exige-se que estudantes trabalhem tanto na resolução do problema, quanto na aprendizagem das regras sintáticas da linguagem (GOMES, AREIAS, *et al.*, 2008).

Alicerçado a este contexto, diversas técnicas têm sido desenvolvidas com o intuito de acelerar a inércia inicial apresentada no aprendizado das linguagens de programação (PIANO, 2019). Neste tópico, cita-se diversos sistemas que recorrem a representações visuais dos algoritmos, como por exemplo: Xtango, Trackla2, BALSAII, BACII e BlueJ (GOMES, AREIAS, *et al.*, 2008). Sendo que todos estes são baseados em metodologias gráficas, ou até mesmo complemento das textuais (GOMES, AREIAS, *et al.*, 2008), como por exemplo as *statecharts* utilizadas neste trabalho.

Portanto, este trabalho busca expor ao leitor uma ferramenta baseada em programação gráfica amplamente utilizada na indústria (PIANO, 2019), (NEME, 2017). Esta

ferramenta que se baseia no conceito gráfico de *statecharts* e será trabalhada pelo *software* da MathWorks dentro da plataforma do Stateflow. Ressalta-se ainda que este TCC (trabalho de conclusão de curso) teve foco na metodologia de Aprendizagem Ativa, em que por meio de tutoriais o leitor se torna protagonista do seu próprio aprendizado. Além de aliar tal conteúdo ao desenvolvimento de aplicações práticas, nas quais o leitor sinta-se familiarizado em sistemas que são cotidianos para possibilitar um aprendizado acelerado.

1.1. Justificativa

O presente trabalho justifica-se em virtude da forte inserção de sistemas embarcados na indústria, para executar tarefas como automatizar processos, interligar máquinas e coletar dados (ROCHEL, 2020). Diante deste fato, os tutoriais propostos como resultado deste trabalho têm como intuito apresentar alternativa para as dificuldades existentes no processo de modelagem de sistemas com o *software* Stateflow da MathWorks. Isso em virtude da escassez de materiais na língua portuguesa sobre este *software*. Além de interligar o tema dos roteiros de aprendizado ao tema atual que são os sistemas embarcados área de atuação de engenheiros eletricistas, eletrônicos e cientistas da computação. São desenvolvidos sistemas reativos pois neste trabalho não são embarcados em microcontroladores, e sem perda de generalidade, visto que ambas definições são similares. Pois, um sistema reativo a aplicação executada é dependente do ambiente em que se situa e age/reage aos eventos externos, e é semelhante a definição de um sistema embarcado.

Por sua vez, o Stateflow é baseado em *statecharts* que oferecem ao usuário uma variedade de benefícios em comparação a outras formas de programação, além de ser tema atual em artigos e conferências da área ao redor do mundo. Fato demonstrado em diversas conferências de 2019 descritas na sequência. Conferência russa: *Ближайшая конференция HolyJS* (PIANO, 2019); Conferência húngara: *CSSConf BP* (KHOURSHID, 2019); Conferência israelita: *React Next* (KHOURSHID, 2019); na conferência italiana: *Reactjsday* (YOUSEFZADEH, 2019); Conferência americana: *Winter Simulation Conference* (MIERLO e VANGHELUWE, 2019). Assim como nos artigos (HAREL, MARELLY, *et al.*, 2020), (GRAICS, MOLNÁR, *et al.*, 2020) e (SCHULZ-ROSENGARTEN, SMYTH e MENDLER, 2019). No qual, todos citam aplicações e/ou a relevância das *statecharts* inseridas no mundo moderno da programação.

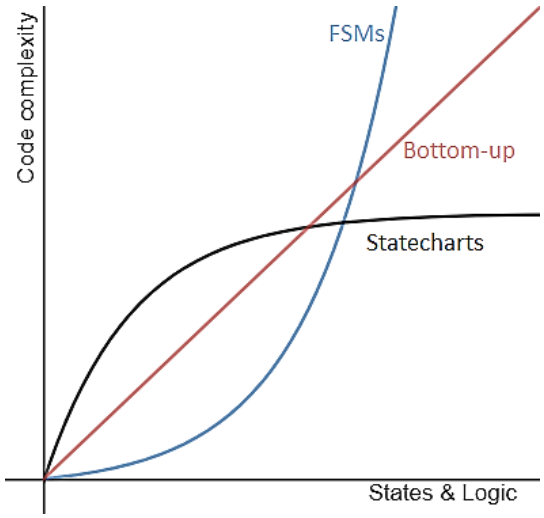
Logo, dentre os diversos aspectos que justificam o fato das *statecharts* serem tema atual cita-se:

- Facilidade ao compreender programação gráfica em comparação as outras formas de código. Baseia em métodos visuais de programação, sendo facilmente rascunhado e colocado em prática (KHOURSHID, 2019);
- Raciocínio simples, justifica-se por todo desenvolvimento baseia-se principalmente em estados e transições (PIANO, 2019);
- Facilidade na alteração de comportamento (HAREL, 1987). Os estados podem ser alterados separadamente de todo o programa;
- Teste independente de cada componente (HAREL, 1987). Devido à possibilidade de verificação rápida em cada estado, esta técnica é difundida e utilizada no padrão AUTOSAR que utiliza o Stateflow para desenvolvimento (IBM CORPORATION, 2018);
- Maior exploração de todos os estados existentes no sistema (KHOURSHID, 2019);
- Além de estudos demonstrarem que o código desenvolvido por *statecharts* possuem contagens de erros mais baixas que os códigos tradicionais (MOGENSEN, 2020).

Ao se confrontar com outras formas de programação, comparando-se a complexidade de um algoritmo à medida que a quantidade de lógicas aumenta, a programação por *statecharts* se demonstra a mais viável (PIANO, 2019). A Figura 1 apresenta graficamente tal fato, mas sem quantitativo matemático pressuposto que fator complexidade não é mensurável, pois distingue-se para diferentes programadores e algoritmos/problemas. No qual, separada por cores expõe que as FSMs (do inglês, máquinas de estados finitas) em azul, representam uma maior dificuldade para programas com grande número de estados e lógicas fato pela difícil visualização quando o número de estados cresce (conhecido como explosão de estados). A programação convencional *bottom-up* (vermelho) aumenta sua dificuldade linearmente de acordo com a complexidade do programa. Ou seja, quanto mais complexo mais linhas de código, porém toda a programação é feita linearmente. E as *statecharts* (preto) que apresentam uma dificuldade inicial maior que as outras formas de programação, porém a complexidade do código não aumenta consideravelmente a medida que são adicionados estados ou condições. Logo, enquanto as linhas no gráfico de FSMs (do inglês, *finite state machines*) e *bottom-up* crescem com taxa de variação positiva ou constante sempre que a complexidade aumenta, o oposto acontece com as *statecharts*. Visto que, a partir da dificuldade inicial de desenvolvimento a programação se torna cada vez mais simples. E esta é a linguagem utilizada pelo Stateflow da MathWorks e de outros *software*s como Xstate (JavaScript), Miros (Python),

USCXML (Lua, Java, C#, Python), e dentre outros que podem implementar as *statecharts* por meio de extensões, bibliotecas e complementos.

Figura 1 - Níveis de complexidade para diferentes formas de programação em comparação ao número crescente de estados e lógica do sistema.



Fonte: (PIANO, 2019)

Portanto, este trabalho propõe uma alternativa para ajustar especificamente este ponto de dificuldade inicial para programação por *statecharts*. Através de tutoriais para apresentar os conceitos essenciais para que o leitor possa romper a inércia inicial desta aprendizagem e esteja apto a desenvolver seus próprios sistemas.

1.2. Objetivo geral

Esta monografia tem como objetivo apresentar base referencial, assim como metodologia empregada para desenvolvimento de tutoriais da ferramenta Stateflow. Empregando para tal a metodologia baseada em problemas (PBL), aliado ao desenvolvimento de sistemas reativos. Isto como forma de superar a inércia inicial de programação com *statecharts* por meio da Aprendizagem Ativa.

1.3. Objetivos específicos

Dentre os objetivos específicos do trabalho cita-se:

a) Apresentar base teórica para que o leitor não familiarizado com teoria da computação compreenda assuntos essenciais para desenvolvimento dos exemplos apresentados no corpo do texto;
b) Apresentar de forma inicial os passos necessários para desenvolver simulação de sistemas discretos no MATLAB com Stateflow;
c) Apresentar conceitos de programação em Stateflow em sintonia ao desenvolvimento de sistemas reativos (com aplicações em sistemas embarcados);
d) Capacitar o leitor para a compreensão de algoritmos em Stateflow e também em linguagens que utilizem as *statecharts*;
e) Incentivar o leitor a desenvolver os algoritmos apresentados e outros de acordo com sua criatividade;
f) Apresentar uma forma de programação não convencional (*statecharts*) tal que o leitor supere a inércia inicial que dificulta o aprendizado no assunto;
g) Expor aplicações reais de desenvolvimento de sistemas embarcados com o uso de *statecharts*;
h) Demonstrar a integração do Stateflow com o ambiente Simulink que possibilita o usuário desde a geração de código, até a especificação de componentes dentro de um sistema.

1.4. Estrutura do Trabalho

O presente trabalho é dividido em oito capítulos e subcapítulos, com o intuito de oferecer uma melhor experiência durante a leitura do mesmo. Os capítulos são:

- Capítulo 2: contém a formulação teórica necessária para a compreensão adequada dos capítulos seguintes;
- Capítulo 3: apresentação dos principais menus, ferramentas e configurações do Stateflow sendo base essencial para compreensão dos tutoriais;
- Capítulo 4: apresentação dos tutoriais e metodologias, com a descrição do problema, especificações e as funções que são trabalhadas dentro do Stateflow separadamente para cada roteiro;
- Capítulo 5: expõe dados da validação dos roteiros com alunos que utilizaram os mesmos para aprendizado do Stateflow;
- Capitulo 6: apresenta a conclusão geral sobre o tema;

- Capítulo 7: apresenta as referências empregadas para desenvolvimento do trabalho;
- Capítulo 8: tem-se presente os anexos, onde encontram-se todos os roteiros didáticos, com os procedimentos, as imagens, e simulações dos exemplos de aplicações da ferramenta.

2. REVISÃO TEÓRICA

2.1. Histórico

A computação é uma ciência que busca por meio de algoritmos solucionar problemas, sejam estes lógicos ou matemáticos (DIANA, 2019). Para isto baseia-se na junção de várias áreas do conhecimento onde ressalta-se as áreas da lógica matemática, álgebra booleana (eletrônica digital) e a lógica de programação (SILVA, 2017).

Antes da consolidação desta área engenheiros eletricistas até a década de 30 desenvolviam circuitos eletrônicos para resolver problemas lógicos e matemáticos, mas tal fato sem qualquer rigor teórico e sem procedimentos (POUNDSTONE, 2005). Motivado a corrigir este fato o engenheiro eletricista Claude E. Shannon em sua tese de mestrado "*A Symbolic Analysis of Relay and Switching Circuits*" (EFFROS, 2017) aplicou a álgebra de Boole em conjuntos eletromecânicos para resolver problemas de lógica. Demonstrou-se a partir deste trabalho que uma aplicação elétrica entrelaçada a álgebra booleana poderia resolver qualquer problema de lógica (SHANNON, 1937). E tal ideia é o conceito básico de toda formulação computacional atual e também auxiliou no desenvolvimento da teoria da informação (GARDNER, 1987).

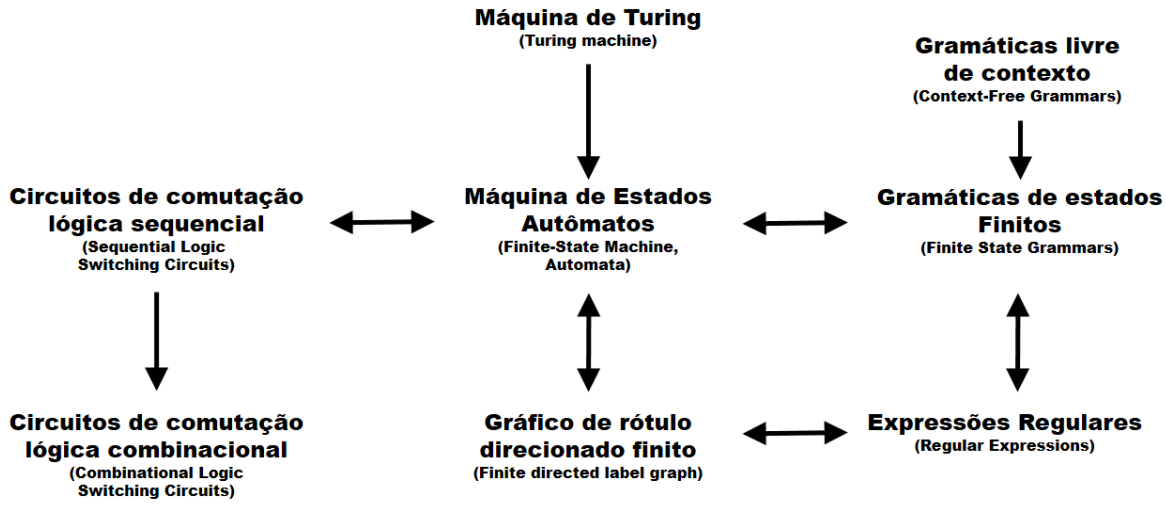
Iniciou-se então a teoria da computação motivada a determinar quais problemas podem ser computados para um dado modelo matemático de computador (SIPSER, 1996), (HOPCROFT, RAJEEV e ULLMAN, 2006). O presente trabalho tem como objetivo desenvolver aplicações em engenharia (sistemas reativos com foco em sistemas embarcados) de um modelo de computador chamado *statecharts*. Estas desenvolvidas por David Harel e são uma extensão de conceitos sobre as máquinas e diagramas de estados (HAREL, 1987). Motiva-se o uso deste modelo teórico por ser uma ferramenta matemática capaz de descrever um computador ou um circuito lógico. Isto por meio de uma formulação matemática, no qual suas aplicações são motivadas para representação de sistemas que mesmo com o aumento da complexidade os sistemas ainda se mantém simples e fáceis de compreender (SIPSER, 1996).

2.2. Fundamentos

As máquinas de estados podem modelar uma variada gama de problemas, sejam estes o processamento de editores de texto, compiladores, *design* de *hardware* e projeto de protocolos de comunicação (VIERA, 201-), (WINSKEL, 1993). Além das ciências exatas também estão inseridos como, por exemplo, na biologia para descrever sistemas neurológicos e em linguística para descrever as gramáticas das linguagens naturais (BELZER, HOLZMAN e KENT, 1975).

Uma analogia com a mecânica (ramo fundamental da física) pode justificar e auxiliar na compreensão do uso deste modelo em suas aplicações. Em mecânica newtoniana as velocidades são baixas e o tamanho dos objetos em trabalho é macroscópico, o que é um bom modelo para representar uma bola de basquete em movimento. Já para partículas do tamanho de um átomo movimentando-se a velocidades próximas a da luz este modelo se torna impreciso (GRIFFITHS, 2011). Logo, em computação existem diversos modelos matemáticos para representar computadores, e conforme apresenta a Figura 2 pode-se visualizar diversos destes. Desde modelos mais gerais e completos como a máquina de Turing até modelos mais restritivos e limitados como as máquinas de estados (VIERA, 201-).

Figura 2 - Representação de vários modelos computacionais e suas equivalências de acordo com suas potências/generalidades. As setas se movem na direção da potência restritiva. As setas bidirecionais mostram equivalências.



Fonte: Adaptado de (COQUAND, 2009).

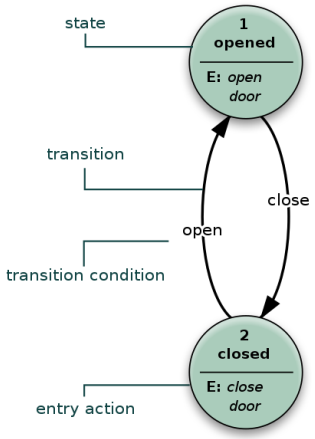
Portanto, cabe ao cientista ou engenheiro determinar qual modelo representa melhor para a formulação ou resolução de um problema. Neste trabalho, como dito anteriormente, serão abordados modelos simples que podem ser representados por modelos matemáticos mais restritivos como as máquinas de estados e *statecharts*.

Logo, restrito ao contexto de máquinas de estado existem dois tipos de máquinas, que são os transdutores (SIPSER, 1996) e reconhecedores (FRIEDL, 2006). O primeiro grupo são sistemas capazes de ler uma fita de entrada, e a partir desta ação descrever o comportamento de saídas (VIERA, 201-). O que relaciona com a descrição de um sistema embarcado (WHITE, 2012), que é um computador dedicado a interpretar entradas e controlar saídas. Ambas definições se aproximam com certo grau de similaridade e será o foco de estudo para aplicações neste trabalho. O segundo grupo avalia a aceitação de linguagens regulares (FRIEDL, 2006), e tem como saída estados de aceitação ou rejeição (SIPSER, 1996). Apesar das diversas aplicações, não será o foco ao desenvolver sistemas reativos neste trabalho.

2.3. Representação

O estudo de máquinas abstratas é motivado pela representação de máquinas concretas (máquinas reais e não somente teóricas), e nestas o comportamento de sistemas (SIPSER, 1996). Embora uma máquina de estados seja tecnicamente uma estrutura matemática é conveniente introduzir seus conceitos por meio de diagramas de estado, por demonstrarem uma maneira mais intuitiva de visualização. Conforme apresentado na Figura 3.

Figura 3 - Exemplo de diagrama de estados.



Fonte: Adaptado de (State diagram, 2020).

Logo, um diagrama de estados apesar de ser estrutura matemática diferente de um grafo dirigido (FEOFILOFF, 2017) tem sua representação equivalente, e por vezes similar. O grafo é matematicamente definido um conjunto não-vazio, cujos elementos são chamados vértices ou nós, e um conjunto de arestas (setas) que são pares ordenados (FOMIN, GENKIN e ITENBERG, 1996). Diversas são as formas gráficas de se representar um grafo (GUEDES, 2001) e se dois grafos mesmo que representados diferentemente mas possuem a mesma relação vértices e arestas, estes são ditos isomorfos (FOMIN, GENKIN e ITENBERG, 1996). Em uma máquina de estados, ao ser comparada com um grafo nomeia-se: o vértice do grafo como estado, e a aresta como transição (HOPCROFT, RAJEEV e ULLMAN, 2006).

2.3.1. Funcionamento

A partir do modelo de representação visual de uma máquina de estados é necessário delimitar o que ele pode ou não realizar, e como ele executa estes processos. Dentre os conceitos básicos para a compreensão do funcionamento descreve-se que uma máquina de estados finita tem as seguintes características:

a) Um estado é a descrição de uma condição que o sistema está apto a exercer (SIPSER, 1996). Por exemplo, em uma máquina de estados que representa o clima pode-se existir estado “chuvoso”, estado “ensolarado” e diversos outros para representar condições de existência climáticas;
b) Uma transição é uma ação a ser executada quando uma condição for satisfeita ou quando o evento é recebido (COQUAND, 2009), (EFFROS, 2017). Por exemplo, se uma máquina de estados se encontra no estado ensolarado e a condição “chover” torna-se verdadeira (começa a chover), logo o sistema irá transitar do estado ensolarado para o estado chuvoso. Ou seja, a transição é regida por uma condição, e se verdadeira acontece um evento em decorrência (WAGNER, 2005);
c) Sendo uma máquina determinística ela estará em somente um estado por vez, que é chamado de estado atual (YOUSEFZADEH, 2019);
d) Tem um estado inicial que é chamado de q_0 assim quando uma fita é lida na máquina este é o primeiro estado que ela estará. Uma analogia deste conceito é o *setup* inicial de um programa de computador (SIPSER, 1996);
e) Uma máquina de estados executa quando lhe é dada uma sequência de entradas em passos de tempo discretos (HOPCROFT, RAJEEV e ULLMAN, 2006);

f) Existem regras para movimentação entre os estados, chamadas funções de transição (BELZER, HOLZMAN e KENT, 1975);
g) As entradas são descritas em fitas (HOPCROFT, RAJEEV e ULLMAN, 2006);
h) Uma máquina de estados lê os símbolos da fita (palavra) de entrada, um após o outro, e faz a transição de estado para outro, de acordo com a função de transição, até a fita ser totalmente lida (HOPCROFT, RAJEEV e ULLMAN, 2006);
i) Uma vez que a palavra de entrada foi lida a máquina de estados deve parar (SIPSER, 1996);
j) O estado no qual a máquina de estados para é chamado de estado final (SIPSER, 1996);

Portanto, a partir desta descrição informal é possível entender o fluxo de funcionamento de uma máquina de estados, além de possibilitar sua descrição matemática.

2.3.2. Descrição matemática

Os tutoriais desenvolvidos neste trabalho baseiam-se no uso de *statecharts* (HAREL, 1987), porém para sua compreensão completa entende-se como necessário introduzir conceitos existentes em máquinas de estado.

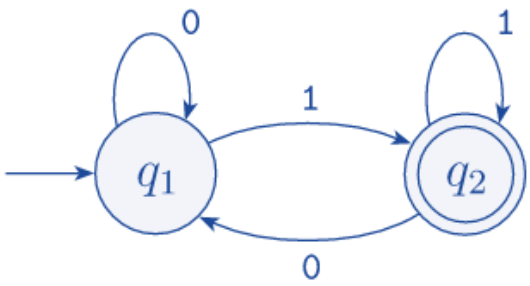
Esta estrutura matemática de uma máquina de estados é constituída por três entidades: estados (Q), transições (δ) e um alfabeto (Σ) (FRIEDL, 2006), (VIERA, 201-). Os dois primeiros itens foram representados anteriormente (Figura 3), e o alfabeto define-se como um conjunto finito de objetos.

Com isto, matematicamente uma máquina de estados (LEWIS e HARILAOS, 1981) é dado pela 5-upla $(Q, \Sigma, \delta, q_0, F)$. Tal que:

a) Q é o conjunto finito de estados;
b) Σ é o alfabeto;
c) $\delta: Q \times \Sigma \rightarrow Q$ é a função de transição;
d) $q_0 \in Q$ é o estado inicial (estado *default*);
e) $F \subseteq Q$ é o conjunto de estados de aceitação.

Os conceitos são exemplificados através de um exemplo. Seja o diagrama de estados M representado pela Figura 4.

Figura 4 - Diagrama de estados M.



Fonte: Adaptado de (SIPSER, 1996).

A vista disso, descreve-se formalmente este diagrama de estados M (LEWIS e HARILAOS, 1981). Ele é composto por 2 círculos que são os estados: $Q = \{q_1, q_2\}$ (SIPSER, 1996). O alfabeto deste diagrama é o conjunto de símbolos que definem as ações (SIPSER, 1996). As setas que apontam de um estado para outro são as transições, e os elementos sobre elas são os símbolos que indicam as condições/regras de mudança de estado (State diagram, 2020). Existe uma seta (aresta) que aponta de um lugar qualquer para o estado q_1, logo ele é o estado inicial (VIERA, 201-). O estado q_2 tem uma representação de dois círculos pois, ele é um estado final, ou seja, faz parte do conjunto dos estados de aceitação $F = \{q_2\}$ (SIPSER, 1996). Portanto:

a) $Q = \{q_1, q_2\}$;
b) $\Sigma = \{0,1\}$;
c) δ: $Q \times \Sigma$

δ: $Q \times \Sigma$	0	1
q_1	q_1	q_2
q_2	q_1	q_2

d) $q_0 = q_1$;
e) $F = \{q_2\}$.

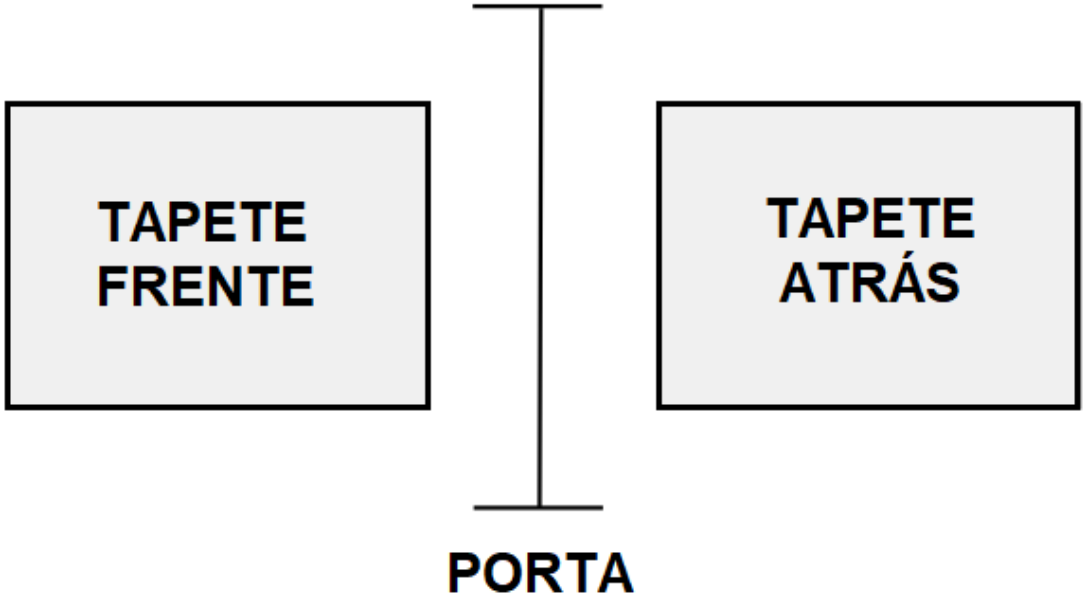
Logo, sendo L o conjunto de linguagens que M aceita então será dada por $L(M) = \{\omega | \omega \text{ termina com } 1\}$ (WINSKEL, 1993). Portanto, a partir desta 5-upla foi descrito um modelo matemático de uma máquina de estados, que abrange todos os elementos que o representam, esta descrição é útil já que descreve uma máquina de estados finita pois é equivalente ao modelo gráfico, e é mais precisa porque evita ambiguidades ou mesmo impressão de isomorfismos (SIPSER, 1996), (HOPCROFT, RAJEEV e ULLMAN, 2006).

2.4. Exemplo de Máquinas de Estado

A fim de ilustrar e familiarizar o leitor com conceitos básicos é apresentado um simples exemplo de um sistema presente em situações cotidianas.

Um sistema eletromecânico de portas automáticas é encontrado frequentemente em entradas e saídas de supermercados e *shoppings*, no qual abrem deslizando a medida que uma pessoa se aproxima. Esta geralmente possui um tapete a sua frente que detecta a presença de uma pessoa que está próxima a atravessar a porta. E um outro tapete localizado no lado oposto da porta de modo que o controlador deste sistema possa manter a porta aberta tempo suficiente que a pessoa venha a passar e não feche em algum momento inoportuno (SIPSER, 1996). Conforme ilustra a Figura 5 o sistema é representado por uma porta e os dois tapetes onde encontram-se os sensores.

Figura 5 - Ilustração do sistema de porta automática.



Fonte: Adaptado de (SIPSER, 1996).

Dessa maneira, existem dois estados possíveis para esta porta, aberta ou fechada. Define-se para este sistema, que a porta iniciará fechada e depois caso necessário execute transições para outro estado.

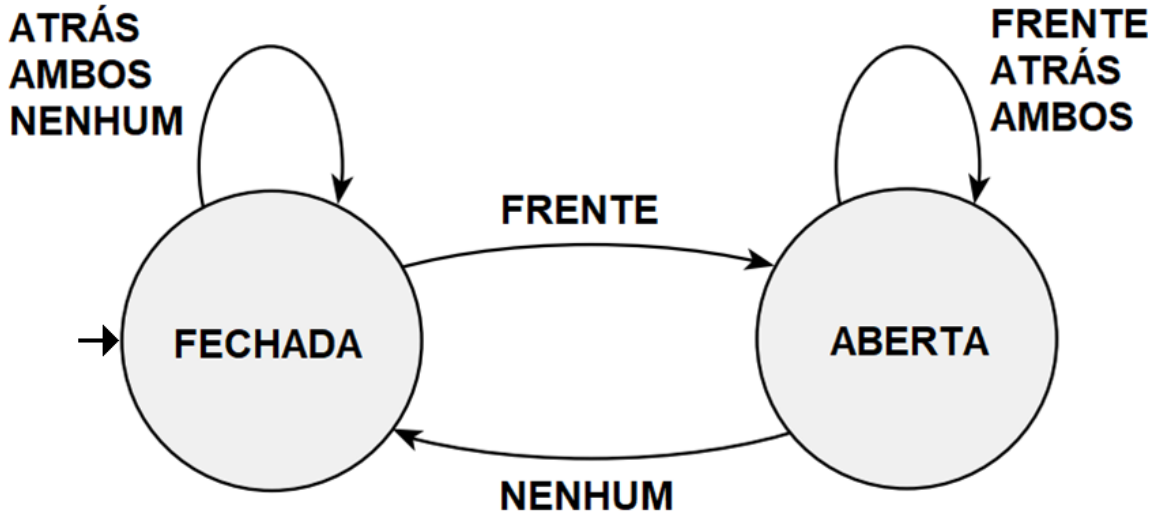
Para as transições é necessário verificar as entradas e interpretá-las para construir a máquina de estados. Para isto, determina-se quatro condições possíveis:

a) “Frente” – significa que uma pessoa está pisando no tapete da frente de porta;

b) “Atrás” – significa que uma pessoa está pisando no tapete atrás da porta;
c) “Ambos” – significa que duas pessoas estão pisando em ambos os tapetes existentes neste sistema;
d) “Nenhum” – significa que nenhuma pessoa está pisando em qualquer um dos tapetes.

O controlador atua neste sistema movendo a porta para o estado de aberta ou movendo para o estado de fechada, isto dependendo da entrada que ele recebe. Quando a porta se encontra no estado “Fechada” e recebe uma entrada “Nenhum” ou “Atrás” ele permanece com a porta no estado “Fechada”. Adicionalmente, para o estado “Fechada”, se a porta recebe a entrada “Ambos” ele deverá permanecer no mesmo estado (“Fechada”), pois ocorre o risco da porta atingir alguém sobre o tapete de trás. Porém, quando chega uma entrada “Frente” a porta irá se mover para o estado “Aberta”, pois uma pessoa quer realizar a passagem e não tem nenhuma outra pisando sobre o tapete do outro lado. No estado “Aberta” se a entrada for “Frente”, “Atrás” ou “Ambos” a porta irá permanecer no estado “Aberta” pois, existe uma pessoa que está realizando a passagem pela porta. Ainda no estado “Aberta”, se o estado recebe a entrada “Nenhum”, indica que a porta pode ser fechada, pois não tem risco de estar colidindo com nenhuma pessoa, e é realizada a transição do estado “Aberta” para o estado “Fechada”. Para ilustrar todo este sistema de transições e suas condições sobre os respectivos estados a Figura 6 representa o sistema elaborado.

Figura 6 - Representação da máquina de estados do sistema de porta automática.



Fonte: Adaptado de (SIPSER, 1996).

Para este exemplo, pode-se simular uma situação típica do controlador. Define-se as entradas sendo: “Frente”, “Atrás”, “Nenhum”, “Frente”, “Ambos”, “Nenhum”, “Atrás”, “Nenhum”. Para esta situação de entrada os estados são: “Fechada” (inicial), “Aberta”, “Aberta”, “Fechada”, “Aberta”, “Aberta”, “Fechada”, “Fechada”, “Fechada” (SIPSER, 1996).

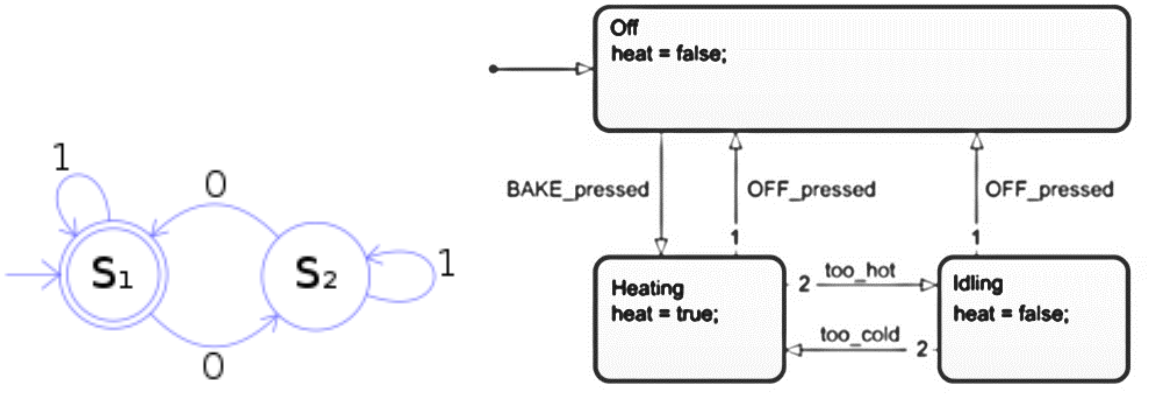
Este controlador de porta automático apresentado como uma máquina de estados é útil para representar os conceitos base com um exemplo cotidiano (SIPSER, 1996). No qual, é um computador que possui apenas um bit de memória capaz de gravar em qual estado o controlador se encontra (SIPSER, 1996). Para controladores com maiores necessidades de informação e maiores tarefas são desenvolvidos sistemas mais complexos, com diversos estados e transições, porém seguem os mesmos conceitos de um controlador básico conforme apresentado.

2.5. Máquinas de Estado de Mealy e de Moore

Dentre as representações da máquina de estados existem dois modelos clássicos chamados de Mealy e Moore (IBM CORPORATION, 2018), (KHOURSHID, 2019). Para entender a diferença entre estes modelos é necessário visualizar onde ocorrem os eventos (CAMPBELL, 201-). No caso de Mealy as ações/eventos são definidos nas transições (MEALY, 1955) e no caso de Moore esses estão definidos dentro dos estados (MOORE, 1956).

Uma máquina de estado que usa as entradas (CAMPBELL, 201-), para que suas saídas dependam de um estado é chamada de modelo de Moore (MOORE, 1956). Conforme expõe a Figura 7.

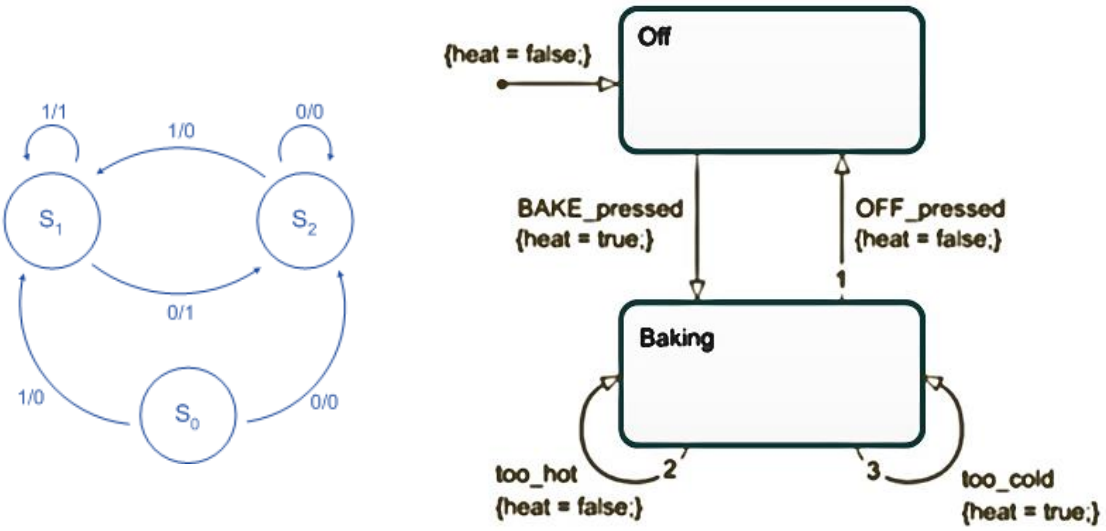
Figura 7 - Exemplo de máquina de Moore.



Fonte: (a) (State diagram, 2020) (b) (CAMPBELL, 201-).

Uma máquina de estado que usa apenas Ações de Entrada (CAMPBELL, 201-) de modo que a saída depende das entradas é chamada de máquina de Mealy (MEALY, 1955). Na Figura 8 é possível ver um exemplo de desenvolvimento de uma máquina de Mealy (CAMPBELL, 201-), no qual é apresentado um modelo de literatura e um modelo elaborado no MATLAB.

Figura 8 - Exemplo de máquina de Mealy.



Fonte: (a) (State diagram, 2020) (b) (CAMPBELL, 201-).

Ao realizar uma comparação entre as duas máquinas percebe-se que máquinas de estado de Mealy na prática são de implementação mais econômica, isto por meio do número de estados (WAGNER, 2005). Fato que se justifica, pois um modelo de Mealy (MEALY, 1955) para uma representação equivalente em uma máquina de Moore (MOORE, 1956) tem menos estados. Porém, uma máquina de Moore pode apresentar maior estabilidade (WAGNER, 2005) em sistemas complexos.

A escolha do modelo no desenvolvimento de um projeto fica a cargo da familiaridade do programador (WAGNER, 2005). E também do sistema que se busca representar, por exemplo quando se trata em sistemas com objetivo de representar um *design* de *hardware* utiliza-se modelo de Moore (MOORE, 1956). Na prática em grandes projetos frequentemente são utilizados modelos mistos (WAGNER, 2005). Para aprendizado neste trabalho serão apresentados programas elaborados pelos dois métodos, pois trata-se de exemplos simples e voltados para a aprendizagem do *software*.

2.6. *Statecharts* de Harel

David Harel através do seu artigo intitulado "*Statecharts: A visual formalism for complex systems*" apresenta uma extensão do formalismo convencional das máquinas e diagramas de estado (HAREL, 1987). Em que, se possibilitou uma revolução na elaboração (*design*) e especificação de sistemas discretos complexos, sistemas de tempo real (sistemas reativos), protocolos de comunicação e unidades de controle digital (HAREL, 1987). Seu uso em comparação aos diagramas clássicos de máquinas de estados produz representações mais simples e claras, visto suas particularidades adicionais (ANTUNES, 2018).

Basicamente a extensão proposta por Harel insere os seguintes elementos: *dealing*, *orthogonality* e *hierarchy* (HAREL, 1987). Sendo assim, possibilita compactar diagramas complexos, além da visualização em diversos níveis de detalhamento e abstração, e explorar especificações e detalhamentos individualmente (HAREL, 1987).

Para que os diagramas de estados sejam úteis e sua representação seja satisfatória é necessário que o mesmo possua as características que otimizam a elaboração de um sistema (ANTUNES, 2018). Basicamente uma *statechart* deve ser capaz de executar as seguintes tarefas:

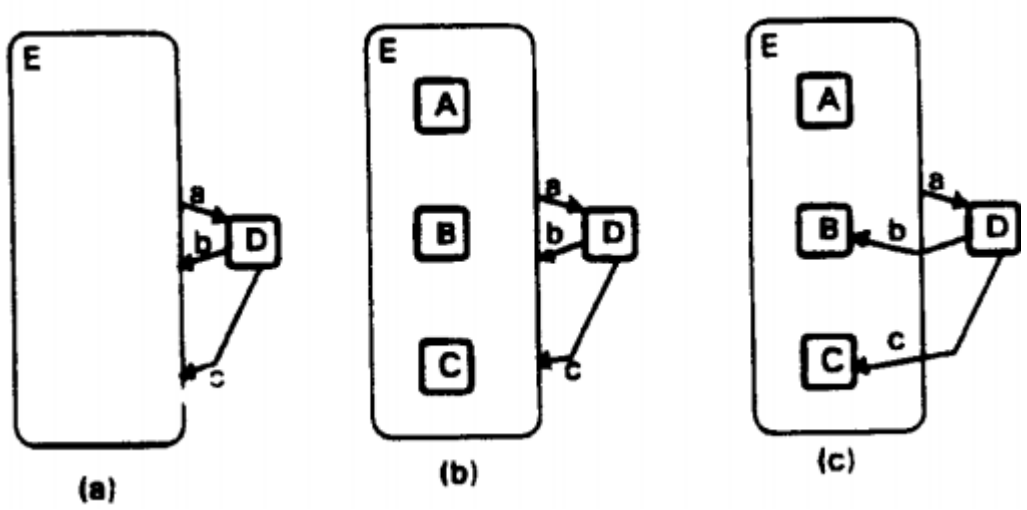
a) "Seja um modelo de *statechart* de aeronave, e esta estando em qualquer estado de operação, quando a alavanca amarela for acionada, o assento deverá ser ejetado" (HAREL, 1987). Logo, a declaração cita a necessidade da ferramenta possuir a capacidade de agrupar estados em um superestado para uma rápida transição;
b) "O estado da marcha de câmbio do carro é independente do seu sistema de freios, que também é independente do sistema AIRBAG" (HAREL, 1987). Dessa maneira, a ferramenta deve introduzir conceitos de independência e ortogonalidade, de modo que todos os estados tenham fácil acesso e operem em paralelo;
c) "Em um *statechart* que representa um rádio digital, quando o botão de seleção for pressionado o sistema deve entrar no modo de seleção" (HAREL, 1987). Deste modo, sugere que é necessário de transições genéricas além uma simples seta identificando um estado, fato para não existir a explosão de estados que acontece frequentemente nas FSMs;
d) "Em uma *statechart* que representa um relógio digital, o modo de exibição deve demonstrar a exibição da hora, a exibição da data e a exibição do cronômetro" (HAREL, 1987). A partir disto, demonstra-se a possibilidade para se refinar e

aumentar o nível de detalhamento dos estados dentro de um estado maior que contém diferentes funcionalidades.

2.6.1. Clustering

O conceito de *clustering* se refere ao mecanismo capaz de abranger estados semelhantes em um superestado (HAREL, 1987). A Figura 9 representa uma configuração de *clustering* no qual um estado E com características diferentes é transformado em um superestado com os estados interiores A, B e C e que são mutuamente exclusivos entre si (OR-exclusivo). Consequentemente, suas transições em comum podem ser diretamente relacionadas com um estado externo que é o estado D.

Figura 9 - Conceito de *clustering* e superestado.



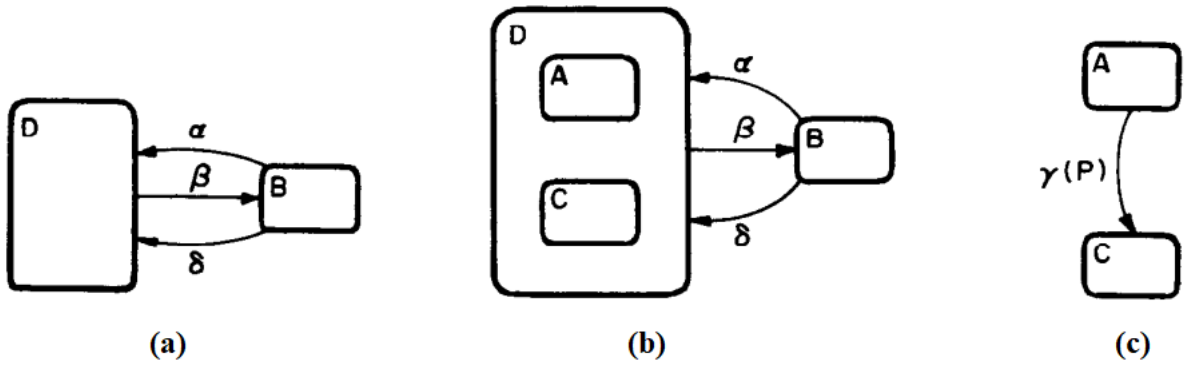
Fonte: (ANTUNES, 2018).

2.6.2. Refinamento

A partir de um sistema com *clustering* que abrange várias situações semelhantes dentro de um estado existe o processo inverso que é o refinamento (HAREL, MARELLY, *et al.*, 2020). Esta parte do fato que de um estado geral ou mais abrangente é possível descrever o sistema dentro de estado e transições menores até que o sistema esteja detalhado por completo (HAREL, 1987). É possível que dentro de um superestado existam superestados inclusos em grupos que minimizam e hierarquicamente detalham todas as ações necessárias (ANTUNES, 2018). Conforme apresenta a Figura 10, este mecanismo permite ao programador analisar e

melhorar o funcionamento de cada particularidade do sistema especificado, no qual possibilita uma exploração maior dos estados.

Figura 10 - Refinamento dos estados.



Fonte: (HAREL, 1987).

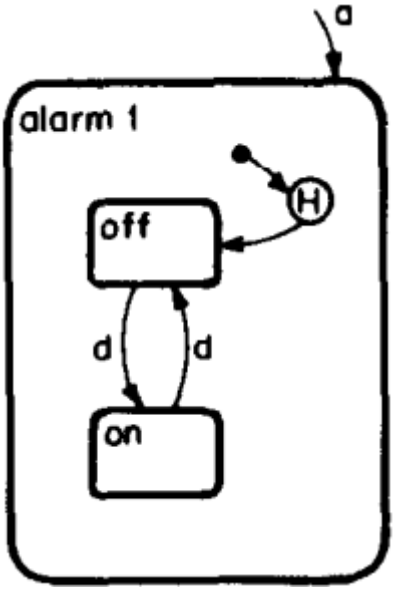
Alinhado com o processo de *clustering* o refinamento encaixa-se com o conceito de hierarquia (State diagram, 2020). O processo lidado no contexto de subestado está em um nível de hierarquia conceitualmente menor (ANTUNES, 2018). O que possibilita diferentes níveis de abstração para o mesmo sistema e capaz de lidar com complexidades, pois é comum ignorar (abstrair) partes de um sistema para visualizações gerais (HAREL, 1987). Logo, são um mecanismo para ocultar detalhes internos, visto que o programador, pode aumentar ou diminuir o zoom para maior detalhamento de uma especificação (ANTUNES, 2018).

2.6.3. Estado default

Assim como as máquinas de estado que apresentam um estado inicial do sistema, as *statecharts* possuem um mecanismo nomeado estado *default* (ANTUNES, 2018). Este possui as mesmas características do estado inicial de uma máquina de estados. E dentro de um superestado também existe um estado inicial para aquele conjunto interno de estados (HAREL, 1987).

Outro mecanismo presente nas *statecharts* é o conceito de memória de um superestado, no qual o sistema quando retorna para um estado que já esteve antes irá retornar para o estado que estava antes de sair (MIERLO e VANGHELUWE, 2019). A Figura 11 demonstra um exemplo do estado *default* com histórico que é um mecanismo adicional das *statecharts*.

Figura 11 - Estado default de um superestado, ressaltando-se um exemplo com entrada por histórico.

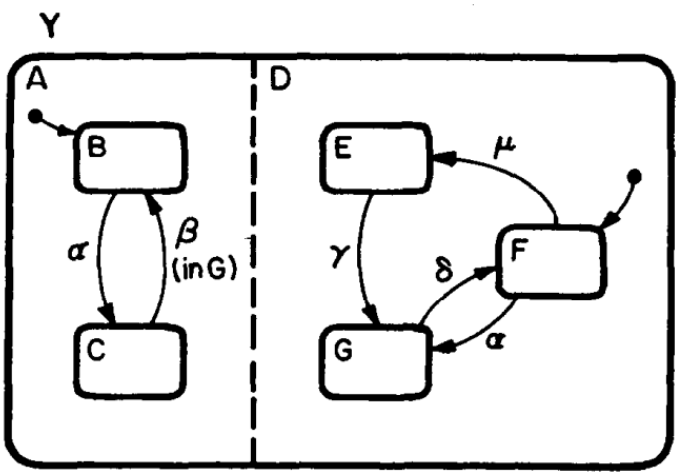


Fonte: (HAREL, 1987).

2.6.4. Ortogonalidade

A ortogonalidade das *statecharts* possibilita representar dentro de um superestado dois ou mais estados concorrentes que operam em paralelo e independentemente (AND exclusivo) (HAREL, 1987). Com este mecanismo é possível representar uma menor quantidade de estados, transições, subestados, e mecanismos de *zoom-out* com transições que ultrapassem as bordas de um estado para o mesmo sistema (ANTUNES, 2018). A Figura 12 representa um sistema genérico Y no qual existem dois subestados (A e D) ortogonais que operam em paralelo (ANTUNES, 2018).

Figura 12 - Representação de um sistema com estados ortogonais.

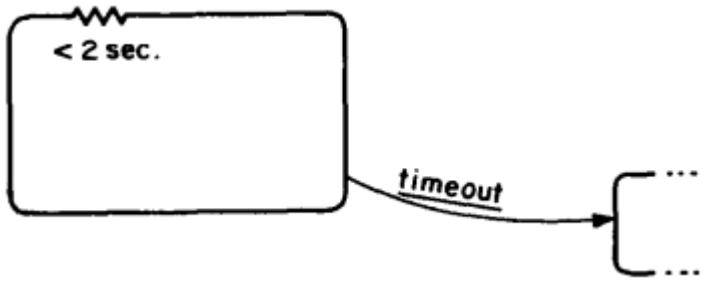


Fonte: (HAREL, 1987).

2.6.5. Temporização

No desenvolvimento de sistema reativos e principalmente quando associados a sistemas embarcados existe o conceito de temporização de eventos. Pelo qual, dentre suas aplicações cita-se situações quando determinado evento chega em uma situação particularmente inconveniente, a máquina está em um estado que não pode lidar com o evento e é necessário que neste caso ocorra uma temporização (HAREL, 1987). Nas *statecharts* é possível modelar tanto *delays* quanto *timeouts* em que adiciona-se este fato na ação do evento ocorrido. A

Figura 13 - Temporização entre ações nas *statecharts*.



Fonte: (HAREL, 1987).

2.6.6. Entradas de seleção

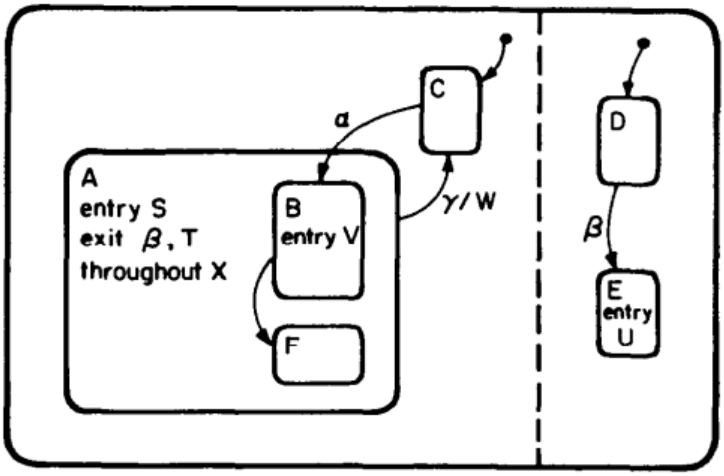
O mecanismo de entradas de seleção das *statecharts* refere-se a uma condição que visa agregar eventos semelhantes (mas, não idênticos) de modo a reduzir a complexidade das transições (HAREL, 1987). Estes são eventos idênticos que podem partir de um mesmo estado por uma transição, e executarem ramificações ou mesmo sistemas condicionais, lógicos ou loops. Dentro das configurações do Stateflow são nomeados como *junctions*.

2.6.7. Ações

As ações são parâmetros que as *statecharts* controlam dentro de um sistema reativo a partir das entradas recebidas (HAREL, 1987). Este mecanismo pode estar dentro dos estados ou mesmo nas transições (analogia as máquinas de Mealy e Moore). Para o primeiro caso é necessário que encaixem em um dos três métodos existentes: *entry* (deve ser executada ao entrar no estado), *exit* (deve ser executada ao sair) e *throughout* (determina as ações a serem executadas enquanto estiver em um superestado) (HAREL, 1987). A Figura 14 demonstra uma

aplicação genérica de uma *statechart* que atua com exemplos de ações dentro de estados, onde é trabalhado os três métodos distintos, e nas transições que são representadas após a condição e separada por uma barra.

Figura 14 - Exemplos de ações dentro de uma *statechart*, observa-se ações dentro dos estados e também nas transições.

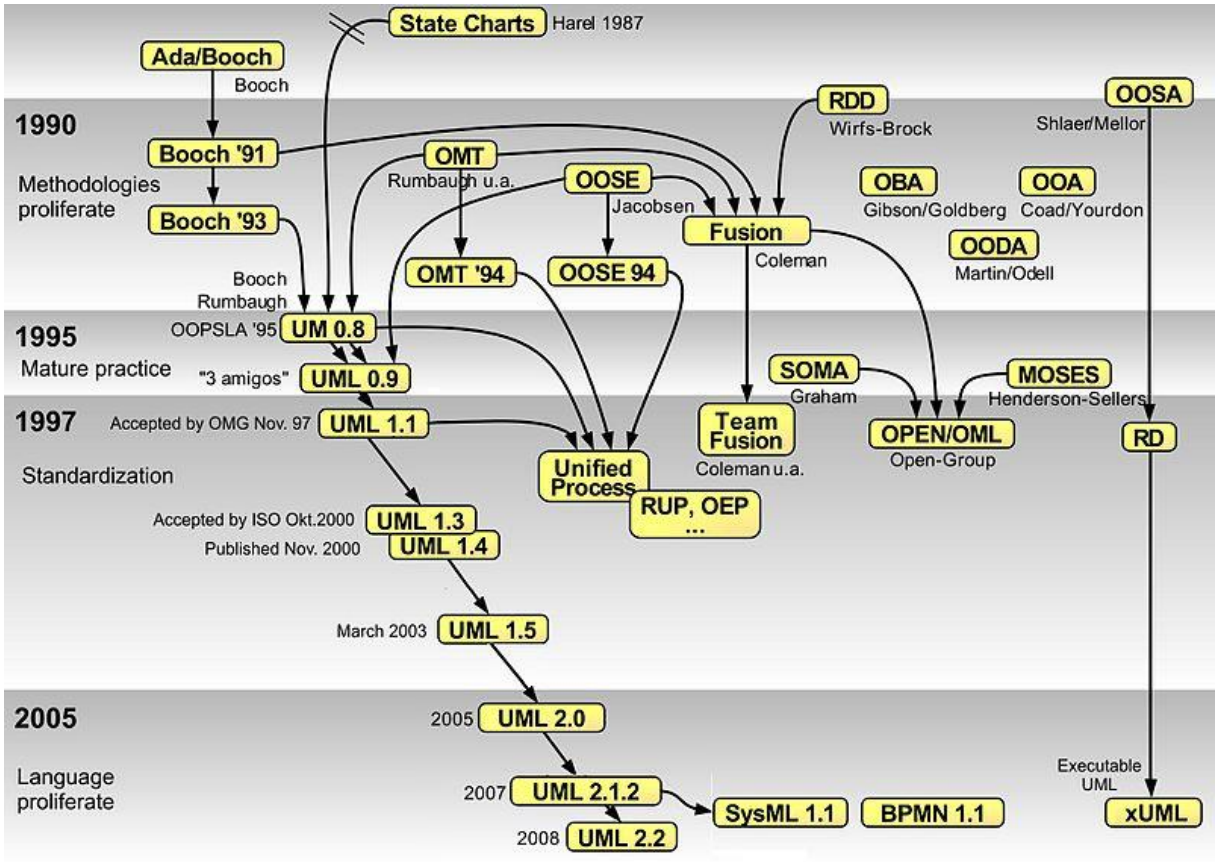


Fonte: (HAREL, 1987).

2.7. Unified Modeling Language (UML)

As *statecharts* de Harel desde a primeira publicação do artigo referente ao assunto ganharam público e apoiadores de forma que seu uso foi generalizado e difundido em diversas plataformas de programação (BOOCH, RUMBAUGH e JACOBSON, 2012). Conforme apresenta a Figura 15 várias linguagens de programação surgem com a base teórica fundamentada nas *statecharts*.

Figura 15 - Desenvolvimento de linguagens de programação a partir das *statecharts*.



Fonte: (ZOCKOLL, SCHEITHAUER e DEKKER, 2012).

Dentre as variantes existentes intensificou-se sua associação com as *Unified Modeling Language* (UML), ou no português Linguagem de Modelagem Unificada (BOOCH, RUMBAUGH e JACOBSON, 2012). São diagramas que permitem a modelagem de superestados, regiões ortogonais e outras características presentes nos artigos de Harel de 1987 (HAREL, 1987). E tem como objetivo utilizar uma linguagem gráfica para visualizar, especificar, construir e documentar informações sobre sistemas complexos e é amplamente utilizada em setores industriais devido sua robustez (BOOCH, RUMBAUGH e JACOBSON, 2012).

2.8. Aplicações do Stateflow

Atualmente o Stateflow é amplamente utilizado em soluções para o desenvolvimento industrial (NEME, 2017). Isto aliado ao fato que possibilita uma modelagem rápida, detalhada e completa para a simulação ou elaboração de um projeto (GRAICS, MOLNÁR, *et al.*, 2020). Fato, que é possível pois as *statecharts* possibilitam ao projetista uma

gama de variadas funções e é ideal para o desenvolvimento de sistemas complexos (HAREL, MARELLY, *et al.*, 2020).

O uso das *statecharts* possibilita ao projetista desde um rápido rascunho do sistema até o maior nível de detalhamento sem utilizar funções complexas para tal fato. E a partir desta informação seu uso se estende a diversos sistemas, como por exemplo desenvolvimento de tecnologias no setor automotivo, aplicações nativas, desenvolvimento baseado em modelos (do inglês: *model based design*) (MATLAB, 2021).

Como aplicação prática cita-se o desenvolvimento de braço robótico (SCARA Robot) pela empresa 3T. Foi necessário o uso do Stateflow para desenvolver os conjuntos de braços robóticos de montagem de conformidade seletiva (SCARA), que são usados na fabricação de semicondutores. E como qualquer maquinário de alta tensão movendo-se em altas velocidades podem causar danos significativos aos seus próprios componentes e às máquinas ao redor foi necessário um desenvolvimento por minucioso em cada estado através das *statecharts* (VAN DER MEER, 2021).

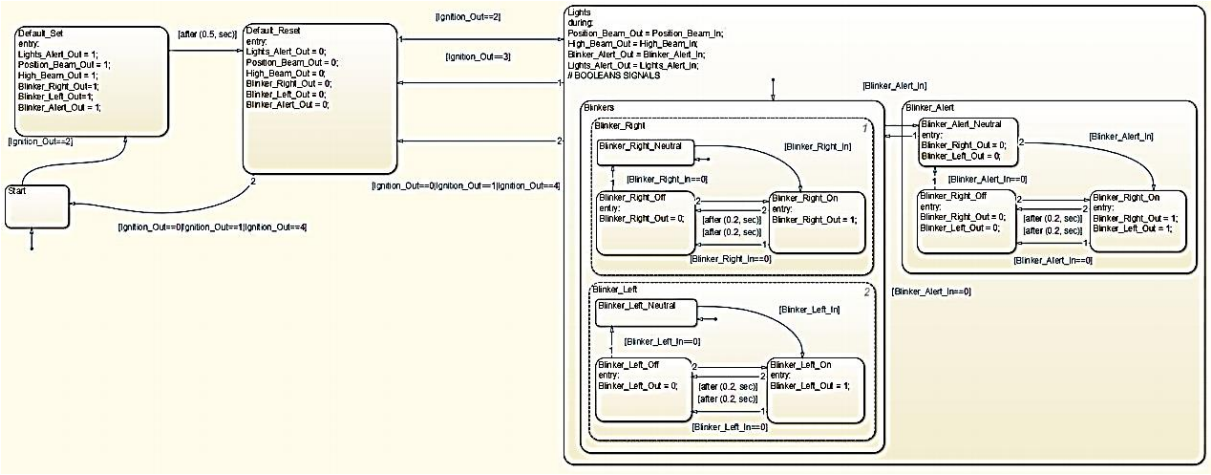
E também cita-se, o desenvolvimento de suspensão a ar controlada eletronicamente para caminhões pesados pela Continental. O uso do Stateflow para esta aplicação tornou-se necessário para implementação rápida do sistema. Visto o alto número de requisitos solicitados pelo cliente, os engenheiros responsáveis pelo projeto necessitaram o desenvolvimento de um modelo de simulação que representasse com exatidão o sistema (EHL, 2019). Com este fato o tempo de verificação foi reduzido em até 50%. Seis meses de esforço de desenvolvimento de *hardware* foram eliminados por problemas identificados já na parte de projeto (EHL, 2019).

Além do desenvolvimento de funções veiculares baseadas a partir do *design* baseado em modelos (MBD) através do qual trabalha-se em desenvolvimento de direções autônomas, central de freios, ECUs, BCMs e também em tecnologias como V2V (do inglês, *vehicle to vehicle*) e V2X (*Vehicle-to-everything*) e sistemas de assistentes inteligentes (SHIMAHARA, 2020). O uso do Stateflow fornece “uma visão mais clara sobre o sistema, visto que estas tecnologias não são mais sistemas de domínio único, e se tornam cada vez mais sistemas de sistemas tecnologias que trabalham em paralelo a outros sistemas veiculares” conforme cita (SHIMAHARA, 2020). Assim, o uso das *statecharts* possibilita maior controle individual de cada sistema além de possibilitar uma visão geral do funcionamento do processo.

Além deste fato, o uso do Stateflow integrado aos componentes da arquitetura de *hardware* e a constante mudança de requisitos possibilita uma alta otimização de código e fácil atribuição de recursos. Fato que permite o trabalho sobre o conceito de “*hardware in the loop”*

(NEME, 2017). A Figura 16 demonstra uma ECU veicular desenvolvida para uso automotivo dentro do ambiente Simulink/Stateflow.

Figura 16 - Diagrama no Stateflow para ECU veicular do grupo luzes.



Fonte: (NEME, 2017).

3. STATEFLOW MENUS E CONFIGURAÇÕES

Este capítulo tem como objetivo introduzir os principais elementos do Stateflow, para dar base ao capítulo educacional. O ambiente de trabalho será exposto juntamente com seus *menus*, funcionalidades e principais comandos. Torna-se necessária a leitura deste capítulo para facilitar a compreensão dos tutoriais desenvolvidos.

Para ter acesso a tela de desenvolvimento do Stateflow é preciso ter o *software* MATLAB da MathWorks com o pacote Simulink Essentials instalado. A versão do utilizada para o desenvolvimento deste TCC foi disponibilizada pelo Polo de Inovação IFMG em parceria ao grupo de pesquisas CNPq GSE.

3.1. O que é o Stateflow

O Stateflow é um ambiente de programação gráfica baseado em *statecharts*, que por sua vez são baseadas em diagramas e máquinas de estados. Ou seja, é uma ferramenta de lógica de controle usada para modelar sistemas reativos[1] por meio de máquinas de estado e fluxogramas[2] dentro de um modelo Simulink (CAMPBELL, 201-). Através do qual possibilita o teste e representações de sistemas dinâmicos[3], em que considera diferentes cenários de simulação e gera o código a partir da representação gráfica (CAMPBELL, 201-). A variante da notação de máquina de estados finitos estabelecida por David Harel (HAREL, 1987) permite a representação de hierarquias, paralelismos e históricos em um gráfico de estados dentro do ambiente de simulação do Stateflow (CAMPBELL, 201-).

Este sistema de simulações do Stateflow serve como um ponto de partida de alto nível[4] para um processo de *design* de *software* associado ao desenvolvimento baseado em modelos MDB (do inglês, *model based design*). No qual, objetiva-se projeto de modelos claros e concisos, onde os diversos elementos de um modelo podem ser desenvolvidos e testados separadamente. O Stateflow é amplamente utilizado quando são necessárias construções de lógica de modo (cada modo discreto detém de um estado), em gerenciamento de falhas e

---

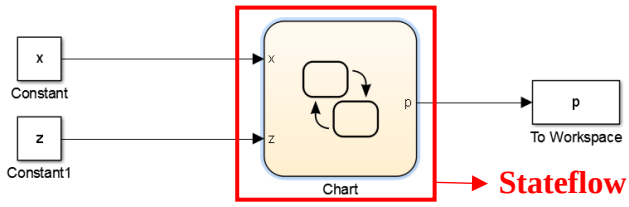
[1] Sistema reativo: a aplicação executada é dependente do ambiente em que se situa. Age a eventos externos. No ambiente de trabalho o Stateflow reage a ambiente que está inserido do Simulink.
[2] Fluxograma: Diagrama que representa um fluxo de trabalho ou processo, sendo a representação diagramática de um algoritmo
[3] Sistema dinâmico: consiste em um conjunto (espaço de estados) e uma lei (transições) que relacionam o estado do sistema no instante n com o seu estado no instante n+1.[3]
[4] Alto nível: em programação é uma linguagem que tem como objetivo ser simples e intuitiva, sendo mais próxima ao humano do que da máquina.

agendamento de tarefas (CAMPBELL, 201-). As aplicações incluem projetos de aeronaves, de automóveis, de sistemas de controle de robótica e em particular neste trabalho sistemas reativos com aplicações em sistemas embarcados.

3.2. Instruções iniciais

No Stateflow a lógica de sistemas é modelada através de *statecharts* conforme é demonstrado pela Figura 17. O ambiente de trabalho do Stateflow é inserido dentro do ambiente do Simulink.

Figura 17 - Ambientes do Simulink e Stateflow relacionando-se para modelagem de um sistema.



Fonte: (Stateflow Onramp, 2020).

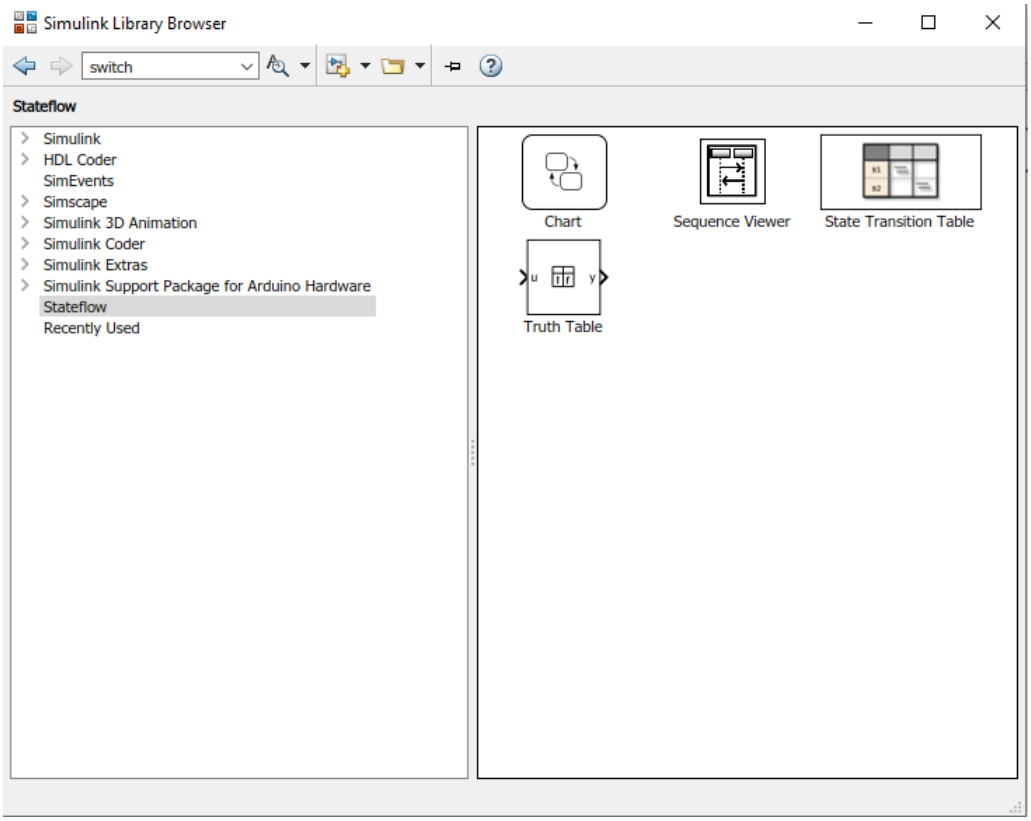
O Simulink permite a modelagem de mudanças continuas em sistemas dinâmicos, como um carro rodando em uma estrada. Já o Stateflow permite a modelagem de eventos instantâneos discretos em sistemas dinâmicos, como o sistema de troca de câmbio em um veículo (CAMPBELL, 201-). Ao combinar as duas interfaces no ambiente de trabalho é possível modelar sistemas completos.

Toda a programação no Simulink é feita em blocos que se comunicam através de setas. E para criar um sistema de simulação do Stateflow basta, dentro da interface Simulink, adicionar o bloco referente ao Stateflow. Este comando pode ser feito ao acessar a opção *Library Browser*, identificar o bloco objetivado, arrastar e soltar o bloco para tela de edição. Com um duplo clique é então aberto uma nova tela, em tom rosa claro, referente ao Stateflow.

3.3. Áreas de trabalho do Stateflow

O ambiente dentro do Simulink é composto por diversas áreas de trabalho com suas respectivas funções. Nesta é possível, pela *Library Browser*, escolher modelos de entrada, saída, funções matemáticas, *dashboards* e também os elementos de edição do Stateflow conforme expõe a Figura 18.

Figura 18 - Library Browser: biblioteca de comandos do Simulink.

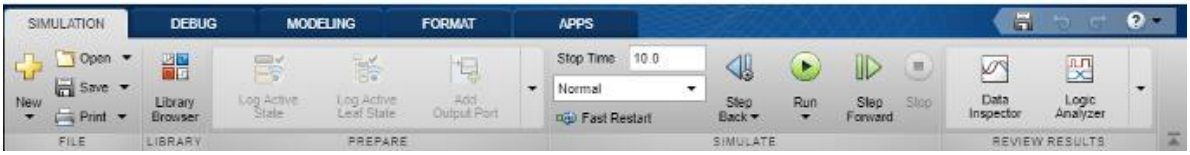


Fonte: Elaborado pelo autor, 2021.

3.3.1. Toolstrip

Conforme mostra a Figura 19 a maioria dos comandos do editor está disponível na barra de ferramentas nomeada *Toolstrip*. Caso o ponteiro fique sobre a funcionalidade por um tempo é possível ver uma dica de utilização.

Figura 19 - Toolstrip bar do Stateflow.

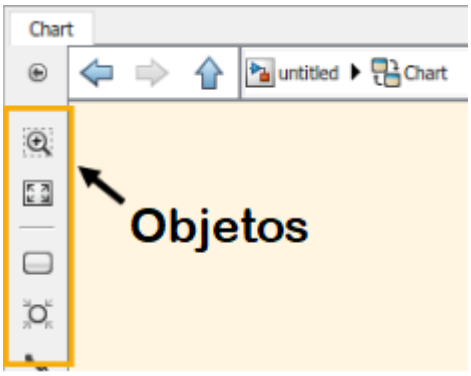


Fonte: Elaborado pelo autor, 2021.

3.3.2. Object pallet

A *Object pallet* (paleta de objetos) exibe um conjunto de ferramentas para desenhar estados, transições e outros objetos de gráfico, conforme representado na Figura 20. Para adicionar um objeto basta arrastar e soltar para a área de edição.

Figura 20 - Object Pallet do Stateflow.



Fonte: Elaborado pelo autor, 2021.

3.3.3. Explore bar

A Explore bar (barra de exploração) mostra os sistemas que você abriu no editor, conforme demonstra a Figura 21. Além disso, define a hierarquia entre as páginas de trabalho desde ambientes Simulink e Stateflow a até para superestados, estados e subestados.

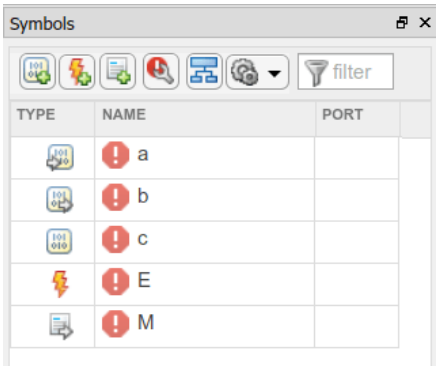
Figura 21 - Explore Bar do Stateflow.

Fonte: Elaborado pelo autor, 2021.

3.3.4. Symbols

O menu Symbols tem funcionalidades como inserção de variáveis, definição das variáveis externas e internas, o tipo, valores e outras funções. Na Figura 22 é possível visualizar estas configurações. É importante sempre checar esta janela antes de executar o programa, visto que irá relacionar as variáveis do Stateflow com o Simulink.

Figura 22 - Menu *Symbols* do Stateflow.



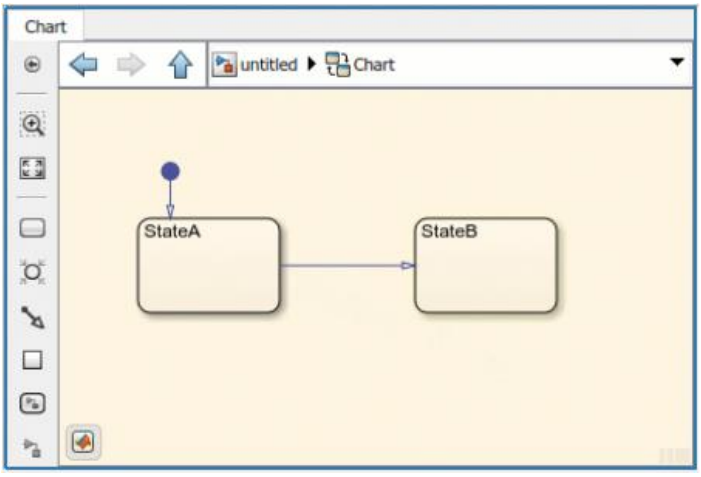
Fonte: Elaborado pelo autor, 2021.

3.4. Elementos de um *statechart*

Para criar um estado é necessário arrastar o ícone retângulo na barra de objetos para a tela de edição do Stateflow, e em seguida basta nomeá-lo. Atenta-se que o primeiro estado adicionado terá uma seta indicativa apontado para ele, isto significa que este é um estado *default*.

Para adicionar transições é necessário aproximar o cursor em uma extremidade dos estados adicionados e então arrastar a seta que irá aparecer até o próximo estado que a transição estará relacionada. A Figura 23 mostra o resultado desde desenvolvimento.

Figura 23 - Dois estados conectados por uma transição.



Fonte: Elaborado pelo autor, 2021.

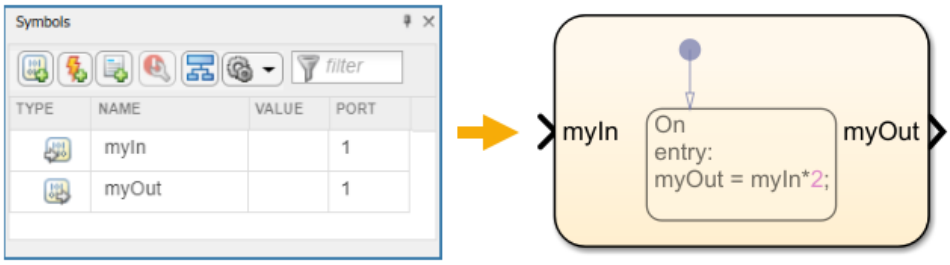
Com a transição elaborada basta sobre ela adicionar uma condição, isto clicando duas vezes e adicionando o texto entre colchetes. O MATLAB reconhece várias condições juntas, basta adicionar operadores como conjunção (*and* representado por &&) ou o operador

disjunção (*or* representado por | |), por exemplo. A sintaxe, além de estar dentro colchetes, deve obedecer às regras de operadores lógicos do MATLAB. Para que uma transição ocorra é necessária que a condição seja verdadeira.

3.5. Variáveis

Dentro do ambiente de trabalho do Stateflow é necessário configurar as variáveis dentro do Menu Symbols, seja para defini-las como: entrada, saída, variável local ou como parâmetro. A Figura 24 (a) mostra um exemplo em que duas variáveis foram adicionadas ao Stateflow, uma como entrada e outra como saída, e na Figura 24 (b) como o sistema interpreta este comando pela tela de edição do Simulink.

Figura 24 - (a) Variáveis no menu *Symbols* no Stateflow e (b) Ambiente *Simulink* e a representação das variáveis.



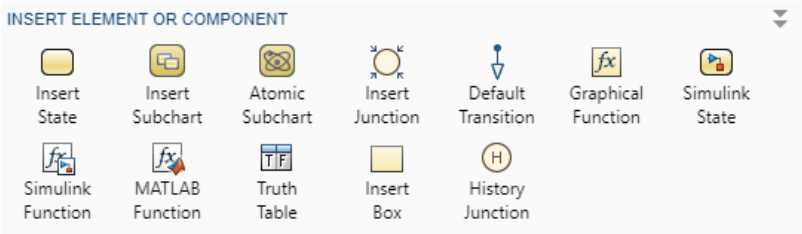
Fonte: Elaborado pelo autor, 2021.

Caso esta tela de trabalho não apareça como visível, basta ir na *Toolstrip* bar na aba *Modeling* e selecionar com o mouse o botão *Symbols Pane*.

3.6. Componentes

Dentro do ambiente Stateflow existe uma série de componentes e elementos para auxiliar na simulação ou na elaboração de sistemas. A Figura 25 demonstra os principais itens pertencentes ao Stateflow. Vale ressaltar que existem funcionalidades adicionais de acordo com a versão do *software* utilizado. Os elementos essenciais para aprendizagem inicial do Stateflow são apresentados nos tutoriais do próximo capítulo.

Figura 25 - Componentes e elementos de simulação presentes no Stateflow.

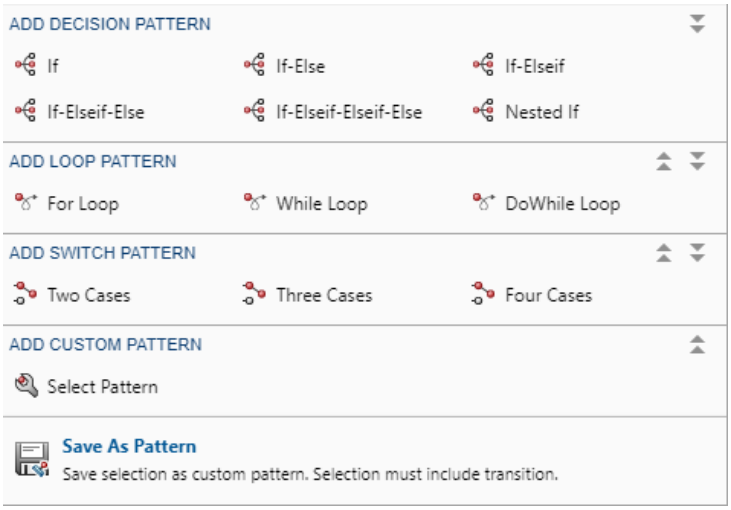


Fonte: Elaborado pelo autor, 2021.

3.7. Lógicas Padrões

Dentro do ambiente do Stateflow existem lógicas com transições pré-modeladas para auxiliar o usuário no desenvolvimento dos seus programas ou simulações. Conforme a Figura 26 apresenta as transições com estruturas de decisão, estrutura de repetição e padrões de chaveamento.

Figura 26 - Sistemas de transições pré-modeladas do Stateflow.



Fonte: Elaborado pelo autor, 2021.

4. METODOLOGIA

Para a aprendizagem do Stateflow, é exposto ao leitor em formato de tutoriais exemplos práticos de desenvolvimento de sistemas reativos que se associam-se com sistemas embarcados reais. Tal uso é motivado para que o usuário seja encorajado a buscar melhorias e especificações detalhadas de cada sistema desenvolvido. Neste capitulo são apresentadas as principais ideias e conceitos por trás de cada um dos sete tutoriais que se encontram em ANEXO (conforme sumário, a partir da página 67 deste documento).

Os exemplos desenvolvidos visam apresentar sistemas reativos baseados em sistemas embarcados reais, porém em sua forma minimalista, visto que o intuito é o aprendizado da plataforma Stateflow em que se busca aplicar os principais conceitos das *statecharts*. Conforme (SIPSER, 1996) cita “Seja de uma máquina de estados ou de um trabalho artístico, projetar é um processo criativo. Como tal ele não pode ser reduzido a uma simples receita ou fórmula”. Logo, os itens subsequentes deste capítulo foram escritos de forma a serem didáticos, mas, não deve-se limitar ao único fluxo criativo descrito nos tutoriais, pois existem diversas formas para o desenvolvimento do mesmo sistema. A partir dos conceitos e das ferramentas é possível empregar os diversos tipos de sistemas e de complexidades distintas pois os conceitos aplicados são base para todo o desenvolvimento de situações deste tipo.

Dentro do *software* Simulink e ambiente Stateflow são necessários alguns passos e configurações para simulação adequada do problema. Inicialmente é necessário realizar a descrição do problema, imaginar quais serão as saídas, as entradas e como o problema irá interagir com a fita, para assim poder descrever o diagrama das *statecharts*. Com isso define-se o modelo matemático do problema, ou mesmo sua descrição (especificação) detalhada textualmente e a partir destes é desenvolvido o sistema no programa. Para teste do sistema será elaborado um modelo em *dashboards* no Simulink para controle do sistema reativo, no qual as interações do usuário resultam nas transições do sistema.

Observa-se que os roteiros são desenvolvidos de maneira que o problema seja analisado de uma visualização macro para micro. Justifica-se, pois, que é fundamental entender o que será desenvolvido e depois para praticar os estados e as transições e elaborar a *statechart*. Sugere-se ao leitor também, que primeiro o leia todo a fim de conhece-lo passo a passo. Consequentemente poderá ter uma compreensão total sobre o tema e assim ir desenvolvendo os passos já com as expectativas dos resultados esperados.

4.1. Roteiro de aprendizagem: Alarme

Este roteiro tem como objetivo simular computacionalmente uma *statechart* com o propósito de entender os conceitos básicos de estados e transições, aliado ao aprendizado no *software* Stateflow.

O roteiro é composto pela elaboração de um sistema que contém uma representação sobre o funcionamento de um alarme simples. O sistema reativo é controlado por um sensor, que tem como entrada e saída valores booleanos. Quando o sensor tem como saída alta indica que o alarme deve ser ativado e caso contrário o alarme deve ser desativado.

Para isso deve-se desenvolver uma *statechart* em que aplica-se os "conceitos" de uma máquina de Moore para um alarme controlado por sensor/chave. Esta é descrita matematicamente por:

a) $Q = \{\text{Desligado}, \text{Ligado}\}$;

b) $\sum = \{0,1\}$;

c) $\delta: Q \times \Sigma$

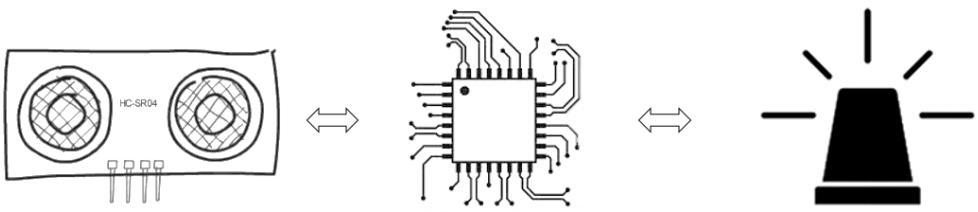
$\delta: Q \times \Sigma$	0	1
Desligado	Desligado	Ligado
Ligado	Desligado	Ligado

d) $q_0 = \text{Desligado}$;

e) $F = Q$.

Com base na descrição textual e matemática é possível desenvolver um sistema em Stateflow que reproduza o problema em questão. A Figura 27 representa ilustradamente o sistema proposto, sendo que na esquerda existe o sensor do sistema, ao centro o controlador (que é o foco deste trabalho) e na direita o alarme. Entre os componentes existe uma seta indicando sua comunicação com o controlador, que para este caso é o sistema a ser desenvolvido no Stateflow.

Figura 27 - Ilustração do sistema embarcado "Alarme" a ser desenvolvido com Stateflow.



Fonte: Elaborado pelo autor, 2021.

O roteiro de desenvolvimento referente a este sistema reativo encontra-se em Anexo, no item Procedimento experimental: Alarme, a partir da página 67.

4.2. Roteiro de aprendizagem: Elevador

Este roteiro tem como objetivo simular computacionalmente uma *statechart* a fim de compreender ferramentas de transição, que auxiliam na formulação de estruturas de decisão ou de repetição. Será trabalhado neste exemplo o uso da ferramenta *junction* visto que ela é a base de toda lógica de transição mais complexa. Isto aliado com o desenvolvimento de um sistema de controle de andar de elevador.

O sistema reativo faz o controle de um elevador simples. Para isto é necessário realizar o procedimento anterior de desenvolvimento do alarme para familiarizar com conceitos básicos referentes ao *software*, principalmente no que se refere a estados e transições.

O sistema se descreve por: um sistema embarcado para controle de elevador feito por *statecharts*. O elevador é controlado por um sistema de botões que define qual andar ele irá se locomover, de modo que ao pressionar, por exemplo, o botão do primeiro andar o elevador se direciona ao primeiro andar e para outros andares sob o mesmo processo. Como saída o sistema indica o andar que o elevador está presente.

A descrição matemática desta *statechart* é dada por:

a) $Q = \{Primeiro, Segundo, Terceiro\}$;

b) $\sum = \{1,2,3\}$;

c) $\delta: Q \times \Sigma$

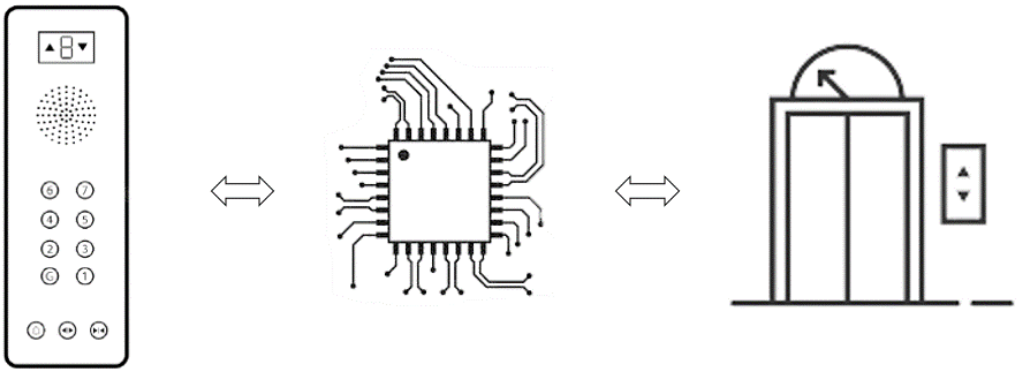
$\delta: Q \times \Sigma$	1	2	3
Primeiro	Primeiro	Segundo	Terceiro
Segundo	Primeiro	Segundo	Terceiro
Terceiro	Primeiro	Segundo	Terceiro

d) $q_0 = Primeiro$;

e) $F = Q$.

Para efeito de ilustração a Figura 28 demonstra o esquema entre controlador, entradas e saídas do roteiro proposto. Indica-se a esquerda um painel de controle dos botões do elevador, ao centro o controlador e na direita a representação de um elevador com indicação analógica do andar.

Figura 28 - Ilustração do sistema embarcado "Elevador" a ser desenvolvido com Stateflow.



Fonte: Elaborado pelo autor, 2021.

O roteiro de desenvolvimento referente a este sistema reativo encontra-se em Anexo, no item Procedimento experimental: Elevador, a partir da página 81.

4.3. Roteiro de aprendizagem: Porta do Elevador

Este roteiro tem como objetivo simular computacionalmente uma *statechart* afim de compreender ferramentas de hierarquias de estados (superestados e subestados). E consequentemente apresentar variáveis internas no *Stateflow* e ferramentas dentro dos estados para ações de entrada, saída e durante a execução, além das configurações de hierarquias entre os estados.

A *statechart* a ser projetada tem como objetivo representar um sistema reativo que por meio de uma máquina de estados faz o controle da porta de um elevador. Para isto é necessário realizar o procedimento anterior de desenvolvimento do elevador para familiarizar com conceitos básicos referentes ao *software*. Ressalta-se que este exemplo é continuidade do item de desenvolvimento do “Elevador”.

O problema se descreve informalmente por: um sistema reativo para controle da porta de um elevador a partir de *statecharts*. No qual, possui o controle também sobre a porta do elevador (além das funções de troca de andares). As entradas são feitas por um sistema de botões (para abrir e fechar) e além de possibilitar o elevador deslocar entre os andares. Como saída o sistema indica qual é a situação da porta.

A descrição desta *statechart* é dada por:

a) $Q = \{\text{Aberto}, \text{Abrindo}, \text{Fechado}, \text{Fechando}\}$;

b) $\sum = \{\text{Abrir}, \text{Fechar}\}$;

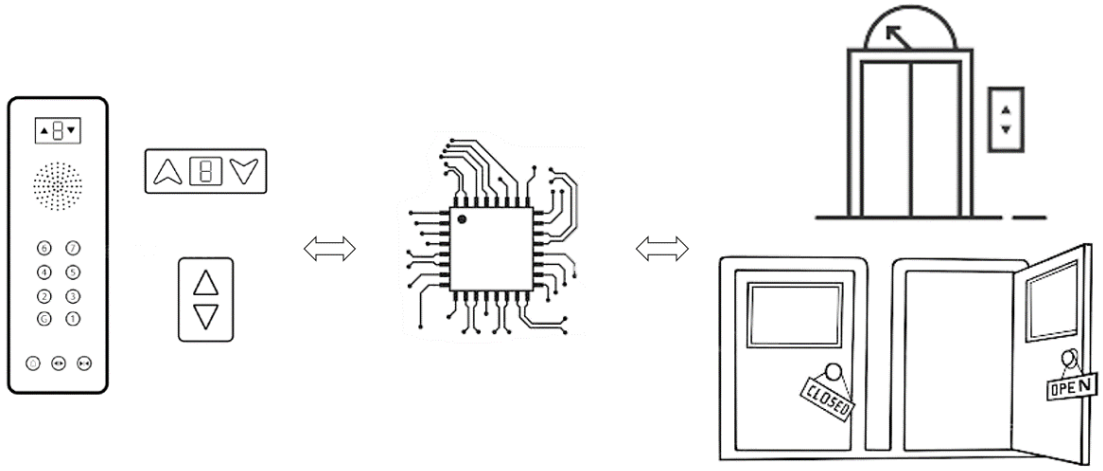
c) $\delta: Q \times \Sigma$

d) $q_0 = \text{Fechado};$

e) $F = \{\text{Aberto}, \text{Fechado}\}.$

Conforme descrito matematicamente e textualmente é possível representar de forma ilustrativa o sistema a ser desenvolvido. A Figura 29 demonstra as entradas e saídas do sistema na esquerda, com os andares do elevador e botões para a posição da porta, ao centro é destacado o controlador e na direita a posição do estado do elevador, e também, uma representação sobre o *status* da porta.

Figura 29 - Ilustração do sistema embarcado "Porta Elevador" a ser desenvolvido com Stateflow.



Fonte: Elaborado pelo autor, 2021.

O roteiro de desenvolvimento referente a este sistema reativo encontra-se em Anexo, no item Procedimento experimental: Porta do elevador, a partir da página 89.

4.4. Roteiro de aprendizagem: Drone

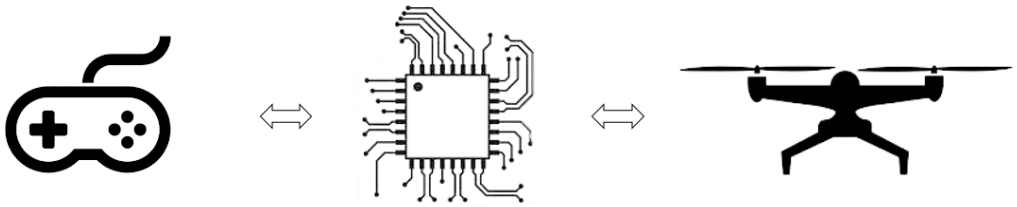
Este roteiro tem como objetivo simular computacionalmente uma *statechart* afim de entender os conceitos básicos sobre entradas e saídas não booleanas. Com um exemplo interativo no qual são encontrados conceitos apresentados nos tutoriais precedentes utiliza-se um exemplo de controle de um Drone.

Para tal é proposto ao leitor desenvolver um sistema reativo para controle de um drone com funções básicas de funcionamento. O sistema é composto por quatro variáveis de entrada, sendo: “liga” para ativar o controle das funções do drone; “voar” para habilitar a

operação de voo do drone, “distância” que se refere ao ponto de distância do ponto de partida e “bateria” referente a tensão na bateria do drone.

Como a complexidade do sistema é maior do que as anteriores, a visualização matemática é comprometida (uma vez que é extensa e não agrega valor quanto outro método de especificação), logo para este exemplo será descrito o funcionamento de maneira textual. A Figura 30 representa, de modo geral, as entradas e saídas deste sistema reativo, em que além das funções do controle, também existem sensores como *status* da bateria e *status* da distância do ponto de origem.

Figura 30 - Ilustração do sistema embarcado "Drone" a ser desenvolvido com Stateflow.



Fonte: Elaborado pelo autor, 2021.

O modo de operação é baseado em quatro estados distintos. Como estado *default* o sistema se encontra em “desligado” e, a partir deste estado são realizadas transições para outros estados se o botão de liga for acionado. O estado “pousado” refere-se ao momento em que o drone não fez sua decolagem ou retornou para o ponto inicial. Sabe-se que um drone tem um limite de distância que pode ser controlado e também há um limite de bateria para fazer um pouso seguro. Logo, existem outros dois estados o “controlado” através do qual o operador faz suas manobras e o “autônomo” em que o sistema retorna para seu ponto inicial porque foi perdido o contato por distância ou a bateria atingiu um nível crítico.

O roteiro de desenvolvimento referente a este sistema reativo encontra-se em Anexo, no item Procedimento experimental: Drone, a partir da página 101.

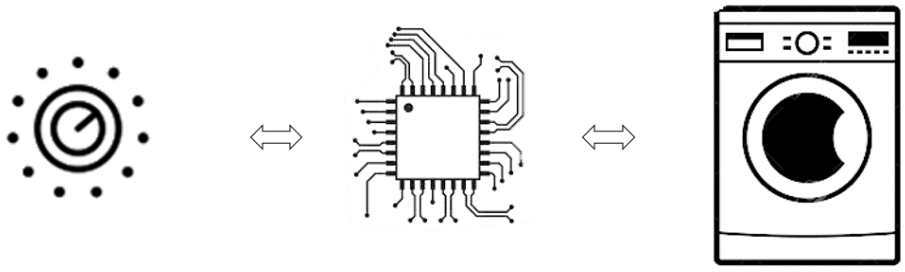
4.5. Roteiro de aprendizagem: Máquina de lavar

Este roteiro tem como objetivo simular computacionalmente uma máquina de estados finita afim de assimilar os conceitos básicos da máquina de Mealy e também princípios de padrões de estruturas condicionais presentes no Stateflow. Para isto, apresenta-se um exemplo de desenvolvimento de uma máquina de lavar simples.

O roteiro tem como intuito desenvolver um sistema embarcado para controle de uma máquina de lavar a partir de uma *statechart*. O sistema é composto por uma entrada que se refere a alimentação a energia elétrica do sistema (liga/desliga), e uma entrada não booleana referente ao modo de trabalho da máquina de lavar, seja para roupas mais leves ou pesadas (cada item irá representar uma entrada de um valor inteiro diferente). A máquina possui quatro modos de funcionamento (funções/ações) após ligada, que são: enxague, molho, agitação e centrifugar que são indicadas como saídas de 1 a 4, e na simulação serão representadas por uma *dashboard* com diferentes cores de indicação. É válido ressaltar que este modelo de máquina de lavar é hipotético e visa apresentar conceitos das *statecharts*.

O funcionamento é baseado em dois estados distintos. Para iniciar a fita, o estado *default* é “inicia” e a partir deste estado é realizada transição caso o sistema for ligado. O estado “funcionamento” refere-se ao momento em que a máquina encontra-se em operação. Para cada entrada distinta, o sistema irá realizar uma diferente transição e dentro da transição é alocada uma ação, o que caracteriza o sistema a ser uma máquina de Mealy. A Figura 31 descreve a ilustração do sistema, com uma entrada com *knob* rotativo para seleção de velocidade no qual comunica-se com o controlador do sistema (*statechart*) e tem como saída a velocidade da máquina.

Figura 31 - Ilustração do sistema embarcado "Máquina de lavar" a ser desenvolvido com Stateflow.



Fonte: Elaborado pelo autor, 2021.

O roteiro de desenvolvimento referente a este sistema reativo encontra-se em Anexo, no item Procedimento experimental: Máquina de lavar, a partir da página 107.

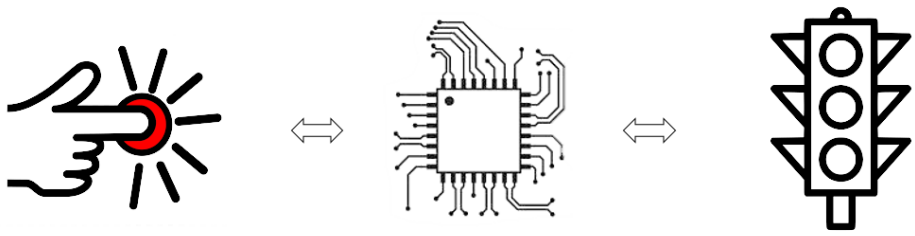
4.6. Roteiro de aprendizagem: Semáforo

Este roteiro tem como objetivo simular computacionalmente uma *statechart* afim de entender os conceitos básicos sobre interrupções de estados, comuns no desenvolvimento de microcontroladores. Com um exemplo interativo no qual são encontrados conceitos

apresentados nos tutoriais precedentes em que utiliza-se um modelo de um semáforo com botão prioritário para pedestre e transições entre estado principal e paralelo.

O roteiro desenvolve um sistema reativo para controle de um semáforo de uma via a partir de uma *statechart*. O sistema é composto por dois semáforos, um para veículos e outro para pedestre. O funcionamento é baseado na lógica convencional de um semáforo e quando ocorre uma interrupção no sistema é realizada uma transição para um estado paralelo, que neste caso é o botão para dar prioridade ao pedestre. Após o tempo definido para o pedestre realizar sua passagem, o funcionamento volta para o modo convencional. A Figura 32 demonstra o funcionamento do sistema, sendo que o sinal de interrupção faz interface com o controlador e este envia sinais para o desempenho do semáforo.

Figura 32 - Ilustração do sistema embarcado "Semáforo" a ser desenvolvido com Stateflow.



Fonte: Elaborado pelo autor, 2021.

O roteiro de desenvolvimento referente a este sistema reativo encontra-se em Anexo, no item Procedimento experimental: Semáforo, a partir da página 115.

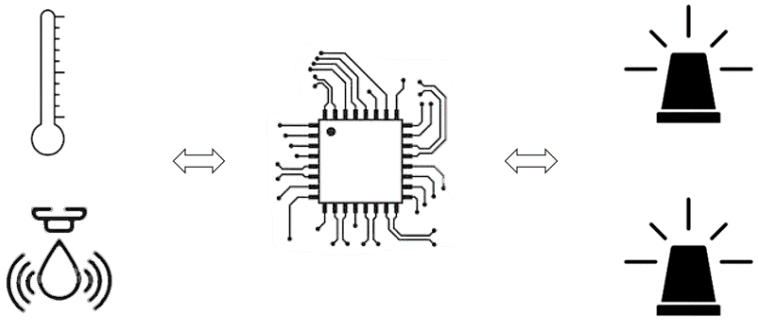
4.7. Roteiro de aprendizagem: Controle de umidade e temperatura

Este roteiro tem como objetivo simular computacionalmente uma *statechart* afim de entender os conceitos de ortogonalidade (AND-exclusivo). Para isto, apresenta-se um exemplo de desenvolvimento de um sistema reativo com simulação de controle de temperatura e umidade, sistema embarcado que é frequentemente visto em granjas.

O roteiro visa desenvolver um sistema reativo para controle da umidade e da temperatura de um sistema ao mesmo tempo. O sistema é composto por uma entrada que se refere a alimentação a energia elétrica do sistema (liga/desliga). Uma entrada booleana para indicar temperatura acima do nível estabelecido e uma outra entrada booleana para umidade com mesma função. Caso sejam recebidas entradas com essas condições em nível lógico alto é acionado um alarme para a respectiva variável mensurada.

A Figura 33 ilustra o sistema proposto, no qual existem duas tarefas ortogonais entre si e o controlador recebe e envia as informações de ambos os sistemas ao mesmo tempo. A esquerda está exposto o controlador de temperatura e umidade respectivamente, ao centro o controlador e a direita os alarmes de sinalização.

Figura 33 - Ilustração do sistema embarcado "Sensor de temperatura e umidade" a ser desenvolvido com Stateflow.



Fonte: Elaborado pelo autor, 2021.

O roteiro de desenvolvimento referente a este sistema reativo encontra-se em Anexo, no item Procedimento experimental: Controle de umidade e temperatura, a partir da página 120.

5. RESULTADOS

Para visualizar a aplicação prática dos roteiros de aprendizagem do Stateflow desenvolvidos neste trabalho o autor em conjunto ao grupo de pesquisas GSE – Grupo de Soluções em Engenheira aplicou a diferentes pesquisadores os tutoriais. Este para alunos de diferentes níveis de graduação e diferentes cursos e também para professores, da engenharia elétrica e da ciência da computação, os tutoriais presentes em Anexo. Conforme apresenta a Tabela 1 são detalhados o corpo amostral desta pesquisa.

Tabela 1 - Pesquisadores expostos aos tutoriais.

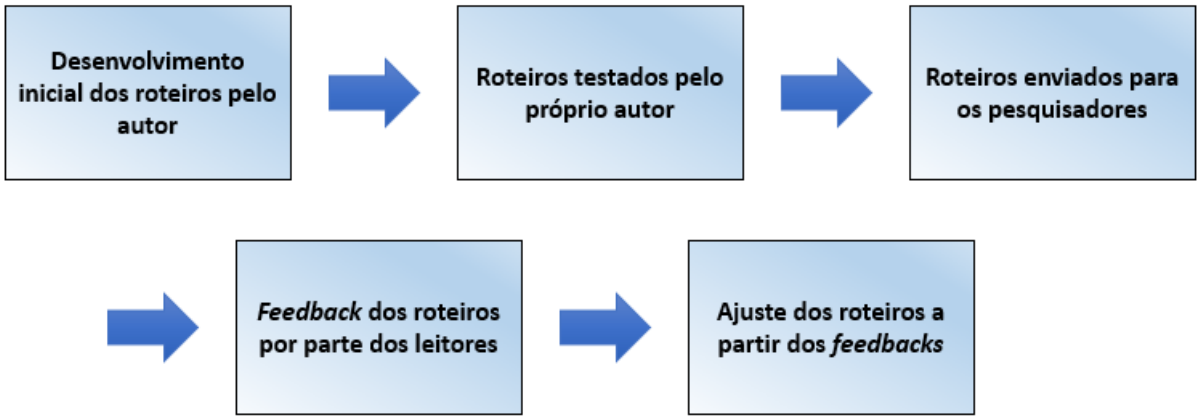
Pesquisador	Titulação
Pesquisador 1	Estudante Ciência da Computação 4º período
Pesquisador 2	Estudante Ciência da Computação 4º período
Pesquisador 3	Estudante Engenharia Elétrica 10º período
Pesquisador 4	Professor Engenharia Elétrica
Pesquisador 5	Professor Ciência da Computação

Fonte: Elaborado pelo autor, 2021.

Os roteiros servem para estes pesquisadores como base para desenvolverem seus próprios sistemas reativos. Isso, no desenvolvimento da Iniciação científica sobre métodos de verificação automática (*Model Checking*) que utiliza a modelagem em sistemas no Stateflow como principal ponto para verificação dos sistemas. Aonde tem como objetivo, por exemplo, o desenvolvimento de sistemas como controle de cruzeiro em um sistema automotivo.

Alicerçado a técnica de Aprendizagem Ativa, os alunos receberam os tutoriais e reproduziam os exemplos descritos, além de implementarem suas próprias funcionalidades e melhorias. O fluxo das atividades é representado na Figura 34, aonde os pesquisadores recebiam os roteiros sem explicações prévias sobre o tema de forma a simular um leitor autodidata que venha a desenvolver os tutoriais contidos neste trabalho.

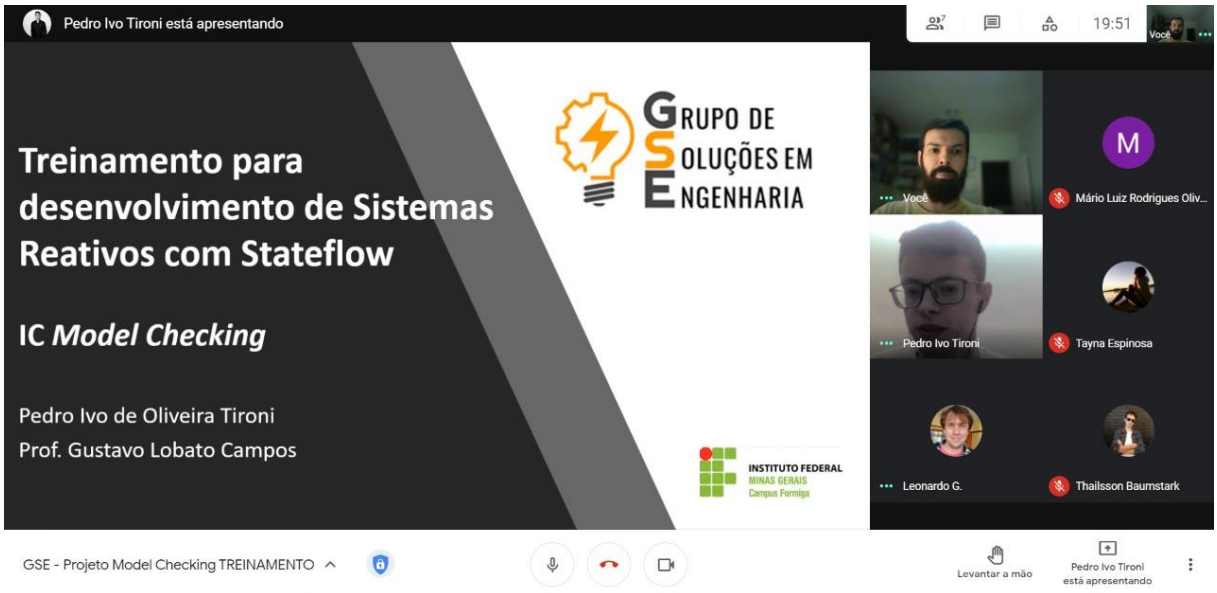
Figura 34 - Fluxo de aplicação dos roteiros aos pesquisadores



Fonte: Elaborado pelo autor, 2021.

Com reuniões semanais foi realizado acompanhamento acerca dos tutoriais à medida que foram disponibilizados aos alunos. A metodologia possibilitou o refinamento dos roteiros, além de melhorias em explicações e em certos pontos simplificação de partes do texto para desenvolvimento mais ágil dos tutoriais desenvolvidos e apresentados neste trabalho. A Figura 35 demonstra um encontro dos membros do grupo de pesquisa para *feedback* de avaliação, acompanhamento e melhorias do tutoriais apresentados.

Figura 35 - Reunião com alunos do grupo de pesquisa para feedback dos roteiros.



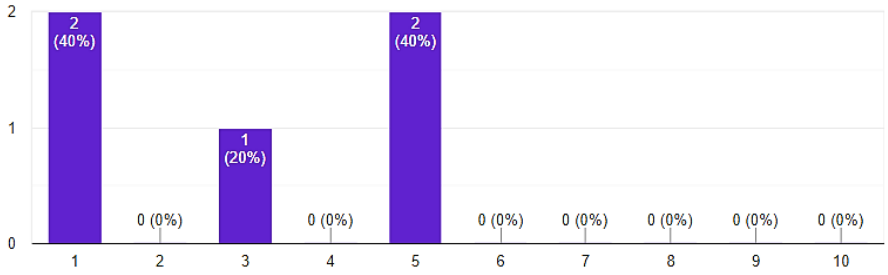
Fonte: Elaborado pelo autor, 2021.

Após todos os roteiros serem desenvolvidos pelo grupo foi realizada uma pesquisa com intuito de avaliar grau de aprendizado e de complexidade. Além de, levantar comentários

sobre os roteiros desenvolvidos com o intuito de analisar se os alunos expostos aos tutoriais pelo método de Aprendizagem Ativa tornam-se aptos a desenvolver seus próprios sistemas para suas aplicações.

Considerando o objeto do treinamento, o *software* MATLAB Simulink, especificamente a Toolbox Stateflow em uma escala de um a dez o nível de conhecimento médio dos alunos expostos aos tutorias era de 30%. Sendo que neste grupo, existiam alunos com domínio médio sobre o *software* e outros com nenhum domínio. A Figura 36 demonstra as respostas dos alunos expostos sobre seus conhecimentos prévios.

Figura 36 - Conhecimento prévio sobre as ferramentas do MATLAB Stateflow dos integrantes do treinamento.

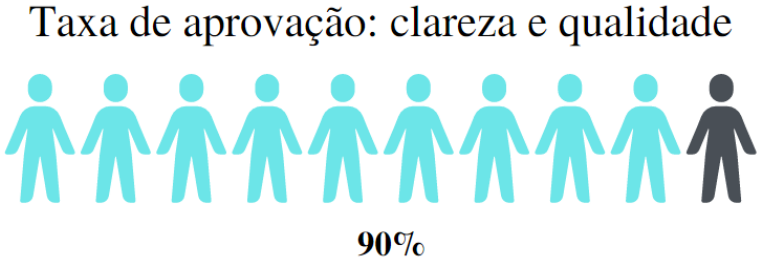


Fonte: Elaborado pelo autor, 2021.

No que se refere a forma do treinamento realizado, em que se diz respeito a roteiros voltados para a Aprendizagem Ativa, o item teve 90% de aceitação dos alunos quando perguntados para avaliarem de um a dez sobre o método de ensino. Além de 100% dos alunos pontuarem que a apresentação dos tutoriais foi produtivo para desenvolverem suas próximas atividades no Stateflow.

Especificamente sobre os roteiros apresentados em uma escala de um a dez, teve-se uma taxa de aprovação de 90% sobre a qualidade e clareza dos mesmos, conforme apresenta a Figura 37. E 100% confirmaram que aprenderam os conceitos necessários para a elaboração de diagramas para simulação das *statecharts*.

Figura 37 - Média da taxa de aprovação conforme avaliado pelos alunos nos itens clareza e qualidade.



Fonte: Elaborado pelo autor, 2021.

Quando indagados sobre a utilidade deste treinamento para desenvolvimentos de projetos futuros com o uso da plataforma Stateflow, até mesmo com complexidade maiores do que as apresentadas, todos os alunos aprovam a leitura e desenvolvimento dos roteiros para capacitação e aprendizado.

Logo, todos os alunos expostos aos tutoriais consideram que aprenderam os conceitos base para simulação de sistemas no Stateflow. Fato que apresentou 90% de aprovação quando perguntados se pessoas com conhecimentos prévios (básicos) sobre programação estavam aptas para desenvolver os tutoriais.

6. CONCLUSÕES

Este trabalho teve como objetivo apresentar os conceitos básicos do desenvolvimento de sistemas reativos com *statecharts* pelo *software* Stateflow, em forma de tutoriais para Aprendizagem Ativa. De maneira que o leitor, com o desenvolvimento dos exemplos presentes no texto, quebre a barreira inicial que dificulta o aprendizado sobre o tema. Os roteiros desenvolvidos passaram por testes práticos de validação, no qual alunos desenvolveram, aplicaram e validaram os passos presentes em cada tutorial apresentado. Fato que possibilita o desenvolvimento de sistemas com complexidade elevada a partir do conhecimento que adquiriram pelos roteiros desenvolvidos.

De modo geral, o texto se destaca pelo fato de tutoriais sobre estas ferramentas (Stateflow e *statecharts*) serem apresentadas em forma de roteiros e na língua portuguesa (visto que para este tema os trabalhos são escassos na literatura nacional). De forma, a especificar detalhadamente cada item necessário para construção de um sistema reativo em ambiente Simulink/Stateflow.

Importante destacar que parte deste trabalho, foi apresentado no XLVIII Congresso Brasileiro de Educação em Engenharia (COBENGE) e III Simpósio Internacional de Educação em Engenharia da ABENGE o trabalho intitulado “Aprendizagem Ativa Para Sistemas Embarcados: Uma Abordagem com SIMULINK/STATEFLOW” (TIRONI, CAMPOS e OLIVEIRA, 2020). Através do qual, além do desenvolvimento de sistemas reativos é apresentado um roteiro prático para embarcar código gerado pelo ambiente de simulação (Simulink) em um microcontrolador, que possibilita o leitor desenvolver seus próprios sistemas embarcados com uso nas *statecharts*. Trata-se inclusive de recomendação de leitura para aqueles que desenvolveram os roteiros presentes neste trabalho e tem como objetivo embarcar os roteiros desenvolvidos (TIRONI, CAMPOS e OLIVEIRA, 2020).

Para trabalhos futuros o autor cita o desenvolvimento de trabalhos que associem também ao ambiente Simulink/Stateflow outras *toolbox* amplamente utilizadas na indústria como Simscape, AUTOSAR Blockset, Simulink Coder e Embedded Coder. Além da integração dos sistemas reativos desenvolvidos no ambiente Simulink com as ferramentas *Statistics and Machine Learning* e *Deep Learning* presentes dentro do *software* MATLAB, pois, abrem diversas possibilidades de integração e desenvolvimentos de sistemas complexos. Sem contar o desenvolvimento já em curso da associação com a metodologia de *Model Checking*.

7. REFERÊNCIAS

ANTUNES, D. C. **Statecharts**. GPT. Notas de aula. 2018.

BELZER, J.; HOLZMAN, A. G.; KENT, A. **Encyclopedia of Computer Science and Technology**. p. 73. ISBN 978-0-8247-2275-3. ed. [S.l.]: CRC Press, v. 25, 1975.

BOOCH, G.; RUMBAUGH, J.; JACOBSON, I. **UML, Guia do Usuário: tradução**. Fábio Freitas da Silva. Rio de Janeiro. 2012.

BROWN, M. H. Exploring algorithms using Balsa-II. **In Computer**, v. 21, n. 5, p. 14-36, maio 1988.

CAMPBELL, W. Understanding State Machines, Part 3: Mealy and Moore Machines. **Videos and Webinars**, 201-. Disponivel em: <https://www.mathworks.com/videos/understanding-state-machines-mealy-and-moore-machines-3-of-4-90490.html>. Acesso em: 04 jun. 2020.

CILLIERS, C.; CALITZ, A.; GREYLING, J. The effect of integrating an iconic programming notation in CS1. **SIGCSE Conference on Innovation and Technology in Computer Science Education**, Monte de Caparica, Portugal, p. 89-93, 2005.

COQUAND, T. Finite Automata. **Automata**, 2009. Disponivel em: <Finite-State Machines>. Acesso em: 10 jun. 2020.

DIANA, J. Significado de Ciência da Computação. **Significados**, 2019. Disponivel em: <https://www.significados.com.br/ciencia-da-computacao/>. Acesso em: 02 jul. 2020.

EFFROS, M. Claude E. Shannon. **IEEE Information Theory Society**, 2017. Disponivel em: <https://www.itsoc.org/about/shannon>. Acesso em: 16 Junho 2020.

EHL, T. MathWorks. **Continental Develops Electronically Controlled Air Suspension for Heavy-Duty Trucks**, 2019. Disponivel em: <https://www.mathworks.com/company/user_stories/continental-develops-electronically-controlled-air-suspension-for-heavy-duty-trucks.html?s_tid=srchtitle>. Acesso em: 15 jun. 2021.

FEOFILOFF, P. Algoritmos para grafos - aulas. **Grafos**, 2017. ISSN IME-USP. Disponivel em: <https://www.ime.usp.br/~pf/algoritmos_para_grafos/aulas/graphs.html>. Acesso em: 30 maio 2020.

FOMIN, D.; GENKIN, S.; ITENBERG, I. **Círculos Matemáticos:** a Experiência Russa. 1ª edição. ed. Rio de Janeiro: AMS/IMPA, 1996. 292 p.

FRIEDL, J. **Mastering Regular Expressions**. 1. ed. Reilly Media, Inc: [s.n.], v. ISBN 0596528124.1, 2006.

GARDNER, H. **The Mind's New Science:** A History of the Cognitive Revolution. ISBN 978-0-465-04635-5. ed. p. 144: Basic Books, 1987.

GOMES, et al. Aprendizagem de programação de computadores: dificuldades e ferramentas de suporte. **Revista Portuguesa de Pedagogia**, Coimbra, Portugal, v. 42, p. 161-179, jul. 2008. Acesso em: 2021.

GRAICS, B. et al. Mixed-semantics composition of statecharts for the component-based design of reactive systems. **Softw Syst Model**, 2020.

GRIFFITHS, D. J. **Mecânica Quântica**. 2 ed. ed. [S.l.]: Pearson Education, 2011.

GUEDES, A. L. P. UFPR. **Noções Básicas Grafos**, 2001. Disponivel em: <http://www.inf.ufpr.br/andre/Disciplinas/BSc/CI065/michel/Intro/intro.html>. Acesso em: 16 jun. 2020.

HAREL, D. Statecharts: a visual formalism for complex systems. Science of Computer Programming, Israel, p. 231-274, 1987.

HAREL, D. et al. Integrating Inter-Object Scenarios with Intra-object Statecharts for Developing Reactive Systems. **IEEE Systems Journal**, Israel Science Foundation, 3 jul. 2020.

HOPCROFT, J. E.; RAJEEV, M.; ULLMAN, J. D. **Introduction to automata theory languages, and computation**. 3. ed. [S.l.]: Freeman, 2006.

IBM CORPORATION. Defining RIMB statecharts and their execution. **IBM Knowledge Center**, 2018. Disponivel em: <https://www.ibm.com/support/knowledgecenter/pt-br/SSB2MU_8.3.0/com.ibm.rhp.autosar.doc/topics/rhp_t_ext_rimbs_defining_statecharts.html>. Acesso em: 11 dez. 2020.

KHOURSHID, D. Crafting Stateful Styles with State Machines by David Khourshid. **CSSConf BP 2019**, 2019. Disponivel em: <https://www.youtube.com/watch?v=0cqeGeC98MA>. Acesso em: 22 jul. 2020.

KHOURSHID, D. Formal Forms with State Machines. **React Next 2019**, 2019. Disponivel em: <https://www.youtube.com/watch?v=hiT4Q1ntvzg>. Acesso em: 21 jul. 2020.

LEWIS, H. R.; HARILAOS,. **Elements of the Theory of Computation**. [S.l.]: Englewood Cliffs, NJ, v. Papadimitriou, 1981.

MATLAB. MathWorks. **User Stories**, 2021. Disponivel em: <https://www.mathworks.com/company/user_stories/search.html?q=stateflow&page=1>. Acesso em: 15 jun. 2021.

MEALY, G. H. “A Method for Synthesizing Sequential Circuits”, Bell System Technical Journal, 1955, 34, (5), pp. 1045–1079, doi: 10.1002/j.1538-7305.1955.tb03788.x 2: [s.n.], 1955.

MIERLO, S. V.; VANGHELUWE, H. Introduction to Statecharts Modeling, Simulation, Testing, and Deployment. **Winter Simulation Conference (WSC)**, National Harbor, MD, USA, 08 dez. 2019.

MOGENSEN, E. Welcome to the world of Statecharts. **Statecharts**, 2020. Disponivel em: <https://statecharts.github.io/>. Acesso em: 2020 jul. 27.

MOORE, E. F. “Gedanken-experiments on Sequential Machines”. Princeton, N.J.: Princeton University Press. Automata Studies, Annals of Mathematical Studies (34): 129–153: [s.n.], 1956.

MORAIS, C. G. B.; MENDES, F. M. N.; OSÓRIO, A. J. M. Difficulties and challenges in the learning process of algorithms and programming in higher education: a systematic literature review. **Research, Society and Development**, v. 9, n. 10, p. e9429109287, 2020.

NEME, J. H. Z. **Um Estudo Da Aplicação Do Padrão AUTOSAR Para Desenvolvimento De Funções Em Sistemas Embarcados Automotivos**. Dissertação de mestrado - Universidade Tecnológica Federal do Paraná. Ponta Grossa - PR, p. 97. 2017.

PIANO, D. K. The visual future of reactive applications with statecharts. **Ближайшая конференция — HolyJS**, 2019. Disponivel em: <https://www.youtube.com/watch?v=o84Xw8qiTCw>. Acesso em: 27 jul. 2020.

POUNDSTONE, W. Fortune's Formula : The Untold Story of the Scientific Betting System That Beat the Casinos and Wall Street.. **Hill & Wang**, 2005. ISSN ISBN 978-0-8090-4599-0.

ROCHEL, L. C. E. Desempenho Setorial. **Associação Brasileira da Indústria Elétrica e Eletrônica**, 2020. Disponivel em: <http://www.abinee.org.br/abinee/decon/decon15.htm#:~:text=O%20faturamento%20da%20Ind%C3%BAstria%20Eletroeletr%C3%B4nica,Pre%C3%A7os%20ao%20Produtor%20do%20IBGE).>. Acesso em: 12 jul. 2020.

SCHULZ-ROSENGARTEN, A.; SMYTH, S.; MENDLER, M. Towards Object-Oriented Modeling in SCCharts. **IEEE**, Southampton, United Kingdom, 21 out. 2019.

SHANNON, C. E. A symbolic analysis of relay and switching circuit. **Massachusetts Institute of Technology, Dept. of Electrical Engineering**, Massachusetts, 10 Agosto 1937. Tese. p - 72.

SHIMAHARA, A. Desafios Model based design. **OpenCadd**, 2020. Disponivel em: <https://opencadd.com.br/events/desafios-model-based-design/>. Acesso em: 16 jun. 2021.

SILVA, G. K. D. **Programação de Computadores**. FIC- Faculdades Integradas de Caratinga. Caratinga MG, p. 12. 2017.

SIPSER, M. **Introduction to the Theory of Computation**. 1. ed. Thomson Publishing: [s.n.], 1996.

TIRONI, P. I. D. O.; CAMPOS, G. L.; OLIVEIRA, P. S. D. Aprendizagem Ativa Para Sistemas Embarcados: Uma Abordagem com SIMULINK/STATEFLOW. **XLVIII Congresso Brasileiro de Educação em Engenharia (COBENGE) e III Simpósio Internacional de Educação em Engenharia da ABENGE**, Bento Gonçalves - RS, p. 10, dez. 2020. Disponivel em: <https://www.researchgate.net/publication/347522943_Aprendizagem_Ativa_Para_Sistemas_Embarcados_Uma_Abordagem_com_SIMULINKSTATEFLOW>. Acesso em: 19 jun. 2021.

VAN DER MEER,. MathWorks. **3T Develops Robot Emergency Braking System with Model-Based Design**, 2021. Disponivel em: <https://www.mathworks.com/company/user_stories/3t-develops-robot-emergency-braking-system-with-model-based-design.html?s_tid=srchtitle>. Acesso em: 15 jun. 2021.

VIERA, N. J. **Linguagens Formais e Automatos**. 1. ed. Capt 2: Thomsom, 201-.

WAGNER, F. StateWORKS: Moore or Mealy model? **StateWORKS**, 2005. Disponivel em: <http://www.stateworks.com/technology/TN10-Moore-Or-Mealy-Model/>. Acesso em: 20 maio 2020.

WHITE, E. **Making Embedded Systems:** Design Patterns for Great Software. [S.l.]: O'Reilly Media , v. ISBN13: 9781449302146, 2012.

WIKIPEDIA. State diagram. **Wikipedia**, 2020. Disponivel em: <https://en.wikipedia.org/wiki/State_diagram>. Acesso em: 01 jun. 2020.

WINSKEL, G. **The Formal Semantics of Programming Languages:** An Introduc-tion. 1. ed. MIT Press: [s.n.], 1993.

YOUSEFZADEH, F. Tackle React component complexity using Reactive Statecharts. **Reactjsday 2019**, 2019. Disponivel em: <https://www.youtube.com/watch?v=2GHtUzDLfwQ>. Acesso em: 21 jul. 2020.

ZOCKOLL, ; SCHEITHAUER, A.; DEKKER, M. D. History of Object Oriented Modeling languages. **Modeling languages history**, 2012. Disponivel em: <https://en.wikipedia.org/wiki/File:OO_Modeling_languages_history.jpg#file>. Acesso em: 23 maio 2020.

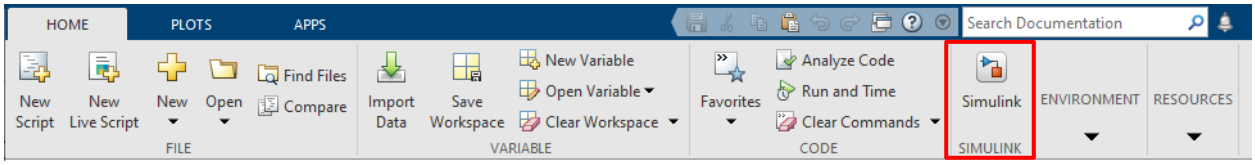
8. ANEXO

Neste capitulo serão apresentados roteiros interativos para que o leitor consiga ler, compreender e replicar em seu computador. O intuito é apresentar exemplos didáticos com os principais conceitos do Stateflow como forma de aprendizado, de modo que, o leitor se torne apto a desenvolver seus próprios projetos. Cita-se que o método descrito aqui tem com intuito a capacitação acerca da ferramenta Stateflow, e todos os sistemas são ilustrativos. Fato, que serve como motivação ao leitor em que na sua execução faça modificações para que fique mais próximo possível a um sistema real.

8.1. Procedimento experimental: Alarme

Primeiramente é necessário criar um modelo *Simulink* no ambiente de trabalho do MATLAB, especificamente na *Toolstrip*. Este se encontra na aba *Home*, conforme expõe a Figura 38.

Figura 38 - Página inicial do ambiente de trabalho do MATLAB.



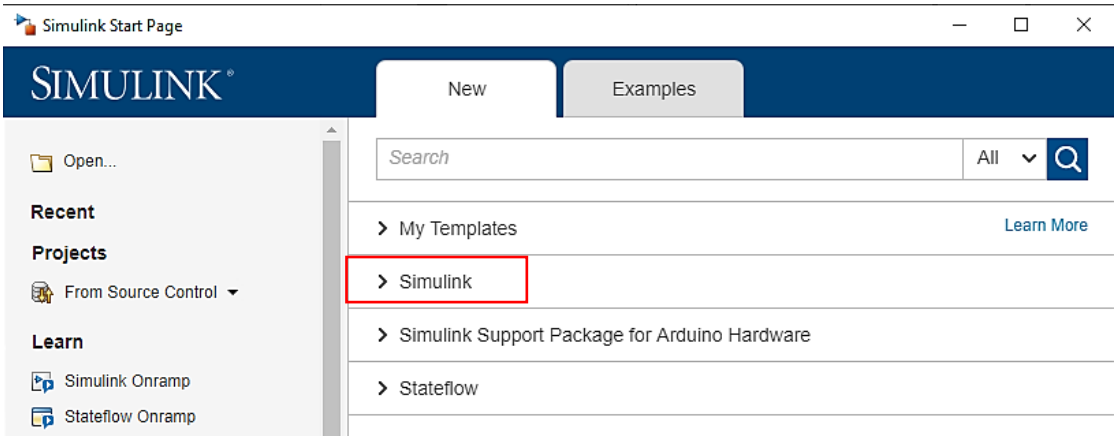
Fonte: Elaborado pelo autor, 2021.

Ao selecionar o campo a página inicial de trabalho do Simulink irá abrir conforme demonstra a Figura 39, e nela encontra-se diversos *menus* como Novos *templates*, exemplos da comunidade e plataformas de aprendizado. Para iniciar um projeto dentro do menu *New* selecione subitem *Simulink*.

Ao selecionar o subitem abre-se diversos modelos pré-desenvolvidos, a fim de facilitar a vida do programador, conforme apresenta a

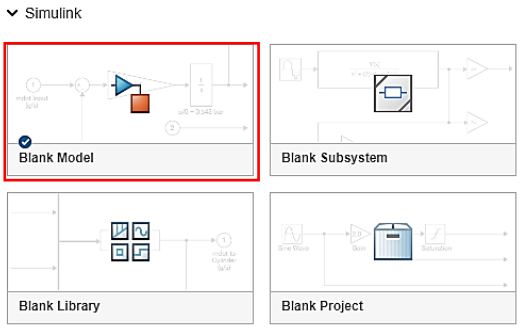
Figura 40. Neste tutorial com o intuito de documentar todos os passos de desenvolvimento, é preciso selecionar um modelo em branco.

Figura 39 - Página inicial do ambiente de trabalho do Simulink.



Fonte: Elaborado pelo autor, 2021.

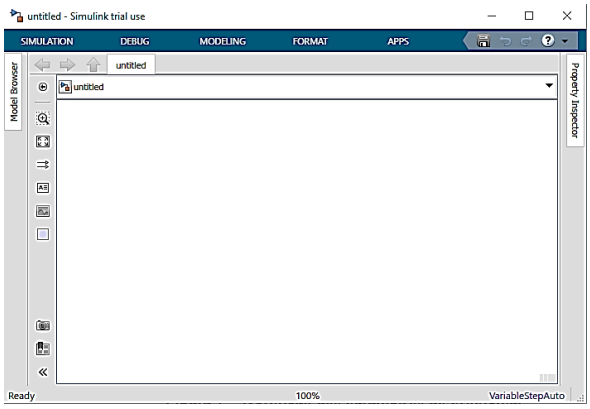
Figura 40 - Exemplos de pré modelos na interface do Simulink.



Fonte: Elaborado pelo autor, 2021.

Seleciona-se um modelo em branco e uma nova janela sobrepõe a atual, conforme a Figura 41, no qual é exposto o ambiente de desenvolvimento de programas do *Simulink*. E será a prancheta para trabalho dos sistemas reativos.

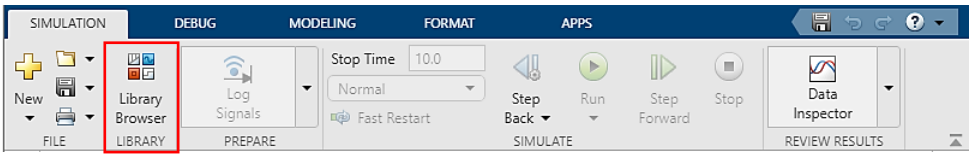
Figura 41 - Ambiente de desenvolvimento do Simulink.



Fonte: Elaborado pelo autor, 2021.

Para adicionar elementos nesta página em branco, é necessário dentro da aba *Simulation* selecionar o ícone *Library Browser* onde todos os elementos estão dispostos, este passo é demonstrado na Figura 42.

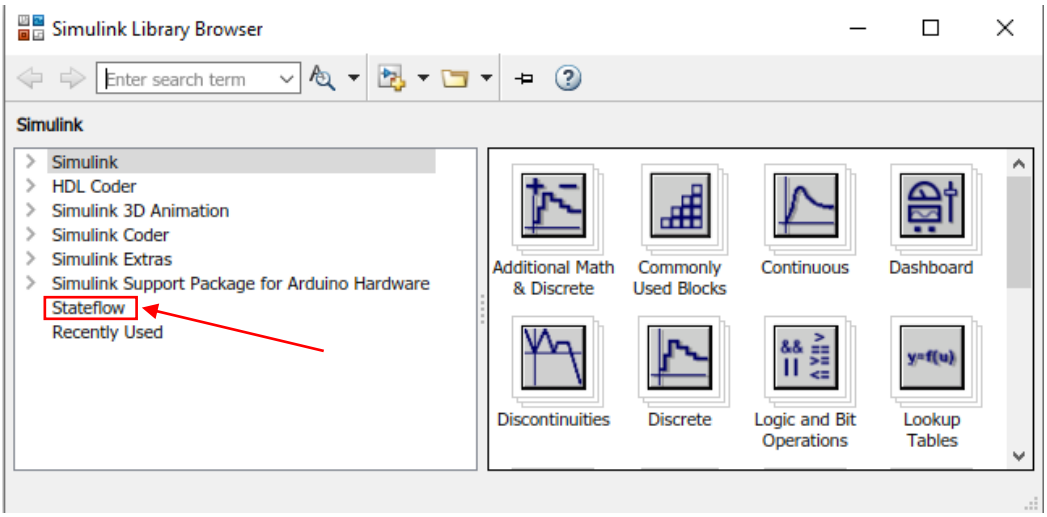
Figura 42 - Indicação do Library Browser para inserção de componentes.



Fonte: Elaborado pelo autor, 2021.

Com o *menu* aberto, visualiza-se uma gama de elementos divididos nas mais diversas funcionalidades, conforme apresenta a Figura 43. Além destes elementos, é possível inserir outros com bibliotecas e pacotes adicionais do *software*.

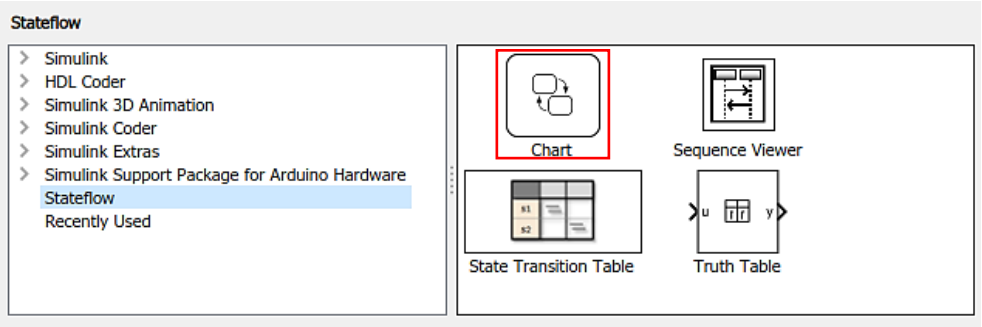
Figura 43 - Diversos elementos de blocos para modelos Simulink presentes no pacote Simulink Essentials.



Fonte: Elaborado pelo autor, 2021.

Seleciona-se o ícone *Stateflow*, que será foco de trabalho no desenvolvimento do sistema embarcado, como demonstra a Figura 44. O *menu* mostra ao usuário diversas ferramentas dentro da *Library Browser* para trabalhar com o Simulink. Para desenvolvimento de aplicações com *statecharts*, é necessário selecionar a opção *chart* que representa um modelo de máquina de estados de Harel sem nenhuma configuração prévia.

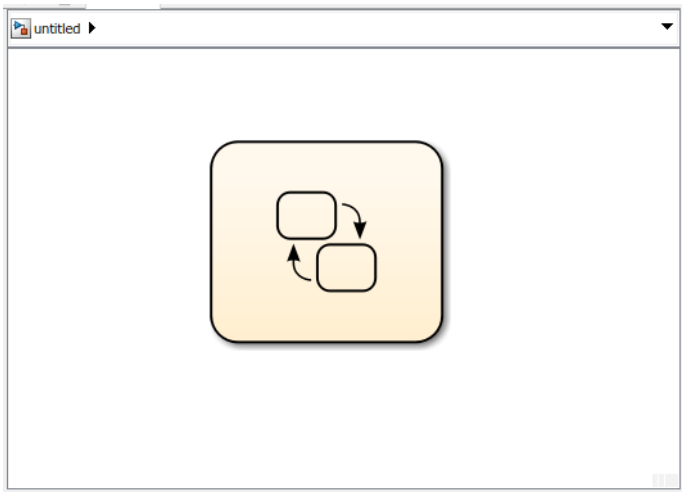
Figura 44 - Página inicial do ambiente de trabalho do MATLAB.



Fonte: Elaborado pelo autor, 2021.

Para selecionar um bloco e começar a trabalhar com o mesmo, é necessário que o usuário o selecione com o *mouse* (ou *touchpad*) e o arraste para o ambiente de trabalho do Simulink. Conforme apresenta a Figura 45 em que foi inserido um bloco *chart* na área de trabalho do Simulink.

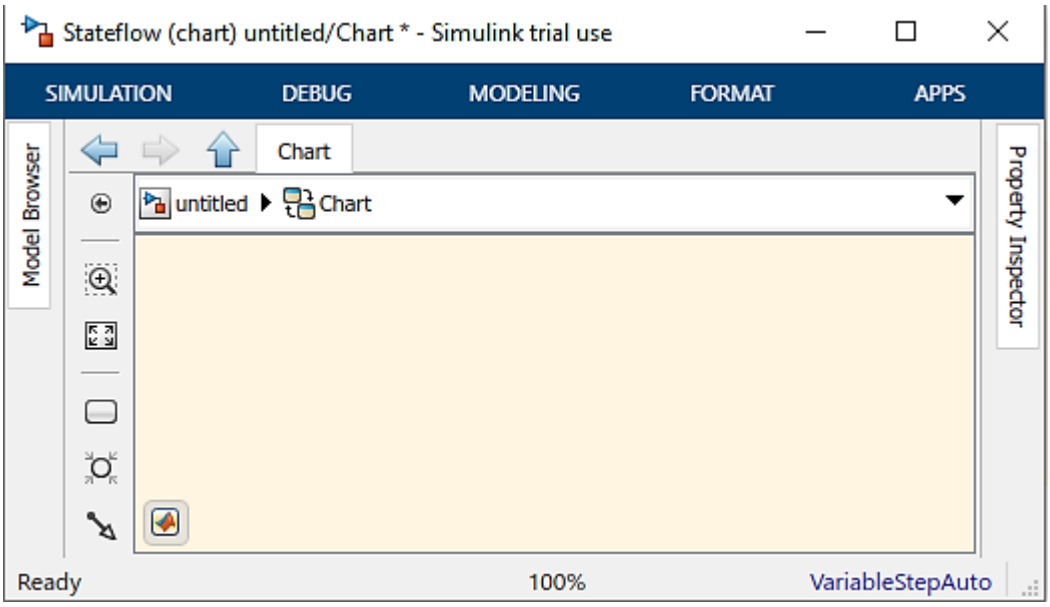
Figura 45 - Modelo em branco com inserção de um bloco no ambiente do Simulink.



Fonte: Elaborado pelo autor, 2021.

Ao pressionar o botão do *mouse* por duas vezes sobre o bloco, é apresentado o ambiente de trabalho do Stateflow e sua cor é dada por um tom de pêssego claro, conforme expõe a Figura 46. Além disto, é possível ver a indicação na *Explore Bar* a hierarquia entre os modelos Simulink e Stateflow.

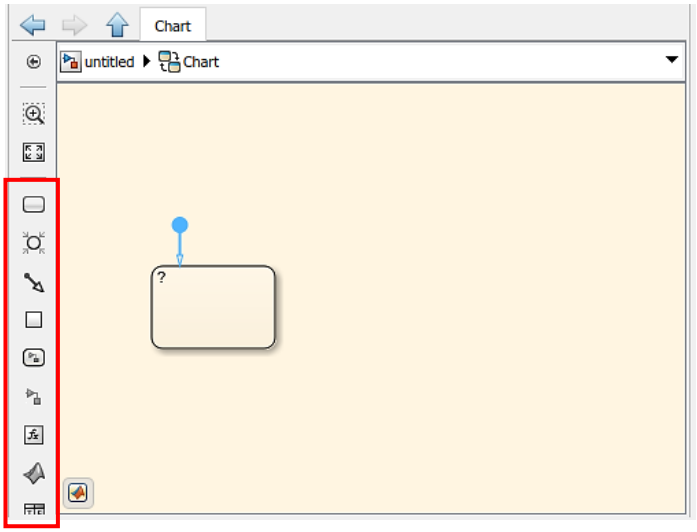
Figura 46 - Ambiente de trabalho do Stateflow.



Fonte: Elaborado pelo autor, 2021.

Dentro do ambiente de trabalho do Stateflow também existe a opção de inserção de diversos blocos, estes estão a esquerda do ambiente de trabalho (na *Object Pallet*) conforme apresenta a Figura 47. Para adicionar um item o usuário deve selecionar qual bloco quer trabalhar e arrastá-lo para sua área de trabalho dentro do Stateflow. Primeiramente é necessário adicionar um estado, que será o estado inicial.

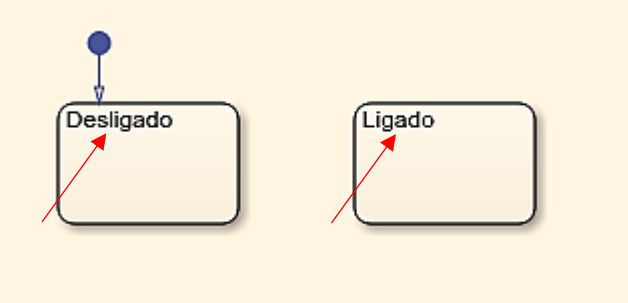
Figura 47 - Insere-se o estado inicial no ambiente de trabalho Stateflow.



Fonte: Elaborado pelo autor, 2021.

Quando o primeiro estado é inserido uma seta direcionada de um círculo para o estado é apontada para este, isto indica que é o estado inicial. Insira outro estado e nomeie-o, conforme presente na Figura 48.

Figura 48 - Nomeação dos estados no ambiente do Stateflow.

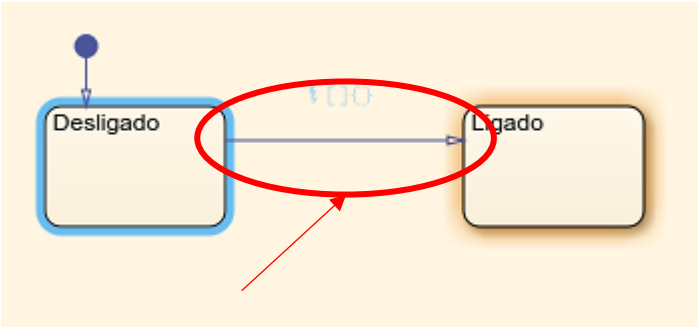


Fonte: Elaborado pelo autor, 2021.

Neste trabalho, conforme a descrição matemática indica os estados se comunicam através de transições baseadas nas entradas do sensor. Ou seja, caso o sistema esteja em um determinado estado e aconteça um evento externo o sistema irá identificar se é necessário realizar uma transição para um estado distinto ao atual.

Conforme aponta a Figura 49, é necessário interligar os estados com transições e estas são inseridas quando é aproximado o *mouse* de um estado até a forma dele ficar em cruz e assim pode-se arrastar para o lado e conectar um estado ao outro. Os passos são executados em ambas as direções.

Figura 49 - Inserindo uma transição.

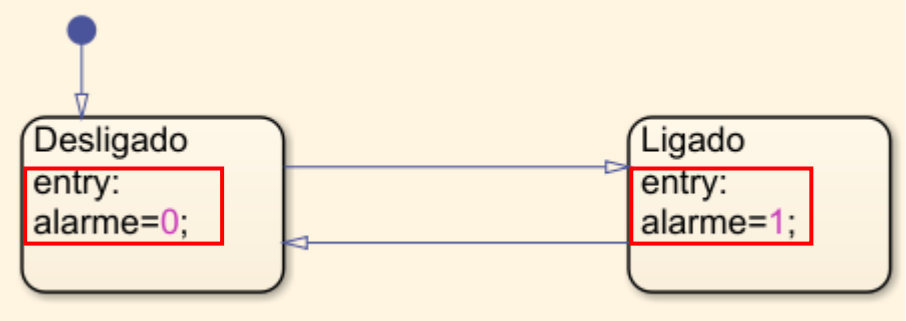


Fonte: Elaborado pelo autor, 2021.

Dentro dos estados são definidas ações para que aconteçam quando a simulação estiver em cada um dos estados. Ou seja, quando o ambiente for simulado e estiver dentro do estado “Desligado” irá acontecer uma ação que está descrita dentro do mesmo. De acordo com os dados apresentados no problema, dentro de cada estado, deve-se indicar se o alarme deve

soar ou não. Isto é feito ao adicionar uma condição na entrada com a função *entry*: que irá executar assim quando o estado é iniciado. Para isto, é preciso dar um clique dentro do estado e digitar todas as funções que o mesmo irá executar. O passo é demonstrado na Figura 50.

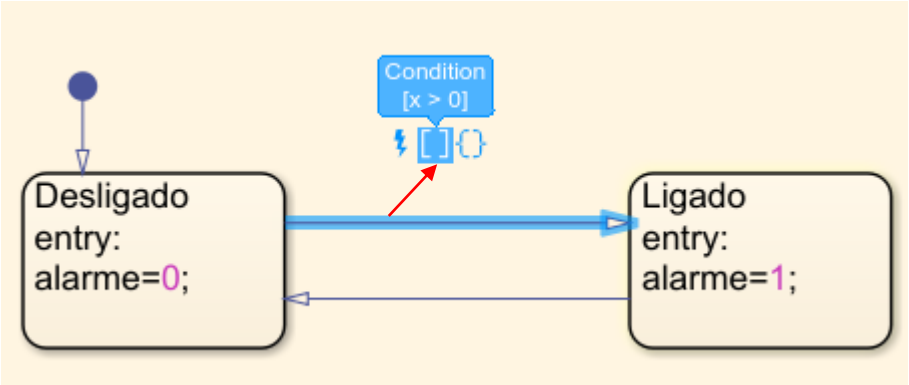
Figura 50 - Ações são definidas dentro dos estados.



Fonte: Elaborado pelo autor, 2021.

Após definir ações que irão acontecer dentro dos estados, é primordial definir como os estados irão se comunicar entre si nesta *statechart*. Para isto, conforme apresenta a Figura 51, é determinado uma transição de condição, ou seja, quando aquele parâmetro for satisfeito na entrada do sistema irá ocorrer uma transição no diagrama de estados.

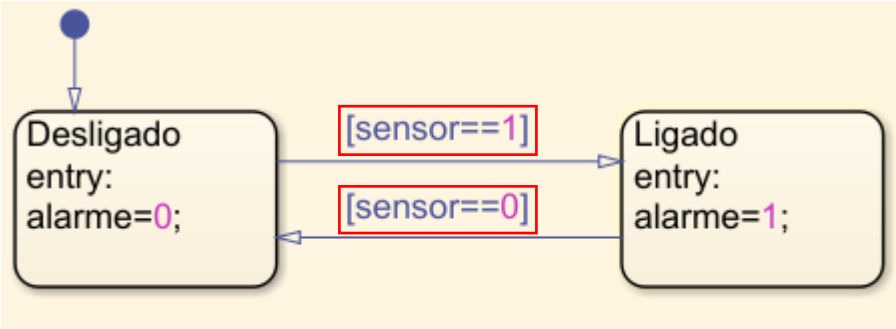
Figura 51 - Condições de transições entre estados devem ser representadas dentre colchetes.



Fonte: Elaborado pelo autor, 2021.

A condição para que aconteça uma transição é: se o sensor de presença acusar que existe alguma movimentação, isto é dado pelo valor lógico 1, o alarme é ligado. Caso o sensor acuse nenhuma movimentação, seu valor lógico estará em 0, e o sistema de alarme pode ser desligado. Isso é representado na Figura 52 nos diagramas do *Stateflow*.

Figura 52 - Sistema descrito pelo Stateflow.

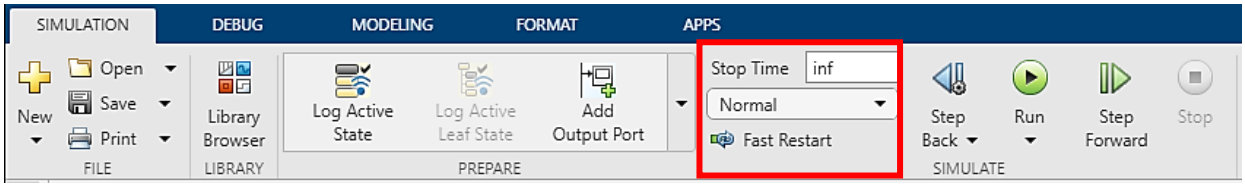


Fonte: Elaborado pelo autor, 2021.

Logo, todo o sistema foi descrito, porém é necessário ajustar alguns parâmetros tal que a simulação seja executada com perfeição.

Primeiro defina o tempo de simulação, conforme é indicado na Figura 53 dentro modelo *Stateflow* na *Toolstrip bar*. Depois na aba de configurações *Simulation* e, na subárea *Simulate* existe a opção de trocar o tempo padrão que é 10 para "inf" (abreviatura de *infinity*) que significa que a simulação irá ocorrer até que o usuário a pause ou encerre (infinitamente).

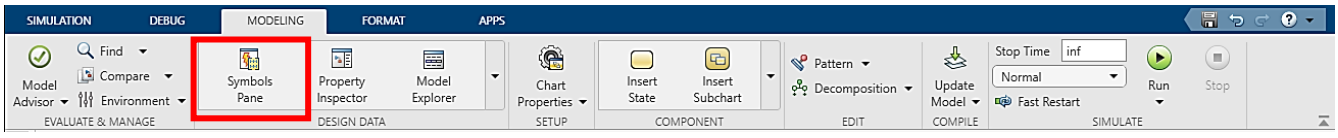
Figura 53 - Configuração do tempo de simulação do sistema em análise.



Fonte: Elaborado pelo autor, 2021.

Para ajustar as variáveis que estão no ambiente *Stateflow*. Selecione a aba *Modeling* e no item *Design Data* clique em *Symbols Pane*, conforme mostra a Figura 54.

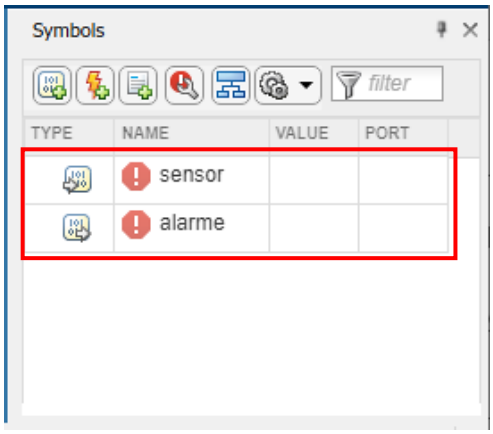
Figura 54 - Aba *Modeling* do Stateflow.



Fonte: Elaborado pelo autor, 2021.

Irá abrir uma nova janela com nome de *Symbols* e apresentará as variáveis do ambiente Stateflow, conforme apresenta a Figura 55. É valido ressaltar que a variável sensor é de entrada, por isto na coluna *Type* o símbolo é uma seta entrando no bloco, e o mesmo também pode ser visto na variável de saída alarme que tem uma seta apontando para fora do bloco.

Figura 55 - Nova janela criada pela aba *Modeling*.



Fonte: Elaborado pelo autor, 2021.

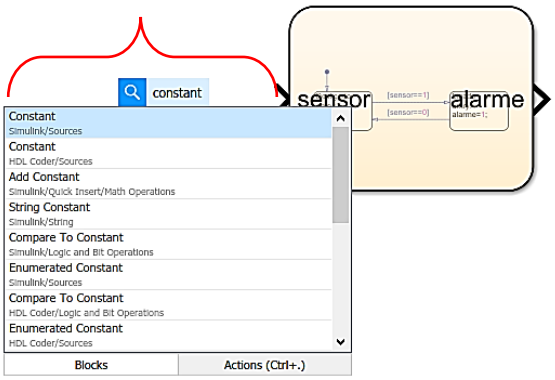
Como destacado na Figura 55, alguns símbolos indicam que estão indefinidos, para isto basta selecionar o botão com chave fixa (“chave de boca”), conforme apresenta a Figura 56 e irá configurar automaticamente estas variáveis.

Figura 56 - Corrigir símbolos com indefinidos.

Fonte: Elaborado pelo autor, 2021.

Com este último passo, indicado na Figura 57, o sistema está finalizado. Agora é necessário desenvolver o ambiente Simulink para que possibilite seu funcionamento. Primeiramente, retorne ao ambiente de trabalho do Simulink e insira o bloco constante que terá valores booleanos dos dados recebidos pelo sensor. Para isto, dê um clique em um local em branco no ambiente de trabalho do Simulink e digite *constant* e selecione a opção que contém como sub menu *Simulink/Sources*.

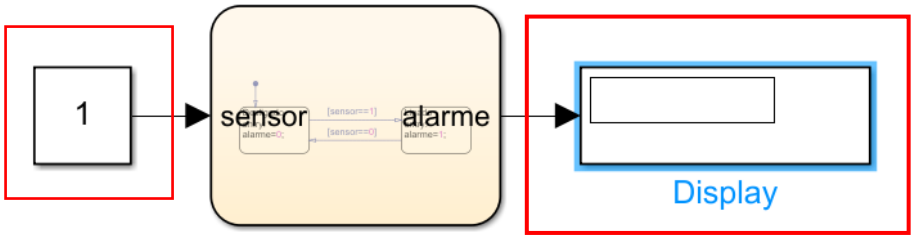
Figura 57 - Ambiente Simulink, procedimento para inserir o bloco constante.



Fonte: Elaborado pelo autor, 2021.

Após inserir adicione com o mesmo processo o bloco *display* (ou seja, ao dar um clique em um local em branco e digite o nome do bloco). Inseridos os dois blocos conecte ambos ao bloco *chart* para isso deve-se aproximar o cursor do bloco pretendido e arrastar a seta até a entrada/saída correta. Visualmente o procedimento citado tem resultado conforme demonstrado na Figura 58.

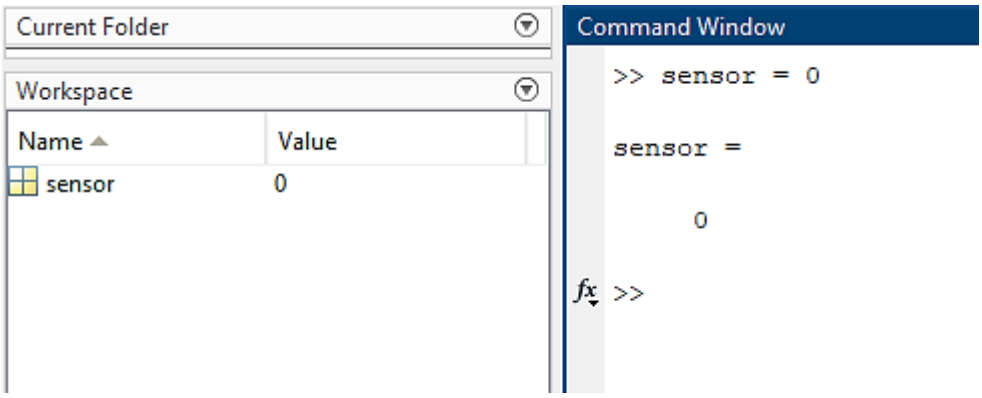
Figura 58 - Adiciona-se o bloco display.



Fonte: Elaborado pelo autor, 2021.

Agora no ambiente de trabalho do MATLAB é necessário configurar a variável sensor. Para isto basta digitar no *Command Window* (menu do MATLAB) o seguinte comando: “sensor=0”. Conforme apresenta a Figura 59, o *software* irá adicionar uma nova variável na janela *Workspace* com seu respectivo valor.

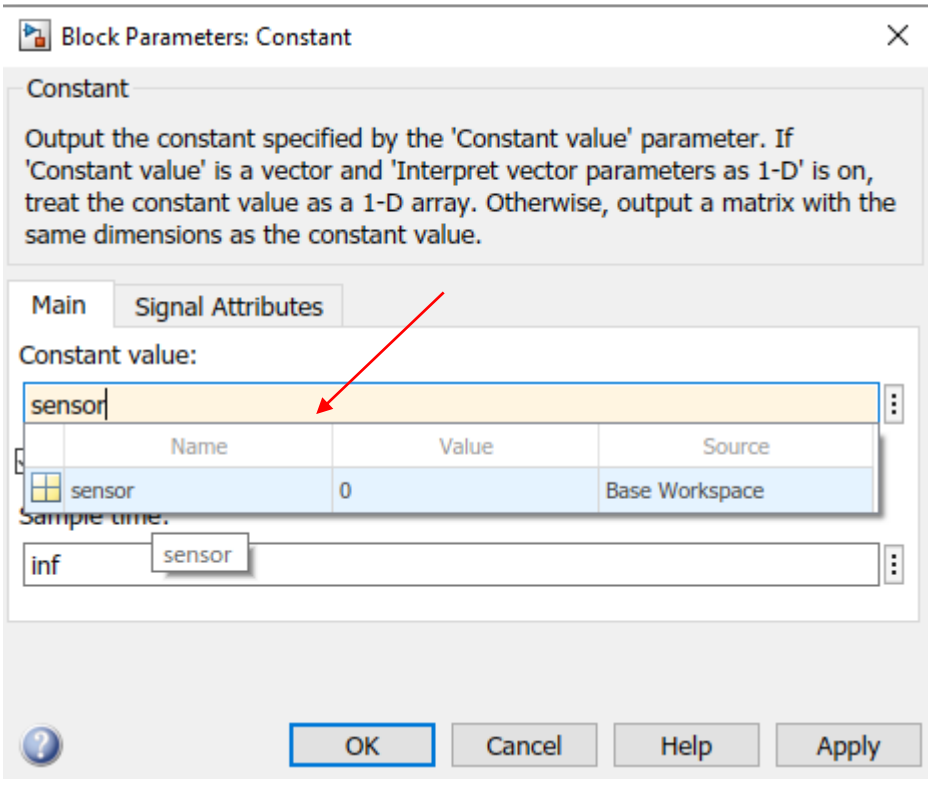
Figura 59 - Configurando a variável auxiliar.



Fonte: Elaborado pelo autor, 2021.

Retorne para a janela de trabalho do *Simulink* para configurar o bloco constante relacionando-o ao valor presente no *Workspace*. Para isso, dê um clique duplo sobre o bloco, e irá abrir uma nova janela nomeada de *Block Parameters: Constant* e defina o valor da constante como sensor (nome da variável). Estes passos são executados no local indicado por *constant value*, e é apresentado pela Figura 60.

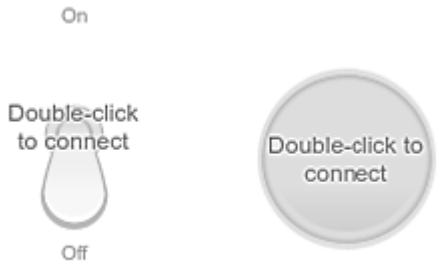
Figura 60 - Configurando o bloco constante.



Fonte: Elaborado pelo autor, 2021.

Após configurado confirme os passos anteriores, insira os *dashboards* que estão na *Library Browser* no sub menu *dashboards* e deve ser inseridos com o procedimento clique e arraste. Neste exemplo de desenvolvimento, escolha os blocos *toggle switch* e *lamp*, conforme apresenta a Figura 61.

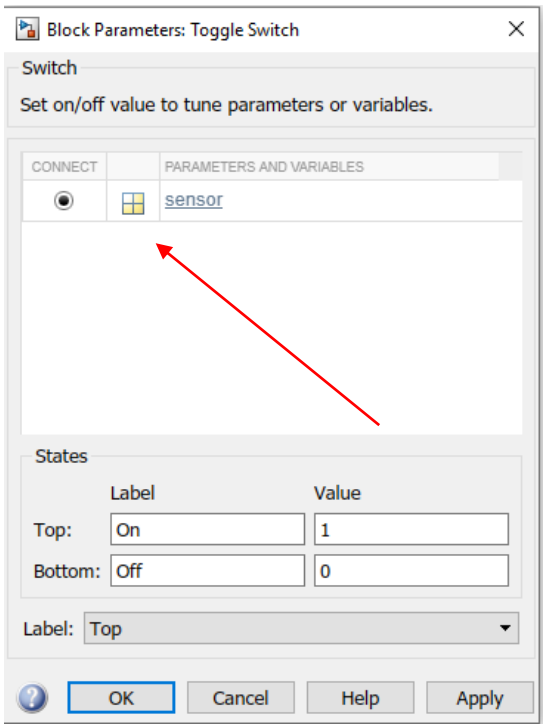
Figura 61 - Blocos de interação com o usuário e para controle do sistema.



Fonte: Elaborado pelo autor, 2021.

Com um clique duplo no bloco de chave pode-se alterar suas configurações. Primeiramente é necessário dar um *refresh* em todo o sistema, isto é feito com o comando "CTRL+T", e após a página atualizar conecte a *dashboard radio button* conforme mostra a Figura 62.

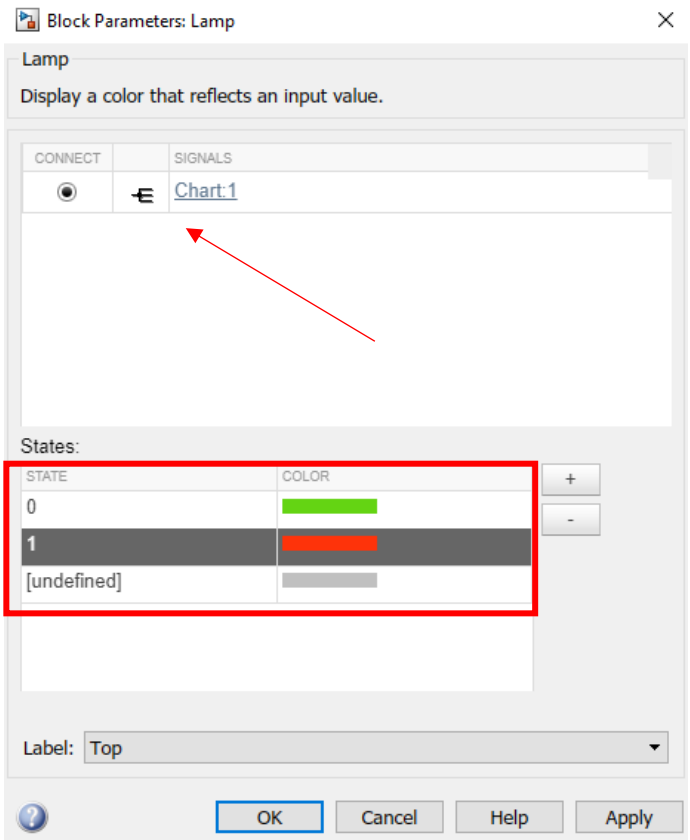
Figura 62 - Conecte o *Toggle Switch* com a variável sensor.



Fonte: Elaborado pelo autor, 2021.

É preciso conectar a *Dashboard Lamp* com a saída referente ao *display*. Para este fim, é necessário dar um duplo clique no bloco e configurar diferentes cores para diferentes estados. Conforme a Figura 63, configurou-se a cor verde para quando o alarme está desligado e vermelho para quando ele é acionado, para adicionar outros estados basta selecionar o botão "+". Para conectá-lo a um sinal deve-se deixar a janela *Block Parameters: Lamp* aberta, e na aba do *Simulink*, e dar um duplo clique no bloco *display*. Para conferir se a ação foi executada corretamente compare as configurações com a Figura 63.

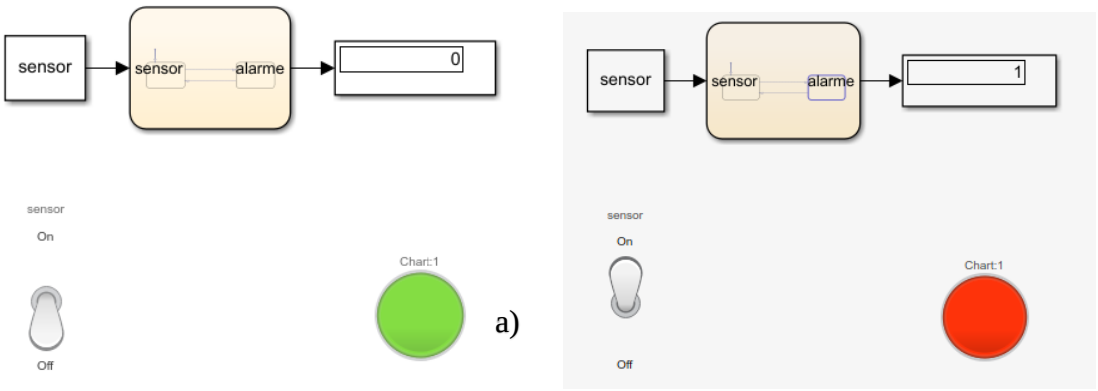
Figura 63 - Configurações do bloco Lamp.



Fonte: Elaborado pelo autor, 2021.

Todas as configurações necessárias para este modelo foram executadas. Após, na aba *Simulation* com um clique em *run* é possível compilar o programa e verificar a execução a *statechart*, conforme a Figura 64.

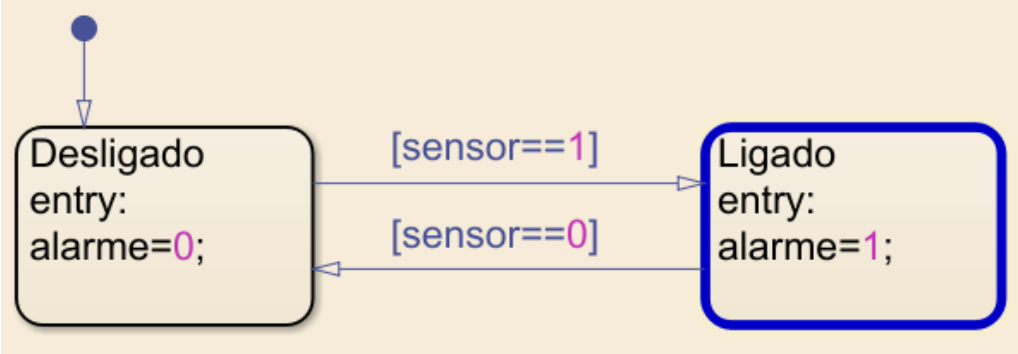
Figura 64 - Sistema em funcionamento com sensor acusando (a) nível 0 e (b) nível 1.



Fonte: Elaborado pelo autor, 2021.

Para visualizar a *statechart* dê um clique no bloco *chart* e entre no ambiente da Stateflow, neste será possível analisar o estado atual e as transições que ocorrem conforme mostra a Figura 65.

Figura 65 - Ambiente Stateflow com sistema em funcionamento.



Fonte: Elaborado pelo autor, 2021.

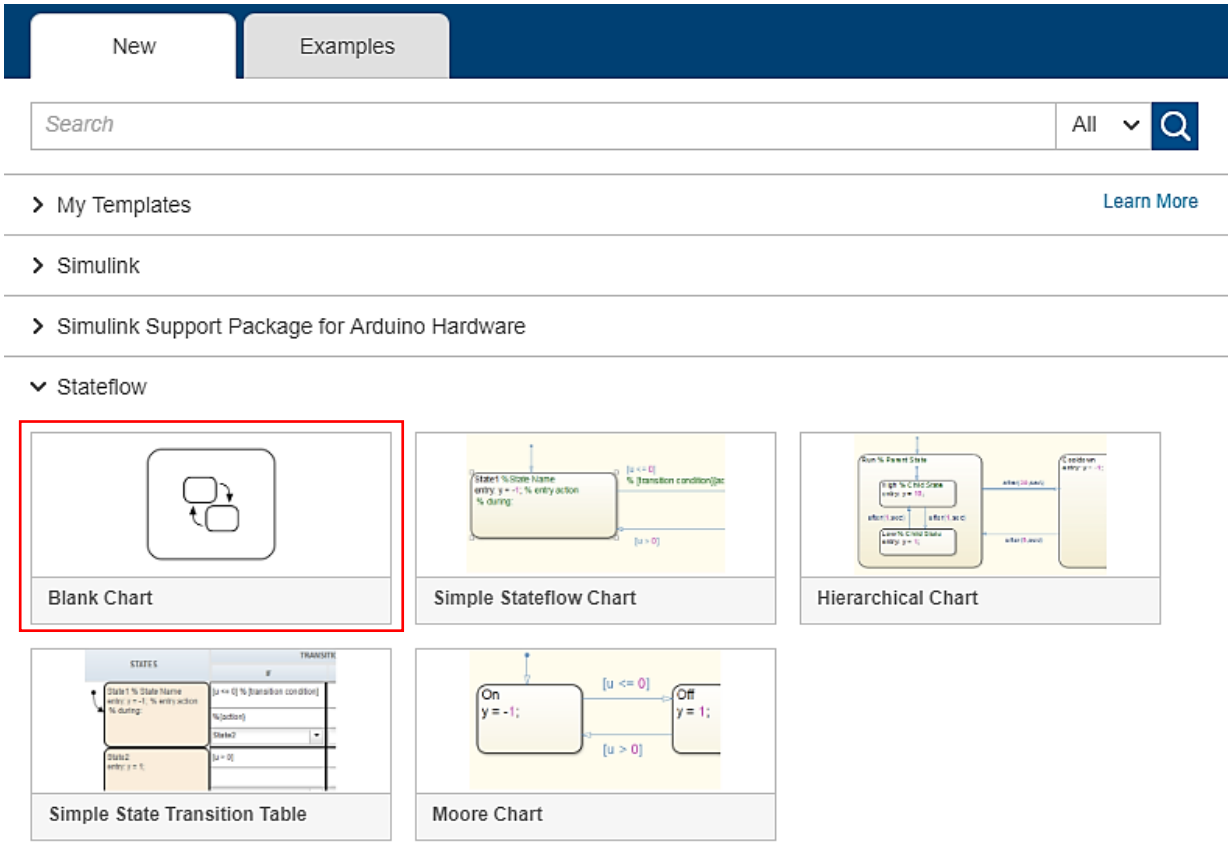
Logo, com o desenvolver dos passos apresentados é possível elaborar um modelo básico de uma *statechart* dentro do ambiente de simulação do MATLAB. É válido destacar que, mesmo sendo um modelo simples este é caracterizado por ser uma máquina de estados não determinística. Para torná-lo determinístico, basta adicionar transições nos estados Desligado e Ligado tal que retornem para si mesmas quando não for necessária realizar uma transição.

Portanto, este roteiro teve como objetivo apresentar os conceitos básicos sobre a elaboração de estados e transições dentro do ambiente Stateflow. E também apresentar a simulação de um sistema reativo dentro do modelo Simulink.

8.2. Procedimento experimental: Elevador

Primeiramente é necessário criar um modelo Simulink no ambiente de trabalho do MATLAB. Ao abrir a nova janela de modelos de Simulink, selecione o modelo *Stateflow* nomeado de *Blank Chart*, conforme apresenta a Figura 66. Esse modelo já vem com o bloco de trabalho do Simulink.

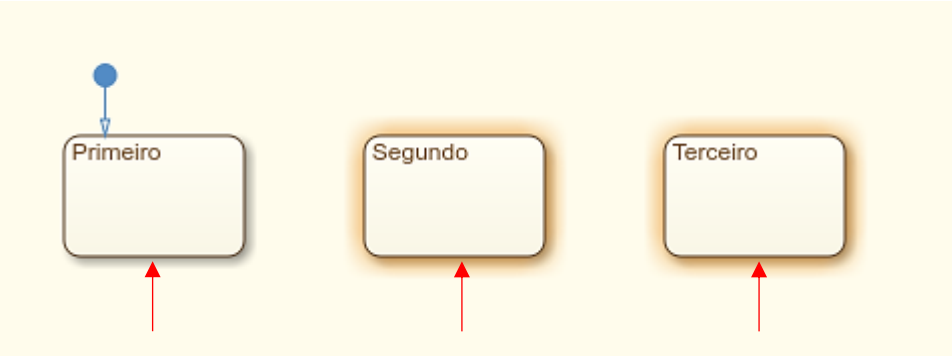
Figura 66 - Criando um modelo Stateflow nomeado de *Blank Chart*.



Fonte: Elaborado pelo autor, 2021.

Após o ambiente de trabalho *Simulink* ser aberto, dê um clique duplo no bloco *chart* que irá estar em branco. Logo, para construir os estados do elevador, conforme descrição matemática (sistema terá três estados referentes a cada andar), para isto basta na *object pallet* selecionar o ícone *State* e, arraste para o ambiente de trabalho de Stateflow conforme apresenta a Figura 67.

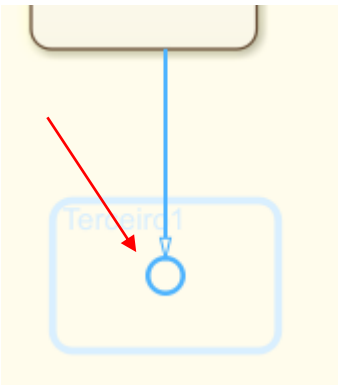
Figura 67 - Inserindo os três estados para o elevador no ambiente do Stateflow.



Fonte: Elaborado pelo autor, 2021.

Tendo os estados dispostos é possível inserir as transições. Este passo é feito como no exemplo do alarme aproximando o *mouse* da extremidade do estado e quando o cursor se transformar em sinal de "+" arrastá-lo com o botão esquerdo pressionado. Porém, neste exemplo não será conectado a outro estado, e sim adicionar uma *junction* de transição, conforme apresenta a Figura 68. Esta tarefa é feita ao arrastar a transição para um local vazio e o *software* dará a opção de adicionar o fim de transição em um *state* ou em uma *junction*. Mas caso não seja uma versão mais atual do *software* é necessário inserir primeiramente uma *junction* que está presente na *object pallet* arrastá-la para o local apropriado na área de trabalho e por fim interligar um *state* até esta por meio de um transição.

Figura 68 - Passo para adicionar uma *junction* de transição.

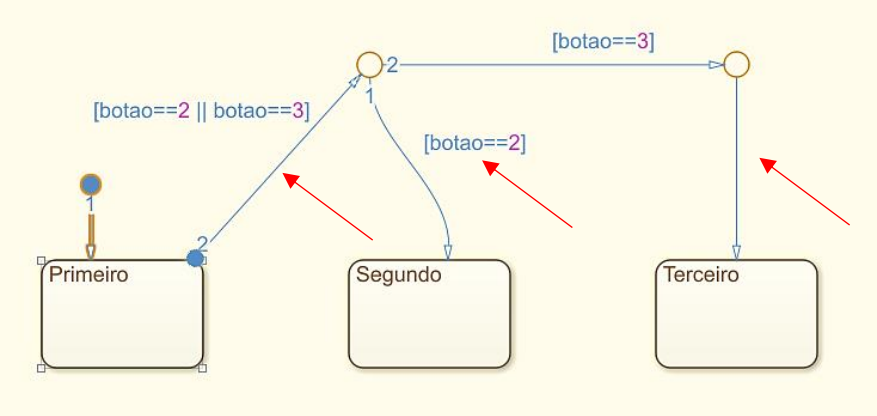


Fonte: Elaborado pelo autor, 2021.

A disposição das *junctions* tem como intuito neste sistema atuar como estrutura de decisão. É possível desenvolver uma *statechart* das mais diversas formas e níveis de complexidade, e trata-se "assim como uma obra de arte um belo processo criativo desenvolver *statecharts*" (WAGNER, 2005), e neste exemplo, deve-se imaginar que o estado atual é o primeiro andar e o usuário deseja subir. Para isto ele tem duas opções, ou para o segundo andar

ou para o terceiro, logo, adiciona-se um caminho que terá essa condição. Entretanto, após essa condição ser aceita ele terá duas opções de estados (*states*) para prosseguir, são esses: segundo ou terceiro andar, e, portanto, cada um terá seu "caminho" (transição) separada. A Figura 69 demonstra esse processo de alocação das transições.

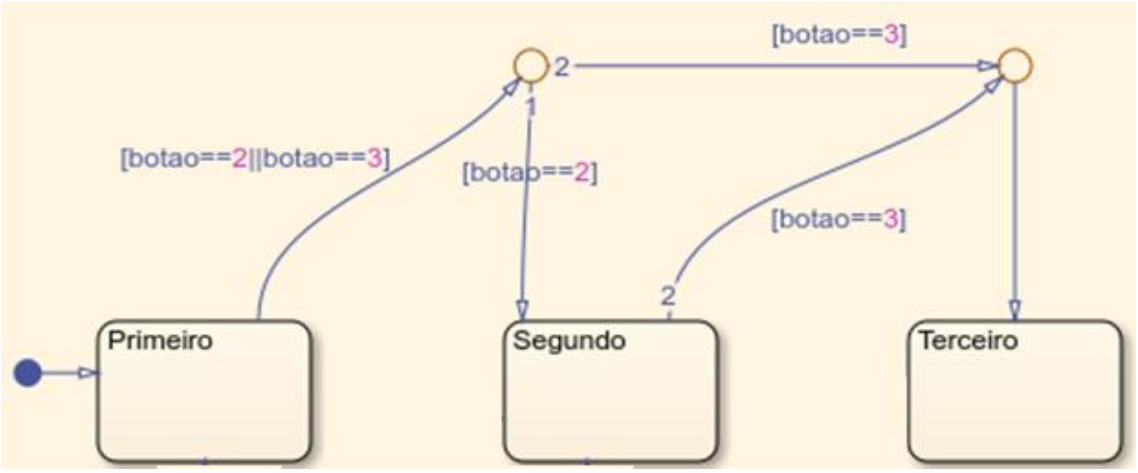
Figura 69 - Passo para criar caminhos para as transições chegarem aos seus respectivos *states*.



Fonte: Elaborado pelo autor, 2021.

Após adicionadas as transições é crucial colocar todas as condições, conforme apresenta a Figura 70 e foi descrito no parágrafo anterior. Vale observar que, para uma transição estar corretamente alocada em um estado ela não deve ter nenhuma indicação de bola azul, é possível observar isto na Figura 69 em que foi colocado propositalmente no *state* do primeiro andar uma conexão errônea, e foi corrigida na Figura 70 para o modo correto de conectar, isto acontece por bugs no sistema que não entende a transição saindo do estado e sim saindo de lugar algum assim como acontece na inserção do primeiro estado.

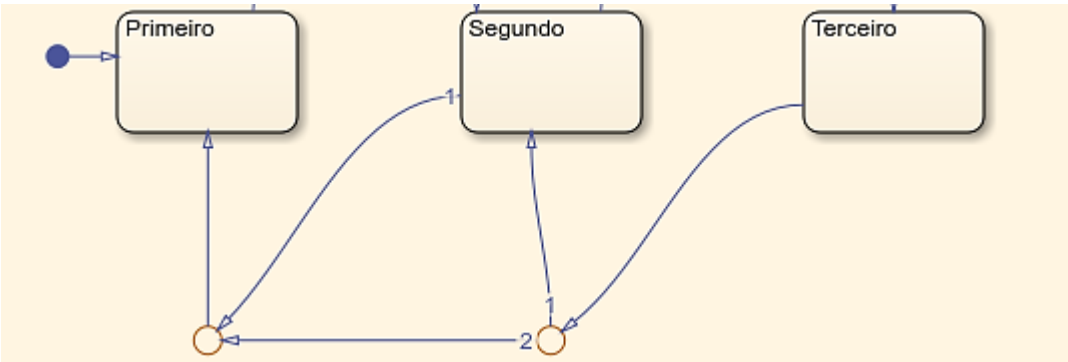
Figura 70 - Condições para subir de andar conectadas corretamente.



Fonte: Elaborado pelo autor, 2021.

Para adicionar as transições de descida do elevador basta copiar as transições de subida, porém inverta a ordem dos estados conforme a lógica matemática do sistema, de acordo a Figura 71. Neste caso, para descer do terceiro andar o elevador tem duas opções, para o primeiro ou segundo andar. E, caso o elevador esteja no segundo andar ele terá uma opção de andar para descer.

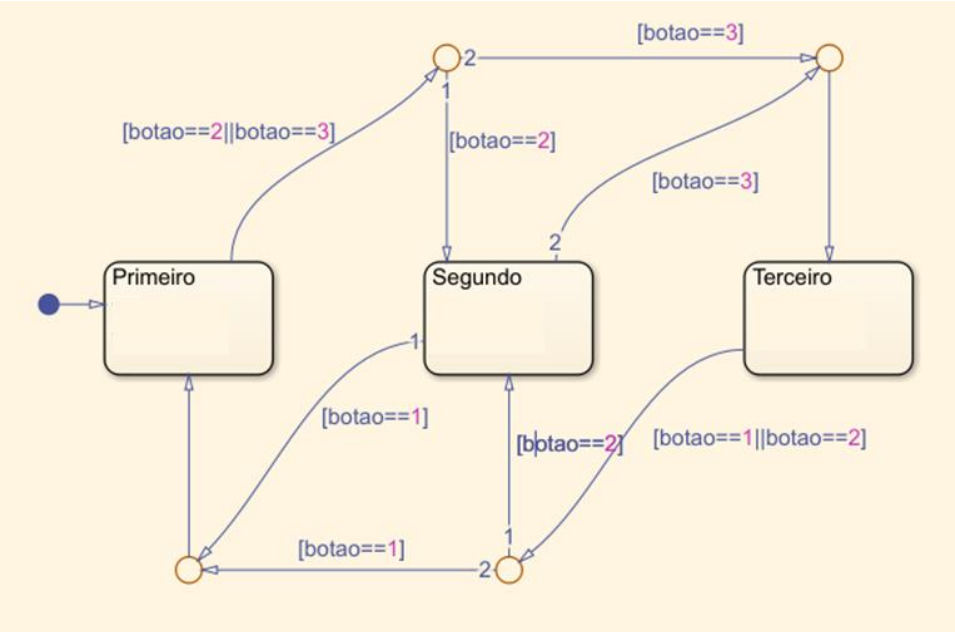
Figura 71 - Transições de descida do elevador.



Fonte: Elaborado pelo autor, 2021.

Após adicionar as *junctions* de descida deve-se colocar as condições que existem para que o elevador funcione corretamente, assim como apresenta a Figura 72 que contém todas as condições de subida e descida. Percebe-se que ambas as lógicas são análogas.

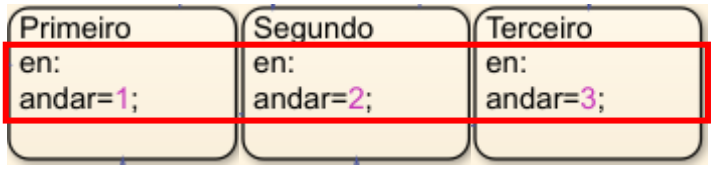
Figura 72 - Máquina de estados do elevador com todas as transições.



Fonte: Elaborado pelo autor, 2021.

Com as transições e suas condições inseridas é possível adicionar ações nos *states*, ou seja, quando a máquina estiver naquele estado irá executar determinada ação. Neste caso, os nomes dos estados são intuitivos e mostram que em cada andar a saída será o número do andar que o elevador estará. Isto foi feito para simplificar o problema, pois diversas ações poderiam ser apresentadas dentro deste estado. E, o resultado de adicionar ações de saída dentro de cada estado é exposta na Figura 73.

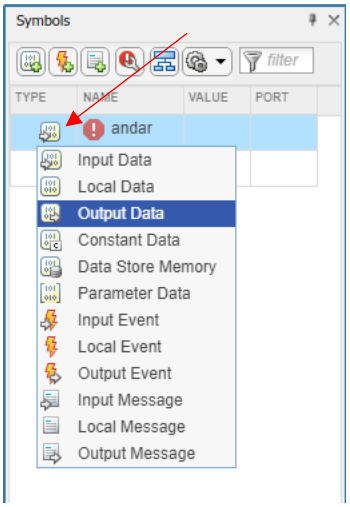
Figura 73 - Ações de saída da máquina.



Fonte: Elaborado pelo autor, 2021.

Configurado os estados e as transições, agora deve-se definir quais variáveis são de entrada e qual são de saída. Em versões mais atuais do *software* é capaz de entender o problema e configurar automaticamente, mas caso alguma configuração esteja errada basta dar um clique em *Type* na janela *Symbols* e configurar se a variável é de saída, entrada, local ou alguma outra opção avançada, isto conforme apresenta a Figura 74.

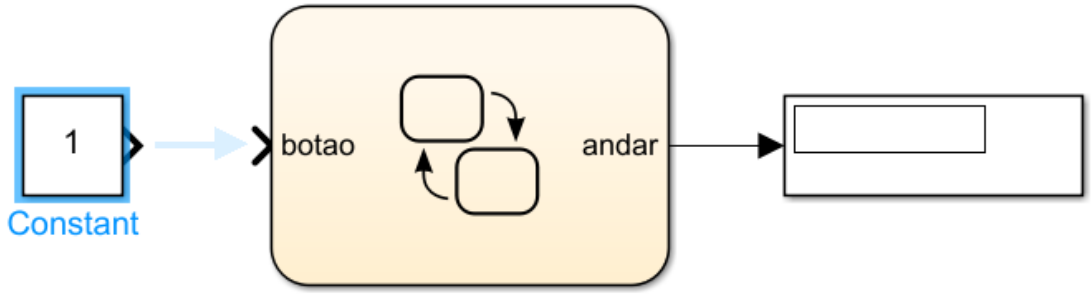
Figura 74 - Configurando as variáveis manualmente.



Fonte: Elaborado pelo autor, 2021.

Após configurado todos os parâmetros do *Stateflow* retorna-se a área de trabalho do *Simulink*. Nesta adicione um bloco *constant* e um bloco de *display*, segundo apresenta a Figura 75.

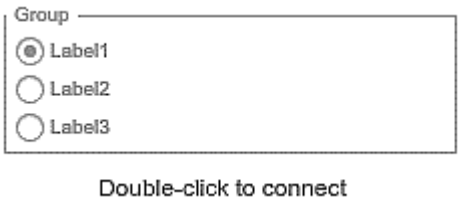
Figura 75 - Configura-se ambiente de trabalho do Simulink.



Fonte: Elaborado pelo autor, 2021.

Para controlar o elevador será adicionado um *radio button* no ambiente *Simulink* que está dentro painel de blocos *Library Browser* dentro da caixa das *dashboards*. Inserido no ambiente de trabalho ele irá se apresentar conforme a Figura 76.

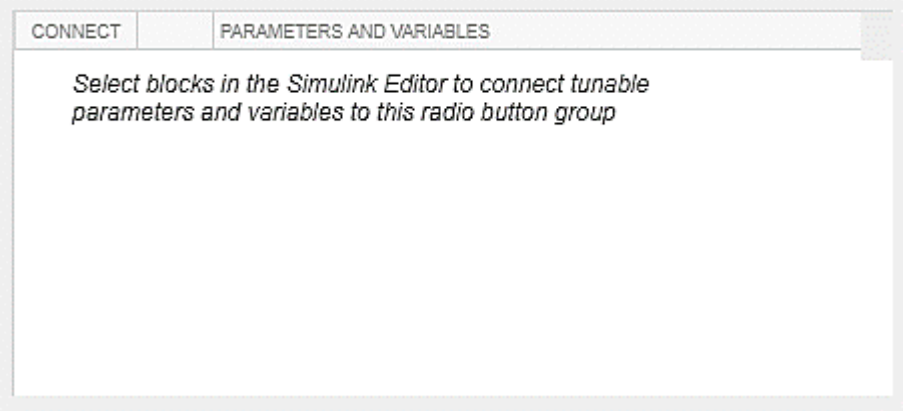
Figura 76 - Conecta-se a *constant* ao bloco *chart* e a saída do Stateflow ao display.



Fonte: Elaborado pelo autor, 2021.

Com um clique duplo no bloco pode-se acessar suas configurações para conectá-lo ao bloco *constant*. Todavia, primeiro é necessário criar uma variável com nome “botao” ambiente MATLAB, mais precisamente no *Command Window*. Seu valor pode ser um número inteiro preferencialmente de 1 a 3. Com a janela do *radio button* aberta, conforme mostra a Figura 77, clique no bloco *constant*.

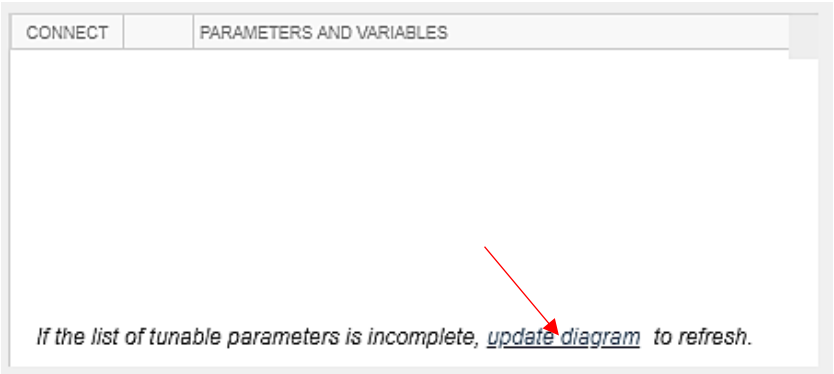
Figura 77 - Adiciona-se e configura-se o *Radio Button*.



Fonte: Elaborado pelo autor, 2021.

No entanto, irá aparecer a mensagem indicada na Figura 78 que sugere uma atualização do sistema. Isto tem como objetivo avaliar as novas variáveis adicionadas na *Command window*.

Figura 78 - Mensagem para editar *radio button* corretamente, no qual recarrega todo o sistema.



Fonte: Elaborado pelo autor, 2021.

Para tal ação, abra as configurações do bloco *constant* e realize a modificação do nome da variável no bloco para “botao”. Assim, ao configurar estes dados no sistema é possível associar o bloco *radio button* com a variável “botao”, consoante a Figura 79

Figura 79 - Configurando a variável no *radio button*.

Fonte: Elaborado pelo autor, 2021.

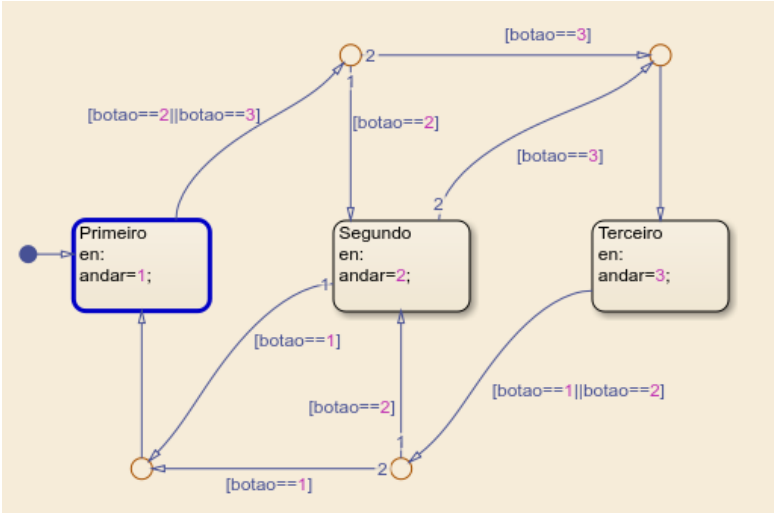
Assim, após executado essa configuração e ainda na janela do *radio button*, adicione os ícones de caixa de seleção e configure de acordo com o número da variável pretendida, isto é exemplificado pela Figura 80.

Figura 80 - Configurando o sistema de caixas de seleção.

Fonte: Elaborado pelo autor, 2021.

A partir desta configuração é possível compilar e rodar o programa, conforme apresenta a Figura 81. No qual é possível ver o trabalho em conjunto dos estados e das junções de transição quando o botão de troca de andar é pressionado.

Figura 81 - Máquina de estados compilando.



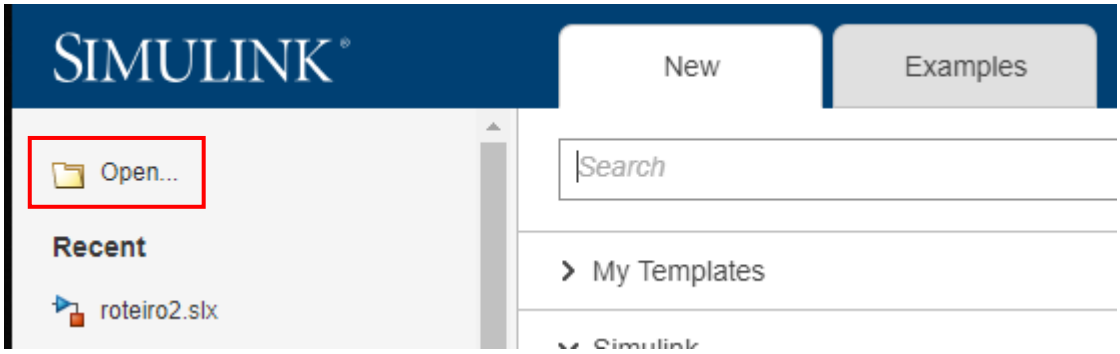
Fonte: Elaborado pelo autor, 2021.

Portanto, este exemplo teve como objetivo ser base para desenvolvimentos mais complexos de programas com *statecharts* tendo apresentado conceitos importantes que possibilitam o gerenciamento manual de qualquer transição complexa entre estados.

8.3. Procedimento experimental: Porta do elevador

Primeiramente é necessário criar um modelo *Simulink* no ambiente de trabalho do MATLAB. Ao abrir a nova janela de modelos de Simulink selecione o modelo *Stateflow* já existente realizado no tutorial anterior em que foi desenvolvido um elevador, conforme apresenta Figura 82.

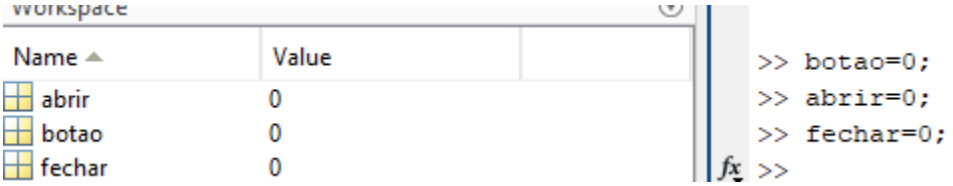
Figura 82 - Passo para abrir novo documento existente na página inicial do Simulink.



Fonte: Elaborado pelo autor, 2021.

Logo, o documento com as configurações feitas no exemplo anterior foi aberto e para testá-lo basta compilar e realizar testes manuais de funcionamento. Após confirmar o bom funcionamento do programa é necessário configurar as novas funcionalidades inseridas na *statechart*. Primeiro, deve-se adicionar duas novas constantes uma para o botão abrir e outra para o botão fechar, e isso é feito no *Command Window* e armazenado e exposto no *Workspace* do ambiente de trabalho do MATLAB, conforme apresenta a Figura 83.

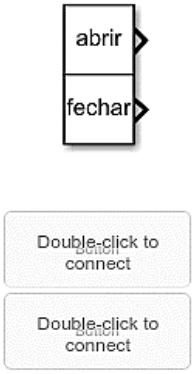
Figura 83 - Criando variáveis necessárias para desenvolvimento do tutorial.



Fonte: Elaborado pelo autor, 2021.

Criado as variáveis é preciso adicionar os blocos que irão receber os valores das mesmas, e isso é feito no ambiente *Simulink*, conforme apresenta a Figura 84. Adicione dois blocos de *constant* e dois blocos de *push buttons* (cada um será responsável por enviar um sinal para um bloco de *constant*). O primeiro bloco de constante irá receber valores da variável abrir, e em concordância com os exemplos anteriores edite seu valor de *input*, o mesmo é feito para o outro bloco que receberá os valores de fechar.

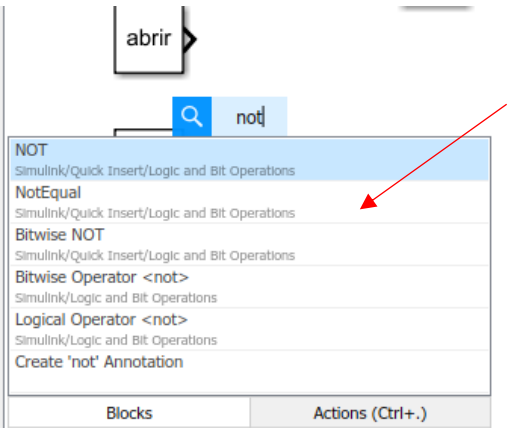
Figura 84 - Criando blocos essenciais de controle da Stateflow.



Fonte: Elaborado pelo autor, 2021.

Logo, adicionou-se os blocos e suas respectivas configurações. Porém, para evitar bugs no sistema no caso de os dois botões serem apertados ao mesmo tempo, é desenvolvida uma lógica para que somente um sinal seja recebido por vez (ou de abrir ou de fechar). Isto é feito primeiramente ao adicionar uma porta de negação conforme expõe a Figura 85. E também são conectados os blocos de constante aos *push buttons*, assim como executado anteriormente nos procedimentos de inserção de *dashboards*.

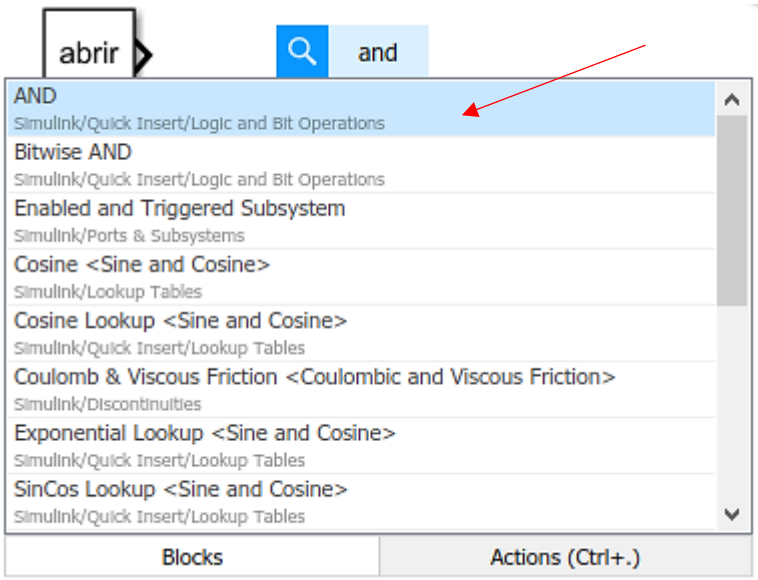
Figura 85 - Criando porta lógica NOR para evitar bugs no sistema.



Fonte: Elaborado pelo autor, 2021.

Posteriormente insere-se uma porta de AND para executar uma conjunção lógica entre o sinal de negação e o sinal de abrir a porta, apresentado na Figura 86. Isto é justificado pois se caso os dois sinais (abrir e fechar) forem iguais a 1 a ação de fechar a porta irá se sobressaltar sobre a ação de abrir.

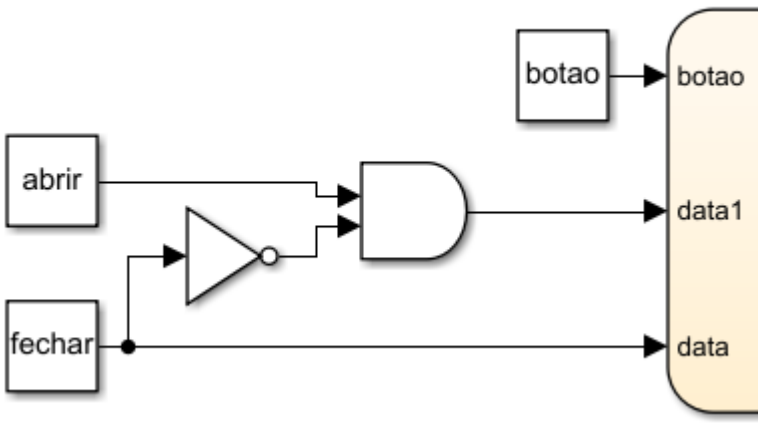
Figura 86 - Criando porta lógica AND complementar o controle lógico da porta NOR.



Fonte: Elaborado pelo autor, 2021.

Ao final das configurações e disposição dos blocos são realizadas as ligações de acordo com o que foi descrito, estas são expostas na Figura 87.

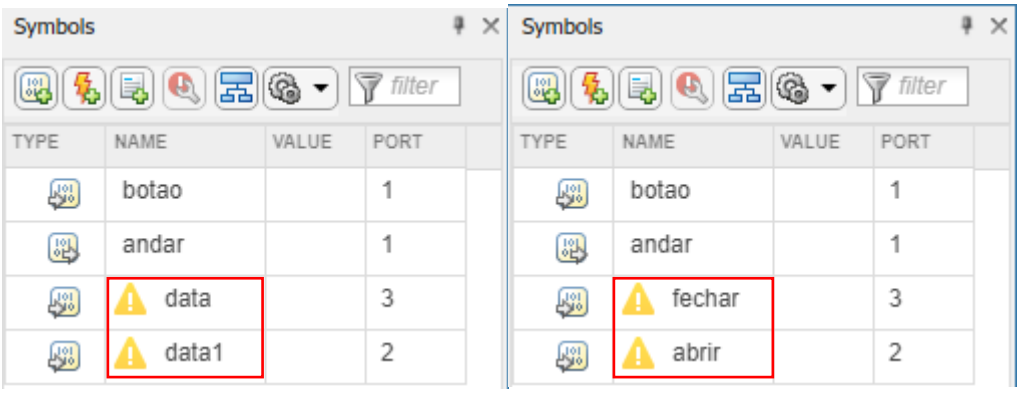
Figura 87 - Realizando as ligações necessárias com as portas lógicas.



Fonte: Elaborado pelo autor, 2021.

Após as variáveis estarem inseridas é necessário configurá-las no ambiente de trabalho do *Stateflow*. Isso é feito dentro da janela *symbols*, no qual alteram-se os nomes *data* e *data1* por fechar e abrir respectivamente. Estes passos são demonstrados na Figura 88.

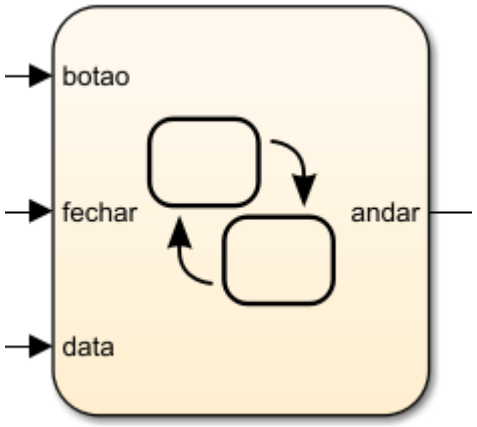
Figura 88 - (A) e (b) renomeando as variáveis de entrada do Stateflow.



Fonte: Elaborado pelo autor, 2021.

Após configurado a aparência do bloco *chart* irá se alterar, pois, os nomes das variáveis de entrada e saída foram alterados, como é demonstrado na Figura 89. Este é um fato de importante consulta no decorrer do desenvolvimento de uma *statechart* dentro do *Simulink*.

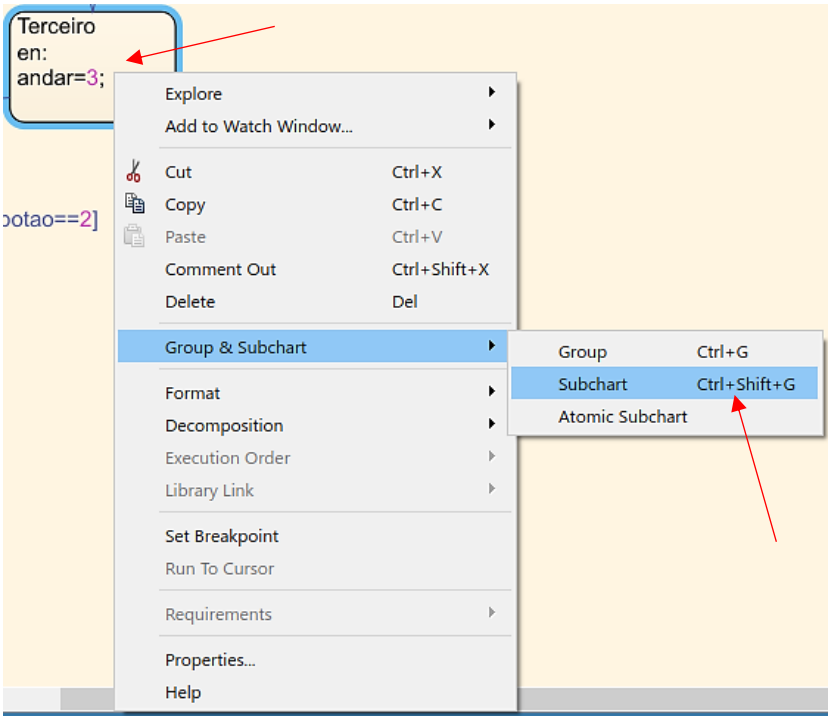
Figura 89 - Aparência do bloco Chart caso as variáveis forem corretamente ajustadas.



Fonte: Elaborado pelo autor, 2021.

Depois das configurações prévias será desenvolvido a máquina de estados. Para isto dê um duplo clique no bloco *chart* e após abrir o ambiente *Stateflow* clique com o botão direito dentro de um dos estados. Conforme apresenta a Figura 90, selecione *Group & Subchart* e depois *Subchart*.

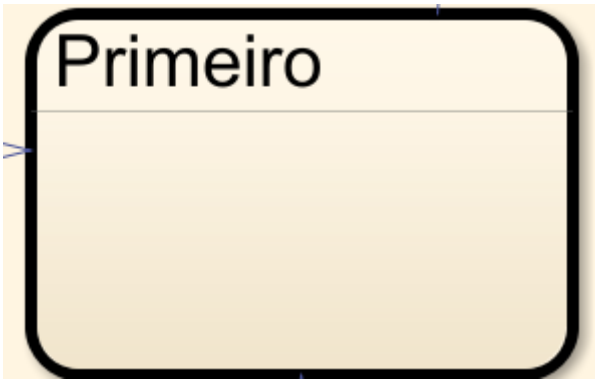
Figura 90 - Cria-se um *subchart* dentro de um char existente de algum *state* da máquina de estados.



Fonte: Elaborado pelo autor, 2021.

Para verificar se o passo foi executado corretamente, confira o *design* do bloco selecionado dentro do *Stateflow* para criar o *subchart* a primeira aparência deste será semelhante a apresentada na Figura 91.

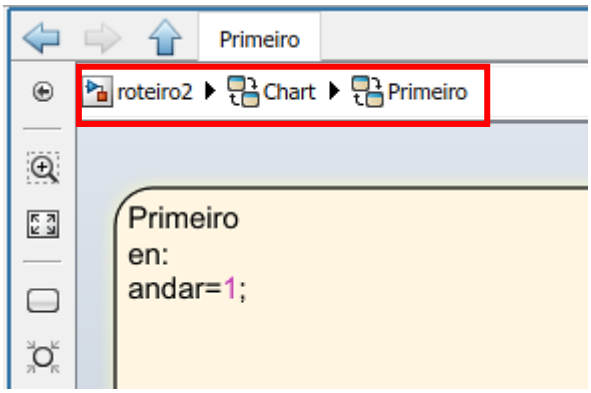
Figura 91 - *Layout* do *state* após a configuração do *subchart.*



Fonte: Elaborado pelo autor, 2021.

Ao dar um clique duplo sobre o novo bloco, entra-se em uma nova janela de ambiente de trabalho do *Stateflow*, que corresponde as configurações do subestado criado. Para visualizar qual janela o desenvolvedor se encontra basta observar a barra *object palllet* do *Simulink* que é apresentada na Figura 92.

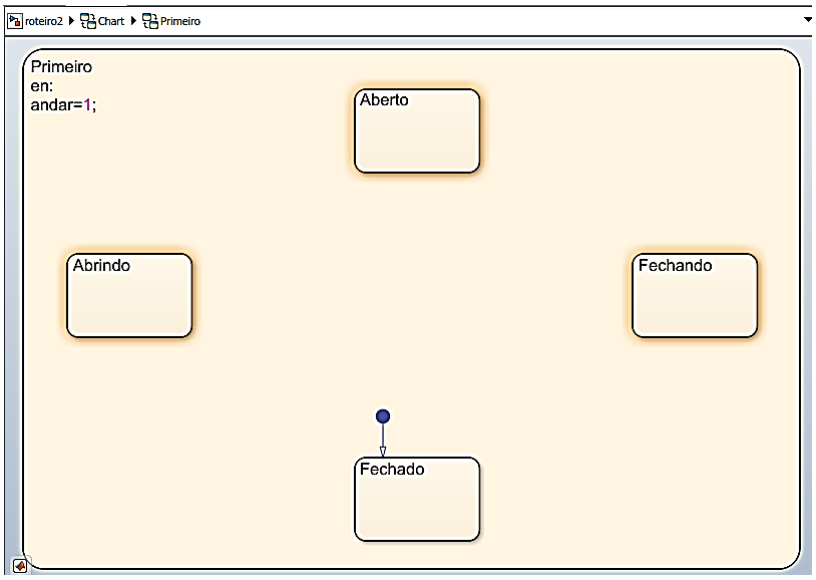
Figura 92 - Estrutura das janelas no *object pallet* do *Simulink*.



Fonte: Elaborado pelo autor, 2021.

O desenvolvimento dentro desta nova janela segue o fluxo convencional que foi adotado nos exemplos anteriores. Primeiramente é preciso inserir os estados conforme a descrição matemática, conforme apresenta a Figura 93. Através do qual, a porta se inicia no estado fechado. Isto justica-se, por exemplo, quando existe a troca de andares, caso o elevado suba do primeiro para o segundo andar, assim que este chega ao segundo sua porta deveria estar fechada durante todo o percurso.

Figura 93 - Criação de estados dentro do estado atual, além de ter uma condição de entrada.



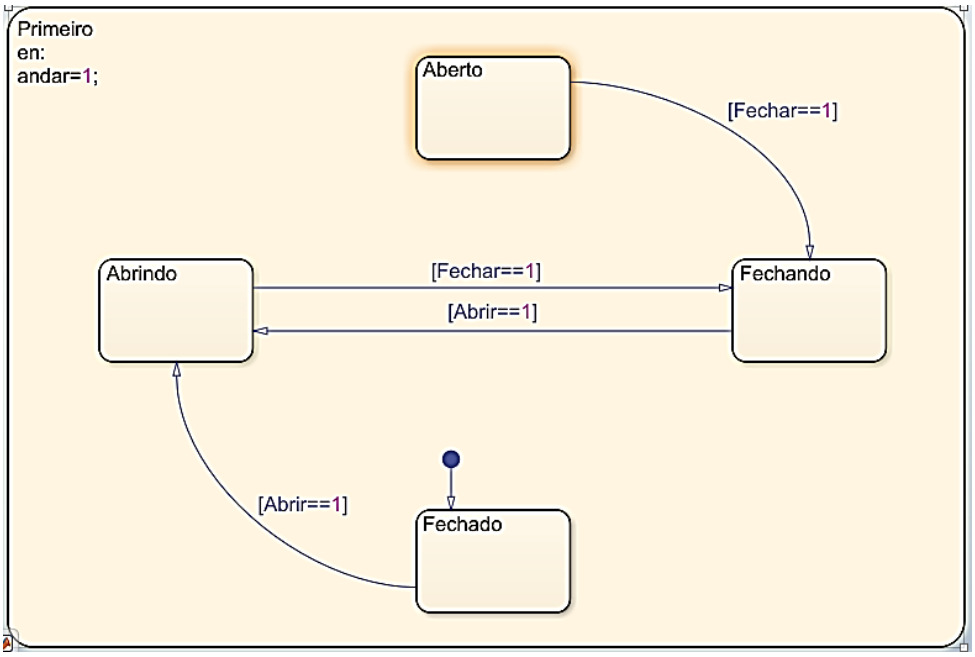
Fonte: Elaborado pelo autor, 2021.

Ao estarem inseridos os estados pode-se criar as transições referentes a abertura e fechamento das portas, isso é feito considerando os fatos:

a) A porta estando aberta e caso o botão fechar seja pressionado ela passa para o estado de fechando, até que essa ação seja concluída ou que a porta de fechando receba ordem para voltar a abrir;
b) A porta fechando continua fechando se receber um comando de fechar e começa a abrir se enquanto estiver fechando receber um comando para abrir. O estado abrindo funciona de maneira análoga inversa;
c) A porta no estado fechado retorna para si mesma comandos para fechar, já que irá permanecer fechada e vai para o estado abrindo caso seja pressionado.

Logo, cada estado contém sua peculiaridade e devem ser seguidos passos lógicos para que as transições ocorram.

Figura 94 - Transições quando os botões abrir e fechar forem acionados.



Fonte: Elaborado pelo autor, 2021.

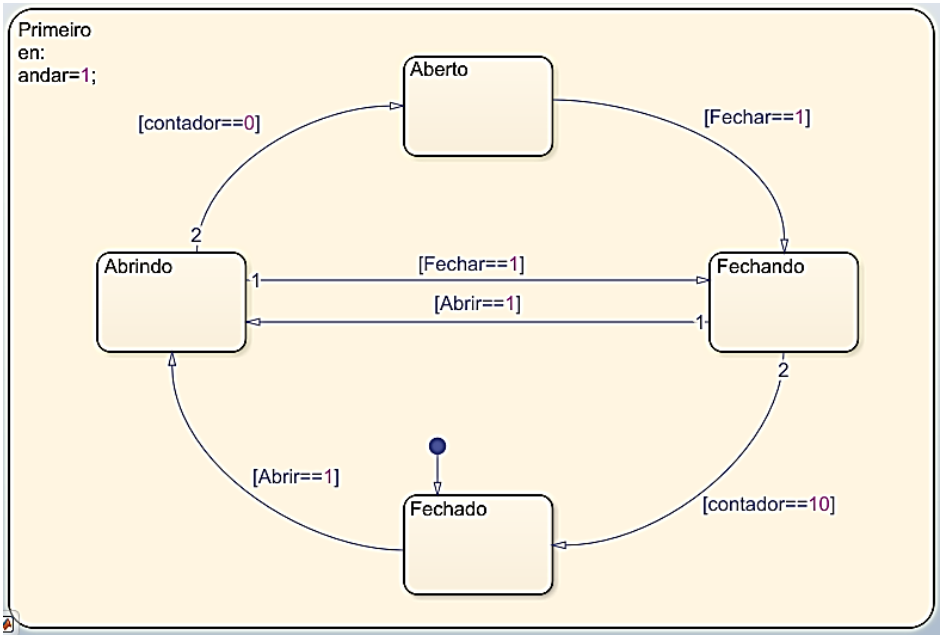
Para os outros estados será implementada uma lógica de contagem de posição, sendo discretizadas as posições do elevador. Isto é quando o elevador estiver na posição 0 ele estará com a porta aberta e quando o elevador estiver na posição 10, sua porta estará fechada. Tal fato, é apresentado na Figura 95, em que a Figura observa-se as portas abertas e terá valor 0, meio aberta e terá valor 5 e fechado com valor 10. É necessário fazer isto pois o *Stateflow* trabalha com valores discretos.

Figura 95 - Posição da porta do elevador e seu respectivo valor, (a) Aberto = 0; (b) Metade = 5 e (c) Fechado = 10.

Fonte: Elaborado pelo autor, 2021.

Fundamentada na Figura 95 é possível desenvolver a máquina de estados que corresponde a esta ideia. Vale ressaltar que, existem diversas formas para executar o controle da porta do elevador, desde as mais simples até as mais complexas. Conforme expõe a Figura 96 adiciona-se transições saindo de “abrindo” e “fechando” quando a porta chega ao nível 0 (no qual, estaria totalmente aberta) e ao 10 (estando totalmente fechada).

Figura 96 - Transições entre os estados envolvendo a lógica de contagem de posição da porta.

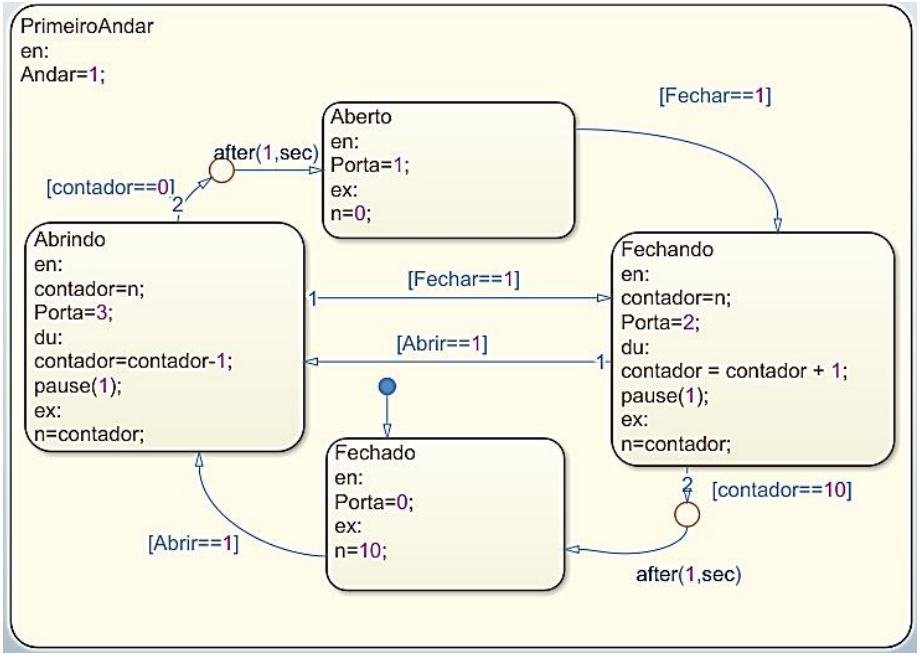


Fonte: Elaborado pelo autor, 2021.

Ao utilizar este contador, o sistema não terá nenhum sensor externo para verificar a posição da porta. Então considera-se que, a cada um segundo é dado um passo discreto de

abertura ou fechamento da porta. Também são configuradas situações de quando a porta estiver fechando pode-se retornar ela no meio do processo para a posição abrir que são as transições centrais conforme exposto a Figura 97. Para as transições entre os estados “fechando” para “fechado” e também de “abrindo” para “aberto” colocou-se um *junction* e outra transição com a função *after* para que na transição o *software* espere um segundo para realizá-la, o objetivo disto é realizar uma temporização do sistema e auxiliar na verificação da transição entre estados na visualização.

Figura 97 - Transição finais dos estados com as condições de entrada, durante e saída para a configuração da lógica que rege a abertura e fechamento da porta do elevador.

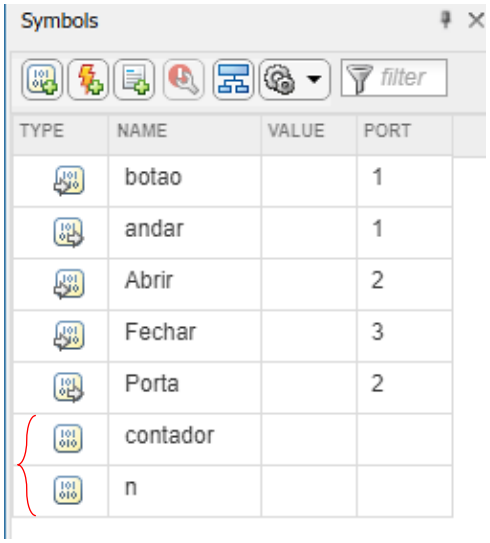


Fonte: Elaborado pelo autor, 2021.

Dentro dos estados da Figura 97 foram desenvolvidas três configurações: entrada (*entry*), durante (*during*) e saída (*exit*). Pelo qual, na entrada é definida a posição discreta atual da porta, se fechada é dada como 10 e se aberta como 0. E ainda na entrada existe uma variável de saída (“Porta”) para visualização no display ou em uma *dashboard* que indica o respectivo estado de acordo com a numeração, sendo estes: “aberto” saída como 1, “fechando” saída como 2, “fechado” saída 0 e por fim “abrindo” como saída 3. Enquanto o sistema estiver dentro dos estados “Abrindo” e “Fechando” será incrementada ou decrementada uma unidade de um contador de posição com um *delay* de 1 segundo até que a porta esteja ou completamente aberta ou fechada.

No final das configurações, é primordial verificar as variáveis de entrada e saída do programa, juntamente com a porta de conexão no ambiente Simulink. Conforme apresenta a Figura 98, as variáveis “n” e “contador” são variáveis apenas internas ao Stateflow e tem como objetivo auxiliar o funcionamento do sistema.

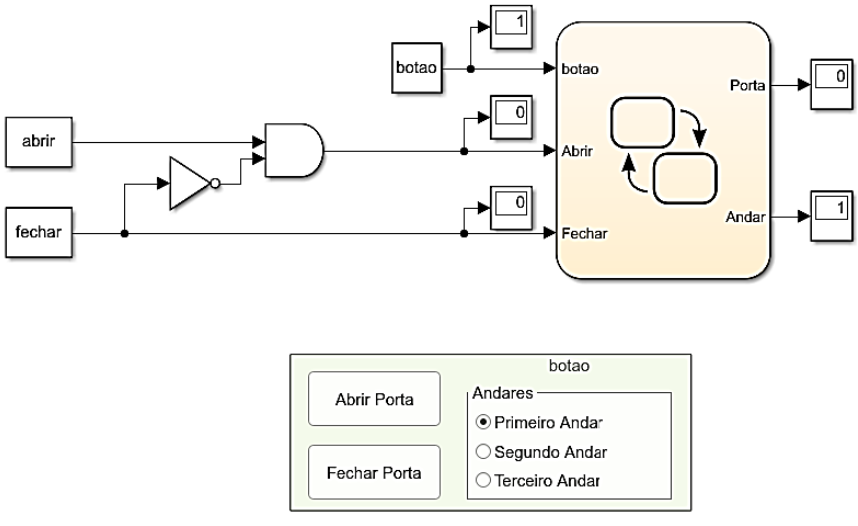
Figura 98 - Configurações das variáveis do programa.



Fonte: Elaborado pelo autor, 2021.

Portanto, é necessário criar um painel de configurações com *displays* e também botões para simulação do sistema. Fato realizado conforme apresentado em tutoriais anteriores anexando o valor da variável ao bloco *constant*. Logo a Figura 99 demonstra o sistema final para controle do sistema reativo. É válido salientar que, os valores nos *displays* irão expor informações dos respectivos estados e subestados do Stateflow.

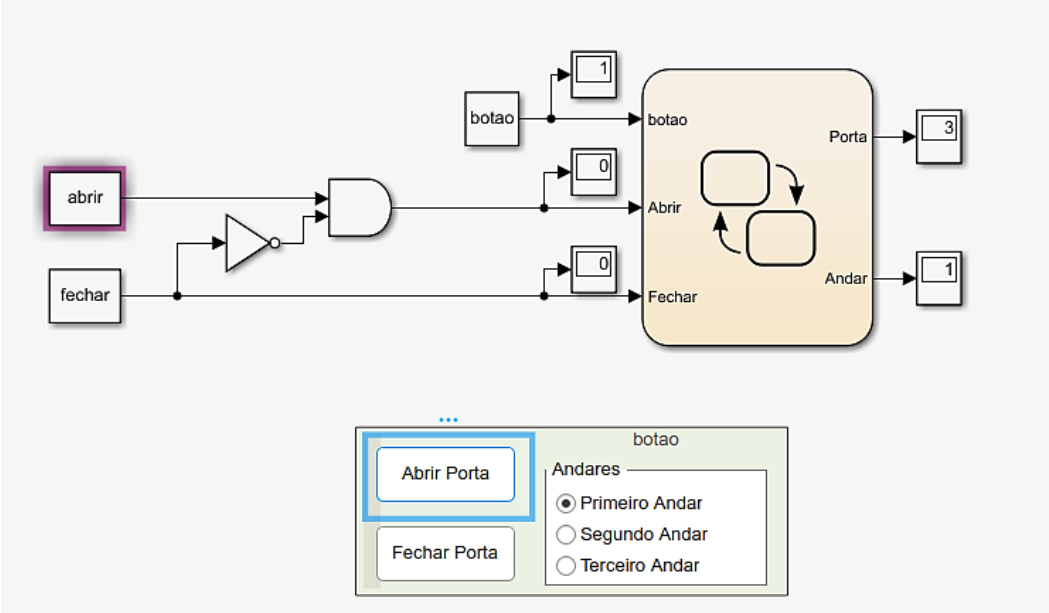
Figura 99 - Ambiente Simulink para o autómato do elevador.



Fonte: Elaborado pelo autor, 2021.

Com um clique no botão *run* o sistema é simulado conforme apresenta a Figura 100, no qual o usuário encontra-se no primeiro andar e pressionado o botão para abrir a porta. No *display* é possível ver o valor 3 para a variável porta indicando que a porta apresenta-se no estado “Abrindo”.

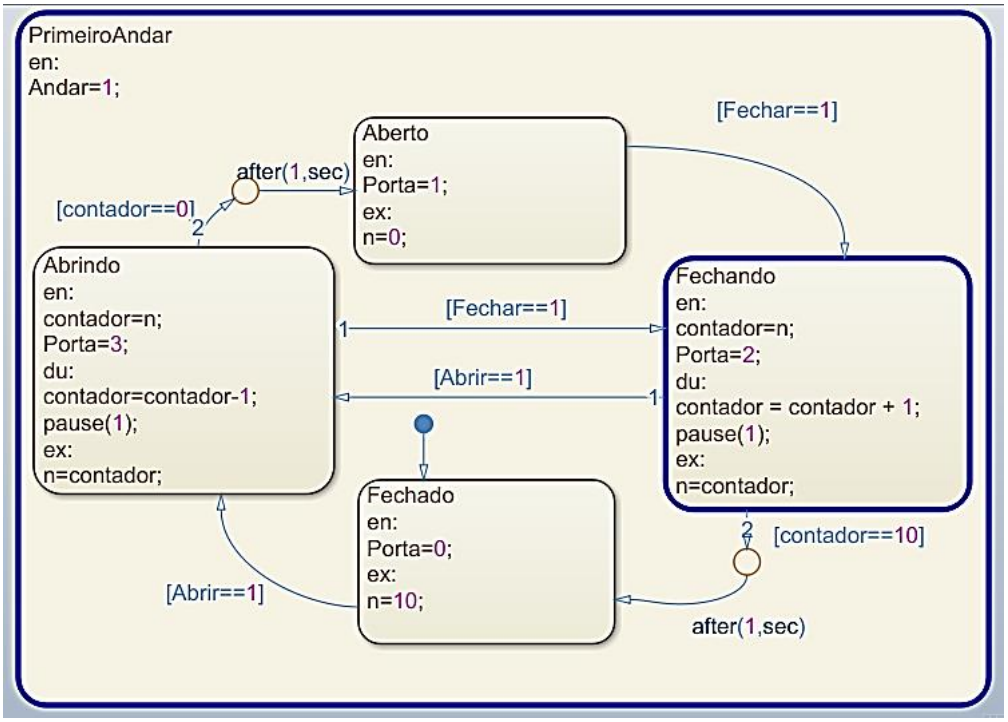
Figura 100 - Simulação do sistema completo da máquina de estados do elevador.



Fonte: Elaborado pelo autor, 2021.

Com o sistema em simulação é possível realizar a visualização dentro dos subestados do sistema, assim como é feito nos estados. Conforme apresenta a Figura 101 no qual foi pressionado o botão para fechar a porta e o sistema encontra-se no subestado “Fechando” dentro do estado primeiro andar.

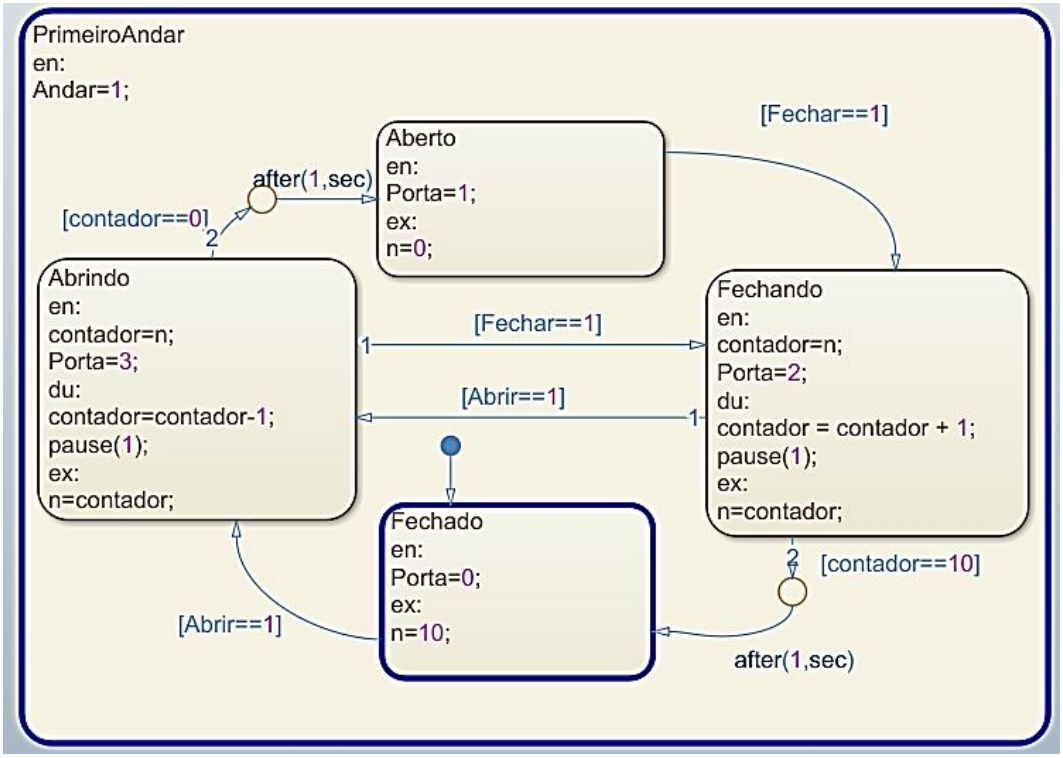
Figura 101 - Visualização dentro dos subestados.



Fonte: Elaborado pelo autor, 2021.

Para visualizar o efeito da função *after* é necessário que o sistema passe, por exemplo, do subestado “Fechando” para “Fechado”, segund mostra a Figura 102. O leitor pode testar esta funcionalidade com tempo maiores para que esta transição temporizada fique mais visível.

Figura 102 - Visualização dos subestados do *statechart*.



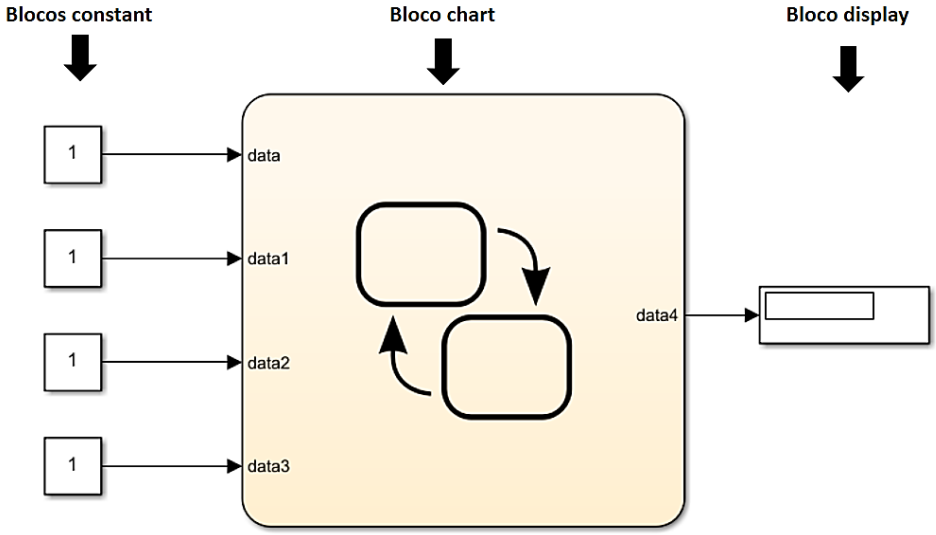
Fonte: Elaborado pelo autor, 2021.

Portanto, este sistema reativo teve como intuito demonstrar os níveis de hierarquia das *statecharts* através de um roteiro prático, fato que é amplamente utilizado seja para *clustering* ou refinamento. Pode-se visualizar o sistema apenas de maneira macro no qual observa-se apenas as mudanças de estados entre os andares, e também de maneira micro em que analisa-se as transições hierarquicamente menores como a transição da porta do elevador.

8.4. Procedimento experimental: Drone

Conforme metodologia apresentada nos exemplos anteriores, primeiramente é necessário em um modelo em branco do Simulink inserir os blocos base para funcionamento do programa. Este blocos são: o *constant* para alocação das variáveis; o bloco *chart*, que contém as funcionalidades do Stateflow e o bloco *display* que tem como objetivo visualizar o estado atual do sistema. A Figura 103 demonstra a configuração inicial do ambiente Simulink.

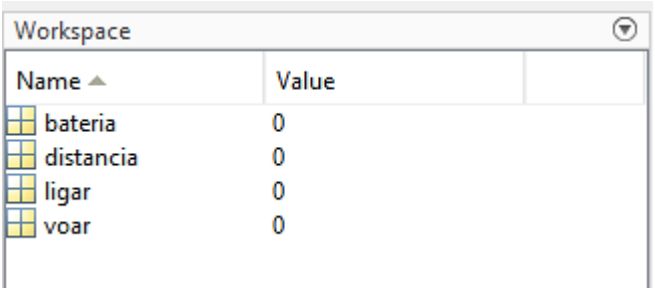
Figura 103 - Parâmetros iniciais do sistema.



Fonte: Elaborado pelo autor, 2021.

Para associar os blocos *constant* a um *dashboard* ou até mesmo a um microcontrolador externo é fundamental declarar as variáveis do programa no *Workspace* do MATLAB, conforme apresenta a Figura 104.

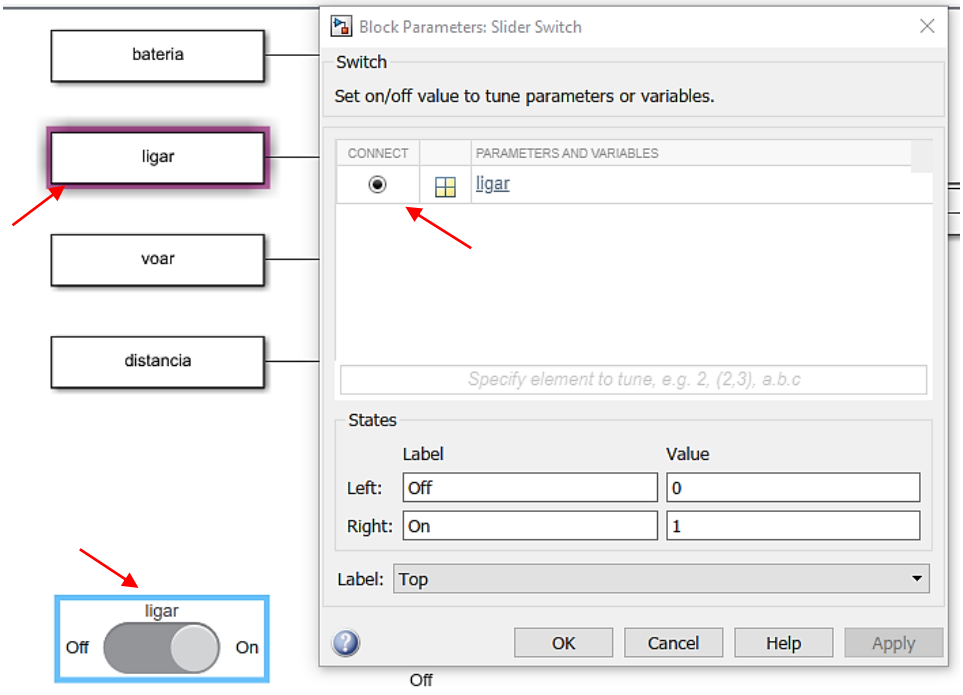
Figura 104 - Alocação das variáveis no *Workspace*.



Fonte: Elaborado pelo autor, 2021.

Após a alocação das variáveis dentro do sistema é possível criar um link com outros componentes presentes no Simulink ou mesmo no MATLAB. Em conformidade com o que é apresentado nos tutoriais anteriores, os *dashboards* são inseridos e relacionados com as respectivas variáveis do sistema. É válido ressaltar que, para as entradas analógicas, é importante a escolha de elementos que possuem a opção de variar a entrada continuamente em determinada faixa de valores. A Figura 105 demonstra o procedimento para conectar as *dashboards* ao modelo de simulação.

Figura 105 - Conexão das os *dashboards*. As setas representam pontos chaves que devem aparecer ao ser conectada a *dashboard*.



Fonte: Elaborado pelo autor, 2021.

Configuradas as entradas e saídas do sistema conforme a descrição é necessário, com um duplo clique no bloco *chart*, inserir os estados mencionados. Este passo é representado a Figura 106.

Figura 106 - Definindo os estados do sistema.

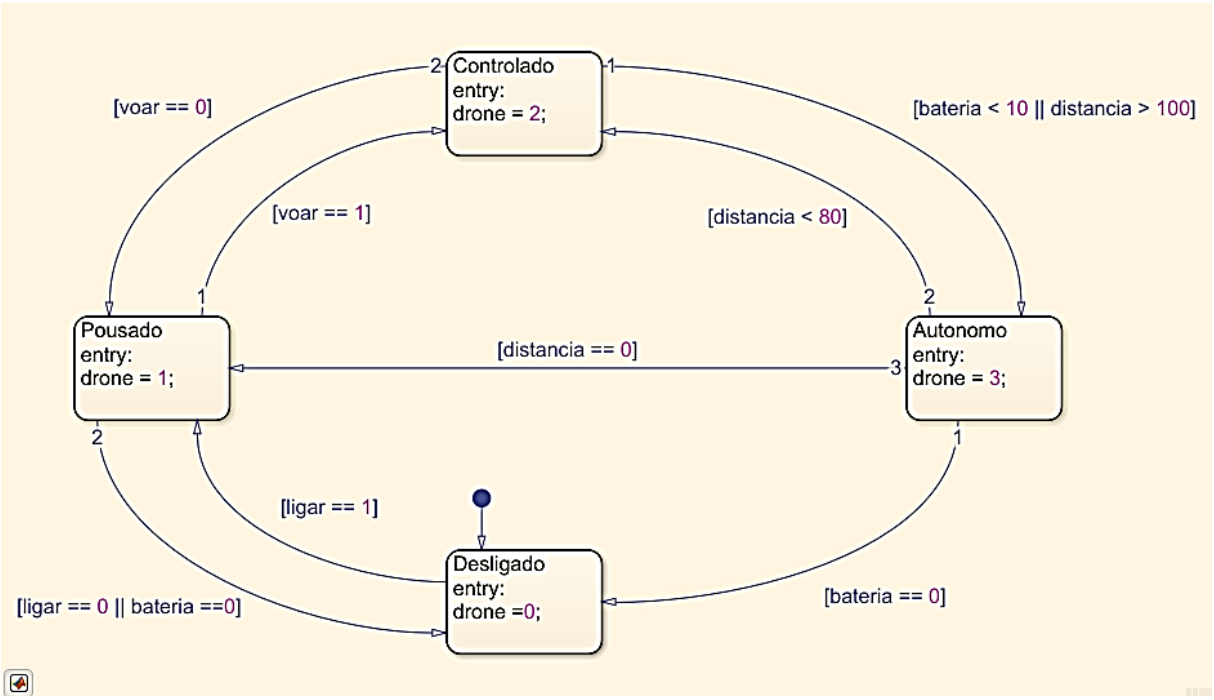
Fonte: Elaborado pelo autor, 2021.

Após inseridos os estados no sistema é indispensável inserir as respectivas transições, no qual seguem a lógica pré-estabelecida com o intuito de trabalhar com valores não booleanos. O sistema pode ser descrito como:

a) Do estado desligado para pousado é necessário que a chave ligar esteja em nível lógico alto;
b) Do estado pousado para desligado a transição é efetuada quando a chave passa para o nível lógico baixo;
c) Do estado pousado para vôo controlado a transição a ser efetuada é a chave voar estar em nível lógico alto;
d) Do estado de vôo controlado para pousado é necessário a chave voar estar em nível lógico baixo;
e) Do estado vôo controlado para vôo autônomo é transitado quando a distância excede 100 metros ou a bateria indica um valor percentual menor do que 10%;
f) Do estado autônomo para vôo controlado é preciso que a distância entre o drone e o controlador seja menor que 80 metros;
g) Do estado autônomo para pousado é fundamental que a distância do drone para o ponto de partida (origem) seja nula;
h) Do estado autônomo para desligado é necessário que a bateria alcance valor nulo, descarregando o drone e o mesmo sendo desligado.

A Figura 107 demonstra o resultado das transições efetuadas no sistema. Como se trata de um exemplo educacional, o número de transições efetuadas foi minimizado do que uma situação para desenvolvimento de um dispositivo aplicado a um ambiente real.

Figura 107 - Definindo as transições do sistema.

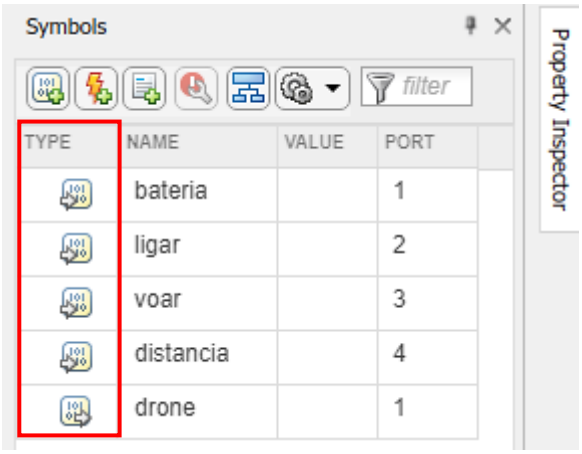


Fonte: Elaborado pelo autor, 2021.

Conforme apresenta a Figura 107 as variáveis não booleanas foram trabalhadas com simbologias lógicas de comparação, no qual, as transições verificam se a condição imposta é verdadeira ou falsa. Fato que independe se a variável é inteira ou ponto flutuante.

Para configurar o sistema para simulação é preciso, conforme indica a Figura 108, a configuração na aba *Symbols* das variáveis com entrada e saída.

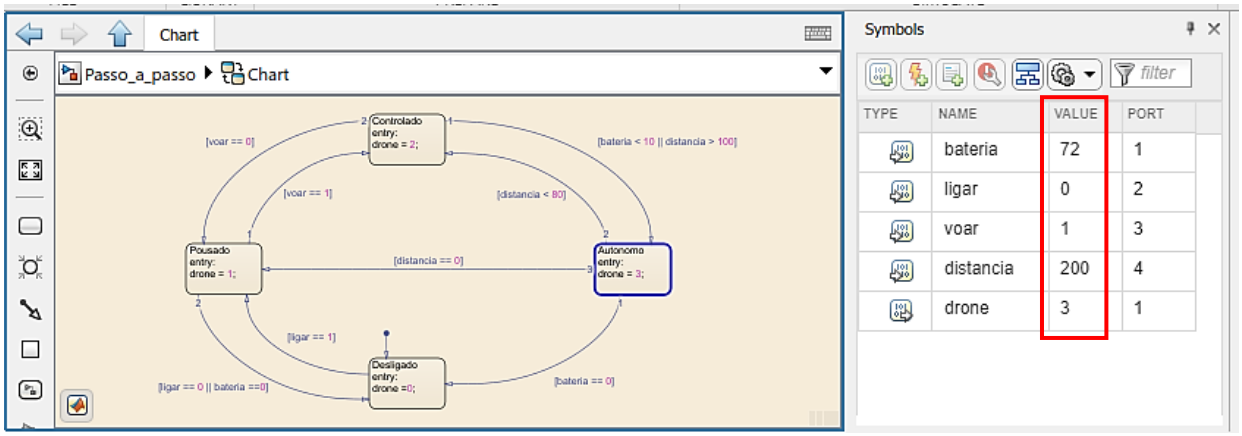
Figura 108 - Configurações das variáveis de entrada e saída.



Fonte: Elaborado pelo autor, 2021.

Após configuradas as variáveis de entrada e saída é possível executar a simulação do sistema, conforme apresenta a Figura 109, pelo qual dentro da aba Symbols é possível visualizar o valor que a variável está atribuída.

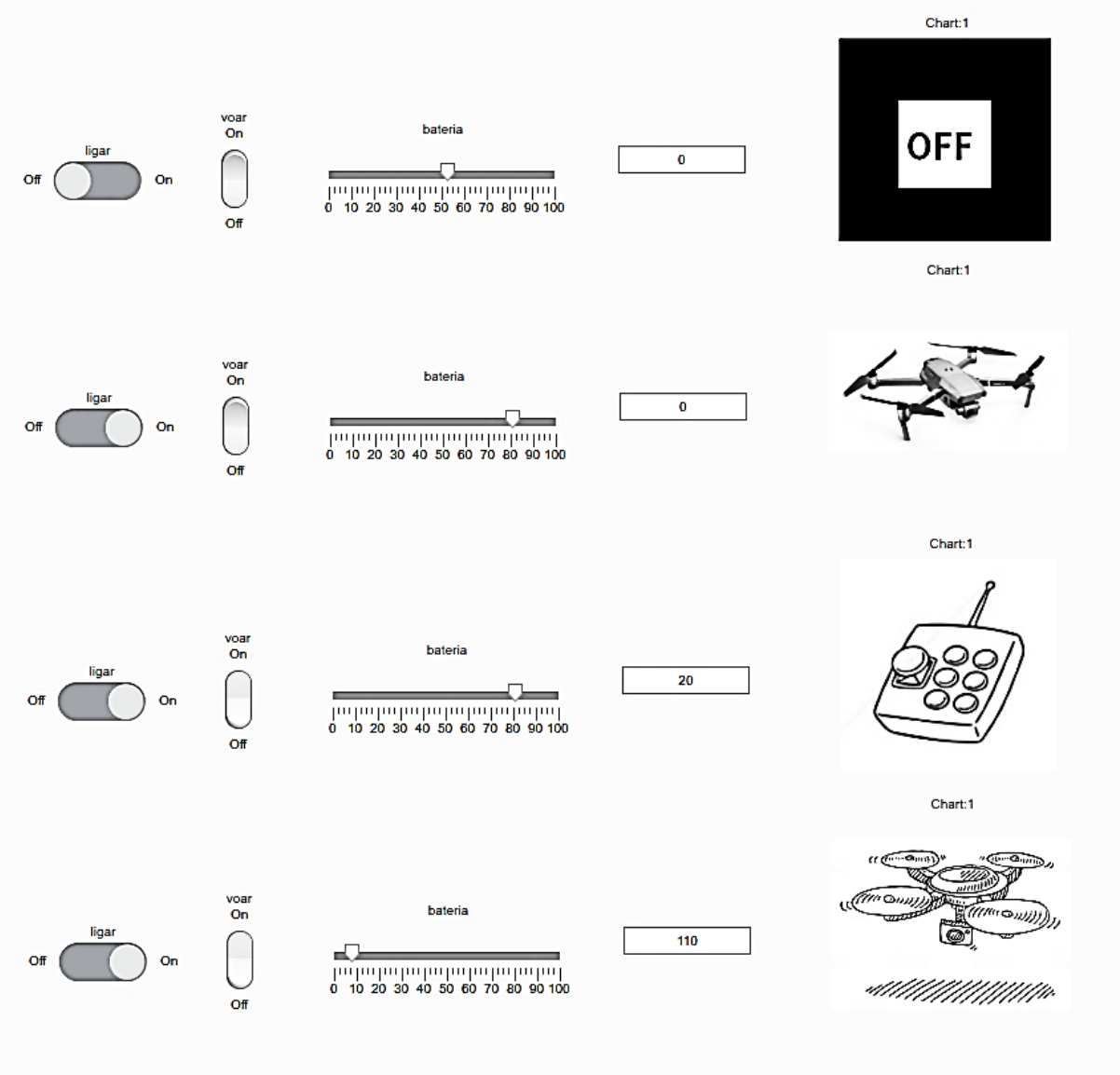
Figura 109 - Simulação do exemplo educacional do sistema embarcado para drone.



Fonte: Elaborado pelo autor, 2021.

No ambiente Simulink pode-se realizar a visualização e controle por meio das *dashboards*. Como o sistema possui variáveis não booleanas, a inserção do bloco *slider* e da caixa de preenchimento auxiliam na simulação. A Figura 110 demonstra configurações da simulação e o estado atual por meio do bloco *chart* de forma a ilustrar as etapas de simulação.

Figura 110 - Simulação da máquina de estados: Drone.



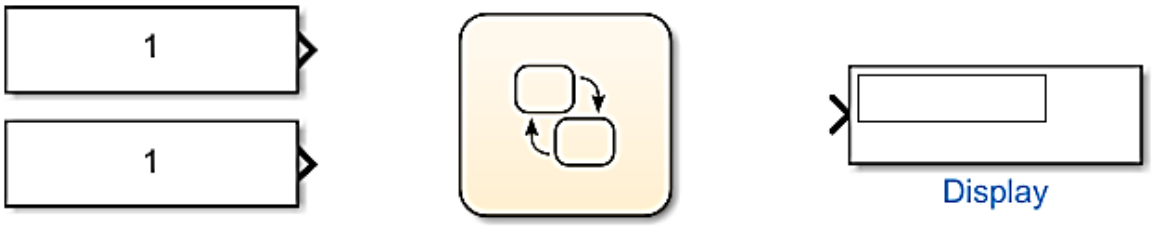
Fonte: Elaborado pelo autor, 2021.

Neste exemplo foi possível verificar o trabalho com variáveis não booleanas e sua interação com os estados e transições do sistema, válido enfatizar que o conteúdo do tutoriais anteriores é primordial para a boa execução desta *statechart*.

8.5. Procedimento experimental: Máquina de lavar

Conforme os exemplos anteriores, primeiramente é necessário em um modelo em branco do Simulink, inserir os blocos base para funcionamento do programa. Este blocos são: o *constant* para alocação das variáveis de entrada, bloco do "*chart*" para o Stateflow e bloco "*display*" para visualização das saídas.

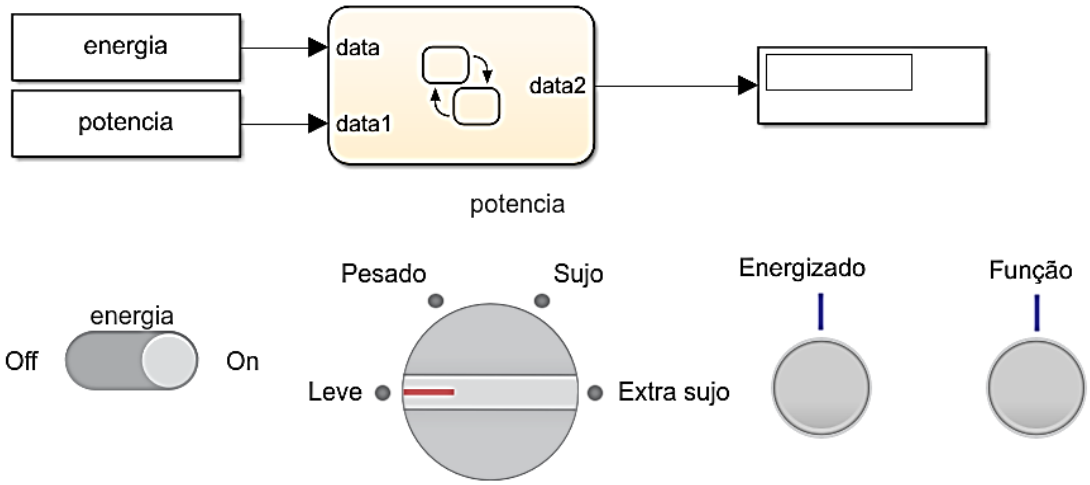
Figura 111 - Componentes Simulink para a máquina de estados drone.



Fonte: Elaborado pelo autor, 2021.

São inseridos no bloco *constant* variáveis nomeadas como “energia” e “potência”, que se referem, respectivamente, como iniciar o sistema (liga/desliga) e modo de funcionamento da lavagem com quatro *status* diferentes (leve, pesado, sujo e extra sujo). Também são inseridos dois blocos para visualização da simulação do sistema. A Figura 112 demonstra o ambiente Simulink após as modificações efetuadas.

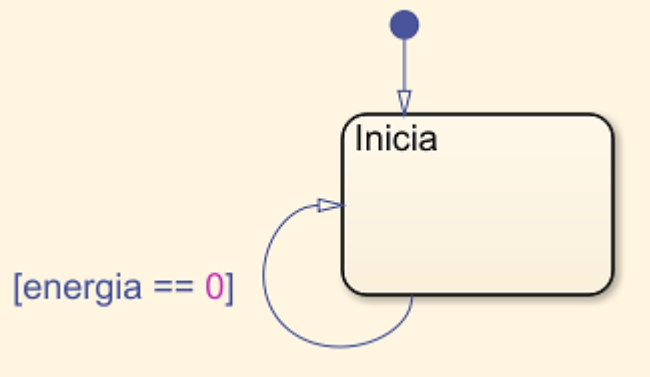
Figura 112 - Componentes *dashboards* para controle do sistema.



Fonte: Elaborado pelo autor, 2021.

No ambiente de trabalho do Stateflow insira um bloco com a condição inicial do sistema e uma transição retornando-o para o mesmo sempre que a entrada “energia” for nível lógico baixo. A Figura 113 indica o procedimento descrito, e para executá-lo é necessário seguir os passos citados nos tutoriais precedentes.

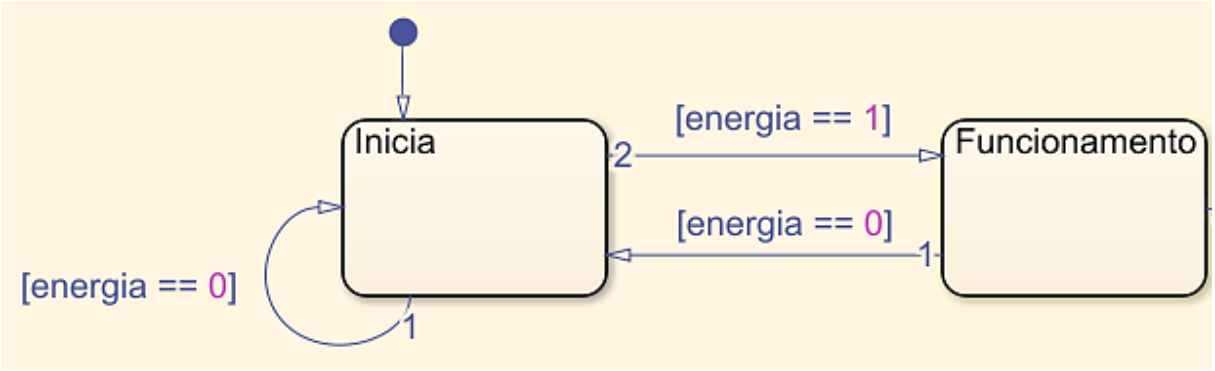
Figura 113 - Estado inicial a ser inserido.



Fonte: Elaborado pelo autor, 2021.

O sistema terá dois estados, um para iniciar e outro para funcionamento. As transições entre os mesmos são inseridas conforme apresenta a Figura 114, no qual são dependentes do valor lógico da entrada da variável “energia”.

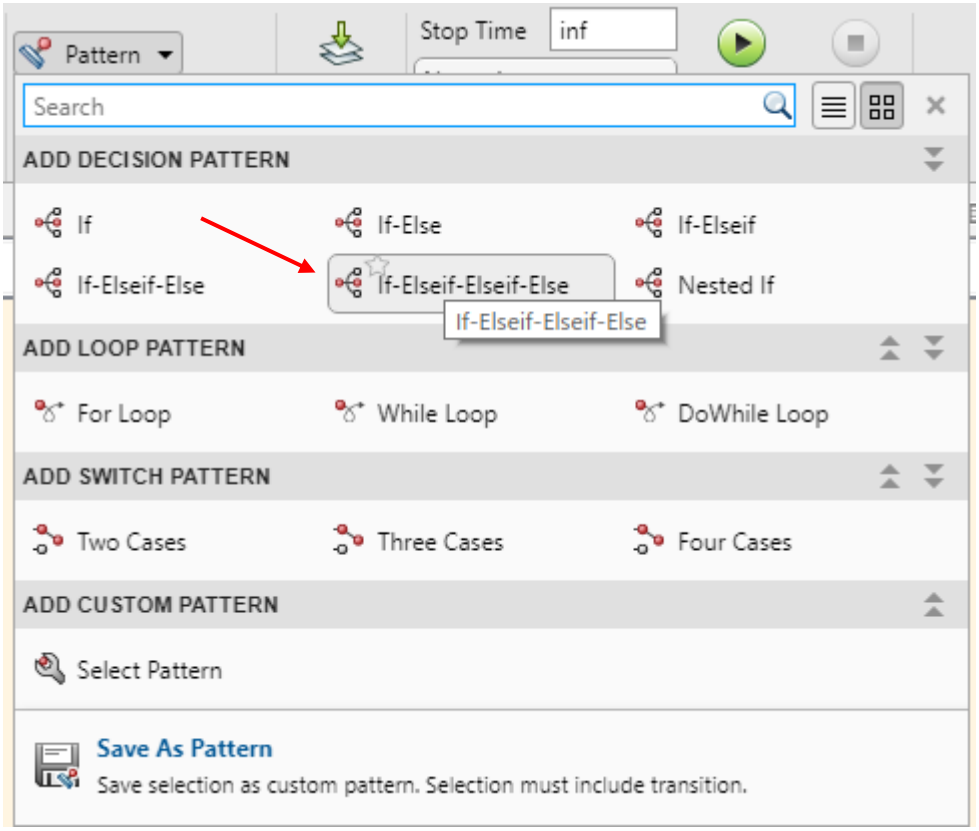
Figura 114 - Estados possíveis do sistema.



Fonte: Elaborado pelo autor, 2021.

De acordo com a descrição da máquina de estados existem diferentes entradas para o modo de funcionamento. Ou seja, para entrada da variável “potencia” igual a 1 se refere ao modo de lavagem leve, para 2 no modo pesado, para 3 no modo sujo e para 4 no modo extra sujo. O sistema realizará diferentes transições para cada entrada, e para isto é necessário implementar um padrão de transições com uma estrutura de decisão. A Figura 115 demonstra o menu referente a esta funcionalidade do Stateflow e a indicação do padrão a ser utilizado. Válido salientar que existem padrões pré-definidos, porém os mesmos podem ser desenvolvidos manualmente pelo usuário conforme sua aplicação.

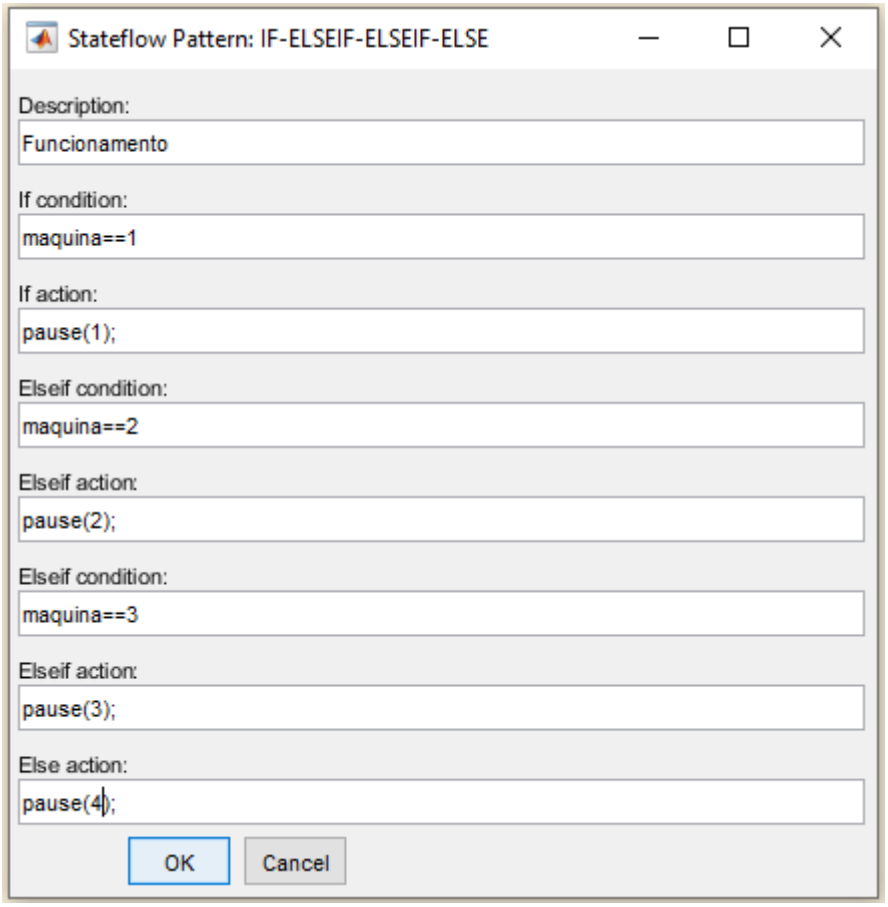
Figura 115 - Estruturas condicionais com transições com destaque para estrutura a ser inserida.



Fonte: Elaborado pelo autor, 2021.

Com um clique sobre o padrão de transições é possível inserir configurações na janela *pop-up* que será aberta. Como o sistema a ser modelado refere-se a uma máquina de Mealy, as ações são definidas nas transições e neste caso a ação será uma pausa (temporização) diferente para cada tipo de trabalho da máquina de lavar, quanto maior a potência para a lavagem maior o tempo. A Figura 116 demonstra a janela que é mostrada ao usuário quando selecionado o padrão de transição. Os campos são preenchidos conforme os tutoriais anteriores no campo de condição e para o campo de ação são preenchidos da mesma maneira que eram inseridas dentro dos estados nos exemplos anteriores, porém sem a necessidade de inserir o momento que a ação irá atuar seja ela de entrada, duração ou saída (*entry*, *during* ou *exit*).

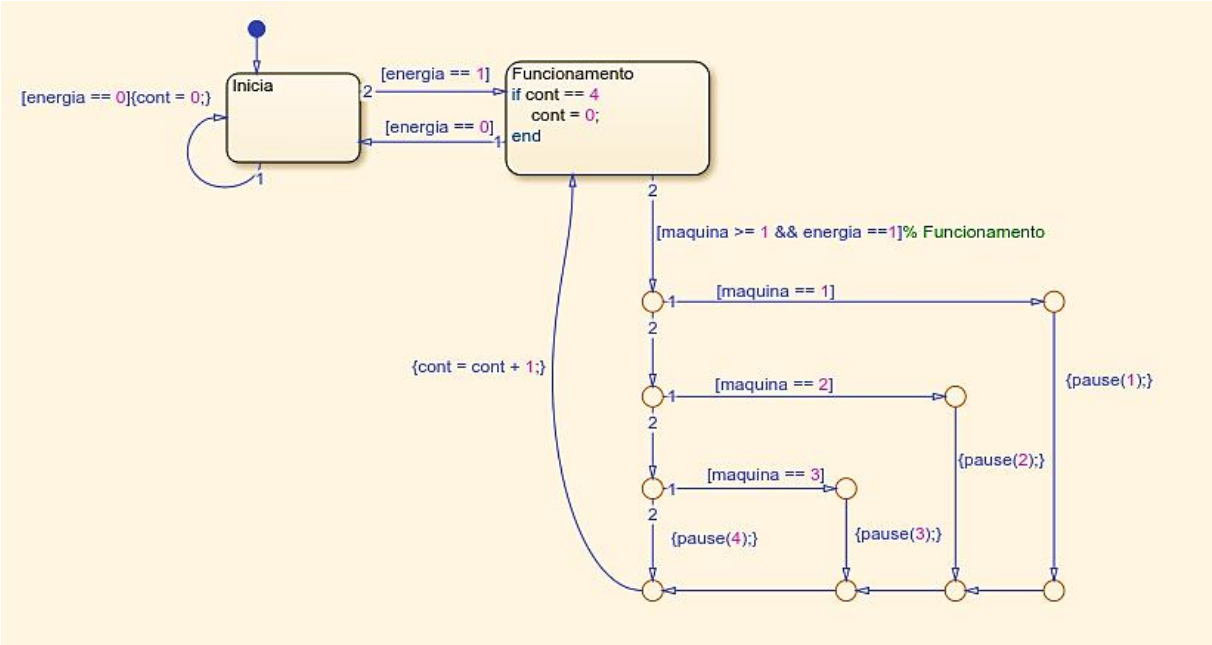
Figura 116 - Padrão condicional do Stateflow com condições e ações do autómato: máquina de lavar.



Fonte: Elaborado pelo autor, 2021.

Após preenchido o padrão, pode ser adicionado ao ambiente de trabalho do Stateflow arrastando-o para o estado de saída indicado. É adicionado um contador no qual se o sistema realizar as ações quatro vezes o mesmo será reiniciado. A Figura 117 demonstra o sistema após serem adicionadas as transições e ações para funcionamento.

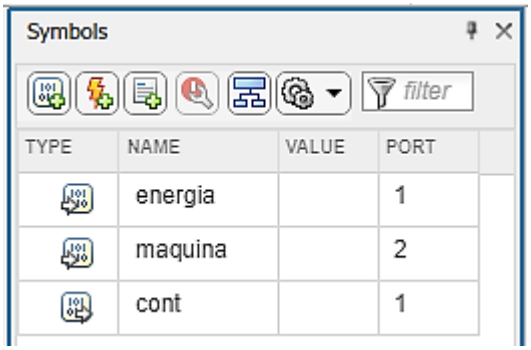
Figura 117 - Estados com transições e condições adicionadas.



Fonte: Elaborado pelo autor, 2021.

Com as configurações executadas é necessário no menu *Symbols* editar as variáveis como sendo de entrada ou saída para sua comunicação com o ambiente Simulink, a Figura 118 indica o procedimento.

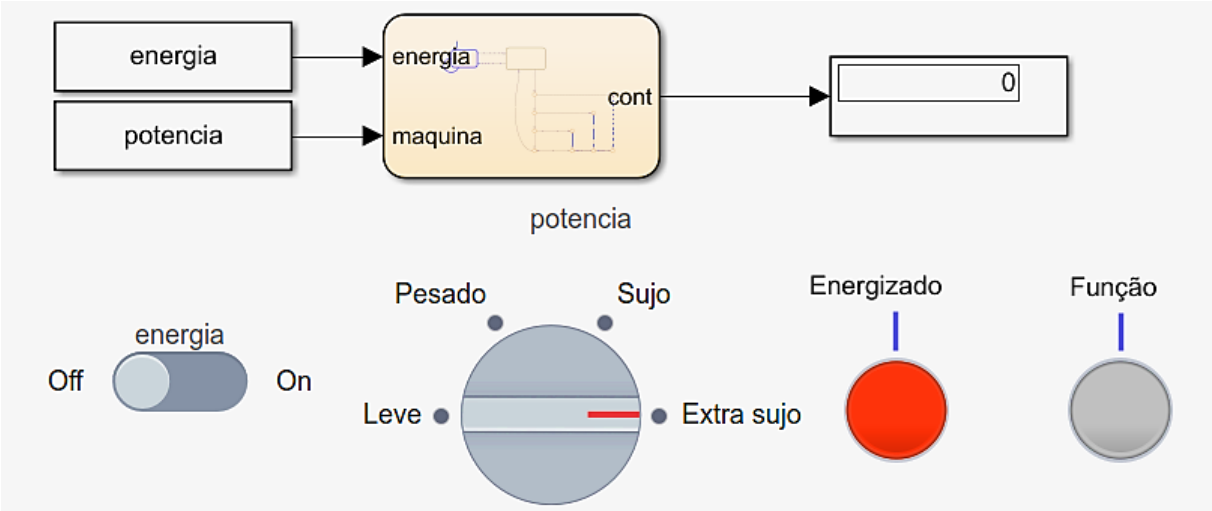
Figura 118 - Configuração das variáveis do sistema.



Fonte: Elaborado pelo autor, 2021.

Retornando ao ambiente Simulink, é possível realizar a simulação do sistema, no qual, observa-se também dentro do bloco *chart* os estados atuais e transições realçados em azul para que o usuário possa acompanhar. A Figura 119 expõe as configurações finais editadas no ambiente Simulink (conforme apresentadas nos tutoriais anteriores) e a simulação do sistema com a máquina de lavar desligada.

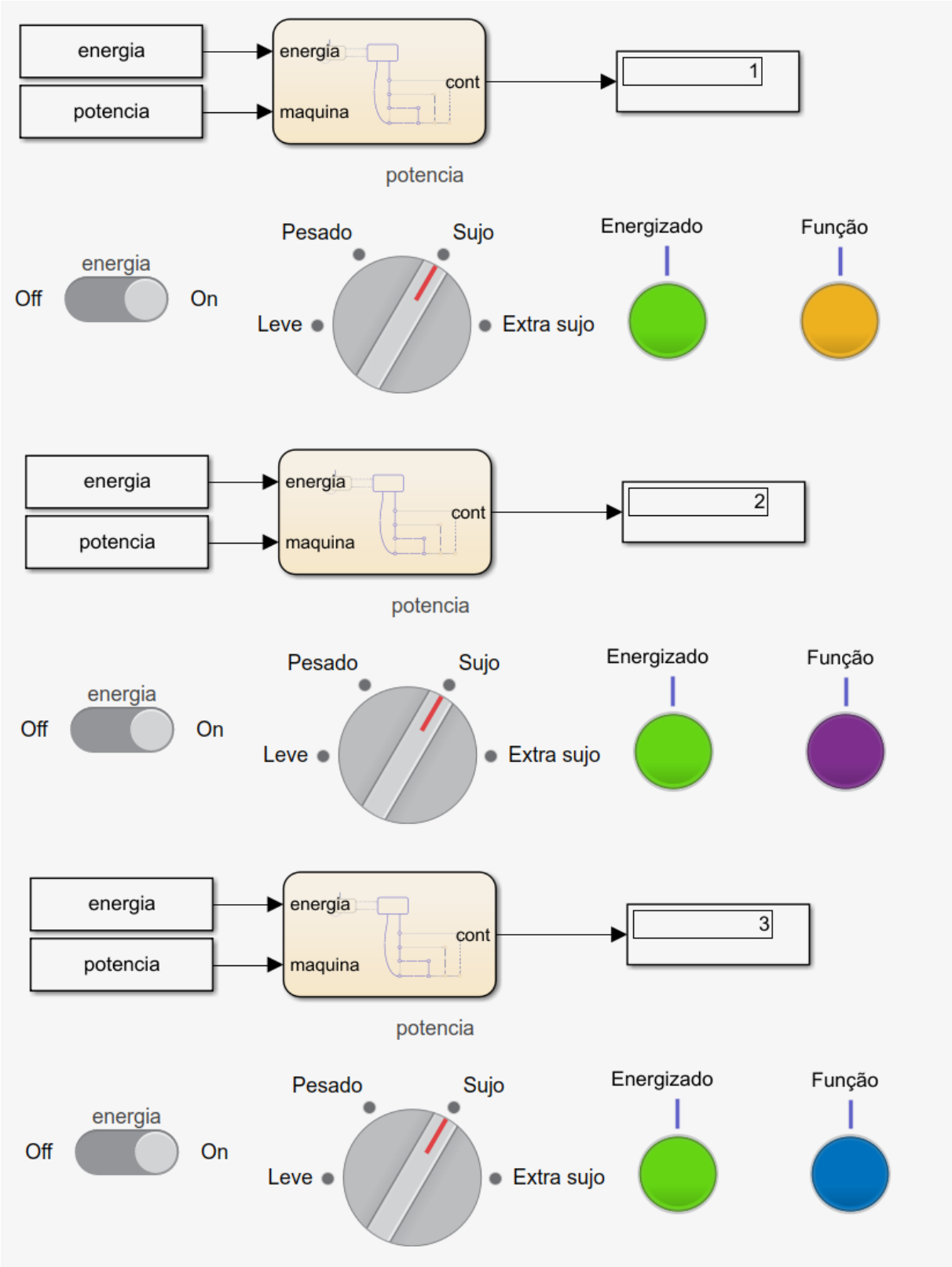
Figura 119 - Simulação do sistema para estado desenergizado.



Fonte: Elaborado pelo autor, 2021.

Ao selecionar um modo de lavagem pela *dashboard* e configurado o sistema para o modo ligado, pode-se perceber o funcionamento da máquina de Mealy conforme indica a Figura 120.

Figura 120 - Simulação do sistema com saídas indicativas.



Fonte: Elaborado pelo autor, 2021.

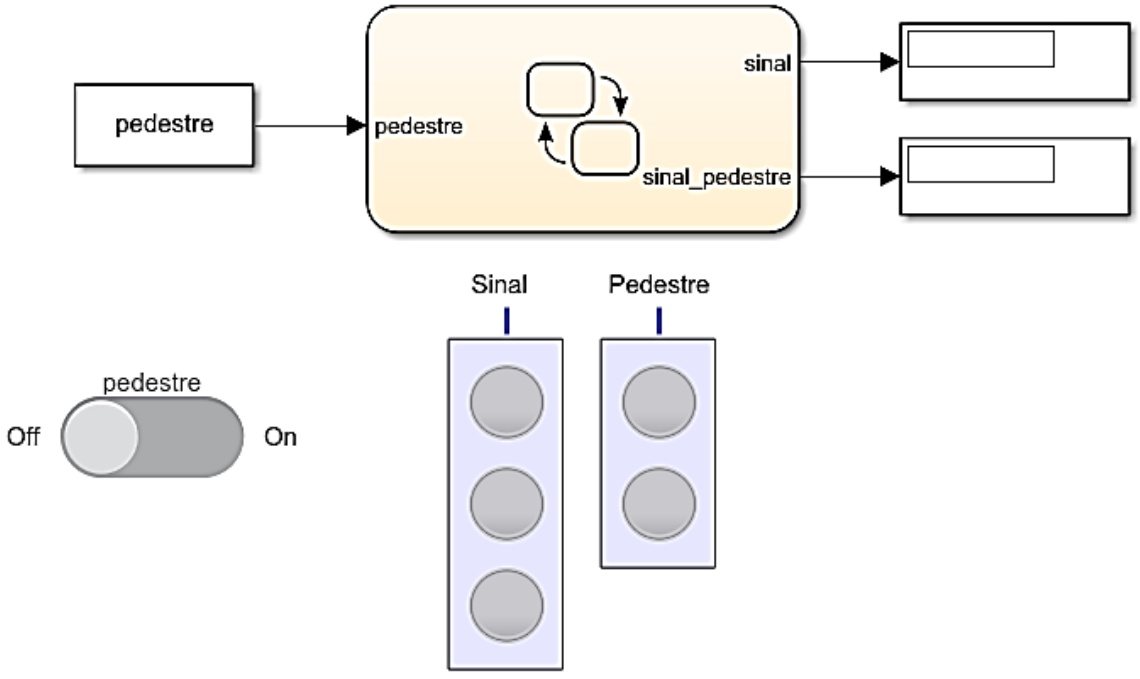
Percebe-se que, o funcionamento externo independe se o sistema foi projetado como uma máquina de Moore ou de Mealy, sendo a escolha do modelo a que mais se adaptar ao estilo do programador ou do própria *statechart*. Ressalta-se que, existem modelos híbridos, onde trabalha-se com ambos modelos de máquinas de estados.

Portanto, o exemplo desenvolvido neste roteiro teve como objetivo apresentar ao leitor conceitos fundamentais sobre a máquina de Mealy e também o método para inserir no ambiente de simulação padrões já existentes no MATLAB que auxiliam no desenvolvimento do projeto.

8.6. Procedimento experimental: Semáforo

Primeiramente é necessário adicionar no ambiente Simulink os elementos de simulação que são: *constants*, *display*, e *dashboards*. A Figura 121 demonstra o ambiente de simulação conforme os itens apresentados na descrição do sistema. Observa-se que o sistema possui duas saídas distintas uma para sinal de trânsito dos veículos (variável “sinal”) e outra para sinal dos pedestres (variável “sinal_pedestre”).

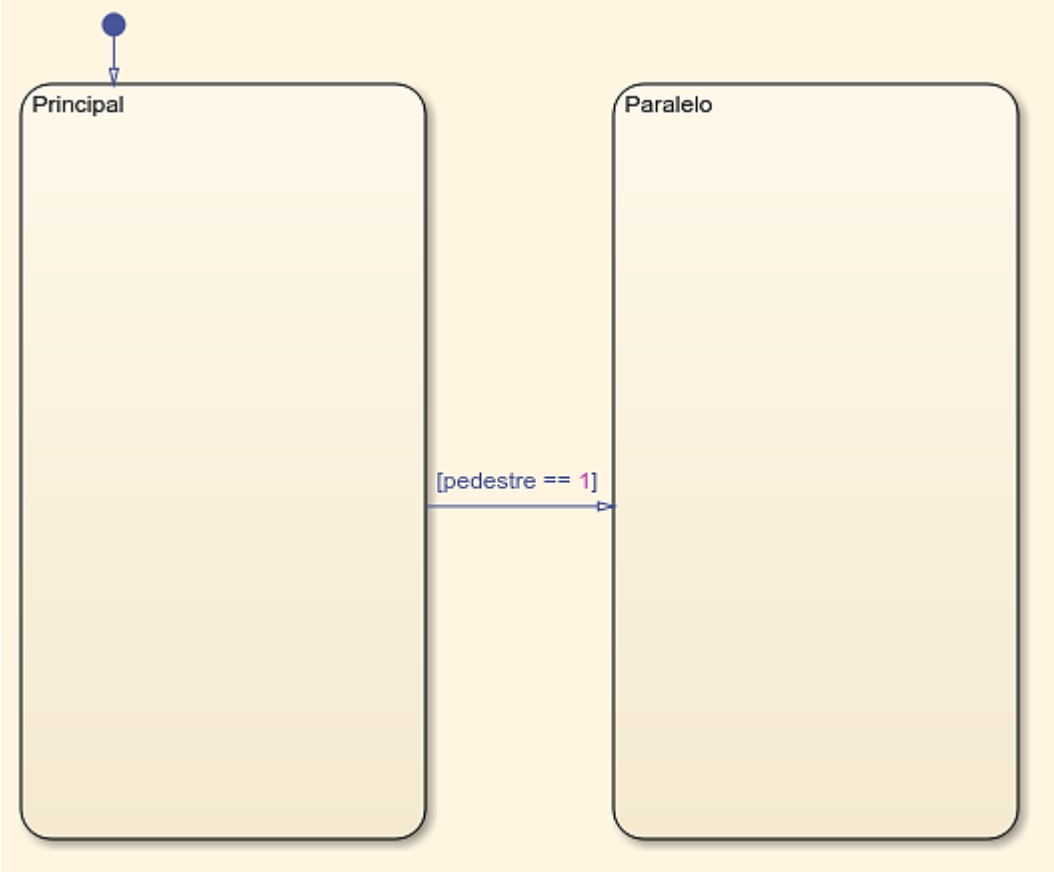
Figura 121 - Ambiente Simulink para simulação da máquina de estados.



Fonte: Elaborado pelo autor, 2021.

No ambiente Stateflow é essencial adicionar dois estados para o sistema. Um estado principal e um estado para a interrupção. O primeiro é responsável pelas configurações dos estados do sinal principal e o segundo para o sinal de pedestres. Na Figura 122 observa-se os estados inseridos no sistema, no qual ressalta-se a condição de transição de um estado para outro quando o botão de pedestres estiver em nível lógico alto.

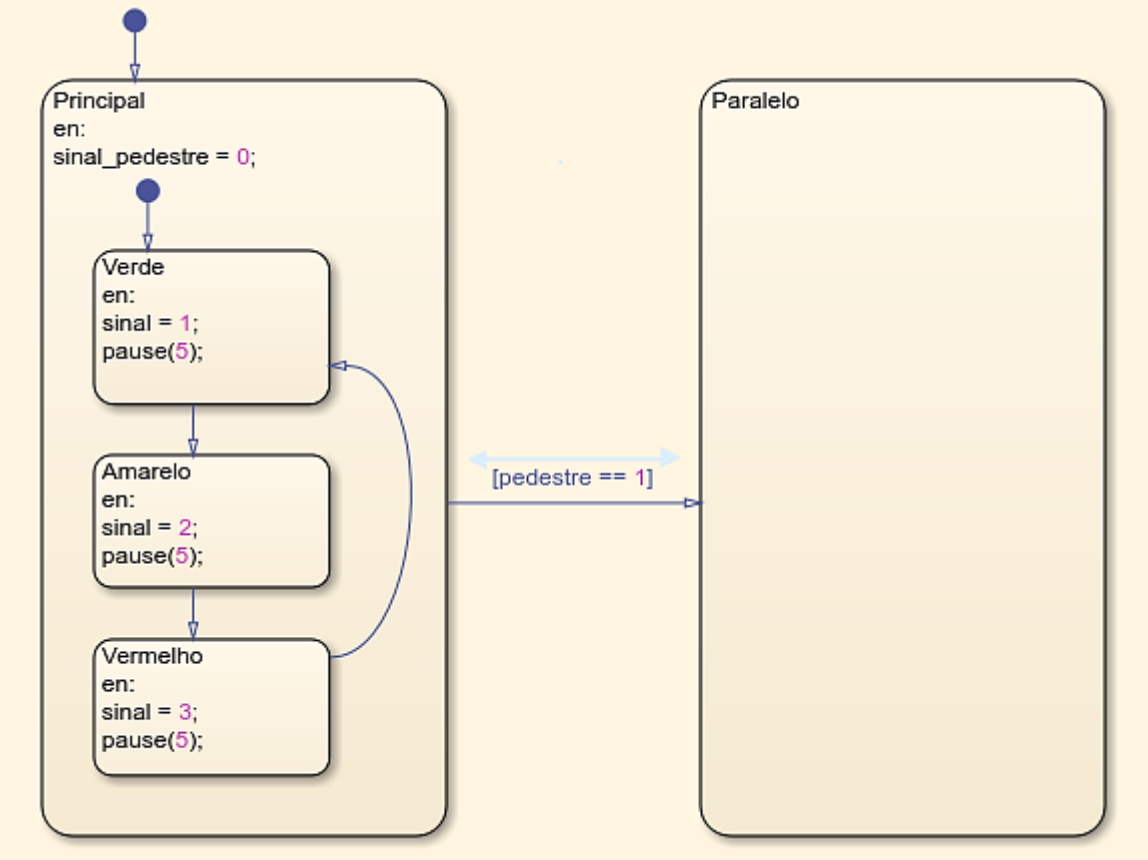
Figura 122 - Estados do semáforo com interrupção.



Fonte: Elaborado pelo autor, 2021.

Para o estado principal, são adicionadas três subestados referentes ao modo de funcionamento de um semáforo convencional, que são verde, amarelo e vermelho. É adicionado uma ação de pausa dentro de cada estado com o objetivo didático de melhorar a visualização de cada estado individualmente quando simulado.

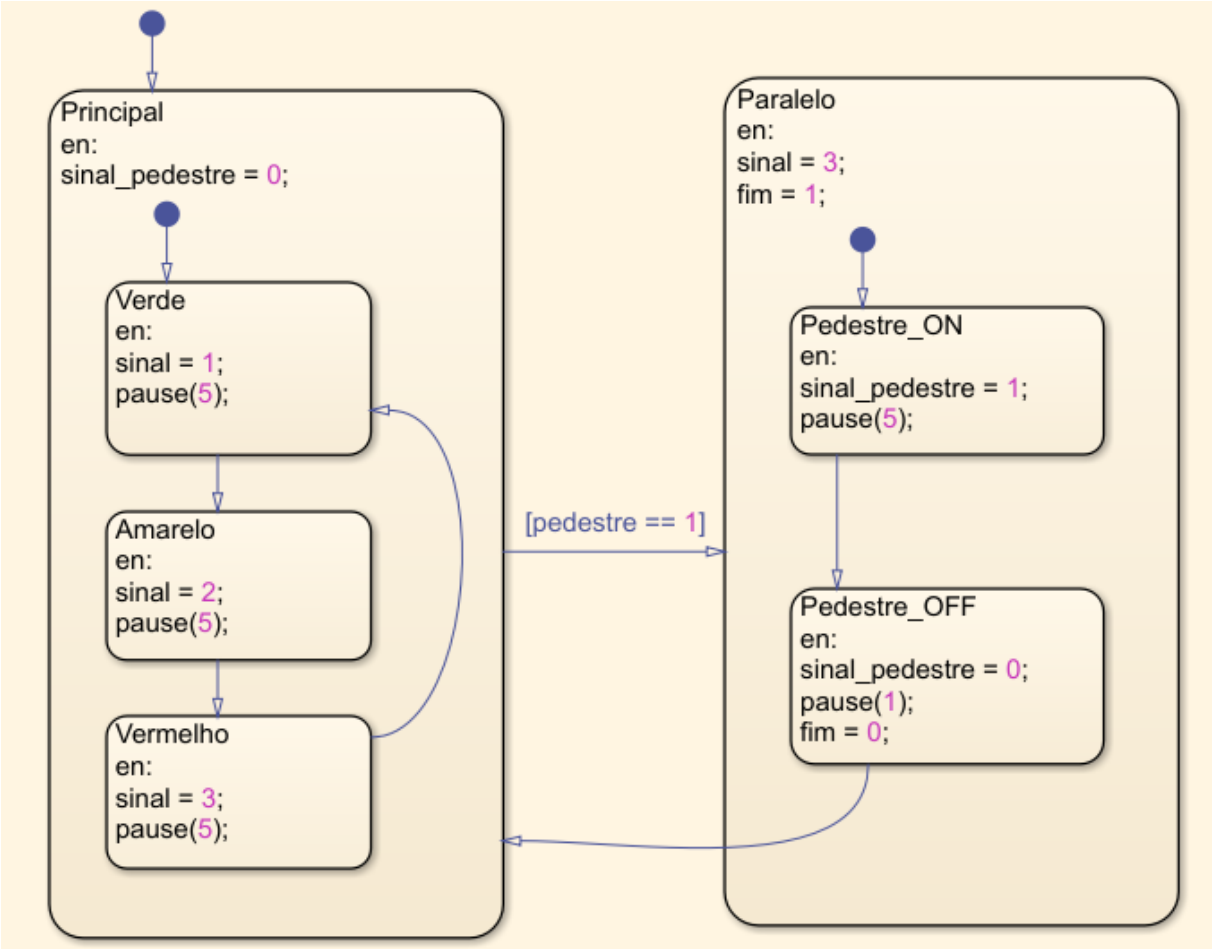
Figura 123 - Configuração do estado principal da máquina de estados.



Fonte: Elaborado pelo autor, 2021.

Para o estado secundário são adicionados os subestados de transição para um semáforo de pedestres, com sinal verde e vermelho. O estado principal é acessado quando ocorre uma interrupção no estado principal e, a condição de saída do estado secundário é tal que após ocorrer a transição do estado do sinal de pedestre para vermelho o sistema retorna ao estado principal até que ocorra outra interrupção. A Figura 124 demonstra visualmente a disposição dos estados e transições descritas.

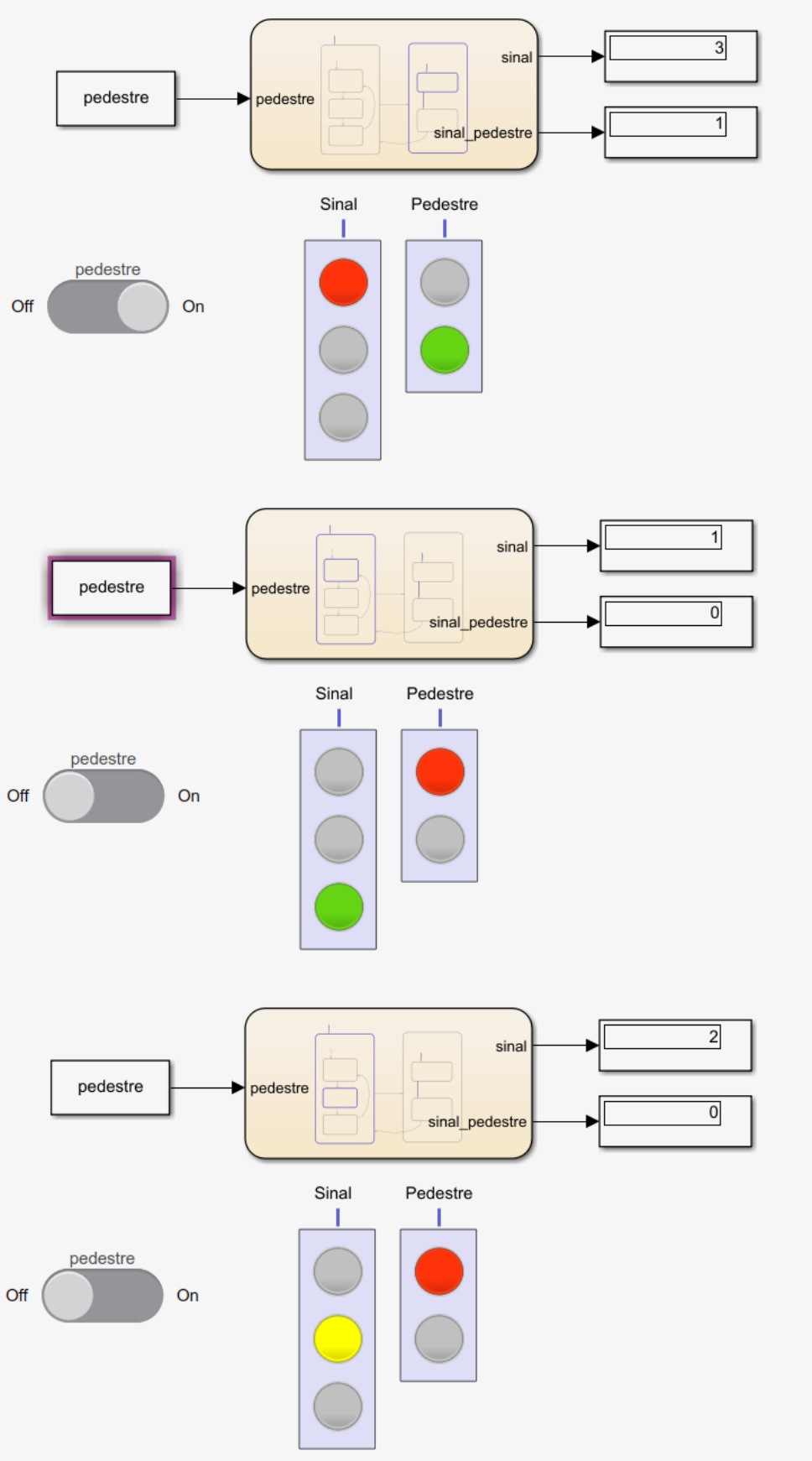
Figura 124 - Configuração do estado secundário, no qual um subestado conecta ao estado principal.



Fonte: Elaborado pelo autor, 2021.

O sistema está apto para ser simulado e pode ser visualizado tanto pelas *dashboards* quanto pelo ambiente Stateflow. A Figura 125 demonstra a máquina de estados atuando com a interrupção do botão de pedestres e também o modo de funcionamento normal, com os estados verde e amarelo como forma a ilustrar as transições do sistema.

Figura 125 - Simulação do sistema em ambiente Simulink.



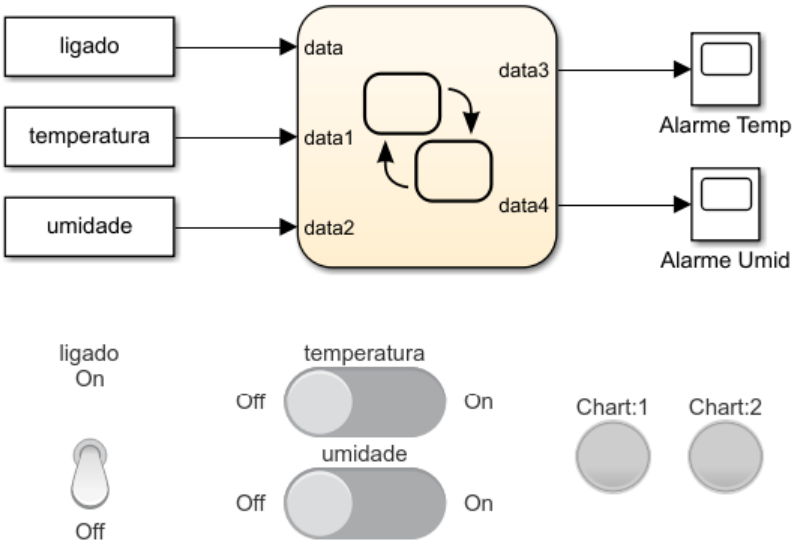
Fonte: Elaborado pelo autor, 2021.

Portanto, o sistema apresentado teve como objetivo inserir interrupções ao sistema embarcado elaborado. Através do qual, pode-se trabalhar com uma interrupção simples de um sistema em que simula-se um semáforo e cita-se a importância do tema para desenvolvimento de sistemas reativos, em que por exemplo, a interrupção pode ser um botão ou mesmo um sinal de fim de curso ou de um sensor.

8.7. Procedimento experimental: Controle de umidade e temperatura

Primeiramente é necessário elaborar o ambiente Simulink com três variáveis booleanas de entrada e duas variáveis de saída. As *dashboards* para o sistema são botões e displays simples de acordo com a criatividade do usuário. A Figura 126 determina uma das configurações possíveis para o sistema conforme a descrição do problema.

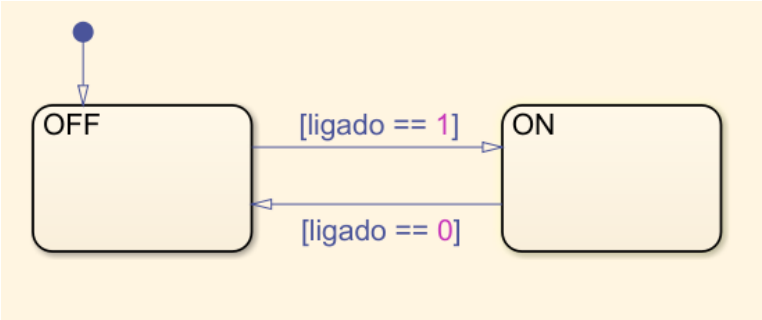
Figura 126 - Sistema Simulink para simulação de estados paralelos.



Fonte: Elaborado pelo autor, 2021.

Neste exemplo, existem dois estados “ON” e “OFF” que tem como transição a chave de liga e desliga do sistema. A Figura 127 demonstra os estados deste sistema.

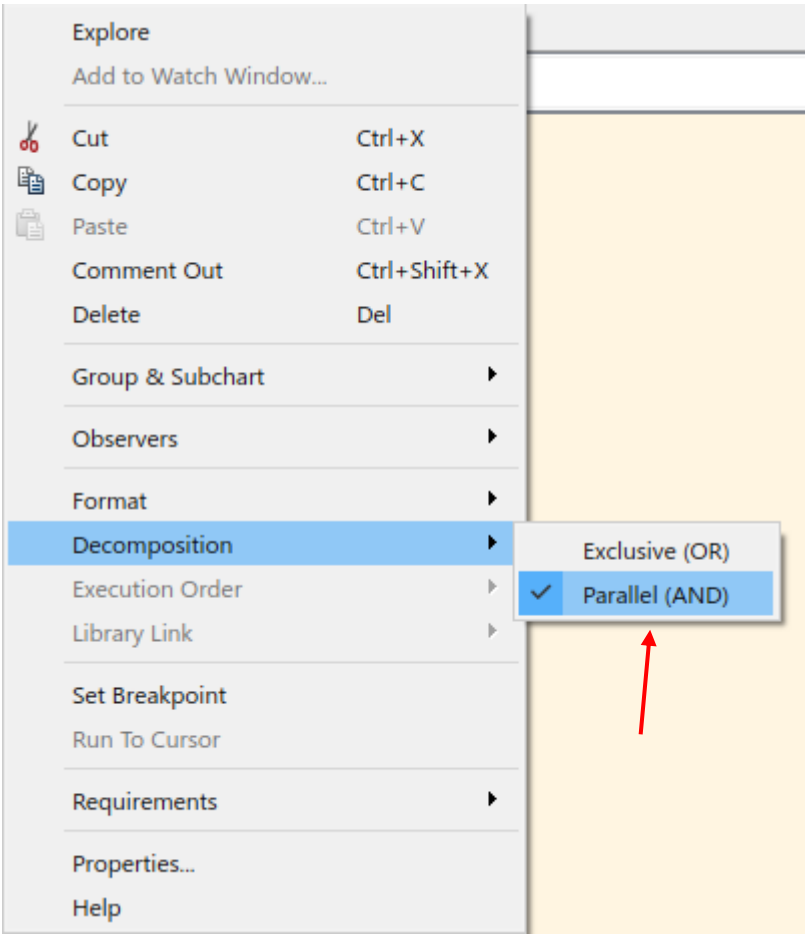
Figura 127 - Estados do sistema.



Fonte: Elaborado pelo autor, 2021.

Para o estado “ON” é necessário que duas tarefas aconteçam de modo paralelo, ou ortogonalmente, conforme exemplificado na revisão teórica. Para isso, é preciso com o botão direito com um clique sobre o estado para acessar suas configurações e depois em *decomposition* e em seguida em *Parallel* (AND). A Figura 128 demonstra o procedimento descrito.

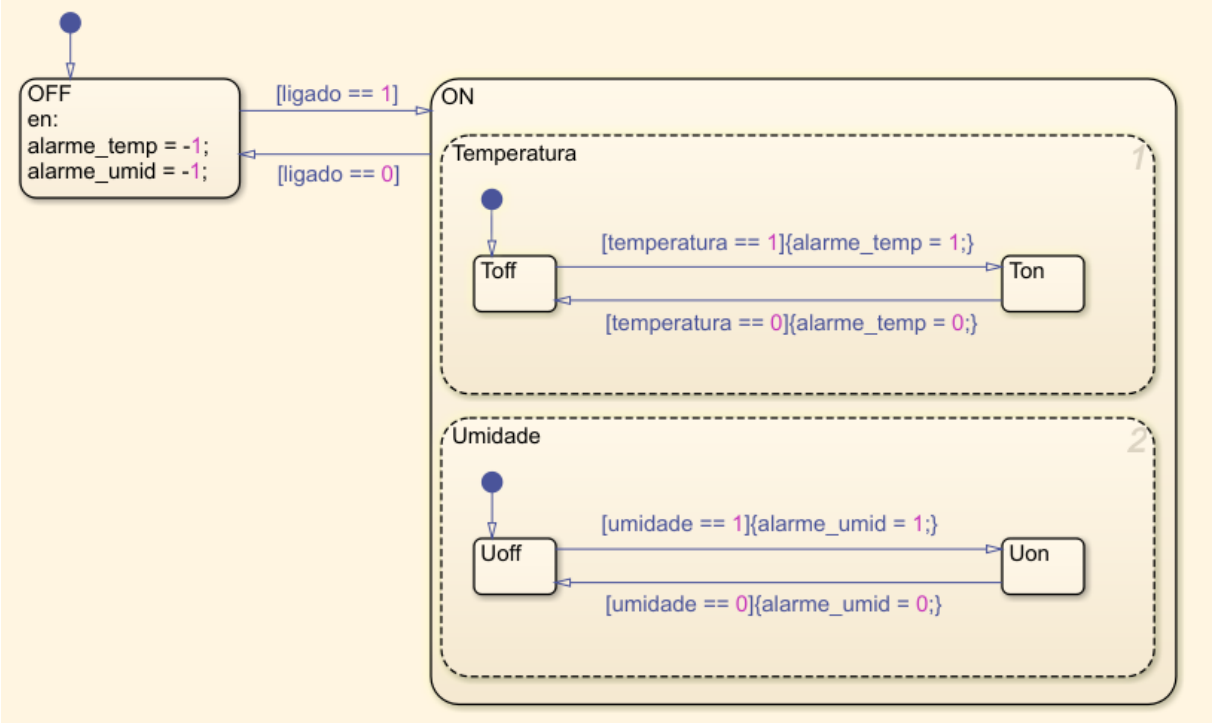
Figura 128 - Configuração para decomposição do sistema em paralelos.



Fonte: Elaborado pelo autor, 2021.

Após as configurações necessárias é possível criar dois grupos de estados, e a partir dos mesmos inserir estados menores. E válido destacar que, os estados paralelos ao serem criados possuem uma borda tracejada ao seu redor. A Figura 129 evidencia o sistema com a inserção dos estados paralelos, que seguem os mesmos passos para serem inseridos que os estados normais citados em tutoriais anteriormente e o sistema já os identifica como paralelos a partir da configuração do superestado “ON”.

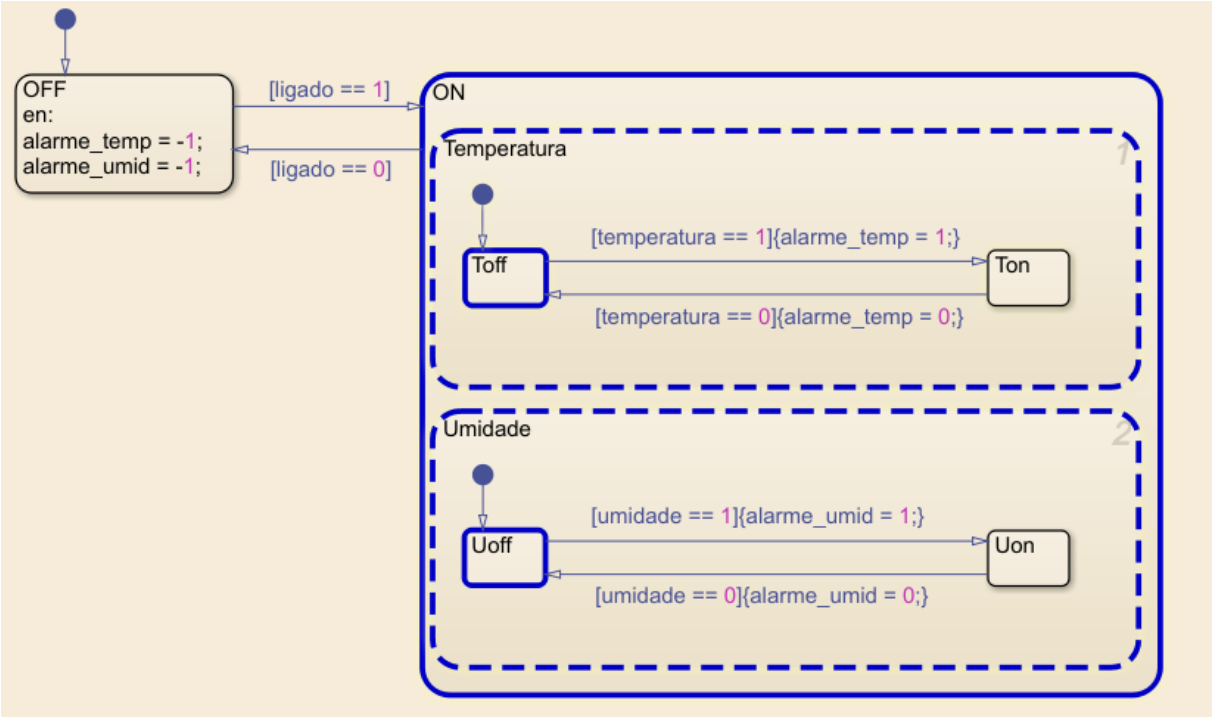
Figura 129 - Configurações dos estados paralelos de temperatura e umidade, dentro do superestado "ON".



Fonte: Elaborado pelo autor, 2021.

Com as configurações efetuadas é possível executar a simulação do sistema. A Figura 130 expõe o ponto da simulação que o estado atual é “ON” e os subestados inseridos no mesmo operam em paralelo. Para o usuário visualizar os estados atuais o Stateflow ressalta as bordas em azul desde o estado principal, quanto os subestados que operam em paralelo.

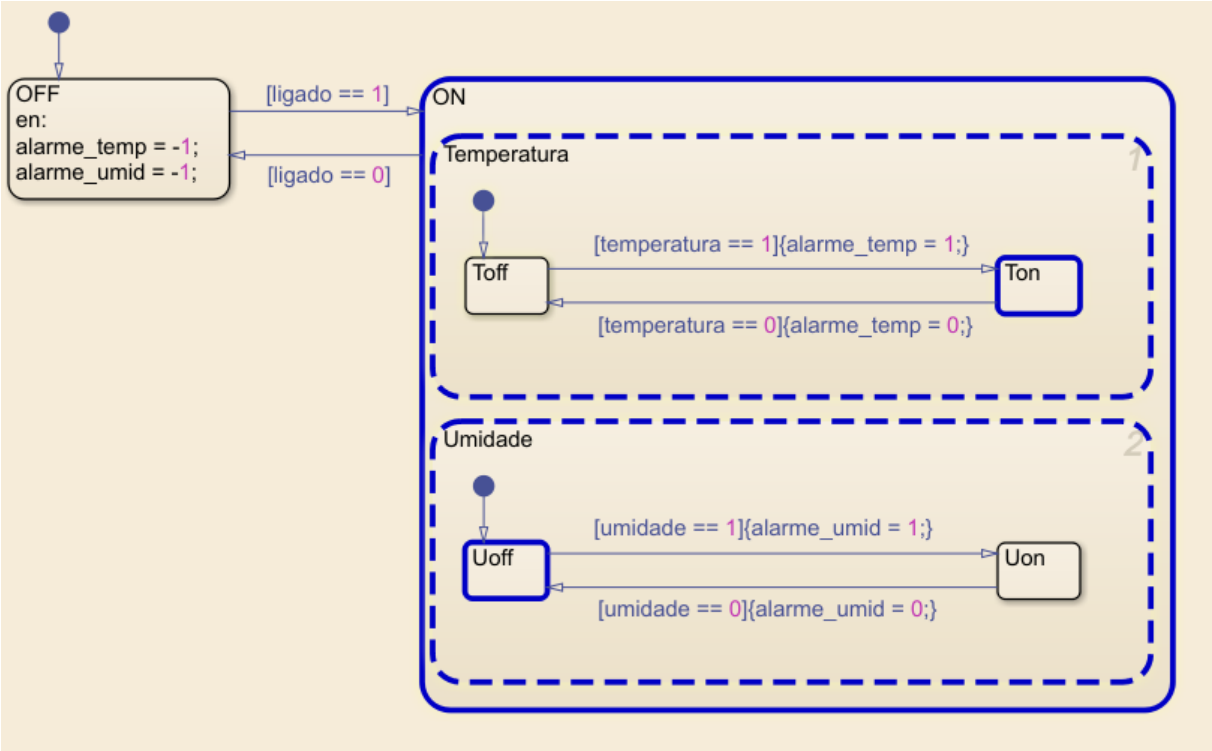
Figura 130 - Simulação do sistema para o estado "ON". Em destaque os estados paralelos operando em conjunto.



Fonte: Elaborado pelo autor, 2021.

Dentro dos estados paralelos o sistema pode realizar transições de maneira independente um do outro. Concomitante ao que é apresentado na Figura 131, o sensor de temperatura atingiu determinado nível que o faz realizar a transição para o subestado “Ton” de maneira a não afetar o funcionamento do outro grupo de estados que operam em paralelo.

Figura 131 - Estados paralelos com transições distintas entre si.



Fonte: Elaborado pelo autor, 2021.

Conforme apresentado neste tutorial é possível observar o funcionamento dos estados paralelos, ou ortogonais. Este conceito é de extrema importância uma vez que, fundamentado junto aos tutoriais anteriores o leitor torna-se apto a desenvolver os mais variados sistemas com os princípios apresentados.