

**FACULDADE CATÓLICA SALESIANA CURSO DE ENGENHARIA DA
COMPUTAÇÃO COM ÊNFASE EM ENGENHARIA DE SISTEMAS
EMBARCADOS**

ADRIEL DE SOUZA MEDEIROS

PROJETO 1 – CONTROLE DE ACESSO AO ESTACIONAMENTO

Macaé – RJ
Setembro/2021

SUMÁRIO

1. Introdução	6
2. Objetivos do projeto	6
2.1 Letra A	6
2.2 Letra B	7
3. Hardware do Sistema Embarcado	7
3.1 Componentes utilizados.....	7
3.2 Estrutura do circuito.....	7
3.2.1 Simulador (Tinkercad).....	7
3.2.2 Físico (Laboratório)	8
3.3 Código do Arduino.....	8
3.3.1 Declarações iniciais.....	9
3.3.2 Setup.....	9
3.3.3 Função de controle do fechamento da cancela	10
3.3.4 Loop	11
3.3.5 Estrutura final.....	12
4. Supervisório do Sistema Embarcado	14
4.1 Desenvolvimento do Sistema Supervisório.....	14
4.1.1 Banco de Dados.....	14
4.1.2 Biblioteca jSerialComn	16
4.1.3 Interface no JAVAFX	16
4.1.4 Código em JAVA	17
4.1.4.1 Primeiros passos.....	17
4.1.4.2 Classe Registro.....	19
4.1.4.3 Funções relacionadas ao banco de dados.....	20
4.1.4.4 Função do botão de conectar a porta selecionada	21
4.1.4.5 Função do botão de desconectar a porta selecionada	24
4.1.4.6 Função de preenchimento da ListView	25
5. Testes Gerais.....	25
6. Conclusão	32

LISTA DE FIGURAS

<i>Figura 1 Estrutura do circuito no simulador tinkercad</i>	8
<i>Figura 2 Estrutura do circuito físico no laboratório</i>	8
<i>Figura 3 Declarações iniciais do código do Arduino</i>	9
<i>Figura 4 Função setup do código do Arduino</i>	10
<i>Figura 5 Função tratar_fechar_cancela do Arduino</i>	11
<i>Figura 6 Função loop do código do Arduino</i>	12
<i>Figura 7 Estrutura completa do código do Arduino</i>	13
<i>Figura 8 Tabela do banco de dados PostgreSQL visto através do PgAdmin</i>	14
<i>Figura 9 Dados salvos no banco PostgreSQL vista através do PgAdmin</i>	15
<i>Figura 10 Interface do sistema supervisorio no JAVAFX</i>	16
<i>Figura 11 Bibliotecas utilizadas na construção do supervisorio</i>	17
<i>Figura 12 Importações do programa JAVA.</i>	18
<i>Figura 13 Declarações iniciais do programa JAVA.</i>	18
<i>Figura 14 Initialize do controller</i>	19
<i>Figura 15 Função carregarPortas</i>	19
<i>Figura 16 Classe Registro</i>	20
<i>Figura 17 Função getConexao</i>	20
<i>Figura 18 Função getPreparedStatement</i>	20
<i>Figura 19 Função getDateTime</i>	21
<i>Figura 20 Função gravar</i>	21
<i>Figura 21 Função consultarDados</i>	21
<i>Figura 22 Função do botão de conectar</i>	23
<i>Figura 23 Função do botão de desconectar</i>	25
<i>Figura 24 Função preencherLista</i>	25
<i>Figura 25 Simulação tinkercad do caso de existir um carro na entrada da cancela.</i>	26
<i>Figura 26 Simulação tinkercad do caso de não se encontrar veículo na cancela do estacionamento.</i>	26
<i>Figura 27 Simulação em laboratório do caso de existir um carro na entrada da cancela.</i>	27

<i>Figura 28 Simulação em laboratório do caso de não se encontrar veículo na cancela do estacionamento.</i>	<i>28</i>
<i>Figura 29 Sistema supervisório ao ser iniciado</i>	<i>29</i>
<i>Figura 30 Sistema supervisório com a ComboBox para apresentação das portas conectadas</i>	<i>30</i>
<i>Figura 31 Sistema supervisório ao realizar a conexão com a porta selecionada</i>	<i>31</i>
<i>Figura 32 Sistema supervisório ao ser desconectado da porta selecionada.</i>	<i>32</i>

LISTA DE TABELAS

Tabela 1 Componentes utilizados na construção do circuito _____ 7

1. Introdução

Um sistema embarcado é um sistema microprocessado no qual o computador é completamente encapsulado ou dedicado ao dispositivo ou sistema que ele controla. Ao contrário dos computadores de uso geral, como os computadores pessoais, os sistemas embarcados executam um conjunto de tarefas predefinidas, geralmente com requisitos específicos. De modo geral, tais sistemas não podem alterar suas funções durante o uso. Caso seja necessário mudar o propósito, deve-se reprogramar todo o sistema.

Os sistemas embarcados estão ficando mais baratos, mais fáceis de acessar, requerem menos consumo de energia e, além de serem mais compactos, também têm capacidades de processamento mais fortes. Com esse poder de processamento cada vez maior, com o tempo, o mundo em que vivemos se tornará cada vez mais micro conectado, onde não apenas os computadores podem acessar a Internet, mas também os objetos ao nosso redor. Até um futuro próximo, todos nós viveremos em um mundo onde a fronteira entre realidade e virtualidade é muito tênue, senão imperceptível. Esse progresso e o progresso que já estão diretamente relacionados aos sistemas embarcados.

Durante este trabalho, será exposto o exemplo de aplicação de um sistema embarcado para controle de uma cancela de estacionamento automatizada. Sendo apresentado e explicado todo processo de montagem e programação do sistema distribuído, além do processo de comunicação e construção de um sistema supervisorio utilizando a linguagem JAVA.

2. Objetivos do projeto

2.1 Letra A

Automação de cancela de entrada de automóveis, dispondo de segurança de um sensor de presença que habilitará a abertura da cancela na chegada do veículo, fechando-a no tempo de 5 segundos contados a partir da não-presença deste. Use um servomotor para simular a cancela (0° fechado, 90° aberto). Adicione circuito relé para ativar lâmpadas de segurança na entrada: Verde – entrada livre(padrão); Vermelho – entrada

em uso por veículos. Observação: para fins de simulação, adote leds como lâmpadas.

2.2 Letra B

Desenvolver a comunicação do sistema embarcado com um computador Supervisório:

- Comunicar computador Supervisório com o sistema embarcado (Arduino) via cabo USB Serial;
- Supervisório deverá registrar eventos de entrada de automóveis (gerar algum tipo de registro de log, em arquivo ou tabela em banco de dados, com interface para usuário consultar informações);
- Uso de Linguagem Java (Swing ou JSF, poderá escolher), entre outras tecnologias e bibliotecas disponíveis que julgar úteis.

3. Hardware do Sistema Embarcado

3.1 Componentes utilizados

Os componentes utilizados pelos circuitos para realizar as tarefas deste relatório estão listados na tabela abaixo:

Componentes	Quantidade
Arduino Uno R3	1
Placa de ensaio pequena	1
Fonte de energia externa	1
Sensor PIR	1
Push Button	1
Transistor npn Tip31	1
Resistor 1 K Ω	1
Resistor 330 Ω	2
Diodo	1
Relé	1
Led vermelho	1
Led verde	1
Micro servo	1

Tabela 1 Componentes utilizados na construção do circuito

3.2 Estrutura do circuito

3.2.1 Simulador (Tinkercad)

Para facilitar a visualização do leitor, definição da melhor arquitetura, além de praticar e entender um pouco mais como

funcionam os componentes utilizados, o circuito foi desenvolvido inicialmente em um simulador virtual, o tinkercad. O resultado final pode ser observado na Figura 1.

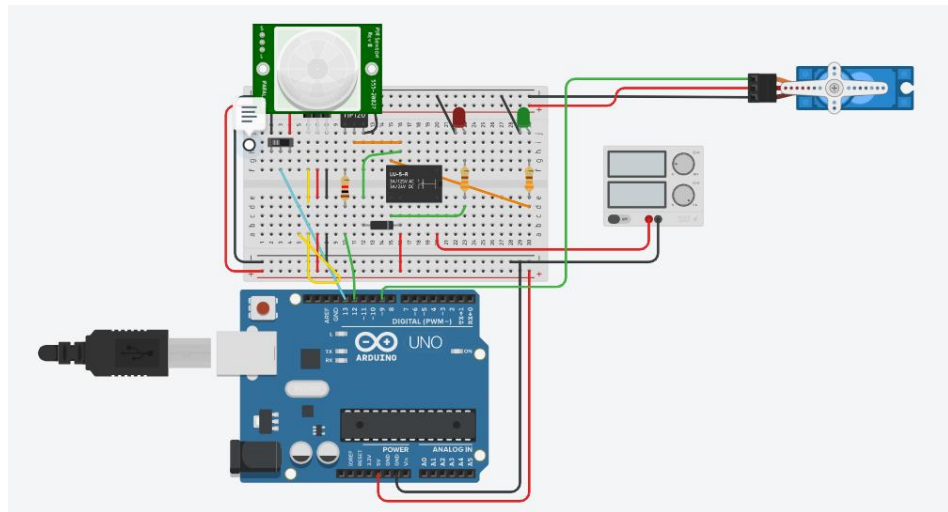


Figura 1 Estrutura do circuito no simulador tinkercad

3.2.2 Físico (Laboratório)

Este projeto exige a conexão do sistema embarcado com um sistema supervisor, tendo esse objetivo, faz-se necessário o desenvolvimento do mesmo com componentes físicos para conseguir assim realizar esta conexão, tendo em mente que em ambiente de simuladores esta tarefa não é possível. O resultado final desta montagem pode ser observado na Figura 2.

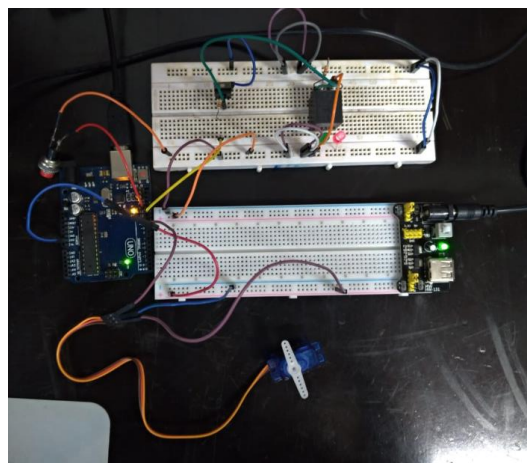


Figura 2 Estrutura do circuito físico no laboratório

3.3 Código do Arduino

Tendo como objetivo não permitir que a cancela feche muito rápido, causando assim um possível acidente, a técnica de criar uma função que monitora a situação do sensor com um delay de 5 segundos precisou ser adotada. A seguir, todo o código utilizado para o controle do Arduino será explicado.

3.3.1 Declarações iniciais

Inicialmente, foi feito a inclusão da biblioteca para utilização do servo motor, além do seu instanciamento na porta 9 do Arduino (Figura 3: linha 1 e linha 3). Além disso, foram declaradas duas constantes do tipo inteiro, uma para armazenar o pino do sensor, e outra para armazenar o pino do Arduino que envia energia para os leds (Figura 3: linha 4 e linha 5), e a variável do tipo inteiro que é utilizada para armazenar o valor atual lido do sensor (Figura 3: linha 6). A estrutura do código das declarações iniciais pode ser observada na Figura 3.

```
1  #include <Servo.h>
2
3  Servo servo_9;
4  const int pino_sensor(13);
5  const int pino(12);
6  int valor_sensor;
```

Figura 3 Declarações iniciais do código do Arduino

3.3.2 Setup

A função Setup realiza a iniciação, sendo iniciado as configurações do monitor serial com sua frequência utilizada (Figura 4: linha 9), o servo motor, com os seus parâmetros necessários (Figura 4: linha 10), além do pino do sensor com INPUT_PULLUP (Figura 4: linha 11), e o pino que envia energia para os leds (Figura 4: linha 12). A estrutura do código da inicialização das variáveis pode ser observada na Figura 4.

```

8 void setup() {
9   Serial.begin(2000000);
10  servo_9.attach(9, 500, 2500);
11  pinMode(pino_sensor, INPUT_PULLUP);
12  pinMode(pino, OUTPUT);
13 }

```

Figura 4 Função setup do código do Arduino

3.3.3 Função de controle do fechamento da cancela

Como foi dito no início do capítulo, a função de controle do fechamento da cancela se faz necessário para evitar que a mesma feche muito rápido, ou até mesmo sobre um veículo que esteja passando atrás de um outro, tendo este objetivo, a função “tratar_fechar_cancela” foi criada. Inicialmente esta função realiza um pequeno delay de 100 milissegundos. Após isso, uma variável auxiliar que controla o tempo denominada “cont_tempo” é declarada e iniciada com o valor 0. Dentro desta função existe um “for” que faz o monitoramento e incrementação da variável “cont_tempo”. Este for vai até esta variável obter o valor 50 (Figura 5: linha 20), que neste cenário equivale ao tempo de 5 segundos, já que dentro do for existe um delay de 100 milissegundos. Dentro deste “for” o valor do sensor é lido (Figura 5: linha 21) e testado em dois “if”. O primeiro “if” testa se o valor lido foi “HIGH”, caso a condição seja atendida, é feito um “print” no monitor serial com a mensagem “Aberto”. Em seguida o envio do valor “HIGH” para o pino de energia para os leds é feito, e o servo motor é redirecionado para a posição de 90°, além resetar o valor da variável “cont_tempo” para 0, reiniciando assim o “for” (Figura 5: linha 22 a linha 26).

O segundo “if” que se encontra dentro do “for” testa se o valor lido do sensor é “LOW”, caso a condição seja aceita, é feito o “print” no monitor serial com a mensagem “Fechado”, e é feito a chamada de um delay de 100 milissegundos. Caso este segundo “if” seja o atendido, a variável “cont_tempo” é incrementada e o loop continua

até que ela chegue a 50, ou que ocorra 50 delays de 100 milissegundos consecutivos no segundo “if” (Figura 5: linha 28 a 31).

Ao concluir o “for”, é feito o envio do valor “LOW” para a porta que envia energia para os leds através de um “digitalWrite” (Figura 5: linha 33), fazendo assim que o led vermelho seja acesso novamente.

A estrutura do código da função “tratar_fechar_cancela” pode ser observada na Figura 5.

```
15 void tratar_fechar_cancela() {
16     delay(100);
17
18     int cont_tempo = 0;
19
20     for(cont_tempo; cont_tempo < 50; cont_tempo++) {
21         valor_sensor = digitalRead(pino_sensor);
22         if(valor_sensor == HIGH) {
23             Serial.println("Aberto");
24             digitalWrite(pino, HIGH);
25             servo_9.write(90);
26             cont_tempo = 0;
27         } else
28             if(valor_sensor == LOW) {
29                 Serial.println("Fechado");
30                 delay(100);
31             }
32     }
33     digitalWrite(pino, LOW);
34 }
```

Figura 5 Função tratar_fechar_cancela do Arduino

3.3.4 Loop

A função loop faz a observação constante dos componentes utilizados no circuito. Neste projeto, ela foi estruturada realizando inicialmente a leitura do sensor (Figura 6: linha 38), sendo em seguida este valor testado em duas condições de “if”. O primeiro “if” testa se o valor lido do sensor é “HIGH”, caso a condição seja atendida, é feito um “print” no monitor serial com a mensagem “Aberto”, em seguida o envio do valor “HIGH” para o pino de energia para os leds é feito, e o servo motor é redirecionado para a posição de 90°, tendo após isso um pequeno delay de 100

milissegundos e a chamada da função “tratar_fechar_cancela”, que faz os testes para saber se a cancela pode ser fechada (Figura 6: linha 40 a linha 46).

O Segundo “if” testa se o valor lido do sensor é “LOW”.Ele é utilizado para manter a cancela fechada no inicio do sistema, onde ainda não foi feita a ativação do sensor e a chamada da função de “tratar_fechar_cancela”. Pode até parecer uma redundância, mas foi adquirida e aceitável dentro desse contexto para garantir a segurança do estacionamento e não permitir de forma alguma que a cancela seja aberta sem q seja necessário. Neste segundo “if”, caso a condição seja aceita, é feito o “print” no monitor serial com a mensagem “Fechado”. Em seguida, o valor “LOW” é enviado para o pino que envia energia para os leds, e o servo é posicionado na posição 0º, e um pequeno delay de 100 milissegundos é chamado (Figura 6: linha 48 a linha 53).

A estrutura do código da função loop pode ser observada na Figura 6.

```
36 void loop() {
37
38     valor_sensor = digitalRead(pino_sensor);
39
40     if(valor_sensor == HIGH){
41         Serial.println("Aberto");
42         digitalWrite(pino,HIGH);
43         servo_9.write(90);
44         delay(100);
45         tratar_fechar_cancela();
46     }
47
48     if(valor_sensor == LOW){
49         Serial.println("Fechado");
50         digitalWrite(pino,LOW);
51         servo_9.write(0);
52         delay(100);
53     }
```

Figura 6 Função loop do código do Arduino

3.3.5 Estrutura final

A estrutura final do código de controle do Arduino pode ser observada completa na Figura 7.

```
1  #include <Servo.h>
2
3  Servo servo_9;
4  const int pino_sensor(13);
5  const int pino(12);
6  int valor_sensor;
7
8  void setup(){
9      Serial.begin(2000000);
10     servo_9.attach(9, 500, 2500);
11     pinMode(pino_sensor, INPUT_PULLUP);
12     pinMode(pino, OUTPUT);
13 }
14
15 void tratar_fechar_cancela(){
16     delay(100);
17
18     int cont_tempo = 0;
19
20     for(cont_tempo; cont_tempo < 50; cont_tempo++){
21         valor_sensor = digitalRead(pino_sensor);
22         if(valor_sensor == HIGH){
23             Serial.println("Aberto");
24             digitalWrite(pino, HIGH);
25             servo_9.write(90);
26             cont_tempo = 0;
27         }else{
28             if(valor_sensor == LOW){
29                 Serial.println("Fechado");
30                 delay(100);
31             }
32         }
33         digitalWrite(pino, LOW);
34     }
35
36     void loop(){
37
38         valor_sensor = digitalRead(pino_sensor);
39
40         if(valor_sensor == HIGH){
41             Serial.println("Aberto");
42             digitalWrite(pino, HIGH);
43             servo_9.write(90);
44             delay(100);
45             tratar_fechar_cancela();
46         }
47
48         if(valor_sensor == LOW){
49             Serial.println("Fechado");
50             digitalWrite(pino, LOW);
51             servo_9.write(0);
52             delay(100);
53         }
54     }
```

Figura 7 Estrutura completa do código do Arduino

4. Supervisório do Sistema Embarcado

O sistema supervisorio tem como principal objetivo a aquisição, supervisão, controle e apresentação dos dados obtidos por um sistema embarcado, facilitando assim a visualização e análise desses dados obtidos. Tendo em vista que os sistemas embarcados podem muitas vezes se encontrarem em locais de difícil acesso, sendo o sistema supervisorio um excelente software para monitoramento desse sistema embarcado.

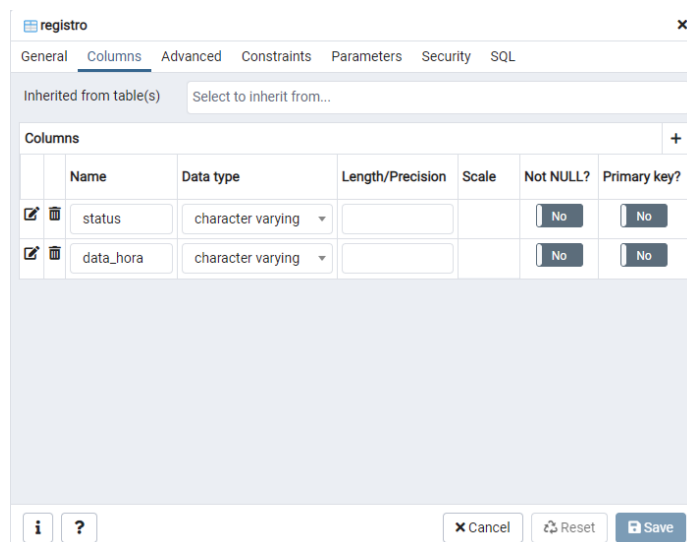
4.1 Desenvolvimento do Sistema Supervisorio

No desenvolvimento deste sistema supervisorio se fez necessário o uso de algumas tecnologias, sendo elas: PostgreSQL, Linguagem de programação JAVA, biblioteca jSerialComn e a biblioteca JAVAFX.

4.1.1 Banco de Dados

O banco de dados utilizado para armazenar as informações lidas da porta serial do Arduino do sistema embarcado desenvolvido neste trabalho foi o PostgreSQL 11, juntamente com a interface PgAdmin4 para modelagem do banco e visualização dos dados salvos no mesmo.

Para este projeto, fez-se necessário a utilização de um banco simples, contendo uma tabela e duas colunas Figura 8.



The screenshot shows the 'registro' table editor in PgAdmin4. The 'Columns' tab is selected, displaying a table with two columns: 'status' and 'data_hora'. Both columns are of type 'character varying'. The 'Not NULL?' and 'Primary key?' options are set to 'No' for both columns. The interface includes tabs for General, Columns, Advanced, Constraints, Parameters, Security, and SQL. At the bottom, there are buttons for 'Cancel', 'Reset', and 'Save'.

	Name	Data type	Length/Precision	Scale	Not NULL?	Primary key?
<input checked="" type="checkbox"/>	status	character varying			No	No
<input checked="" type="checkbox"/>	data_hora	character varying			No	No

Figura 8 Tabela do banco de dados PostgreSQL visto através do PgAdmin

A forma como os dados foram salvos no banco pode ser observado na Figura 9.




	Data Output	Explain	Notifications
	 status character varying		data_hora character varying 
1	Aberto		12/09/2021 21:04:15
2	Aberto		12/09/2021 21:04:26
3	Aberto		12/09/2021 21:04:31
4	Aberto		15/09/2021 21:30:55
5	Aberto		17/09/2021 21:22:15
6	Aberto		17/09/2021 21:22:27
7	Aberto		17/09/2021 21:29:56
8	Aberto		17/09/2021 21:29:57
9	Aberto		17/09/2021 21:29:58
10	Aberto		17/09/2021 21:30:18
11	Aberto		17/09/2021 21:30:27
12	Aberto		17/09/2021 21:30:35
13	Aberto		17/09/2021 21:30:44
14	Aberto		17/09/2021 21:30:47
15	Aberto		17/09/2021 21:31:01
16	Aberto		17/09/2021 21:32:48
17	Aberto		17/09/2021 21:34:06
18	Aberto		17/09/2021 21:34:13
19	Aberto		17/09/2021 21:35:26
20	Aberto		17/09/2021 21:35:31
21	Aberto		17/09/2021 21:35:32
22	Aberto		17/09/2021 21:35:33
23	Aberto		17/09/2021 21:38:36
24	Aberto		17/09/2021 21:38:40
25	Aberto		17/09/2021 21:38:41
26	Aberto		17/09/2021 21:38:42
27	Aberto		17/09/2021 21:39:37
28	Aberto		17/09/2021 21:40:05

Figura 9 Dados salvos no banco PostgreSQL vista através do PgAdmin

4.1.2 Biblioteca jSerialComn

A biblioteca jSerialComn destina-se a ser uma biblioteca JAVA que fornece de uma maneira independente acessar portas seriais padrão sem exigir bibliotecas externas, código nativo ou qualquer outra ferramenta. Tendo maior facilidade de uso com relação as outras alternativas encontradas para fazer ao acesso as portas seriais encontradas atualmente, e contendo ainda um suporte aprimorado para intervalos de tempo e a capacidade de abrir várias portas simultaneamente.

4.1.3 Interface no JAVA FX

Para o desenvolvimento da interface do sistema supervisor foi utilizada a biblioteca JAVA FX do JAVA. Contendo esta interface uma ComboBox para listagem das portas seriais conectadas no computador, um botão para realizar a desconexão da porta, um botão para conexão com a porta selecionada, além de uma ListView para apresentar os dados vindo do banco. O resultando final da interface pode ser visto na Figura10.

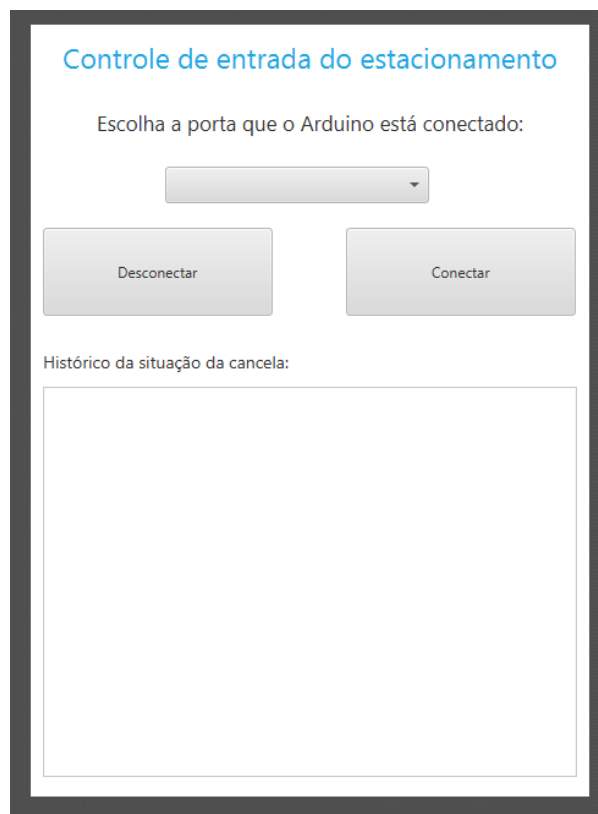


Figura 10 Interface do sistema supervisor no JAVA FX

4.1.4 Código em JAVA

No desenvolvimento deste supervisor foi utilizada a linguagem de programação JAVA, explorando sua biblioteca jSerialComn para leitura da porta serial do Arduino, e o JAVAFX para criação de interface.

4.1.4.1 Primeiros passos

Inicialmente, foi construído um projeto javaFX utilizando o JAVA 8, e importada a biblioteca jSerialComn no projeto, além da importação do driver JDBC do PostgreSQL, sendo este driver necessário para a comunicação com o banco de dados, conforme mostra a Figura 11.

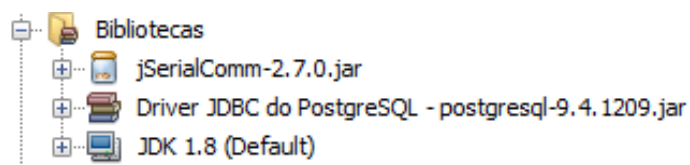


Figura 11 Bibliotecas utilizadas na construção do supervisor

Dentro do código, foi feito as importações necessárias de funcionalidades tanto do JAVA, como do JAVAFX que são necessárias no projeto (Figura 12). Além da declaração da ComboBox, o Button de conectar e o de desconectar, a ListView do tipo Registro, a SerialPort e a Thread (Figura 13).

```

1 package cancela_estacionamento;
2
3 import cancela_estacionamento.Model.Registro;
4 import com.fazecast.jSerialComm.SerialPort;
5 import java.io.InputStream;
6 import java.net.URL;
7 import java.util.ResourceBundle;
8 import javafx.fxml.FXML;
9 import javafx.fxml.Initializable;
10 import javafx.scene.control.Button;
11 import javafx.scene.control.ComboBox;
12 import java.sql.Connection;
13 import java.util.Date;
14 import java.sql.DriverManager;
15 import java.sql.PreparedStatement;
16 import java.sql.ResultSet;
17 import java.sql.SQLException;
18 import java.sql.Statement;
19 import java.text.DateFormat;
20 import java.text.SimpleDateFormat;
21 import java.util.ArrayList;
22 import java.util.List;
23 import java.util.logging.Level;
24 import java.util.logging.Logger;
25 import javafx.collections.FXCollections;
26 import javafx.collections.ObservableList;
27 import javafx.scene.control.ListView;

```

Figura 12 Importações do programa JAVA.

```

31 @FXML
32 private ComboBox cbPortas;
33
34 @FXML
35 private Button btnConectar;
36
37 @FXML
38 private Button btnDesconectar;
39
40 @FXML
41 private ListView<Registro> lstRegistros;
42
43 private SerialPort porta;
44
45 Thread thread;

```

Figura 13 Declarações iniciais do programa JAVA.

O “initialize” do “controler” realiza a chamada das funções preencherLista, carregarPortas e a getConexao (Figura 14).

```

47  @Override
48  public void initialize(URL url, ResourceBundle rb) {
49      preencherLista();
50      carregarPortas();
51      try {
52          getConexao();
53      } catch (SQLException ex) {
54          Logger.getLogger(FXMLDocumentController.class.getName()).log(Level.SEVERE, null, ex);
55      }
56  }

```

Figura 14 Initialize do controller

A função `carregarPortas` realiza um “for” buscando as portas que possuem conexão no computador (Figura 15), salvando-as para preencher a ComboBox com o nome dessas portas. Já as outras funções chamadas no “initialize” serão explicadas nos tópicos seguintes.

```

58  private void carregarPortas() {
59      SerialPort[] portNames = SerialPort.getCommPorts();
60
61      for (SerialPort portName : portNames) {
62          cbPortas.getItems().add(portName.getSystemPortName());
63      }
64  }

```

Figura 15 Função `carregarPortas`

4.1.4.2 Classe Registro

A classe registro foi desenvolvida para realizar a manipulação e controle dos dados que vão ser salvos e consultados do banco de dados. Dentro de sua estrutura encontra-se a declaração das variáveis referentes as colunas do banco de dados, ou métodos “get” e “set” de cada uma delas, além do método de retorno no formato desejado. A estrutura da classe registro pode ser observada na Figura 16.

Já a função `getDateTime` (Figura 19), é responsável por disponibilizar a data e hora atual no formato correto para salva-la no banco quando necessário.

```
145 private String getDateTime() {  
146     DateFormat dateFormat = new SimpleDateFormat("dd/MM/yyyy HH:mm:ss");  
147     Date date = new Date();  
148     return dateFormat.format(date);  
149 }
```

Figura 19 Função `getDateTime`

A função `gravar` (Figura 20), quando chamada, realiza a gravação no banco de dados do status da cancela e da data e hora atual do sistema.

```
151 public void gravar(String response) throws Exception {  
152     String sql = "insert into registro (status, data_hora) values (?,?)";  
153     PreparedStatement ps = getPreparedStatement(false, sql);  
154  
155     ps.setString(1, response);  
156     ps.setString(2, getDateTime());  
157     ps.executeUpdate();  
158 }
```

Figura 20 Função `gravar`

Enquanto a função `consultarDados` (Figura 21), quando chamada, realiza a busca dos dados atuais contidos no banco de dados e os retorna.

```
160 public List<Registro> consultarDados() throws Exception {  
161     String sql = "SELECT * FROM registro order by data_hora desc";  
162     PreparedStatement ps = getPreparedStatement(false, sql);  
163  
164     ResultSet rs = ps.executeQuery();  
165  
166     List<Registro> registros = new ArrayList<Registro>();  
167     while (rs.next()) {  
168         Registro registro = new Registro();  
169         registro.setStatus(rs.getString("status"));  
170         registro.setDataHora(rs.getString("data_hora"));  
171         registros.add(registro);  
172     }  
173  
174     return registros;  
175 }
```

Figura 21 Função `consultarDados`

4.1.4.4 Função do botão de conectar a porta selecionada

A ação do clique no botão `conectar` chama o método `btnConectarAction` (Figura 22), que começa realizando o preenchimento da `ListView` com os dados mais recentes do

banco de dados, e em seguida é feita a atribuição do valor do `SerialPort`, começando com um `getCommPort` da porta selecionada na `ComboBox`, e em seguida um `setComPortTimeouts` com o parâmetro `TIMEOUT_READ_SEMI_BLOCKING`, além de 1000 milissegundos nos dois parâmetros subsequentes, que são referentes ao tempo limite de leitura de semi-bloqueio. Após esta etapa é feito o `setBaudRate` para ajustar a velocidade de acordo com a configurada na IDE do Arduino, como parâmetro foi passado o valor mais alto possível, 2000000, pois foi o que apresentou o melhor desempenho de comunicação com a porta serial. Após estas etapas, é declarado um `InputStream` que recebe como valor o `getInputStream` da porta selecionada.

```

66 @FXML
67 private void btnConectarAction() throws SQLException {
68
69     preencherLista();
70
71     porta = SerialPort.getCommPort(cbPortas.getSelectionModel().getSelectedItem().toString());
72     porta.setComPortTimeouts(SerialPort.TIMEOUT_READ_SEMI_BLOCKING, 10000, 10000);
73     porta.setBaudRate(2000000);
74     InputStream in = porta.getInputStream();
75
76     thread = new Thread() {
77         String aux = "";
78
79         public void run() {
80             int availableBytes = 0;
81             System.out.println(availableBytes);
82             do {
83                 try {
84                     System.out.println("Thread");
85                     availableBytes = porta.bytesAvailable();
86                     System.out.println(availableBytes);
87                     if (availableBytes > 0) {
88                         byte[] buffer = new byte[1024];
89                         int bytesRead = porta.readBytes(buffer, Math.min(buffer.length, porta.bytesAvailable()));
90                         String response = new String(buffer, 0, bytesRead);
91                         System.out.println(response);
92                         if (!response.equals(aux)) {
93                             if (response.startsWith("A")) {
94                                 gravar(response);
95                             }
96                         }
97                         aux = response;
98                     }
99                     Thread.sleep(1000);
100                     in.close();
101                 } catch (Exception e) {
102                     e.printStackTrace();
103                 }
104             } while (availableBytes > -1);
105         }
106     };
107
108     porta.openPort();
109     cbPortas.setDisable(true);
110     thread.start();
111     btnConectar.setDisable(true);
112     btnDesconectar.setDisable(false);
113 }

```

Figura 22 Função do botão de conectar

Após essas declarações, uma Thread é criada para ser executada durante todo o tempo que o software está em execução. Dentro desta Thread inicialmente é feito a declaração de uma variável auxiliar que recebe o valor vazio. Em seguida um “do while” é criado para garantir que o código contido dentro dele seja executado sempre que o availableBytes seja maior que -1, sendo que o availableBytes é o valor referente a quantidade de bytes que estão sendo lidos pela porta serial. Dentro do “do while”, primeiro o valor do availableBytes é setado com o bytesAvailable da porta selecionada, e em seguida um “if” confirma se este valor é maior do que 0. Com a condição do

“if” atendida, um array de byte é criado recebendo o valor de 1024, e em seguida é criada uma variável inteira chamada “bytesRead” que recebe a leitura dos bytes vindos da porta serial. Após esta etapa, a string vinda da porta serial é atribuída a uma variável chamada “response”, que é testada em um “if” para saber se ela não é igual ao valor da variável auxiliar criada no início da Thread. Caso não seja igual um novo teste é feito para saber se a variável “response” inicia com a letra “A”, uma referência a mensagem “Aberto” vinda da porta serial, caso seja atendida a condição é feita a gravação no banco do valor de “response”. Após estes testes a variável auxiliar recebe o valor atual de response para poder fazer o teste no próximo loop do “do while” e saber se a mensagem do “response” é igual ao seu valor, ou sejam, se o “response” atual possui o mesmo do “response” passado, não é feita a gravação no banco. Em seguida o Thread.sleep com o atributo 100 é chamado e o InputStream é fechado.

Quando o availableBytes obter o valor maior que -1 o loop é finalizado e fora da Thread a porta selecionada recebe um openPort, a comboBox é desabilitada, a Thread é iniciada, o botão de conectar é desabilitado para evitar o duplo clique do usuário e o botão de desconectar é habilitado.

4.1.4.5 Função do botão de desconectar a porta selecionada

O botão de desconectar chama a função btnDesconectarAction (Figura 23), que por sua vez, realiza o preenchimento da ListView com os dados mais recentes do banco de dados, realiza um interrupt na thread, e fecha a porta conectada, sendo que essa ação faz com que a variável availableBytes receba o valor de -1, parando assim o loop que acontece na Thread contida na função do botão de conectar. Após isso, a ComboBox e o botão de conectar

são habilitados novamente, e o botão de desconectar é desabilitado para impedir o duplo clique pelo usuário.

```
115 @FXML
116 public void btnDesconectarAction() {
117
118     preencherLista();
119
120     thread.interrupt();
121     porta.closePort();
122     cbPortas.setDisable(false);
123
124     btnConectar.setDisable(false);
125     btnDesconectar.setDisable(true);
126 }
```

Figura 23 Função do botão de desconectar

4.1.4.6 Função de preenchimento da ListView

A função de preenchimento da ListView (Figura 24) inicia declarando uma lista de Registro chamada registros, que vai ser utilizada para armazenar os registros do banco. Em seguida essa lista recebe o valor retornado do chamado da função consultarDados, que retorna os dados do banco de Dados. Após esta etapa um ObservableList de Registro chamado data é criado recebendo FXCollections.observableArrayList da lista de registros. Por fim, a ListView é setada com os valores contidos no ObservableList data.

```
177 private void preencherLista() {
178     List<Registro> registros;
179     try {
180         registros = consultarDados();
181         ObservableList<Registro> data = FXCollections.observableArrayList(registros);
182         lstRegistros.setItems(data);
183     } catch (Exception e) {
184         e.printStackTrace();
185     }
186 }
187 }
```

Figura 24 Função preencherLista

5. Testes Gerais

Para fins de simulação, foi adotado um botão como sensor, tendo em vista que a manipulação do sensor PIR é um pouco difícil para realizar simulações sequenciais.

Observa-se que o circuito se comportou exatamente como esperado ao simular o caso de possuir um veículo na entrada da cancela, realizando o posicionamento do servo motor em 90º graus, e acendendo o led verde, indicando que o veículo pode prosseguir, conforme pode ser visto na Figura 25.

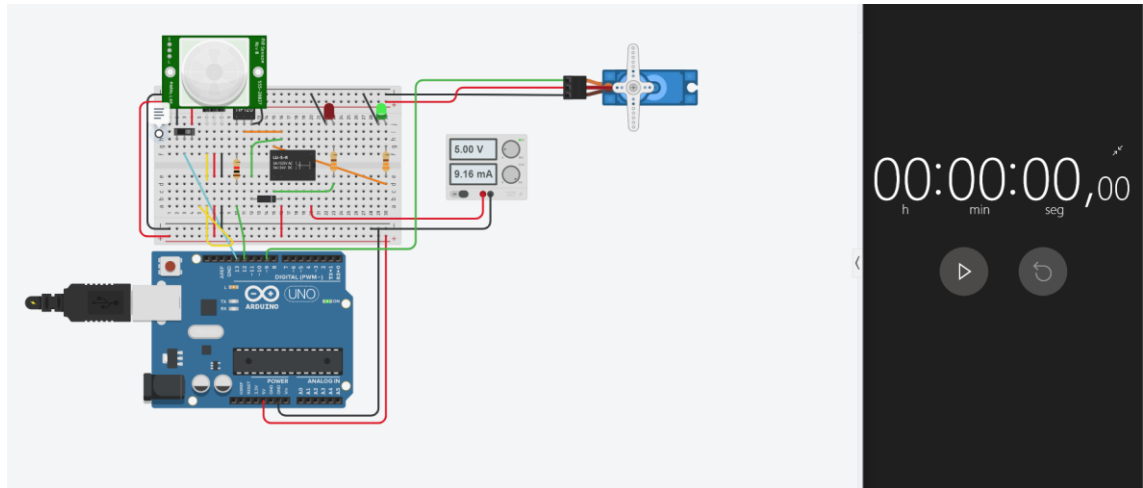


Figura 25 Simulação tinkercad do caso de existir um carro na entrada da cancela.

Ao simular a retirada do veículo, observa-se que o circuito também se comporta como esperado, realizando o fechamento da cancela 5 segundos após a saída do mesmo, conforme mostra a Figura 26. Vale ressaltar que a contagem do tempo pode não ter sido tão precisa, e de exatos 5 segundos devido a diferença do tempo de conseguir clicar no botão para simular a saída do veículo, e o click no cronômetro para contagem do tempo.

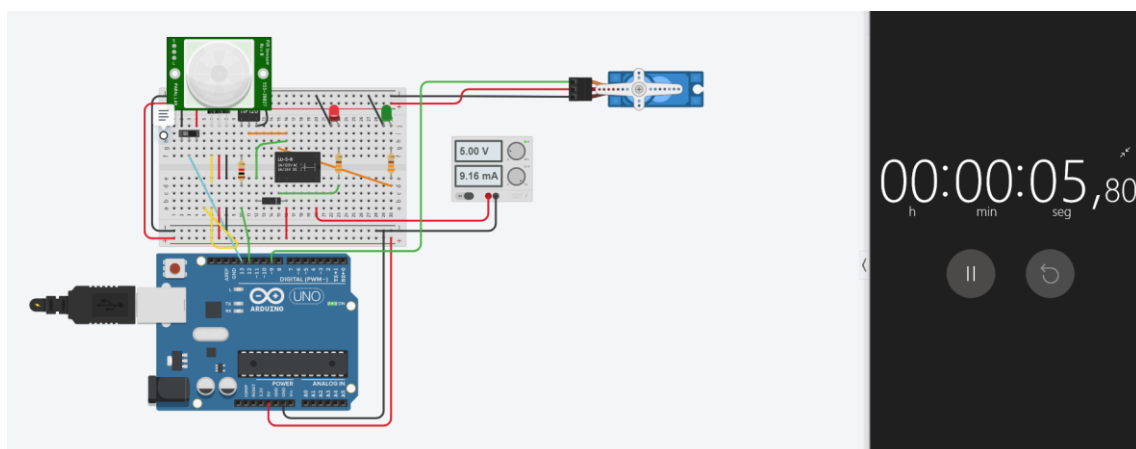


Figura 26 Simulação tinkercad do caso de não se encontrar veículo na cancela do estacionamento.

A visualização da simulação em laboratório pode não ficar tão nítida em imagens, mas mesmo assim essas serão apresentadas neste relatório, tendo em vista que o resultado foi o mesmo que obtido no simulador. Os testes em laboratório seguem a mesma ordem de teste do exemplo com o simulador, e são apresentadas na Figura 27 e Figura 28.

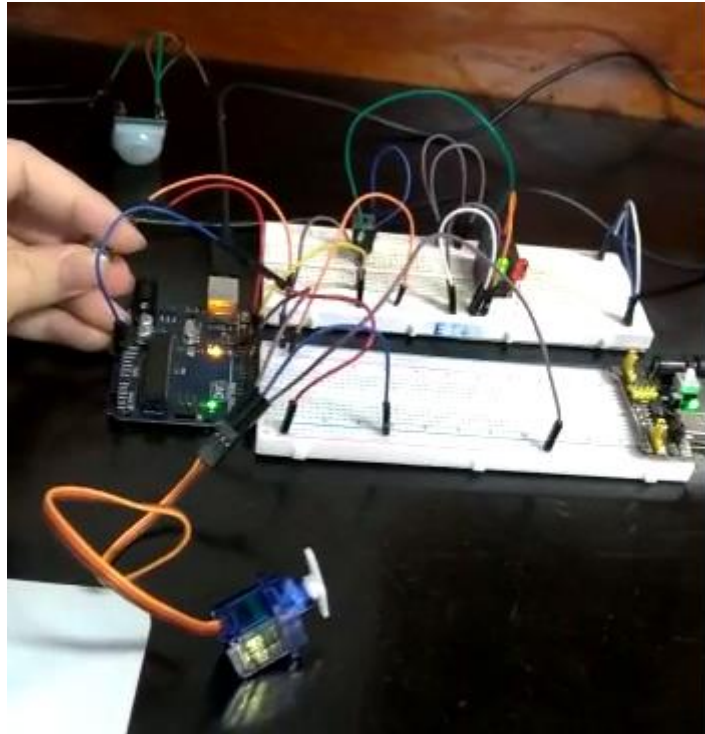


Figura 27 Simulação em laboratório do caso de existir um carro na entrada da cancela.

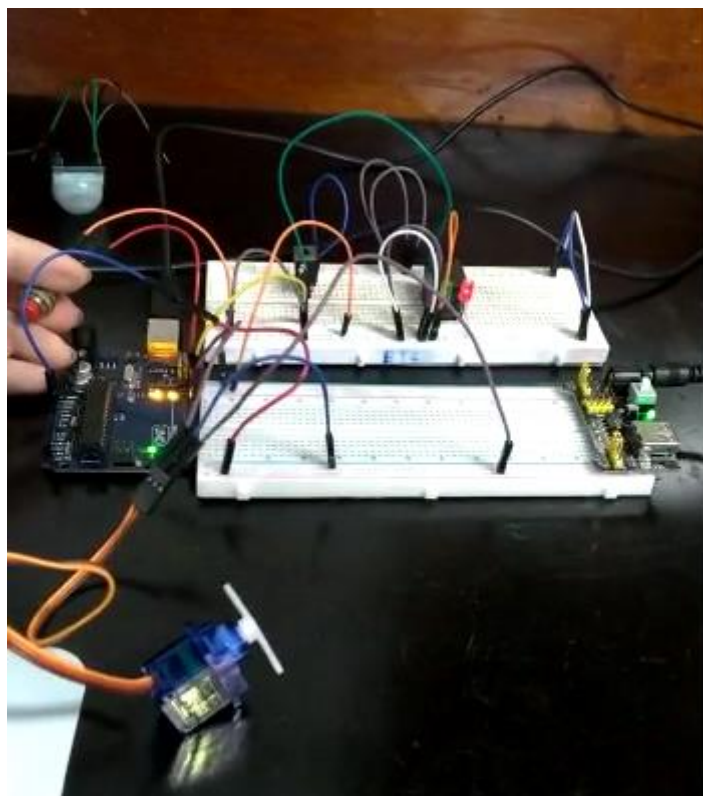


Figura 28 Simulação em laboratório do caso de não se encontrar veículo na cancela do estacionamento.

O sistema supervisorio em execução é apresentado nas imagens que estão a seguir (Figura: 29, 30, 31 e 32), contendo nelas a ilustração de como ele se comporta durante sua execução. Com os dados atuais do banco sendo apresentados na interface, controle das portas conectadas no computador e o controle dos botões para evitar o uso indevido do mesmo pelo usuário.

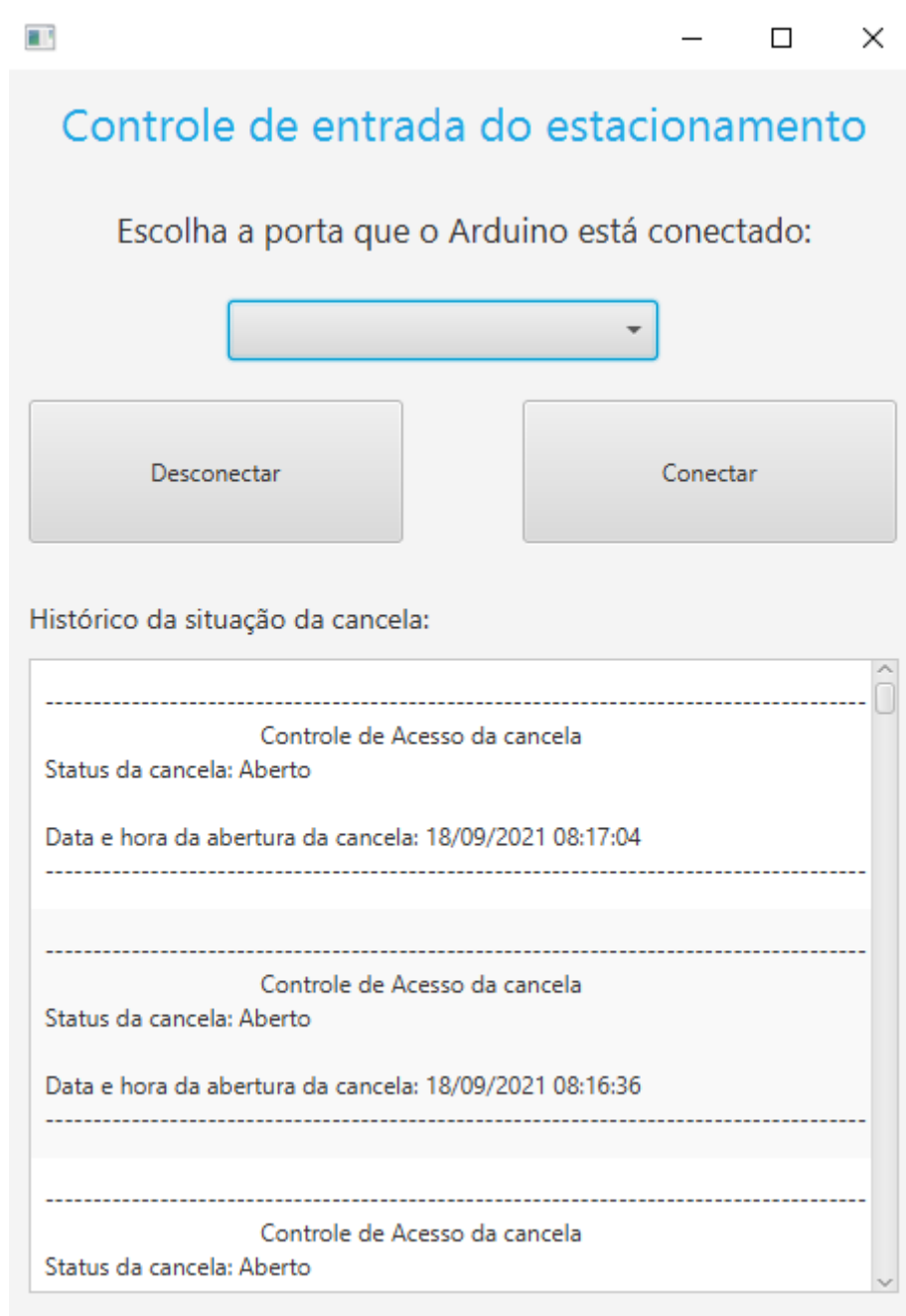


Figura 29 Sistema supervisorio ao ser iniciado

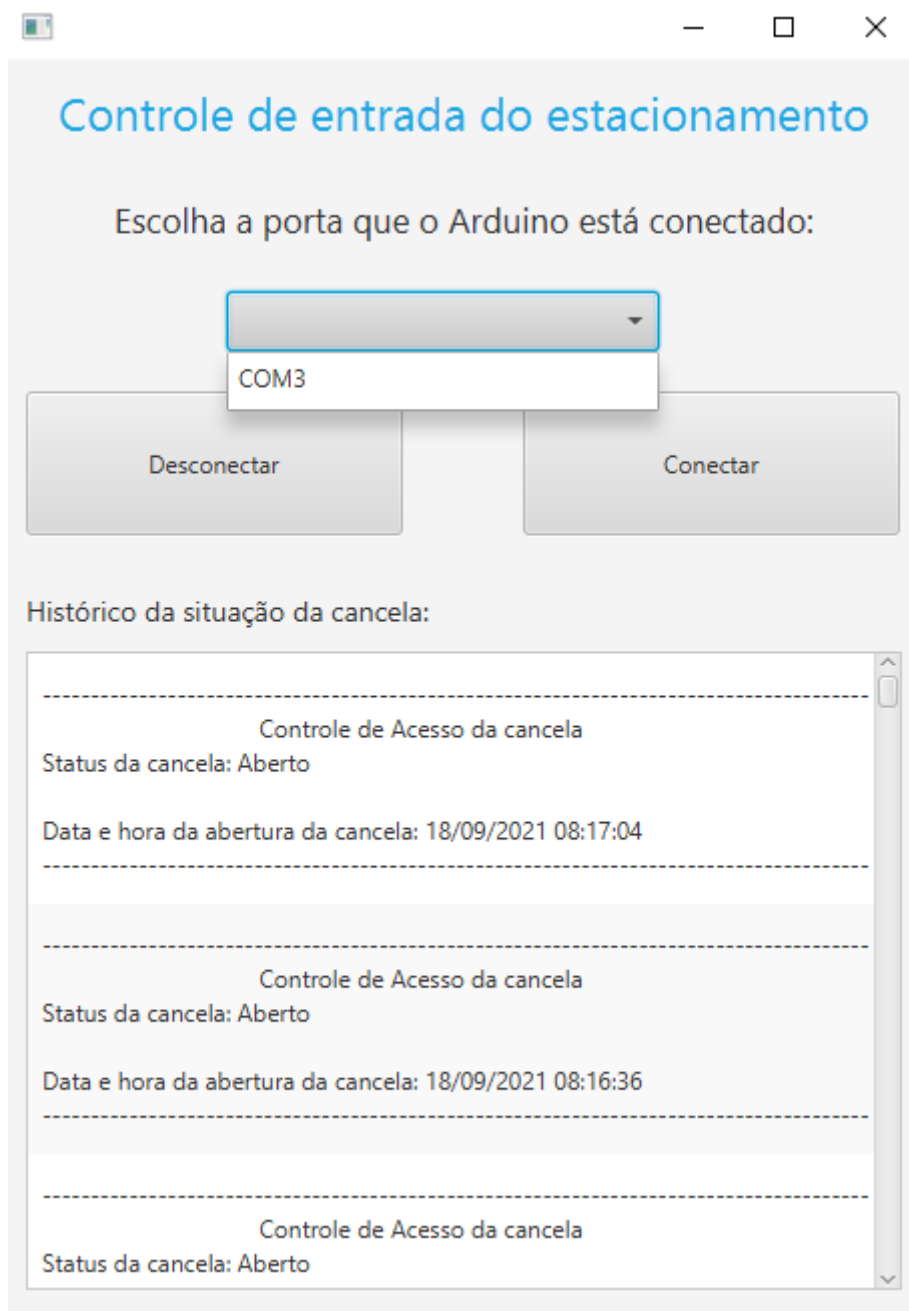


Figura 30 Sistema supervisorio com a ComboBox para apresentação das portas conectadas

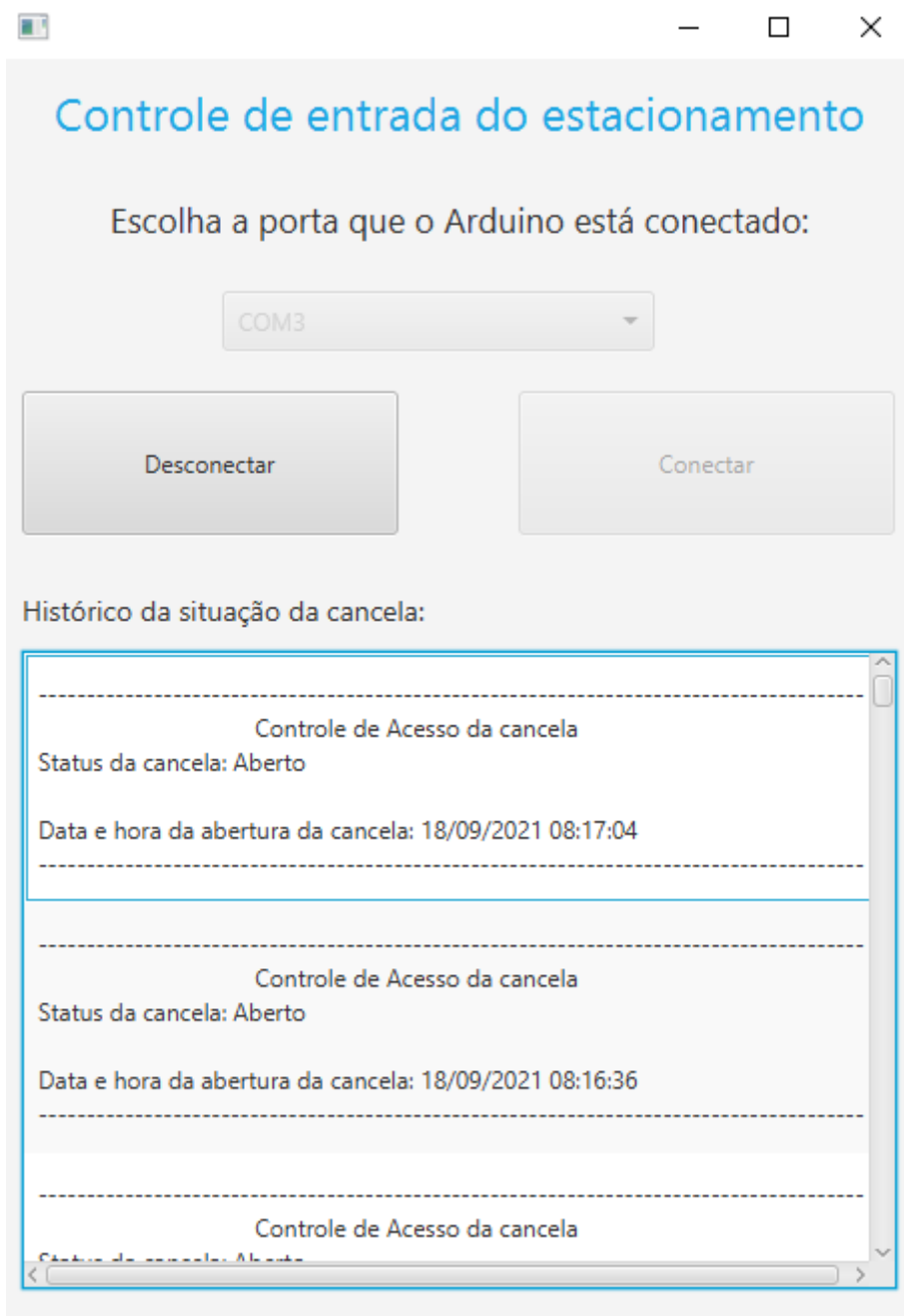


Figura 31 Sistema supervisorio ao realizar a conexão com a porta selecionada

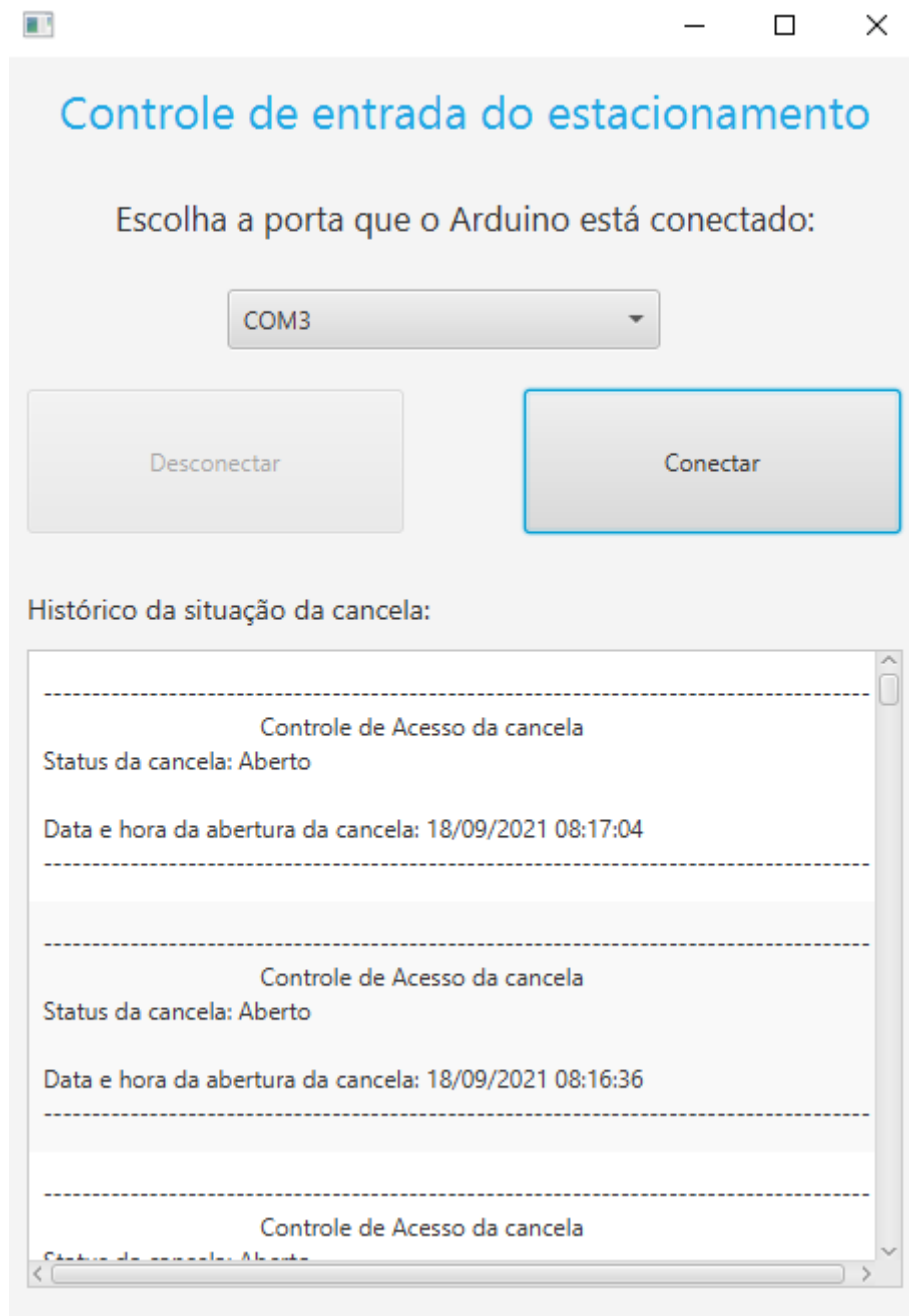


Figura 32 Sistema supervisorio ao ser desconectado da porta selecionada.

6. Conclusão

A aplicação de sistemas embarcados pode ser aplicada em diferentes situações, e vem cada vez mais sendo implantada no dia a dia das pessoas, sem mesmo que consigam perceber.

Neste projeto, pode-se observar na prática como funciona todo o processo de criação de um sistema embarcado que contém um sistema supervisorio para criação de um hardware e software de controle de cancela

de estacionamento, sendo este um trabalho acadêmico que estimula o aluno a aplicar seus conhecimentos teóricos no desenvolvimento de soluções para os problemas do mundo real.