



DESENVOLVIMENTO DE UM ALIMENTADOR AUTOMÁTICO PARA GATOS COM CONTROLE VIA APLICATIVO

Resumo Este artigo apresenta o desenvolvimento de um alimentador automático para gatos, que permite o controle dos horários de alimentação por meio de um aplicativo. O sistema utiliza um ESP32 para processamento e conectividade WiFi, além de um display LCD e uma interface touch para interação direta. O alimentador foi projetado para oferecer praticidade aos tutores, garantindo que os gatos sejam alimentados nos horários programados, mesmo na ausência dos donos. O desenvolvimento incluiu a integração de hardware e software.

Palavras-chave: Alimentador automático. ESP32. WiFi. Aplicativo móvel. Interface touch.

Abstract: *This paper presents the development of an automatic cat feeder, which allows feeding schedule control through a mobile application. The system uses an ESP32 for processing and WiFi connectivity, along with an LCD display and a touch interface for direct interaction. The feeder was designed to provide convenience to pet owners, ensuring that cats are fed at scheduled times, even in the absence of their owners. The development included hardware and software integration.*

Keywords: *Automatic feeder. ESP32. WiFi. Mobile application. Touch interface.*

INTRODUÇÃO

A alimentação regular e adequada dos gatos é essencial para sua saúde e bem-estar. No entanto, muitos tutores enfrentam dificuldades em manter uma rotina alimentar devido a compromissos diários. Para solucionar esse problema, os alimentadores automáticos se tornaram uma alternativa viável, permitindo que os animais sejam alimentados mesmo na ausência de seus donos.

Neste contexto, este artigo apresenta o desenvolvimento de um alimentador automático para gatos, que permite o controle dos horários de alimentação por meio de um aplicativo. Diferente de alguns modelos disponíveis no mercado, este projeto foca exclusivamente na programação dos horários das refeições, sem a necessidade de controle de porção. O objetivo é proporcionar praticidade ao tutor e garantir que o gato receba a alimentação nos momentos programados, contribuindo para uma rotina mais organizada.

MATERIAIS E MÉTODOS

A tabela a seguir apresenta os principais componentes utilizados no desenvolvimento do sistema:

Módulos	Quantidade
Display LCD	1
ESP32	1
Servo Motor	1
Fonte 5V/2A	1
Botão touch	3

O desenvolvimento do projeto seguiu as seguintes etapas:

- Testes Individuais dos Componentes:** Foram realizados testes iniciais dos módulos individuais, como display LCD, servo motor, botões capacitivos e RTC interno do ESP32.
- Desenvolvimento do Hardware:** O circuito foi projetado e montado, incluindo a criação de uma PCB personalizada no software Altium Designer.

3. **Desenvolvimento do Firmware:** Foi implementado o código no ESP32 utilizando Espressif IDE, garantindo a comunicação WiFi e o controle dos periféricos.
4. **Desenvolvimento do Aplicativo:** A interface do aplicativo foi criada no App Inventor 2, permitindo a configuração remota dos horários de alimentação.
5. **Testes e Validação:** O sistema foi testado para verificar a sincronização entre aplicativo e alimentador, bem como a resposta dos componentes eletrônicos.

Testes dos Componentes

Os testes iniciais foram realizados para validar o funcionamento dos módulos antes da integração ao sistema. A imagem a seguir mostra a conexão utilizada para testar o LCD 16x2 com ESP32:

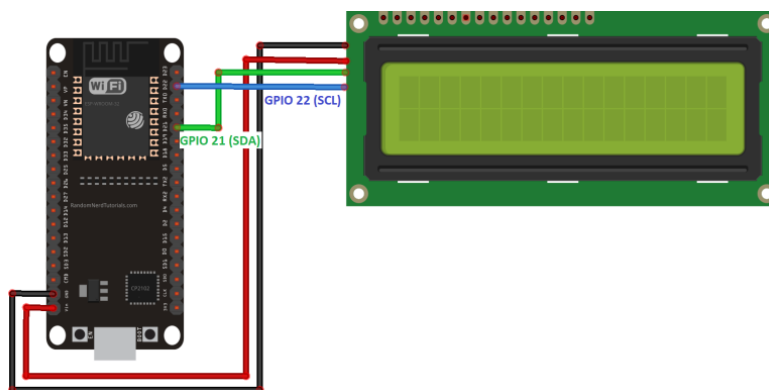


Figura 1 - Esp 32 com LCD

No teste do LCD foi utilizado a EspressifIDE e o módulo I2c. Para teste do RTC, utilizamos a RTC interno do ESP32. A seguir temos a parte do código utilizado com as informações do terminal.

```

35 while (1)
36 {
37     vTaskDelay(pdMS_TO_TICKS(1000)); //Espera 1 seg
38
39
40     time_t tt = time(NULL); //Obtem o tempo atual em segundos.
41     data = "gtime(&tt); //Converte o tempo atual e atribui na estrutura
42
43
44     char data_formatada[64];
45     strftime(data_formatada, 64, "%d/%m/%Y %H:%M:%S", &data); //Cria uma String formatada da estrutura "data"
46
47     printf("\nUnix Time: %ld\n", (long)tt); // Mostra na Serial o Unix time.
48     printf("Data formatada: %s\n", data_formatada); //Mostra na Serial a data formatada
49
50
51     //Fica ligando e desligando o led da placa a cada 1 segundo só para ver que o firmware está rodando
52     if(var == 0 )
53     {
54         printf("LED ligado!\n");
55         gpio_set_level(LED, 1);
56         var=1;
57     }
58     else
59     {
60         printf("LED Desligado!\n");
61         gpio_set_level(LED, 0);
62         var=0;
63     }
64

```

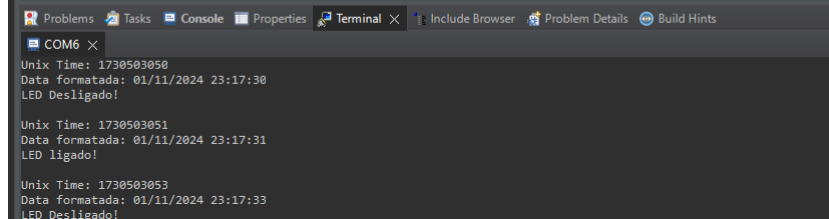


Figura 2 - RTC

Para validar o funcionamento, utilizamos o terminal do próprio Espressif IDE para debugar o código e observar a alteração automática da data. Além disso, habilitamos o LED da placa para ficar piscando e conseguirmos observar que o firmware está rodando.

Iniciamos utilizando o projeto de exemplo existente na própria Espressif Ide, para configurar o servo na Esp Idf é utilizado o periférico LEDC para gerar o sinal PWM. Foi feito o teste em bancada utilizando o ESP32 e o servo motor selecionando anteriormente. No arquivo servo.h temos as definições dos pinos e a escolha de qual timer utilizar.

```

#ifndef MAIN_SERVO_H_
#define MAIN_SERVO_H_

#define SERVO_PIN GPIO_NUM_12 //define o pino GPIO12
#define BT GPIO_NUM_4 //Define o pino GPIO4
#define LEDC_CHANNEL LEDC_CHANNEL_0 //Canal 0 do controlador
#define LEDC_TIMER LEDC_TIMER_0 //define o timer 0
#define LEDC_MODE LEDC_LOW_SPEED_MODE //modo de baixa velocidade
#define LEDC_FREQUENCY 50 // Frequência PWM

int ServoAngle(int angle);

void moveServo(int angle);

void configure_ledc_timer(void);

void configure_ledc_channel(void);

void initServo();

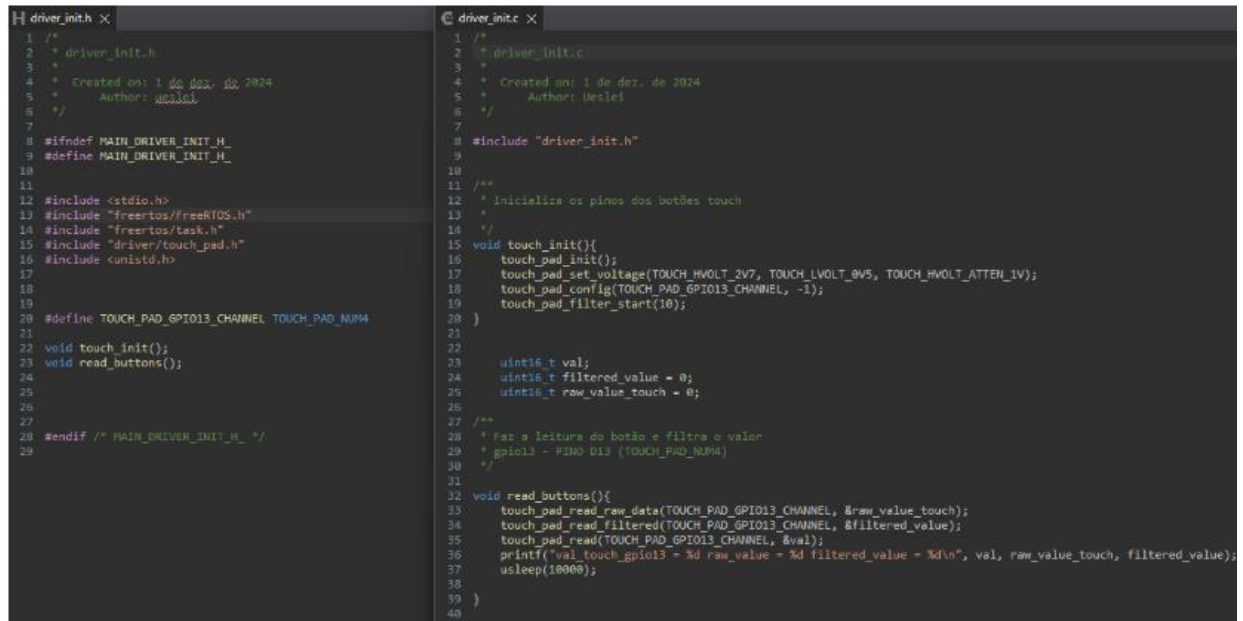
void validaServo();

void touchbt();


```

Figura 3 - TIMER

Para fazer os testes do touch, iniciamos um projeto de exemplo do próprio Espressif Ide chamado "Touch Pad Read".



```
1 /*
2  * driver_init.h
3  *
4  * Created on: 1 de dez. de 2024
5  * Author: deslei
6  */
7
8 #ifndef MAIN_DRIVER_INIT_H_
9 #define MAIN_DRIVER_INIT_H_
10
11
12 #include <stdio.h>
13 #include "freertos/freertos.h"
14 #include "freertos/task.h"
15 #include "driver/touch_pad.h"
16 #include <unistd.h>
17
18 #define TOUCH_PAD_GPIO13_CHANNEL TOUCH_PAD_NUM4
19
20 void touch_init();
21 void read_buttons();
22
23 #endif /* MAIN_DRIVER_INIT_H_ */
```

```
1 /*
2  * driver_init.c
3  *
4  * Created on: 1 de dez. de 2024
5  * Author: deslei
6  */
7
8 #include "driver_init.h"
9
10 /**
11  * Inicializa os pines dos botões touch
12  */
13
14 void touch_init(){
15     touch_pad_init();
16     touch_pad_set_voltage(TOUCH_HVOLT_2V7, TOUCH_LVOLT_0V5, TOUCH_HVOLT_ATTEN_1V);
17     touch_pad_config(TOUCH_PAD_GPIO13_CHANNEL, -1);
18     touch_pad_filter_start(10);
19 }
20
21
22 uint16_t val;
23 uint16_t filtered_value = 0;
24 uint16_t raw_value_touch = 0;
25
26 /**
27  * Faz a leitura do botão e filtra o valor
28  * gpio13 - PINO D13 (TOUCH_PAD_NUM4)
29  */
30
31 void read_buttons(){
32     touch_pad_read_raw_data(TOUCH_PAD_GPIO13_CHANNEL, &raw_value_touch);
33     touch_pad_read_filtered(TOUCH_PAD_GPIO13_CHANNEL, &filtered_value);
34     touch_pad_read(TOUCH_PAD_GPIO13_CHANNEL, &val);
35     printf("val touch_gpio13 = %d raw_value = %d filtered_value = %d\n", val, raw_value_touch, filtered_value);
36     usleep(10000);
37 }
38
39 }
```

Figura 4 - Teste do Touch

Desenvolvimento do Aplicativo

Para configurar as 24 programações de alimentação por dia vai ser desenvolvido um aplicativo com conectividade via WiFi no APP Inventor2. Inicialmente começamos desenvolvendo a interface de botões e a troca de tela no aplicativo, a lógica de horário também foi iniciada. A tela inicial do aplicativo foi desenvolvida, na próxima figura ela é apresentada em:



Figura 5 - Desenvolvimento do Aplicativo

Na próxima imagem temos a figura que mostra uma parte dos blocos do aplicativo desenvolvido. Nessa configuração é realizada a comunicação wifi.

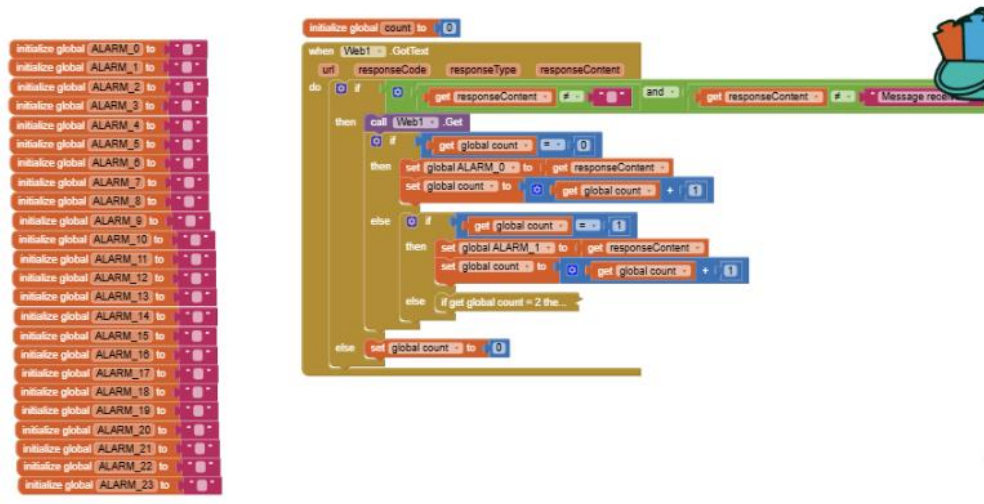


Figura 6 - Desenvolvimento do Aplicativo

Na figura 7 temos a tela inicial do aplicativo desenvolvida, para inicial a operação é necessário clicar no botão conectar. Na figura 8 temos a tela de configuração do aplicativo, nela, temos o botão para sincronizar o horário , voltar, programar alimentação, atualizar alarme na tela e remover os alarmes salvos.



Figura 7 - Tela Principal do Aplicativo

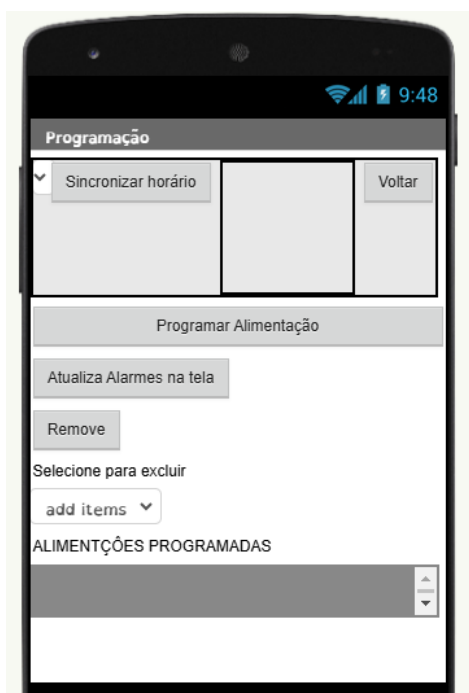


Figura 8 - Tela de Conexão do Aplicativo

Desenvolvimento da PCB

A PCB foi projetada no EasyEDA, contendo 3 botões touch, regulador de tensão, conexão para o servo motor e interface I2C para o LCD.

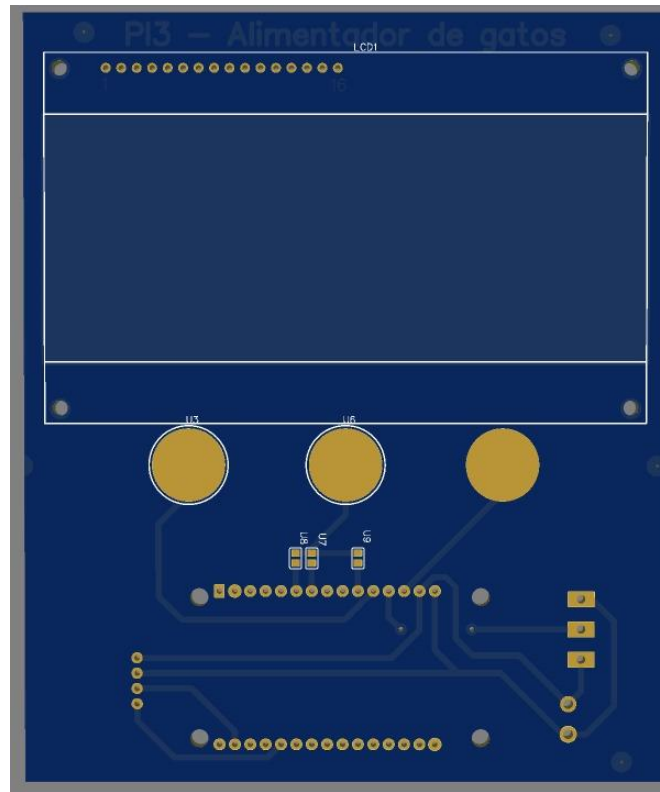


Figura 9 - EasyEDA

Na figura 10 e na figura 11 temos a placa montada e não montada respectivamente, essa placa vai ser fixada na parte superior da estrutura final do alimentador.

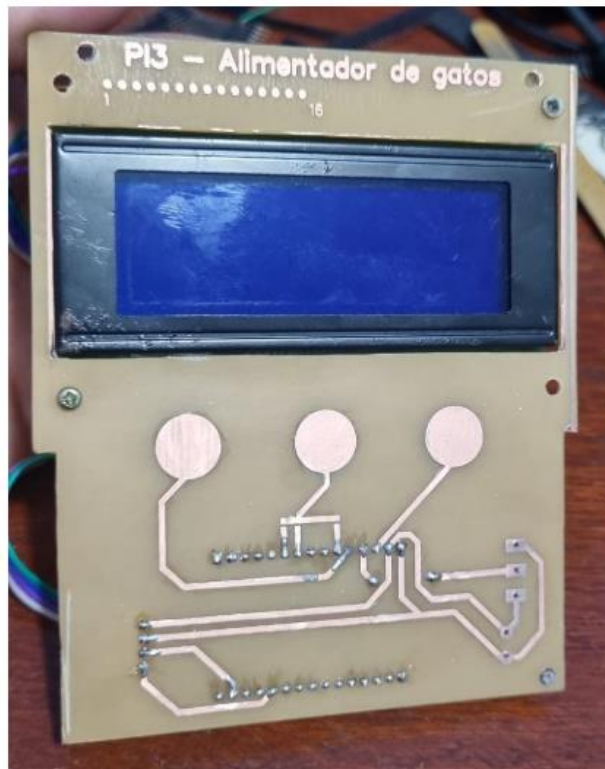


Figura 10 - Placa Desenvolvida

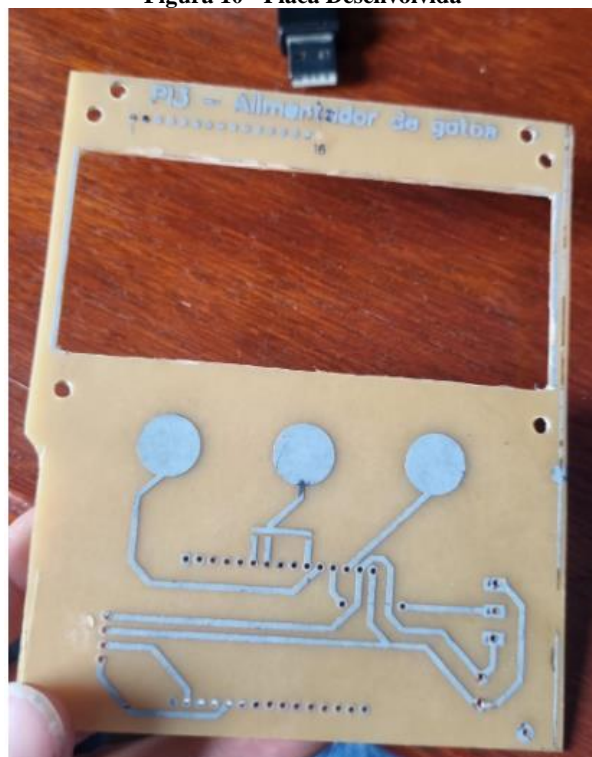


Figura 11 - Placa Desenvolvida

CONCLUSÃO

A implementação de um sistema de alarmes para um alimentador automático de gatos no App Inventor, utilizando uma tabela para exibir horários e doses, permite ao usuário visualizar e gerenciar facilmente as programações de alimentação.

Com a abordagem de armazenar alarmes em uma lista e atualizar dinamicamente a interface, garantimos uma experiência intuitiva e eficiente. Além disso, ao integrar TimePicker e Notifier, o usuário pode definir horários e doses sem complicações.

REFERÊNCIAS

A2ROBOTICS. **Display LCD 16x2**. [S.d]. Disponível em: https://www.a2robotics.com.br/display-lcd-16x2-azul-com-i2c-acoplado?utm_source=Site&utm_medium. Acesso em: 16 mar. 2024.

MAKERHERO. **Servo motor 6221mg**. [S.d]. Disponível em: <https://www.makerhero.com/produto/servo-jx-pdi-6221mg-alto-torque-20kg/>. Acesso em: 16 mar. 2024.

FRANZININHO. **Exemplos ESPIDF**. [S.d]. Disponível em: <https://docs.franzininho.com.br/docs/franzininho-wifi/exemplos-espidef/primeiros-passos>. Acesso em: 21 mar. 2024. <https://randomnerdtutorials.com/esp32-esp8266-i2c-lcd-arduino-ide/>