Alimentador automático para animais de estimação

Relatório Final

Arthur Torres Magalhães – 15/0006063
Universidade de Brasília - UnB
Brasília-DF - Brasil
arthurtorres26@outlook.com

Jovelino Caetano Braz Junior – 14/0043641
Universidade de Brasília - UnB
Brasília-DF - Brasil
jovelinocbjunior@gmail.com

***Resumo* — Este trabalho propõe a criação de um protótipo com o uso de conceitos de sistemas embarcados e do microcontrolador MSP430, para a implementação de um alimentador automático para animais de estimação, com custo muito menor em relação a outros produtos com a mesma finalidade.**

Palavras chaves – MSP430; alimentador automático; animais de estimação.

I. JUSTIFICATIVA

Segundo dados do IBGE em parceria com a Abinpet (Associação Brasileira da Indústria de Produtos para Animais de Estimação), a população total de animais de estimação no Brasil é a 4ª maior do mundo, com mais de 130 milhões de animais. Se forem contados somente a população total de cachorros, gatos e aves canoras e ornamentais (mais de 112 milhões), o país se torna o 2º maior no mundo [1].

Esses animais precisam ser alimentados de tempos em tempos e essa é uma tarefa aparentemente simples para os donos. Entretanto devido a correria e as ocupações do dia-a-dia, os donos desses animais podem acabar esquecendo de alimentá-los, causando um mal tanto nos animais, que ficam sem se alimentar, quanto nos donos que ficam com um mal-estar e sentimento de culpa ao descobrir que esqueceram de fazer algo tão básico. Além do mais, deixar de alimentar animais domésticos é considerado maus tratos com o animal, que por sua vez é crime [2]. Outro ponto a ser observado é que nem todos os animais tem noção do que devem comer, nem dos horários. Eles tendem a comer toda comida da tigela assim que ela é colocada, o que pode trazer problemas de peso e até de ansiedade neles.

Portanto, uma solução para isto está no projeto de um alimentador automatizado de ração para animais de estimação, capaz de encher o recipiente do animal em intervalos programados de tempo e de maneira remota, não permitindo que este animal fique sem acesso à comida caso o dono esqueça ou não esteja em casa no momento (esteja numa viagem, por exemplo).

II. OBJETIVO

Elaboração de um projeto com protótipo de um alimentador automatizado para animais de estimação, que despeja ração em intervalos regulares de tempo e de maneira remota, capaz de avisar ao dono quando o reservatório de ração está perto de ficar vazio.

III. REQUISITOS

A. Disponibilizar ao usuário a possibilidade de determinar o intervalo de tempo em que serão despejadas as rações, assim como a quantidade desejada;

Tal funcionalidade será possível a partir de uma interface, que se comunicará com o MSP430 a partir de um módulo Wi-Fi. O usuário será capaz de determinar um intervalo de tempo (como um timer) onde a ração será despejada.

B. Despejar a quantidade de ração desejada pelo usuário nas horas programadas;

Tal função se torna possível pelo fato de que o tempo em que a "porta" do dispositivo está aberta, ou seja, o tempo em que a ração está sendo despejada, determina a vazão de ração. Consequentemente, ao se determinar o tempo de abertura da "porta", pode-se determinar a quantidade de ração despejada.

C. Disponibilizar ao usuário a possibilidade de acionar o dispositivo através de um botão;

O usuário será capaz de ativar o sistema através de um botão físico no dispositivo, independentemente do tempo programado para o timer.

D. Disponibilizar ao usuário acesso remoto ao dispositivo;

O usuário será capaz de ativar o sistema através de uma interface via Wi-Fi, independentemente do tempo programado para o timer.

E. Disponibilizar ao usuário a informação de que o reservatório está perto de esvaziar;

O sensor ultrassônico medirá a distância do nível de ração para a tampa do reservatório. Com isso, a quantidade de ração no reservatório será medida. Quando esta quantidade atingir um limite crítico (inicialmente foi estabelecido 10% da quantidade), uma notificação é enviada ao usuário via Wi-Fi.

F. Ter uma alimentação de energia que dure dias.

O dispositivo deve ser capaz de se manter ligado por uma quantidade considerável de dias.

IV. TABELA DE MATERIAIS

Tabela 1. Materiais usados no projeto.

Quantidade	*Equipamento*	*Marca*
01	MSP430G2553LP	Texas Instruments
01	Módulo Wi-Fi – ESP8266	Espressif Systems
01	Servo Motor – SG90	Tower Pro
01	Sensor de Distância Ultrassônico HC-SR04	ElecFreaks
-	Jumpers	-
01	Protoboard	Hikari
01	Bateria 9V	-
01	Tabua de madeira	-
01	Cano PVC	-
01	Bocal	-

V. HARDWARE E SOFTWARE

O hardware do projeto engloba quatro pontos principais: a placa MSP430, o servo motor, o sensor ultrassônico e o módulo Wi-Fi. Ou seja, basicamente, envolve um microcontrolador, um sensor, um atuador e um responsável pela comunicação.

A placa MSP430 por ser um dos objetos de estudo da disciplina, será utilizada como um microcontrolador no projeto, ou seja, toda a programação lógica para resolução do problema estará nessa placa. Ela funciona como o "cérebro" do sistema. A versão utilizada será a MSP-EXP430G2553.

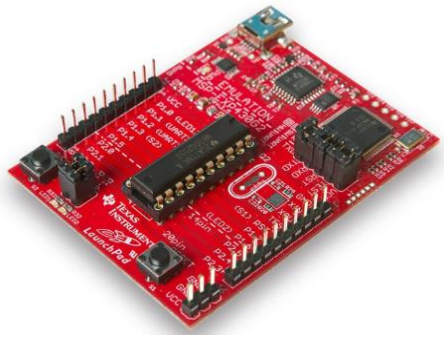

Fig. 1. MSP430G2553 da Texas Instruments.

O servo-motor escolhido para a aplicação foi o SG90. Sua escolha foi devido ao fato desse motor ser capaz de realizar movimentos entre 0º e 180º (suficientes para a realização do projeto) e ter uma programação de fácil entendimento, que permite modificar a angulação e a velocidade do movimento. Além disso, ele tem um torque de 1,2 kg.cm (4,8V) a 1,6 kg.cm (6,0V) e é mais acessível economicamente, custando menos de R$ 20,00. No projeto, ele realiza o movimento responsável por liberar comida do reservatório.

Fig. 2. Micro Servo SG90.

O módulo Wi-Fi por sua vez, é o responsável por fazer a comunicação entre o equipamento com o usuário através da internet, seja para que o usuário defina o timer, seja para que ele receba notificações sobre o estado de risco de falta de ração do reservatório.

Fig. 3. NodeMCU ESP8266.

Já o sensor ultrassônico HC-SR04 será usado no projeto para determinar a quantidade de ração contida dentro do reservatório. Terá como principal função retornar ao MSP340 o valor da distância do nível de ração até a tampa do reservatório e assim o usuário receberá um aviso indicando risco para a falta de ração. Esse aviso será determinado quando a ração no reservatório representar 10% do total do compartimento de ração.

Fig. 4. Sensor HC-SR04.

O software utilizado na integração dos hardwares com o microcontrolador foi o Code Composer. Conseguiu-se fazer o

código do servo motor e do sensor ultrassônico, tal código, assim como os comentários explicando o que cada parte do código faz, pode ser visto no Anexo 1 deste relatório e em [3].

O código do wi-fi é dividido em três arquivos, um .ino, outro .h e um terceiro arquivo .cpp. Os três fazem a comunicação do nosso projeto com o usuário. O arquivo .ino é a main da comunicação, onde dentro deste código se coloca o destinatário e o assunto do e-mail. Também no .ino, se preenche os dados para o ESP-8266 se conectar a uma rede Wi-Fi, no caso preenchendo a SSID e a senha da rede. O arquivo .h é a biblioteca da comunicação, onde também se preenche qual é o e-mail que estará enviando a mensagem.

Quando conectado a uma rede Wi-Fi, o módulo envia um e-mail para o usuário com a seguinte mensagem: "Nível de ração crítico: REABASTEÇA!". Os três códigos pode ser vistos no anexo 2 deste relatório.

O fluxograma inicial, que define basicamente o funcionamento do projeto pode ser visto na Fig. 5 a seguir.

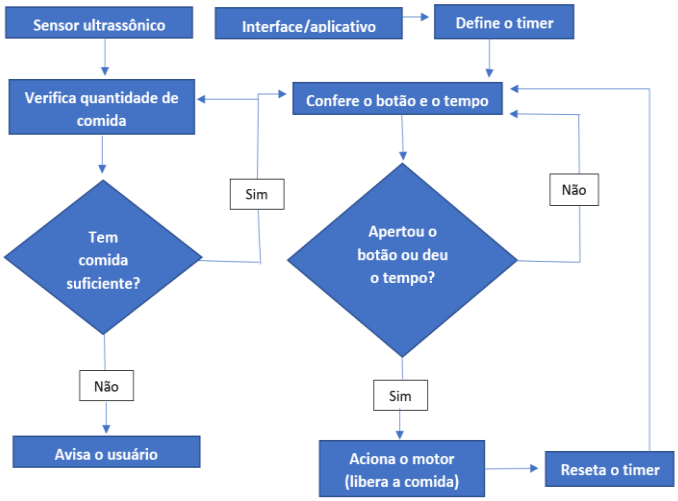

Fig. 5. Fluxograma do projeto.

A montagem do circuito, com as ligações correspondentes, pode ser vista na figura a seguir.

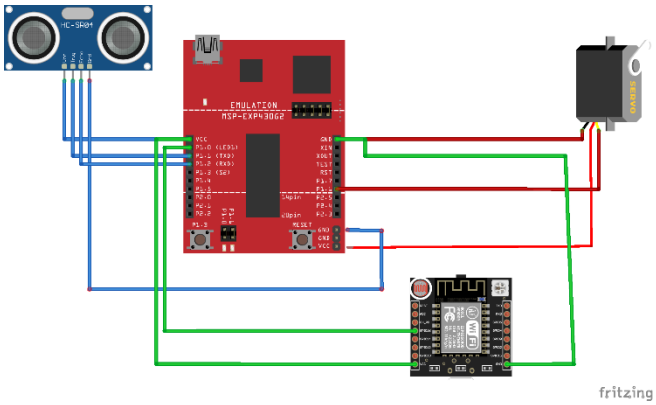

Fig. 6. Montagem do circuito.

As ligações também podem ser vistas na tabela 2 abaixo.

Tabela 2. Ligações entre os compnentes

De: SENSOR ULTRASSONICO	***Para: MSP430***
VCC	VCC
TRIG	P1.1
ECHO	P1.2
GND	GND
De: SERVO	***Para: MSP430***
PWM	P1.6
GND	GND
VCC	5V
De: NODEMCU ESP8266	**Para: MSP430**
D0 (PIN 4)	P1.0

VI. ESTRUTURA

A estrutura foi montada utilizando uma tábua de madeira, uma haste de madeira para o reservatório e o cano PVC para o depósito de ração. A tabua foi desenhada e cortada para dar um design mais bonito e ter uma base bem firme e rente ao chão, para não ter queda do reservatório para nenhum dos lados. A haste de madeira foi colada e pregada com a base para com o uso de arame ser presa ou depósito de ração. Para controlar a vazão de ração pelo bocal, foi utilizado duas buchas de cano. Ainda possui uma caixa para guardar o circuito, para que este não fique exposto ao animal.

Fig. 7. Estrutura do projeto

Outro ponto a se observar é em relação a tampa, onde o sensor é preso, que mede o nível de ração no reservatório tendo como referência, outra tampa, que acompanha a ração, conforme pode ser visto na figura x a seguir.

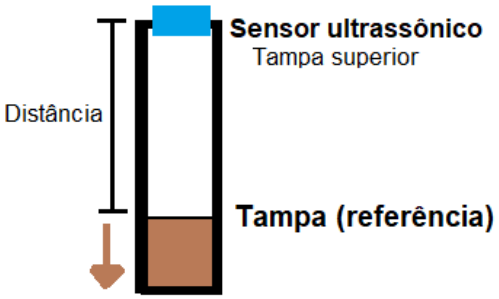

Fig. 8. Funcionamento da medição do reservatório

É importante frisar que toda a construção da estrutura, foi feita com materiais recicláveis, de baixíssimo custo. Tendo em vista o escopo da disciplina.

O reservatório, feito com cano PVC, comporta um volume de aproximadamente 0.002m^3, ou seja, 2L. Tal volume corresponde a aproximadamente 1.5kg. Tendo em vista que um cachorro de pequeno porte, come por dia, em torno de 100 a 150g, e que cada porção (cada acionamento do servo), despeja de 50 a 55g de ração [7], temos então que o cachorro de pequeno porte precisa de 3 porções por dia para se alimentar, o que por sua vez representa que o reservatório tem autonomia de aproximadamente 10 dias, se estiver cheio.

VII. Benefícios

O uso de um alimentador automático beneficia pessoas com animais de estimação que não ficam muito tempo em casa, que estão constantemente viajando ou até mesmo aquelas pessoas que constantemente tem imprevistos que as impedem de alimentar seus animais na hora correta.

Também é muito útil para pessoas que tem animais com problemas alimentares (animais com sobrepeso, diabéticos...) e que precisam ser alimentados em intervalos certos de tempo, com quantidades controladas.

Além disso, um alimentador automático permite que o dono possa definir as quantidades e horários em que a ração será despejada, o que ajudaria na criação de uma rotina mais saudável para os animais. Ao fazer isso, o dono além de ter mais comodidade ainda pode economizar tempo.

VIII. Revisão Bibliográfica

Atualmente, existem vários modelos de alimentadores automáticos para animais de estimação no mercado. Eles podem ser feitos especificamente para cães, gatos, pássaros ou até mesmo para peixes. Alguns desses modelos oferecem muitas funcionalidades, o que pode elevar bastante o preço. Dois exemplos desses modelos são:

A. *Alimentador eletronônico Hoison: babá robô para pet [4]*

1) Funcionalidades:

a) Alimentador programável por aplicativo;

b) Comedouro removível e lavável;

c) Capacidade de até 2kg de ração;

d) Liberação de ração

- Automaticamente nos horários programados no aplicativo;
- Remotamente por app, sem necessidade de programação;
- Manualmente, pressionando o botão no topo do aparelho.

e) Necessária rede Wi-fi para estabelcer conexão com smartphone.

2) Custo: R$ 1.349,90.

IX.

A. *Alimentador Eletrônico Eatwell Plus 5 refeições – Amicus [5]*

1) Funcionalidades:

a) Evita a exposição prolongada de alimentos no ambiente;

b) Compartimentos de comida removível que facilita a higienização;

c) Até cinco refeições programadas por hora;

2) Custo: R$ 549,90.

Também existem projetos de alimentadores eletrônicos que são construídos de forma mais simples, porém com custo menor de fabricação e que viabilizam o mesmo objetivo dos demais. Não necessariamente estes projetos são comercializados. Um exemplo disso é o projeto feito através do uso do Arduino [6], que assim como diversos projetos encontrados, se enquadra melhor no escopo da disciplina.

X. Resultados

Desde o último ponto de controle, houveram avanços no projeto. A estrutura foi aprimorada, acrescentando-se uma caixa para comportar o circuito, evitando que animais mexam com o microcontrolador. Também foi acrescentado o suporte para o servo motor, além de outros componentes da estrutura importantes para o funcionamento do projeto, como as tampas (onde fica fixado o sensor ultrassônico e a segunda tampa, referência de distância do sensor, que desce junto com a ração).

Já em relação ao código, conseguiu-se realizar a comunicação wi-fi parcialmente. Foi possível enviar o email para o usuário, a partir do nodeMCU-ESP8266, na qual quando o sensor detectava nível crítico de ração, setava uma flag que aciona o envio do e-mail.

O controle de despejo de ração é feito através do código do microcontrolador. A ideia de disponibilizar o controle de intervalo de alimentar o Pet, através de um aplicativo, não foi concluída devido as dificuldades encontras. Porém a cada abertura para a vazão de ração, se despeja entre 50g a 55g. O que possibilita ao usuário controlar quantas vezes no dia ele deseja acionar o mecanismo, dependendo do porte de seu animal.

Um dos requisitos propostos inciicialmente, de disponibilizar ao usuário controle remoto, não foi concluído

com sucesso, devidoa dificuldades na implementação do código.

Em comparação aos produtos existentes no mercado, como proposto desde o início, nosso projeto custou menos de 75 reais. Lembrando que apesar de não possuir um design dos produtos existentes, ressaltamos que o projeto representa uma economia de cerca de 1200 reais.

O projeto ainda pode ser otimizado. Pode-se implementar o uso através do celular, controlando o despejo de ração, ter o nível em tempo real da quantidade de ração disponível no reservatório ou até mesmo disponibilizar ao usuário a quantidade de ração que ele quer que seja despejada.

Sendo assim, conclui-se que o projeto obteve relativo sucesso, apesar de ainda haver algumas funcionalidades a serem ainda implementadas corretamente. Percebeu-se que o projeto é viável e de baixo custo, podendo ser reduzido ainda mais.

REFERÊNCIAS

[1] IBGE. População de animais de estimação no Brasil – 2013. Disponível em:http://www.agricultura.gov.br/assuntos/camaras-setoriais-tematicas/documentos/camaras-tematicas/insumos-agropecuarios/anos-anteriores/ibge-populacao-de-animais-de-estimacao-no-brasil-2013-abinpet-79.pdf. Acesso em 4 de setembro de 2018.

[2] Brasil. Lei nº 9.605 de 12 de Fevereiro de 1998. Disponível em: http://www.planalto.gov.br/ccivil_03/LEIS/L9605.htm. Acessado em 4 de setembro de 2018.

[3] Github. Disponível: https://github.com/arthurtorrs/Eletronica-Embarcada/tree/master/2_PCs/Codigos

[4] Petlove. Disponível em: https://www.petlove.com.br/hoison?destaque=alimentador-eletronico-hoison-baba-robo-para-pet&sku=1571337&gclid=CjwKCAjw2rjcBRBuEiwAheKeL8s7IJ7LQdUQnehyGO5Im-zucs7HsaKT1QRsox7gYqfsjv2UOH1r8BoCNK0QAvD_BwE.Acesso em 4 de setembro de 2018.

[5] Petlove. Disponível em: https://www.petlove.com.br/amicus?destaque=alimentador-eletronico-eatwell-plus-5-refeicoes---amicus-3102969&sku=3102969. Acesso em 4 de setembro de 2018.

[6] Arduino Project Hub. Disponível em: https://create.arduino.cc/projecthub/circuito-io-team/iot-pet-feeder-10a4f3?ref=tag&ref_id=pets&offset=0. Acesso em 3 de setembro de 2018.

[7] Canal do Pet. Disponível em: https://canaldopet.ig.com.br/cuidados/comidas/2017-07-25/racao-para-cachorro.html. Acesso em 28 de novembro de 2018.

Anexo 1

Código

```
#include <msp430.h>
int k;
int miliseconds;
int distance;
long sensor;

void delay(volatile unsigned long i)
{
   do (i--);                                        //Loop de delay
   while (i != 0);
}

int main(void)
{
  BCSCTL1 = CALBC1_1MHZ;
 DCOCTL = CALDCO_1MHZ;                  // Clock de 1Mhz
 WDTCTL = WDTPW + WDTHOLD;              // Desliga o WatchdogTimer
 CCTL0 = CCIE;                          // Habilita a interrupção pelo CCR0
 CCR0 = 1000;                       // 1ms em 1mhz
 TACTL = TASSEL_2 + MC_1;               // SMCLK, upmode
 P1IFG  = 0x00;                     // Limpa todas as flagas de interrupçao
 P1DIR |= 0x01;                         // P1.0 como saida do LED
 P1OUT &= ~0x01;                        // Desliga o LED
 _BIS_SR(GIE);                          // Habilita a interrupcao global

 while (1)
 {
     P1IE &= ~0x01;                 // Desabilita a interrupcao
     P1DIR |= 0x02;                 // Pino do TRIGGER (P1.1) como saida
     P1OUT |= 0x02;                 // Gera o pulso
     __delay_cycles(10);            // Delay de 10us
     P1OUT &= ~0x02;                // Para o pulso
     P1DIR &= ~0x04;                // Pino do ECHO (P1.2) como entrada
     P1IFG = 0x00;                  // Limpa a flag, por precaução
     P1IE |= 0x04;                  // Habilita a interrupção pelo pino do ECHO
     P1IES &= ~0x04;                // Define a borda de subida no pino ECHO
     __delay_cycles(30000);         // Delay de 30ms (depois desse tempo, o ECHO acaba se nenhum objeto for detectado)
     distance = sensor / 58;        // Converting o valor de ECHO para cm

     P1DIR |= BIT6;                     // P1.6/TA0.1 É USADO PARA O PWM, QUE FUNCIONA COMO A SAÍDA -> servo 1
     P1OUT = 0;                         // LIMPA AS SAIDAS P1
     P1SEL |= BIT6;                     // P1.6 SELECIONA TA0.1

    P1DIR &= ~BIT3;
     P1REN |= BIT3;
     P1OUT |= BIT3;

     // COMO O CLOCK É DE 1MHz, E 1000ms EQUIVALEM A 1Hz, ENTAO SETANDO CCR0 PARA 20000
     // TEMOS QUE 20000(1000000 / 1000 * 20) É UM PERIODO DE 20ms

     TA0CCR0 = 20000-1;                 // PERIODO DO PWM TA0.1
     TA1CCR0 = 20000-1;                 // PERIODO DO PWM TA1.1
```

```
      // SETANDO 2000 -> 0deg (POS SERVO)
      TA0CCR1 = 2000;                   // CCR1 PWM duty cycle
      TA1CCR1 = 2000;                   // CCR1 PWM duty cycle

      TA0CCTL1 = OUTMOD_7;              // CCR1 reset/set
      TA0CTL   = TASSEL_2 + MC_1;       // SMCLK, up mode
      TA1CCTL1 = OUTMOD_7;              // CCR1 reset/set
      TA1CTL   = TASSEL_2 + MC_1;       // SMCLK, up mode

   if (distance > 30 && distance != 0) {
    P1OUT |= 0x01;          // Se a distancia for menor que 20cm e diferente de 0, o LED acende
   }
   else {
    P1OUT &= ~0x01;         // Caso contrario, ele permanece desligado.
   }

   if((P1IN&BIT3)==0)
      {
         for(k=0;k<3;k++){
         liga_servo();                  //Chama a função pra ligar o servo
         __delay_cycles(10000000);      //Tempo de espera entre as porções
         }
      }
  }
}
void liga_servo(){
      delay(30000);
         TA0CCR1 = 1000;
         TA1CCR1 = 2000;                //Abre a escotilha
      delay(30000);
         TA0CCR1 = 2000;
         TA1CCR1 = 1000;                //Fecha a escotilha
    }
#pragma vector=PORT1_VECTOR
__interrupt void Port_1(void)
{
  if (P1IFG & 0x04)          // Verifica se tem alguma interrupcao pendente
  {
   if (!(P1IES & 0x04))      // Verifica se tem uma borda de subida
   {
    TACTL |= TACLR;          // Se tem, limpa o timer A
    miliseconds = 0;
    P1IES |= 0x04;          // Borda de descida
   }
   else
   {
    sensor = (long)miliseconds * 1000 + (long)TAR;    // Calcula o comprimento de ECHO
   }
   P1IFG &= ~0x04;                  // Limpa a flag
  }
}
#pragma vector=TIMER0_A0_VECTOR
__interrupt void Timer_A (void)
{
  miliseconds++;
}
```

Anexo 2
Código do Wi-Fi

Main.ino:

```
#include <ESP8266WiFi.h>
#include "Gsender.h"

#pragma region Globals
const char* ssid = "Iphone";                        // WIFI network name
const char* password = "12345678";                  // WIFI network password
uint8_t connection_state = 0;                    // Connected to WIFI or not
uint16_t reconnect_interval = 10000;             // If not connected wait time to try again
#pragma endregion Globals


uint8_t WiFiConnect(const char* nSSID = nullptr, const char* nPassword = nullptr)
{
   static uint16_t attempt = 0;
   Serial.print("Connecting to ");
   if(nSSID) {
      WiFi.begin(nSSID, nPassword);
      Serial.println(nSSID);
   } else {
      WiFi.begin(ssid, password);
      Serial.println(ssid);
   }

   uint8_t i = 0;
   while(WiFi.status()!= WL_CONNECTED && i++ < 50)
   {
      delay(200);
      Serial.print(".");
   }
   ++attempt;
   Serial.println("");
   if(i == 51) {
      Serial.print("Connection: TIMEOUT on attempt: ");
      Serial.println(attempt);
      if(attempt % 2 == 0)
         Serial.println("Check if access point available or SSID and Password\r\n");
      return false;
   }
   Serial.println("Connection: ESTABLISHED");
   Serial.print("Got IP address: ");
   Serial.println(WiFi.localIP());
   return true;
}

void Awaits()
{
   uint32_t ts = millis();
   while(!connection_state)
   {
```

```
        delay(50);
        if(millis() > (ts + reconnect_interval) && !connection_state){
            connection_state = WiFiConnect();
            ts = millis();
        }
    }
}

void setup()
{
    Serial.begin(115200);
    connection_state = WiFiConnect();
    if(!connection_state)  // if not connected to WIFI
        Awaits();          // constantly trying to connect

    Gsender *gsender = Gsender::Instance();    // Getting pointer to class instance
    String subject = "Racao do Dogao";
    if(gsender->Subject(subject)->Send("diogogarcia@unb.br", "Nível de ração baixa: REABASTEÇA!")) {
        Serial.println("Message send.");
    } else {
        Serial.print("Error sending message: ");
        Serial.println(gsender->getError());
    }
}

void loop(){}
```

gsender.h:

```
/* Gsender class helps send e-mails from Gmail account
*  using Arduino core for ESP8266 WiFi chip
*  by Boris Shobat
*  September 29 2016
*/
#ifndef G_SENDER
#define G_SENDER
#define GS_SERIAL_LOG_1        // Print to Serial only server responce
//#define GS_SERIAL_LOG_2      //  Print to Serial client commands and server responce
#include <WiFiClientSecure.h>

class Gsender
{
    protected:
        Gsender();
    private:
        const int SMTP_PORT = 465;
        const char* SMTP_SERVER = "smtp.gmail.com";
        const char* EMAILBASE64_LOGIN = "am92ZWxpbm9jYmp1bmlvckBnbWFpbC5jb20=";
        const char* EMAILBASE64_PASSWORD = "Z2Fsb3BhdG8xMTQ0Nzc=";
        const char* FROM = "jovelinocbjunior@gmail.com";
        const char* _error = nullptr;
        char* _subject = nullptr;
        String _serverResponce;
        static Gsender* _instance;
        bool AwaitSMTPResponse(WiFiClientSecure &client, const String &resp = "", uint16_t timeOut = 10000);
```

```
    public:
        static Gsender* Instance();
        Gsender* Subject(const char* subject);
        Gsender* Subject(const String &subject);
        bool Send(const String &to, const String &message);
        String getLastResponce();
        const char* getError();
};
#endif // G_SENDER
```

Gsender.cpp:

```
#include "Gsender.h"
Gsender* Gsender::_instance = 0;
Gsender::Gsender(){}
Gsender* Gsender::Instance()
{
    if (_instance == 0)
        _instance = new Gsender;
    return _instance;
}

Gsender* Gsender::Subject(const char* subject)
{
  delete [] _subject;
  _subject = new char[strlen(subject)+1];
  strcpy(_subject, subject);
  return _instance;
}
Gsender* Gsender::Subject(const String &subject)
{
  return Subject(subject.c_str());
}

bool Gsender::AwaitSMTPResponse(WiFiClientSecure &client, const String &resp, uint16_t timeOut)
{
  uint32_t ts = millis();
  while (!client.available())
  {
    if(millis() > (ts + timeOut)) {
      _error = "SMTP Response TIMEOUT!";
      return false;
    }
  }
  _serverResponce = client.readStringUntil('\n');
#if defined(GS_SERIAL_LOG_1) || defined(GS_SERIAL_LOG_2)
  Serial.println(_serverResponce);
#endif
  if (resp && _serverResponce.indexOf(resp) == -1) return false;
  return true;
}

String Gsender::getLastResponce()
{
  return _serverResponce;
}
```

```
const char* Gsender::getError()
{
  return _error;
}

bool Gsender::Send(const String &to, const String &message)
{
  WiFiClientSecure client;
#if defined(GS_SERIAL_LOG_2)
  Serial.print("Connecting to :");
  Serial.println(SMTP_SERVER);
#endif
  if(!client.connect(SMTP_SERVER, SMTP_PORT)) {
    _error = "Could not connect to mail server";
    return false;
  }
  if(!AwaitSMTPResponse(client, "220")) {
    _error = "Connection Error";
    return false;
  }

#if defined(GS_SERIAL_LOG_2)
  Serial.println("HELO friend:");
#endif
  client.println("HELO friend");
  if(!AwaitSMTPResponse(client, "250")){
    _error = "identification error";
    return false;
  }

#if defined(GS_SERIAL_LOG_2)
  Serial.println("AUTH LOGIN:");
#endif
  client.println("AUTH LOGIN");
  AwaitSMTPResponse(client);

#if defined(GS_SERIAL_LOG_2)
  Serial.println("EMAILBASE64_LOGIN:");
#endif
  client.println(EMAILBASE64_LOGIN);
  AwaitSMTPResponse(client);

#if defined(GS_SERIAL_LOG_2)
  Serial.println("EMAILBASE64_PASSWORD:");
#endif
  client.println(EMAILBASE64_PASSWORD);
  if (!AwaitSMTPResponse(client, "235")) {
    _error = "SMTP AUTH error";
    return false;
  }

  String mailFrom = "MAIL FROM: <" + String(FROM) + '>';
#if defined(GS_SERIAL_LOG_2)
  Serial.println(mailFrom);
#endif
  client.println(mailFrom);
```

```
  AwaitSMTPResponse(client);

  String rcpt = "RCPT TO: <" + to + '>';
#if defined(GS_SERIAL_LOG_2)
  Serial.println(rcpt);
#endif
  client.println(rcpt);
  AwaitSMTPResponse(client);

#if defined(GS_SERIAL_LOG_2)
  Serial.println("DATA:");
#endif
  client.println("DATA");
  if(!AwaitSMTPResponse(client, "354")) {
    _error = "SMTP DATA error";
    return false;
  }

  client.println("From: <" + String(FROM) + '>');
  client.println("To: <" + to + '>');

  client.print("Subject: ");
  client.println(_subject);

  client.println("Mime-Version: 1.0");
  client.println("Content-Type: text/html; charset=\"UTF-8\"");
  client.println("Content-Transfer-Encoding: 7bit");
  client.println();
  String body = "<!DOCTYPE html><html lang=\"en\">" + message + "</html>";
  client.println(body);
  client.println(".");
  if (!AwaitSMTPResponse(client, "250")) {
    _error = "Sending message error";
    return false;
  }
  client.println("QUIT");
  if (!AwaitSMTPResponse(client, "221")) {
    _error = "SMTP QUIT error";
    return false;
  }
  return true;
}
```