UNIVERSIDADE FEDERAL DE OURO PRETO
ESCOLA DE MINAS
COLEGIADO DO CURSO DE ENGENHARIA DE CONTROLE E AUTOMAÇÃO - CECAU

Moises de Morais Henriques

ALIMENTADOR AUTOMÁTICO COM SUPERVISÓRIO PARA ANIMAIS DE PEQUENO PORTE

MONOGRAFIA DE GRADUAÇÃO EM ENGENHARIA

DE CONTROLE EAUTOMAÇÃO

Ouro Preto, 2015

Moises de Morais Henriques

ALIMENTADOR AUTOMÁTICO COM SUPERVISÓRIO PARA ANIMAIS DE PEQUENO PORTE

Monografia apresentada ao Curso de Engenharia de Controle e Automação da Universidade Federal de Ouro Preto como parte dos requisitos para obtenção do Grau de Engenheiro de Controle e Automação.

Orientador: Adrielle de Carvalho Santana

Ouro Preto

Escola de Minas – UFOP

Fevereiro/2015

AGRADECIMENTOS

Gostaria de agradecer primeiramente a minha orientadora Adrielle de Carvalho Santana, que mesmo com a distância e com as dificuldades me auxiliou, apoiou e deu forças para que eu conseguisse concluir este trabalho. Ao professor Jose Álvaro Tadeu por todo conhecimento intelectual e pessoal passado durante minha graduação. A minha família e namorada pelo apoio incondicional. As republicas Tabajara e Suavizinha por serem meu lar e me acolherem tão bem durante todo esse tempo em Ouro Preto, e por fim a todos os professores da Escola de Minas e da UFOP que fizeram parte da minha jornada ate aqui, pois sem o apoio e ensinamento dos mesmos eu não estaria finalizando mais esta etapa da minha vida.

“ Nenhuma grande vitória é possível sem que tenha sido precedida de pequenas vitórias sobre nós mesmos. ”

(L. M. Leonov.)

RESUMO

Em um mundo de altas exigências e grande competitividade onde tempo significa dinheiro, muitas vezes falta tempo para obrigações domésticas e pessoais básicas, como por exemplo, o simples fato de alimentar um animal de estimação. É sobre este contexto que a automação começa a ser utilizada de forma a facilitar as tarefas do dia a dia e otimizar a utilização do tempo disponível das pessoas. Com base neste cenário, predominante em grande parte dos lares na atualidade, este trabalho foi idealizado. Seu objetivo é propor uma solução simples, eficaz e viável que seja responsável por automatizar o processo de alimentar um animal de estimação, bem como monitorar o estoque de ração disponível exibindo um alerta no supervisório quando a mesma estiver próxima a acabar. Para tanto foi utilizado um microcontrolador da plataforma Arduino, mais especificamente o Arduino Yun, que possui um módulo wifi integrado, possibilitando dessa forma a comunicação wireless entre microcontrolador e usuário e o envio de alertas via e-mail. Toda a lógica de controle foi desenvolvida utilizando a IDE do Arduino para desenvolver a programação responsável por comandar sensores e o servo de rotação contínua o qual realiza o acionamento de uma válvula borboleta, responsável por liberar as porções de ração para a vasilha de alimentação do animal. Esse trabalho foi desenvolvido para o ambiente domiciliar, porém podendo facilmente ser implementado em grande escala para ambientes industriais como criações de animais destinados a abate.

Palavras Chave: Alimentador, animal, microcontrolador, Arduino, Wifi, Sensores, Servo Motor.

ABSTRACT

In a world with high demands and great competition where time many times mean money, for sometimes there is a lack of time to complete basic domestic and personal obligations, as the simple act of feed a pet for example. In this context the automation usage starts to be done looking for turn more easy and simple the daily obligations and optimize the way people use their time. Based on this scenario, which is predominant in great part of homes nowadays, this work was idealized. Its main purpose is offer a simple, efficient and reliable solution responsible for automatize the pet feeder process, also it will provide a way to track the remaining pet food supply and send alerts via e-mail when it is near the end. To do so an Arduino Yun microcontroller which has a built-in wifi shield, which allows us to set up a wireless communication between microcontroller and user and also send supply alerts through e-mail. All the logic part was developed using Arduino's IDE for programming the code responsible for control sensors and a continuous rotation servo motor responsible for act the butterfly valve which dispose the food in the animals feed bowl. This work was developed for domestic usage, but it surely can be expanded for industrial usage as in the creation of animals for slaughter.

Key words: Pet feeder, animal, microcontrollers, Arduino, Wifi, Sensors, Servo Motor.

LISTA DE FIGURAS

Figura 2.1 - Diagrama de Blocos de um microcontrolador. 14

Figura 2.2 - Diagrama de blocos do funcionamento da arquitetura Havard. 15

Figura 2.3 - Diagrama de blocos do funcionamento da arquitetura Von Neumann. ..16

Figura 2.4 - Diagrama de blocos do funcionamento dos microcontroladores da família AVR. 17

Figura 3.1 - Sensor Ultrassônico HC-SR04. 21

Figura 3.2 – Controle de um servo motor. 22

Figura 3.3 - Servo motor de rotação continua (SM-S4303R) 22

Figura 3.4 - Arquitetura do arduino Yun. 23

Figura 3.5 - Arduino Yun. 24

Figura 3.6 - Modulo RTC. 25

Figura 4.1 - Teste e aferimento do sensor ultrassónico. 26

Figura 4.2 - Teste dos LEDs. 27

Figura 4.3 - Teste e aferimento do servo motor. 27

Figura 4.4 - (A) Servo motor acoplado a válvula borboleta, (B) Sensor ultrassónico de nível do recipiente de alimentação, (C) Sensor ultrassónico de nível do reservatório de ração e (D) Dispenser em acrílico que funciona como reservatório do sistema. ..29

Figura 4.5 - (A) Led de alerta de nível de ração baixo no reservatório, (B) Fonte de alimentação do sistema, (C) Placa arduino e (D) protoboard com RTC acoplado. ...30

Figura 4.6 – Esquema do circuito elétrico do protótipo. 31

Figura 4.7 - Configuração do *Data Source* LED. 34

Figura 4.8 - Configuração do *Data Point* LED. 34

Figura 4.9 - Configuração final do graphic view do supervisório. 35

Figura 4.10 – (A) Servo parado e (B) Servo em funcionamento.....36

Figura 4.11 - Mensagem de programação feita com sucesso.....37

Figura 4.12 - Erro gerado ao entrar com um valor invalido para o horário. 37

LISTA DE ABREVIATURAS

IDE – *Integrated Development Enviroment*

MC – Microcontrolador

Wifi – *Wireless Fidelity*

CA – Corrente Alternada

CC- Corrente Continua

CPU – *Central Processing Unit*

ABINPET - Associação Brasileira de Produtos para Animais de Estimação

I/O – *Input / Output*

RISC – *Reduced Instruction Set Computer*

CISC – *Complex Instruction Set Computer*

SCADA – *Supervisory Control and Data Acquisition*

CLP – Computador Logico Programável

DC – *Direct current* (corrente continua)

RTC – *Real Time Clock*

I2C – *Inter Integrated Cicuit*

REST – *Representational State Transfer*

Regex – *Regular Expression*

SUMÁRIO

1. Introdução 12
1.1 Considerações iniciais 12
1.2 Objetivos 13
1.3 Justificativa 13
1.4 Metodologia 13
1.5 Estrutura do Trabalho 13
2. Base teórica bibliográfica 14
2.1. Microcontroladores 14
2.1.1. Famílias de microcontroladores 15
2.2. Sistemas supervisórios 17
2.3. Protocolos de comunicação 18
3. Componentes do protótipo 20
3.1. Sensores e atuadores 20
3.1.1. Sensor ultrassônico (HC-SR04) 20
3.1.2. Servo motor de rotação continua (SM-S4303R) 21
3.2. Microcontrolador 23
3.2.1. Características da plataforma Arduino Yun 23
3.3. RTC 24
4. Desenvolvimento e resultados 26
4.1. Aferimento e testes 26
4.1.1. Sensores ultrassônicos 26
4.1.2. LEDs 27
4.1.3. Servo Motor 27
4.2. Hardware 28
4.2.1. Circuito 30
4.3. Software 32

4.3.1. Arduino 32
4.3.2. Sistema supervisório 33
4.4. Funcionamento 35
5. Conclusão e Trabalhos Futuros 39
6. Referencias Bibliográficas.....41
Apêndice A - Código para o Arduino. 44
Apêndice B – Manual de inicialização do protótipo. 56

1. Introdução

1.1 Considerações iniciais

Os constantes avanços tecnológicos propiciam a criação de sistemas e aplicações visando à melhoria da qualidade de vida das pessoas. Estes produtos e sistemas estão hoje presentes em diversas áreas e aplicações diárias, se integrando tão bem ao ambiente como um todo que muitas vezes sua presença não é nem mesmo notada. Este é um mercado que sempre dirigiu e atraiu a atenção das pessoas, que por muitas vezes são guiadas pelas novidades e facilidades propiciadas como solução às correrias da vida cotidiana.

Segundo ABINPET (2012), o Brasil conta com mais de 102 milhões de animais de estimação, o que movimenta um montante de cerca de 15 bilhões de reais anualmente. A mesma ainda aponta outro fator que demonstra o imenso potencial do setor pet no Brasil sendo este o crescimento da quantidade de animais de estimação no país, que soma uma taxa de 5% de crescimento ao ano.

Estes são dois mercados promissores e com grandes potenciais a serem explorados. O que será discutido nesse trabalho se baseia nas projeções demonstradas pela ABINPET para o crescimento do setor, e bem como a necessidade de muitas vezes os donos se afastem de casa por um período prolongado de tempo, o que sempre levanta o receio com relação aos cuidados com o animal de estimação (CARRIDE, 2008). Assim, este trabalho se propõe a tentar utilizar sistemas embutidos para desenvolver um alimentador (*pet feeder*) para cuidado de animais de estimação com monitoramento remoto.

Existem outros trabalhos desenvolvidos que abordam a mesma temática. Todos concluem ressaltando as vantagens de um sistema como esse sobre sistemas tradicionais, como por exemplo a facilidade se poder definir os horários que gostaria que as porções de ração fossem servidas, não ficando preso a um determinado horário (OCHAKOWSKI, 2007) e a garantia que tal equipamento supriria a necessidade alimentar de cães e gatos de forma confiável (CARRIDE, 2008).Tais características juntamente com as projeções feitas pela ABINPET constituem um grande agente motivador para o desenvolvimento do protótipo proposto.

1.2 Objetivos

Elaborar um protótipo capaz de colocar ração para um animal de estimação em um determinado horário pré-programado do dia. Este protótipo deve contar ainda com meios para monitoramento remoto da quantidade de ração disponível e de envio de alertas no supervisório quando a mesma se aproximar do fim.

1.3 Justificativa

Ambos os mercados, pet e de eletrônicos, se encontram aquecidos no momento e com boas projeções futuras. Além disso, as preocupações com os animais de estimação durante viagens fazem parte da realidade vivida por grande parte dos lares brasileiros, levando à necessidade de uma solução tal como a proposta neste trabalho, bem como constituindo um alto grau de aceitação para o protótipo.

1.4 Metodologia

O primeiro passo para o desenvolvimento do sistema foi definir suas funcionalidades e com base nestas, foram definidos os materiais que seriam necessários para a sua implementação, começando pela relação de sensores e atuadores necessários. Então, com base no número de I/O necessários e os requisitos de comunicação, foi escolhido a plataforma Arduino Yun como sendo a melhor placa microcontrolada pois atendia a todos os requisitos, além do custo-benefício, facilidades de implementação e por possuir IDE livre e com suporte de várias bibliotecas que dão suporte aos módulos e sensores. O microcontrolador selecionado conta ainda com um servidor interno que será utilizado para montar um sistema supervisório para a aplicação desejada.

1.5 Estrutura do Trabalho

No primeiro capítulo é feita uma breve introdução a respeito do trabalho em si, os objetivos, justificativa e metodologia também são discutidos neste capítulo. Seguindo então para uma revisão bibliográfica a respeito de microcontroladores e sistemas supervisórios no segundo capítulo. No terceiro capítulo todos os componentes eletrônicos utilizados são apresentados e têm seus princípios de funcionamento brevemente explicados, após isso o capítulo quatro é responsável por explicar o funcionamento do projeto e programação do *pet feeder* e discutir todas as suas funcionalidades. E no quinto e último capítulo são discutidos os resultados e considerações finais a respeito do projeto.

2. Base teórica bibliográfica

Neste capitulo serão abordados tópicos específicos dos componentes do trabalho e funcionamento do sistema com a finalidade de se criar uma base teórica que facilite a compreensão do funcionamento dos componentes e do sistema de uma forma geral.

2.1. Microcontroladores

Hoje em dia com os avanços tecnológicos, e com a necessidade de se criar componentes eletrônicos cada vez menores e com maior poder de processamento para suprir os requisitos dos atuais aparelhos eletrônicos, os microcontroladores aparecem como componentes eletrônicos poderosos e versáteis utilizados em uma grande variedade de aplicações.

Cada microcontrolador é dotado de um ou mais núcleos de processamento, memória e dispositivos para entrada e saída de dados, sendo o núcleo de processamento a principal parte de um microcontrolador (HAMACHER, VRANESIC, ZAKY e MANJIKIAN, 2012). Este é responsável por processar as informações recebidas por meio dos dispositivos de entrada e saída de dados de acordo com a programação preexistente na memória do microcontrolador, um esquema da estrutura de um microcontrolador pode ser encontrado na figura 2.1. Todas estas características serão discutidas durante esse capítulo.

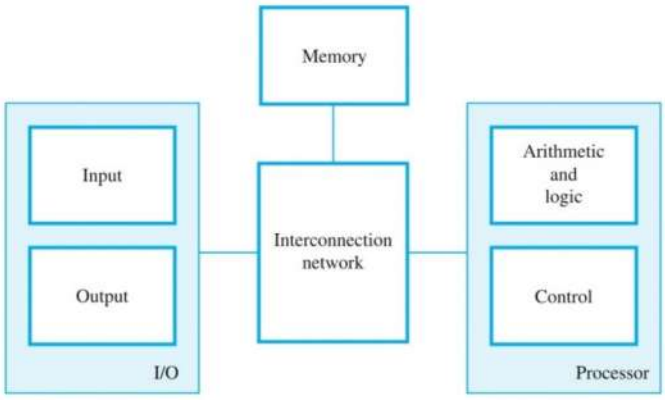

Figura 2.1 - Diagrama de Blocos de um microcontrolador.

Fonte: Hamacher, Vranesic, Zaky e Manjikian, 2012

2.1.1. Famílias de microcontroladores

Apesar das similaridades, existem diversas famílias de microcontroladores, cada uma aprestando certas características que as diferem das demais, alguns exemplos de famílias são: Freescale 68HC11 e 68k/ColdFire, intel 8051 e MCS-96 todas dotadas de processadores CISC, e existem também microcontroladores AVR que possuem processadores no modelo RISC.

2.1.1.1. Processadores da família Intel 8051 e MCS-96

Os processadores da família Intel 8051 e MCS-96 são construídos com arquitetura Harvard. Esse tipo de arquitetura possui barramento de dados e de instruções fisicamente separados, o que permite que o barramento de instruções seja acessado simultaneamente com o barramento de dados (ARGADE e BETKER,1996). A figura 2.2 mostra o esquema de processamento dos processadores com arquitetura Harvard.

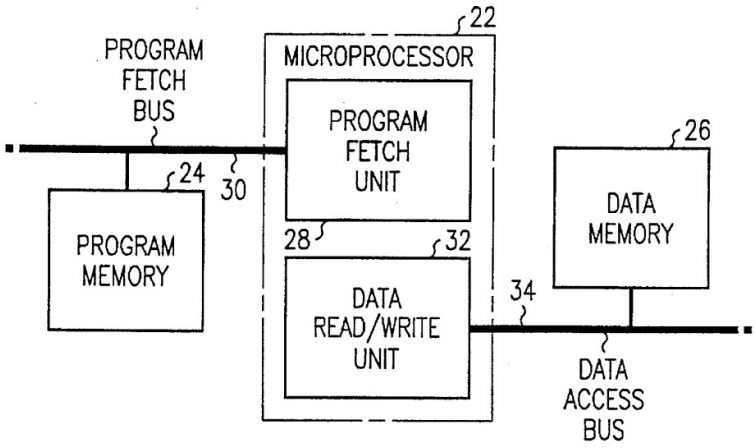

Figura 2.2 - Diagrama de blocos do funcionamento da arquitetura Havard.
Fonte: Argade e Betker,1996

Essa família possuía ainda como característica microprocessadores de 8 bits porem com 16 bits para endereçamento de endereços, 6-72 portas de I/O para uso geral, 0 a 64Kb de ROM e 128 a 256 bytes de RAM (INTEL, 2015).

2.1.1.2. Processadores da família Freescale 68HC11

Estes processadores são construídos sobre a arquitetura Von Neumann. Diferente da arquitetura Havard, os processadores construídos na arquitetura Von Neumann

possuem um único sistema de memoria que armazena tanto dados como instruções. Assim o microprocessador pode acessar apenas dados ou apenas instruções em dado período de tempo mais nunca conseguira acessas ambos simultaneamente como na arquitetura Havard (ARGADE e BETKER,1996). A figura 2.3 demonstra a forma de funcionamento da arquitetura Von Neumann.

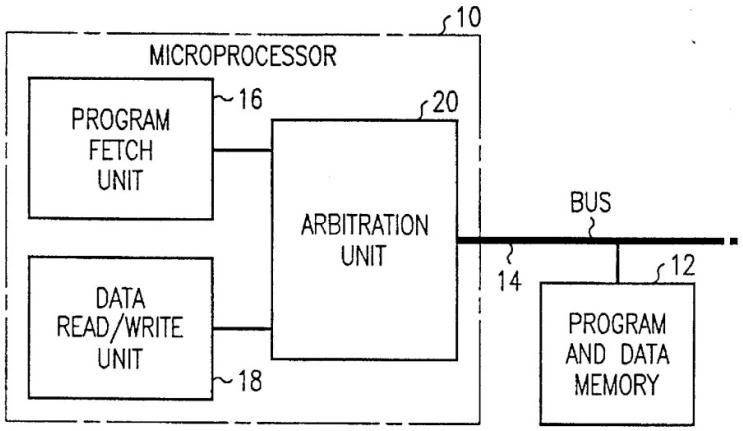

Figura 2.3 - Diagrama de blocos do funcionamento da arquitetura Von Neumann.
Fonte: Argade e Betker,1996

Outras características complementares da família Freescale 68HC11 são microcontroladores de 8Kb de memoria ROM, 512 bytes de memoria EPROM, 256 bytes de memoria RAM e 40 portas de I/O para uso geral (FREESCALE, 2015).

2.1.1.3. Processadores da família AVR

Com o passar dos anos e as altas demandas e exigências do mercado novas famílias de microcontroladores surgiram, uma delas foi a AVR. Tais microcontroladores possuem uma arquitetura Harvard modificada desenvolvida pela Atmel, onde as instruções possuem 16 bits enquanto os dados possuem 8 bits. As instruções viajam por um barramento de 16 bits separado do barramento de 8 bits destinado aos dados. Essa arquitetura é a mesma encontrada nos microprocessadores das placas Arduino. Na figura 2.4 podemos ver o diagrama de blocos do funcionamento dos microcontroladores da família AVR que compreende dentre outros o microcontrolador ATmega32U4, que é utilizado no Arduino Yun (ARGADE e BETKER,1996). Essa foi a placa de desenvolvimento secionada para o desenvolvimento do protótipo proposto neste trabalho.

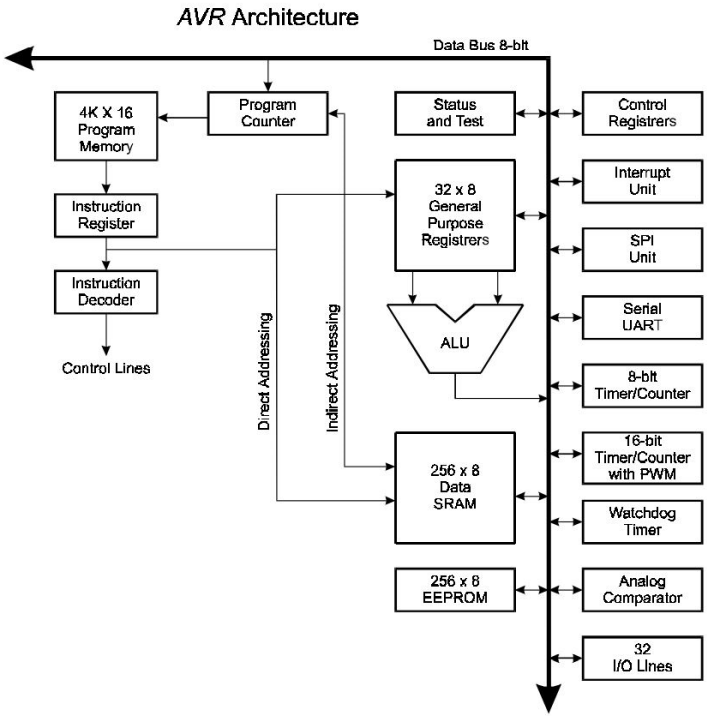

Figura 2.4 - Diagrama de blocos do funcionamento dos microcontroladores da família AVR.

Fonte: Argade e Betker,1996

Outras características importantes de tal arquitetura são que os mesmos são otimizados para funcionalidades que demandam baixo custo e baixo consumo de energia, possuem de 0 a 8 KB de memoria flash, 64 a 512 bytes de memoria SRAM, 64 bytes EEPROM, 32x8 registradores e executam uma instrução a cada ciclo (ATMEL, 2015).

2.2. Sistemas supervisórios

Quando se projeta uma aplicação utilizando sistemas embarcados, na maioria das vezes é desejável que se tenha um software de supervisão para que seja realizado o monitoramento do funcionamento da aplicação. Este software é chamado de sistema supervisório.

Supervisórios do tipo SCADA controlam e monitoram plantas ou instalações industriais de uma central de controle. Nesta central, todas as variáveis do processo envolvido nas instalações podem ser monitoradas e controladas sem a necessidade de se deslocar ate o local onde os mesmos estejam ocorrendo. Um exemplo de

programa utilizado para a confecção dos sistemas supervisórios é o ScadaBR, um software gratuito que possibilita a criação e comunicação de um supervisório com uma aplicação microcontrolada.

O controle entre supervisório e aplicação se da por intermédio de algum dispositivo controlador, como por exemplo um CLP ou microcontrolador. O mecanismo de funcionamento do mesmo é de fácil entendimento, em uma rede formada por um sistema automatizado utilizando como controlador um CLP e um computador dotado de um sistema supervisório, o supervisório realizará a leitura das portas do CLP periodicamente e caso haja alguma mudança indesejada em algum dos valores, o supervisório poderá tanto desencadear uma serie de ações de controle previamente programadas como gerar um alerta e esperar por uma ação de controle enviada pelo usuário do supervisório. A aplicação desenvolvida durante o desenvolvimento desse protótipo se utiliza se um supervisório do tipo SCADA com comunicação Wifi.

2.3. Protocolos de comunicação

Para que o sistema de supervisão seja capaz de realizar o controle e monitoramento de uma aplicação, o mesmo antes deve ser capaz de realizar a comunicação com o controlador. Dependendo da aplicação e do tipo de controlador utilizado existem diversos protocolos de comunicação disponíveis para se implementar a comunicação entre supervisório e controlador. Alguns exemplos são os protocolos Modbus, ethernet e wifi.

O sistema de transmissão de dados wifi estabelece a transmissão de dados dentro de uma rede sem a utilização de um meio físico conectando as máquinas participantes da rede. Para tanto tal padrão utiliza a propagação das ondas eletromagnéticas para realizar a comunicação dentro da rede.

De um ponto de vista industrial, a utilização de redes wifi possuem aspectos positivos e negativos que devem ser levados em consideração quando se planeja utilizar a mesma. O principal aspecto negativo da utilização de uma rede wifi é a susceptibilidade a interferências eletromagnéticas que podem ser causadas pela proximidade com o maquinário industrial, bem como a cobertura de rede reduzida (NOGUEIRA, 2009). Caso exista a possibilidade de se descartar estes dois problemas, a rede wifi se torna uma alternativa extremamente interessante pois proporciona entre outros a facilidade de alteração do layout da rede sem a

preocupação com cabeamentos, fácil e rápida integração com dispositivos de rede e fácil instalação. Por esses pontos positivos, este será o protocolo de comunicação utilizado neste trabalho.

3. Componentes do protótipo

Todas as aplicações e dispositivos requerem algum tipo de dado de entrada para reconhecer o ambiente em que o mesmo está inserido e desencadear uma ação de controle com base no estado dos periféricos de entrada de dados. Esses dados podem ser gerados e adquiridos de várias maneiras, sendo a mais comum a utilização de sensores e com base no estado dos mesmos o microcontrolador é capaz de acionar de forma precisa os diversos atuadores de uma aplicação eletrônica.

Neste capítulo serão abordados os dispositivos elétricos e eletrônicos utilizados na confecção do *pet feeder*, suas características e razão para escolha dos mesmos.

3.1. Sensores e atuadores

Neste projeto foram utilizados um sensor ultrassônico HC-SR04 responsável por constantemente checar o nível dos reservatórios, um servo motor de rotação contínua modelo SM-S4303R responsável por liberar a ração do reservatório para a vasilha de alimentação do animal e dois LED's responsáveis por sinalizar nível crítico no reservatório.

3.1.1. Sensor ultrassônico (HC-SR04)

O sensor ultrassónico SR04 utiliza o mesmo princípio de funcionamento que morcegos e golfinhos utilizam para detectar objetos a sua frente. O sensor possui um emissor e um receptor de ondas ultrassônicas como mostra a figura 3.1. O emissor emite um sinal ultrassônico que ao encontrar um objeto será refletido e captado pelo receptor, com base no tempo gasto e na velocidade de propagação das ondas ultrassônicas o receptor é capaz de determinar a distancia entre o sensor e o objeto.

O SR04 possui uma resolução de 0.3 cm podendo detectar objetos entre 2 e 400 cm de distancia e posicionados dentro de uma angulação de ate +15°/-15° em relação ao sensor. Para seu funcionamento o mesmo requer uma fonte de alimentação de 5V DC (CYTRON, 2015).

Sua escolha foi feita com base na necessidade de se medir o nível de comida nos reservatórios. Outra opção para desempenhar tal função seria o sensor infravermelho. O que levou a escolha do ultrassônico frente ao infravermelho foi o

fato do infravermelho ser mais suscetível a ruídos externos como luz ambiente e superfícies refletivas.

Figura 3.1 - Sensor Ultrassônico HC-SR04.
Fonte: *Cytron Product User's Manual - HC-SR04 Ultrasonic Sensor*

3.1.2. Servo motor de rotação continua (SM-S4303R)

Para realizar a liberação de ração pelo *pet feeder* foi utilizado um servo motor de rotação contínua como o da figura 3.2. Ele será responsável por atuar uma válvula borboleta que por sua vez será responsável por liberar a ração para a vasilha de alimentação do animal.

O servo motor opera utilizando 3 pinos, sendo um responsável pela alimentação de 4.8V, um pino GND (terra) e um pino PWM. O pino PWM é responsável por controlar o servo; O mesmo funciona enviando um sinal de 5V ao servo por um período de tempo. Com base na duração da largura de pulso que o servo ele se movimenta para um lado, para outro, ou fica imóvel (Silva, 2013). Como mostrado na figura 3.2 o servo matem 3 estados de acordo com a largura do pulso recebido. Caso o mesmo receba um sinal de 5V do arduino por 2ms, ele atuara na direção anti-horária. Caso este sinal tenha duração de 1,5ms o mesmo permanecera imóvel. E por fim, caso tal sinal tenha duração de 1ms ele atuara na direção horaria.

No arduino, por meio da biblioteca servo.h, é possível se realizar o controle de uma forma mais simplificada. A biblioteca utiliza valores de 0 a 180, correspondentes ao ângulo de atuação, sendo 90 parado ao meio, 0 movendo-se a esquerda e 180

movendo-se a direita. Tais valores são análogos aos tempo de exposição do servo aos 5V. Assim 2ms corresponde a 0°, analogamente 1ms corresponde a 180° em quanto 1,5ms corresponde a 90°.

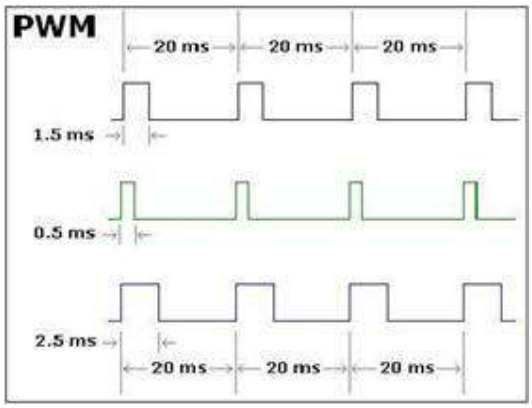

Figura 3.2 – Controle de um servo motor.
Fonte: Silva, 2013.

O S4303R possui um torque de 3.3 kg.cm quando ligado a uma tensão de 4.8V e 4.8 kg.cm quando ligado a uma tensão de 6V, sendo o ultimo mais indicado para atuar acionar a válvula borboleta (SPRINGRC, 2015). A escolha do mesmo foi realizada com base na facilidade para se desenvolver um controle preciso em cima do numero de rotações do servo frente a outros tipos de motores.

Figura 3.3 - Servo motor de rotação continua (SM-S4303R)

3.2. Microcontrolador

Alguns dos requisitos para se desenvolver este projeto era que o microcontrolador possuísse pelo menos 7 portas digitais, portas SDA e SCL para comunicação I2C com o RTC, pino PWM para controle do servo, memória suficiente para suportar a programação e possibilitar alguma forma de comunicação *wireless*. A maioria dos microcontroladores necessitaria de um circuito dedicado a estabelecer a conexão wireless, porém atendiam aos demais requisitos. Assim foi decidido pela utilização de microcontroladores da plataforma Arduino pela facilidade de programação e disponibilidade de bibliotecas para auxiliar a elaboração do programa de controle.

As características das placas Arduino foram analisadas até se chegar à conclusão de que o Arduino Yun seria o que mais se adequaria ao projeto. Ele atenderia a todos os requisitos básico para a implementação do sistema, bem como ainda ofereceria uma plataforma multiprocessada que utiliza o processador ATmega 32u4 exclusivamente para processamento dos dados e instruções e o processador Lilino AR9331 para controlar e gerenciar as conexões wifi, ethernet, USB e ainda com cartões SD (ARDUINO, 2015). Essas relações podem ser vistas na figura 3.3. Em outras palavras o Arduino Yun já vem de fábrica com os Shields wifi, Ethernet, USB e de leitor de cartões SD integrado ao mesmo.

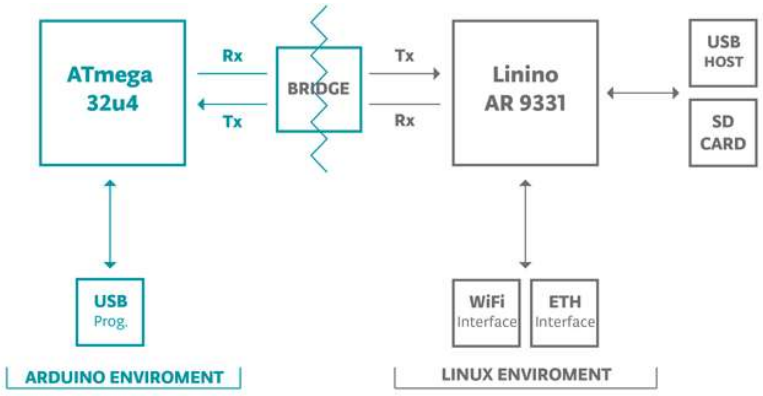

Figura 3.4 - Arquitetura do arduino Yun.

Fonte: ARDUINO, 2015.

3.2.1. Características da plataforma Arduino Yun

O Arduino Yun é um dos mais recentes lançamentos da Arduino. O microcontrolador possui suporte a rede ethernet e wifi, porta USB-A, leitor de cartões microSD, 20 entrada/saída digitais das quais 7 podem ser utilizadas como saídas PWM e 12

como saídas analógicas, cristal de 16 MHz e conexão micro USB (ARDUINO, 2015). Todo este hardware está embutido em uma pequena placa como mostrado na figura 3.4.

Além das características básicas, o fato do Arduino Yun possuir suporte wifi integrado a placa facilitaria a implementação de um supervisório sem fio, oferecendo suporte nativo tanto para aplicações em rede local como através da internet.

Figura 3.5 - Arduino Yun.
Fonte: ARDUINO, 2015.

3.3. RTC

Como o sistema proposto deve atuar em horários específicos e predeterminados, é desejável que se possa manter controle sobre o tempo. Infelizmente o microcontrolador não possui embutido a ele uma ferramenta confiável para se manipular tal variável. O que o arduino Yun tem são alguns *timers* embutidos, que se utilizam da frequência do cristal para manipular o tempo. O grande problema em se utiliza-los é o fato de que esses *timers* são voláteis, ou seja caso a conexão com a fonte de alimentação seja encerrada os mesmos perdem os valores que guardavam, dessa forma caso haja uma súbita falta de energia o sistema seria resetado juntamente com os *timers*, sendo impossível assim realizar alguma tarefa crítica no tempo utilizando-se apenas os mesmos (ARDUINO, 2015).

Assim, foi necessário utilizar-se um componente externo que guardasse o tempo atual e o mantivesse mesmo em casos de perda de conexão com a fonte de

alimentação. Essa é a função desempenhada pelo RTC mostrado na figura 3.5, ele é responsável por controlar o fluxo de tempo e utiliza uma bateria interna para que mesmo em casos extremos consiga manter o tempo sempre atualizado. Dessa forma com a utilização do microcontrolador para processar as informações e dados e o RTC para gerenciar o tempo, caso o sistema enfrente uma falta de energia e seja reiniciado ele não enfrentará problemas, pois o RTC manterá o tempo sempre atualizado e o microcontrolador (MAXIM, 2015).

Sua comunicação com o microcontrolador se dá através do protocolo de comunicação I2C. Este protocolo diminui a quantidade de pinos necessários para estabelecer a comunicação entre dois circuitos integrados, para utilizar o mesmo é necessário apenas dois pinos o SDA e o SCL, o que acaba por baratear e reduzir o tamanho e complexidade das placas (NXP, 2015).

Neste projeto foi utilizado o RTC DS1307. Este é alimentado por uma fonte externa de alimentação de 5V, porém possui um circuito integrado ao chip que é responsável por sentir as variações na linha de energia. Quando uma queda de energia é detectada, ele automaticamente troca a forma de alimentação do chip da fonte externa para a fonte backup e continua mantendo a contagem de fluxo do tempo a partir da fonte de backup ate que a fonte externa de alimentação seja restaurada (MAXIM, 2015).

Figura 3.6 - Modulo RTC.

4. Desenvolvimento e resultados

A implementação do *pet feeder* pode ser dividida em 3 etapas distintas, aferimento dos sensores e testes, hardware e software. A etapa de aferimento e teste serve para garantir que os sensores estavam em perfeito estado de funcionamento e evitar erros futuros devido a um sensor defeituoso. Na parte do hardware, todos os componentes são ligados e devidamente conectados e preparados para funcionamento. E finalmente na etapa do desenvolvimento de software, foi desenvolvida a programação responsável por controlar o hardware montado e um sistema de supervisão para monitoramento remoto do mesmo.

4.1. Aferimento e testes

O primeiro passo para a montagem do *pet feeder* foi garantir que cada peça utilizada no projeto estava devidamente funcional. Para tanto cada peça foi testada separadamente ou em conjunto com algum outro componente para garantir que nenhum dos sensores ou componentes estava defeituoso.

4.1.1. Sensores ultrassônicos

Inicialmente foi montado o circuito necessário para o funcionamento de um sensor ultrassônico. Após a conclusão da montagem foi realizado o upload para o arduino de um programa simples que apenas ativava o sensor e retornava no terminal a distância entre objeto e sensor. Foram testados os dois sensores (O sensor do reservatório e da vasilha de alimentação) e com o auxílio de uma régua, conforme pode ser visto na figura 4.1, foi possível constatar que todos apresentaram um erro máximo de cerca de 1 cm entre o resultado mostrado no terminal do IDE e a régua.

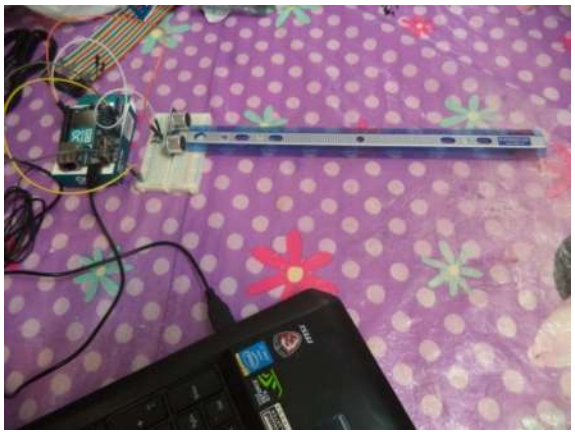

Figura 4.1 - Teste e aferimento do sensor ultrassónico.

4.1.2. LEDs

Para o teste dos mesmos foi utilizada a programação implementada para o teste do sensor ultrassônico com algumas modificações. Foram realizadas algumas modificações no circuito de forma a possibilitar que o LED fosse adicionado ao mesmo. O programa checava a distância entre sensor e objeto e acionava a luz vermelha caso o valor ultrapassasse um set point, este teste pode ser visto na figura 4.2. Nessa etapa todos os LEDs foram testados e um dos mesmos teve de ser descartado pois não estava funcionando corretamente.

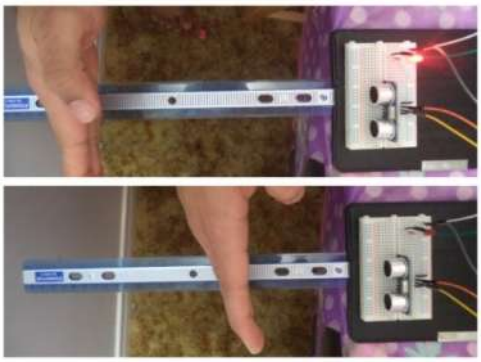

Figura 4.2 - Teste dos LEDs.

4.1.3. Servo Motor

O último dispositivo a ser testado e aferido foi o servo motor. Para tanto foi criado um circuito para conectar o mesmo ao arduino conforme mostrado na figura 4.3, e feito o upload de um programa simples onde o servo deveria rodar para o lado esquerdo por 2 segundos, para o lado direito por 2 segundos e ficar parado por 1 segundo antes de começar tudo novamente. Ao final do teste o servo funcionou como o esperado.

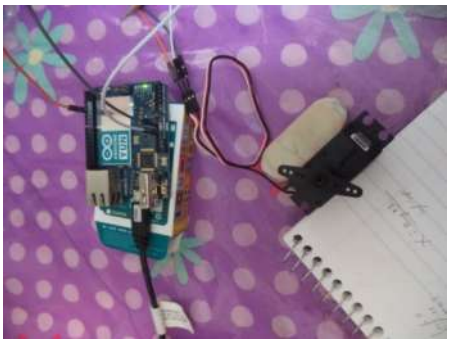

Figura 4.3 - Teste e aferimento do servo motor.

4.2. Hardware

Com todos os componentes eletrônicos funcionais e prontos, foi iniciada a montagem do *pet feeder*. O mesmo foi montado em um *dispenser* de cereais, onde a ração é armazenada, neste reservatório existe um LED assinalando nível crítico e um sensor ultrassónico que realiza a checagem de nível constantemente. O mesmo ocorreu na vasilha de alimentação do animal. O servo motor foi instalado na válvula borboleta do *cereal dispenser*, e no horário predeterminado o servo rotaciona a válvula borboleta liberando a ração até que a mesma atinja um nível predeterminado.

Após a montagem e aferimento dos sensores, foram encontrados alguns problemas não esperados. Para alguns destes foi possível achar uma solução provisória, porem para os outros a solução pode apenas ser indicada.

Um dos problemas encontrados foi o fato de o servo motor não possuir torque suficiente para rotacionar a válvula borboleta. Ao atuar, o servo tentava rotacionar a válvula, porem ao aplicar força na mesma ele se desarmava continuamente devido ao alto torque requerido. Neste caso, como não existia outro servo motor que pudesse ser usado no projeto, não foi possível solucionar o problema, porem a solução seria a troca do servo por um servo de maior torque.

A montagem final pode ser conferida nas figuras 4.4 e 4.5, sendo a primeira a vista frontal e a segunda a vista das costas do protótipo.

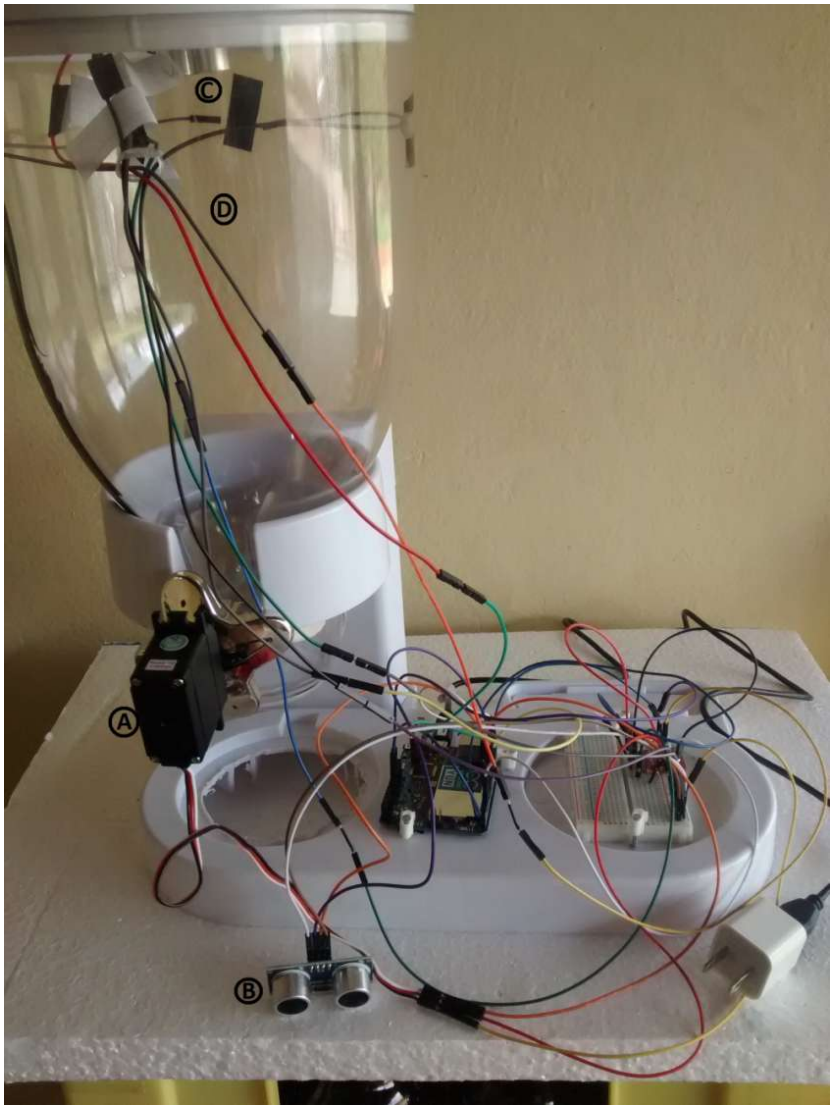

Figura 4.4 - (A) Servo motor acoplado a válvula borboleta, (B) Sensor ultrassónico de nível do recipiente de alimentação, (C) Sensor ultrassónico de nível do reservatório de ração e (D) Dispenser em acrílico que funciona como reservatório do sistema.

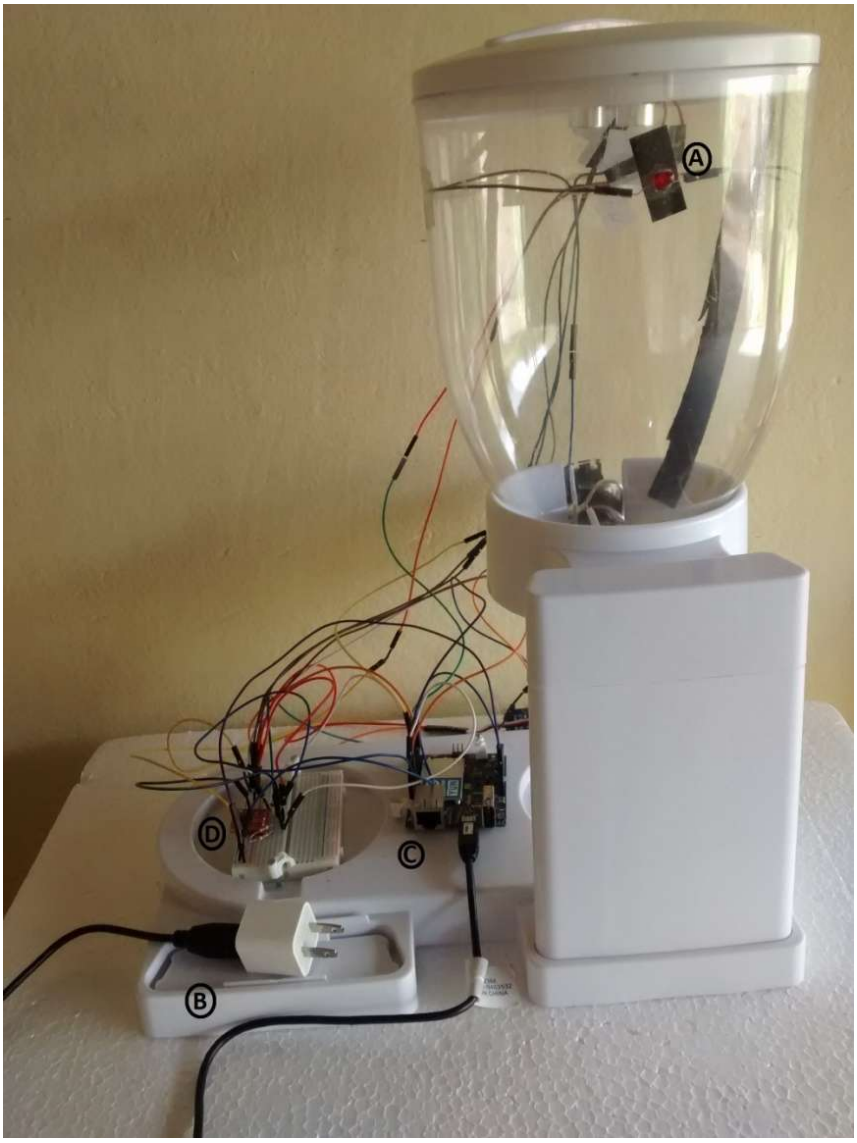

Figura 4.5 - (A) Led de alerta de nível de ração baixo no reservatório, (B) Fonte de alimentação do sistema, (C) Placa arduino e (D) protoboard com RTC acoplado.

4.2.1. Circuito

O circuito foi montado observando algumas particularidades de cada componente. Para tanto, um protoboard foi utilizado para auxiliar as ligações. Primeiramente ligou-se a porta de 5V e GND do protoboard ao arduino. Todos os componentes são alimentados e aterrados por meio destas duas linhas de portas no protoboard. Após tal passo, começou-se a montagem dos atuadores e sensores.

Na montagem dos sensores, os sensores ultrassónicos e o RTC foram ligados ao protoboard e arduino. Os sensores ultrassónicos possuem 4 pinos, sendo estes:

Echo, Trigger, 5V e GND. Os pinos Trigger foram conectados diretamente no arduino nas portas digitais 12 e 6. Os pinos Echo por sua vez foram conectados as portas digitais 11 e 5.

O RTC por sua vez necessita de um circuito mais especifico. Este utiliza resistores pull-up(4,7 kΩ) para o correto funcionamento da comunicação I2C. Tais resistores tem a função de manter as portas SDA e SCL em alta quando o barramento está livre e elevar as mesmas de baixo para alto no período requerido (NXP, 2015). O RTC possui 5 portas, SDA, SCL, SQW, GND e 5V. As portas SDA e SCL são as únicas que necessitam de tais resistores. Cada resistor foi conectado a uma das portas de 5V do protoboard e a outra porta conectada ao arduino e ao RTC nas respectivas portas.

O servo motor e o LED foram os últimos a serem conectados. A conexão dos dois é relativamente simples. O servo possui 3 pinos, 5V, GND e Signal. O único requisito do mesmo é que a saída signal seja conectada a uma porta digital com PWM. Assim a mesma foi conectada a porta digital 9 do arduino que possui tal funcionalidade.

Por fim foi feita a conexão entre o LED e o circuito. Para tal conexão foi utilizado um resistor de 220Ω em serie com o resistor. Uma perda do resistor foi conectada a porta digital 4 do arduino e a outra conectada ao anodo do LED. A outra perna do LED, Catodo, foi conectado ao GND finalizando a montagem do circuito. A figura 4.6 mostra o esquema do circuito montado.

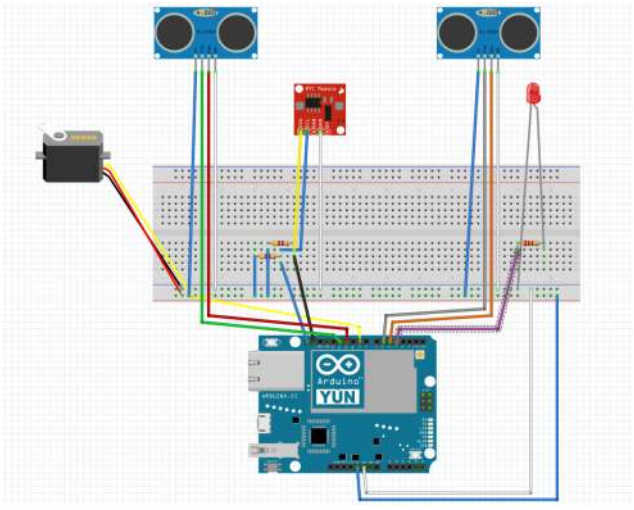

Figura 4.6 – Esquema do circuito elétrico do protótipo.

4.3. Software

A parte de software pode ser dividida em duas, a programação do microcontrolador utilizando a IDE do arduino, e a configuração do sistema supervisório utilizando o ScadaBR.

4.3.1. Arduino

A programação do microcontrolador foi desenvolvida utilizando-se a própria IDE do arduino. Um dado interessante a ser ressaltado e que como o arduino Yun possui o modulo wifi integrado ao mesmo, caso seja necessário realizar alguma alteração na programação do microcontrolador não é necessário nenhum tipo de ligação física com o mesmo, as alterações podem ser feitas em um computador conectado a mesma rede que o arduino, e o upload pode ser feito via rede wifi direto da IDE selecionando como porta o IP do arduino Yun na rede. Isso torna o projeto mais amigável a futuros upgrades na programação.

O programa que roda no arduino inicia o seu funcionamento checando se existe alguma requisição do sistema supervisório, caso exista ele irá processar a requisição e enviar o valor requisitado ao mesmo, caso não exista seguira com a execução normal. Então, o programa irá ficar constantemente checando o horário guardado pelo RTC e comparando este ao horário programado pelo usuário, quando ambos os horários forem os mesmo o programa irá então ver a quantidade de ração que existe na vasilha de alimentação e completar o mesmo até uma quantidade predeterminada (por padrão ira encher 80% da vasilha de alimentação). Após isto ele voltará ao estado inicial e iniciará tudo novamente.

A comunicação com o sistema supervisório se dá no padrão cliente-servidor, o sistema supervisório requisita um dado através de um endereço HTTP, o arduino constantemente irá checar se existe algum cliente requisitando informações no servidor local, caso exista ele fará o processamento da requisição HTTP e enviará a resposta para o cliente.

Para realizar a implementação desse programa algumas bibliotecas foram utilizadas para facilitar a programação e comunicação com os vários periféricos de I/O. Todas

estas bibliotecas foram retiradas diretamente do site do Arduino. A relação de bibliotecas utilizadas segue a baixo:

Biblioteca	Descrição
Bridge.h YunServer.h YunClient.h	Usa requisições do tipo REST para acessar pinos analógicos e digitais na placa.
NewPing.h	Faz melhorias as bibliotecas ping.h e ultrasonic.h corrigindo problemas e melhorando o desempenho dos sensores ultrassónicos.
Time.h	Adiciona a capacidade de cronometrar o tempo à placa arduino.
Wire.h	Permite comunicação com periféricos utilizando o protocolo I2C.
DS1307RTC.h	Permite o acesso a dispositivos RTC compatíveis com o DS1307
Servo.h	Permite a placa arduino controlar servo motores. Suporta entre 12 e 48 motores dependendo da placa arduino utilizada.

Tabela 1. Relação de bibliotecas utilizadas na programação do arduino.

A programação completa utilizada neste projeto se encontra o fim do trabalho no apêndice A, e pode ser utilizada de forma a melhorar a compreensão do trabalho proposto pelo protótipo.

4.3.2. Sistema supervisório

Para a criação do sistema supervisório, que utiliza a plataforma ScadaBr, o primeiro passo foi a configuração dos *Data Sources*. Como explicado a comunicação entre o supervisório e o arduino se dá por meio de requisições HTTP, por tanto o *data source* a ser utilizado em nosso caso é o HTTP *Retriever*. Este *data source* requisita informações de um servidor HTTP e utiliza Regex para filtrar a informação desejada.

Existem dois campos a serem preenchidos quando se cria um novo *data source* (figura 4.6), o nome do *data source* e a URL vinculada ao mesmo. O campo URL

deve ser preenchido com o endereço HTTP para qual os dados serão requisitados. Outro campo importante a ser alterado é o campo *Update period*, este foi alterado para que o supervisório faça atualização do valor a cada 500 ms.

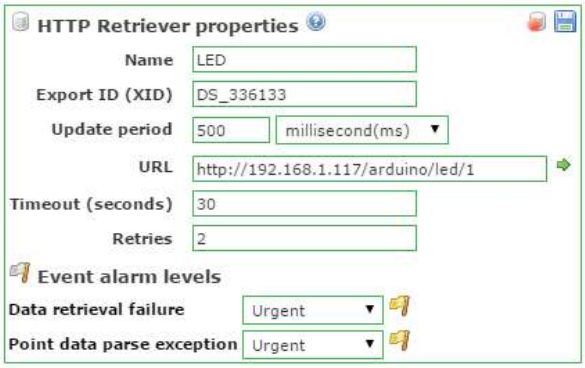

Figura 4.7 - Configuração do *Data Source* LED.

Após a configuração do *data source* os *Data points* (figura 4.7) devem ser configurados. Points são dados de entrada relacionados ao *data source* criado, em outras palavras são os valores das leituras da(s) porta(s) configuradas no URL do *data source*. Para configurar o mesmo é necessário selecionar o tipo de dado que será retornado pela URL e configurar o Regex que deverá ser aplicado a resposta do servidor HTTP para se isolar apenas o dado desejado.

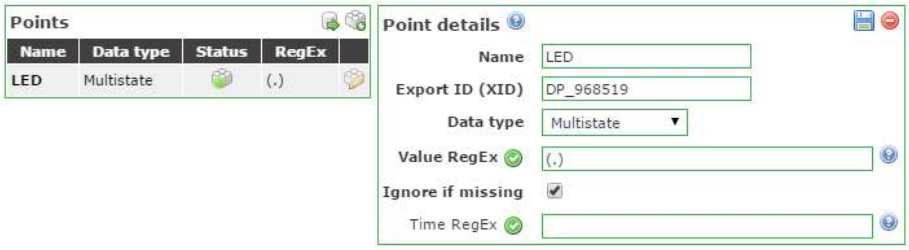

Figura 4.8 - Configuração do *Data Point* LED.

Após ter todos os *data sources* e data points configurados o monitoramento já pode ser iniciado através do menu *watch list*. O monitoramento pode ser realizado também através do menu *graphical view mostrado na figura 4.8*, para tanto foi necessária a configuração dos componentes da *graphical view*.

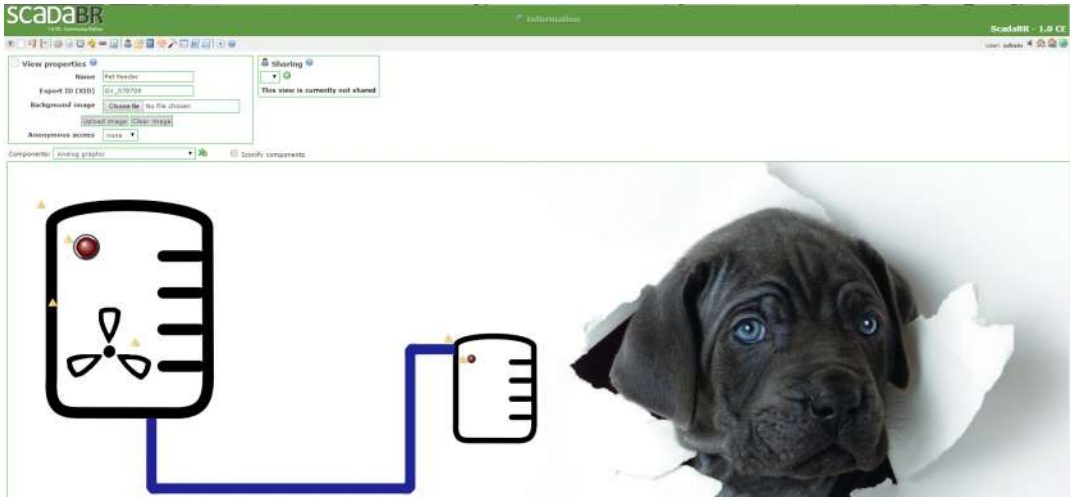

Figura 4.9 - Configuração final do graphic view do supervisório.

O sistema supervisório elaborado pode ser acessado através de qualquer dispositivo eletrônico conectado a mesma rede que o Arduino *Yun*. Para realizar o acesso é preciso apenas realizar o *login* no ScadaBR, e o monitoramento será iniciado.

4.4. Funcionamento

Ao ser iniciado alguns problemas foram encontrados, como o problema com o servo citado no capitulo 4.2. Porem o supervisório funcionou conforme o idealizado, o mesmo conseguia se comunicar de forma wireless com o dispositivo e receber os níveis de ração nos recipientes. Além disso, quando o movimento do servo motor era iniciado o movimento do mesmo começava no sistema supervisório conforme visto na figura 4.10.

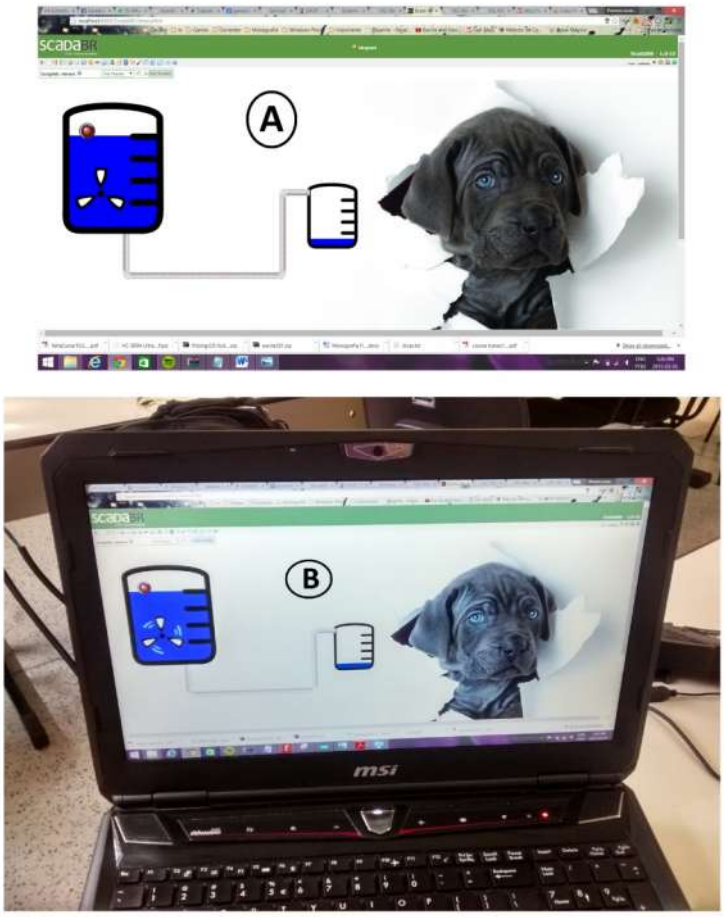

Figura 4.10 – (A) Servo parado e (B) Servo em funcionamento

Outro problema encontrado foi o fato de que não foi possível achar uma maneira de fácil implementação que tornasse possível a comunicação bidirecional do arduino com o sistema supervisório de forma *wireless*. Dessa forma o sistema supervisório pode ler os estados das portas digitais do arduino, porem não consegue passar o horário de funcionamento para o *pet feeder.*

Para contornar o problema com o horário foi utilizado o próprio servidor do arduino para que o horário pudesse ser definido a partir de qualquer dispositivo e não apenas dentro do código do mesmo. Dessa forma o usuário precisa entrar a partir de qualquer navegador no endereço: 192.168.1.117/arduino/time/XXXX. E substituir o XXXX pelo horário conforme mostra a figura 5.1, onde os dois primeiros X significam o horário de 0 a 23 horas, e os dois seguintes os minutos de 0 a 59 minutos, caso este horário seja digitado de forma errada a página retornará uma mensagem de

erro conforme mostra a figura 5.2. Dessa forma caso seja desejado que o protótipo atue as 15:15 deve se digitar a seguinte url no navegador: 192.168.1.117/arduino/time/1515.

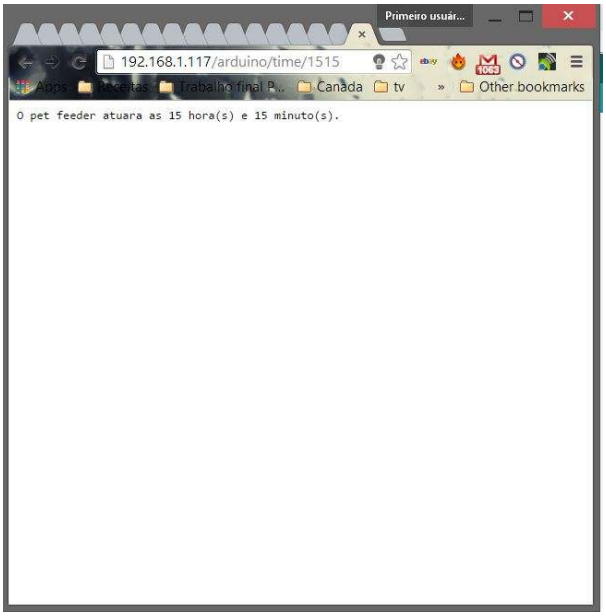

Figura 4.11 - Mensagem de programação feita com sucesso.

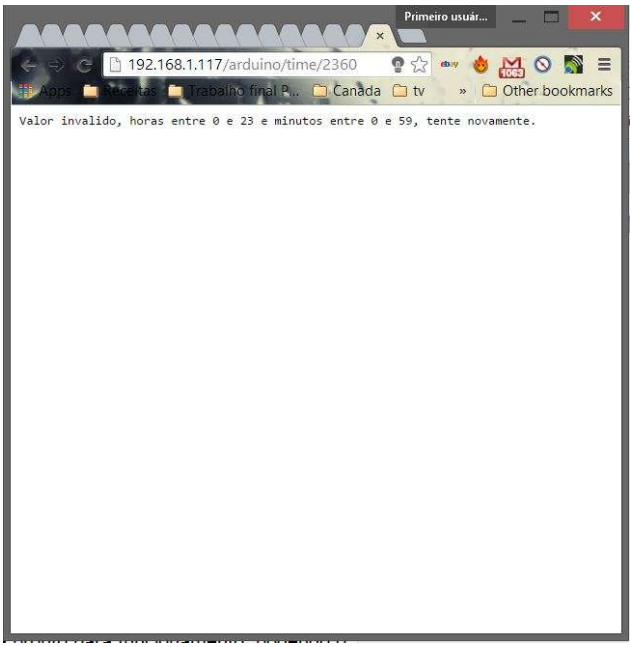

Figura 4.12 - Erro gerado ao entrar com um valor invalido para o horário.

Por fim o protótipo apresentou mais um problema com um dos sensores ultrassónicos, o qual apresentou leituras incorretas. Este sensor estava medindo o

nível de comida na vasilha de alimentação e deverá ser substituído para o correto funcionamento do sistema.

Para utilização pelo consumidor final deste protótipo, foi necessário a criação de um manual de instruções que contivesse todos os passos necessários para a correta configuração inicial do *pet feeder*. Esta configuração definirá os níveis cheio e vazio do recipiente, horário de atuação e configuração da rede wireless. Após tal configuração o protótipo estará pronto para funcionamento, podendo o usuário apenas monitorar o mesmo através do supervisório e alterar o horário através de qualquer navegador.

5. Conclusão e Trabalhos Futuros

Ao se confrontar os resultados obtidos no teste do protótipo com os resultados esperados, se conclui que apenas parte dos objetivos foi alcançada. Apesar dos principais objetivos terem sido alcançados, existem muitas melhorias que ainda precisam ser feitas no mesmo para que o mesmo funcione corretamente.

Ao se analisar quais objetivos tiveram êxito em seus resultados conclui-se ainda que a maior parte dos erros foram causados pela má escolha do hardware. Caso um servo com maior torque fosse utilizado, o problema com a rotação da válvula seria contornado. Ainda, caso um novo sensor ultrassónico fosse acoplado, as variações de leitura do sensor responsável pela vasilha de alimentação poderiam ser reduzidas.

Além dos erros de hardware, existe um erro de software que não pode ser desprezado. Durante o a elaboração do protótipo não foi possível estabelecer a comunicação bilateral entre supervisório e arduino. Neste caso, seria necessário a implementação do supervisório em outra plataforma que não o ScadaBR ou um estudo mais detalhado do mesmo para determinar se é possível ou não tal comunicação via wireless com o arduino.

Por fim, para se obter um protótipo funcional conclui-se que mudanças tanto em nível de software como de hardware são necessárias. Uma vez que tais mudanças sejam feitas, não existem mais motivos aparentes para que o protótipo não funcione como o idealizado.

Após as mudanças feitas e com o protótipo funcional, este trabalho poderá abrir espaço a um amplo leque de inovações. Tais inovações podem ser implementadas em cima do projeto original e algumas são citadas abaixo como sugestões para desenvolvimento futuro do protótipo:

- Envio de alertas a respeito de nível de ração nos reservatórios por e-mail. Esta funcionalidade pode ser implementado tanto no supervisório como diretamente no arduino Yun.
- Envio de alertas por mensagens SMS e comunicação e controle do dispositivo por mensagens SMS, pode ser implementado ao utilizar-se um shield SMS para arduino em conjunto com o arduino Yun.

- Estender o suporte do supervisório para que o mesmo possa ser acessado remotamente através da internet e não apenas da rede local.
- Aplicar a mesma ideia para realizar o controle da quantidade de agua na vasilha do animal.
- Definir horário de funcionamento através do supervisório.

Estas foram apenas algumas das funcionalidades que poderiam ser implementadas em cima do projeto proposto. O que demostra quão flexível e quantas possibilidades para futuras atualizações e desenvolvimentos esse protótipo apresenta.

6. Referencias Bibliográficas

ABINPET – Associação Brasileira de Indústria de Produtos Para Animais de Estimação. **População de pets cresce 5% ao ano e Brasil é quarto no ranking mundial**. Disponível em: <http://abinpet.org.br/imprensa/noticias/populacao-de-pets-cresce-5-ao-ano-e-brasil-e-quarto-no-ranking-mundial/>. Acesso em: 29/01/2015.

ARDUINO. ***Arduino YUN***. Disponível em: <http://arduino.cc/en/Main/ArduinoBoardYun?from=Products.ArduinoYUN>. Acesso em: 29/01/2015.

ARDUINO. ***Arduino Time Library***. Disponível em: <http://playground.arduino.cc/Code/Time>. Acesso em: 29/01/2015.

ARDUINO. ***Arduino Wire Library***. Disponível em: <http://arduino.cc/en/Reference/Wire>. Acesso em: 29/01/2015.

ARDUINO. ***Arduino Servo Library***. Disponível em: <http://arduino.cc/en/Reference/Servo>. Acesso em: 29/01/2015.

ARDUINO. ***Arduino NewPing Library***. Disponível em: <http://playground.arduino.cc/Code/NewPing>. Acesso em: 29/01/2015.

Argade, Pramod V. e Betker, Michael R**.** ***Apparatus and method for computer processing using an enhanced Harvard architecture utilizing dual memory buses and the arbitration for data/instruction fetch*****.*** Patente 230. 1996.

ATMEL Corporation. ***The AVR Microcontroller and C Compiler Co-Design***. Disponível em: <http://www.atmel.com/Images/compiler.pdf>. Acesso em: 29/01/2015.

CARRIDE, Raul Diego Ocanha de Almeida. **Alimentador automático para cachorros**. Monografia de graduação. Universidade São Francisco. Itatiba, 2008.

Cytron Technologies. ***HC-SR04 Ultrasonic Sensor User's Manual***. Disponível em: <http://cytron.com.my/p-sn-hc-sr04?search=hc-sr04>. Acesso em: 29/01/2015.

Freescale Semiconductor, Inc. ***M68HC11 Microcontrollers Reference Manual***. Disponível em:

<http://www.freescale.com/files/microcontrollers/doc/ref_manual/M68HC11RM.pdf>. Acesso em: 29/01/2015.

HAMACHER, C.; VRANESIC, Z.; ZAKY, S. e MANJIKIAN, N. ***Computer Organization and Embedded Systems.*** 6 ed. McGraw-Hill, 2012.

Intel Corporation. ***MCS 51 Microcontroller Family User's Manual***. Disponível em: <http://www.cs.cmu.edu/~varun/cs315p/INTEL8051MAN.pdf>. Acesso em: 29/01/2015.

Intel Corporation. ***8051 Single-Chip Microcomputer Architectural Specification and Functional Description***. Disponível em: <http://archive.org/stream/bitsavers_intel80518liminaryArchitecturalSpecificationMay80_6120863/8051_Microcomputer_Preliminary_Architectural_Specification_May80#page/n0/mode/2up>. Acesso em: 29/01/2015.

Maxim Integrated Products, Inc. ***DS1307 64x8, Serial, I²C Real-Time Clock***. Disponível em: <http://datasheets.maximintegrated.com/en/ds/DS1307.pdf>. Acesso em: 29/01/2015.

Modbus Organization. **Modbus FAQ**. Disponível em: <http://www.modbus.org/faq.php>. Acesso em: 29/01/2015.

NOGUEIRA, Thiago Augusto. **Redes de comunicação para sistemas de automação industrial.** Monografia de graduação. Universidade Federal de Ouro Preto. Ouro Preto, 2009.

NXP Semiconductors N.V. ***I²C-bus specification and user manual***. Disponível em: <http://www.nxp.com/documents/user_manual/UM10204.pdf>. Acesso em: 29/01/2015.

OCHAKOWSKY, Nádia. **Protótipo de um Alimentador automático para Animais de estimação**. Monografia de graduação. Universidade Regional de Blumenau, 2007.

OKI, Nobuo e MANTOVANI, Suely Cunha Amaro. **TEEE I – Projeto de robôs Móveis**. Faculdade de Engenharia de ilha Solteira. 2013.

SILVA, Leticia Thais da. **Uma Proposta Para Automação Residencial Utilizando Uma Plataforma De Prototipagem Eletrônica Arduino.** Monografia de graduação. Pontifica Universidade Católica de Minas Gerais. 2013.

SILVA, Maria do Rosário Gomes. **Uso de Internet Embedded em microcontroladores para aplicações de monitoramento e automação de baixo custo.** Relatório Final Bic-Jr. CEFET - MG. 2006.

SpringRC. ***43R Servo(360° Rotation) Specification***. Disponível em: <https://www.sparkfun.com/datasheets/Robotics/servo-360_e.pdf>. Acesso em: 29/01/2015.

Apêndice A - Código para o Arduino.

```
//Includes


#include <Bridge.h>
#include <YunServer.h>
#include <YunClient.h>
#include <NewPing.h>
#include <Time.h>
#include <Wire.h>
#include <DS1307RTC.h>
#include <Servo.h>


//Constantes
#define TRIGGER_PIN_BOWL   11   // Pino do Arduino ligado ao pino trigger do sensor ultrassónico.
#define ECHO_PIN_BOWL      12   // Pino do Arduino ligado ao pino ECHO do sensor ultrassónico.
#define MAX_DISTANCE_BOWL  50  // Distancia máxima que será medida pelo sensor.


#define TRIGGER_PIN_DISPENSER  6  // Pino do Arduino ligado ao pino trigger do sensor ultrassónico.
#define ECHO_PIN_DISPENSER     5  // Pino do Arduino ligado ao pino ECHO do sensor ultrassónico.
#define MAX_DISTANCE_DISPENSER 25 // Distancia máxima que será medida pelo sensor.


#define LED_D4 4


#define DISPENSER_EMPTY 23
#define DISPENSER_FULL 5


#define SERVO_PWM_D9  9


//Variaveis globais
int BOWL_EMPTY    =   35;
```

```
int BOWL_FULL     =  5;// 25;
float BOWL_LVL      =   0;
float CONTAINER_LVL =   0;
String startString;
long hits = 0;
int disp_hours=11;
int disp_minutes=0;

//Flags
boolean servo_attached = false;
boolean servo_acting = false;
boolean led_status = false;

NewPing setup of pins and maximum distance for Ultrasound
Sensor.
NewPing          sonar(TRIGGER_PIN_BOWL,          ECHO_PIN_BOWL,
MAX_DISTANCE_BOWL);
NewPing    sonar2(TRIGGER_PIN_DISPENSER,    ECHO_PIN_DISPENSER,
MAX_DISTANCE_DISPENSER);

//Objeto RTC
tmElements_t time;

//Objeto do servo de rotação continua
Servo servo;

//Objeto YunServer
YunServer server;

//Comunicação I2C
int RTC_SDA_2=2;
int RTC_SCL_3=3;

int c=0;

void setup() {
```

```
pinMode(LED_D4,OUTPUT);
Serial.begin(115200);

// Inicialização do Servo
servo.attach(SERVO_PWM_D9);
if(servo.attached() == 1)
     servo_attached=true;
else
     servo_attached=false;

//Inicialização da comunicação bridge
pinMode(13,OUTPUT);
digitalWrite(13, LOW);
Bridge.begin();
digitalWrite(13, HIGH); //Acende o LED 13 no arduino assim que
a comunicação bridge for estabelecida

server.listenOnLocalhost();
server.begin();
}

void loop() {

     // Pega as requisições feitas ao servidor
     hasClients();

     if(!servo_attached)
     {
          Serial.println(" Problems to attach the servo... ");
     }

     if(chk_lvl(TRIGGER_PIN_DISPENSER) < .3)
     {
          digitalWrite(LED_D4, HIGH);
          led_status = true;
     }
```

```
     else //Aciona o LED
     {
          digitalWrite(LED_D4, LOW);
          led_status = false;
     }

     if(RTC.read(time))
     {
          Serial.println("RTC Ready");
          Serial.print("SET TIME: ");
          Serial.print(disp_hours);
          Serial.print(":");
          Serial.println(disp_minutes);

          Serial.print("RTC: ");
          Serial.print(time.Hour);
          Serial.print(":");
          Serial.println(time.Minute);

          if(time.Hour          ==          disp_hours          &&
          time.Minute==disp_minutes)
          {
               act(.9);
          }
     }
     else
          Serial.println("Problemas com o RTC");
}

void hasClients()
{
     YunClient client = server.accept();

     // Existe um novo cliente?
     if (client) {
          // Processa a requesição
```

```
            process(client);
            // Encerra a conexão.
            client.stop();
      }
}

//Processa a requisição do cliente
void process(YunClient client) {
      // Le o identificador da requesição
      String command = client.readStringUntil('/');
      Serial.print("Typed: ");
      Serial.println(command);

      // a requisição foi "ultrasound? ultrasound/1 -> bowl
      if (command == "ultrasound") {
            Serial.println(command);
            readUltrasound(client);
      }

      // a requisição foi "time"? time/1100 -> 11:00
      if (command == "time"){
            Serial.println(command);
            SetTime(client);
      }

      // a requisição foi "led"?
      if (command == "led") {
            Serial.println(command);
            ledStatus(client);
      }

      // a requisição foi "getfull"?
      if (command == "getfull") {
            Serial.println(command);
            getFull(client);
      }
```

```
      // a requisição foi "getempty"?
      if (command == "getempty") {
           Serial.println(command);
           getEmpty(client);
      }

      // a requisição foi "act"?
      if (command == "act") {
           Serial.println(command);
           actuateServo(client);
      }

      // a requisição foi "readServo"?
       if (command == "readServo") {
           Serial.println(command);
           readServo(client);
      }
}

// Leitura do Ultrassom
      void readUltrasound(YunClient client) {
      int sensor;
      float value;
      // Le o sensor desejado
      sensor = client.parseInt();
      chk_lvl(TRIGGER_PIN_DISPENSER);
      chk_lvl(TRIGGER_PIN_BOWL);


      // 1. Bowl / 2. Dispenser
      value = CONTAINER_LVL;
      if(sensor == 1)
      {
           value = BOWL_LVL;
      }
```

```
     if(value>1)
          value=1;
     if(value<0)
          value=0;

     // Manda um feedback para o cliente
     client.println(value);

     // Update datastore
     String key = "D";
     key += sensor;
     Bridge.put(key, String(value));
}

//Status do LED
void ledStatus(YunClient client) {
     int value;

     // Le o status do LED
     if(led_status)
          value=1;
     else
          value=0;

     // Manda um feedback para o cliente
     client.println(value);

     // Update datastore
     String key = "D";
     key += LED_D4;
     Bridge.put(key, String(value));
}


//define o nível vazio da vasilha de alimentação
```

```
void getEmpty(YunClient client) {
     int value;

     value=chk_lvl(TRIGGER_PIN_BOWL);

     BOWL_EMPTY = value;

     // Manda um feedback para o cliente
     client.print(value);
     client.println(" cm.");

     // Update datastore
     String key = "D";
     key += LED_D4;
     Bridge.put(key, String(value));
}

//define o nível cheio da vasilha de alimentação
void getFull(YunClient client) {
     int value;

     value=chk_lvl(TRIGGER_PIN_BOWL);

     BOWL_FULL = value;

     // Manda um feedback para o cliente
     client.print(value);
     client.println(" cm.");

     // Update datastore
     String key = "D";
     key += LED_D4;
     Bridge.put(key, String(value));
}

//Define horário de funcionamento
```

```
void SetTime(YunClient client) {
     int value,temp_hours,temp_minutes;

     value = client.parseInt();

     //Grava o horário
     temp_hours=value/100;
     temp_minutes=value-temp_hours*100;

     // Manda um feedback para o cliente
     if(temp_hours <= 23 && temp_minutes <= 59)
     {
          disp_hours=value/100;
          disp_minutes=value-disp_hours*100;
          client.print("O pet feeder atuara as ");
          client.print( temp_hours );
          client.print( " hora(s) e " );
          client.print( temp_minutes );
          client.println( " minuto(s)." );
     }
     else
     {
          client.println("Valor invalido, horas entre 0 e 23 e
          minutos entre 0 e 59, tente novamente.");
     }

     // Update datastore
     String key = "D";
     key += LED_D4;
     Bridge.put(key, String(value));
}

//Le o estado do servo, se em rotação ou parado
void readServo(YunClient client) {
     int value;
```

```
     if(servo_acting)
          value=1;
     else
          value=0;

     // Manda um feedback para o cliente
     client.println(value);

     // Update datastore
     String key = "D";
     key += LED_D4;
     Bridge.put(key, String(value));
}

// Atua o servo
void actuateServo(YunClient client) {

     float perc;

     perc = client.parseFloat();
     Serial.println(perc);

     if(perc>1)
          perc=0.9;
     if(perc<0)
          perc=0;

     act(perc);

}

void act(float perc)         //Atua o servo ate que o nível de
comida na vasilha atinja o percentual perc
{
     while(chk_lvl(TRIGGER_PIN_BOWL) < perc)
     {
```

```
          if(chk_lvl(TRIGGER_PIN_BOWL) < perc)
          {
               Serial.print("lvl: ");
               Serial.println(chk_lvl(TRIGGER_PIN_BOWL));
               servo.write(180);
               servo_acting=true;
               hasClients();
          }
          else
               break;
     }
     servo.write(90);
     servo_acting=false;
}

float chk_lvl(int sensor) //Checa o nível de comida no sensor
{
     //Initialize the variables
     int    range_inf    =    DISPENSER_EMPTY,    range_sup    =
     DISPENSER_FULL;
     if(sensor==TRIGGER_PIN_BOWL)
          int range_inf=BOWL_EMPTY, range_sup=BOWL_FULL;
     float          x=(pingSensor(sensor)-range_sup)/(range_inf-
     range_sup);

     if(sensor==TRIGGER_PIN_BOWL)
     {
          BOWL_LVL=1-x;
          return BOWL_LVL;
     }
     CONTAINER_LVL=1-x;
     return CONTAINER_LVL;
}

float pingSensor(int sensor)
{
```

```
    if(sensor == TRIGGER_PIN_BOWL)
    {
        delay(50);
        unsigned int uS = sonar.ping(); // Send ping, get
        ping time in microseconds (uS).
        BOWL_LVL=sonar.convert_cm(uS);
        return sonar.convert_cm(uS);
    }

    delay(50);
    unsigned int uS = sonar2.ping(); // Send ping, get ping
    time in microseconds (uS).
    Serial.print("Ping: ");
    Serial.print(sonar2.convert_cm(uS)); // Convert ping time
    to distance and print result (0 = outside set distance
    range, no ping echo)
    Serial.println("cm");
    CONTAINER_LVL=sonar2.convert_cm(uS);
    return sonar2.convert_cm(uS);
}
```

Apêndice B – Manual de inicialização do protótipo.

O primeiro passo para se iniciar o protótipo pela primeira vez é configurar a comunicação wireless do mesmo. Para tanto o seguinte passo a passo deve ser seguido:

1. Ao conectar o sistema a rede elétrica, abra algum dispositivo eletrônico que permita acesso a um navegador e conexão wifi.
2. Procure pela rede ArduinoYun-XXXXXXXXXXXX e se conecte a mesma.
3. Uma vez conectado, abra o navegador e entre no endereço: http://arduino.local, digitando a senha “arduino” quando solicitado. Assim que o *login* for realizado aperte o botão “*configure*”.
4. Na página aberta vá até wireless parameters e selecione a sua rede wifi, digite a senha da sua rede e então clique em “*configure and restart*”.

Após essa configuração você já pode retornar a sua rede doméstica e começar o segundo passo da configuração.

1. Primeiro posicione a vasilha de alimentação vazia do seu animal de estimação embaixo do sensor de forma que a mesma fique centralizada ao sensor.
2. Va ate o seu navegador e digite: 192.168.1.117/arduino/getempty/1
3. Após encha a vasilha com ração e digite: 192.168.1.117/arduino/getfull/1
4. E por fim digite: 192.168.1.117/arduino/time/XXXX. Onde XXXX é o horário que você deseja que o sistema atue, por exemplo 1530 para 15 horas e 30 minutos.

Encerrado o processo de calibração o seu sistema esta pronto para ser usado. Você pode abrir o ScadaBr e monitorar os níveis de ração e operações do sistema através do mesmo e sempre que necessário é só utilizar o ultimo passo descrito a cima para programar um novo horário de atuação do sistema.