**Fluxograma de Software:**

1. Adquirir os componentes de software:

Sensor de carga;

Reles de comutação;

Microcontrolador;

Botões de posição da vassoura;

2. Definir parâmetros do ADC em função do sensor de carga;

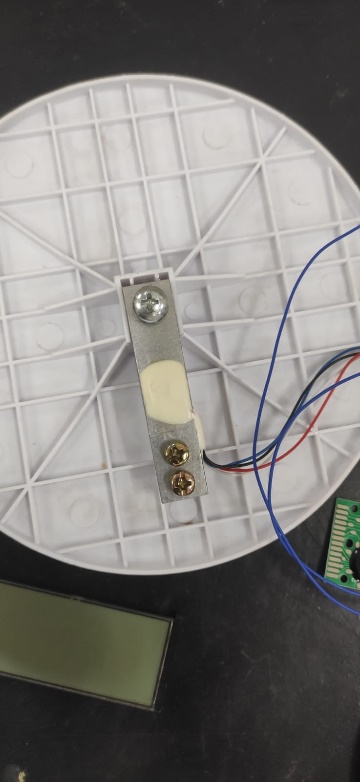
3. Criar código para funcionamento do sistema microcontrolado;

4. Testar sensores, botões e microcontrolador sem hardware;

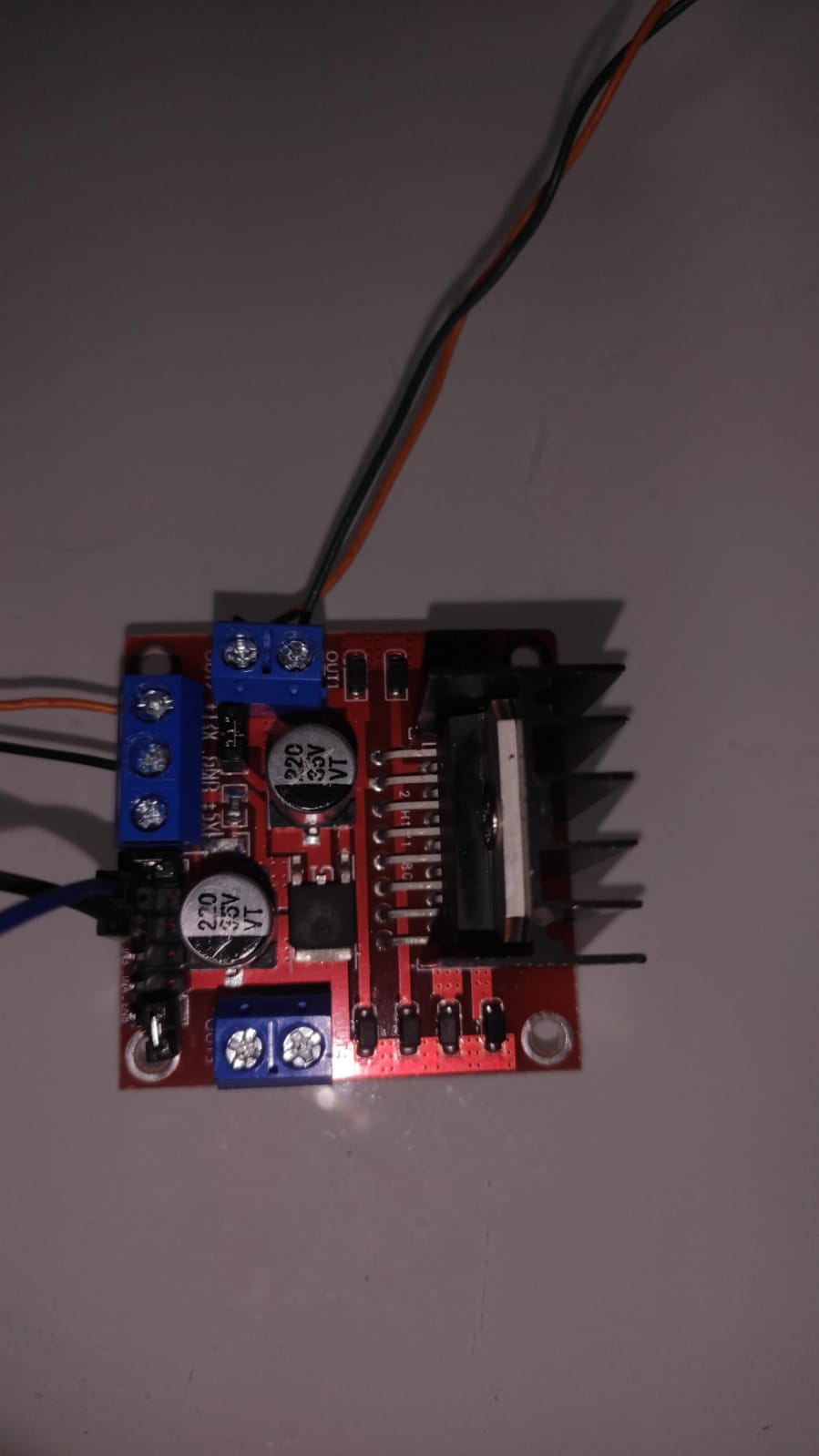
5. Documentar.

Já com o hardware pronto, comecei a aquisição dos materiais necessários para implementação do software.

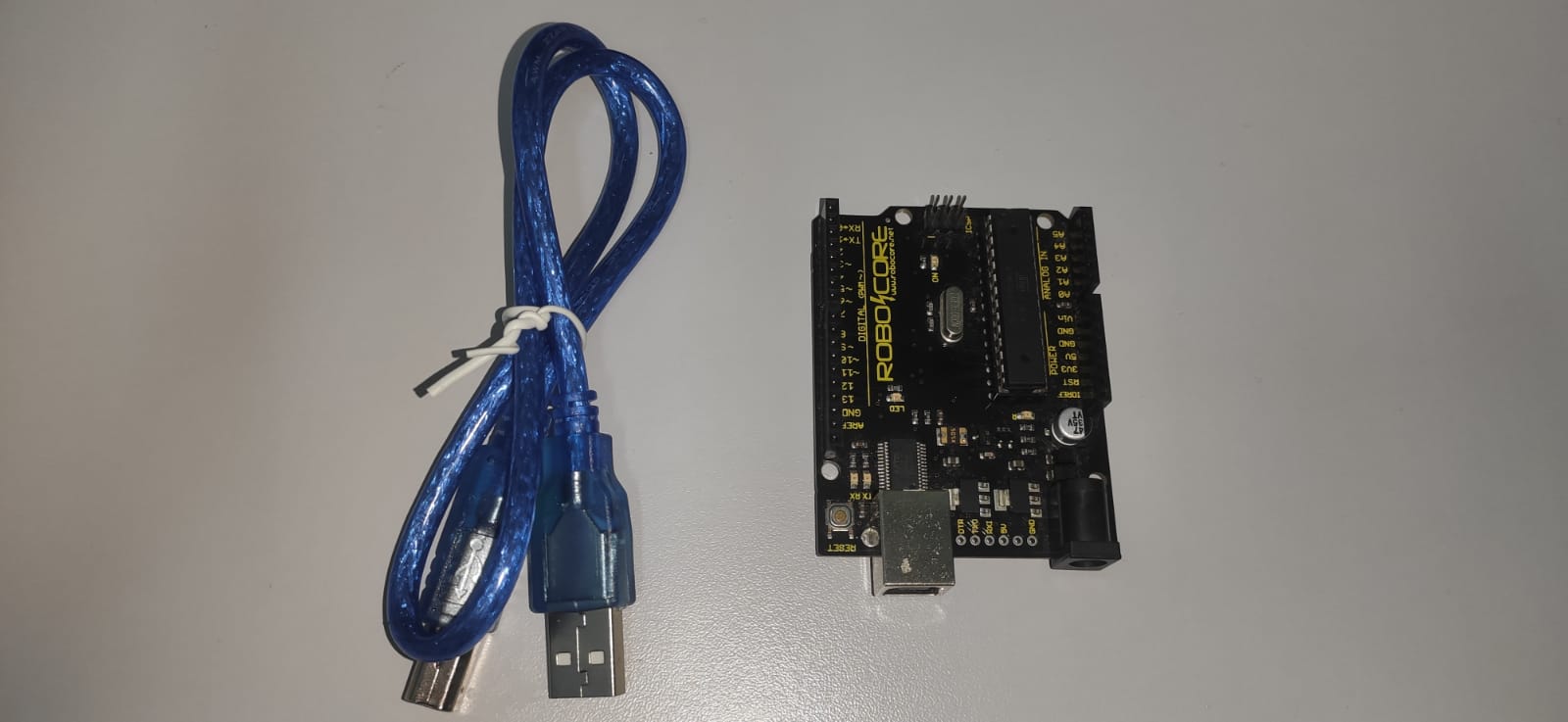
Primeiramente, fiz a aquisição de uma balança de até 10Kg de carga, mais do que o suficiente para este protótipo.



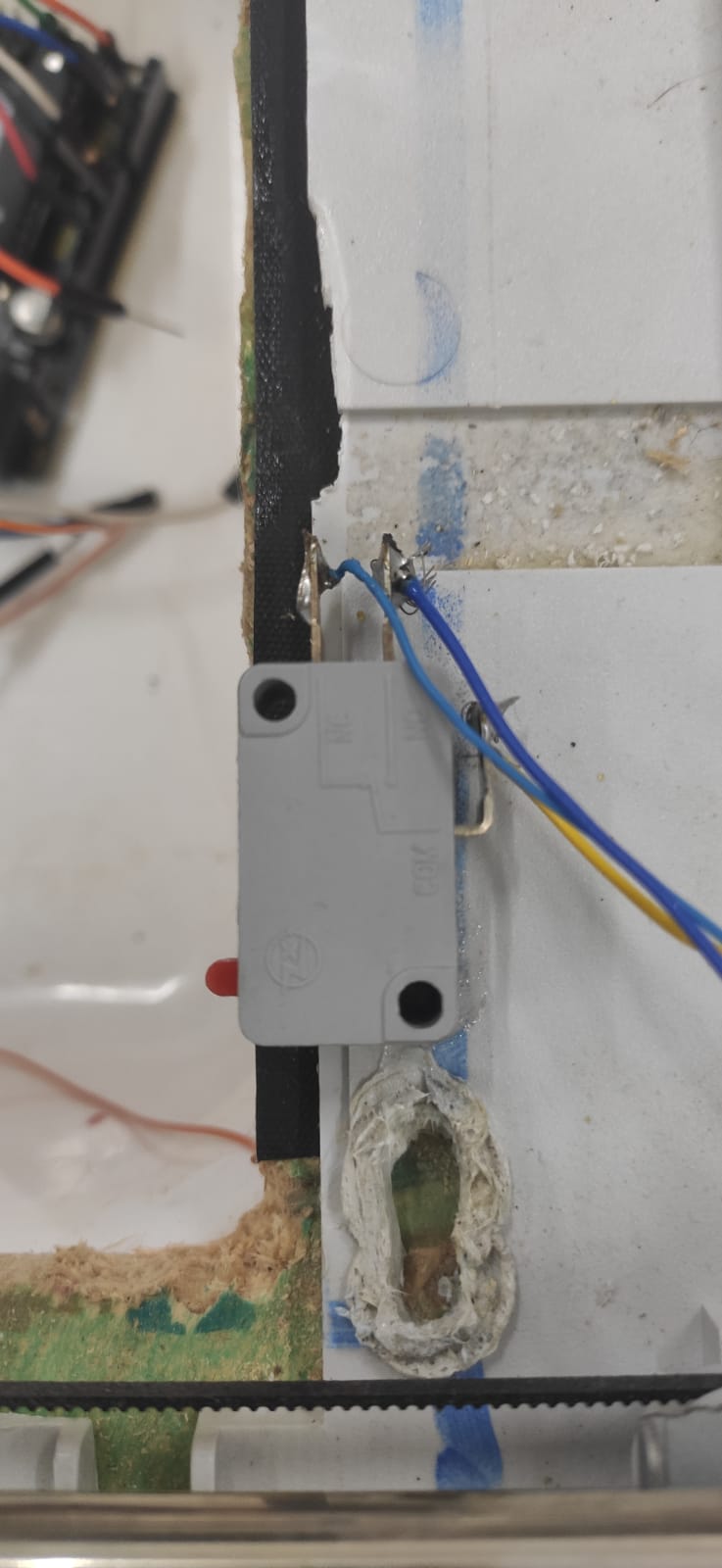
Para realizar o acionamento e a reversão do motor DC de 12V, escolhi a ponte-H modelo L298N disponível em loja física.



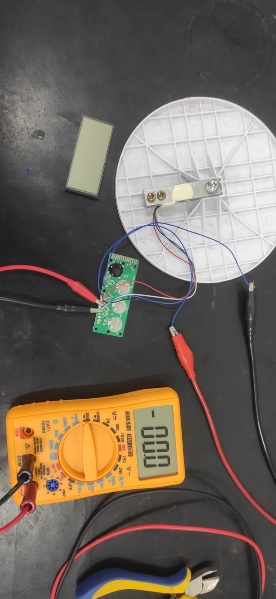
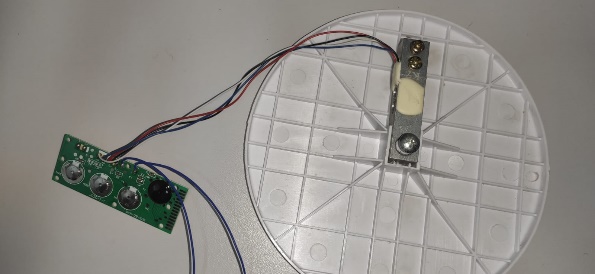
Optei por utilizar um microcontrolador AVR328, que já possuo e é suficiente para o protótipo proposto.



Os botões para delimitar os extremos da posição da vassoura, utilizei os mesmos que compõe o sistema da porta de um forno de microcondas.



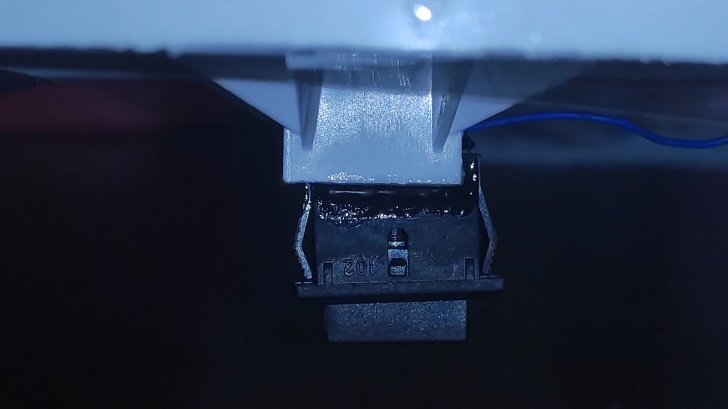
Com teste em bancada, defini que o sensor de carga varia aproximadamente 1mV por quilo, como não há necessidade de precisão, este sensor irá detectar presença, deverá funcionar perfeitamente.



A partir daqui, comecei a construção do código para programação do sistema. Imediatamente percebi que a utilização de um ADC para detectar somente a presença do gato sobre a caixa de areia não era necessária. Para o propósito um sinal binário seria o suficiente.

O projeto inicial da utilização de um medidor de peso seria a verificação da quantidade de areia que ainda resta na caixa. Para este fim teríamos que especificar o tipo de areia a ser utilizada, já que a silica, tem massa muito diferente da betonita que também é muito diferente de uma areia fina de farelos de vegetais, enfim, neste momento não se tornou interessante a utilização deste sensor.

Substitui o sistema por algo mais simples e mais eficaz, um botão.



O código de programação passou por diversos ajustes, até chegar na solução que considero satisfatória e que preve um funcionaento simples.

#define F\_CPU 16000000UL

#include <avr/io.h>

#include <util/delay.h>

#include <avr/interrupt.h>

#include "funsape/globalDefines.hpp"

#include "funsape/peripheral/timer1.hpp"

#define MOTOR\_FORWARD\_PIN PC1

#define MOTOR\_REVERSE\_PIN PC2

#define BUTTON1\_PIN PD2

#define BUTTON2\_PIN PD3

#define BUTTON3\_PIN PD4

#define TIMER\_INTERVAL\_MS 64 // Tempo do intervalo do timer em milissegundos

volatile bool timer\_done = false;

volatile bool motor\_on = false;

volatile bool motor\_reverse = false;

volatile bool estadoBotao1 = false;

volatile bool estadoBotao2 = false;

volatile bool estadoBotao3 = false;

volatile uint16\_t timer\_counter = 0; // Contador de interrupções do timer

void timer1CompareACallback(void)

{

    timer\_counter++;

    if(timer\_counter >= (10000 / TIMER\_INTERVAL\_MS)) {

        timer\_done = true;

        timer\_counter = 0; // Reseta o contador do timer

    }

}

void setup()

{

    // Configura os pinos do motor como saída

    setBit(DDRC, MOTOR\_FORWARD\_PIN);

    setBit(DDRC, MOTOR\_REVERSE\_PIN);

    // Configura os botões como entrada

    clrBit(DDRD, BUTTON1\_PIN);

    clrBit(DDRD, BUTTON2\_PIN);

    clrBit(DDRD, BUTTON3\_PIN);

    // Ativa pull-up nos botões

    setBit(PORTD, BUTTON1\_PIN);

    setBit(PORTD, BUTTON2\_PIN);

    setBit(PORTD, BUTTON3\_PIN);

    // Configura o timer para gerar interrupções a cada 64 ms

    timer1.init(Timer1::Mode::CTC\_OCRA, Timer1::ClockSource::PRESCALER\_1024);

    timer1.setCompareAValue((F\_CPU / 1024 / 1000) \* TIMER\_INTERVAL\_MS - 1);

    timer1.deactivateCompareAInterrupt();

    sei(); // Habilita interrupções globais

}

void loop()

{

    // Verifica o estado do botão 1 (PD2)

    if(!(PIND & (1 << BUTTON1\_PIN))) {

        while(!(PIND & (1 << BUTTON1\_PIN)))

        {

            \_delay\_ms(1); // Anti-repique

        }

        // if(!(PIND & (1 << BUTTON1\_PIN)) && !estadoBotao1) {

            estadoBotao1 = true;

            timer1.activateCompareAInterrupt();

            if(!motor\_on) {

                // Reinicializa o timer e o contador

                timer\_done = false;

                timer\_counter = 0;

            }

        // }

    } else {

        estadoBotao1 = false;

    }

    // Verifica se o timer terminou e se o motor ainda não está ligado

    if(timer\_done && !motor\_on) {

        // Liga o motor no sentido horário após 20 segundos

        motor\_on = true;

        setBit(PORTC, MOTOR\_FORWARD\_PIN);  // Ativa o motor para frente

        clrBit(PORTC, MOTOR\_REVERSE\_PIN);

        timer\_done = false; // Reseta a flag do timer

    }

    // Verifica o estado do botão 2 (PD3)

    if(!(PIND & (1 << BUTTON2\_PIN))) {

        \_delay\_ms(10); // Anti-repique

        if(!(PIND & (1 << BUTTON2\_PIN)) && !estadoBotao2) {

            estadoBotao2 = true;

            if(motor\_on && !motor\_reverse) {

                // Desativa o motor para frente e ativa o reverso

                clrBit(PORTC, MOTOR\_FORWARD\_PIN);

                setBit(PORTC, MOTOR\_REVERSE\_PIN);

                motor\_reverse = true;

            }

        }

    } else {

        estadoBotao2 = false;

    }

    // Verifica o estado do botão 3 (PD4)

    if(!(PIND & (1 << BUTTON3\_PIN))) {

        \_delay\_ms(10); // Anti-repique

        if(!(PIND & (1 << BUTTON3\_PIN)) && !estadoBotao3) {

            estadoBotao3 = true;

            timer1.deactivateCompareAInterrupt();

            if(motor\_reverse || motor\_on) {

                // Desliga o motor e reseta o sistema

                clrBit(PORTC, MOTOR\_FORWARD\_PIN);

                clrBit(PORTC, MOTOR\_REVERSE\_PIN);

                motor\_on = false;

                motor\_reverse = false;

                // Reseta os botões e o timer

                timer\_done = false;

                timer\_counter = 0;

            }

        }

    } else {

        estadoBotao3 = false;

    }

}

int main()

{

    setup();

    while(1) {

        loop();

    }

    return 0;

}

O funcionamento ocorre quando o gato sai da caixa, ou seja, o botão 1 vai de 0 para 1, disparando um timer para posterior acionamento do motor. Se o botão ficar pressionado ou se for pressionado diversas vezes dentro do tempo programado no timer, irá valer o timer referente a última mudança de estado do botão 1 de 0 para 1.

O que garante que o motor não seja acionado enquanto o gato está dentro da caixa é um laço while que sempre estará em loop até que ocorra a mudança de estado do botão 1 de nível baixo para alto, ressaltando que retorna para o laço caso o botão volte para o nível baixo enquanto ainda estiver correndo o tempo do timer.

Após o acionamento, como foi utilizado uma ponte H temos duas conexões entre ela e o AVR328, nesse caso se as duas saidas estiverem iguais o motor fica parado e nos dois casos possíveis das saídas distintas temos rotação para um lado ou para o outro do motor DC, o sistema em movimento aguarda o toque no botão 2 que ao alterar seu estado de 1 para 0, altera essas saídas para que o motor gire no sentido oposto.

Em movimento de retorno o sistema aguarda a mudança de estado do botão 3, de 1 para 0 e assim igualando as duas saídas, parando o motor, desligando o contador e colocando o sistema novamente em espera.

Em simulação no Proteus tudo ocorreu na forma esperada, assim como no sistema real após a gravação do AVR328 com sensores e motor, agora o próximo passo é montar toda estrutura, e testar com o sistema a vazio.