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;

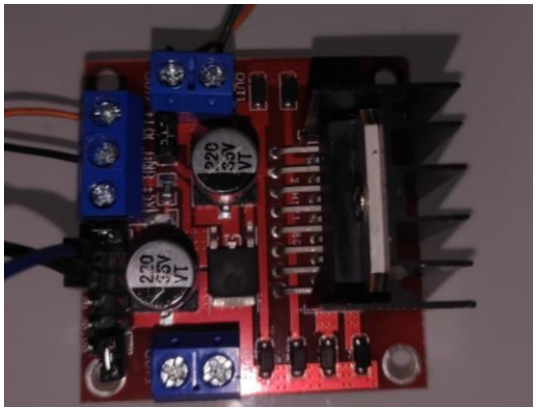
4. Testar sensores, botões e microcontrolador sem hardware;

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.

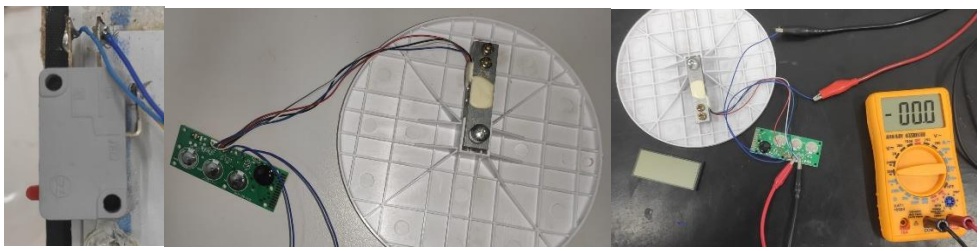
Para realizar o acionamento e a reversão do motor DC de 12V, escolhi a ponte-H modelo L298N disponível em loja física.



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 micro-ondas.

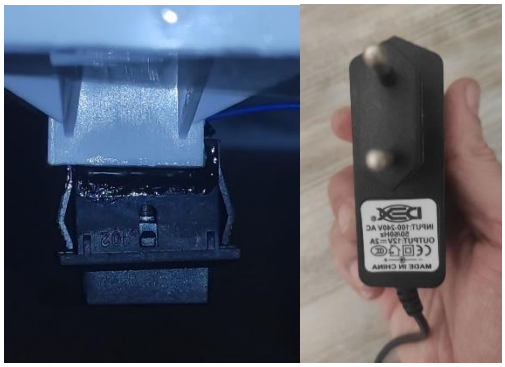
Com teste em bancada, defini que o sensor de carga varia aproximadamente 1mV por quilo, como não há necessidade de precisão, este sensor irá detectar presença, deverá funcionar perfeitamente.



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.



A fonte de alimentação de 12V serve tanto para o microcontrolador AVR328, quanto para a ponte H, assim como também para o motor.

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 "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
        }
            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 o motor.

Agora o próximo passo é montar toda estrutura, juntando hardware e software.