**PONTIFÍCIA UNIVERSIDADE CATÓLICA DO PARANÁ**

**CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA**

**ENGENHARIA DA COMPUTAÇÃO**

# PROJETO INTEGRADO

# SEMÁFORO INTELIGENTE

**CURITIBA**

**2010**

**GILBERTO JOSÉ VERONA FILHO**

**IGOR PLÁCIDO BERTUOL NEGRÃO**

**Projeto Integrado: Semáforo Inteligente**

Este documento será apresentado ao curso de Engenharia da Computação do Centro de Ciências Exatas e de Tecnologia da Pontifícia Universidade Católica do Paraná, como parte integrante da nota do primeiro semestre na disciplina de Microprocessadores I.

**Professores-orientadores:**

Afonso Ferreira Miguel

**CURITIBA**

**2010**

# SUMÁRIO

# 1. INTRODUÇÃO

Com o aumento quase exponencial presenciado nos últimos anos em relação à venda de carros novos no Brasil nos deparamos com um problema que, ainda já previsto, não se imaginava que seria tão próximo, o caos instaurado no trânsito brasileiro.

As grandes metrópoles estão sofrendo com o significativo aumento de carros em circulação. Várias medidas já foram tomadas nos últimos anos, porém, a maioria delas baseia-se em idéias inviáveis economicamente ou geograficamente, muitas vezes impraticáveis no cotidiano de muitas cidades.

Como opções já foram instaurados sistemas de controle de trafego de carros, como o realizado por São Paulo já a alguns anos, o Rodízio de Placas, que visa permitir a circulação de carros em determinados dias de acordo com seu número de placa. Porém esta e outras opções acabam sendo de emergência e muitas vezes não geram resultados, visto que muitas pessoas acabam comprando outros carros com placas diferenciadas para poder andar no dia-a-dia normalmente.

Uma opção real e de grande resultado é a implementação de um controle de tráfego que realmente funcione, em São Paulo tem-se um exemplo de um bom controle, porém, muitas cidades presenciam uma total falta de sincronismo entre semáforos e má utilização destes em alguns cruzamentos, como o caso de Curitiba, em que é possível “perder” vários minutos esperando os vários semáforos de uma rua abrir por falta de sincronismo.

Além destes problemas, existe um de suma importância em locais de baixa segurança, que é a espera ociosa em cruzamentos.

Visando melhorar o sistema de tráfego em geral, pensamos em produzir um semáforo capaz de gerenciar o seu próprio tempo de espera e prioridades de uma maneira fácil e eficiente.

# 2. OBJETIVO

O sistema elaborado prevê o tempo de espera ocioso e tenta concertar isto de uma maneira efetiva e prática, por exemplo, se existe um carro parado em um cruzamento, e não existem carros passando nem se aproximando do semáforo oposto, o sistema é capaz de detectar isto e converter o semáforo de forma a permitir que o carro que está aguardando possa passar e logo após isso, o sistema retorna a seu ciclo normal.

Além disso, o sistema também foi elaborado prevendo a necessidade da travessia de pedestres, seja ela por preempção de tempo ou somente quando solicitado, como quando o pedestre aciona o botão, por exemplo.

Está além do escopo deste projeto a elaboração de sincronismo entre semáforos, para tal seria necessária a utilização de diversos sistemas do mesmo porte e um projeto de monitoramento dos mesmos.

# 3. DESCRIÇÃO

Para a elaboração deste projeto, utilizou-se como base a plataforma Arduino que se trata de uma plataforma de hardware livre, projetada com um microprocessador de placa única com suporte de I/O – entrada/saída – embutido e uma determinada linguagem para a programação do mesmo.

Neste projeto, utilizou-se especificamente um Arduino Duemilanove baseado em um microprocessador ATmega328.

Toda a programação utilizada foi desenvolvida na linguagem própria do Arduino, esta baseada em C/C++, e em ambiente de programação e compilador próprio da plataforma – Arduino Alpha.

Para simulação e apresentação do projeto, determinou-se um cruzamento em que os semáforos são sincronizados de dois em dois, opostamente, e que o carro possa apenas seguir no sentido em que se encontra ou dobrar a sua direita, como apresentado na Figura 1, e que os pedestres tenham prioridade em relação aos carros, esperando apenas o ciclo padrão terminar para que a travessia seja possível.

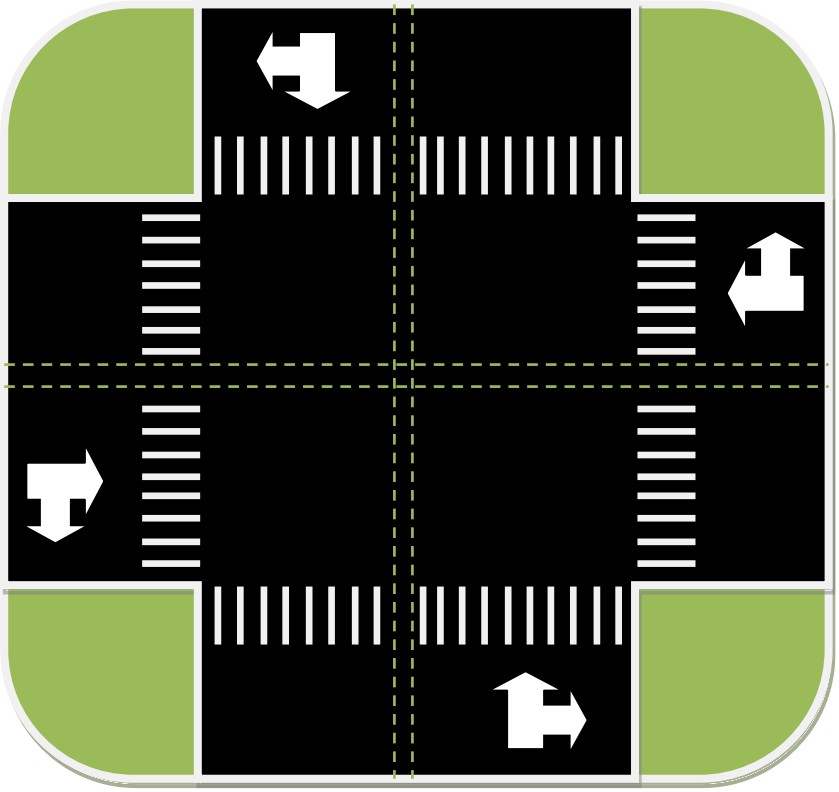


Figura 1 – Modelo de Cruzamento Escolhido.

## 3.1. PROJETO LÓGICO

O software embarcado neste projeto foi elaborado pensando em facilidade para a implementação dos mais diversos tipos de cruzamentos que podem ser necessários quando se trata do uso de semáforos. Portando, buscou-se realizar uma programação tendo como base orientação a objeto, porém como o código de demonstração não seria tão complexo, no fim o programa apresenta diversas funções e estratégias visando deixar a área de programa principal – loop – somente com chamadas de funções e leituras que não são realizáveis fora da mesma área.

Com a definição principal do programa focada à utilização no cruzamento determinado anteriormente – 2 x 2 com prioridade de pedestres – algumas condições necessárias para o bom funcionamento de prioridades tornaram necessária a implementação de código, que não função, no loop principal, porém nada que prejudique o código final e sua facilidade de adaptação.

Sendo assim, o loop principal ficou caracterizado de acordo com o diagrama da Figura 2 a seguir:

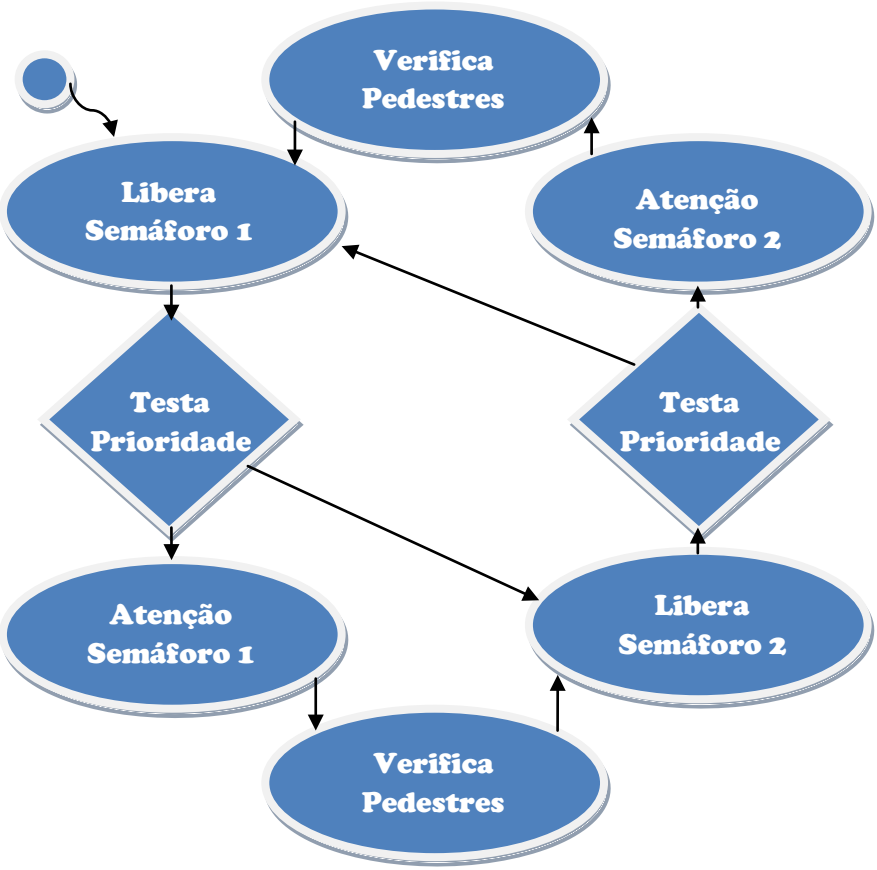



Figura 2 – Diagrama de Blocos do Loop Principal do Software.

De uma maneira básica, o programa prevê as funções de Liberação do Semáforo 1 – a qual além de liberar este semáforo 1 também já bloqueia o semáforo 2, Atenção Semáforo 1 – esta deixa o semáforo 1 em estado de alerta antecedendo seu bloqueio, Verificação de Pedestres – realiza a verificação do acionamento de um botão de pedestre, para tal utiliza uma interrupção(interruptAttack), a qual aciona uma flag neste instante, Liberação Semáforo 2 – que realiza o oposto do Liberação Semáforo 1, e Atenção Semáforo 2 – com a mesma função do Atenção Semáforo 1, porém para o semáforo 2.

O Teste de Prioridade não foi determinado como uma função, pois o mesmo utiliza do recurso *goto* proveniente da linguagem C, com o qual é possível "mandar" o programa para um local desejado com extrema facilidade, porém com os devidos cuidados para evitar problemas como um loop infinito.

O Teste de Prioridade tem como objetivo testar se o semáforo está sendo bem utilizado, por exemplo, caso o semáforo 1 esteja aberto, porém não há fluxo nele, e há um carro esperando ou chegando próximo ao semáforo 2, o Teste de Prioridade determina que deve-se abrir o semáforo 2 e fechar o 1 mesmo que para isso tenha que interromper o ciclo habitual. Além disso, o teste também prevê se há necessidade de manter o fluxo estabelecido por padrão, ou pode-se deixar apenas uma das vias abertas visando facilitar o fluxo de carros.

Mais detalhes sobre as funções e o Teste de Prioridades podem ser obtidos através do código original em anexo.

## 3.2. PROJETO FÍSICO

O projeto físico consiste basicamente em uma PCI - placa de circuito impresso - contendo LEDs e botões visando facilitar a visualização do sistema de tráfego com base com o que ficou estabelecido pelo projeto lógico. Ou seja, a PCI foi elaborada de maneira com que facilitasse a visualização dos LEDs dos semáforos, sejam de carros ou pedestres, e com fácil acesso para simulação do botão de pedestre.

O projeto ideal para representar este sistema de tráfego com semáforos seria como o sugerido na Figura 3, este contem todos os semáforos das quatro vias, bem como todos os dos pedestres, além de quatro botões, sendo um para cada via de pedestres e um par de sensores infravermelhos para cada via de passagem de carros.

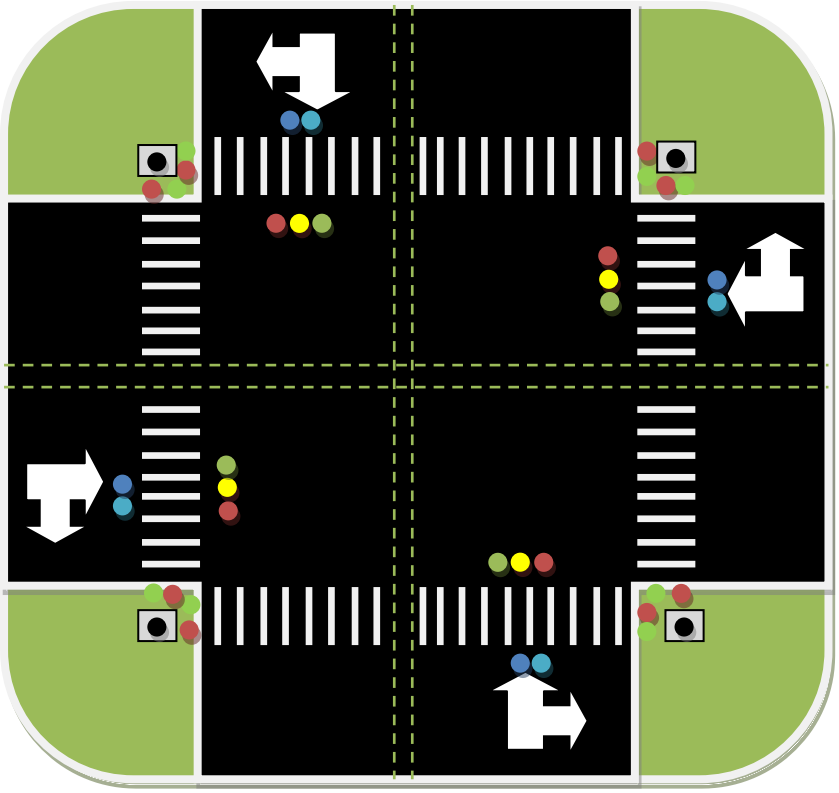


Figura 3 – Projeto de Cruzamento Ideal.

Porém para facilitar a confecção da PCI sem prejuízos visuais ou logísticos, preferiu-se retirar três dos quatro botões de pedestres existentes – visto que o projeto trata os pedestres como prioritários e, portanto, todos os semáforos de carros são bloqueados quando os dos pedestres são liberado – fazendo assim com que apenas um único botão fosse implementado. Além disto, preferiu-se adotar apenas dois

sensores infravermelhos detectores de presença de carros, já que o projeto lógico prevê, neste caso, que os semáforos opostos estão sempre no mesmo estado, portanto, o cruzamento torna-se espelhado pela metade.

Assim, a PCI confeccionada segue o design de acordo com a Figura 4, mostrando como ficará a organização dos componentes na superfície de vista superior da PCI.

A PCI não prevê local para suporte do Arduino controlador do sistema, apenas prevê um barramento de ligação para tal. Através deste barramento, a ligação com o Arduino se dá com um cabo flat.

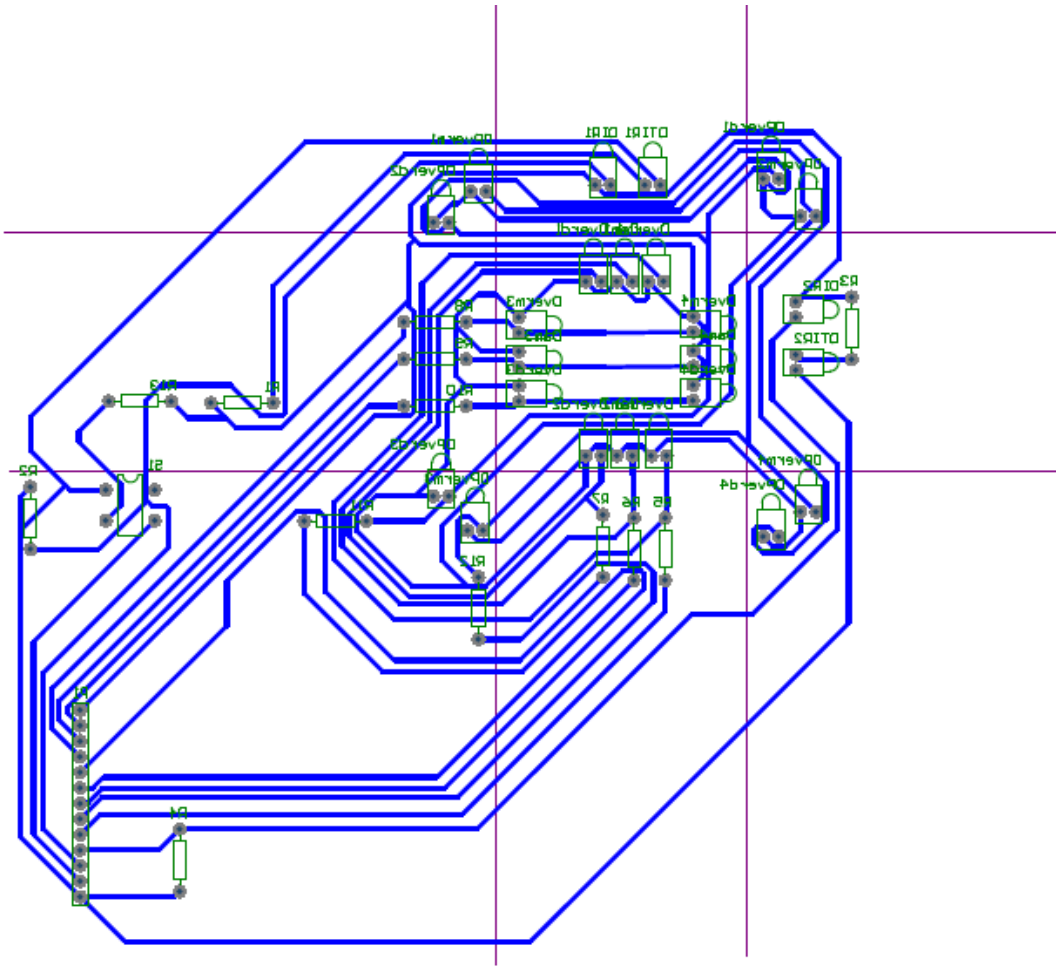


Figura 4 – Projeto Físico da PCI

A alimentação do sistema é feita pelo próprio Arduino, porém, para a alimentação do Arduino foi prevista a utilização de uma bateria de 9V, ou a alimentação pela própria porta USB presente na plataforma.

O diagrama elétrico da PCI, mostrando os componentes e ligações utilizadas, pode ser visualizado na Figura 5.

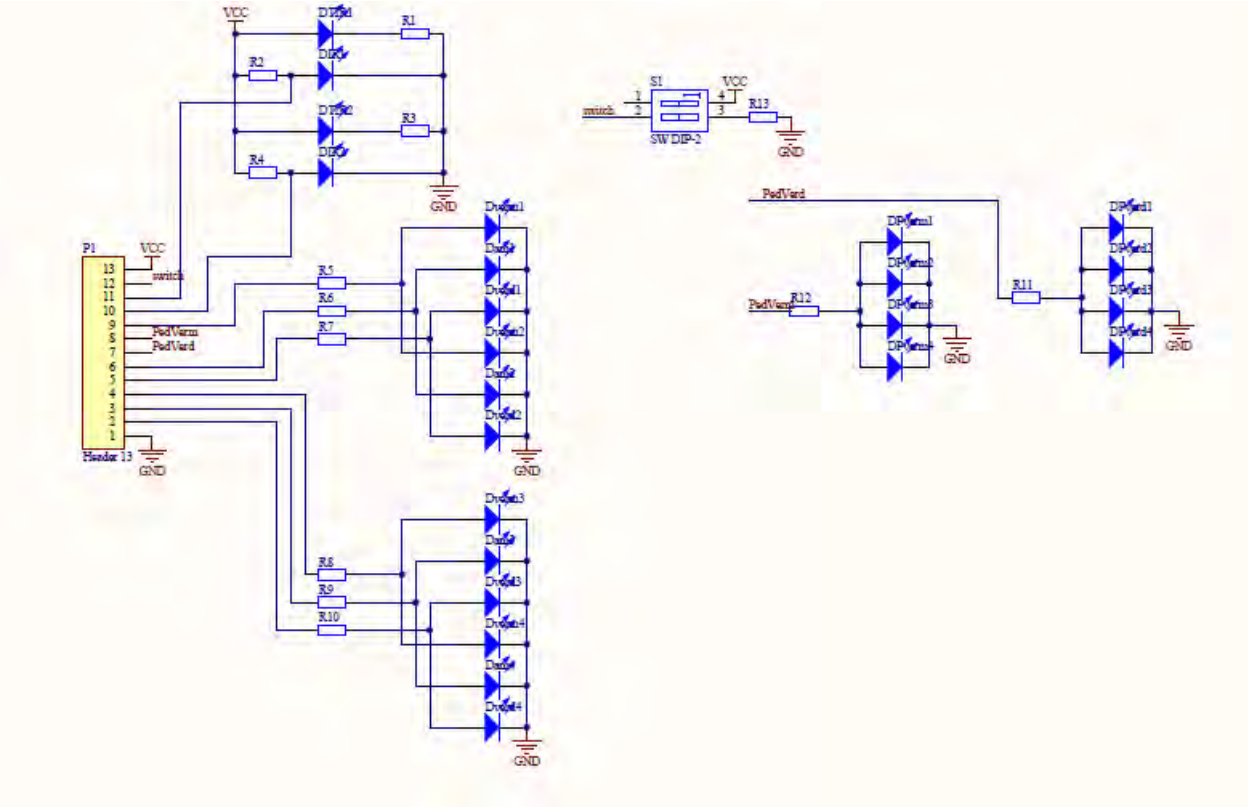



Figura 5 – Diagrama Elétrico da PCI.

Como detalhado na Figura 5, os componentes da PCI são:

- ➔ 8 Resistores de 270Ω - R1, R3, R5, R6, R7, R8, R9 e R10
- ➔ 2 Resistores de 100Ω - R11 e R12
- ➔ 2 Resistores de 10kΩ - R2 e R4
- ➔ 1 Resistor de 1kΩ - R13
- ➔ Botão DIP6
- ➔ 8 LEDs Vermelhos
- ➔ 8 LEDs Verdes
- ➔ 4 LEDs Amarelos
- ➔ 2 LEDs IR
- ➔ 2 Transistores IR

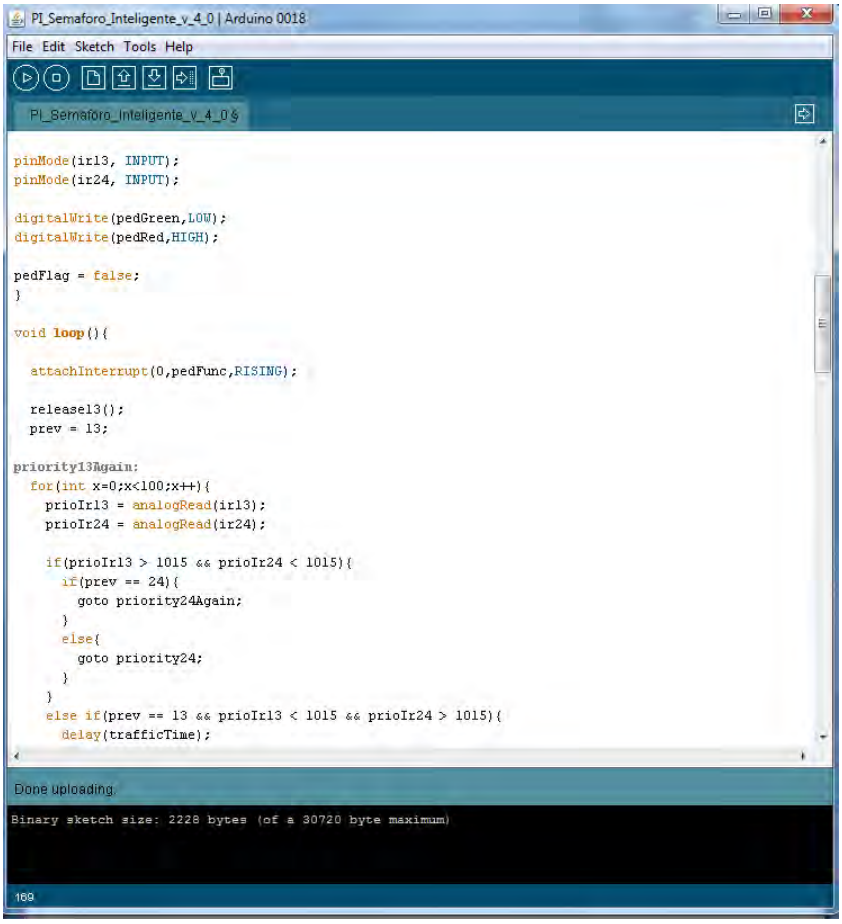
# 4. CONCLUSÃO

Para a realização deste projeto, pode-se observar que mesmo utilizando uma plataforma de processamento – Arduino – o tempo para a confecção total do projeto acaba sendo maior do que prevista. Sempre há necessidade de novos testes e, com isso, o projeto acaba sofrendo mudanças que não foram previstas, ocasionando em maior tempo para a realização do mesmo.

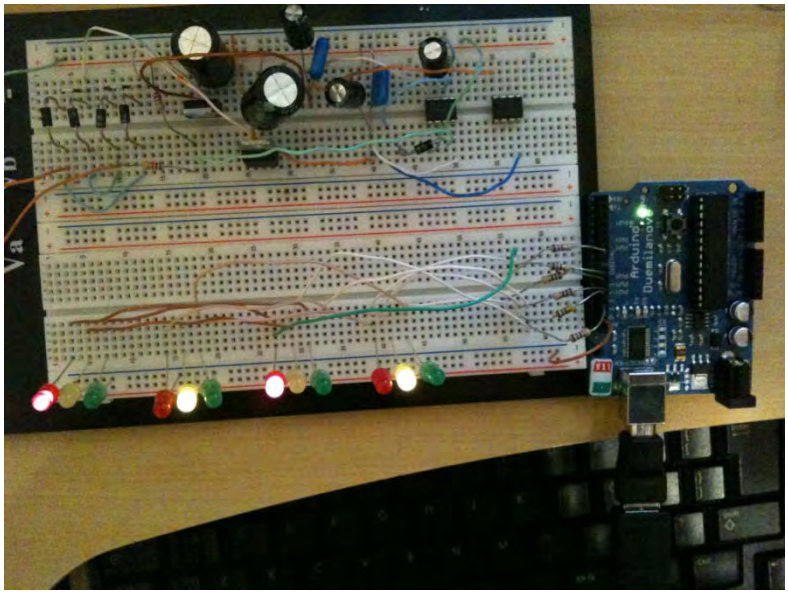
Dos problemas habituais que envolvem todos os projetos deste tipo, apenas sofremos com um de cunho acadêmico, o fato de não haver material disponível na universidade para corrosão de uma PCI maior que 15x15cm. Para tal, houve a necessidade de comprar todo o material necessário e realizar a corrosão isoladamente.

A implementação ocorreu de acordo com o planejado e o sistema simula exatamente o que foi proposto inicialmente, porém, a visualização do produto final poderia ter sido mais elaborada, mas por questão de tempo, preferiu-se um modelo funcional, mesmo que sem um design mais elaborado.

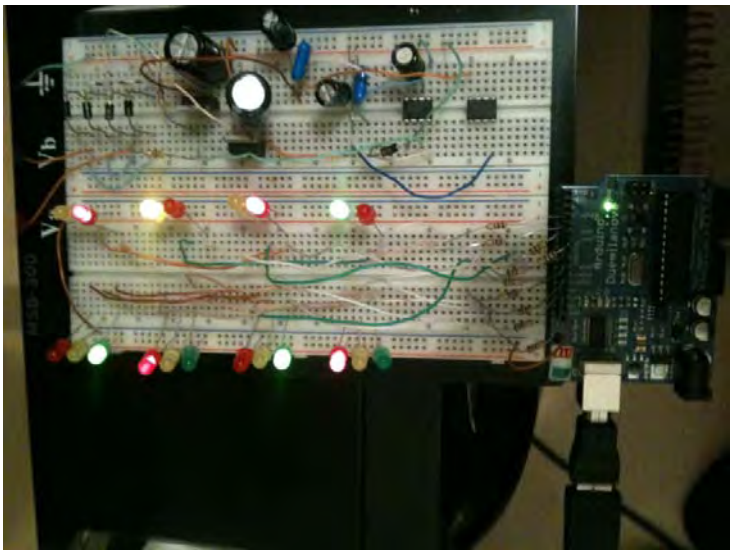
# 5. ANEXOS



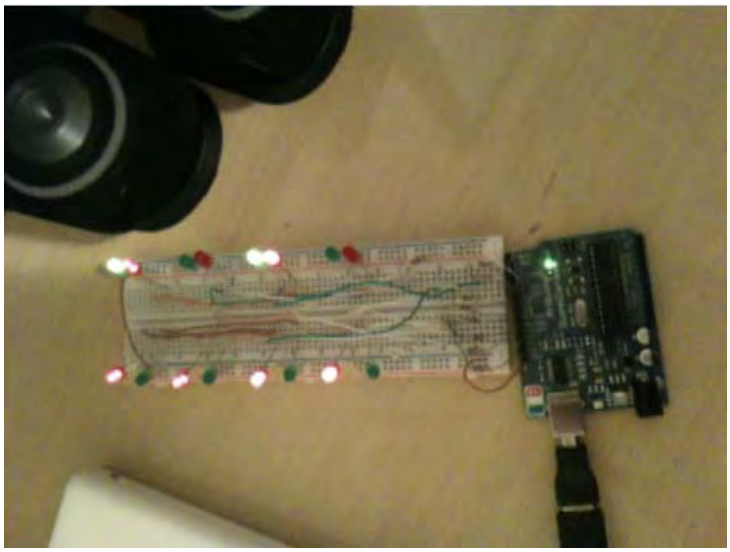

Anexo 1 – Ambiente de Programação Arduino 0018.



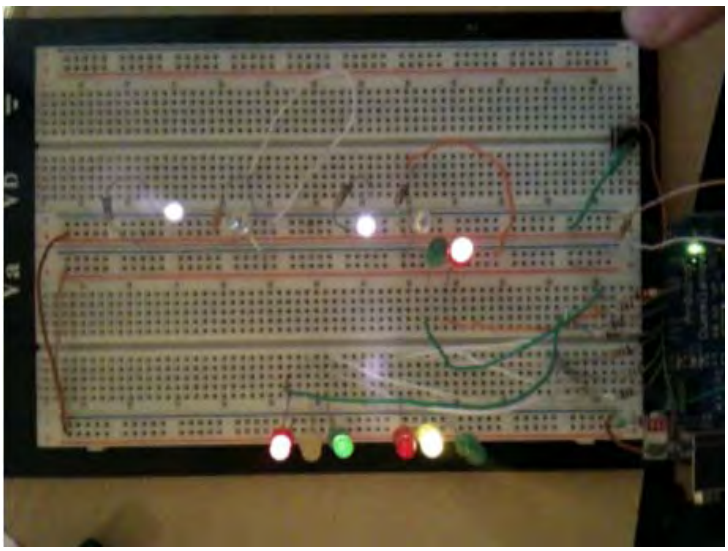
Anexo 2 – Teste da Lógica de Semáforo Simples.



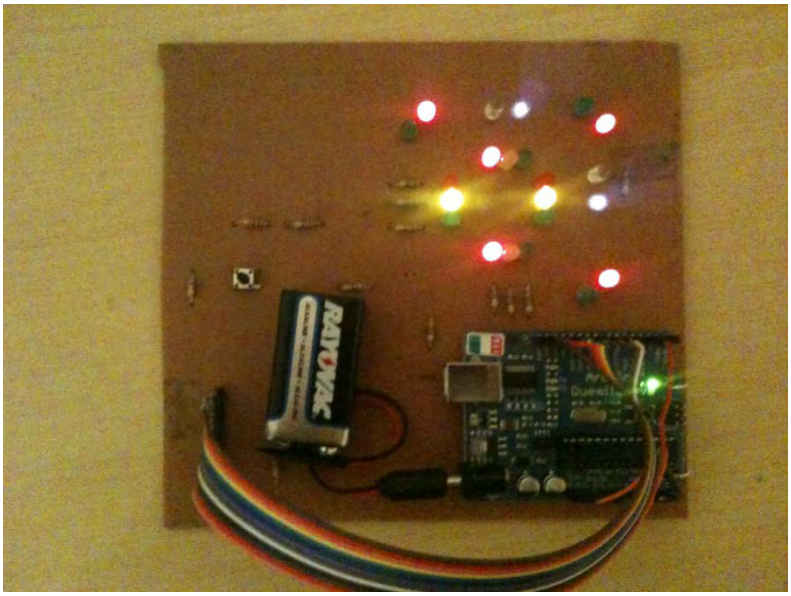
Anexo 3 – Teste da Lógica de Semáforo com Pedestres.



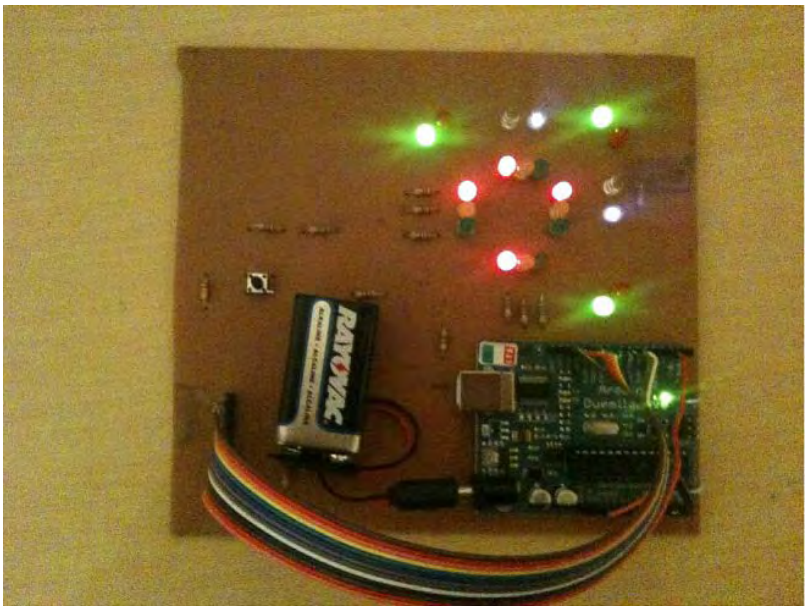
Anexo 4 – Teste Interrupção para Pedestres.



Anexo 5 – Teste do Sensor Infravermelho.




Anexo 6 – Teste Final da PCI com Arduino.




Anexo 7 – Teste Final da PCI com Arduino.

## 5.1. Programa Arduino

```
int carGreen24 = 13; // Verde 2 e 4
int carYellow24 = 12; // Amarelo 2 e 4
int carRed24 = 11; // Vermelho 2 e 4

int carGreen13 = 6; // Verde 1 e 3
int carYellow13 = 9; // Amarelo 1 e 3
int carRed13 = 10; // Vermelho 1 e 3

int pedGreen = 8; // Pedestre Verde
int pedRed = 7; // Pedestre Vermelho

int pedButton = 2; // Botão de Pedestres

int passTime = 4000; //Tempo Travessia Pedestre
int warningTimePed = 1000/4; //Tempo de Atenção Pedestre
int warningTimeCar = 1000; //Tempo de Atenção Carros
int trafficTime = 4000/100; //Tempo de Alternancia de Trafego

int pedFlag; //Testa se Botão de Pedestre foi Acionado

int ir13 = 4;  //IR 13
int ir24 = 3;  //IR 24

int prioIr13;
int prioIr24;

int prev;


void setup() {
pinMode(carYellow24, OUTPUT);
pinMode(carYellow13, OUTPUT);
pinMode(carGreen24, OUTPUT);
pinMode(carGreen13, OUTPUT);
pinMode(carRed24, OUTPUT);
pinMode(carRed13, OUTPUT);
pinMode(pedRed, OUTPUT);
pinMode(pedGreen, OUTPUT);
pinMode(pedButton, INPUT);

pinMode(ir13, INPUT);
pinMode(ir24, INPUT);

digitalWrite(pedGreen,LOW);
digitalWrite(pedRed,HIGH);

pedFlag = false;
}

void loop(){

  attachInterrupt(0,pedFunc,RISING);

  release13();
  prev = 13;

priority13Again:
  for(int x=0;x<100;x++){
    prioIr13 = analogRead(ir13);
    prioIr24 = analogRead(ir24);

    if(prioIr13 > 1015 && prioIr24 < 1015){
      if(prev == 24){
```

```
        goto priority24Again;
      }
      else{
        goto priority24;
      }
    }
    else if(prev == 13 && prioIr13 < 1015 && prioIr24 > 1015){
      delay(trafficTime);
      goto priority13Again;
    }
    delay(trafficTime);
  }

priority24:
  warning13();

  pedTest();

  release24();
  prev = 24;

priority24Again:
  for(int x=0;x<100;x++){
    prioIr13 = analogRead(ir13);
    prioIr24 = analogRead(ir24);

    if(prioIr13 < 1015 && prioIr24 > 1015){
      if(prev == 13){
        goto priority13Again;
      }
      else{
        goto priority13;
      }
    }
    else if(prev == 24 && prioIr13 > 1015 && prioIr24 < 1015){
      delay(trafficTime);
      goto priority24Again;
    }
    delay(trafficTime);
  }

priority13:
  warning24();

  pedTest();
}

void release13(){
  digitalWrite(carGreen13, HIGH);
  digitalWrite(carYellow13, LOW);
  digitalWrite(carRed13, LOW);
  digitalWrite(carGreen24, LOW);
  digitalWrite(carYellow24, LOW);
  digitalWrite(carRed24, HIGH);
}

void release24(){
  digitalWrite(carGreen13, LOW);
  digitalWrite(carYellow13, LOW);
  digitalWrite(carRed13, HIGH);
  digitalWrite(carGreen24, HIGH);
  digitalWrite(carYellow24, LOW);
  digitalWrite(carRed24, LOW);
}

void warning13(){
  digitalWrite(carGreen13, LOW);
```

```
  digitalWrite(carYellow13, HIGH);
  digitalWrite(carRed13, LOW);
  digitalWrite(carGreen24, LOW);
  digitalWrite(carYellow24, LOW);
  digitalWrite(carRed24, HIGH);

  delay(warningTimeCar);
}

void warning24(){
  digitalWrite(carGreen13, LOW);
  digitalWrite(carYellow13, LOW);
  digitalWrite(carRed13, HIGH);
  digitalWrite(carGreen24, LOW);
  digitalWrite(carYellow24, HIGH);
  digitalWrite(carRed24, LOW);

  delay(warningTimeCar);
}

void pedestres4x1(){
  digitalWrite(carRed13, HIGH);
  digitalWrite(carRed24, HIGH);
  digitalWrite(carYellow13, LOW);
  digitalWrite(carYellow24, LOW);
  digitalWrite(carGreen13, LOW);
  digitalWrite(carGreen24, LOW);

  digitalWrite(pedGreen,HIGH);
  digitalWrite(pedRed,LOW);

  delay(passTime);

  digitalWrite(pedGreen,LOW);

  for(int i=0; i<4; i++){
    digitalWrite(pedRed,LOW);
    delay(warningTimePed);
    digitalWrite(pedRed,HIGH);
    delay(warningTimePed);
  }
  pedFlag = false;
}

void pedTest(){
  if(pedFlag == true){
    pedestres4x1();
  }
}

void pedFunc(){
  pedFlag = true;
}
```