
**CENTRO UNIVERSITÁRIO DE BRASÍLIA -UniCEUB**

**CURSO DE ENGENHARIA DE COMPUTAÇÃO**

**FLAVIO CARDOSO DE OLIVEIRA LENZI**

# DISPOSITIVO MÓVEL CONTROLADO POR SMARTPHONE COM TRANSMISSÃO DE VÍDEO EM TEMPO REAL

**Orientador: Prof. MsC Francisco Javier De Obaldia**

Brasília

Novembro, 2012

**FLAVIO CARDOSO DE OLIVEIRA LENZI**

# DISPOSITIVO MÓVEL CONTROLADO POR SMARTPHONE COM TRANSMISSÃO DE VÍDEO EM TEMPO REAL

Trabalho apresentado ao Centro Universitário de Brasília (UniCEUB) como pré-requisito para a obtenção de Certificado de Conclusão de Curso de Engenharia de Computação.
Orientador: Prof. MsC Francisco Javier De Obaldia

Brasília

Novembro, 2012

**FLAVIO CARDOSO DE OLIVEIRA LENZI**

# DISPOSITIVO MÓVEL CONTROLADO POR SMARTPHONE COM TRANSMISSÃO DE VÍDEO EM TEMPO REAL

Trabalho apresentado ao Centro Universitário de Brasília (UniCEUB) como pré-requisito para a obtenção de Certificado de Conclusão de Curso de Engenharia de Computação.
Orientador: Prof. MsC Francisco Javier De Obaldia

Este Trabalho foi julgado adequado para a obtenção do Título de Engenheiro de Computação, e aprovado em sua forma final pela Faculdade de Tecnologia e Ciências Sociais Aplicadas -FATECS.

_____

Prof. MsC Francisco Javier De Obaldia
Orientador

**Banca Examinadora:**

_____

Prof. Vera Lucia Alves Duarte
Mestre

_____

Prof. Sidney Cerqueira Bispo dos Santos
Doutor

_____

Prof. Luís Cláudio Lopes de Araujo
Mestre

## AGRADECIMENTOS

Agradeço à minha família, meu pai Luiz Fernando, minha mãe Maria Fernanda e minha irmã Raquel que me deram toda atenção, suporte, carinho para que pudesse concluir esta etapa da minha vida.

Agradeço a todos os meus amigos da faculdade que compartilharam comigo este desafio de se tornar engenheiro e que foram essenciais durante o curso, em especial Lucas Rehem, Jefferson Santos, Caio de Bem, Bruno Queiroz, Raphael Palmer, Diogo Dantas, Matheus Assis, José Carlos, Lucas Mesquita e Emerson da Hora.

Aos meus colegas de trabalho, em especial Guilherme Silva e Gabriel Freitas.

Ao meu orientador, professor Francisco Javier De Obaldia, por ter acreditado no meu trabalho.

Aos meus professores pela orientação, ensinamento e ajuda e principalmente pelo conhecimento indispensável para a minha formação.

**RESUMO**

Este trabalho apresenta o desenvolvimento de um protótipo que visa realizar o controle e transmissão de vídeo em tempo real de um dispositivo automotivo utilizando um smartphone através da rede wireless. Para seu funcionamento, o microcontrolador *Arduino Duemilanove* é ligado a um Shield WiFi e à placa do controle remoto original do dispositivo automotivo. As aplicações desenvolvidas irão receber, tratar e enviar os comandos enviados pelo smartphone para o microcontrolador. Toda a solução de controle e transmissão de vídeo estão presentes em uma única interface, que o usuário acessa através do navegador do smartphone, tornando desnecessário a instalação de um aplicativo adicional.

**Palavras Chave:** *Arduino*, Shield WiFi, transmissão de vídeo, tempo real, smartphone.

## ABSTRACT

This work presents the development of a prototype which aims to realize the control and transmission of real-time video of an automobile using a smartphone device through the wireless network. For its operation, the microcontroller Arduino Duemilanove is connected to a WiFi Shield and to the device's remote control. Applications developed will receive, process and send the commands sent by the smartphone to the microcontroller. The control and video transmission are present in a single interface, that the user accesses through the smartphone's browser, making it unnecessary to install an additional application.


**Key words:** *Arduino*, Shield WiFi, video transmission, real-time, smartphone.

**SUMÁRIO**

## LISTA DE FIGURAS

## LISTA DE TABELAS

# CAPÍTULO 1 – INTRODUÇÃO

## 1.1 – Visão Geral do Projeto

O projeto é um protótipo de um dispositivo automotivo que é controlado através de um smartphone utilizando um serviço web para fazer a comunicação com o dispositivo. Este possui uma câmera que envia imagens em tempo real para o serviço web. Tanto o controle do dispositivo automotivo quanto as imagens transmitidas pela câmera são acessados através de uma única interface.

## 1.2 – Apresentação do Problema

As principais motivações para a realização deste projeto foram os acidentes que, devido a algumas circunstâncias**,** geram dificuldades de acesso a locais da estrutura onde se pode encontrar pessoas feridas ou objetos de alto risco. Como exemplo, podemos citar o acidente na central nuclear na cidade de Fukushima, no Japão, após um forte terremoto que atingiu o país. Devido ao vazamento de radiação, a exposição de pessoas neste local deve ter o menor tempo possível. Com isso, um dispositivo que pode ser controlado a distância e que consegue enviar imagens em tempo real, possibilita que as equipes de resgate consigam se direcionar exatamente para onde se encontram vítimas e estruturas que precisam de reparos. Será possível desenvolver um protótipo de baixo custo e de fácil controle que realize todas essas ações?

## 1.3 – Objetivos do Trabalho

O projeto tem como objetivo geral especificar, desenvolver e implementar protótipo de veículo com câmera acoplada que pode ser controlado a uma distância segura limitada apenas pelo alcance da rede wireless e que envie imagens em tempo real.

Os objetivos específicos se dividem em alguns itens:

- Desenvolver um código otimizado que consiga enviar os comandos com o mínimo de *delay* possível.

- Integrar o controle do dispositivo automotivo e a recepção do vídeo em tempo real em apenas uma única interface.
- Permitir que a interface de controle seja acessada por qualquer smartphone que tenha conexão à rede especificada e que possua um navegador.
- Permitir o acesso à interface de controle sem a necessidade de instalar um aplicativo no smartphone.

## 1.4 – Escopo do Trabalho

Código desenvolvido e implementado para o sistema de controle que permite que o dispositivo automotivo ande para frente, para trás e para os lados, será acoplada uma câmera no dispositivo automotivo que enviará imagens em tempo real para o servidor, será desenvolvida uma única interface de controle para o usuário.

O sistema não realizará qualquer tipo de controle na velocidade do dispositivo automotivo e nem salvará os vídeos gerados pela câmera.

## 1.5 – Resultados Esperados

Ao final do projeto, é esperado que o protótipo possua uma interface de controle de simples entendimento, que a transmissão do vídeo ocorra de fato em tempo real e em qualidade suficiente para que seja possível identificar objetos e pessoas, que a transmissão dos controles tenha o mínimo de *delay* possível, que todo o sistema possa ser utilizado sem a necessidade de fios e que a interface de controle possa ser acessado por qualquer tipo de smartphone.

## 1.6 – Estrutura do Trabalho

Todo o trabalho desenvolvido e apresentado nesta monografia está estruturado da seguinte maneira:

Capítulo 1: Nesse capítulo é feita a introdução do tema abordado.

Capítulo 2: Nesse capítulo é apresentado o contexto do problema, tratamentos atuais, soluções e tecnologias existentes, e como a proposta apresentada pretende solucionar estes problemas.

Capítulo 3: Nesse capítulo é contemplado todo o referencial teórico e tecnológico necessários para a compreensão daquilo que foi desenvolvido.

Capítulo 4: Nesse capítulo é apresentada a solução proposta, assim como a sua explicação detalhada e forma de funcionamento.

Capítulo 5: Nesse último capítulo, todas as conclusões são apresentadas e propostas de projetos futuros relacionados ao projeto aqui descrito.

# CAPÍTULO 2 – APRESENTAÇÃO DO PROBLEMA

## 2.1 – Contexto geral do problema

É inevitável que no mundo em que vivemos estejamos completamente preparados para alguma catástrofe, porém, é de extrema importância poder utilizar a tecnologia para o nosso favor. São de conhecimento os limites do corpo humano. Sabemos que a radiação emitida pelo combustível das usinas nucleares (em geral urânio ou plutônio) tem a propriedade de alterar a carga elétrica dos elementos das células humanas, e que em um ambiente em chamas, a fumaça dificulta a visibilidade, causa lacrimejamento, irritação dos olhos, intoxicação e etc. Com ambientes tão inóspitos para o ser humano, é natural que sejam desenvolvidos os mais diferentes tipos de tecnologia para solucionar problemas específicos.

Analisando o crescente número na venda de smartphones em todo mundo e especificamente no Brasil, onde, apenas no primeiro semestre de 2012, ocorreu um aumento de 77% nas vendas em comparação com o mesmo período de 2011, que são 6,8 milhões de unidades vendidas, representado 25% do mercado de celulares (dados da IDC), é viável investir tempo e dinheiro em soluções que utilizem um smartphone como meio de controle.

Apesar do grande aumento no desenvolvimento de aplicativos para smartphones, para este projeto, não é viável o desenvolvimento do mesmo já que seu custo é extremamente alto e seria necessário desenvolver uma aplicativo diferente para cada plataforma diferente, como o iOS e o Android. Com isso, a interface acessada pelo usuário é feita pela WEB, utilizando o navegador do smartphone. A tecnologia bluetooth presente nos smartphones comercializados atualmente possui um alcance médio de 10 metros, tornando soluções que a utilizem inviáveis e reforçando a necessidade de utilizar a rede wireless na solução.

## 2.2 – Tecnologias existentes

Nos últimos anos temos vivenciado constantemente evoluções nas tecnologias de suporte às equipes de resgate. Devido ao seu alto custo e necessidade de profissionais qualificados para manuseá-los, são poucos que podem ter uma ferramenta como esta. Não foi encontrada no mercado uma

ferramenta de baixo custo que utiliza a rede wireless como forma de controle, porém temos alguns produtos semelhantes. Um exemplo é o RoBBB, que possui uma câmera integrada com controle sem fio que possui um monitor colorido de 2'' acoplado. O seu alcance é de 30 metros e seu custo é de R$500,00, tornando o produto inviável para devidos fins (Fonte: http://www.globomarcas.com.br).

Figura 2.1 – RoBBB
FONTE: (http://www.globomarcas.com.br/images/400/265252_1_400.jpg)

# CAPÍTULO 3 – REFERENCIAL TEÓRICO E TECNOLÓGICO

Para o desenvolvimento do protótipo foram utilizados diversos componentes e ferramentas com o objetivo de criar a conexão entre o smartphone e o carrinho. Para a montagem final do dispositivo, é necessário o conhecimento teórico de alguns pontos específicos e do conhecimento de certos componentes, softwares e ferramentas.

## 3.1 – TCP

O TCP (*Transmission Control Protocol)* é um protocolo de nível da camada de transporte de Modelo OSI. Existem dois extremos numa conexão TCP, de um lado o servidor (que abre um socket e espera passivamente por ligações), e do outro lado, o cliente. Antes que um processo da aplicação possa começar a enviar dados a outro, eles precisam se identificar, ou seja, eles devem enviar alguns segmentos preliminares um ao outro para estabelecer os parâmetros da transferência de dados. Como parte do estabelecimento da conexão TCP, ambos os lados da conexão iniciarão variáveis associadas com a conexão TCP (KUROSE, 2009).

O processo de aplicação cliente informa à camada de transporte no cliente que ele quer estabelecer uma conexão com um processo no servidor. A camada de transporte no cliente então passa a estabelecer uma conexão. Uma vez estabelecida uma conexão TCP, os dois processos de aplicação podem enviar dados um para o outro. O processo cliente passa uma cadeia de dados através do socket, que em seguida, são passados para o TCP que está rodando no cliente (KUROSE, 2009).

O TCP direciona seus dados para o buffer de envio da conexão, que é um dos buffers reservados durante a apresentação de três vias inicial (procedimento que estabelece a conexão entre o cliente e o servidor). A quantidade máxima de dados que pode ser retirada e colocada em um segmento é limitada pelo tamanho máximo do segmento, ou MSS (*Maximum Segment Size*) (KUROSE, 2009).

O MSS normalmente é estabelecido primeiramente determinando o tamanho do maior quadro de camada de enlace que pode ser enviado pelo hospedeiro remetente local (denominado unidade máxima de transmissão – *maximum transmission unit* - MTU) e, em seguida, estabelecendo um MSS que garanta que um segmento TCP (quando encapsulado em um datagrama IP) caberá em um único

quadro de camada de enlace. Valores comuns da MTU são 1.460 *bytes*, 536 *bytes* e 512 *bytes* (KUROSE, 2009).

O TCP combina cada fração de dados do cliente com um cabeçalho TCP, formando segmentos TCP. Esses segmentos são passados para baixo, para a camada de rede, onde são encapsulados separadamente dentro dos datagramas IP de camada de rede, que são enviados para dentro da rede. Quando o TCP recebe um segmento na outra extremidade, os dados do segmento são colocados no buffer de recepção da conexão, onde a aplicação lê a cadeia de dados desse buffer. Cada lado da conexão tem seus próprios buffers de envio e seu próprio buffer de recepção. Logo, uma conexão TCP consiste em buffers, variáveis e um socket de conexão de um processo em um hospedeiro e outro conjunto de buffers, variáveis e um socket de conexão de um processo em outro hospedeiro (KUROSE, 2009).

No presente trabalho será desenvolvida a aplicação cliente (no caso o usuário com o smartphone) servidor (onde está rodando a aplicação em Python) para a comunicação do socket (que será detalhado no próximo tópico), através de uma conexão que utiliza o protocolo TCP.

### 3.2 – Socket

O Socket é a interface entre a camada de aplicação e a camada de transporte dentro de uma máquina, ou seja, o Socket é um mecanismo que permite a comunicação entre aplicativos, criando um canal de conexão entre eles. O processo em que acontece a conexão ocorre sobre o TCP. Este processo funciona da seguinte forma: O Servidor define uma porta e aguarda conexões nessa porta. Em seguida, o Cliente (que deve saber o host e a porta do servidor) solicita uma conexão com esse Servidor. Caso não haja nenhum problema, o Servidor aceita esta conexão e cria um canal de comunicação entre Cliente/Servidor, esse canal é o Socket (KUROSE, 2009).

O cliente tem a tarefa de iniciar o contato com o servidor. Para que isso seja feito, o servidor não pode ser inativo, ou seja, tem de estar rodando como um processo antes de o cliente tentar iniciar o contato e o programa tem que ter alguma porta, mais precisamente um socket, que acolha algum contato inicial de um processo cliente que esteja rodando em uma máquina qualquer (KUROSE, 2009).

Com o processo servidor em execução, o processo cliente pode iniciar uma conexão TCP com o servidor, o que é feito no programa cliente pela criação de um socket. Quando cria seu socket, o cliente especifica o endereço do processo servidor, a saber, o endereço IP do hospedeiro servidor e um número de porta do processo servidor. Com a criação do socket no programa cliente, o TCP no cliente inicia uma apresentação de três vias e estabelece uma conexão TCP com o servidor. A apresentação de três vias é completamente transparente para os programas cliente e servidor (KUROSE, 2009).

Durante a apresentação de três vias, o cliente chama o socket de entrada do servidor. Quando o servidor responde, é criada uma nova porta (mais precisamente um novo socket) dedicada àquele cliente (KUROSE, 2009).

### 3.3 – Thread

Thread é uma forma de um processo dividir a si mesmo em duas ou mais tarefas que podem ser executadas simultaneamente. Cada processo tem um espaço de endereçamento em um único thread (fluxo) de controle. Porém, frequentemente é desejável ter múltiplos threads de controle no mesmo espaço de endereçamento executando em paralelo, como se eles fossem processos separados. Uma thread permite, por exemplo, que o usuário de um programa utilize uma funcionalidade do ambiente enquanto outras linhas de execução realizam outros cálculos e operações.

Quando um processo com múltiplos threads é executado em um sistema com apenas uma CPU, os threads esperam a vez para executar. Ao alterar entre vários processos, o sistema dá a ilusão de processos sequenciais distintos executando em paralelo. O multithread funciona do mesmo modo. A CPU alterna entre os threads dando a impressão de que os threads estão executando em paralelo, embora em uma CPU mais lenta que a CPU real (TANENBAUM, 2003).

Threads distintos em um processo não são tão independentes quanto processos distintos. Todos os threads tem exatamente o mesmo espaço de endereçamento, o que significa que eles também compartilham as mesmas variáveis globais. Como cada thread pode ter acesso a qualquer endereço dentro do espaço de endereçamento do processo, um thread pode ler, escrever ou até mesmo apagar completamente a pilha de outro thread (TANENBAUM, 2003).

Cada thread tem sua própria pilha. Cada trilha de thread possui uma estrutura para cada procedimento chamado. Essa estrutura possui as variáveis locais do procedimento e o endereço de retorno para usá-lo. Cada thread geralmente chama procedimentos diferentes resultando uma história execução diferente.

Quando ocorre a execução de múltiplos threads, os processos iniciam com um único thread. Esse thread tem a capacidade de criar novos threads chamando um procedimento da biblioteca. Não é necessário especificar qualquer coisa sobre o espaço de endereçamento do novo thread, já que ele executa automaticamente no espaço de endereçamento do thread em criação (TANENBAUM, 2003).

Para este projeto, utilizamos uma classe Thread da linguagem de programação python para que o programa rode simultaneamente o serviço do servidor HTTP o qual é acessado pelo smartphone e o serviço que envia os comandos tratados para o microcontrolador através de um socket.

### 3.4 – Servidor HTTP

O HTTP (*HyperText Transfer Protocol*) é um protocolo de camada de aplicação da Web e é implementado em dois programas: um programa cliente e outro servidor. Os dois programas, executados em sistemas finais diferentes, conversam entre si por meio da troca de mensagens HTTP. O HTTP define a estrutura dessas mensagens e o modo como o cliente e o servidor as trocam. O HTTP usa o TCP como seu protocolo de transporte. O cliente HTTP inicia uma conexão TCP com o servidor. Uma vez que esta conexão é estabelecida, os processos do *browser* e do servidor acessam o TCP através de suas interfaces sockets. No lado do cliente, a interface socket é a porta entre o processo cliente e a conexão TCP; no lado do servidor, ela é a porta entre o processo servidor e a conexão TCP.

O cliente envia mensagens de requisição HTTP para sua interface socket e recebe mensagens de resposta HTTP de sua interface socket. De maneira semelhante, o servidor HTTP recebe mensagens de requisição de sua interface socket e envia mensagens de resposta para sua interface socket. Assim que o cliente envia uma mensagem para sua interface socket, a mensagem sai de suas mãos e “passa para as mãos do TCP” (KUROSE, 2009).

Neste projeto, utilizamos um programa em Pyhton que faz o papel de um serviço HTTP. O smartphone, que ocupa o papel de cliente, acessa uma página

HTML, a qual está rodando no servidor HTTP, onde é possível visualizar o vídeo que está sendo transmitido pela câmera e onde é feito o controle do dispositivo automotivo.

### 3.5 – Proxy

Um proxy é um servidor que atua como um intermediário que atende a requisições do cliente que provém de outros servidores. Por exemplo, um usuário conecta-se em um servidor proxy, requisitando algum serviço, como um arquivo, conexão, página web ou outro recurso disponível em outro servidor.

Neste caso, temos o arduino (no papel de cliente) que acessa o notebook (no papel de servidor) para requisitar os comandos que são obtidos e tratados através de outro serviço.

### 3.6 – WiFi

A WiFi é uma LAN sem fio que utiliza o padrão 802.11. Existem diversos padrões 802.11 para tecnologia de LAN sem fio, como: 802.11b, 802.11a e 802.11g. A Tabela 3.1 apresenta um resumo das principais características de cada padrão.

Tabela 3.1 – Características dos padrões 802.11

| Padrão | Faixa de frequência | Taxa de dados |
|---|---|---|
| **802.11b** | 2,4 - 2,485 GHz | até 11 Mbps |
| **802.11a** | 5,1 – 5,8 GHz | até 54 Mbps |
| **802.11g** | 2,4 – 2,485 GHz | até 54 Mbps |

FONTE: (KUROSE, 2009)

Os três padrões 802.11 compartilham muitas características. Todas utilizam o mesmo protocolo de acesso ao meio, CSMA/CA, e todos utilizam a mesma estrutura de quadro para seus quadros de camada de enlace. Todos os padrões tem a capacidade de reduzir sua taxa de transmissão para alcançar maiores distâncias e todos eles permitem ‘modo de infraestrutura’ e ‘modo ad hoc’.

Como podemos observar na Tabela 3.1, o padrão 802.11b possui uma taxa de dados de 11 Mbps e opera na frequência não licenciada de 2,4 a 2,485 GHz, competindo por espectro de frequência com telefones e fornos micro-ondas de 2,4

GHz. O padrão 802.11a pode funcionar a taxas de bits mais altas, porém com frequências mais altas, fazendo com que a distância de transmissão dessas LANs seja mais curta para um dado nível de potência. O padrão 802.11g opera na mesma faixa de frequência mais baixa do padrão 802.11b, mas com taxas de transmissão mais altas da 802.11a (KUROSE, 2009).

O bloco construtivo fundamental da arquitetura 802.11 é o conjunto básico de serviço (*basic service set* - BSS). Um BSS contém uma ou mais estações sem fio e uma estação-base central, conhecida como ponto de acesso (*access point* - AP). Em uma rede residencial típica, há apenas um AP e um roteador (normalmente integrados como uma unidade) que conecta o BSS à internet (KUROSE, 2009).

Como acontece com dispositivos Ethernet, cada estação sem fio 802.11 tem um endereço MAC de 6 bytes que é armazenado no suporte lógico inalterável (firmware) do adaptador da estação. Cada AP também tem um endereço MAC para sua interface sem fio. Como na Ethernet, esses endereços MAC são administrados pelo IEEE e são globalmente exclusivos.

Neste projeto, utilizamos o próprio notebook como um roteador com a ajuda do programa Connectity, que é um software para Windows que utiliza a placa de rede para tornar o computador um *Hotspot* (local onde a tecnologia WiFi está disponível).

### 3.7 – Microcontrolador

Neste projeto o microcontrolador tem a função de tratar de maneira correta os comandos que são transmitidos pelo smartphone através da rede wireless que utiliza o servidor como uma “ponte” entre os dois.

O microcontrolador caracteriza-se por ser uma pastilha que em seu interior possui a CPU (*Central Processing Unit*), que tem a função principal no sistema, pois controla os acionamentos e comunicações com todas as vias, sempre obedecendo às diretrizes definidas na ROM (*Read Only Memory*), Esta se caracteriza por ser onde são feitas as gravações iniciais e que não podem ser apagadas. A RAM (*Random Access Memory*) detém os dados temporários e os *timers* como componentes principais, podendo haver outros. Um microcontrolador pode ser entendido como um microprocessador e seus periféricos reunidos em um só *chip* (NICOLOSI, 2007).

### 3.7.1 – ATmega328

O ATmega328 é um microcontrolador de 8 bits desenvolvido pela empresa ATMEL. A arquitetura é baseada em um RISC (*Reduced Instruction Set Computer*) e opera em uma frequência máxima de 20 MHz. O ATmega328 possui memórias Flash de 32KB, EEPROM de 1KB e SRAM de 2KB. Ele possui uma tensão de operação que varia de 1.8-5.5V (ATMEL, 2012).

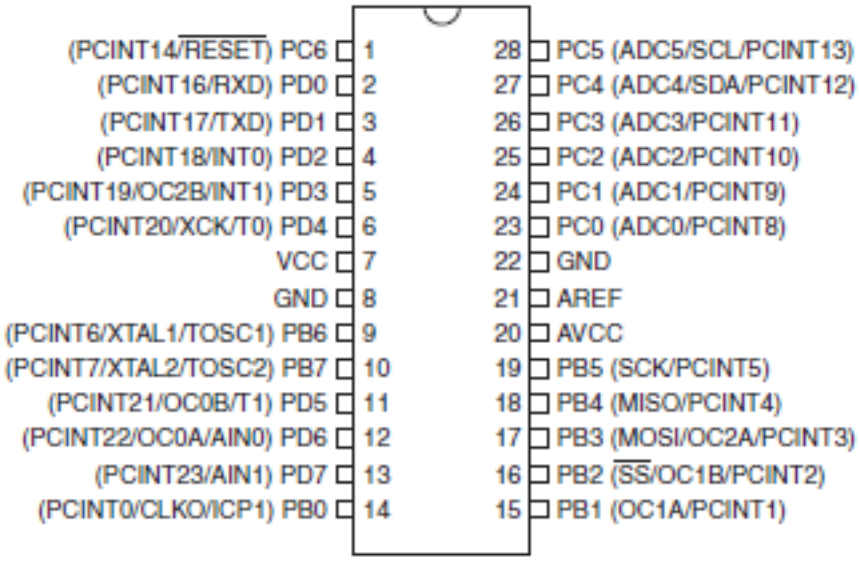


Figura 3.1 – Configuração dos pinos do microcontrolador ATmega328
FONTE: (ATmega48A/48PA/88A/88PA/168A/168PA/328/328P DATASHEET)

### 3.7.2 – Arduino

O Arduino é uma plataforma de prototipagem eletrônica *open-source* e é baseada na flexibilidade do hardware e na facilidade de uso por meio de software e é destinada para qualquer tipo de pessoa que esteja interessada em desenvolver ou criar projetos ou ambientes interativos (BANZI).

Devido às inúmeras entradas e modelos, o arduino pode ser aplicado em diferentes funções e aplicações e combinado com a grande variedade de motores, sensores e *LED's.* Sua linguagem de programação é o *Wiring*, linguagem com a mesma sintaxe C/C++.

### 3.7.3 – Arduino Duemilanove

O Arduino Duemilanove, como pode ser observado na Figura 3.2, é uma placa de microcontrolador baseada no ATmega328. “Duemilanove” significa 2009 em italiano e o nome foi escolhido pelo ano de lançamento. Ele possui 14 pinos de entrada/saída digital (dos quais 6 podem ser usados como saídas analógicas PWM), 6 entradas analógicas, um cristal oscilador de 16 MHz, uma conexão USB, uma entrada para alimentação, um cabeçalho ICSP e um botão de reset. Ele contém toda estrutura que é necessária para que o microcontrolador funcione.

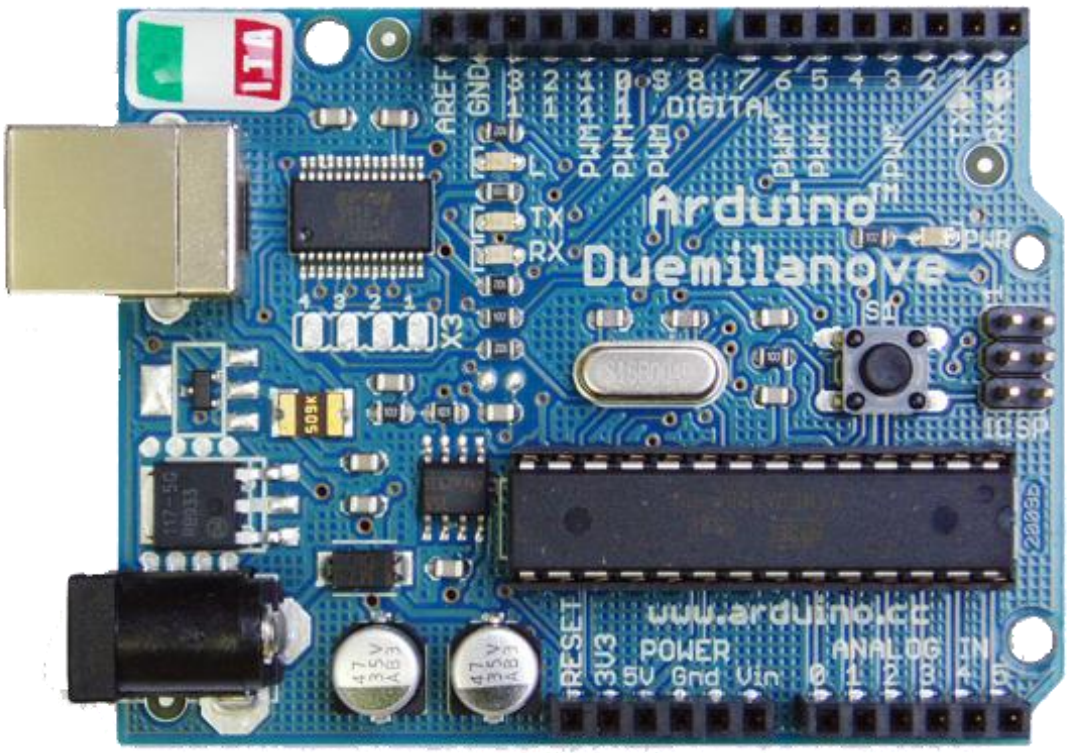


Figura 3.2 – Placa Arduino Duemilanove
FONTE: (O autor)

### 3.8 – Shield WiFi 802.11b

O Shield WiFi utiliza um módulo WiFi RN171 e necessita de dois pinos para se conectar no arduino, oferecendo uma rede wireless 802.11b/g ao dispositivo. Ele possui uma antena independente que cobre maiores áreas e transmite sinais mais fortes do que quando está sem esta antena. Possui compatibilidade com os protocolos TCP, UDP e FTP. Utiliza 3 tipos de autenticação para WiFi: WEP-128,

WPA-PSK (TKIP) e WPA2-PSK (AES). Na Tabela 3.2 é possível observar as especificações do Shield WiFi 802.11b.

Tabela 3.2 – Especificações do Shield WiFi 802.11b

| Item | Mínimo | Normal | Máximo | Unidade |
|---|---|---|---|---|
| **Voltagem** | 3.3 | 5 | 5.5 | VDC |
| **Corrente** | 25 | 60 | 400 | mA |
| **Frequência** | | 2402~2480 | | MHz |
| **Taxa de Rede** | | 1-11 Mbps para 802.11b / 6-54 Mbps para 802.11g | | |
| **Dimensões** | | 60x56x19 | | mm |

FONTE: (*Shield WiFi 802.11b Datasheet*)

Na Figura 3.3 é possível observar a interface do Shield WiFi 802.11b.

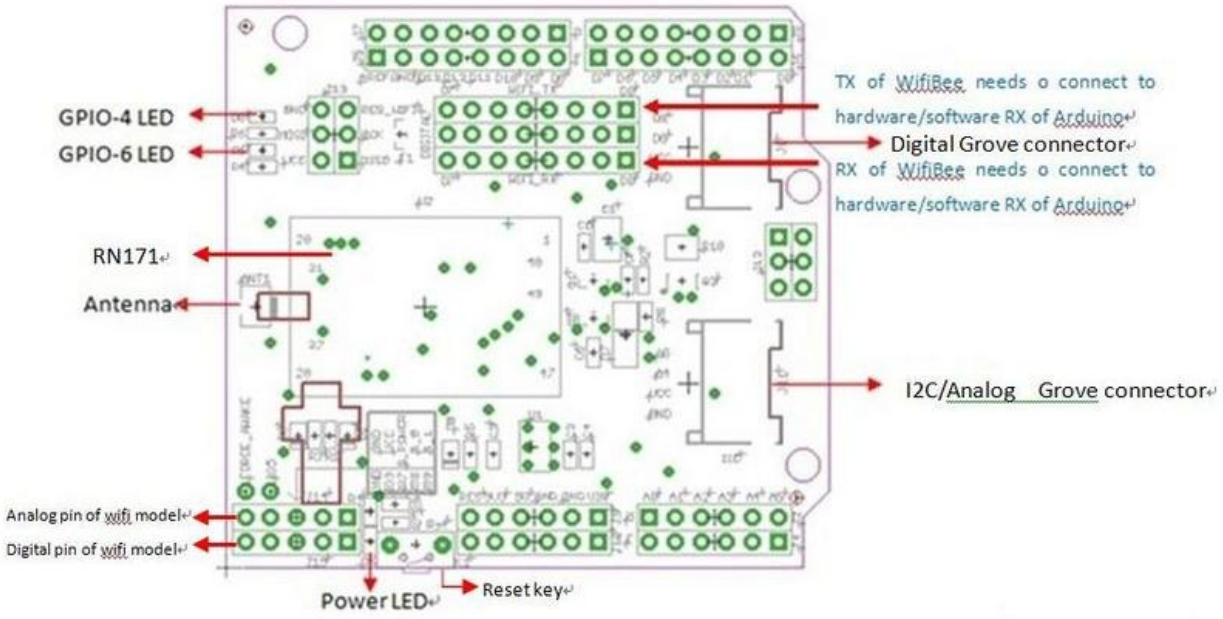


Figura 3.3 – Interface do Shield WiFi 802.11b
FONTE: (*Shield WiFi 802.11b Datasheet*)

# CAPÍTULO 4 – DESENVOLVIMENTO DO DISPOSITIVO MÒVEL CONTROLADO POR SMARTPHONE

## 4.1 – Apresentação Geral do Modelo Proposto

O modelo proposto demanda que certas etapas do projeto sejam seguidas em uma determinada ordem, e outras partes que podem ser desenvolvidas em paralelo. Como pode ser observado na Figura 4.1, o projeto consiste em três processos: a conexão da câmera com o servidor, o desenvolvimento do controle pelo smartphone e a integração do controle e câmera em uma única interface. A conexão da câmera com o servidor pode ser realizada em paralelo com o desenvolvimento do controle pelo smartphone, enquanto que a integração entre os dois demanda que ambas as partes estejam finalizadas.

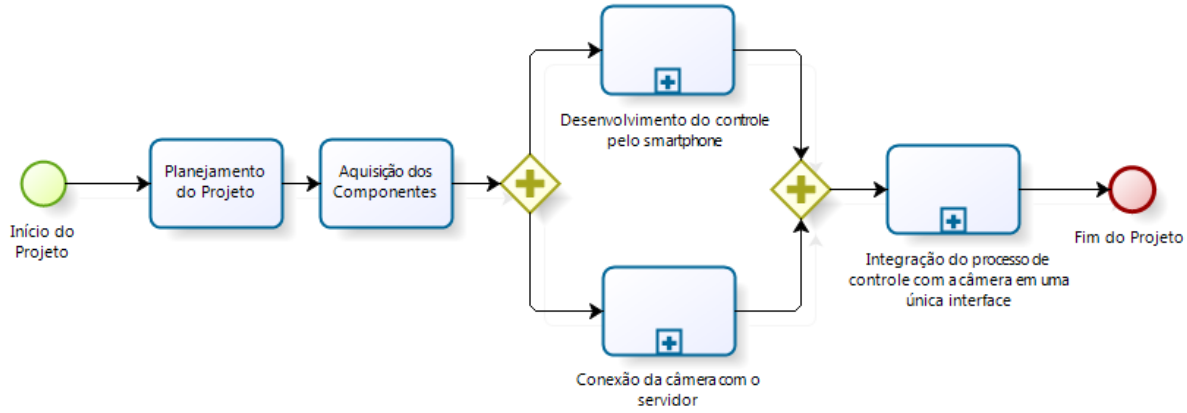


Figura 4.1 – Processo de desenvolvimento do projeto
FONTE: (O autor)

Como pode ser observado na Figura 4.2, a seguir, que representa o funcionamento resumido do processo de controle na forma de diagrama, ao executar o programa em python que está no servidor, o programa fica aguardando a conexão com o smartphone e com o arduino. Assim que a conexão com o smartphone e com o arduino é estabelecida, o programa já é capaz de receber os comandos da porta 80 (as coordenadas que são determinadas pelo ato do usuário de arrastar o dedo pela tela do smartphone), tratar (definir o comando específico de acordo com as coordenadas x e y recebidas no passo anterior) e enviar os comandos para o arduino. Quando o arduino recebe os comandos ele os interpreta

e executa uma ação específica para cada comando diferente. Este processo fica sendo executado até que a conexão com o smartphone ou com o arduino seja interrompida por algum motivo.

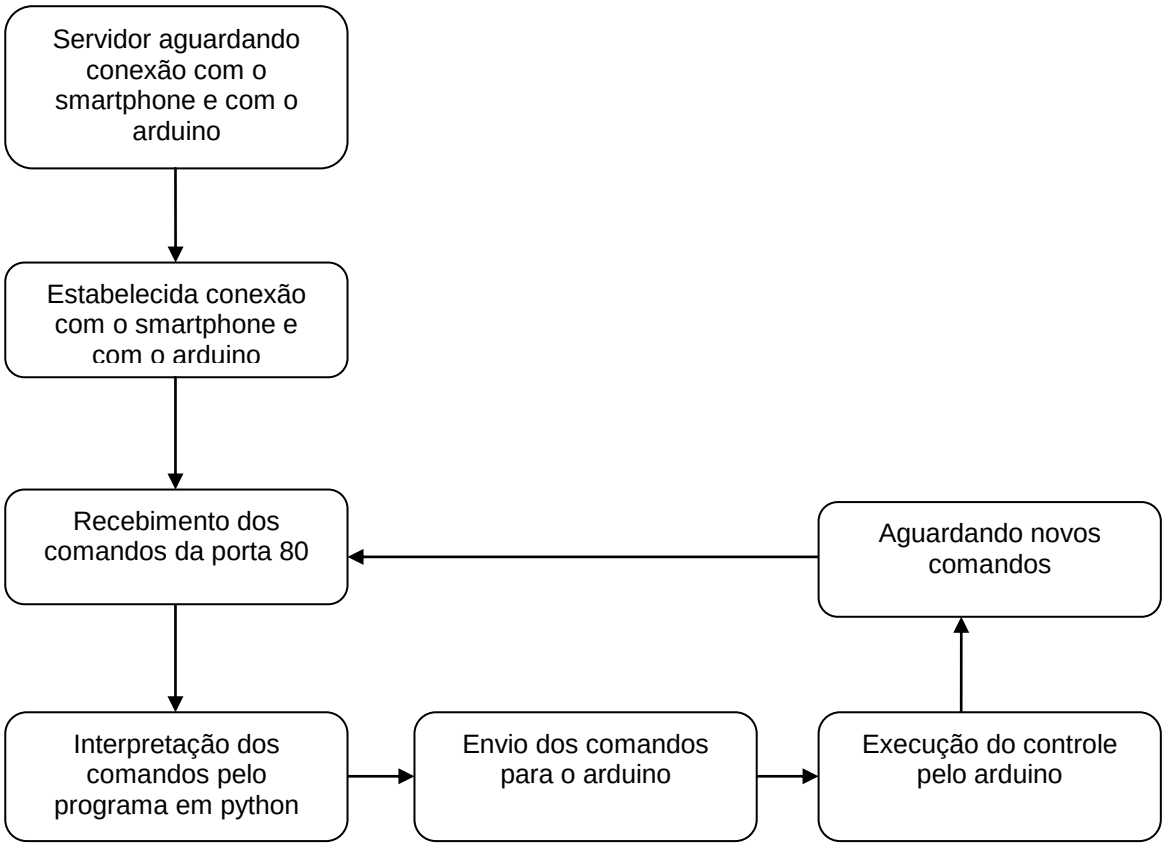


Figura 4.2 – Diagrama de funcionamento do processo de controle
FONTE: (O autor)

### 4.2 – Esquemático do protótipo do projeto

Para o êxito do projeto, foi necessário construir um protótipo físico que, durante o período de montagem e programação, foi utilizado como base para realizar toda a integração dos componentes e todos os testes.

#### 4.2.1 – Aplicação WEB

Como pode ser observado na Figura 4.3, o subprocesso do desenvolvimento do controle pelo smartphone é dividido em 4 tarefas que devem ser realizadas em uma determinada ordem lógica para facilitar no desenvolvimento, sendo elas: A – Desenvolvimento da página WEB que captura as coordenadas geradas pelo usuário; B – Desenvolvimento do programa de captura e tratamento das

coordenadas, resultando no envio para o arduino; C – Desenvolvimento do código do arduino que é responsável pela conexão na rede e o recebimento dos parâmetros enviados; D – Montagem dos componentes do processo de controle no dispositivo automotivo.

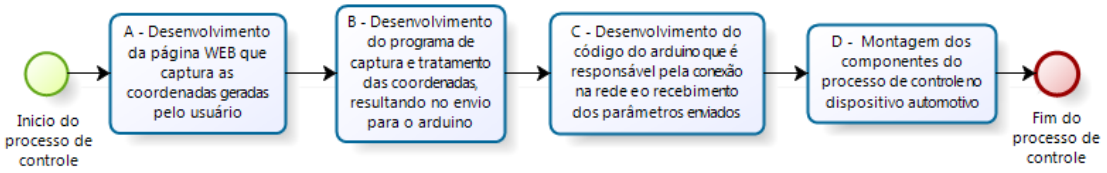


Figura 4.3 – Subprocesso do desenvolvimento do controle
FONTE: (O autor)

Na tarefa A, foram utilizadas as linguagens HTML e JavaScript para desenvolver a página WEB que o usuário acessa com o smartphone. Na fase de testes, foi utilizado o programa WampServer (o qual instala Apache, MySQL e PHP para Windows) como ambiente de desenvolvimento pois ele permite que sejam criadas aplicações WEB de forma simples. Com o andamento do projeto, o programa desenvolvido em Python assume o papel do WampServer.

No código da tarefa A, temos o código escrito em JavaScript para capturar o posicionamento do dedo do usuário na tela e enviar essas coordenadas para serem tratadas. Na Figura 4.4 está sendo declarada a url base da página para onde são enviadas as coordenadas (como essa página é utilizada apenas como uma “ponte” entre a aplicação WEB e a aplicação em Python, podemos atribuir qualquer nome para ela, contanto que este mesmo nome seja definido na aplicação em Python), e as variáveis utilizadas no código. As variáveis *inicialx* e *inicialy* são as variáveis utilizadas no momento que o usuário toca na tela do smartphone e as variáveis *coordenadax* e *coordenaday* são as variáveis utilizadas no movimento do dedo na tela.

```
8   <script type="text/javascript" charset="utf-8">
9
10  urlBase = "/enviarComando";
11
12  var inicialx = 0;
13  var inicialy = 0;
14  var coordenadax = 0;
15  var coordenaday = 0;
```

Figura 4.4 – Definição das variáveis
FONTE: (O autor)

Na Figura 4.5, é declarada a função *ontouchstart*, que é iniciada no momento em que o usuário coloca o dedo na tela do smartphone. Nesta função é definido que, em qualquer ponto que o usuário coloque o dedo, este ponto será a coordenada (0,0), ou seja, 0 no eixo x e 0 no eixo y. Em seguida, é escrito no elemento com que tenha como Id o valor *controle*, onde mais a frente será atribuído a um elemento do HTML de divisória chamado de <*div*>, a palavra *Início*, indicando para o usuário que ele pode movimentar o dedo pela tela do smartphone.

```
17  document.ontouchstart = function(e){
18      var controle = e.touches[0];
19      inicialx = controle.pageX;
20      inicialy = controle.pageY;
21      document.getElementById("controle").innerHTML = "Inicio";
22  }
```

Figura 4.5 – Função ontouchstart
FONTE: (O autor)

Na Figura 4.6 é declarada a função *ontouchend*, que é inicializada no momento em que o usuário tira o dedo da tela do smartphone. Nesta função, são resetadas todas as variáveis para o valor 0, é escrito para o usuário a palavra *Fim* e são enviadas as coordenadas com o valor 0, desse modo o dispositivo automotivo para no momento em que o usuário tira o dedo da tela.

```
24  document.ontouchend = function(e){
25      inicialx = 0;
26      inicialy = 0;
27      coordenadax = 0;
28      coordenaday = 0;
29      document.getElementById("controle").innerHTML = "Fim";
30
31      var xmlhttp = new XMLHttpRequest();
32      xmlhttp.open("GET", urlBase+"?x="+coordenadax+"&y="+coordenaday,true);
33      xmlhttp.send(null);
34  }
```

Figura 4.6 – Função ontouchend
FONTE: (O autor)

Na Figura 4.7 está sendo declarada a função *ontouchmove*. Enquanto o usuário estiver movimentando o dedo pela tela do smartphone, as variáveis *coordenadax* e *coordenaday* estarão recebendo a posição do dedo na tela. A subtração das variáveis *inicialx* e *inicialy* ocorre para poder simular o local em que o

usuário coloque o dedo como sendo o ponto inicial, ou seja, quando o usuário coloca o dedo na tela, serão geradas as coordenadas (0,0) mesmo que a posição real seja, por exemplo, (100,100). Para que possamos obter de maneira correta a posição (1,0), é feita a subtração das variáveis dos pontos iniciais, assim teriamos 101 da posição real sendo subtraido de 100, que é a posição inicial e que está sendo representado pela variável *inicialx*. A *coordenaday* tem o seu sinal invertido pois o comando *pageY* considera no eixo Y como sendo positivo indo para baixo, logo invertemos o sinal da variável para obter o seu valor correto. Enquanto isso, automaticamente está sendo escrito para o usuário as coordenadas de seu dedo na tela.

```
36  document.ontouchmove = function(e){
37      e.preventDefault();
38      if(e.touches.length == 1){
39          var controle = e.touches[0];
40          coordenadax = controle.pageX - inicialx;
41          coordenaday = - (controle.pageY - inicialy);
42          document.getElementById("controle").innerHTML = "Coordenadas x " + coordenadax + " y " + coordenaday;
43      }
44  }
```

Figura 4.7 – Função ontouchmove
FONTE: (O autor)

Como podemos observar na Figura 4.8, a função *enviaCoordenadas* envia as coordenadas geradas no movimento do dedo do usuário na tela do smartphone para serem tratadas. O comando *setInterval* executará a função *enviaCoordenadas* a cada 200 milisegundos.

```
46  function enviaCoordenadas(){
47      var xmlhttp = new XMLHttpRequest();
48      xmlhttp.open("GET", urlBase + "?x="+coordenadax+"&y="+coordenaday,true);
49      xmlhttp.send(null);
50  }
51
52  setInterval("enviaCoordenadas()",200);
53
54  </script>
```

Figura 4.8 – Função enviaCoordenadas
FONTE: (O autor)

### 4.2.2 – Aplicação Python

Após finalizar a tarefa A, podemos dar inicio a tarefa B, que consiste no desenvolvimento da aplicação que deve estar no servidor sendo responsável por

manter o serviço WEB, capturar as coordenadas que foram enviadas na tarefa A, tratar essas coordenadas e por fim, enviá-las para o microcontrolador.

Na Figura 4.9 podemos observar o *import* das bibliotecas utilizadas. A biblioteca *serial* só é utilizada na fase de testes, já que o envio dos comandos para o microcontrolador é realizado pela WiFi e não pela porta serial. São declaradas as variáveis *portaRequest*, que define a porta que será utilizada pelo socket para realizar a conexão entre o cliente e o servidor, e a variável *pipe*, que receberá o comando gerado pelas coordenadas.

```
1   import serial
2   import socket
3   import SocketServer
4   import SimpleHTTPServer
5   from threading import Thread
6   from urlparse import urlparse, parse_qs
7
8
9   portaRequest = 80
10  pipe=""
```

Figura 4.9 – imports do código em Python
FONTE: (O autor)

Como podemos observar na Figura 4.10, é criada a classe *gerenciadorRequests* e é definada a função *do_GET*. Então é verificado se a *request* possui a string */enviarComando*, que se refere a página para onde são enviadas as coordenadas pela aplicação WEB. Caso seja encontrada a string, é executada a função *tratar*, que será descrita mais a frente, que atribui o seu retorno (as coordenadas tratadas) às variáveis *comando* e *pipe*. Caso a string não seja encontrada, a função vai fazer apenas o papel de um servidor simples HTTP, substituindo o WampServer como foi comentando anteriormente.

```
class gerenciadorRequest(SimpleHTTPServer.SimpleHTTPRequestHandler):
    def do_GET(self):
        global pipe
        if(self.path.find("/enviarComando?") != -1):
            comando = tratar(self.path)
            pipe = comando
        else:
            SimpleHTTPServer.SimpleHTTPRequestHandler.do_GET(self)
```

Figura 4.10 – Função tratar
FONTE: (O autor)

Como mostrado nas Figuras 4.12 e 4.13 a seguir, utilizamos a função *tratar* para transformar as coordenadas recebidas em um comando mais simples para ser enviado para o microcontrolador. Primeiramente quebramos a *query* (Figura 4.6 e 4.8) que foi passada via GET pela aplicação WEB e atribuimos à variável *x* a coordenada x e à variável *y* a coordenada y. Para gerar os comandos, dividimos a tela do smartphone em 6 áreas como é mostrado na Figura 4.11, sendo o centro do quadrado o ponto inicial quando o usuário coloca o dedo na tela, ou seja, o ponto das coordenadas (0,0).

Ao movimentar o dedo pela tela, as coordenadas são tratadas e é gerado um comando para cada área específica. Deste modo, temos 6 comandos de movimentos diferentes mais o comando para parar, que é gerado no momento em que o usuário tira o dedo da tela e é representado pelo número 0.

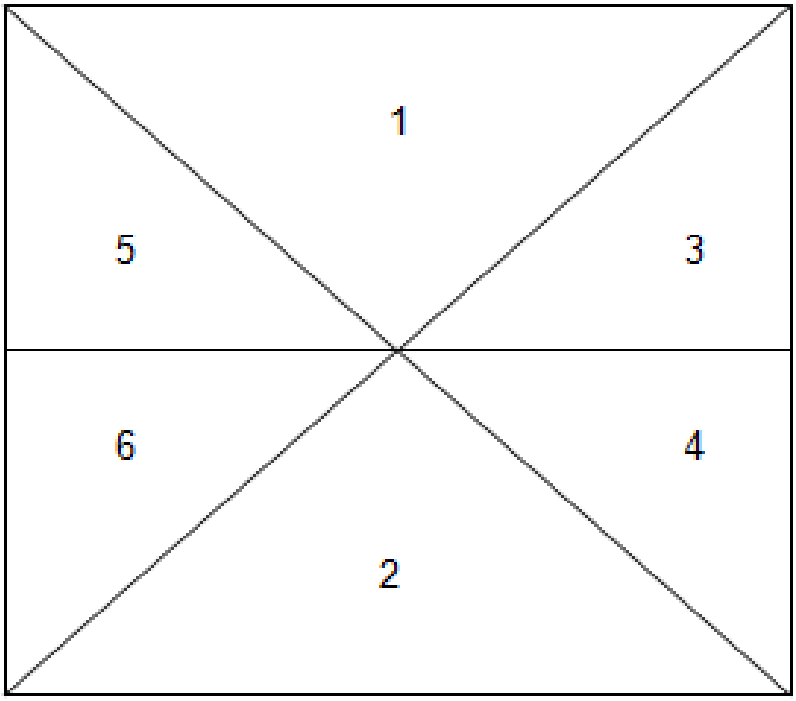


Figura 4.11 – Representação da divisão da tela do smartphone em áreas
FONTE: (O autor)

```
def tratar(path):
    coordenadas = parse_qs(urlparse(path).query)
    x = float(coordenadas['x'][0])
    y = float(coordenadas['y'][0])

    #parar
    if (x == 0 and y == 0):
        comando = 0

    #frente = 1
    if (x>=0 and y>0 and y>=x):
        comando = 1
    if (x<0 and y>0 and y>x):
        comando = 1
```

Figura 4.12 – Função tratar
FONTE: (O autor)

```
    #tras = 2
    if (x<=0 and y<0 and abs(y)>=abs(x)):
        comando = 2
    if (x>0 and y<0 and abs(y)>x):
        comando = 2

    #frente-direita = 3
    if (x>0 and y>=0 and y<x):
        comando = 3

    #tras-direita = 4
    if (x>0 and y<0 and abs(y)<=x):
        comando = 4

    #frente-esquerda = 5
    if (x<0 and y>=0 and y<abs(x)):
        comando = 5

    #tras-esquerda = 6
    if (x<0 and y<0 and abs(y)<abs(x)):
        comando = 6

    return comando
```

Figura 4.13 – Continuação da função tratar
FONTE: (O autor)

Na Figura 4.14 temos a criação da classe *ServicoProxyArduino*, que possui as funções que são responsáveis pela comunicação com o arduino e que serão descritas nos próximos itens, e a criação do *__init__*, que é o construtor da nossa classe. Em python, quando uma classe é instanciada, o __init__ é chamado para

definir um comportamento adicional, basicamente definindo alguns valores iniciais para aquele objeto ou para executar alguma rotina necessária no instanciamento.

```
66  class ServicoProxyArduino:
67      def __init__(self, host, port):
68          self.message = ""
69          self.host = host
70          self.port = port
```

Figura 4.14 – Definição da classe ServicoProxyArduino
FONTE: (O autor)

A comunicação com o microcontrolador é realizada utilizando outro socket, que tem a sua porta definida, como será mosrtado na Figura 4.18. A função que realiza essa comunicação é a *escutarSocket*, como podemos observar na Figura 4.15. Nesta função, aguardamos a conexão com o arduino. No momento que a conexão é realizada, é executada a função *repassarComandos* que está representada na Figura 4.16. Caso a conexão com o microcontrolador seja perdida, uma mensagem de aviso é mostrada para o usuário através da aplicação em Python e o envio dos comando é interrompido até que uma nova conexão seja estabelecida.

```
73      def escutarSocket(self):
74          self.novoSocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
75          self.novoSocket.bind((HOST,PORT))
76          self.novoSocket.listen(0) #sem fila
77          print  "Aguardando conexao do arduino"
78          while(True):
79              self.conexao, self.endereco_arduino_wifly = self.novoSocket.accept()
80              try:
81                  while(True):
82                      self.repassarComandos()
83              except socket.error:
84                  print  "Conexao encerrada. Aguardando nova conexao do arduino"
```

Figura 4.15 – Função escutarSocket
FONTE: (O autor)

Como foi comentado anteriormente, no momento em que a conexão com o microcontrolador é realizada, a função *repassarComandos* ,que está sendo descrita na Figura 4.16, é executada. Por sua vez, a função *repassarComandos* executa outras duas funções: *lerProximoComando* e *tratarComando*. A função *lerProximoComando* armazena o próximo comando, já tratado, que veio do socket com a aplicação WEB.

```
88      def repassarComandos(self):
89          comando = self.lerProximoComando()
90          self.tratarComando(comando)
91
92      def lerProximoComando(self):
93          global pipe
94          return str(pipe)
```

Figura 4.16 – Funções repassarComandos e lerProximoComando
FONTE: (O autor)

Como podemos observar na Figura 4.17, a função *tratarComando* que é exutada pela função *repassarComandos* apenas verifica se o novo comando recebido é igual ao último comando. Se os comandos forem iguais, não é feito nada, mas se os comandos forem diferentes, é executada a função *enviarComandoParaArduino*. Desta maneira, ao invés de passar um comando novo a cada 200 milisegundos para o microcontrolador, só enviamos um comando caso ele seja diferente do último comando. Com isso, ganhamos desempenho ao não sobrecarregar o *socket* com inúmeros comandos iguais. Na função *enviarComandoParaArduino*, enviamos o comando para o microcontrolador através do método *sendall*, que escreve o comando no socket.

```
96      def tratarComando(self, comando):
97          self.tmp = ""
98          self.tmp = comando
99          if (self.tmp != self.message):
100             self.enviarComandoParaArduino(comando)
101             self.message = self.tmp
102             print self.message
103
104     def enviarComandoParaArduino(self, comando):
105         self.conexao.sendall(comando)
```

Figura 4.17 – Funções tratarComando e enviarComandoParaArduino
FONTE: (O autor)

Na Figura 4.18 temos a definição das variáveis *HOST* e *PORT*, e da função *wifiMain*. As variáveis *HOST* e *PORT* determinam os parâmetros da conexão com o microcontrolador. A porta definida neste ponto deve ser a mesma porta definida no código do microcontrolador, desta forma, a comunicação é feita através do mesmo socket. Se o parâmetro do host estiver em branco, o serviço escuta de todos os

nomes disponíveis no sistema. A função *wifiMain* executa paralelamente toda a comunicação com o smartphone e com o microcontrolador, recebendo as coordenadas da aplicação WEB, tratando essas coordenadas e enviando o comando para o microcontrolador.

```
109 HOST='' #quando nao define nenhum nome de host, o servico ouve em todos os nomes disponiveis no sistema
110 PORT=2310
111
112 def wifiMain():
113     threadSmartphone = Thread(target=iniciarServicoSmartphone)
114     threadSmartphone.start()
115     servico = ServicoProxyArduino(HOST, PORT)
116     servico.escutarSocket()
```

Figura 4.18 – Função wifiMain
FONTE: (O autor)

Como descrito anteriormente e que está sendo demonstrado na Figura 4.19, a função *iniciarServicoSmartphone* cria o socket que utiliza a porta 80 e executa a função *gerenciadorRequest*.

```
120 def iniciarServicoSmartphone():
121     print "Recebendo comandos na porta", portaRequest
122     httpd = SocketServer.ThreadingTCPServer(('', portaRequest), gerenciadorRequest)
123     httpd.serve_forever()
124
125
126 wifiMain()
```

Figura 4.19 – Função iniciarServicoSmartphone
FONTE: (O autor)

### 4.2.3 – Código Arduino

Finalizando a aplicação em Python, a tarefa B está concluída e pode-se dar início a tarefa C. Até este ponto, o sistema já consegue capturar as coordenadas geradas pelo usuário, ler essas coordenadas através do socket que usa a porta 80, tratá-las e enviá-las para o microcontrolador através de outro socket que usa a porta 2310. A tarefa C determina que seja desenvolvido o código do arduino, responsável pela conexão do mesmo na rede e pelo recebimento dos parâmetros enviados pela aplicação em Python através do socket que utiliza a porta 2310.

No código do arduino, para realizar a conexão com a rede, é necessário acoplar o Shield WiFi 802.11b no arduino. Na Figura 4.20, é feito o include da

biblioteca utilizada pelo Shield WiFi 802.11b, a *Wifly.h*, que se encontra no Anexo A deste documento.

```
#include "Wifly.h"
#include <SoftwareSerial.h>
WiflyClass Wifly(2,3);

int pinol0 = 10;
int pinol1 = 11;
int pinol2 = 12;
int pinol3 = 13;
char comando;

void setup(){

  Serial.begin(115200);
  Wifly.init();
  Wifly.setConfig("Connectify-Flavio","projetofinal");
  Wifly.join("Connectify-Flavio");
  Wifly.checkAssociated();
  while(!Wifly.connect("192.168.123.1","2310"));
  Wifly.writeToSocket("Connected!");

  pinMode(pinol0, OUTPUT); //frente
  pinMode(pinol1, OUTPUT); //tras
  pinMode(pinol2, OUTPUT); //direita
  pinMode(pinol3, OUTPUT); //esquerda
}
```

Figura 4.20 – Código de conexão com a rede
FONTE: (O autor)

No comando *WiflyClass Wifly(2,3)*, a definição dos parâmetros 2 e 3 representam o RX e TX baseados nos pinos D2 e D3 respectivamente, como pode ser observado na Figura 4.21.

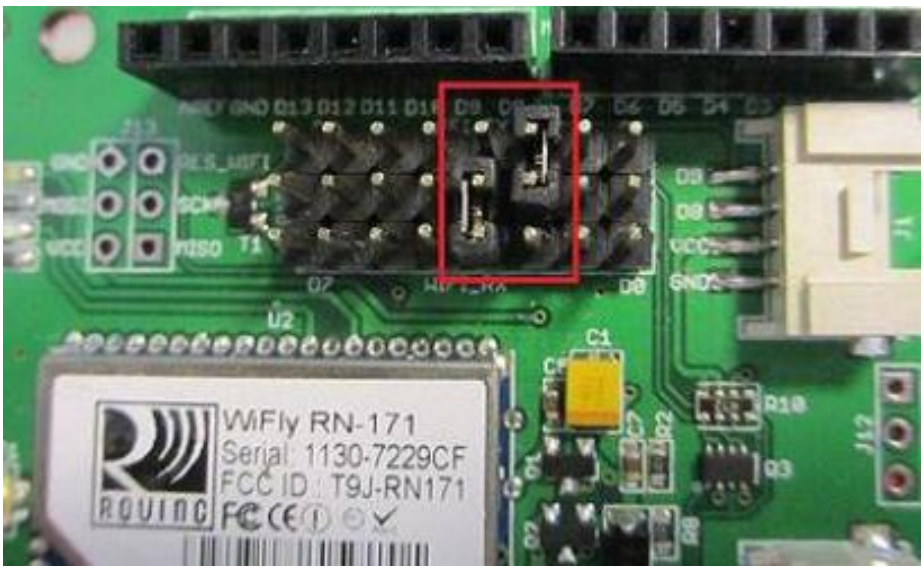


Figura 4.21 – RX e TX do Shield WiFi
FONTE: (O autor)

Antes de executar o código para a conexão na rede, 5 variáveis são declaradas: *pino10, pino11, pino12 e pino13*, que são pinos do arduino e recebem como valor inicial 10, 11, 12 e 13 respectivamente, e a variável *comando*. Com as variáveis declaradas, podemos conectar o microcontrolador na rede. São definidos alguns parâmetros como o SSID, senha, ip e porta, onde neste exemplo possuem os valores *Connectify-Flavio, projetofinal, 192.168.123.1* e *2310* (aqui é utilizada a mesma porta definida na aplicação em Python que será utilizada pelo socket) respetivamente. Neste caso, utilizamos um programa chamado Connectify-me que permite utilizar o computador como um roteador. Ao final do *setup*, declaramos os pinos do microcontrolador que serão utilizados.

Assim que a conexão com a rede é efetuada, executamos uma função *loop* que lê o que está escrito no socket e atribui esse valor à variável *comando*. Para cada comando recebido, uma ação diferente é executada. Se o comando recebido tiver o valor 0, o que significa que o dispositivo automotivo deve parar, todos os pinos tem o seu estado alterados para *LOW*. Caso o comando recebido seja 1, o que indica a ação de ir para frente, apenas o pino 10 tem seu valor alterado para *HIGH*, como podemos observar na Figura 4.22. Isso ocorre até o comando 6, sendo que para cada comando temos um ou dois pinos com seu valor alterado para *HIGH*.

```
void loop(){
  while(Wifly.canReadFromSocket()){ //verifica se alguma mensagem

    comando = char(Wifly.readFromSocket());

      if(comando == '0'){ //parar
        digitalWrite(pino10, LOW);
        delay(100);
        digitalWrite(pino11, LOW);
        delay(100);
        digitalWrite(pino12, LOW);
        delay(100);
        digitalWrite(pino13, LOW);
      }
      if(comando == '1'){ //frente
        digitalWrite(pino10, HIGH);
        delay(100);
        digitalWrite(pino11, LOW);
        delay(100);
        digitalWrite(pino12, LOW);
        delay(100);
        digitalWrite(pino13, LOW);
      }
```

Figura 4.22 – Trecho do código
FONTE: (O autor)

### 4.2.4 – Montagem dos Componentes do Processo de Controle

Finalizado o código do arduino inicia-se a tarefa D, que consiste na montagem física dos componentes no dispositivo automotivo. O microcontrolador recebe os comandos através da rede wireless utilizando o Shield WiFi. Para o microcontrolador se comunicar com o dispositivo automotivo, é necessário conectá-lo à placa do controle remoto.

Como podemos observar na Figura 4.23, o sinal enviado pelo controle remoto para o dispositivo automotivo ocorre quando, ao movimentar o *joystick*, encostamos o objeto (1) no objeto (2). Utilizando-se de um multímetro, verificamos que o objeto (1) tem o papel de terra. Para simular essa ação vindo do microcontrolador, foram utilizados transistores 2N7000 e resistores de 100k Ohm.

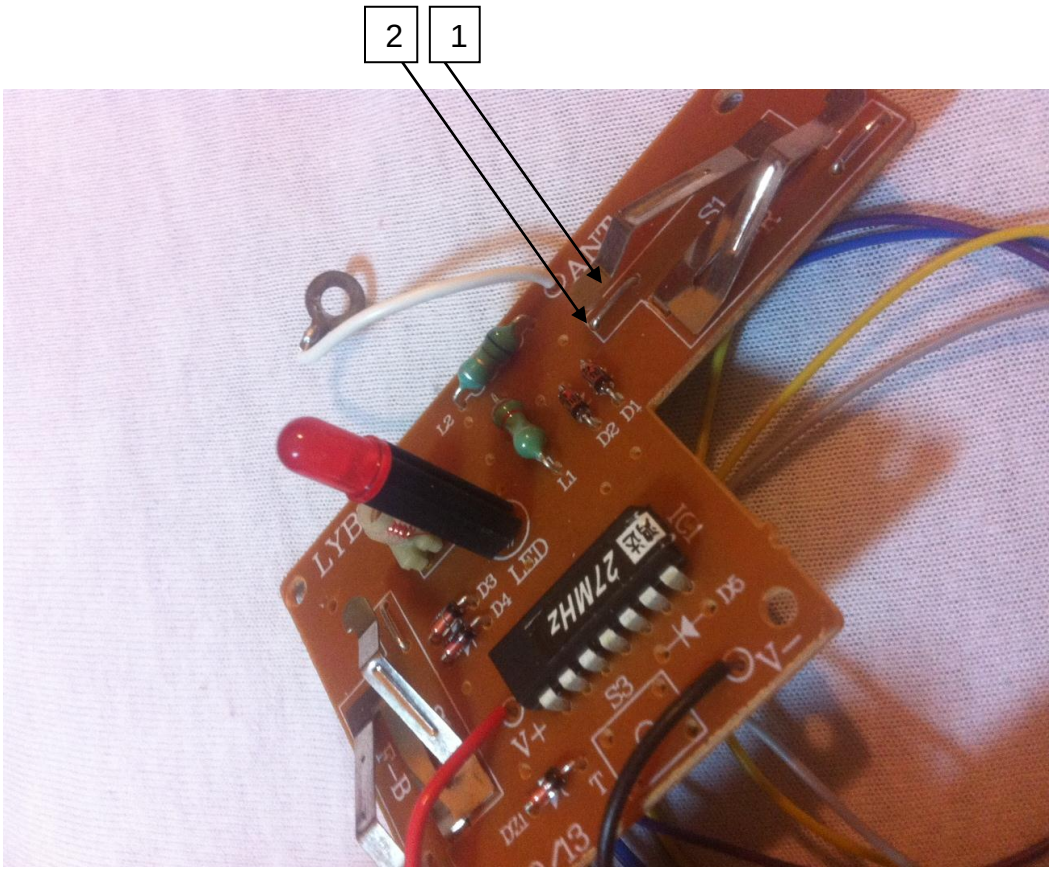


Figura 4.23 – Placa do Controle Remoto
FONTE: (O autor)

Por questão de espaço dentro do dispositivo automotivo, foi confeccionada uma placa para realizar a comunicação entre o microcontrolador e a placa do controle remoto. Como podemos observar na Figura 4.24, a placa possui 4 transistores 2N7000 e 4 resistores de 100k Ohm e ela se encaixa perfeitamente no

Arduino Duemilanove. O transistor possui 3 pernas, sendo a 1 o *source,* a 2 o *gate* e a 3 o *drain*. A perna 1 de todos os transistores deve ir para o *ground* do microcontrolador. A perna 2 deve se ligar ao pino referente ao microcontrolador através de um resistor. Neste caso, o transistor T1 se conecta ao pino D13, o T2 ao pino D12, o T3 ao pino D11 e o T4 ao pino D10. A perna 3 de cada resistor se conecta a um dos comando da placa do controle remoto, sendo que o transistor T1 se conecta ao comando de ir para a esquerda, o transistor T2 se conecta ao comando de ir para a direita, o transistor T3 se conecta ao comando de ir para trás e o transistor T4 se conecta ao comando de ir para frente.

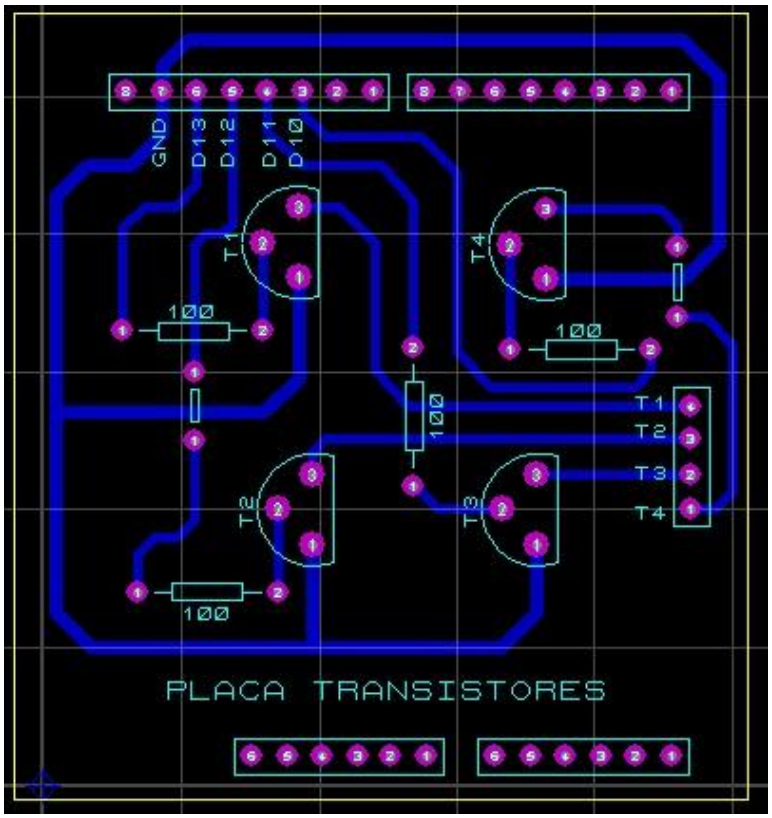


Figura 4.24 – Circuito da Placa Transistores
FONTE: (O autor)

Finalizada a placa, aproveitamos para conectar a fonte de energia da placa do controle remoto ao própiro microcontrolador, ligando o cabo vermelho à sáida de 5V do arduino e o fio preto ao *ground*, como pode ser observado na Figura 4.25.

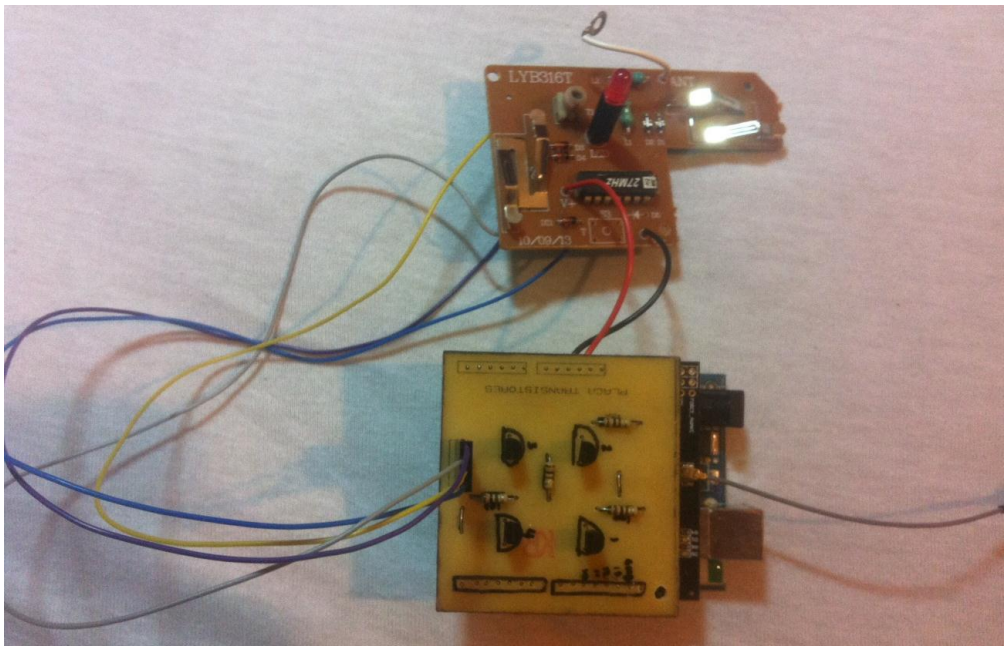

Figura 4.25 – Placa Transistores
FONTE: (O autor)

### 4.2.5 – Conexão da Câmera

Com todas as placas finalizadas, a tarefa D e todo o processo de controle do dispositivo está concluída. É dado ínicio ao processo de conexão da câmera com o servidor, que pode ser realizado em paralelo com o processo de controle do dispositivo automotivo, e que possui três tarefas que podem ser observadas na Figura 4.26, sendo elas:

A – Realizar a conexão da câmera ao servidor via cabo;

B – Configurar a conexão da câmera ao servidor WiFi;

C – Substituir a fonte de energia para que a câmera não dependa da tomada;

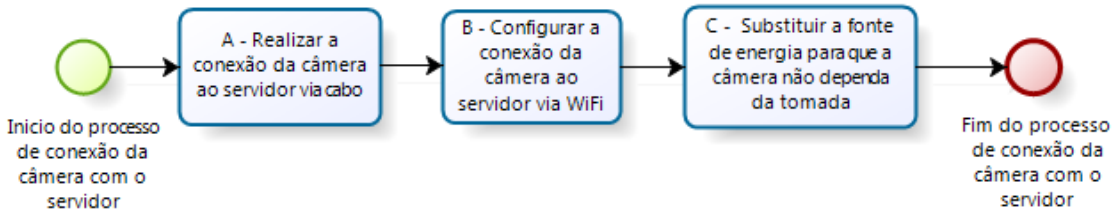


Figura 4.26 – Subprocesso da conexão da câmera com o servidor
FONTE: (O autor)

Para este processo, foi utilizada uma câmera de segurança que se conecta à rede sem a necessidade de um receptor. A câmera escolhida, ao ser ligada em uma fonte de energia e ao roteador, automaticamente recebe um IP na rede. Para

configurar a conexão na rede wireless, foi necessário selecionar a rede WiFi desejada, onde para este projeto, selecionamos a rede criada pelo programa Connectify-me, como podemos observar na Figura 4.27.

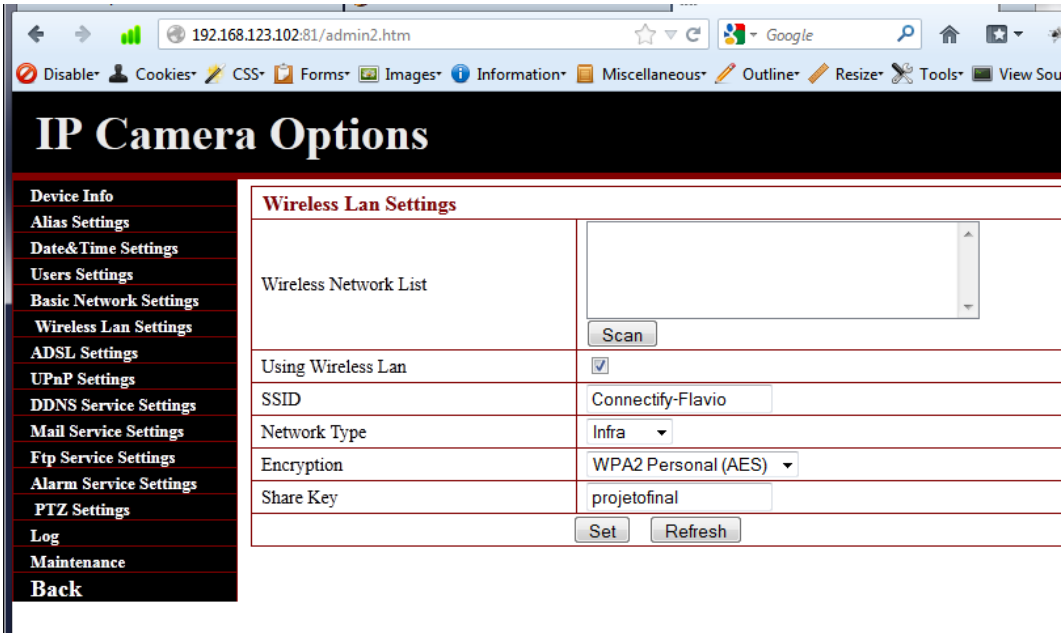


Figura 4.27 – Tela de Configuração da Câmera
FONTE: (O autor)

Para substituir a fonte de energia da câmera, foi utilizada uma bateria de 5V e 2A.

### 4.2.6 – Integração do Sistema de Controle e Transmissão de Vídeo

Para realizar a integração do sistema de controle com a transmissão de vídeo em uma única interface foi adicionado um parâmetro no código da aplicação WEB. Como podemos observar na Figura 4.28, foi adicionada a *tag iframe*, que neste caso pega a imagem do vídeo de sua página original, representada pelo *src="http://192.168.123.102:81/pda.htm*", e disponibiliza o vídeo na aplicação WEB.

```
57
58  <div style="width:310px;height:260px;position:absolute;z-index:1;" id="controle">
59  </div>
60
61  <div style="z-index:0;">
62      <iframe width='300px' height='400px' frameborder='0' src='http://192.168.123.102:81/pda.htm'></iframe>
63  </div>
64
65  </body>
66  </html>
```

Figura 4.28 – Integração do Sistema de Controle e Transmissão de Vídeo
FONTE: (O autor)

Para testar o protótipo, primeiramente deve-se executar a aplicação em Python desenvolvida na tarefa B para que o serviço WEB fiquei disponível. Através do navegador do smartphone, deve-se acessar a aplicação WEB que foi desenvolvida na tarefa A. Neste caso, acessamos o endereço *192.168.123.1/controle.html*, sendo que *controle.html* é o arquivo da aplicação WEB. Também é necessário que o microcontrolador esteja conectado na rede. Ao movimentar o dedo pela tela do smartphone, os comandos devem ser visualizados na aplicação Python, como podemos observar na Figura 4.29, onde o comando 0 significa que o dispositivo automotivo deve parar, o comando 1 significa ir para frente, o comando 2 significa ir para trás, o comando 3 significa ir para direita para frente, o comando 4 significa ir para direita para trás, o comando 5 significa ir para esquerda para frente e o comando 6 significa ir para esquerda para trás.

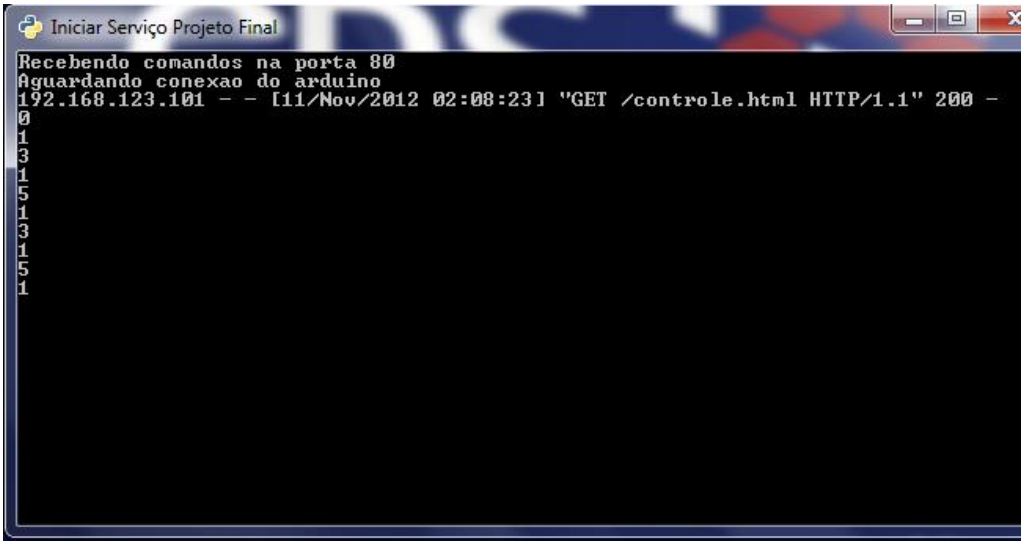


Figura 4.29 – Aplicação em Python recebendo os comandos da aplicação WEB
FONTE: (O autor)

Os comandos recebidos devem estar sendo escritos no *socket* da porta 2310. Caso tudo esteja correto, o microcontrolador deve estar enviando os comandos para a placa do controle remoto que por sua vez, envia os comandos para o dispositivo automotivo. Ao confirmar o funcionamento do sistema, é possível colocar o microcontrolador e a câmera dentro do dispositivo automotivo, finalizando o processo de integração do sistema de controle e a transmissão de vídeo.

## 4.3 – Aplicações do modelo proposto

Por ser um protótipo acadêmico, foram realizados apenas testes de implementação. Com o estudo, avaliação e investimento, nota-se a viabilidade comercial do sistema e a possibilidade de implementação nos ambientes propostos. A aplicação do protótipo é voltada inicialmente para auxiliar à equipes de resgate. Como a solução é focada no controle do dispositivo automotivo, na transmissão de vídeo e na integração dessas soluções em apenas uma única interface, o protótipo ainda necessitaria desenvolvimento no veículo, de forma que seja possível a locomoção em territórios acidentados.

## 4.4 – Resultados do Projeto

O protótipo do projeto alcançou o objetivo proposto que foi a possibilidade de controlar um dispositivo automotivo à distância através da rede wireless, a transmissão de vídeo e a integração dessas soluções em uma única interface, sendo ela acessada por um smartphone.

Certos problemas exigiram mudanças de alguns componentes. O planejamento inicial era utilizar um módulo de câmera do arduino, como mostrado na Figura 4.30, para realizar a transmissão de vídeo, porém foi constatado através de testes que o arduino não possui capacidade de processamento para transmissão de vídeo, obtendo uma média de 4 frames por minuto.

Figura 4.30 – Módulo câmera do arduino
FONTE: (O autor)

Com isso, foi optado pela utilização de uma câmera de segurança para realizar a transmissão de vídeo.

### 4.5 – Custos do Projeto

Quanto ao orçamento do projeto, este ficou na expectativa devido à característica de ser um projeto acadêmico. No decorrer do projeto, necessitou-se de outros insumos. A compra de itens no mercado nacional ajudou a elevar os custos. Com isso, a sugestão de fazer a compra destes artefatos via importação pode-se tornar muito mais vantajosa. Os custos estão representados na Tabela 4.1. Foram incluídos os valores do smartphone e do servidor, mas considerando-se que são objetos comuns no nosso dia-a-dia, o custo final do projeto fica bastante reduzido.

Tabela 4.1 – Custo total dos equipamentos utilizados no projeto

| DESCRIÇÃO | QTD | VALOR UNITÁRIO | VALOR TOTAL |
|---|---|---|---|
| Arduino Duemilanove | 01 | R$ 69,00 | R$ 69,00 |
| Shield WiFi 802.11b | 01 | R$ 229,00 | R$ 229,00 |
| Carrinho | 01 | R$ 100,00 | R$ 100,00 |
| Câmera de segurança | 01 | R$ 270,00 | R$ 270,00 |
| Smartphone | 01 | R$ 1500,00 | R$ 1500,00 |
| Servidor | 01 | R$ 2500,00 | R$ 2500,00 |
| Itens para montagem do protótipo | 01 | R$ 50,00 | R$ 50,00 |
| | | **TOTAL:** | **R$ 4718,00** |

FONTE: (O autor)

# CAPÍTULO 5 - CONCLUSÃO

## 5.1 – Conclusões do projeto

Após todo o processo de escrita e desenvolvimento do projeto final, pude agregar conhecimentos adquiridos no decorrer do curso e aplicá-los em um projeto. Durante o curso, o contato com a área de lógica digital, programação, circuitos eletrônicos, gerência de projetos, microcontroladores, redes e sistemas de tempo real proporcionaram o conhecimento necessário para a construção do projeto. Isso abrange desde o processo de planejamento do tema e pesquisa de materiais a finalização do projeto. Para desenvolver todo o processo também foram necessárias várias horas de programação e testes.

A concepção inicial do projeto foi o desenvolvimento do processo de controle utilizando um smartphone através da rede wireless e transmissão de vídeo em tempo real de um dispositivo automotivo. Foi proposto o protótipo de uma aplicação WEB que é acessada pelo smartphone, uma aplicação em Python que recebe, trata e envia os comandos para um microcontrolador *Arduino* acoplado em um Shield WiFi, uma câmera acoplada no dispositivo automotivo que envia as imagens em tempo real para a aplicação WEB, de maneira prática, arrastando o dedo pela tela do smartphone, é possível controlar o dispositivo automotivo através da rede wireless e ao mesmo tempo visualizar as imagens da câmera em uma única interface.

Durante o projeto, ocorreram alguns problemas com a transmissão de vídeo e com a transmissão de dados entre as aplicações. Em relação à transmissão de vídeo, originalmente, o planejado era utilizar uma câmera módulo do *Arduino*, porém foi constatado durante a fase de testes que o *Arduino* não tem capacidade de processamento de vídeo, resultando em uma transmissão de quatro *frames* por minuto. Como solução, foi necessário utilizar uma câmera de segurança e deixar a transmissão de vídeo independente do microcontrolador. Em relação à transmissão de dados entre as aplicações, originalmente foi planejado utilizar uma arquivo txt entre a aplicação WEB e a aplicação em Python, de forma que a aplicação WEB escrevia no arquivo e a aplicação em Python lia os comandos e apagava os dados do arquivo, permitindo que novos comandos fossem escritos, porém foi constatado que este processo gerava *delay*. Para otimizar o processo, o sistema foi alterado

para que a aplicação WEB enviasse diretamente os comandos para a aplicação em Python.

Em alguns pontos, não obteve-se os resultados de forma satisfatória, devido a uma série de fatos. Dentre eles, o alto custo das peças no Brasil que elevou o custo do projeto de maneira considerável. Outro ponto que impactou de maneira negativa o projeto foi a demora da chegada das peças que resultou no atraso geral do projeto.

Foi possível concluir neste projeto que mesmo com o elevado custo para um projeto acadêmico, a implementação é viável devido ao seu custo/benefício. Como a aplicação WEB é acessada através de um navegador presente no smartphone, não há a necessidade da instalação de um aplicativo adicional e permite que qualquer smartphone com acesso à rede que está rodando a aplicação consiga acessar a interface de controle. Ao final do projeto é possível verificar que, de fato, os objetivos traçados e planejados foram cumpridos e o dispositivo funcionou de forma satisfatória.

## 5.2 – Sugestões para projetos futuros

Como evidenciado no dia a dia e neste projeto, a necessidade de dispositivos que auxiliem equipes de resgate é muito alta. Em relação a esse projeto e utilizando o mesmo conceito, é possível aprimorar as questões referentes ao processo de controle eliminando a placa do controle remoto ao conectar o microcontrolador diretamente na placa do dispositivo automotivo, diminuindo o tempo de resposta, desenvolver uma placa intermediária para que todos os componentes que estão no dispositivo automotivo utilizem apenas uma fonte de energia e utilizar uma câmera de vídeo mais compacta por questões de peso e tamanho.

## REFERÊNCIAS BIBLIOGRÁFICAS

ATMEL. ATmega328 Preliminary Summary. San Jose: 2010. 1,2. Disponível em: http://www.atmel.com/Images/doc8271.pdf. Acesso em: 10 out. 2012.

KUROSE, James; ROSS, Keith. Redes de Computadores e a Internet – Uma Abordagem Top-Down. 5ª edição. São Paulo: Pearson, 2010.

LUTZ, Mark; ASCHER, David. Aprendendo Python – Programação orientada a objetos. 2ª edição. Porto Alegre: Bookman, 2007.

NICOLOSI, Denys. Microcontrolador 8051 Detalhado. 4ª edição. São Paulo: Editora Érica, 2004.

TANENBAUM, Andrew. Sistemas Operacionais Modernos. 2ª edição. São Paulo: Pearson, 2005.

WIRING, Wiring. Disponível em: < http://wiring.org.co/reference/ >. Acesso em: 15 out 2012.

**APÊNDICE A – Código WEB**

```
/*================================================================
PROJETO FINAL - Engenharia da Computação - UniCEUB
2o. Semestre de 2012

FLAVIO CARDOSO DE OLIVEIRA LENZI
RA: 2081732/2

DISPOSITIVO AUTOMOTIVO CONTROLADO POR SMARTPHONE
COM TRANSMISSÃO DE VÍDEO EM TEMPO REAL
================================================================*/

<html>
<head>
<meta name="viewport" content="width=310px; initial-scale=1.0; maximum-scale=1.0;">
<title>
Controle
</title>

<script type="text/javascript" charset="utf-8">

urlBase = "/enviarComando"; // Página para onde são enviadas as coordenadas

// Variáveis que definem a posição do dedo do usuário
var inicialx = 0;
var inicialy = 0;
var coordenadax = 0;
var coordenaday = 0;

// Função executado no momento do toque. Definida a posição inicial como (0,0)
document.ontouchstart = function(e){
        var controle = e.touches[0];
        inicialx = controle.pageX;
```

```
        inicialy = controle.pageY;
        document.getElementById("controle").innerHTML = "Inicio";
}

// Função executada ao final do toque. Envia para a urlBase as coordenadas (0,0), indicando que o dispositivo automotivo deve parar
document.ontouchend = function(e){
        inicialx = 0;
        inicialy = 0;
        coordenadax = 0;
        coordenaday = 0;
        document.getElementById("controle").innerHTML = "Fim";
        var xmlhttp = new XMLHttpRequest();
        xmlhttp.open("GET", urlBase+"?x="+coordenadax+"&y="+coordenaday,true);
        xmlhttp.send(null);
}

// Função executada durante o toque. Gera as coordenadas de acordo com a posição do dedo do usuário na tela
document.ontouchmove = function(e){
        e.preventDefault();
        if(e.touches.length == 1){  //  Verifica se o usuário está com o dedo na tela
                var controle = e.touches[0];
                coordenadax = controle.pageX - inicialx; // Calcula a coordenada X de acordo com o ponto inicial
                coordenaday = - (controle.pageY - inicialy); // Calcula a coordenada Y de acordo com o ponto inicial
                document.getElementById("controle").innerHTML = "Coordenadas x " + coordenadax + " y " + coordenaday; // Escreve na tela as coordenadas atuais
  }
}

// Função que envia dinamicamente as coordenadas para a urlBase
function enviaCoordenadas(){
```

```
    var xmlhttp = new XMLHttpRequest();
    xmlhttp.open("GET", urlBase + "?x="+coordenadax+"&y="+coordenaday,true); // Envia via GET para a url base as coordenadas
    xmlhttp.send(null);
}

// Define o tempo para executar a função enviaCoordenadas e cada 200ms
setInterval("enviaCoordenadas()",200);

</script>
</head>

// Divisão utilizada para mostrar para o usuário as coordenadas atuais e como camada que recebe os toques. Possui z-index:1 para que fique por cima das outras camadas
<div style="width:310px;height:260px;position:absolute;z-index:1;" id="controle">
</div>

// Divisão utilizada para exibir o iframe da transmissão do vídeo. Recebe z-index:0
<div style="z-index:0;">
        <iframe width='300px' height='400px' frameborder='0' src='http://192.168.123.102:81/pda.htm'></iframe> // IP da camera de vídeo
</div>

</body>

</html>
```

**APÊNDICE B – Código Python**

```
import serial
import socket
import SocketServer
import SimpleHTTPServer
from threading import Thread
from urlparse import urlparse, parse_qs

portaRequest = 80 # Porta utilizada pela aplicação WEB
pipe="" # Variável que recebe os comandos já tratados

# Classe gerenciadorRequest
class gerenciadorRequest(SimpleHTTPServer.SimpleHTTPRequestHandler):
	def do_GET(self): # Função do_GET
		global pipe
		if(self.path.find("/enviarComando?") != -1): # Verifica se a request possui a string /enviarComando
			comando = tratar(self.path) # Caso possua, executa a função tratar
			pipe = comando # Atribui à variável pipe o valor em comando
		else: # Caso não possua, faz o papel de um servidor HTTP
			SimpleHTTPServer.SimpleHTTPRequestHandler.do_GET(self)

# Função que recebe as coordenadas x e y e gera um comando para cada situação. Tem como retorno o comando gerado
der tratar(path):
	coordenadas = parse_qs(urlparse(path).query) # Desmembra as coordenadas da query
	x = float(coordenadas['x'][0]) # Atribui à variável x a coordenada x
	y = float(coordenadas['y'][0]) # Atribui à variável y a coordenada y

	#parar
	if (x == 0 and y == 0):
		comando = 0
```

```
        #frente = 1
        if (x>=0 and y>0 and y>=x):
                comando = 1
        if (x<0 and y>0 and y>x):
                comando = 1

        #tras = 2
        if (x<=0 and y<0 and abs(y)>=abs(x)):
                comando = 2
        if (x>0 and y<0 and abs(y)>x):
                comando = 2

        #frente-direita = 3
        if (x>0 and y>=0 and y<x):
                comando = 3

        #tras-direita = 4
        if (x>0 and y<0 and abs(y)<=x):
                comando = 4

        #frente-esquerda = 5
        if (x<0 and y>=0 and y<abs(x)):
                comando = 5

        #tras-esquerda = 6
        if (x<0 and y<0 and abs(y)<abs(x)):
                comando = 6

        return comando # retorna o comando tratado

# Classe ServicoProxyArduino
class ServicoProxyArduino:
```

```
def __init__(self, host, port): # Função construtor. Define alguns valores iniciais para o objeto
        self.message = ""
        self.host = host
        self.port = port


def escutarSocket(self): # Função que realiza a comunicação com o microcontrolador
        self.novoSocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.novoSocket.bind((HOST,PORT))
        self.novoSocket.listen(0) # sem fila
        print  "Aguardando conexao do arduino"
        while(True):
                self.conexao, self.endereco_arduino_wifly = self.novoSocket.accept()
                try:
                        while(True): # Caso a conexao seja bem sucedida, executa a função repassarComandos
                                self.repassarComandos()
                except socket.error: # Caso a conexão seja perdida, o envio dos comandos é interrompido até que uma nova conexão seja estabelecida
                        print  "Conexao encerrada. Aguardando nova conexao do arduino"

def repassarComandos(self): # Executa as funções lerProximoComando e tratarComando
        comando = self.lerProximoComando()
        self.tratarComando(comando)

def lerProximoComando(self): # Armazena o próximo comando, já tratado
        global pipe
        return str(pipe)
```

```
        def tratarComando(self, comando): # Função tratarComando
                self.tmp = ""
                self.tmp = comando
                if (self.tmp != self.message): # Verifica se o comando recebido é diferente ao último comando recebido
                        self.enviarComandoParaArduino(comando) # Caso seja diferente, executa a função enviarComandoParaArduino
                        self.message = self.tmp
                        print self.message

        def enviarComandoParaArduino(self, comando): # Envia o comando para o microcontrolador  através do método sendall
                self.conexao.sendall(comando)

HOST='' #quando nao define nenhum nome de host, o servico ouve em todos os nomes disponiveis no sistema
PORT=2310 # Porta utilizada pela aplicação WEB

def wifiMain(): # Executa paralelamente a comunicação com o smartphone e microcontrolador
        threadSmartphone = Thread(target=iniciarServicoSmartphone)
        threadSmartphone.start()
        servico = ServicoProxyArduino(HOST, PORT)
        servico.escutarSocket()

def iniciarServicoSmartphone(): # Cria o socket que utiliza a porta 80 e executa a função gerenciadorRequest
        print ''Recebendo comandos na porta'', portaRequest
        httpd = SocketServer.ThreadingTCPServer((', portaRequest), gerenciadorRequest)
        httpd.serve_forever()

wifiMain()
```

**APÊNDICE C – Código Arduino**

```
#include "Wifly.h"
#include <SoftwareSerial.h>
WiflyClass Wifly(2,3);

// Definição dos pinos utilizados pelo microcontrolador e da variável que recebe os comandos do socket
int pino10 = 10;
int pino11 = 11;
int pino12 = 12;
int pino13 = 13;
char comando;

void setup(){

  Serial.begin(115200);
  Wifly.init();
  Wifly.setConfig("Connectify-Flavio","projetofinal"); // Se conecta na rede chamada Connectify-Flavio utilizando a senha projetofinal
  Wifly.join("Connectify-Flavio");
  Wifly.checkAssociated();
  while(!Wifly.connect("192.168.123.1","2310")); // Utiliza a porta 2310 do IP 192.168.123.1, referente a rede Connectify-Flavio
  Wifly.writeToSocket("Connected!"); // Se bem sucedida, escreve no socket Connected

// Inicialização dos pinos utilizados
  pinMode(pino10, OUTPUT); //frente
  pinMode(pino11, OUTPUT); //tras
  pinMode(pino12, OUTPUT); //direita
  pinMode(pino13, OUTPUT); //esquerda
}

void loop(){
```

```
  while(Wifly.canReadFromSocket()){ //verifica se alguma mensagem foi enviada do servidor

    // Lê as informações do socket e as escreve na variável comando
    comando = char(Wifly.readFromSocket());

      // Para cada comando recebido, executa uma ação diferente. Para o comando 0, não manda sinal para nenhum pino, indicando para parar.
      if(comando == '0'){ // parar
        digitalWrite(pino10, LOW);
        delay(100);
        digitalWrite(pino11, LOW);
        delay(100);
        digitalWrite(pino12, LOW);
        delay(100);
        digitalWrite(pino13, LOW);
      }
      if(comando == '1'){ // frente
        digitalWrite(pino10, HIGH);
        delay(100);
        digitalWrite(pino11, LOW);
        delay(100);
        digitalWrite(pino12, LOW);
        delay(100);
        digitalWrite(pino13, LOW);
      }
      if(comando == '2'){ // tras
        digitalWrite(pino10, LOW);
        delay(100);
        digitalWrite(pino11, HIGH);
        delay(100);
        digitalWrite(pino12, LOW);
        delay(100);
        digitalWrite(pino13, LOW);
```

```
}
if(comando == '3'){ // frente-direita
  digitalWrite(pino10, HIGH);
  delay(100);
  digitalWrite(pino11, LOW);
  delay(100);
  digitalWrite(pino12, HIGH);
  delay(100);
  digitalWrite(pino13, LOW);
}
if(comando == '4'){ // tras-direita
  digitalWrite(pino10, LOW);
  delay(100);
  digitalWrite(pino11, HIGH);
  delay(100);
  digitalWrite(pino12, HIGH);
  delay(100);
  digitalWrite(pino13, LOW);
}
if(comando == '5'){ // frente-esquerda
  digitalWrite(pino10, HIGH);
  delay(100);
  digitalWrite(pino11, LOW);
  delay(100);
  digitalWrite(pino12, LOW);
  delay(100);
  digitalWrite(pino13, HIGH);
}
if(comando == '6'){ // tras-esquerda
  digitalWrite(pino10, LOW);
  delay(100);
  digitalWrite(pino11, HIGH);
  delay(100);
  digitalWrite(pino12, LOW);
```

```
        delay(100);
        digitalWrite(pino13, HIGH);
      }
  }
}
```

**ANEXO A – Wifly.h**

```
#ifndef WIFLY_H
#define WIFLY_H

#include <Arduino.h>
#include "Hardware.h"
#include "Debug.h"
#include <SoftwareSerial.h>

#define NB_TRY_BEFORE_REBOOT 10
#define READ_TIMEOUT 10000
//
class WiflyClass: public SoftwareSerial {
//class WiflyClass {
public:
	WiflyClass(int,int);
	void init();

//WiflyClass:WiflySerial(int,int);
//		{
//			WiflySerial = new SoftwareSerial(int,int);
//		}
	//SoftwareSerial WiflySerial = new SoftwareSerial(tx,rx);
	//SoftwareSerial::SoftwareSerial WiflySerial = new SoftwareSerial::SoftwareSerial(tx,rx);
	void join(const char *ssid);
	void waitForReady(boolean dhcp);
	void setConfig(const char *ssid, const char *passphrase);
	void setConfig(const char *ssid, const char *passphrase, const char *ip, const char *mask ,const char *gateway);
	void reset();
	void closeAndExit();
	void reboot();
```

```
        bool checkConnection();
        void checkAssociated();
    bool connect(const char *host, const char *port);
    void writeToSocket(const char *data);
    char readFromSocket();
        bool canReadFromSocket();
        bool sendCommand(const char *cmd, char *expectedReturn, boolean multipart=false);
        void factoryReset();
        char Read();
        void flush();
        bool find(char *target);
        bool findUntil(char *target, char *terminate);
        void skipChar(int count);
        void print(char data);
private:
        unsigned char disconCount;
        bool _dhcp;
        uint8_t unavailConn;
        //SoftwareSerial WiflySerial;
};

//extern WiflyClass Wifly;
//extern SoftwareSerial WiflySerial(int,int);
#endif // WIFLY_H
```

**ANEXO B – Hardware.h**

```
#ifndef HARDWARE_H
#define HARDWARE_H

#define SDFAT_CHIP_SELECT 77
#define VS1011_RST 73
#define VS1011_XDCS 72
#define VS1011_XCS 71
#define VS1011_DREQ 70
#define WIFLY_RST 49
#define WIFLY_GPIO6 76
#define GREEN_LED 78
#define RED_LED 79

#endif // HARDWARE_H
```

**ANEXO C – Debug.h**

```
#ifndef __DEBUG_H__
#define __DEBUG_H__

static const char sChar = 0x2;
static const char eChar = 0x3;
static const char dStringStart = ':';

//#define DEBUG
#define DEBUG_LEVEL 0

#define DEBUG_LOG(level, message) \
  if (DEBUG_LEVEL >= level) {\
          Serial.print(sChar);\
          Serial.print(dStringStart);\
     Serial.print(message);\
          Serial.print(eChar);\
  };
#endif
```