FERNANDO MASCAGNA BITTENCOURT LIMA

CARRO INSPETOR RÁDIO CONTROLADO COM TRANSMISSÃO DE IMAGEM EM TEMPO REAL PARA INSPEÇÃO DE LOCAIS DE DIFÍCIL ACESSO

São Carlos

2015

FERNANDO MASCAGNA BITTENCOURT LIMA

CARRO INSPETOR RÁDIO CONTROLADO COM TRANSMISSÃO DE IMAGEM EM TEMPO REAL PARA INSPEÇÃO DE LOCAIS DE DIFÍCIL ACESSO

Trabalho de Conclusão de Curso apresentado à Escola de Engenharia de São Carlos, da Universidade de São Paulo

Curso de Engenharia de Computação

ORIENTADOR: Prof. Dr. Evandro Luís Linhari Rodrigues

São Carlos

2015

AUTORIZO A REPRODUÇÃO TOTAL OU PARCIAL DESTE TRABALHO, POR QUALQUER MEIO CONVENCIONAL OU ELETRÔNICO, PARA FINS DE ESTUDO E PESQUISA, DESDE QUE CITADA A FONTE.

Lima, Fernando Mascagna Bittencourt
L732c CARRO INSPETOR RÁDIO CONTROLADO COM TRANSMISSÃO DE IMAGEM EM TEMPO REAL PARA INSPEÇÃO DE LOCAIS DE DIFÍCIL ACESSO / Fernando Mascagna Bittencourt Lima; orientador Evandro Luís Linhari Rodrigues. São Carlos, 2015.

Monografia (Graduação em Engenharia de Computação) -- Escola de Engenharia de São Carlos da Universidade de São Paulo, 2015.

1. Inspeção. 2. Raspberry-Pi. 3. Arduino. 4. Linux Embarcado. 5. NRF24L01. 6. Câmera. I. Título.

FOLHA DE APROVAÇÃO

Nome: Fernando Mascagna Bittencourt Lima

Título: "Carro inspetor rádio controlado com transmissão de imagem em tempo real para inspeção de locais de difícil acesso"

Trabalho de Conclusão de Curso defendido em 26/06/2015.

Comissão Julgadora:	**Resultado:**
Prof. Associado Evandro Luís Linhari Rodrigues (Orientador) - SEL/EESC/USP	Aprovado
Prof. Dr. Eduardo do Valle Simões SSC/ICMC/USP	Aprovado
Prof. Dr. Valdir Grassi Júnior SEL/EESC/USP	Aprovado

Coordenador do Curso Interunidades - Engenharia de Computação:

Prof. Associado Evandro Luís Linhari Rodrigues

A Deus, por ser minha fonte de sustento e me ter dado força para poder ter chegado até aqui. A meu pai, minha mãe, meus irmãos e meus amigos que me apoiaram durante esta longa e dura jornada.

Agradecimentos

A Deus, por ter me dado saúde e força para superar todas as dificuldades encontradas no caminho.

Aos meus pais, Benedito e Janete, e aos meus irmãos, Felipe e Fábio, que sempre me incentivaram, apoiaram e me ajudaram na medida do possível.

A meus amigos, que me ajudaram a solucionar problemas encontrados neste projeto.

Ao Prof. Evandro, que me ensinou, motivou e orientou ao longo deste projeto.

Aos professores, pelos ensinamentos técnicos e de vida.

Resumo

Em muitas situações na atualidade é necessário fazer algum tipo de inspeção em um determinado local para verificar algo desejado, sejam elas em procura de vítimas em acidentes ou incêndios, atividades militares, entre outras. Levando isto em conta, o objetivo deste trabalho é desenvolver uma estrutura móvel capaz de fazer a inspeção destes locais transmitindo imagens em tempo real, eliminando a necessidade de uma pessoa realizar tal inspeção. Para isso, foram desenvolvidos um controle remoto possuindo um display gráfico e uma estrutura mecânica no formato de um carro composto por quatro rodas, dois motores para movimentação, um servomotor para a movimentação do eixo de direção e dois para movimentação da câmera. Foram utilizadas as plataformas Arduino e Raspberry Pi para realizarem o controle dos motores e servomotores e para fazer a leitura dos dados da câmera e visualização das imagens capturadas em um display respectivamente. Tanto a estrutura quanto o controle remoto possuem um Arduino Nano e uma Raspberry Pi. A Raspberry Pi opera com um sistema operacional Linux embarcado. Para fazer a comunicação entre os Arduinos foram utilizados módulos de rádio NRF24L01 e para efetuar a comunicação entre as Raspberry Pi, adaptadores wireless USB. Este projeto visa ser uma solução de baixo custo e de fácil implementação. Após a finalização do trabalho, o sistema atendeu aos objetivos e se mostrou capaz para realizar inspeções.

Palavras-chave: Inspeção, Raspberry-Pi, Arduino, Linux Embarcado, NRF24L01, Câmera.

Abstract

In many nowadays situations some kind of inspection is necessary at a certain dangerous place to check up something, whether in search of victims in accidents or fires, military activities, among others. Taking this into account, the objective of this work it is to develop a mobile structure, able to do the inspection in these hard reach places, that transmit images of these places in real time, therefore, the necessity of an inspection person is eliminated. For this, a remote control containing a graphic display and a structural automotive shape was developed and it is composed by four wheels, moved with two motors, a servo motor for moving the steering shaft and two servo motors for camera movement. The Arduino and Raspberry Pi platforms were used to perform the control of motors and servo motors and to read the camera data and image captured on a display respectively. Both the structure and the remote control have an Arduino Nano and Raspberry PI. The Raspberry Pi works with an embedded Linux operating system. To make communication between the Arduinos, two radio modules NRF24L01 have been used and for the Raspberries Pi USB wireless adapters were adopted. This project aims to be a solution of low cost and easy to implement, what makes it viable. By the end of the project, the device has achieved its objectives as it has shown capability to do the inspections.

Keywords: Inspection, Raspberry-Pi, Arduino, Embedded Linux, NRF24L01, Camera.

Lista de Siglas

ARM – Advanced RISC Machines
API - Application Programming Interface
GPIO – General Pourpose Input Oputput
IP - Internet Protocol
LCD – Liquid Crystal Display
PCI – Placa de circuito impresso
PWM – Pulse-width Modulation
SD – Secure Digital
SSH – Secure Shell
SO – Sistema Operacional
SPI – Serial Peripheral Interface
TFT - Thin Film Transistor
USB – Universal Serial Bus

Sumário

1 Introdução

1.1 Introdução

Em muitas situações na atualidade existe a necessidade de se realizar algum tipo de inspeção em um determinado local para verificar-se algo desejado, sejam elas em procura de vítimas em acidentes ou incêndios, atividades militares, entre outras. Em muitas destas situações, não é possível que esta inspeção seja feita por uma pessoa devido ao alto risco que este local pode proporcionar ao inspetor.

Levando isto em consideração, este trabalho se trata do desenvolvimento de uma plataforma móvel capaz de fazer a inspeção destes locais de difícil acesso transmitindo imagens destes locais em tempo real, eliminando a necessidade de uma pessoa para realizar tal inspeção. Para facilitar o uso desta plataforma, um display posicionado no controle remoto do operador da plataforma mostra em tempo real a imagem capturada a fim de direcionar o operador a seguir o caminho correto na inspeção.

O desenvolvimento deste trabalho é baseado no uso de software e de hardware *open source*[1]. Havia um certo ceticismo com relação às plataformas *open sourc*e[1], porém, com o passar do tempo, tornou-se evidente que este modelo de desenvolvimento é bastante sólido e eficiente.

Com o desenvolvimento crescente da eletrônica e a modernização dos sistemas embarcados, os benefícios proporcionados pelo hardware *open source*[1] têm conquistado como aliado o Sistema Operacional Linux, também *open source*[1].

Uma vantagem do desenvolvimento em Linux embarcado é que o Linux oferece uma camada de abstração implícita acima da camada de hardware, o que permite que não seja necessário um desenvolvimento baseado na configuração direta de registradores, onde a mudança para uma nova família de microcontroladores significaria reescrever grande parte do software já desenvolvido. Desta forma, caso seja necessária uma migração entre processadores, um sistema operando com Linux não exige a necessidade de um redesenvolvimento de firmware.

Este projeto visa ser uma solução de baixo custo, o que exige que tecnologias *open source*[1] sejam utilizadas.

Como prova de caso, a estrutura do carro foi proposta para se movimentar em terrenos planos, limpos e sem obstáculos. Mudando-se a estrutura, o comando pode ser utilizado e adaptado a situações distintas.

1.2 Objetivos

1.2.1 Objetivos Gerais

O principal objetivo deste trabalho é realizar a integração de software de alto nível com sistemas eletrônicos, mecânicos e eletromecânicos a fim de se desenvolver uma estrutura capaz de realizar inspeções radio controladas.

1.2.2 Objetivos Específicos

A plataforma móvel que será capaz de realizar inspeções deve apresentar as seguintes características:

- Utilização de hardware *open source*[1];
- Utilização de Linux embarcado;
- Interface entre o operador e a plataforma;
- Comunicação wireless entre dispositivos;
- Controlabilidade intuitiva;
- Baixo custo de desenvolvimento;

1.3 Justificativa

O desenvolvimento completo desta plataforma móvel exige o estudo e conhecimentos multidisciplinares, tais como temas de engenharia eletrônica, engenharia de computação e engenharia mecânica.

Este trabalho permite que acidentes envolvendo inspeções em locais de difícil acesso possam, em alguns casos, vir a ser eliminados além de permitir que locais em que antes o acesso era inviável se tornem viáveis.

Como o desenvolvimento da plataforma é baseado em hardware e software *open source*[1], o custo para o seu desenvolvimento é bastante baixo, o que possibilita que este trabalho seja replicado e difundido.

1.4 Organização do Trabalho

Este trabalho está estruturado em mais quatro capítulos, sendo que estes estão divididos da seguinte forma:

- **Fundamentação Teórica:** apresentação dos conceitos relevantes para possibilitar o entendimento do trabalho;
- **Desenvolvimento:** apresentação dos materiais utilizados e do desenvolvimento eletrônico, mecânico e computacional da plataforma móvel e do controle remoto;
- **Resultados:** apresentação e discussão dos resultados obtidos;
- **Conclusão:** validação dos objetivos do trabalho e considerações finais.

2 Fundamentação Teórica

A elaboração deste projeto envolve a utilização de diferentes componentes, cada um com suas funcionalidades e particularidades. Nas subseções seguintes será desenvolvido um breve embasamento teórico de alguns dos componentes e tecnologias utilizados ao longo do desenvolvimento deste trabalho.

2.1 Arduino

O Arduino[2] foi projetado e desenvolvido com a finalidade de ser de uma plataforma de fácil entendimento, de fácil programação e de fácil aplicação, sendo também multi plataforma, o que possibilita a sua utilização em diferentes sistemas operacionais como Linux, Mac OS e Windows. Um grande diferencial desta plataforma é ser mantido por uma comunidade que trabalha com a filosofia *open source*[1], desenvolvendo e divulgando gratuitamente seus projetos.

De acordo com o site oficial [2], o Arduino[2] é uma plataforma de prototipagem eletrônica de hardware livre, com suporte de entrada e saída embutido e uma linguagem de programação padrão. O objetivo desse projeto de hardware livre é criar ferramentas que são acessíveis, com baixo custo, flexíveis e de fácil utilização por artistas e amadores.

O Arduino[2] é utilizado em vários programas educacionais ao redor do mundo, possibilitando o ensino de robótica para pessoas com um baixo nível de conhecimento mais detalhado sobre a arquitetura da plataforma.

Arduino[2] é muito conhecido pelo seu hardware, porém é necessário um software para programar esse hardware. Este software se consiste de uma IDE intuitiva e de fácil utilização. Tanto o software como o hardware são chamados de "Arduino[2]". A combinação dos dois permite a criação de projetos com diversas finalidades.

2.2 Raspberry Pi

A Raspberry Pi[3] é uma plataforma aberta pronta para uso e prototipagem rápida de hardware, desenvolvimento de software e firmware desenvolvida no Reino Unido pela Fundação Raspberry Pi[3]. Todo o hardware é integrado numa única placa. O principal objetivo é promover o ensino em Ciência da Computação básica em escolas. A Fundação Raspberry Pi começou a comercializar a Raspberry Pi[3] a partir de 29 de fevereiro de 2012.

A comunidade raspberrypi.org[4] é uma comunidade *open source*[1] que fornece aos desenvolvedores e entusiastas os recursos de que precisam para conceber novas ideias e desenvolver rapidamente novos produtos para o mercado. A comunidade hospeda os mais recentes desenvolvimentos de software, fóruns e chats ao vivo e interativos que colaboram para um fácil desenvolvimento de soluções na plataforma. A organização também criou

várias plataformas de hardware para ajudar a simplificar desenvolvimentos baseados em ARM[5].

Uma característica interessante da Raspberry Pi[3] é que ela pode ser ligada a uma outra Raspberry Pi[3] ou qualquer computador Linux via USB ou Ethernet e operar como um módulo de expansão para ele. Além disso, a Raspberry Pi permite a expansão em módulos conhecidos como *shields*. Esse conceito é o mesmo das *shields* do Arduino[2], permitindo adicionar periféricos de maneira simples e altamente customizável.

Existem diversas plataformas de hardware *open source*[1] disponíveis no mercado, como por exemplo a Beagle Bone[6], que tem ganhado bastante visibilidade. Algumas das vantagens da Beagle Bone[6] incluem um alto poder de processamento de imagens e uma comunidade extremamente ativa além de possuir um grande número de portas de entrada e saída e possuir uma memória interna não volátil relativamente grande. Entretanto, existem algumas características da Raspberry Pi[3] que justificam a sua escolha em detrimento da Beagle Bone[6]. Essas características envolvem o seu baixo custo de mercado e uma comunidade mais ativa.

Essas características fazem da plataforma um sucesso em aplicações como, por exemplo, o desenvolvimento de robôs autônomos ou controlados, kits educacionais de eletrônica e programação, dispositivos de jogos, automação residencial, processamento moderado de imagens, entre outras. A Raspberry Pi[3] é uma plataforma de desenvolvimento profissional adequada para engenheiros, designers, desenvolvedores e *hobbistas*.

O sistema operacional padrão que é distribuído para operar com a Raspberry Pi[3] é conhecido como Raspbian[7]. Este SO nada mais é do que a distribuição Linux Debian[8] compilada para operar em conjunto com o hardware da Raspberry Pi[3]. O modelo mais novo desenvolvido, a Raspberry Pi 2 Model B possibilita também o uso do SO Windows 10.

2.3 Linux Embarcado

Existem muitos sistemas operacionais para um sistema embarcado, tanto proprietárias como de código aberto. O Linux é uma destas opções.

Não importa o que será utilizado pela máquina de desenvolvimento, seja Linux, Windows ou Mac. É necessário aprender a programar usando o sistema operacional alvo. A este respeito, utilizar Linux embarcado não é muito diferente de se utilizar VxWorks, WindowCE, ou outro sistema operacional. É necessária uma compreensão de como o sistema operacional foi projetado, como configurar o SO, e como programar usando sua API.

Alguns fatores fazem com que aprender a programar em Linux seja mais fácil do que outros sistemas operacionais embarcados. Existem muitos mais livros e tutoriais sobre o Linux, bem como Unix a partir do qual é derivado, do que para outros sistemas operacionais.

Recursos online para Linux são amplos, enquanto outros sistemas operacionais têm uma presença muito menor ou é resguardado pelo fabricante do sistema operacional. Linux é *open source*, e é possível ler o código para obter uma compreensão de exatamente o que o OS está fazendo, algo que muitas vezes é impossível com um sistema operacional proprietário distribuído como binários.

O fator mais significativo que diferencia o Linux dos outros sistemas operacionais é que o mesmo kernel é usado para todos os sistemas, desde as menores placas até sistemas de desktop e grandes servidores. Isso significa que é possível aprender a programar em Linux em um desktop que é muito mais flexível do que usar uma placa-alvo com todas as complexidades de se conectar ao alvo, fazer o download de um programa e realizar a execução dos testes. Todos os conceitos básicos da maioria das APIs são os mesmos para um desktop Linux e um Linux embarcado.

O que diferencia um Linux embarcado de um Linux desktop é que o primeiro é compilado para a arquitetura alvo em que será executado, o que geralmente implica em uma eliminação de funcionalidades para reduzir o tamanho do SO e deixar o sistema mais leve. Isso é interessante devido às plataformas embarcadas serem limitadas em memória se comparadas a um desktop moderno.

2.4 Acess Point

Um *Acess Point*[9] ou ponto de acesso é um dispositivo que, assim como um roteador sem fio, permite que dispositivos sem fio se conectem a uma rede. A maioria dos *Acess Points*[9] estão embutidos em roteadores, enquanto outros devem ser conectados a um roteador a fim de fornecer acesso à rede. Em ambos os casos, os pontos de acesso são tipicamente programados para outros dispositivos, tais como *switches* de rede ou modems de banda larga.

Os pontos de acesso podem ser encontrados em muitos lugares, incluindo casas, empresas e locais públicos. Na maioria das casas, o ponto de acesso é um roteador sem fio, que é conectado a um modem DSL ou a um cabo. No entanto, alguns modems podem incluir capacidades sem fio, tornando a si mesmo um ponto de acesso. As grandes empresas muitas vezes oferecem vários pontos de acesso, o que permite que os funcionários se conectem a uma rede sem fio central de uma grande variedade de locais. Pontos de acesso público podem ser encontrados em lojas, cafés, restaurantes, bibliotecas e outros locais. Algumas cidades fornecem pontos de acesso públicos sob a forma de transmissores sem fio que estão conectados a postes e outros objetos públicos.

Enquanto pontos de acesso tipicamente oferecem acesso sem fio à Internet, alguns são destinados apenas para fornecer acesso a uma rede fechada. Por exemplo, uma empresa pode fornecer pontos de acesso seguro aos seus empregados para que eles possam acessar arquivos sem fio a partir de um servidor de rede.

A maioria dos pontos de acesso proporciona acesso Wi-Fi, porém é possível para um ponto de acesso se referir a um dispositivo Bluetooth ou outro tipo de conexão sem fio.

O termo "*Acess Point*[9]" é muitas vezes usado como sinônimo de estação de base, embora estações base sejam tecnicamente apenas os dispositivos Wi-Fi. Ela também pode ser abreviada como AP ou WAP (*Wireless Access Point*). No entanto, WAP não é tão comumente usado como AP devido a WAP ser a sigla padrão para *Wireless Access Protocol*.

Neste trabalho, foi criado um *Acess Point*[9] com a finalidade de o mesmo ser o gerenciador da rede. Este *Acess Point*[9] está presente no carro e é responsável por permitir a conexão entre as duas Rasperries Pi[3] presentes o carro e no controle remoto. Sua função será permitir a transmissão de imagens e a transferência de arquivos em tempo real.

2.5 Motion

O “Motion”[10] é um programa que controla um sinal de vídeo a partir de uma ou mais câmaras e é capaz de detectar se uma parte significativa da imagem mudou. Em outras palavras, ele é capaz de detectar movimento.

O programa é escrito em C e é feito para o sistema operacional Linux. Ele é uma ferramenta baseada em linhas de comando. Ela não tem absolutamente nenhuma interface gráfica do usuário. Toda a configuração é feita através de linha de comando ou através de um conjunto de arquivos de configuração (arquivos ASCII simples que podem ser editados por qualquer editor ASCII). As saídas do “Motion”[10] podem ser arquivos .jpg, ppm ou sequências de vídeo MPEG.

Optou-se por utilizar este software devido ao fato de ser um software *open source*[1], disponível no repositório do Raspbian[7] e executável como *deamon.* Este software permite também que processamento de imagens seja realizada, de forma que, caso seja necessário o processamento das imagens capturadas antes de transmiti-las para o controle remoto, será possível utilizá-lo para esta função.

2.6 Servidor Apache

O projeto “Apache HTTP Server”[11] é um esforço para desenvolver e manter um servidor HTTP de código aberto para sistemas operacionais modernos, incluindo UNIX e

Windows NT. O objetivo deste projeto é fornecer um servidor seguro, eficiente e extensível que fornece serviços HTTP em sincronia com os padrões HTTP atuais.

Apache HTTP, lançado em 1995, tem sido o servidor web mais popular na Internet desde abril de 1996. O Apache HTTP Server ("httpd") é um projeto da Apache Software Foundation.

3 Desenvolvimento

Para realizar o desenvolvimento deste trabalho, foi necessário desenvolver um carro rádio controlado e um controle remoto. O desenvolvimento dos mesmos foi realizado baseando-se no diagrama presente na Figura 3.1.

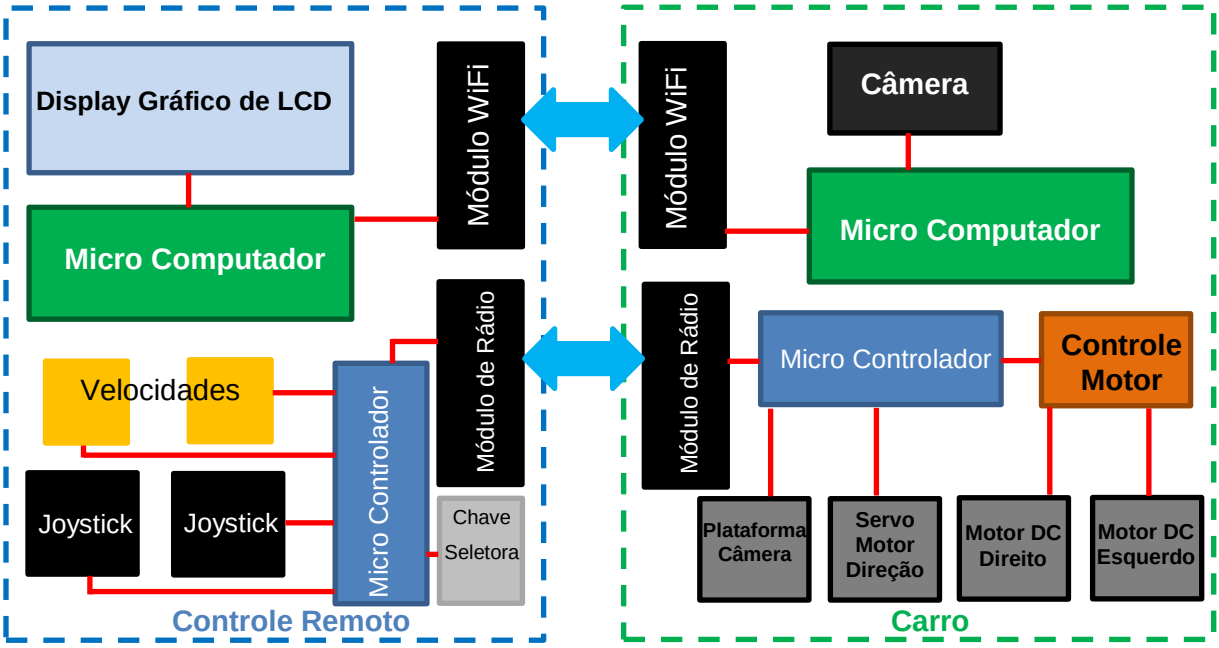

Figura 3.1 – Diagrama de Conexão dos Componentes presentes no Projeto

Podemos notar pelo diagrama presente na Figura 3.1, que o trabalho como um todo é constituído por dois sistemas independentes: um responsável pela movimentação do carro e outro responsável pela transmissão de imagem. Esta separação foi realizada para facilitar o desenvolvimento e para evitar possíveis falhas no sistema. Os meios de comunicação sem fio são independentes e a comunicação da movimentação apresenta um alcance maior, o que faz com que, caso haja perda do sinal por parte da imagem, seja possível retornar o carro para um local no qual a imagem volte a ser transmitida. As reconexões são realizadas automaticamente quando a conexão é desfeita, desde que o carro esteja na região de alcance dos módulos sem fio presentes no controle remoto.

Para descrever o desenvolvimento do trabalho como um todo, esta seção é dividia em quatro subseções: Materiais Utilizados, Desenvolvimento do Controle Remoto, Desenvolvimento do Carro e Conexão dos Componentes.

3.1 Materiais Utilizados

Para a realização desse projeto foram utilizados diversos materiais. Nesta Seção é feita uma breve descrição dos principais componentes utilizados bem como as suas especificações técnicas principais.

3.1.1 Arduino Nano

A versão do Arduino[2] utilizada nesse projeto é conhecida como Arduino Nano[10] e pode ser visto na Figura 3.2.

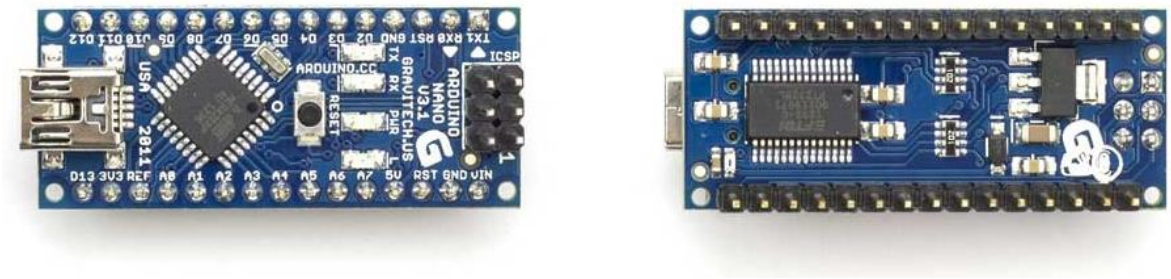

Figura 3.2 - Arduino Nano.

O Arduino Nano[12] é uma placa microcontroladora baseada no chip ATmega328 (Arduino Nano 3.x) ou ATmega168 (Arduino Nano 2.x). Neste projeto foi utilizada a versão 3.0. A principal diferença entre o Arduino Nano[12] e outras versões é o seu tamanho reduzido mantendo todas as funcionalidades do Arduino Uno[13], já que ambos utilizam o mesmo microcontrolador. Isso faz com que ela seja uma placa relativamente robusta e com dimensões reduzidas.

Esse modelo foi escolhido devido a seu baixo custo de mercado e principalmente devido a possuir uma área bastante reduzida. Qualquer um dos outros modelos poderiam ter sido utilizados, com uma necessidade apenas de ajustar poucos detalhes de programação e ajustar a PCI que recebe a placa.

A Tabela 3.1 resume as especificações técnicas do Arduino Nano[12] utilizado.

Tabela 3.1 - Características do Arduino Nano.

Microcontrolador	ATmega328
Tensão de Operação	5 V
Tensão de Entrada (recomendado)	7 – 12 V
Tensão de Entrada (limites)	6 – 20 V
Pinos digitais de entrada e saída	14
Canais PWM	6

Canais de entrada analógica	8
Corrente DC por pino de entrada/saída	40 mA
Memória Flash	32 kB
SRAM	2 kB
EEPROM	1 kB
Clock Speed	16 MHz

3.1.2 Raspberry Pi

A solução de hardware utilizada neste projeto para executar uma distribuição Linux embarcada foi a Raspberry Pi [3]. Neste projeto foram utilizados dois modelos da placa, o modelo B e o modelo B+. Ambos podem ser vistos na Figura 3.3.

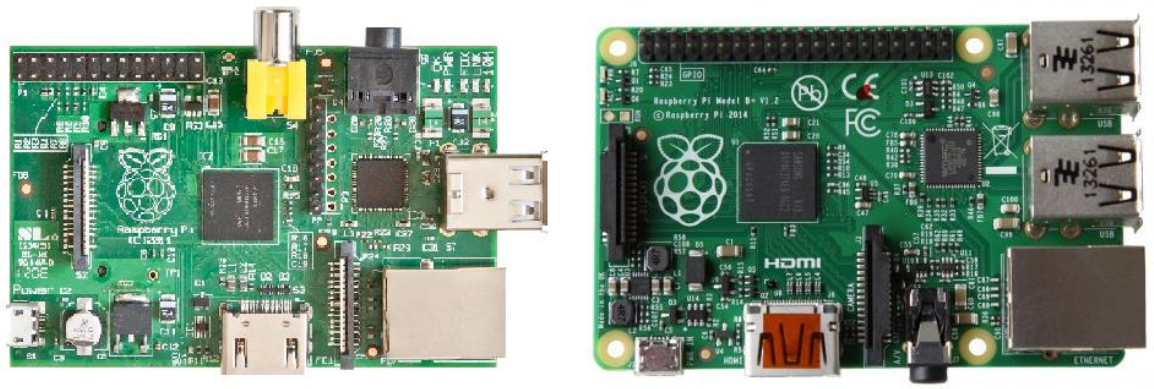

Figura 3.3 - Raspberry Pi modelo B (esquerda) e modelo B+ (direita).

O Raspberry Pi[3] é um computador de baixo custo do tamanho de um cartão de crédito que se pode se conectar a um monitor de computador ou TV. Ele é um pequeno dispositivo que permite que as pessoas de todas as idades possam explorar a computação e aprender a programar em linguagens como Scratch e Python. Ele é capaz de fazer tudo que você esperaria de um computador desktop como navegar na internet, reprodução de vídeo de alta definição, fazer planilhas, processamento de texto, jogar jogos, entre outros, porém com uma certa limitação de memória RAM e de clock.

Foram utilizados dois modelos diferentes devido à disponibilidade do uso das duas. Poderiam ter sido utilizadas tanto duas placas do modelo B quanto duas do modelo B+.

Ambos os modelos não possuem memória não volátil, de forma que tanto o sistema operacional quanto os programas e arquivos gerados são gravados em um cartão de memória SD (modelo B) ou em um cartão de memória micro SD (modelo B+).

A Tabela 3.2 resume as especificações técnicas dos modelos de Raspberry Pi[3] utilizados.

Tabela 3.2 - Características dos modelos B e B+ da placa Raspberry Pi.

	Modelo B	**Modelo B+**
Processador	BCM2835	BCM2835
Memória RAM	512 MB	512 MB
Alimentação	5 V micro USB	5V micro USB
Pinos Entrada/Saída	26	40
Portas USB	2	4
Slot de Cartão de Memória	SD	Micro SD
Massa	45 gramas	45 gramas

3.1.3 Bateria

As baterias utilizadas neste projeto são baterias do tipo LiPo[14] (Lítio Polímero). A imagem das baterias utilizadas pode ser vista na Figura 3.4.

A bateria de LiPo[12] é uma bateria feita com um polímero de Lítio. É uma bateria relativamente leve e de baixa manutenção, uma vantagem que a maioria das baterias químicas não possuem. Nenhum ciclismo programado é necessário para prolongar a vida da bateria e a mesma não possui nenhum tipo de memória. Além disso, a auto-descarga é inferior à metade em relação à bateria de níquel-cádmio, o que a torna adequada para aplicações que exigem alta energia.

Cada célula da bateria possui uma tensão nominal de 3.7 V, sendo que sua tensão de carga máxima atinge 4,2 V. A máxima corrente de descarga da bateria é proporcional à carga da mesma multiplicada por uma constante de descarga específica de cada bateria. O mesmo vale para a máxima corrente de carga, porém utilizando uma constante de carga.

Figura 3.4 - Bateria LiPo 2S (esquerda) e 3S (direita).

Foram utilizadas duas baterias, uma de duas células (7,4 V) com capacidade de carga de 1300 mAh e uma de três células (11,1 V) e capacidade de carga de 2200 mAh. A de duas células foi escolhida para ser utilizada no controle remoto, visto que todo o controle remoto opera em 5V e a de 3 células para ser utilizada no carro, visto que o mesmo possui motores descritos na seção 3.1.6 que necessitam de uma tensão de cerca de 12 V.

Este tipo de bateria foi utilizado no projeto devido ao seu baixo custo e peso e ao fato de possuir um tempo de recarga baixo e uma corrente de descarga elevada.

3.1.4 Regulador de Tensão

O regulador de tensão utilizado neste projeto é um conversor DC-DC chaveado abaixador de tensão do tipo “buck”[15] do modelo LM2596S e pode ser visto na Figura 3.5. A tensão de saída pode ser ajustada através de seu “trimpot”. Suas características mais relevantes para o projeto estão descritas na Tabela 3.3.

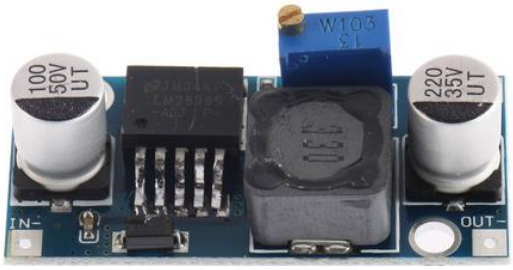

Figura 3.5 - Regulador de Tensão Abaixador de Tensão tipo “Buck”.

Tabela 3.3 - Características do Regulador de Tensão utilizado.

Tensão de Entrada	3.2V - 40V
Tensão de Saída	1.25V – 35V
Corrente máxima de saída	3A
Eficiência de transferência	92%
Frequência de chaveamento	65 KHz

Este regulador de tensão foi utilizado para converter a tensão das baterias para 5 V para alimentar tanto os Arduinos[2] quanto as Raspberries Pi[3] e os servomotores descritos na seção 3.1.5.

3.1.5 Servomotor

Para controlar a direção do carro, foi utilizado um servomotor do modelo MG995. Já para controlar a movimentação da câmera (descrita na seção 3.1.7) foram utilizados dois servomotores do modelo SG90.

As especificações técnicas mais relevantes dos dois modelos de servomotores utilizados estão relacionadas na Tabela 3.4.

Tabela 3.4 - Características dos servomotores utilizados

	MG995	**SG90**
Tensão Mínima de Operação	3,5 V	4,2 V
Tensão Máxima de Operação	8,4 V	6,0 V
Torque (5 V)	13,3 kg/cm	1,5 kg/cm

É importante ressaltar que o servo utilizado para controlar a direção possui uma boa capacidade de torque, o que se faz necessário para a aplicação desenvolvida.

Ambos os modelos utilizados podem ser vistos na Figura 3.6.

Figura 3.6 - Servomotores SG90 (esquerda) e MG995 (direita).

3.1.6 Motores

Para realizar a movimentação do carro, foram utilizados dois motores DC do modelo “Pittman 12V 5.9:1”. Suas características mais relevantes estão descritos na Tabela 3.5. O motor utilizado pode ser visto na Figura 3.7.

Tabela 3.5 - Características dos motores utilizados

Tensão nominal de Operação	12 V
Velocidade (sem carga)	1000 RPM
Potência	24 W
Torque (stall)	1.5 N.m
Corrente (sem carga)	0,33 A
Corrente (stall)	15,5 A
Massa	450 g

Figura 3.7 - Motor “Pittman 12V 5.9:1”

3.1.7 Câmera

A câmera utilizada neste projeto foi uma WebCam USB que pode ser vista na Figura 3.8. Suas principais características podem ser vistas na Tabela 3.6.

Figura 3.8 - WebCam USB

Tabela 3.6 - Características da câmera utilizada

Sensor	CMOS
Resolução máxima	5 MP
Formato de vídeo	24-bit RGB
Frame Rate	(640 x 480) 30 f/s; (1280 x 960) 10 -15 f/s
Compatibilidade	USB 2.0 / USB 1.1
Alcance de Foco	De 3 cm até infinito

3.1.8 Display LCD TFT

O display utilizado para apresentar a imagem capturada pela câmera no controle remoto é um display de LCD TFT de 3.2'', *touch screen* do tipo resistivo, com comunicação SPI e desenvolvido especificamente para ser utilizado em placas Raspberry Pi[3] modelo B e B+. Ele possui uma resolução de 320x240 pixels. Ele pode ser visto na Figura 3.9.

Figura 3.9 - Display LCD TFT

Uma versão do SO compilada para operar com este display é distribuída pelo fabricante, sendo uma distribuição Debian[8] do Linux para Raspberry Pi[3], o Raspbian[7]. Portanto o SO executado pela Raspberry Pi[3] deve ser o distribuído pelo fabricante do display a fim de possibilitar a exibição do ambiente gráfico do SO no display.

3.1.9 Adaptador Wireless

A fim de possibilitar a comunicação entre as duas Rasbperries Pi[3], foram utilizados dois adaptadores wireless BL-LW05-AR5. Este adaptador wireless possui conectividade via USB, núcleo Realtek RTL8188, é capaz de operar como "access point"[9]. Possui uma

antena de +5DB, transmite a 150 Mbps e possui um longo alcance do sinal. Durante testes, concluiu-se que seu alcance é de cerca de 200 metros. O adaptador utilizado pode ser visto na Figura 3.10.

Figura 3.10 - Adaptador Wireless USB

3.1.10 Módulo de Rádio

A fim de possibilitar a comunicação entre os dois Arduinos[2], foram utilizados dois módulos de rádio NRF24L01. Trata-se de um módulo de rádio que se comunica com o Arduino[2] via SPI, opera em uma faixa de frequência de 2,4 a 2,5 GHz com possibilidade para 125 frequências diferentes e suporta até seis canais. O módulo utilizado possui uma antena, o que aumenta seu alcance para 1 km. O módulo utilizado pode ser visto na Figura 3.11.

Figura 3.11 - Módulo de Rádio NFR24L01

3.1.11 Módulo Joystick

A fim de facilitar a usabilidade do controle remoto, foram utilizados dois módulos joystick analógicos para enviar a informação de movimento para o carro. Foi utilizado um para leitura da movimentação do carro e o outro para a leitura da movimentação da câmera. Ele possui duas saídas analógicas (uma no eixo x e outra no eixo y) e uma saída digital (eixo z). O módulo utilizado pode ser visto na Figura 3.12.

Figura 3.12 - Módulo Joystick Analógico

3.1.12 Plataforma para Câmera

Para realizar a movimentação da câmera no eixos *pitch* e *yaw*, foi utilizado uma plataforma como a mostrada na Figura 3.13. Ela possui suporte para os servomotores SG90 que foram utilizados neste projeto. Ela foi adquirida pronta, necessitando-se apenas montá-la.

Figura 3.13 - Plataforma para Câmera

3.1.13 Rodas

Para realizar a movimentação do carro, foram utilizadas quatro rodas conforme mostrado na Figura 3.14. Estas rodas possuem diâmetro externo igual a 52 mm e largura igual a 26 mm. Seus pneus são de borracha, o que aumenta o seu atrito com o solo. Com estas rodas e com o motor utilizado, espera-se que a velocidade do carro seja igual a 9,8 km/h.

Figura 3.14 - Roda Utilizada

3.1.14 Transistores

Para fazer o controle do sentido de rotação e de velocidade dos motores, foi desenvolvido um circuito ponte h que será visto na seção 3.3.1.1.1. Os transistores utilizados foram o transistor bipolar NPN 2N3904 e os transistores mosfets IRF9540N (canal P) e IRF540N (canal N). Estes transistores mosfets foram utilizados por possuírem uma corrente máxima de dreno alta o suficiente para suprir a necessidade do motor mesmo em “stall”. A corrente máxima do IRF940N é de 23 A e do IFR540N é de 33 A. A Figura 3.15 mostra os três modelos de transistor utilizados. O bipolar utilizado apresenta encapsulamento T0-92 e ambos os mosfets utilizados apresentam encapsulamento T0-220.

Figura 3.15 - Transistores 2N3904 (esquerda), IRF9540N (meio) e IRF540N (direita).

3.2 Desenvolvimento do Controle Remoto

O desenvolvimento do controle remoto será dividido em três seções: Desenvolvimento Eletrônico, Mecânico e de Software. Em cada seção é listados os materiais utilizados descritos na seção 3.1. Ao final desta seção, são mostradas as imagens do controle remoto finalizado nas Figuras 3.28 e 3.29.

3.2.1 Desenvolvimento Eletrônico

Para o desenvolvimento eletrônico foram necessários os materiais listados na Tabela 3.7. Alguns materiais na tabela não foram descritos na seção 3.1 devido a serem materiais de propósito geral e que não precisam ser descritos com mais detalhes.

Tabela 3.7 - Componentes Eletrônicos utilizados no Controle Remoto

Material	**Quantidade**
Arduino Nano	1
Raspberry Pi Modelo B	1
Cartão de Memória SD de 8 GB	1
Bateria LiPo 2 Células	1
Regulador de Tensão	2
Display de LCD TFT	1
Adaptador Wireless	1
Módulo de Rádio NFR24L01	1
Módulo Joystick	2
Potênciomêtro de 10 kΩ	2
Resitor de 10 kΩ	1
Chave Mecânica	2

Conector de Bateria LiPo	1
Cabo Micro USB	1
Placa Fenolite Cobreada 0,5 oz 15,5 x 10 cm	1
Par de cabo flexível com 2,5 mm diâmetro de seção	20 cm
Conector de parafuso para PCI	1

O controle remoto possui duas partes distintas: a parte do tratamento de movimentação do carro e a parte de recepção de imagem enviada pelo carro. Cada uma destas partes será discutida nas seções 3.2.1.1 e 3.2.1.2 respectivamente.

3.2.1.1 Tratamento de Movimentação por parte do Controle Remoto

Para fazer o tratamento da movimentação do carro por parte do controle remoto foi utilizado o Arduino Nano[12]. Ele recebe valores analógicos dos dois joysticks e de dois potenciômetros e um valor digital de uma chave. O joystick do lado esquerdo do controle remoto é responsável pelo acionamento dos motores do carro e do servomotor de direção. O joystick do lado direito é responsável pela movimentação dos servomotores da plataforma da câmera. Os potenciômetros são responsáveis por estabelecer a velocidade de rotação dos motores e a chave digital é responsável por estabelecer se o carro se movimentará como um carro convencional ou como um tanque, sendo que neste modo, a barra de direção fica parada e para fazer curva com o carro, cada motor gira para um lado, de forma que o carro gire sobre o próprio eixo. Este tipo de movimentação é útil em lugares no qual o espaço é reduzido e deseja-se fazer uma mudança rápida da direção do carro.

Assim que todos estes valores são lidos, são transmitidos ao módulo de rádio NRF24L01 que retransmite esses valores via ondas de rádio para o módulo de rádio do receptor presente no carro.

Para alimentar o Arduino[2], os potenciômetros e os módulos joysticks a tensão da bateria é regulada para 5V através do regulador de tensão.

O esquemático do circuito desenvolvido pode ser visto na Figura 3.16. Neste também está presente o regulador de tensão utilizado para regular a tensão da bateria em 5V para alimentar a Raspberry Pi[3].

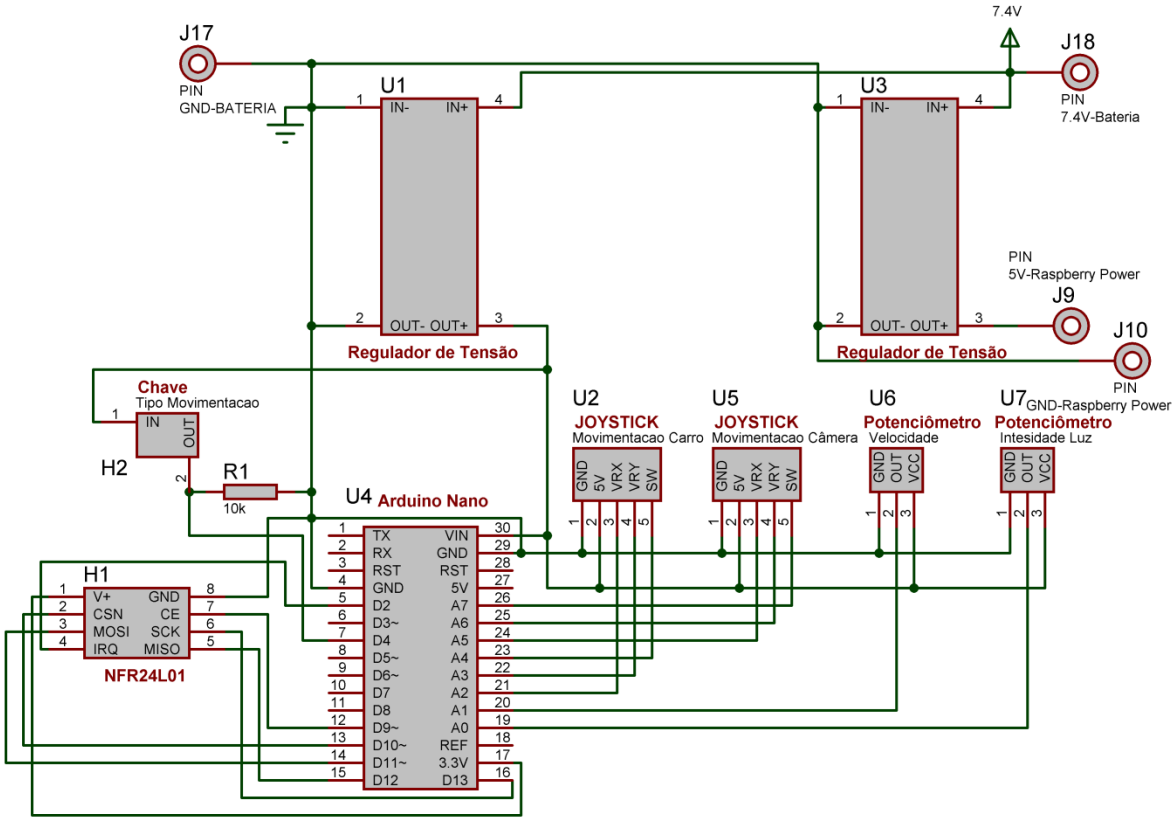

Figura 3.16 - Esquemático Controle Remoto.

Tanto o esquemático quanto o layout para PCI deste circuito e de todos desenvolvidos neste projeto foram desenvolvidos com o auxílio do software Proteus 7.6 Profissional. Para o desenvolvimento do esquemático foi utilizado o software ISIS e para o desenvolvimento do layout das PCIs o software ARES. O layout do PCI gerado a partir deste esquemático pode ser visualizado na Figura 3.17.

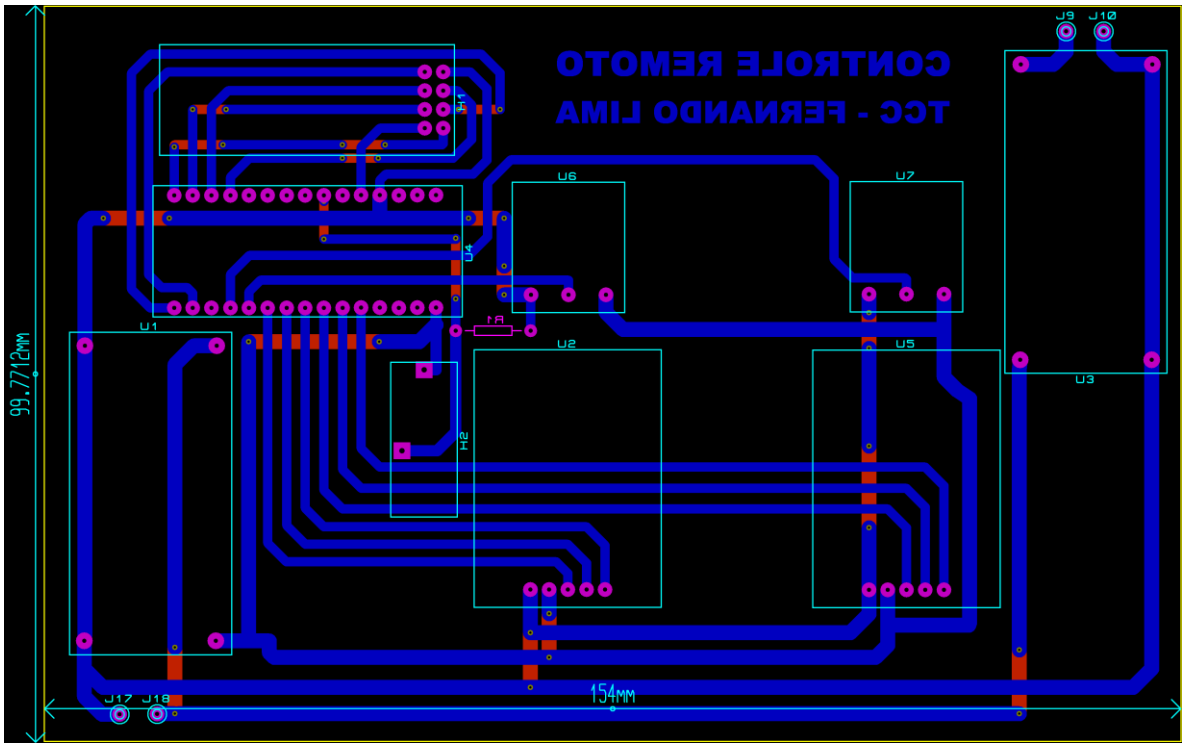

Figura 3.17 - Layout do PCI do controle Remoto.

O processo utilizado para o desenvolvimento desta PCI foi o mesmo utilizado em todas as PCIs deste projeto. O passo a passo dele é mostrado abaixo:

- Passo1: Imprimir o layer *bottom cooper* do layout da PCI em um papel fotográfico com uma impressora a laser.
- Passo2: Recortar uma placa de fenolite cobreada no tamanho do layout.
- Passo3: Transferir termicamente a tinta presente no papel fotográfico para a placa de fenolite no lado cobreado. Pode-se utilizar um ferro de passar roupas para isso ou uma prensa térmica.
- Passo4: Correr a placa com percloreto de ferro. Ao entrar em contato com o percoloreto de ferro, o cobre se dissolve, de forma que o cobre que está coberto pela tinta não se dissolve.
- Passo5: Retirar a tinta que está no cobre. Para isso, pode-se utilizar uma esponja de aço.
- Passo6: Furar a placa em todos os PADs.
- Passo7: Soldar “jumpers” no lado de cima da placa em substituição do layer *top cooper*.
- Passo8: Soldar todos os componentes.

As Figuras 3.18, 3.19, 3.20 e 3.21 mostram o passos indicados acima. Estas figuras são referentes ao PCI presente no carro que será discutido na seção 3.3.1.1.

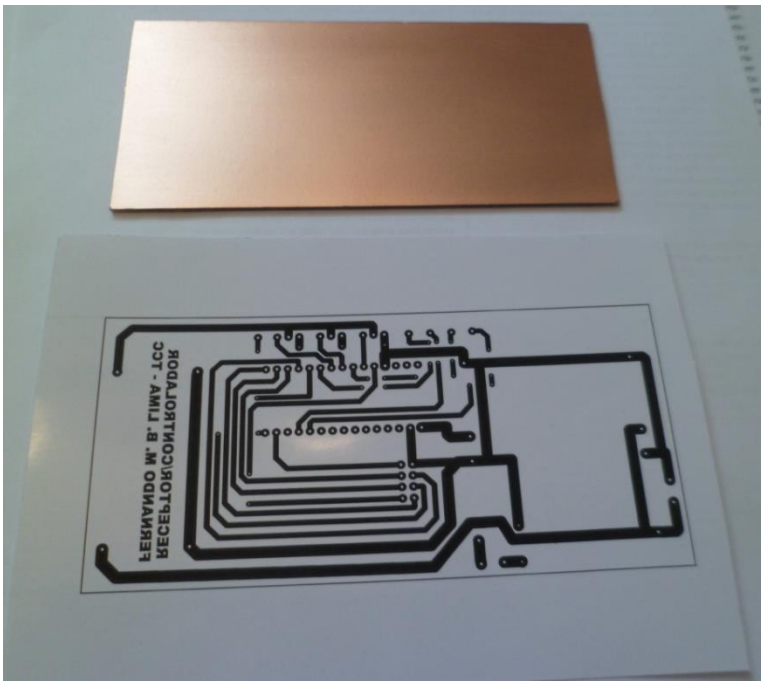

Figura 3.18 - Passos 1 e 2 do processo de confecção de uma PCI.

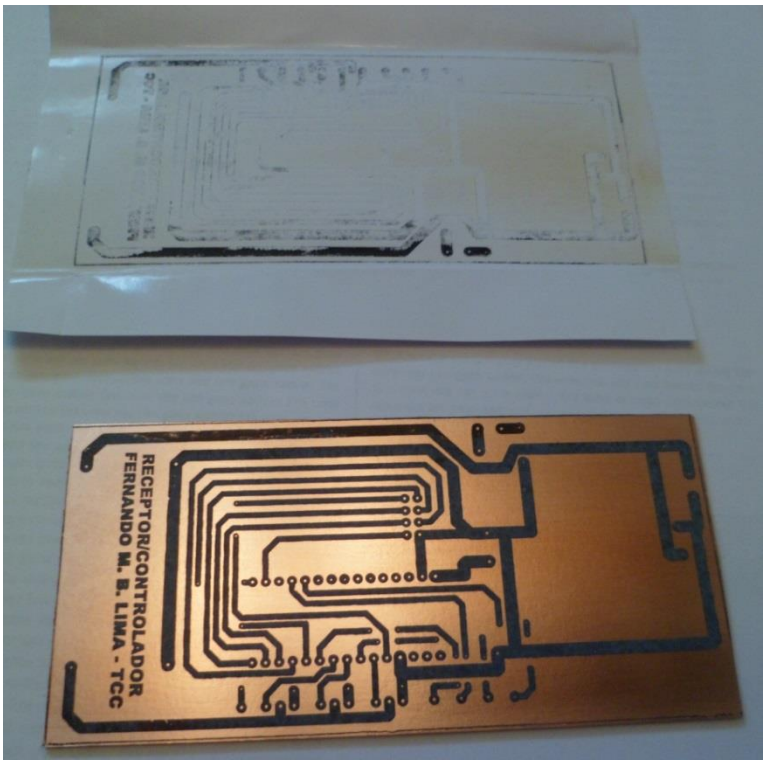

Figura 3.19 - Passo 3 do processo de confecção de uma PCI.

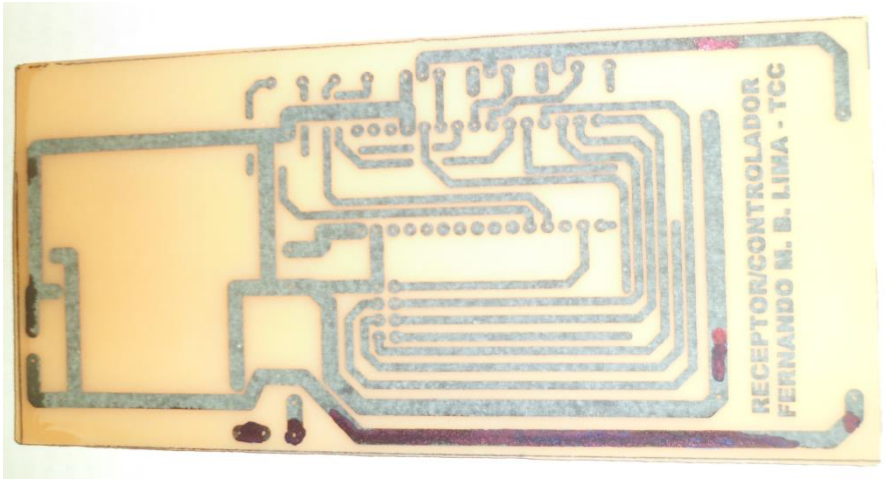

Figura 3.20 - Passos 4 do processo de confecção de uma PCI.

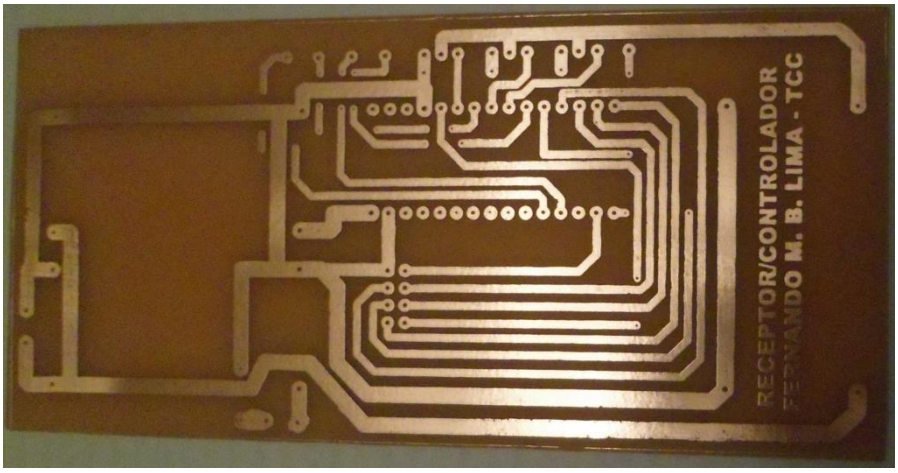

Figura 3.21 - Passos 5 do processo de confecção de uma PCI.

A PCI finalizada do controle remoto pode ser vista nas Figuras 3.22, 3.23 e 3.24.

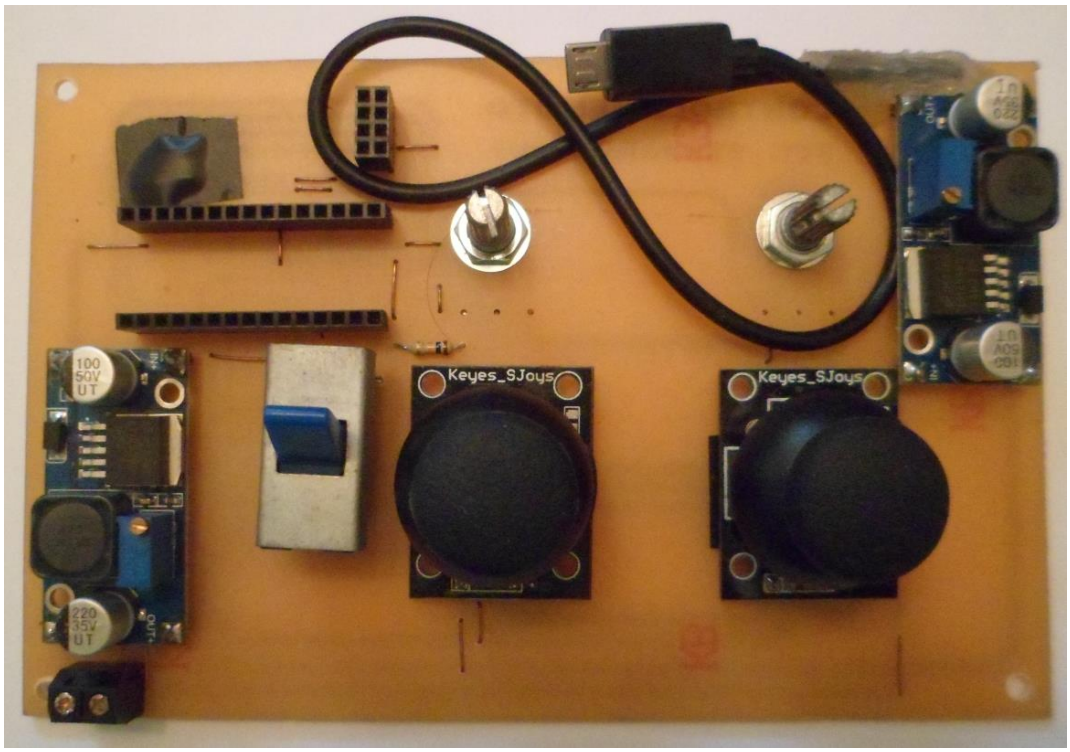

Figura 3.22 - PCI Controle Remoto – Vista de Cima.

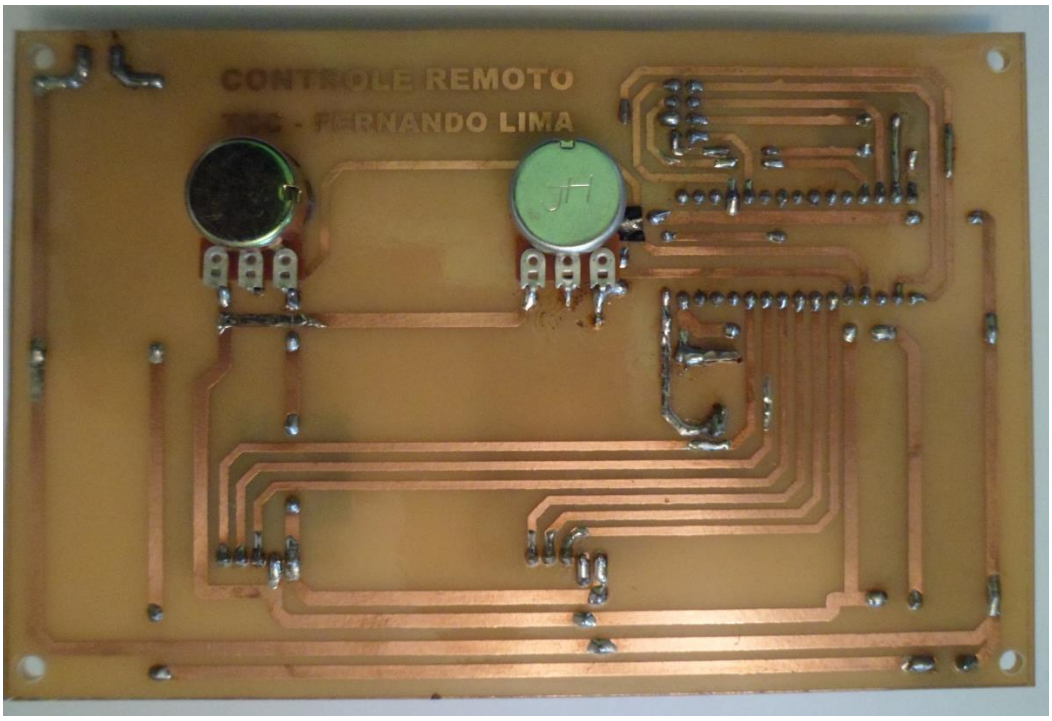

Figura 3.23 - PCI Controle Remoto – Vista de Baixo.

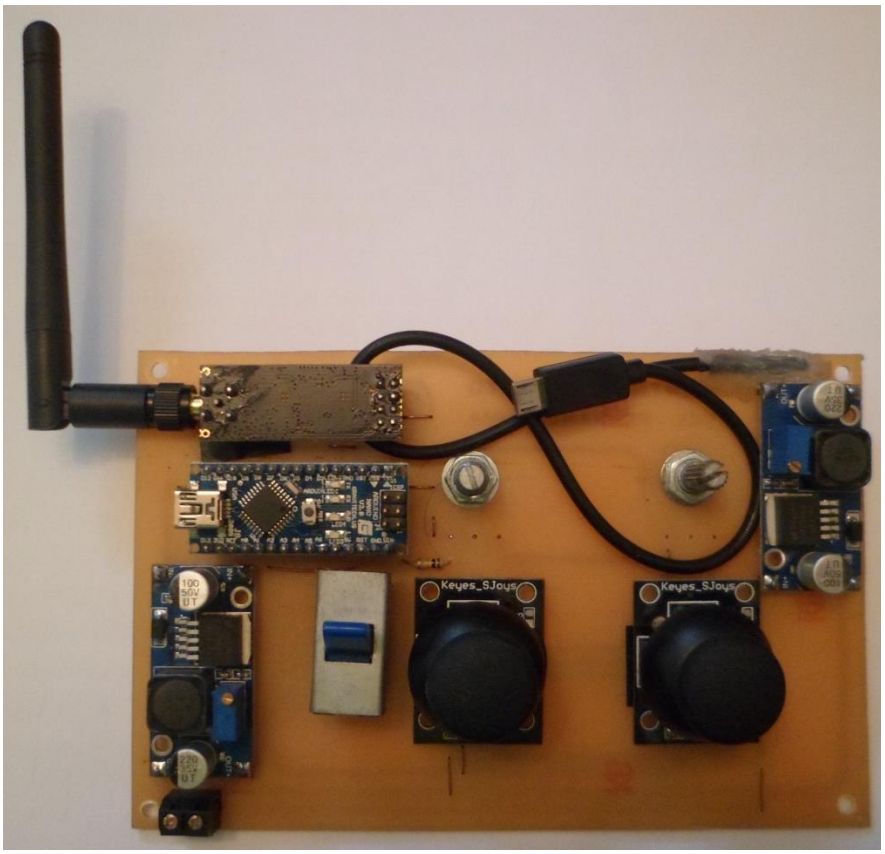

Figura 3.24 - PCI Controle Remoto com Arduino Nano e NRF24L01.

3.2.1.2 Tratamento da recepção e exibição da imagem capturada

Para fazer a recepção e a exibição da imagem capturada pela câmera presente no carro, foram utilizados uma Raspberry Pi[3], um display de LCD TFT para a exibição da imagem recebida, um Adaptador Wireless USB para fazer a conexão com a rede “WiFi” criada pelo carro, um regulador de tensão para regular a tensão da bateria para 5V e um cabo USB para conectar a saída do regulador de tensão à Raspberry Pi[3]. O display utilizado é encaixado sobre os pinos de entrada e saída da Raspberry Pi[3]. A Figura 3.28 mostra esta parte do controle remoto. Ela está posicionada na parte superior central.

Após a finalização dos dois tratamentos, ligou-se a bateria ao par de cabos passando um deles por uma chave interruptora. Uma das extremidades dos cabos está soldado no conector para a bateria que está conectado na bateria e o outro está parafusado no conector de parafuso para PCI na PCI. A Figura 3.29 mostra a parte traseira do controle que mostra a fiação desenvolvida.

3.2.2 Desenvolvimento Mecânico

Para o desenvolvimento mecânico do controle remoto foram utilizados os materiais listados na Tabela 3.8

Tabela 3.8 - Componentes Mecânicos utilizados no Controle Remoto

Material	**Quantidade**
Chapa de Plástico Polietileno 16 x 16 cm	1
Parafuso	4
Porca	12
Abraçadeira auto travante de nylon	4

Para este desenvolvimento, recortou-se uma chapa de plástico polietileno com formato retangular com as dimensões de 16 x16 cm e efetuou-se alguns furos nela para permitir a fixação da PCI, da chave interruptora, da bateria e da Raspberry Pi[3] modelo B. Esta chapa de plástico pode ser vista na Figura 3.25.

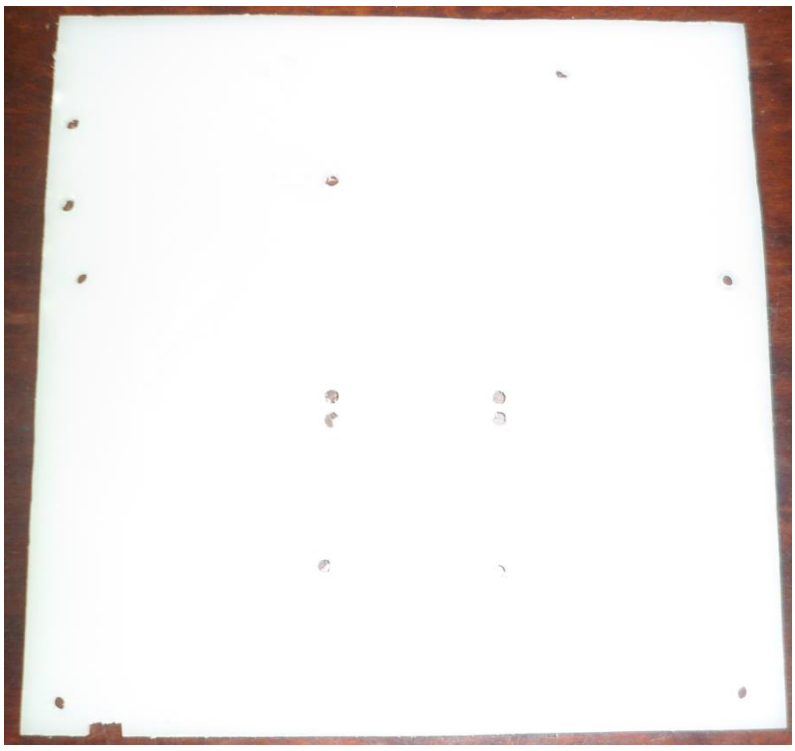

Figura 3.25 - Chapa de Plástico para Confecção do Controle Remoto.

Para a fixação da PCI, efetuaram-se quatro furos nas extremidades da PCI e quatro furos com mesmos espaçamentos na chapa de plástico. Após isso, colocou-se a PCI como os componentes virados para cima sobre a chapa e passou-se os quatro parafusos pelos buracos na direção da chapa para a PCI, colocando-se três porcas por parafuso: uma rente a chapa de plástico, uma embaixo da PCI e outra sobre a PCI. Foi necessária a colocação de três porcas por parafuso para permitir que a PCI ficasse espaçada da placa de plástico devido ao volume ocupado na parte inferior da PCI pelos potenciômetros. A passagem dos

parafusos e a fixação do PCB sobre a chapa podem ser visualizadas nas Figuras 3.26 e 3.27.

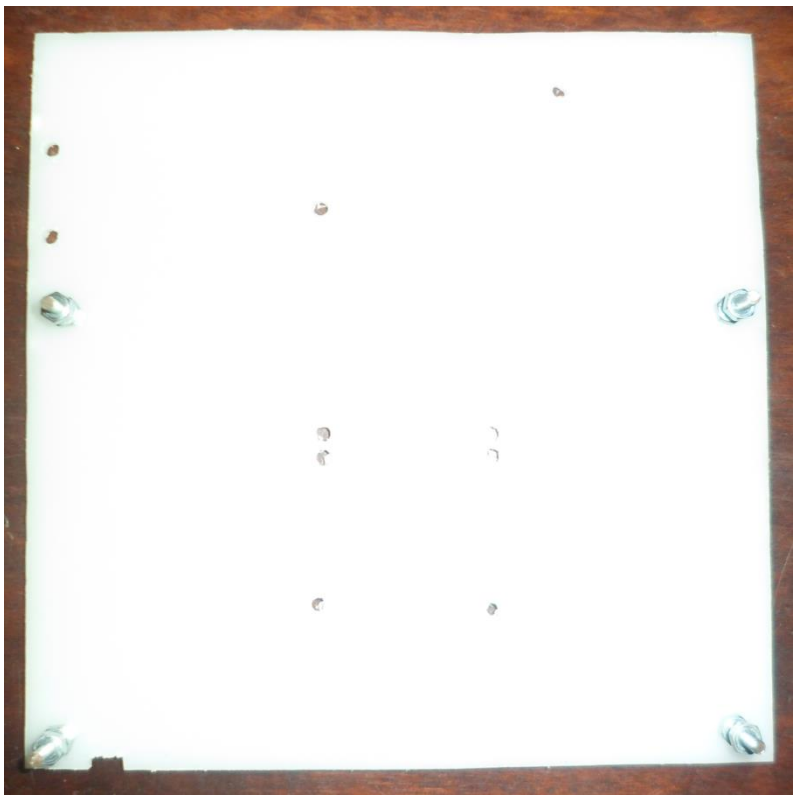

Figura 3.26 - Chapa de Plástico com os parafusos passados e com duas porcas.

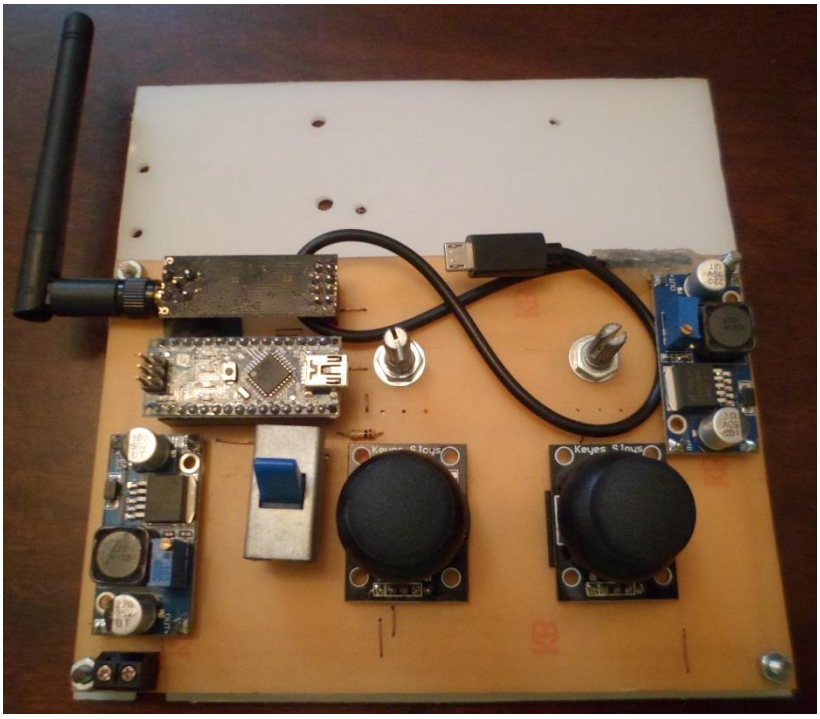

Figura 3.27 - PCI do Controle Remoto fixado sobre a chapa de plástico.

Após a fixação do PCI sobre a chapa, foi-se fixado o cabo com o conector de bateria e com a chave interruptora e a bateria sobre a parte traseira da chapa com o auxílio de abraçadeiras auto travantes de nylon.

Também foi fixado a Raspberry Pi[3] sobre a parte superior da chapa utilizando abraçadeiras auto travantes de nylon. Após todas estas fixações, restou apenas encaixar o display TFT sobre a Raspberry Pi[3], conectar o adaptador wireless UBS em uma das entradas USB da Raspberry Pi[3], encaixar o cartão de memória no socket da Raspberry Pi[3] e conectar o cabo USB na entrada micro USB da Raspberry Pi[3] a fim de alimentá-la. O resultado final do controle remoto montado pode ser visto nas Figuras 3.28 e 3.29. Na Figura 3.28 já é mostrado o controle remoto em funcionamento.

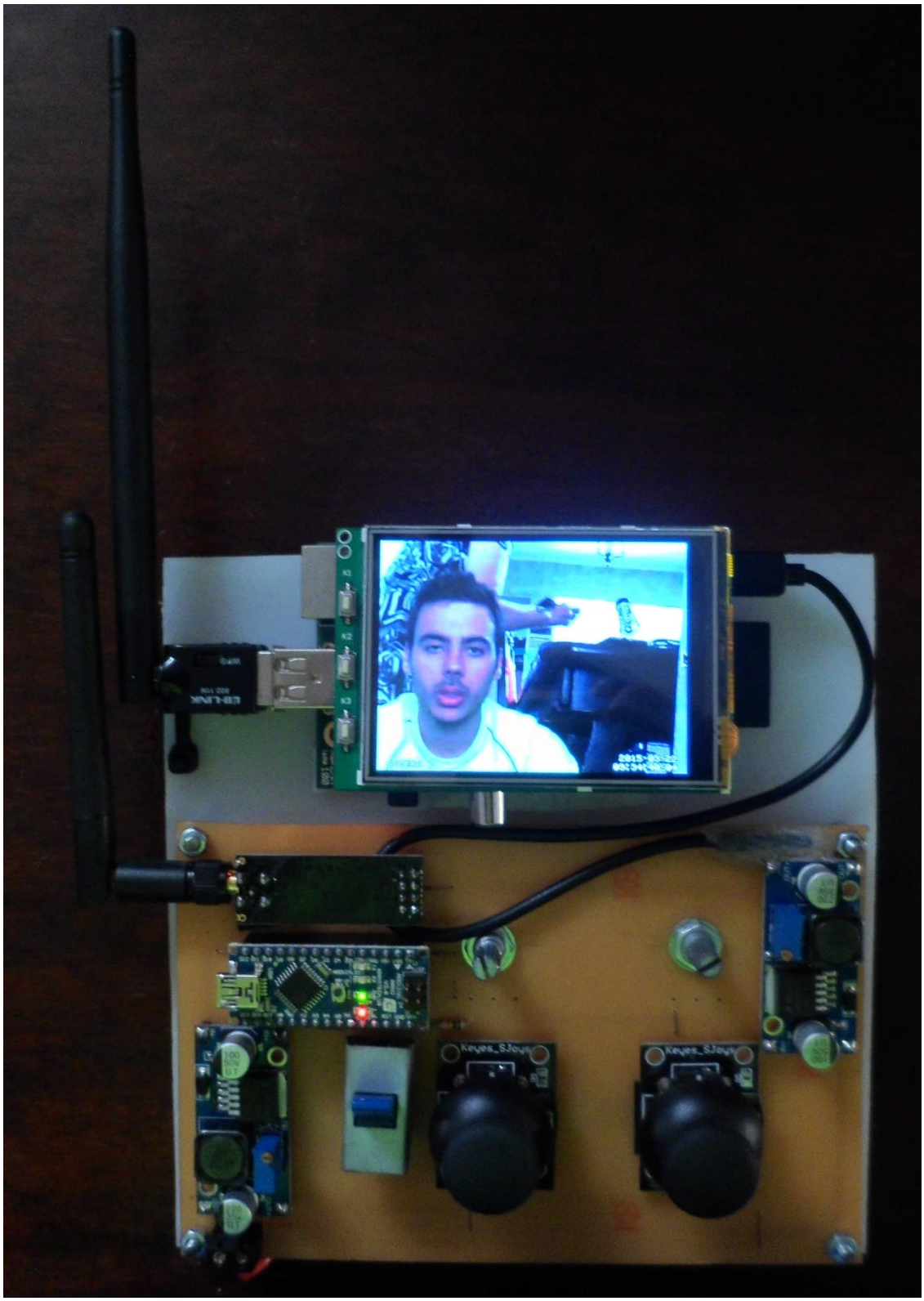

Figura 3.28 - Controle Remoto finalizado – Vista de Cima.

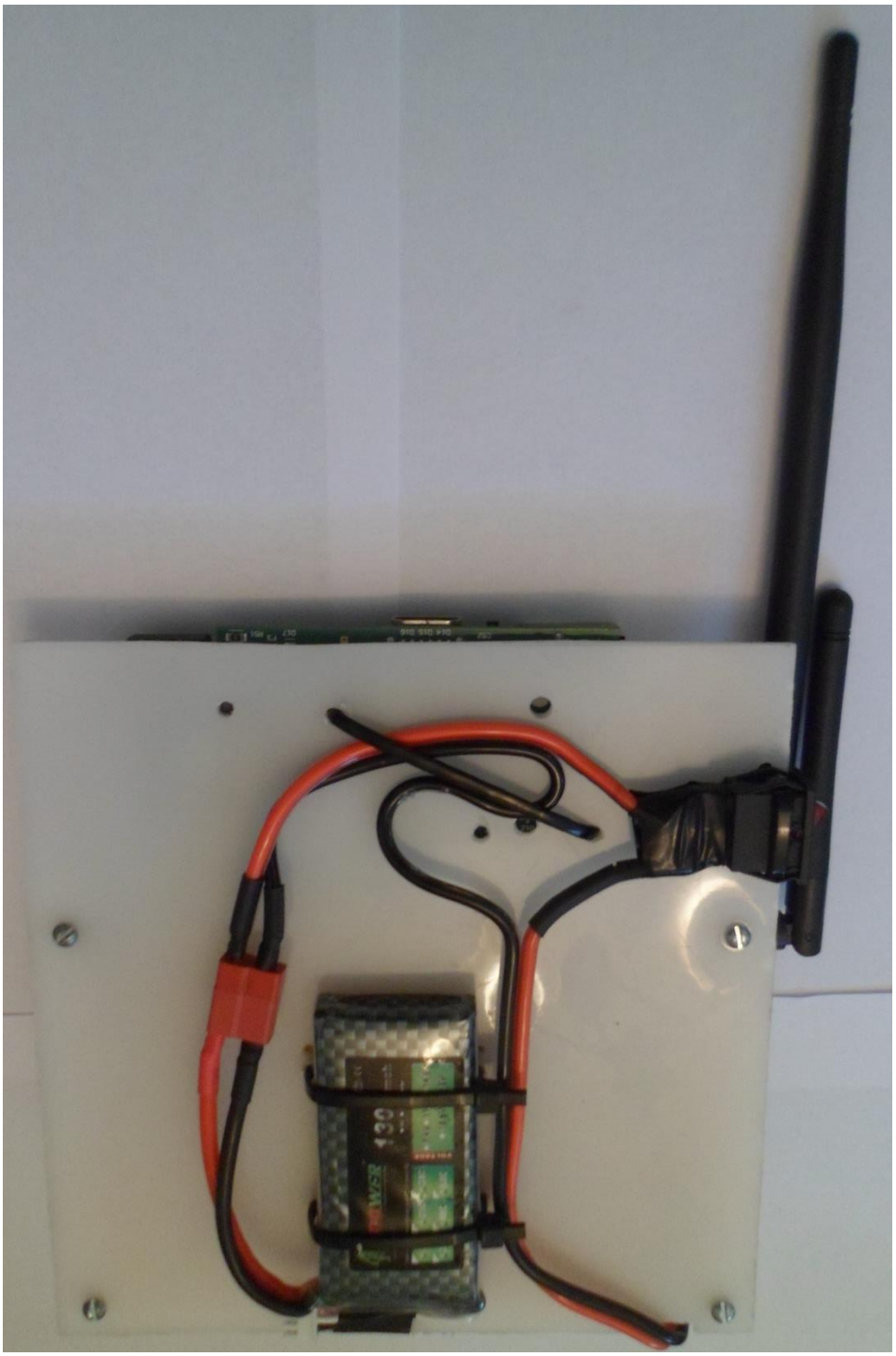

Figura 3.29 - Controle Remoto finalizado – Vista de Baixo.

3.2.3 Desenvolvimento do Sistema de Software

Para descrever o desenvolvimento do sistema do software, esta seção é dividida em duas: a seção 3.2.3.1 para descrever o sistema que é executado no Arduino e a seção 3.2.3.2 para descrever o sistema que é executado na Raspberry Pi[3].

3.2.3.1 Desenvolvimento do sistema executado no Arduino no Controle Remoto

O Arduino[2] presente no controle remoto é responsável por receber os dados analógicos dos joysticks e dos potenciômetros e o sinal digital de uma chave e enviar estes dados ao módulo de rádio através de uma interface SPI. Estes dados serão recebidos pelo módulo de rádio presente no carro e será transmitido ao Arduino[2] presente no carro que irá interpretar estes dados e atuar nos motores e servomotores.

Portanto é necessário direcionar os dados dos joysticks e dos potenciômetros para as portas de entrada analógica do Arduino[2] e direcionar o sinal digital da chave seletora de tipo de movimentação para alguma porta qualquer, seja ela digital ou analógica, visto que as entradas analógicas podem ser utilizadas como entradas digitais no Arduino[2]. O módulo de rádio deve ser conectado em um conjunto de portas que operem como SPI do Arduino[2], sendo que no modelo Nano, estas portas são as portas 11(MISO),12(MOSI) e13(SCK) e duas portas digitais qualquer (CE e CSN).

Para fazer a comunicação entre o Arduino[2] e o módulo de rádio foi utilizada a biblioteca NR24, disponível em [16]. Também foi necessário incluir a biblioteca SPI, disponível por default na IDE.

Para o desenvolvimento do código fonte, compilação e upload do arquivo hexadecimal gerado a partir do código fonte para a programação do microcontrolador foi utilizada a IDE Arduino 1.6.

O código fonte compilado pode ser visualizado no Código A.1, presente no Anexo A.

Após o desenvolvimento do código fonte, basta dar o upload do arquivo hexadecimal gerado pelo computador no Arduino[2] conectando-o ao computador através de um cabo USB.

3.2.3.2 Desenvolvimento do sistema executado na Raspberry Pi no Controle Remoto

A imagem capturada pela câmera no carro é disponibilizada em uma porta IP. A seção 3.3.3.2 descreve o funcionamento desta captura. O sistema contido na Raspberry Pi[3] do controle remoto é capaz de acessar a rede WiFi criada pelo carro utilizando um adaptador wireless USB e exibir no display de LCD a imagem que está disponível na porta

IP através de um browser. O desenvolvimento completo do sistema é mostrado nas seções de 3.2.3.2.1 à 3.2.3.2.5.

3.2.3.2.1 Instalação do Sistema Operacional

O sistema operacional executado na Raspberry Pi[3] do controle remoto é uma distribuição Linux Debian[8] compilada para a Raspberry Pi[3], o Raspbian[7]. A versão utilizada é uma versão especial compilada para operar em conjunto com o display de LCD TFT e é disponibilizada pelo fabricante do display em um DVD. A imagem deste SO está em um arquivo chamado "TFT320TP-Raspberry-140921-V3.0.img". Como a Raspberry Pi[3] não possui memória não volátil, tanto o sistema operacional quanto os dados devem ser salvos em uma memória externa. No caso deste trabalho foi utilizado apenas um cartão de memória SD de 8GB de capacidade classe 2.

Para o desenvolvimento de todo o sistema foi utilizado um notebook DELL VOSTRO 3550 com o sistema operacional Ubuntu 14.04 instalado. Portanto, todos os comandos listados nos passos a seguir são comandos nativos do SO Linux.

Para a gravação do sistema operacional no cartão de memória basta seguir os passos descritos no Apêndice B.1. Após seguir esses passos, o SO já está instalado e pronto para ser utilizado.

3.2.3.2.2 Instalação do Chromium

A imagem capturada pelo webcam será disponibilizada em uma rede WiFi através de uma porta IP. Para visualizar a imagem no controle remoto, deve-se instalar um navegador capaz de reproduzir um streaming de imagem. Por este motivo, o navegador escolhido foi o "chromium". Para instalá-lo basta executar o seguinte comando:

```
sudo apt-get install chromium
```

Para isso, é necessário que a Raspberry Pi[3] esteja conectada em uma rede cabeada com acesso à internet.

3.2.3.2.3 Configuração da Rede WiFi

A Raspberry Pi[3] presente no carro cria uma rede WiFi (seção 3.3.3.2.4). Para fazer acesso a essa rede, deve-se conectar o adaptador wireless USB na Raspberry Pi[3] e alterar dois arquivos. Os detalhes sobre a alterações destes arquivos se encontram no Anexo B.2.

O primeiro arquivo foi modificado para conseguir-se acessar a rede WiFi, já o segundo foi modificado para obter-se um endereço de IP estático. Manter o IP estático será útil para criar o sistema de arquivos visto na próxima seção.

3.2.3.2.4 Sistema de Arquivos

O arquivo contendo o vídeo comprimido capturado pela câmera presente no carro é salvo tanto no carro quanto no controle remoto. Para isso, foi necessária a criação de um sistema de arquivos para a criação de uma pasta compartilhada. Esta pasta está localizada na Raspberry Pi[3] do controle remoto e é acessado pela Raspberry Pi[3] localizada no carro. Portanto, a Raspberry Pi[3] do controle remoto será o servidor e a Raspberry Pi[3] do carro será o cliente.

O sistema de arquivos escolhido foi o sistema de arquivos NFS (Network File System). Este tipo de sistema foi desenvolvido para permitir que se possam montar partições ou diretórios remotos como se fosse um disco local. Ele permite especificar diferentes permissões de acesso a cada cliente de acesso.

Para configurar a Raspberry Pi[3] do controle remoto como servidor, deve-se efetuar os passos contidos no Apêndice B.3, também descritos em [17].

3.2.3.2.5 Inicialização do Chromium em Full Screen

É desejado que, ao iniciar o sistema, a tela mostre a imagem capturada pela câmera em tempo real. Para isso, ela deve acessar a página web criada pelo servidor presente na Raspberry Pi[3] do carro. Para isso, deve-se acessar o endereço da página do servidor em um navegador, no caso, o “chromium”. Para que o “chromium” seja inicializado no momento em que o sistema é carregado, deve-se executar comando sudo nano /etc/xdg/lxsession/LXE/autostart e adicionar a seguinte linha ao arquivo:

```
@chromium --kiosk 192.168.42.1:2015
```

3.3 Desenvolvimento do Carro

O desenvolvimento do carro, assim como o desenvolvimento do controle remoto, será dividido em três seções: Desenvolvimento Eletrônico, Mecânico e de Software. Em cada seção são listados os materiais utilizados descritos na seção 3.1. Ao final desta seção, é mostrada a imagem do carro finalizado nas Figura 3.50, 3.51 e 3.52.

3.3.1 Desenvolvimento Eletrônico

Para o desenvolvimento eletrônico foram necessários os materiais listados na Tabela 3.9. Alguns materiais na tabela não foram descritos na seção 3.1 devido a serem materiais de propósito gerais e que não precisam ser descritos com mais detalhes.

Tabela 3.9 - Componentes Eletrônicos utilizados no Carro

Material	**Quantidade**
Arduino Nano	1
Raspberry Pi Modelo B+	1
Cartão de Memória Micro SD de 8GB	1
Bateria LiPo 3 Células	1
Regulador de Tensão	3
Adaptador Wireless USB	1
Módulo de Rádio NFR24L01	1
Chave Mecânica	1
Transistor MOSFET Canal N IRF540	4
Transistor MOSFET Canal P IRF9540	4
Transistor BJT NPN 2N3904	4
Diodo 3A	8
Resistor de 100 Ω	4
Resistor de 10 kΩ	4
Motor “Pittman 12V 5.9:1”	2
Servomotor MG995	1
Servomotor SG90	2
Conector de Bateria LiPo	1
Cabo Micro USB	1
Placa Fenolite Cobreada 0,5 oz 5,5 x 10 cm	1
Placa Fenolite Cobreada 0,5 oz 13,8 x 6,3 cm	1
Par de cabo flexível com 2,5 mm diâmetro de seção	25 cm
Conector de parafuso para PCI	9
WebCam USB	1

O carro possui duas partes distintas: a parte do tratamento de movimentação do carro e a parte do tratamento da recepção de imagem. Cada uma destas partes será discutida nas seções 3.3.1.1 e 3.3.1.2 respectivamente.

3.3.1.1 Tratamento de Movimentação do carro

Para fazer o tratamento da movimentação do carro foi utilizado o Arduino Nano[12]. Ele recebe os valores transmitidos pelo controle remoto através do módulo de rádio NRF24L01. Estes dados são então interpretados e os motores e servomotores são acionados da forma desejada.

O circuito projetado para fazer a integração entre o Arduino Nano[12], o módulo NRF24L01 e a placa controladora do motor (seção 3.3.1.1.1) é mostrado na Figura 3.30. Este circuito possui três reguladores de tensão que regulam a tensão da bateria de cerca de 12,6 V para 5 V, sendo que um é utilizado para alimentar o Arduino[2], outro para alimentar os servomotores e um para alimentar a Raspberry Pi[3]. O layout da PCI deste circuito pode ser visto na Figura 3.31. O PCI real implementado pode ser visto nas Figuras 3.32 e 3.33.

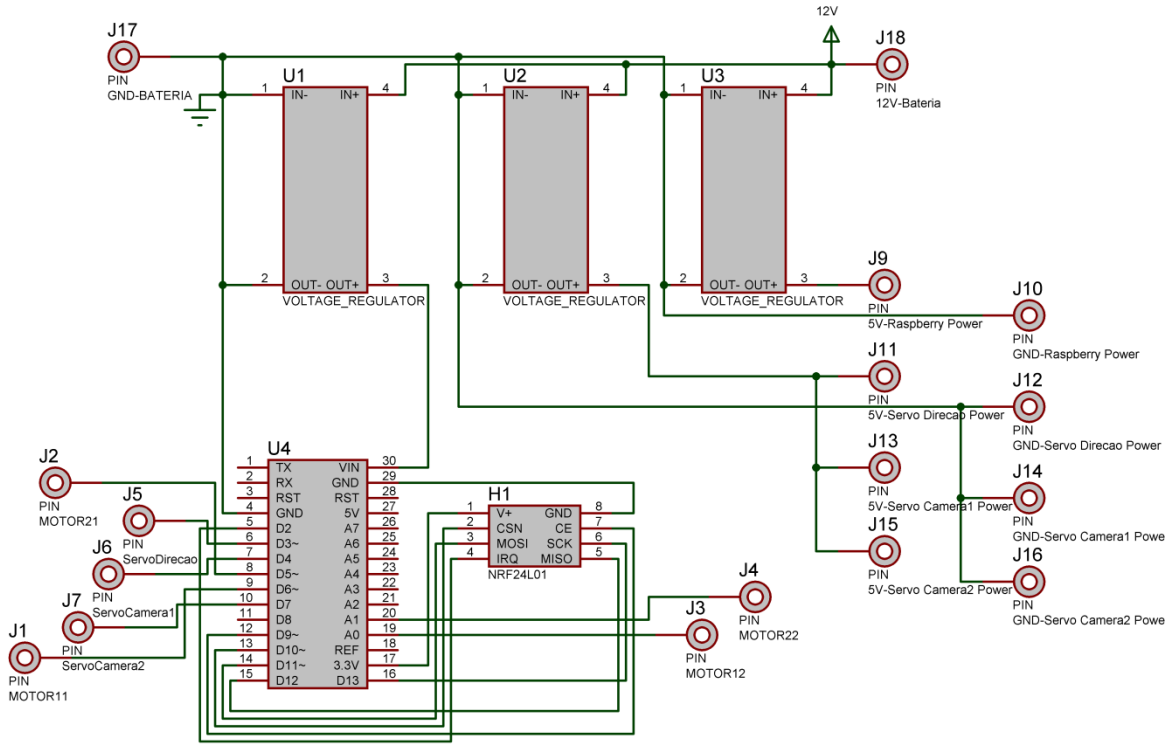

Figura 3.30 - Esquemático do Receptor

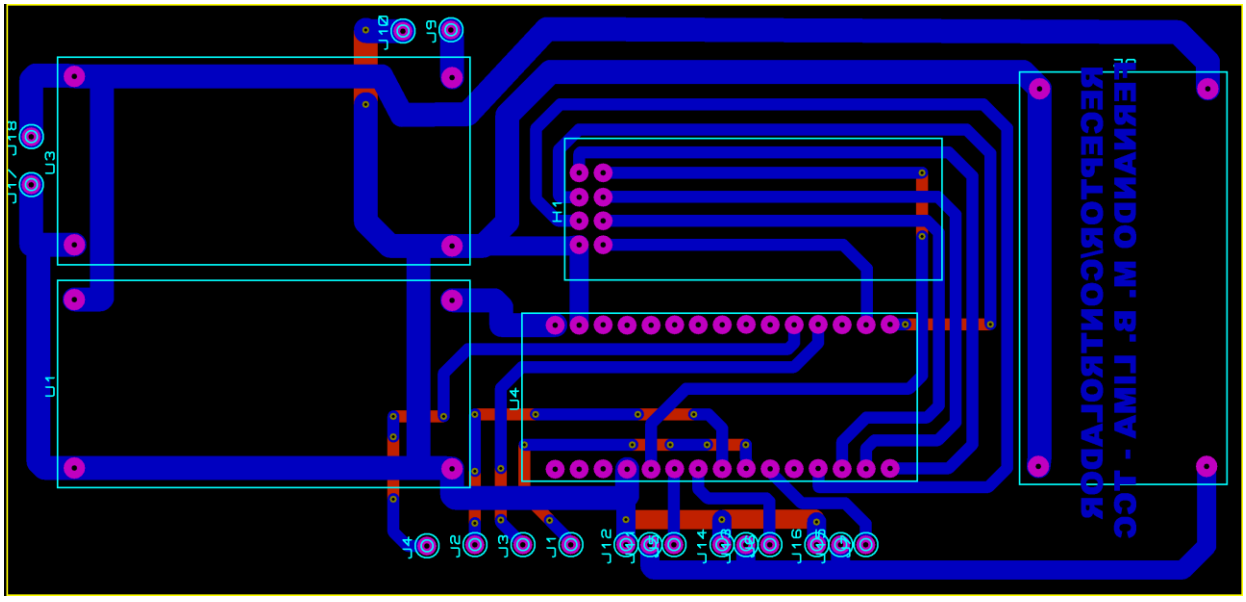

Figura 3.31 – Layout do Receptor

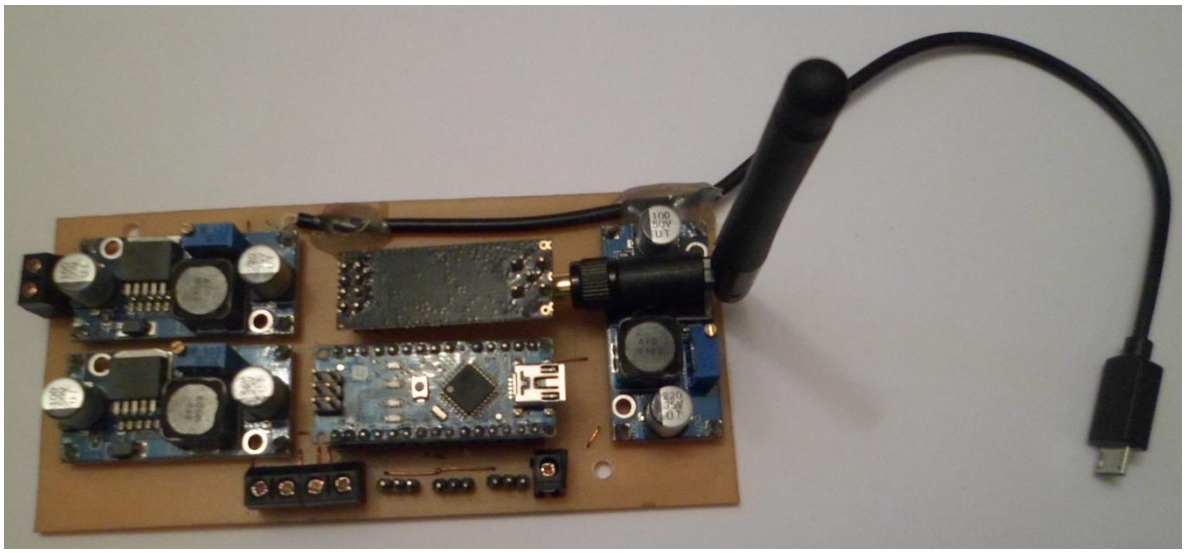

Figura 3.32 – PCI do Receptor – Vista de cima

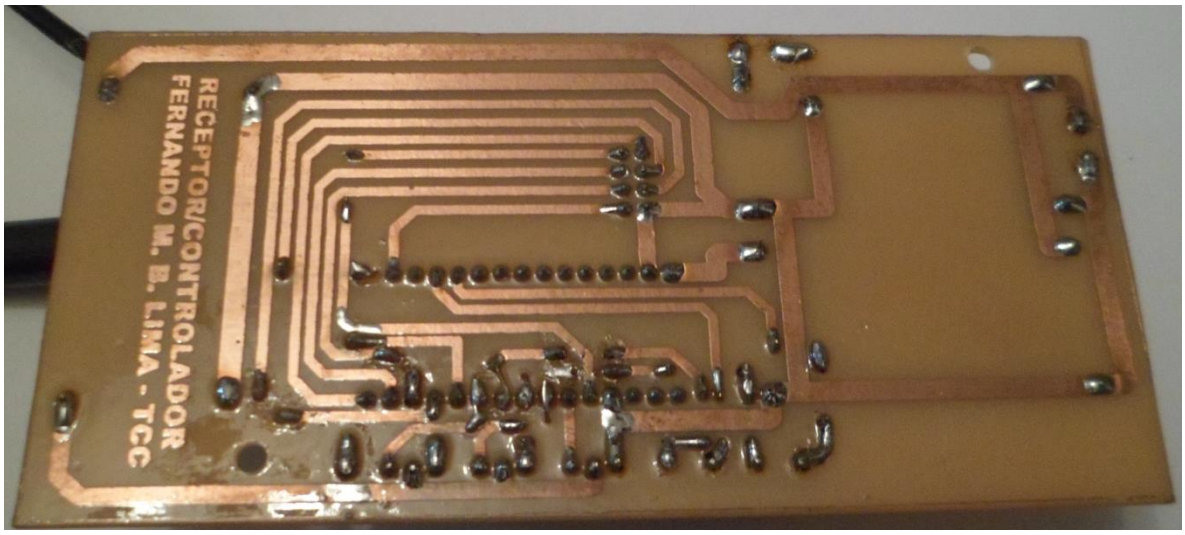

Figura 3.33 – PCI do Receptor – Vista de baixo

3.3.1.1.1 Controle de Direção e Velocidade dos motores

Para fazer o controle de velocidade e direção dos motores, foi desenvolvida a ponte H mostrada na Figura 3.34. Ela é constituída de dois transistores mosfets canal N, dois transistores mosfets canal P, dois transistores bipolar NPN, quatro resistores e quatro diodos. Duas saídas analógicas do Arduino[2] são ligadas nas entradas “IN1” e “IN2”. Os transistores bipolar tem como função poder polarizar os *gates* dos transistores mosfets com 12 V ou 0 V, visto que se ligássemos a saída do Arduino[2] direto no *gate* dos transistores mosfets apenas poderíamos polarizar com 5 V ou 0 V. Quando os *gates* dos transistores mosfets são polarizados com 12 V, o transistor mosfet canal N entra em condução e o canal P entra em corte. Quando os *gates* dos transistores são polarizados com 0 V, o transistor

mosfet canal P entra em condução e o canal N entra em corte. Desta forma, podemos controlar a polaridade do motor que está ligado nas saídas M1 e M2.

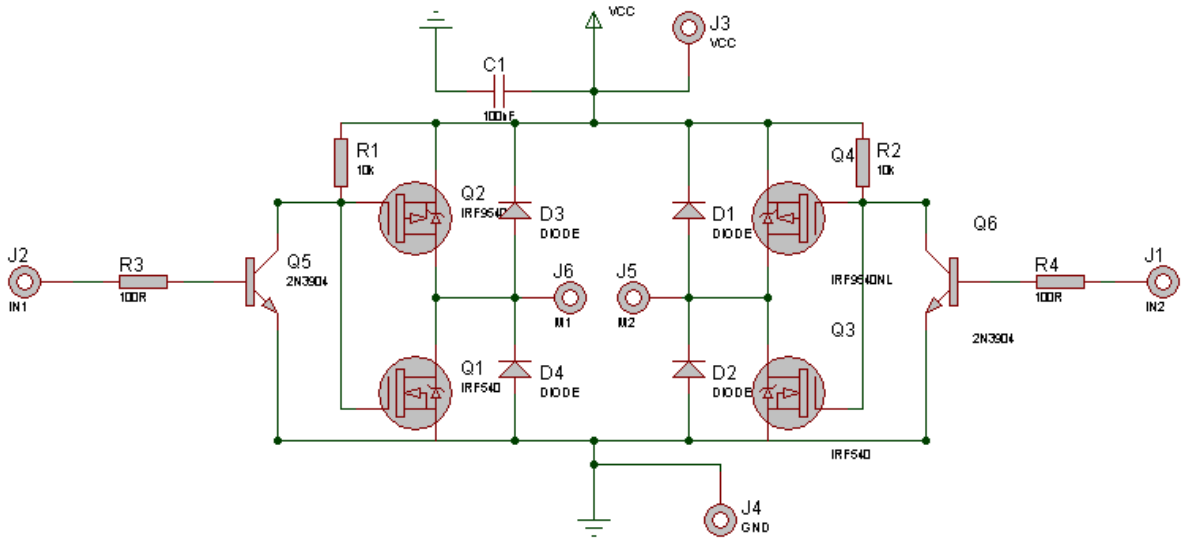

Figura 3.34 - Esquemático da Ponte H.

O funcionamento da ponte h se dará da seguinte forma:

- Caso 1: IN1 = 5 V e IN2 =0 V, os *gates* dos transistores Q1 e Q2 estarão em um nível de tensão igual a 0 V e os *gates* dos transistores Q3 e Q4 estarão em um nível de tensão igual a 12 V. Desta forma, os transistores Q2 e Q3 estarão em condução e os transistores Q1 e Q4 estarão em corte. Desta forma a tensão entre os terminais M1 e M2 valerá 12 V.
- Caso 2: IN1 = 0 V e IN2 =5 V, os *gates* dos transistores Q1 e Q2 estarão em um nível de tensão igual a 12 V e o *gate* dos transistores Q3 e Q4 estarão em um nível de tensão igual a 0 V. Desta forma, os transistores Q2 e Q3 estarão em corte e os transistores Q1 e Q4 estarão em condução. Desta forma a tensão entre os terminais M1 e M2 valerá -12 V.
- Caso 3: IN1 = 0 V e IN2 =0 V, os *gates* dos transistores Q1, Q2, Q3 e Q4 estarão em um nível de tensão igual a 12 V. Desta forma, os transistores Q1 e Q3 estarão em condução e os transistores Q2 e Q4 estarão em corte. Desta forma a tensão entre os terminais M1 e M2 valerá 0 V.
- Caso 4: IN1 = 5 V e IN2 =5 V, os *gates* dos transistores Q1, Q2, Q3 e Q4 estarão em um nível de tensão igual a 0 V. Desta forma, os transistores Q1 e Q3 estarão em corte e os transistores Q2 e Q4 estarão em condução. Desta forma a tensão entre os terminais M1 e M2 valerá 0 V.

Sendo assim, supondo que com os terminais M1 e M2 do motor ligados a +12V e a 0V respectivamente o motor rotacione no sentido horário, se tivermos a intenção de rotacionar o motor no sentido horário, basta aplicarmos as entradas IN1 = 5 V e IN2 = 0V. Caso queiramos rotacionar o motor no sentido anti-horário, basta aplicarmos as entradas

IN1 = 0V e IN2 =5 V. Se quisermos que o motor se mantenha parado, basta aplicarmos IN1 = 0 V e IN2 = 0 V ou IN1 = 5 V e IN2 = 5 V.

Os 4 diodos presentes no circuito tem com função descarregar a carga que pode ficar armazenada no motor e devolvê-la para a bateria.

Para realizarmos o controle de velocidade do motor, basta ao invés de aplicarmos uma tensão constante de 5 V em uma das entradas, aplicarmos um sinal modulado em largura (PWM), de forma que os transistores fiquem chaveando em uma frequência alta, o que fará com que a tensão média no motor seja menor do que 12V e será proporcional à porcentagem de tempo em que o sinal fica em 5 V.

O layout para PCI desta ponte H pode ser visto na Figura 3.35.

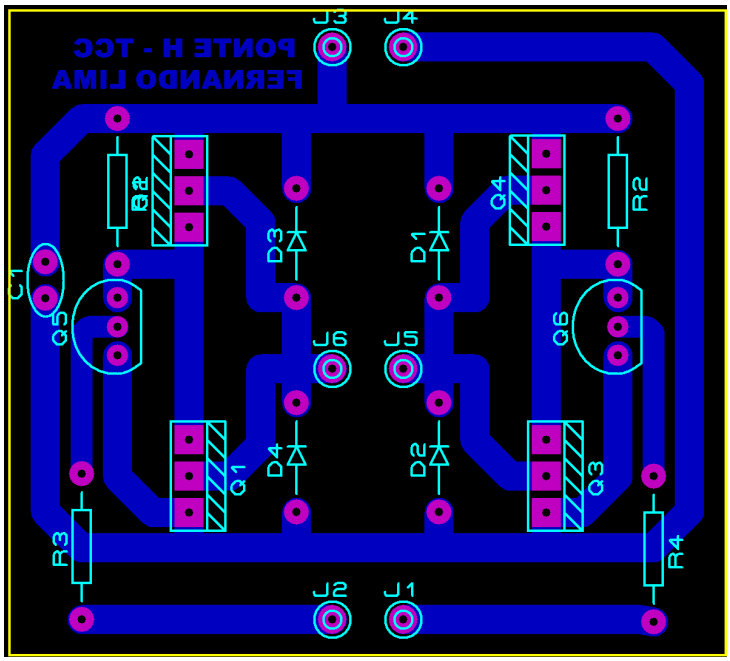

Figura 3.35 - Layout da Ponte H.

A PCI real implementada pode ser vista nas Figuras 3.36 e 3.37. Nela foram confeccionadas duas pontes H lada a lado em uma mesma placa de fenolite, sendo que será usada uma ponte H por motor.

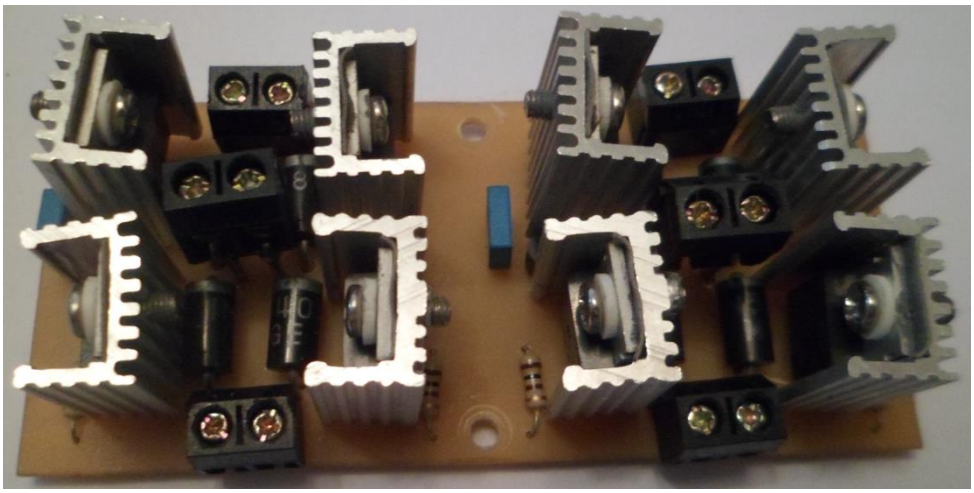

Figura 3.36 – PCI da Ponte H – Vista de cima

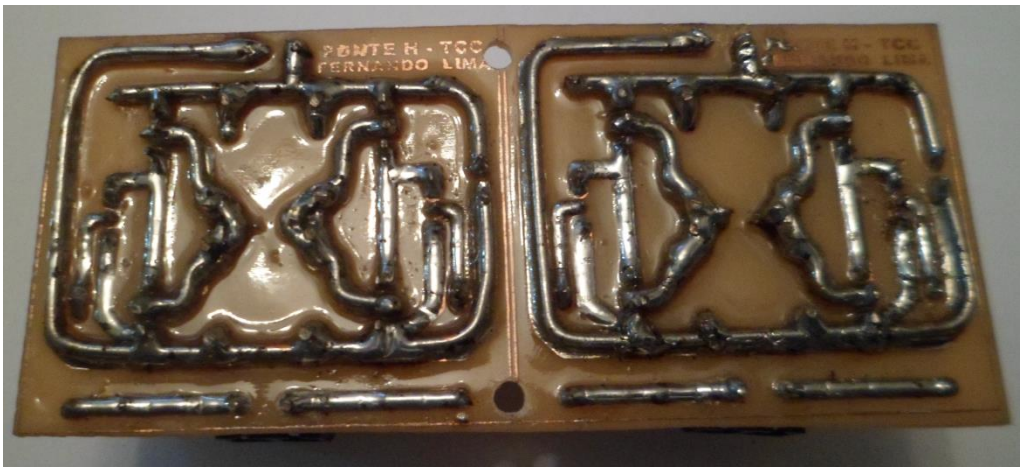

Figura 3.37 – PCI da Ponte H – Vista de baixo

Após concluída a placa receptora e a placa controladora do motor, basta apenas fazer a ligação das saídas da placa receptora aos servomotores e à placa controladora dos motores e fazer a ligação da bateria aos terminais de alimentação da placa receptora e da placa controladora do motor passando antes por uma chave. Esta ligação pode ser vista na Figura 3.46 e 3.48.

3.3.1.2 Tratamento da captura e transmissão da imagem

Para fazer captura e transmissão da imagem no carro foram utilizados uma Raspberry Pi[3] modelo B+, uma câmera Webam USB para fazer a captura das imagens, um Adaptador Wireless USB para criar uma rede “WiFi”, um regulador de tensão para regular a tensão da bateria para 5V e um cabo USB para conectar a saída do regulador de tensão à Raspberry Pi[3] a fim de alimentá-la.

3.3.2 Desenvolvimento Mecânico

Para o desenvolvimento mecânico do carro foram utilizados os materiais listados na Tabela 3.10.

Tabela 3.10 - Componentes Mecânicos utilizados no Carro.

Material	**Quantidade**
Chapa de Plástico PVC 32 x 25 cm	1
Chapa de Plástico PVC 8 x 25 cm	1
Chapa MDF 7mm 12 x 5cm	1
Parafuso para porca	17
Parafuso sem porca	24
Porca	17
Abraçadeira auto travante de nylon	5
Motor “Pittman 12V 5.9:1”	2
Servomotor MG995	1
Servomotor SG90	2
Sarrafo retangular madeira 2,0 x 2,0 cm	4 x 5,0 cm; 4 x 3,0 cm; 1 x 19 cm
Tábua retangular de madeira 5 x 5 x 2 cm	1
Plataforma para Câmera	1
Chapa galvanizada 4,5 x 12 cm	2
Recorte retangular câmara de ar 4,5x12 cm	2
Arame de Aço 6,5 cm	2
Rodas 52 mm de diâmetro	4
Caixa Acrílica para Raspberry Pi	1

Para este desenvolvimento, recortou-se uma chapa de plástico PVC com formato apropriado para servir como parte do chassi com as dimensões de 32 x 25 cm e efetuou-se alguns furos e recortes nela para permitir a fixação das PCI, da chave interruptora, da bateria, dos motores, dos servomotores, da plataforma da câmera, da Raspberry Pi[3] modelo B+, entre outros. Esta chapa de plástico que será utilizada como parte do chassi pode ser vista na Figura 3.38.

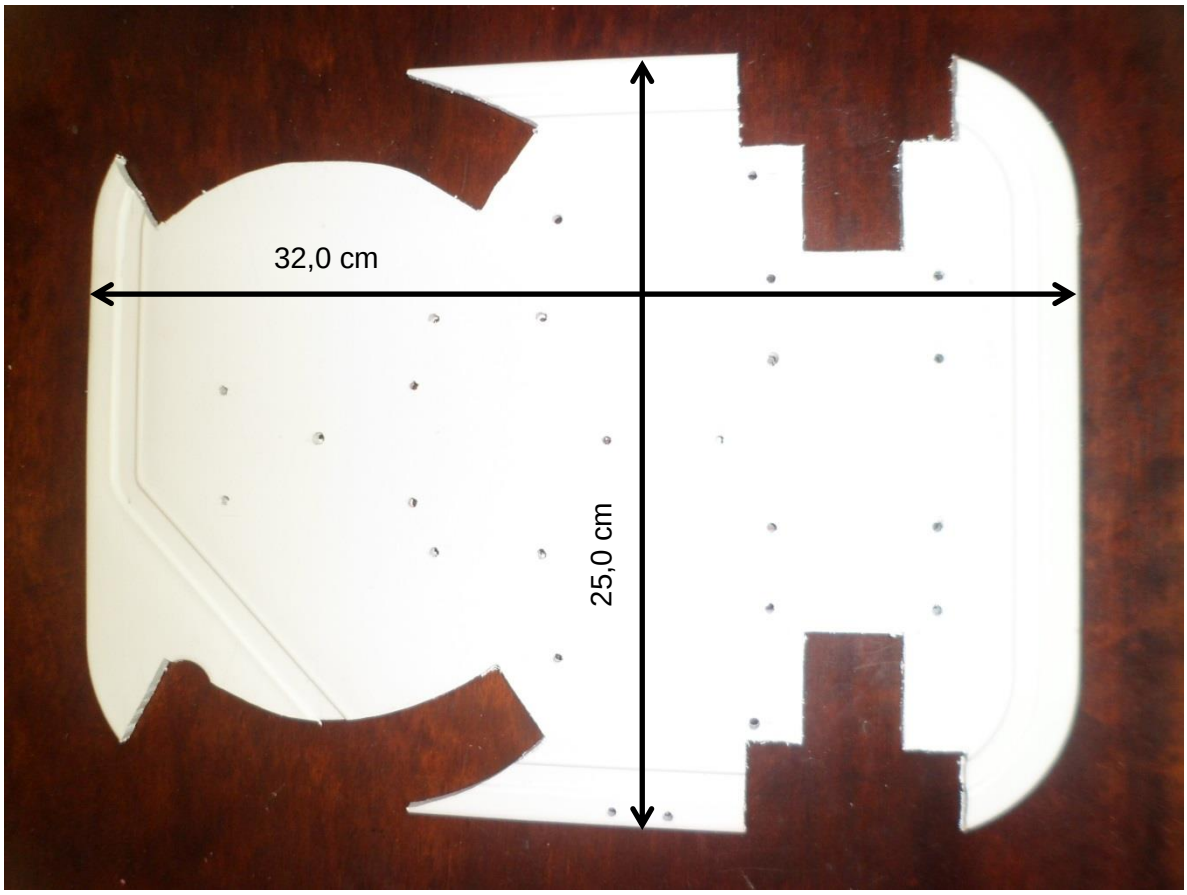

Figura 3.38 - Chapa de Plástico que servirá como parte do chassi do carro.

Após realizados este recorte e furos, foi realizado a fixação das rodas traseiras nos motores. Para isso foi utilizado um arame de aço passando pela roda e pelo motor na direção perpendicular ao motor. Isto foi possível devido ao fato de o eixo do motor já possuir um furo e após realizar dois furos em cada roda. A forma com a qual as rodas foram fixadas nos motores pode ser vista na Figura 3.39.

Para realizar a fixação de cada motor na chapa de plástico, foi necessária uma chapa de ferro galvanizada, um recorte de borracha de câmara de ar, 4 quatro parafusos e 4 porcas. A chapa galvanizada e o recorte de borracha podem ser vistos na Figura 3.40 e os motores já fixados na chapa de plástico podem ser vistos na Figura 3.41.

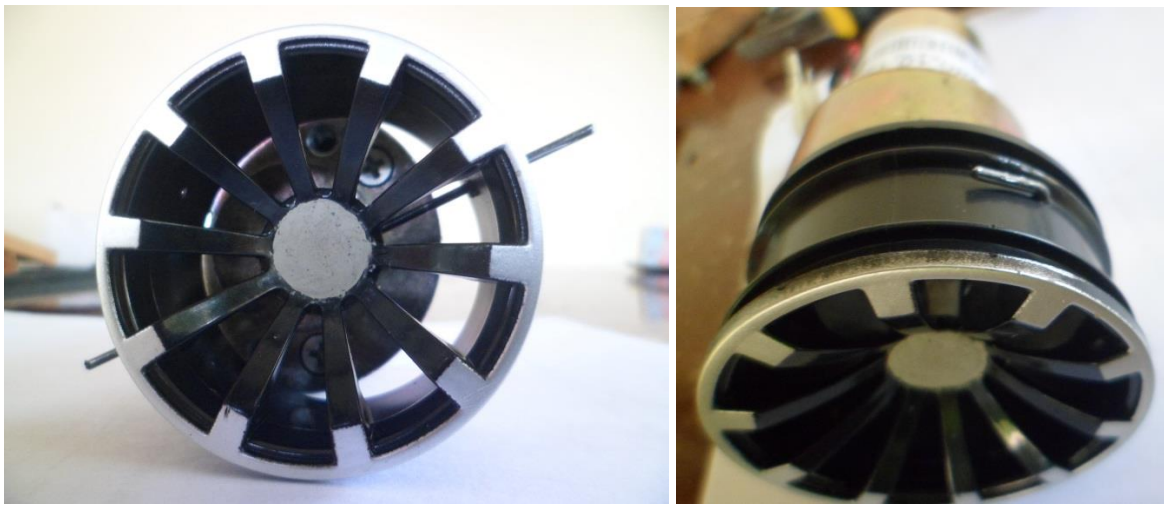

Figura 3.39 - Fixação das rodas nos motores.

Figura 3.40 - Chapa Galvanizada e recorte de borracha para fixação do motor.

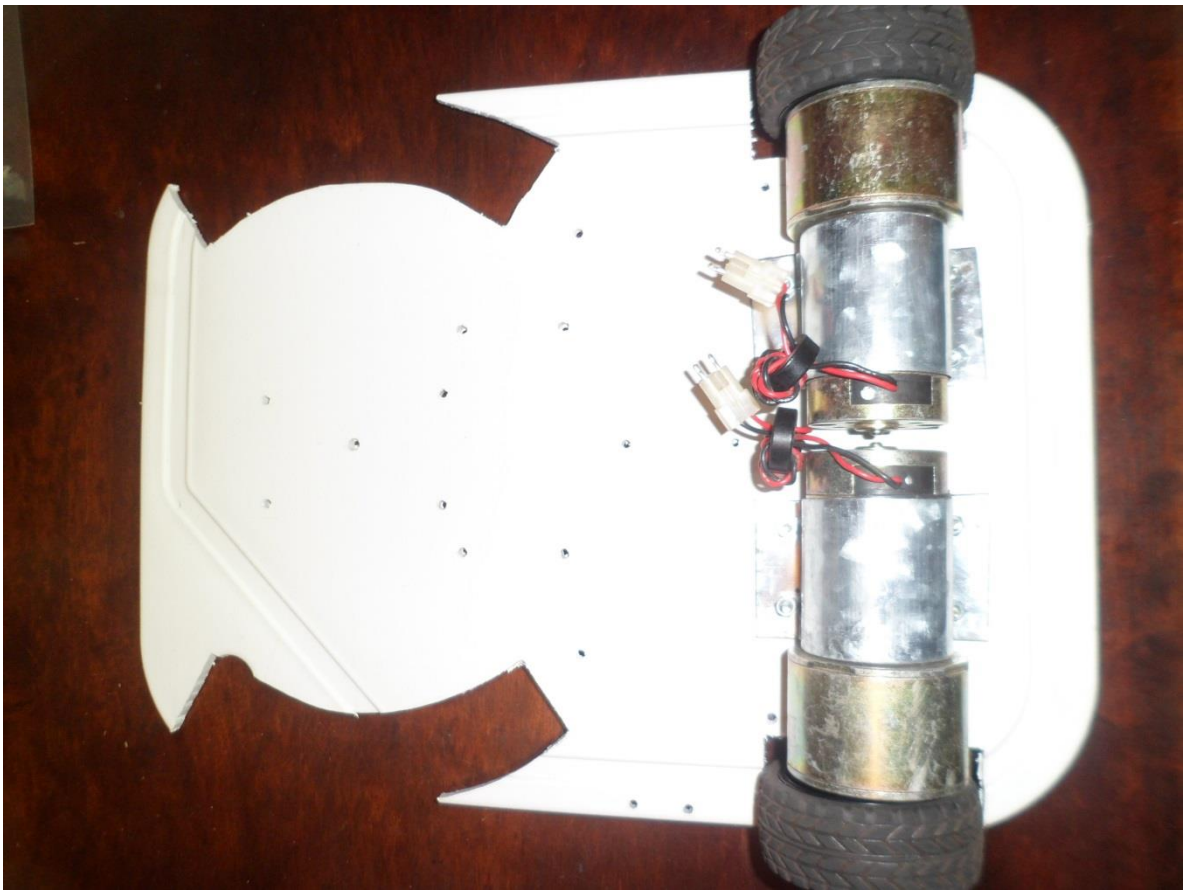

Figura 3.41 - Motores fixados na chapa de plástico.

Após fixados os motores, é necessário desenvolver a estrutura que irá controlar a direção do carro. Para desenvolver esta estrutura, foi necessário um servomotor MG995, 4 sarrafos de 2x2x3 cm, um sarrafo de 2x2x19 cm, uma chapa de MDF de 7mm x 12 cm x 5cm, seis parafusos sem porca e quatro parafusos com porcas.

Duas rodas foram parafusadas, uma em cada extremidade, no sarrafo de 19 cm. Após isto, uma plataforma composta pela chapa de MDF e os quatro sarrafos de 3 cm foi montada a fim de fixar o servomotor com uma orientação virada para baixo. Em seguida, o eixo do servo motor foi rotacionado para que ele ficasse posicionado na posição de 90 graus. Este posicionamento foi efetuado com auxílio do Arduino[2]. Posteriormente a isso, o servo motor foi fixado sobre a plataforma com o auxílio de quatro parafusos com porcas e o eixo do servomotor foi fixado sobre o sarrafo de 19 cm com o auxílio de dois parafusos e servirá como eixo fixo para as rodas da frente. Foi também fixada uma tábua retangular de madeira de 5x5x2 cm para servir como apoio para a plataforma da câmera. O resultado final desta montagem realizada para permitir a mudança de direção de movimentação do carro pode ser vista nas Figuras 3.42 e 3.43.

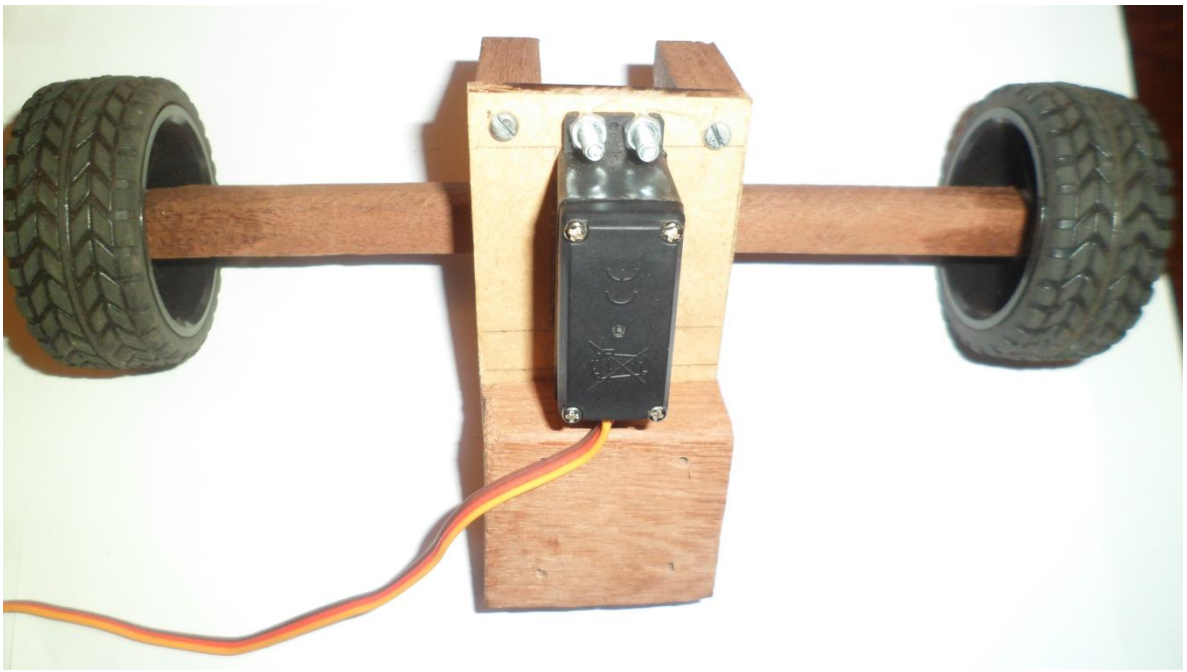

Figura 3.42 - Plataforma desenvolvida para o controle de Direção – Vista de cima.

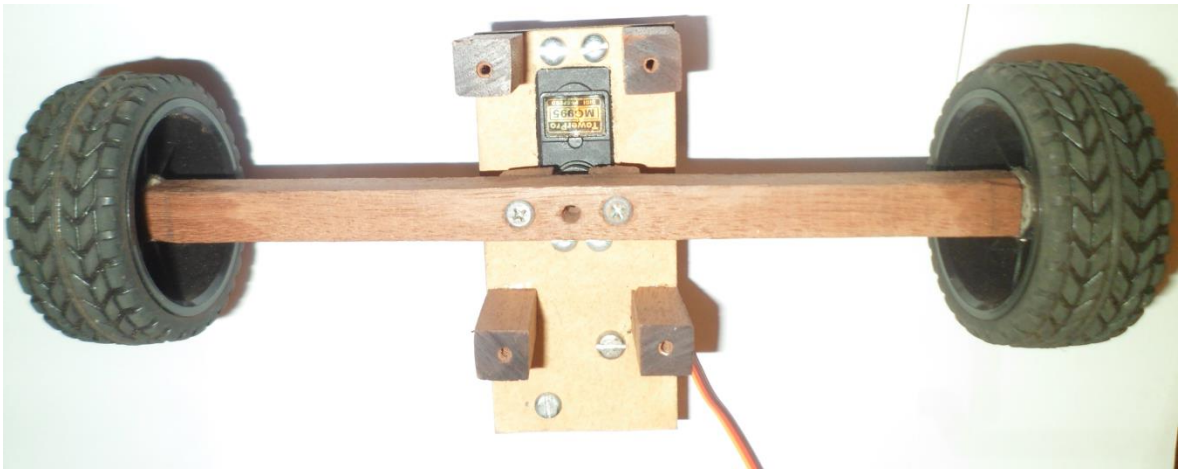

Figura 3.43 - Plataforma desenvolvida para o controle de Direção – Vista de baixo.

Após o desenvolvimento da plataforma, devemos fixá-la sobre a chapa de plástico. Para isso foram utilizados 4 parafusos sem porca e um parafuso com porca. O parafuso com porca passa sobre o furo central do sarrafo de 19 cm e os outros 4 parafusos são utilizados um em cada sarrafo de 3 cm.

Com a plataforma já fixada sobre a chapa de plástico, foi fixada a PCI controladora do motor com o auxílio de dois parafusos com porcas. A PCI e a plataforma fixadas sobre a chapa de plástico podem ser visualizadas nas Figuras 3.44 e 3.45.

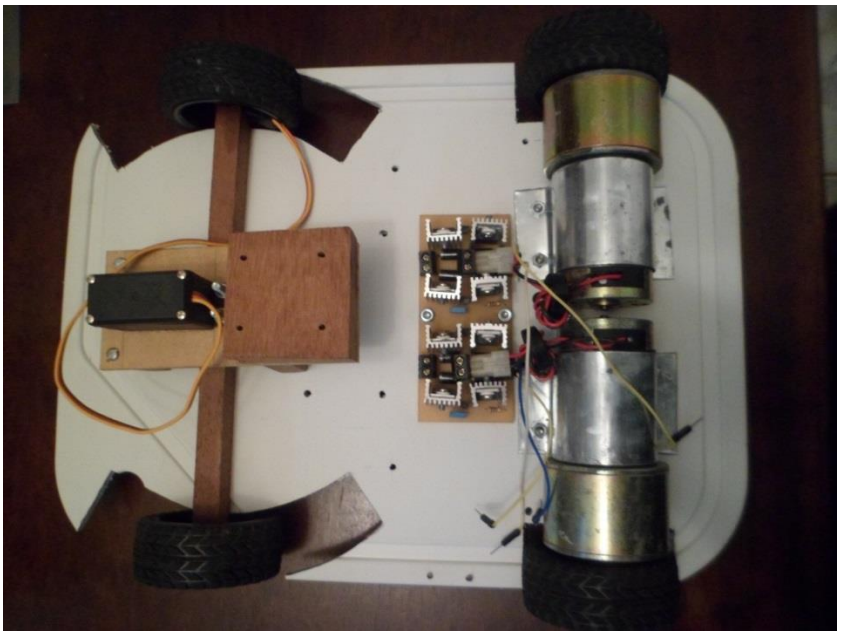

Figura 3.44 - Chapa de Plástico com motores, Ponte H e Plataforma de Direção fixados - Vista de Cima.

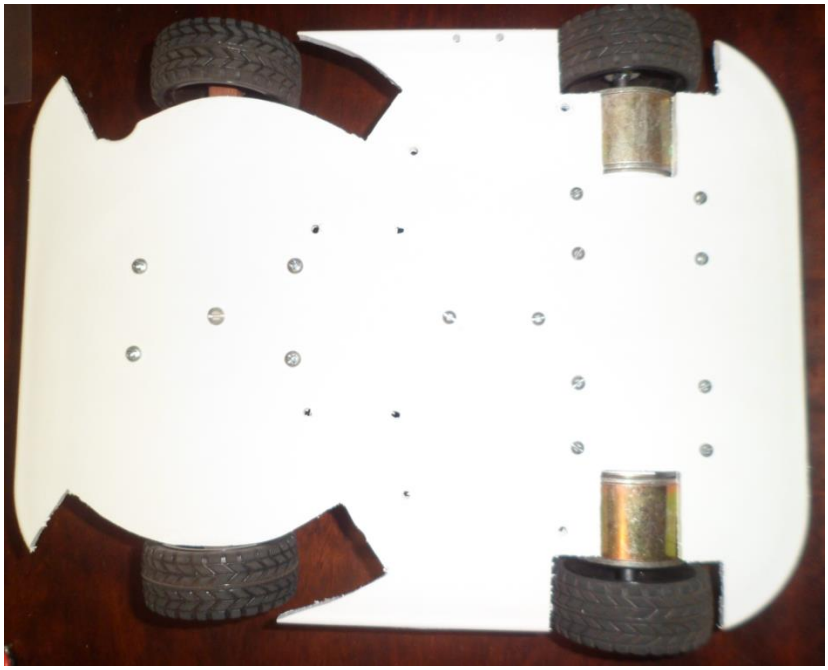

Figura 3.45 - Chapa de Plástico com motores, Ponte H e Plataforma de Direção fixados - Vista de Baixo.

O próximo passo é a fixação da bateria sobre a chapa. Para isso foram utilizadas duas abraçadeiras autotravantes de nylon. Após fixar a bateria, foi instalada a fiação e chave interruptora. O par de cabos de 2.5 mm de espessura foi dimensionado de tal forma que em uma de suas extremidades foi soldado um conector para a bateria e a outra foi dividida em três, sendo que duas destas derivações foram direcionadas para as duas pontes H e a outra derivação foi direcionada para a placa receptora. Antes de haver a derivação, o

cabo vermelho passou por uma chave interruptora para permitir que o desligamento do carro fosse efetuado utilizando-se apenas uma chave. Os cabos dos motores foram ligados nas saídas M1 e M2 de suas respectivas pontes H e dois fios foram fixados em cada uma das pontes H, sendo que a outra extremidades destes fios será posteriormente fixados na placa receptora. A Figura 3.46 mostra a bateria e a fiação instaladas.

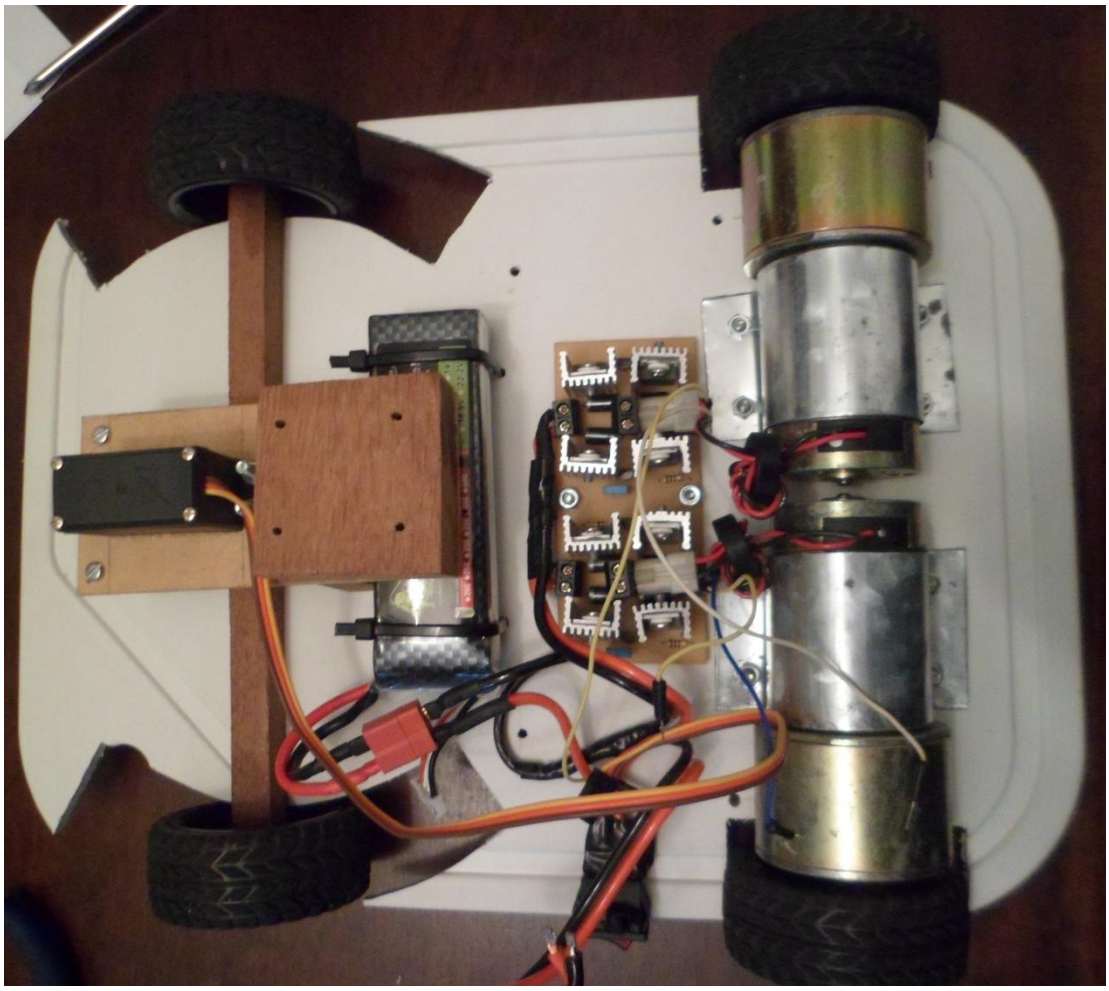

Figura 3.46 – Carro com a Bateria e Fiação instaladas

Para fazer a fixação da placa receptora e da Raspberry Pi[3], foi necessário o desenvolvimento de um tablado, utilizando para isso quatro sarrafos de 2x2x5 cm, uma chapa de plástico PVC 8 x 25 cm e quatro parafusos sem porcas. Para fixar este tablado na chapa de PVC do fundo, foram utilizados 4 parafusos sem porcas. O tablado já fixado sobre a chapa de PVC do fundo pode ser visto na Figura 3.47.

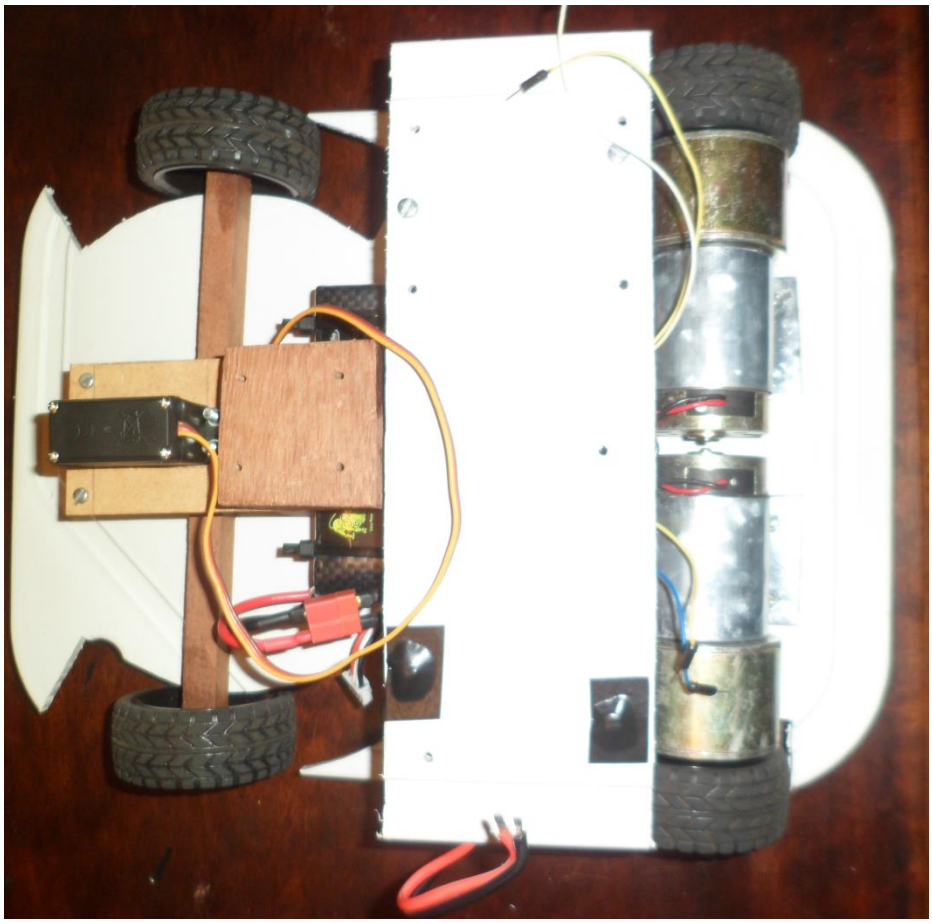

Figura 3.47 - Carro com o tablado fixado.

Após a fixação do tablado, foi fixado a PCI receptora. Para isso, foram efetuados dois furos sobre a PCI e foram passados dois parafusos com porca de baixo para cima sobre o tablado. A PCI fixada sobre o tablado pode ser vista na Figura 3.48.

Após a fixação da placa receptora, foi fixado sobre o carro a plataforma responsável pela movimentação da câmera. Para fixar a câmera sobre a plataforma, foi utilizada fita isolante. Para fixar a plataforma sobre o carro, foram utilizados quatro parafusos sem porcas, sendo que a fixação foi feita sobre a tábula retangular presente na plataforma responsável pelo controle de direção do carro. A plataforma com a câmera fixada assim com a ligação de toda a fiação pode ser vista na Figura 3.49.

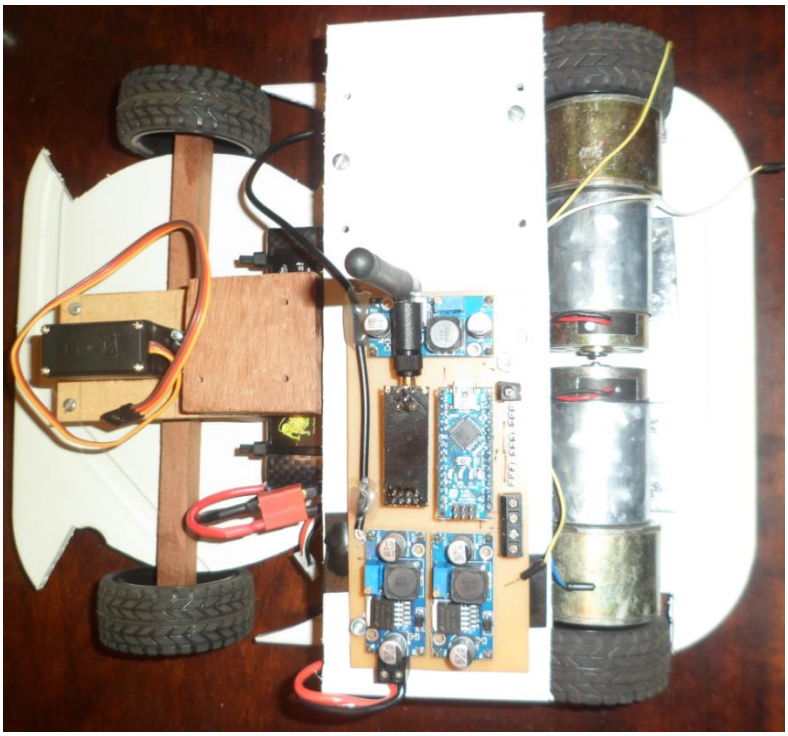

Figura 3.48 - Placa receptora fixada sobre o tablado.

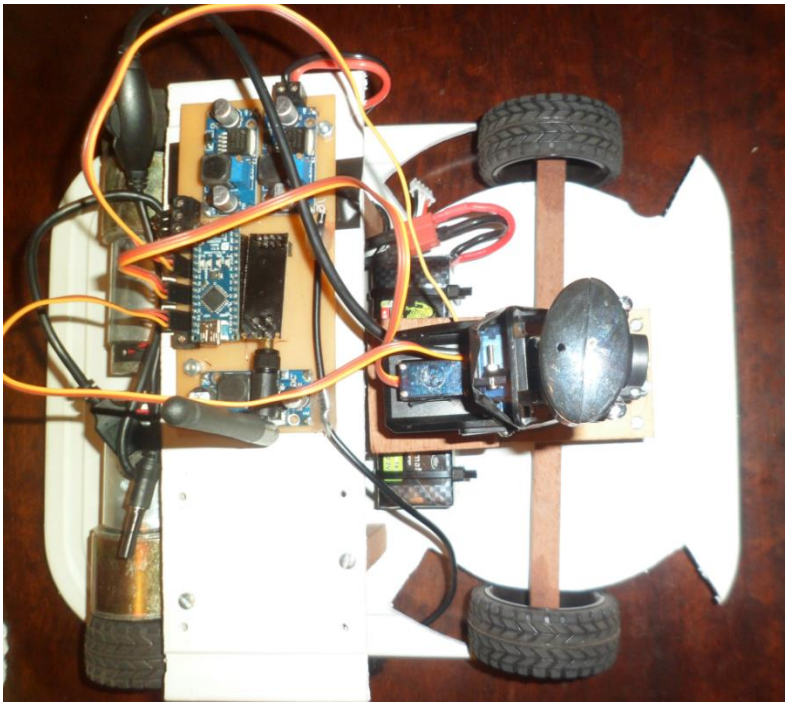

Figura 3.49 - Plataforma da câmera fixada sobre o carro.

Agora, basta apenas fixar a Raspberry Pi[3] e fazer as ligações necessárias. A Raspberry Pi[3] foi colocada dentro de uma caixa de acrílico para protegê-la e esta caixa foi fixada sobre o tablado com o auxílio de duas abraçadeiras auto travantes de nylon. As Figuras 3.50, 3.51 e 3.52 mostram o carro finalizado.

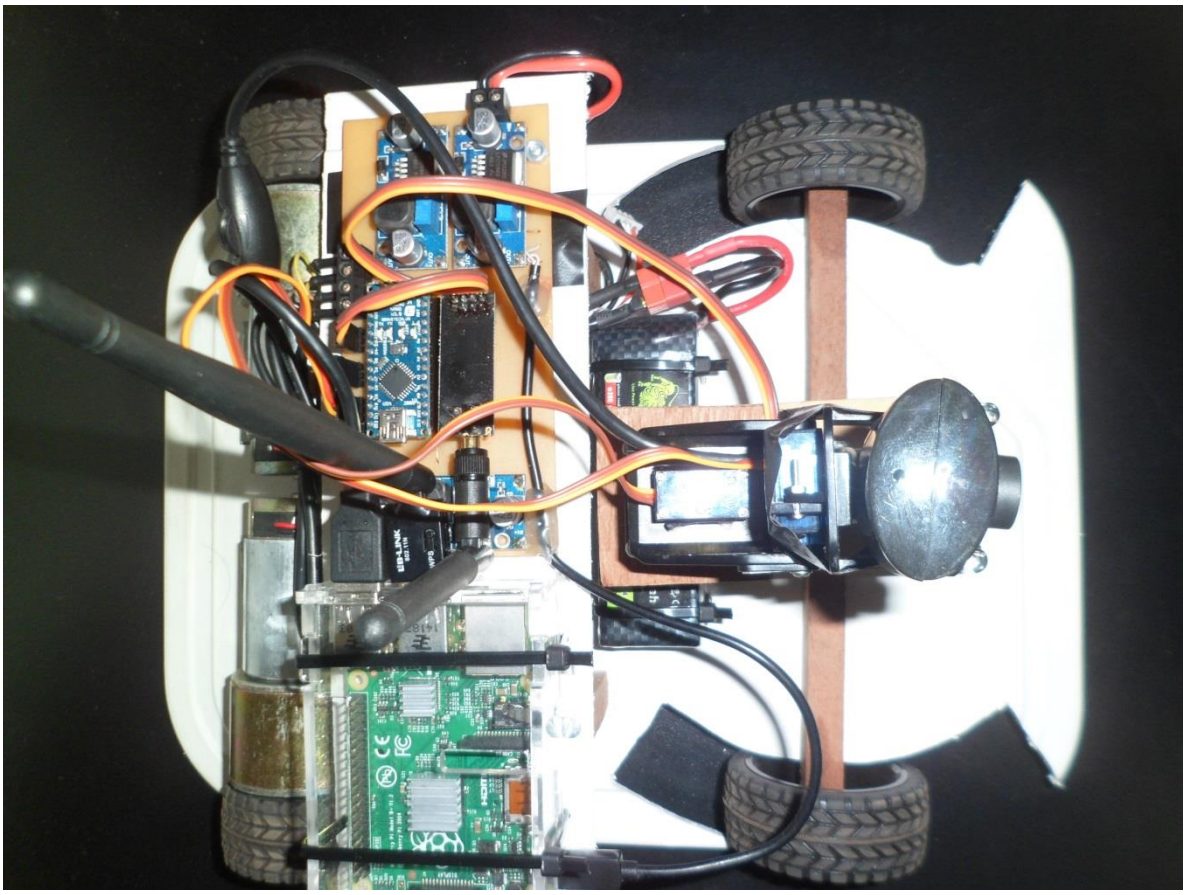

Figura 3.50 - Carro finalizado – Vista de Cima.

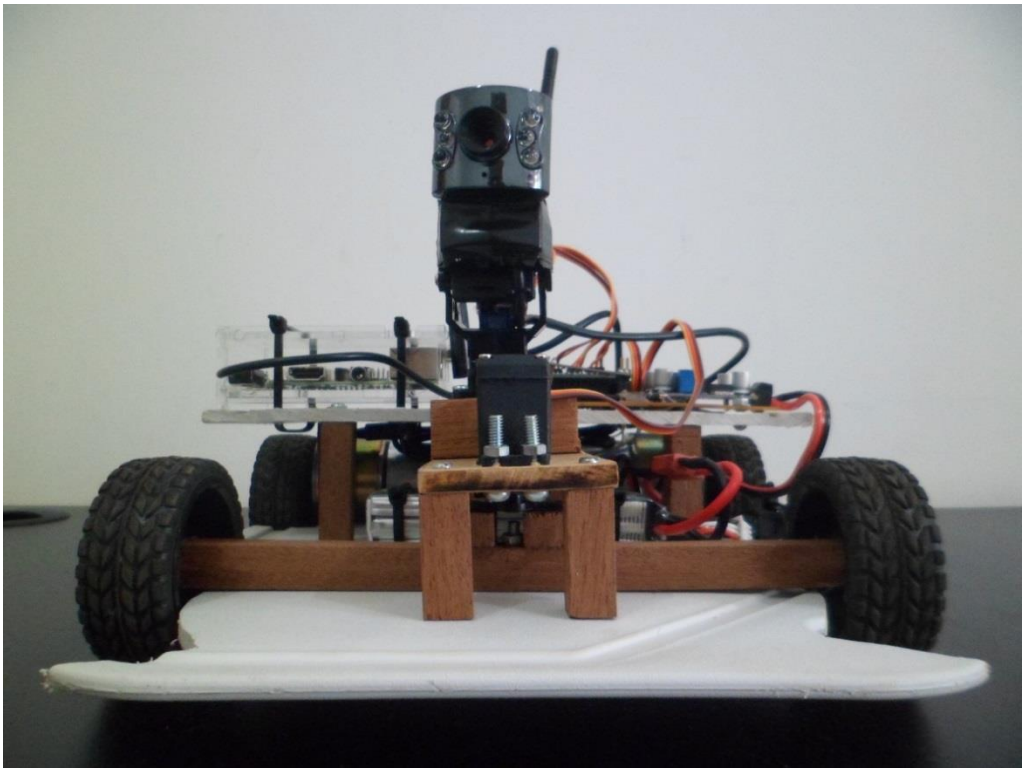

Figura 3.51 - Carro finalizado – Vista de Frente.

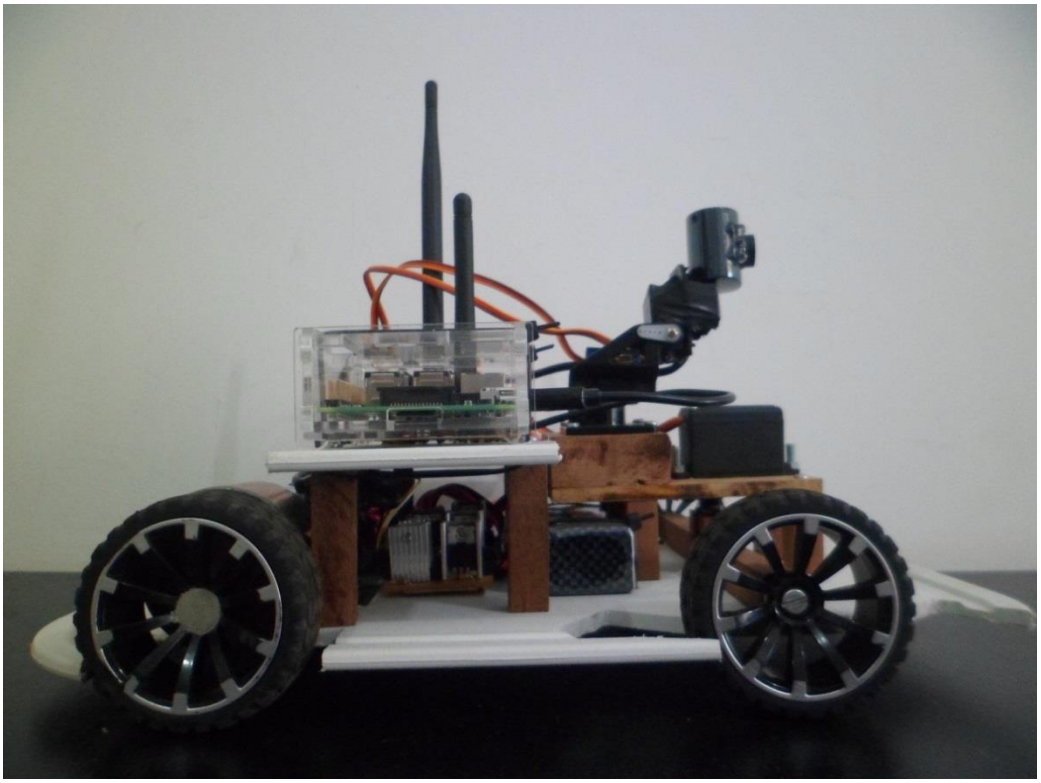

Figura 3.52 - Carro finalizado – Vista de Lado.

3.3.3 Desenvolvimento do Sistema de Software

Para descrever o desenvolvimento do sistema do software, esta seção é dividida em duas: a seção 3.3.3.1 para descrever o sistema que é executado no Arduino[2] e a seção 3.3.3.2 para descrever o sistema que é executado na Raspberry Pi[3].

3.3.3.1 Desenvolvimento do sistema executado no Arduino no Carro

O Arduino[2] presente no carro é responsável por receber os dados via ondas rádio enviados pelo controle remoto e acionar adequadamente os servomotores e motores presentes no carro.

Os dados são enviados do módulo de rádio NRF24L01 para o Arduino[2] através de uma interface SPI. O módulo de rádio deve ser conectado em um conjunto de portas que operem como SPI do Arduino[2], sendo que no modelo Nano, estas portas são as portas 11(MISO),12(MOSI) e13(SCK) e duas portas digitais qualquer (CE e CSN). Para fazermos a comunicação entre o Arduino[2] e o módulo de rádio foi utilizada a biblioteca “NR24.h”, disponível em [16]. Também foi necessário incluir a biblioteca “SPI.h”, disponível por default na IDE.

As saídas responsáveis pelo acionamento do motor devem ser saídas analógicas já que serão capazes de controlar a velocidade dos motores. Uma saída analógica por motor é

o suficiente. A outra saída pode ser digital. No caso deste trabalho, foram utilizadas uma saída analógica e uma saída digital para o controle de cada motor. Estas saídas foram ligadas nas pontes H responsáveis por controlar a direção de rotação dos motores.

As saídas responsáveis pelo acionamento dos servomotores devem ser algumas das saídas de D2 a D13. Para fazer o controle do posicionamento do eixo do servomotor, foi utilizada a biblioteca “Servo.h” disponível por default na IDE.

Para o desenvolvimento do código fonte, compilação e upload do arquivo hexadecimal gerado a partir do código fonte para a programação do microcontrolador foi utilizada a IDE Arduino 1.6.

O código fonte compilado pode ser visualizado no Código A.2, presente no Anexo A.

Após o desenvolvimento do código fonte, basta dar o upload do arquivo hexadecimal gerado pelo computador no Arduino[2] conectando-o ao computador através de um cabo USB.

3.3.3.2 Desenvolvimento do sistema executado na Raspberry Pi no Carro

O sistema contido na Raspberry Pi[3] do carro é capaz de criar uma rede WiFi utilizando um adaptador wireless USB. A imagem capturada pela câmera no carro é disponibilizada em uma porta IP e funciona como um servidor Apache. O desenvolvimento completo do desenvolvimento do sistema é mostrado nas seções de 3.3.3.2.1 à 3.3.3.2.5. A escolha por uma comunicação utilizando o protocolo WiFi pra a transmissão da imagem foi realizada, visto que ela permite que vários dispositivos móveis possam se conectar na rede WiFi criada pelo carro e possam exibir as imagens captadas pela câmera presente no carro.

3.3.3.2.1 Instalação do Sistema Operacional

O sistema operacional executado na Raspberry Pi[3] do carro é uma distribuição Linux Debian[8] compilada para a Raspberry Pi[3], o Raspbian[7]. A versão utilizada é uma versão NOOB e pode ser encontrada em [18]. A imagem deste SO está em um arquivo chamado "2015-05-05-raspbian-wheezy.img”. Como a Raspberry Pi[3] não possui memória não volátil, tanto o sistema operacional quanto os dados devem ser salvos em uma memória externa. No caso deste trabalho utilizamos apenas um cartão de memória micro SD de 8GB de capacidade classe 2.

Para o desenvolvimento de todo o sistema foi utilizado um notebook DELL VOSTRO 3550 com o sistema operacional Ubuntu 14.04 instalado. Portanto, todos os comandos listados nos passos são comandos nativos do SO Linux.

Para a gravação do sistema operacional no cartão de memória basta seguir os passos descritos no Apêndice B.4. Após seguir esses passos, o SO já está instalado e pronto para ser utilizado.

3.3.3.2.2 Instalação do Motion

Para fazer a captura das imagens e disponibilizá-las em uma porta IP, é necessário um software que faça isso. Para realizar esta tarefa, foi utilizado o software “Motion” [10]. Para fazer a instalação dele, deve-se executar o seguinte comando:
sudo apt-get install motion.

Além de fazer a captura e disponibilizar as imagens em uma porta IP, o “Motion” terá como função salvar as imagens capturas em um arquivo de vídeo comprimido utilizando uma compressão MPEG. Para realizar isto, deve-se alterar o arquivo /etc/motion/motion.conf com o comando sudo nano /etc/motion/motion.conf.

Em seguida deve-se apagar todo o conteúdo do arquivo, colar o código C.1 presente no Apêndice C e salvar o arquivo.

As principais alterações contidas neste arquivo em relação ao arquivo original são a definição da resolução da imagem, da taxa de compressão, da quantidade de frames por segundo, a sensibilidade de movimento para iniciar a gravação, onde deve ser salvo o arquivo e o número da porta IP na qual estará disponível a imagem capturada. Como é desejado que o arquivo seja salvo tanto no carro quanto no controle remoto, foram necessárias as criações de duas threads, sendo que uma delas é responsável por salvar o arquivo localmente e a outra é responsável por salvar o arquivo no controle remoto através de uma pasta compartilhada na rede pelas duas Raspberries Pi[3]. Os arquivos responsáveis pela execução das threads são os arquivos etc/motion/thread1.conf e etc/motion/thread2.conf. Deve-se modificar estes arquivos através dos comandos sudo nano /etc/motion/thread1.conf e sudo nano /etc/motion/thread2.conf. O conteúdo de cada um destes arquivos deve ser substituído pelos Códigos C.2 e C.3 respectivamente presentes no Apêndice C.

3.3.3.2.3 Instalação do Servidor Apache e criação da página web

A Raspberry Pi[3] contida no carro irá funcionar como um servidor e irá disponibilizar a imagem recebida pela câmera em uma porta IP. Para fazer com que a Raspberry Pi[3] funcione como um servidor, basta fazer a instalação de um servidor Apache[11]. Para isso, basta executar o seguinte comando:
sudo apt-get install apche2.

Após a instalação do Apache, deve-se configurar a página web a qual a Raspberry Pi[3] do controle remoto irá realizar o acesso.

Ao instalarmos o servidor Apache, uma página web default já é criada e pode ser acessada por um browser digitando o endereço IP da Raspberry Pi[3] no endereço acessado pelo browser. Deve-se editar a página web de tal forma que ela exiba as imagens

capturadas pela webcam. Está página web está localizada em /var/www/index.php. Deve-se então executar o seguinte comando:

```
sudo nano /var/www/index.php
```

Em seguida deve-se colar o código 3.1 abaixo e salvar o arquivo.

Código 3.1 - index.php

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">
<html>
    <head></head>
    <body>
        <a href="http://192.168.42.1:2015/">
        <img src="http://192.168.42.1:2015/" alt="Camera 1" width="320"$
    </body>
</html>
```

Vale a pena ressaltar que o endereço da imagem deve possuir o mesmo número da porta IP que foi definida no arquivo /etc/motion/motion.conf.

3.3.3.2.4 Configuração da Rede WiFi

A Raspberry Pi[3] presente no carro irá funcionar como um "*Acess Point*[9]". Para isso, a Raspberry Pi[3] deve estar conectada a uma rede cabeada com acesso à internet e deve estar com o adaptador wireless USB conectado. Deve-se seguir uma sequência de passos presentes no Apêndice B.5. Esses passos foram executados seguindo a referencia [19].

Após seguir estes 19 passos, a Raspberry Pi[3] funcionará como um "*Acess Point*"[9] todas as vezes em que ela for iniciada, criando a Rede WiFi TCC, com o endereço estático de IP 192.168.42.1, que será também o endereço que deverá ser acessado no browser pela Raspberry Pi[3] do controle remoto para acessar a página do servidor Apache criado para exibição das imagens capturadas através de uma porta IP.

3.3.3.2.5 Sistema de Arquivos

O arquivo contendo o vídeo comprimido capturado pela câmera presente no carro é salvo tanto no carro quanto no controle remoto. Para isso, foi necessária a criação de um sistema de arquivos para a criação de uma pasta compartilhada. Esta pasta está localizada na Raspberry Pi[3] do controle remoto e é acessado pela Raspberry Pi[3] localizada no carro. Portanto, a Raspberry Pi[3] do controle remoto será o servidor e a Raspberry Pi[3] do carro será o cliente.

O sistema de arquivo escolhido foi o sistema de arquivos NFS (Network File System). Este tipo de sistema foi desenvolvido para permitir que se possam montar partições ou diretórios remotos como se fosse um disco local. Ele permite especificar diferentes permissões de acesso a cada cliente de acesso.

Para configurar a Raspberry Pi[3] do carro como cliente, deve-se efetuar os passos descritos no Apêndice B.6, também descritos em [17].

3.4 Conexão dos Componentes

As conexões entre todos os componentes de Hardware, tanto do carro quanto do controle remoto, podem ser vistas na Figura 3.53. A linha vermelha indica que existe uma ligação entre os componentes. A Seta azul significa uma conexão sem fio entre os componentes.

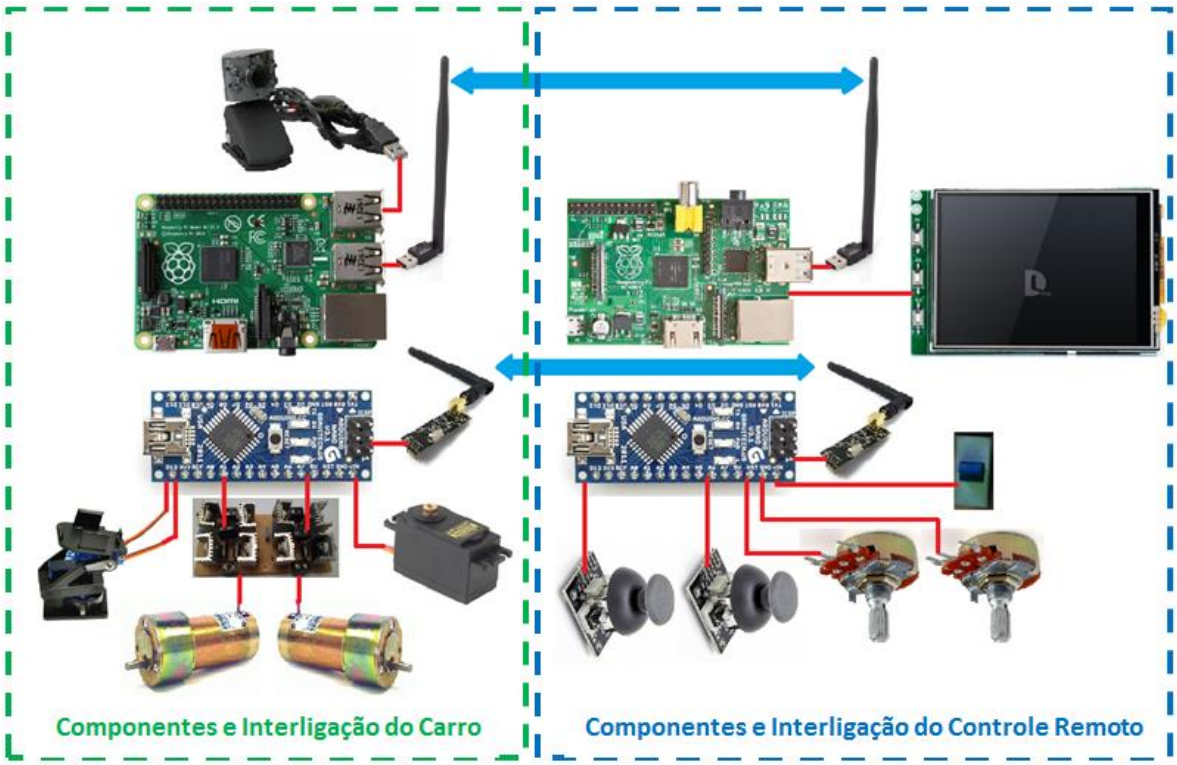

Figura 3.53 – Diagrama de Conexão entre os componentes de Hardware do projeto

As conexões entre todos os componentes de Software, tanto do carro quanto do controle remoto, podem ser vistas na Figura 3.54. A linha vermelha indica que existe uma ligação entre os componentes. A Seta azul significa uma conexão sem fio entre os componentes.

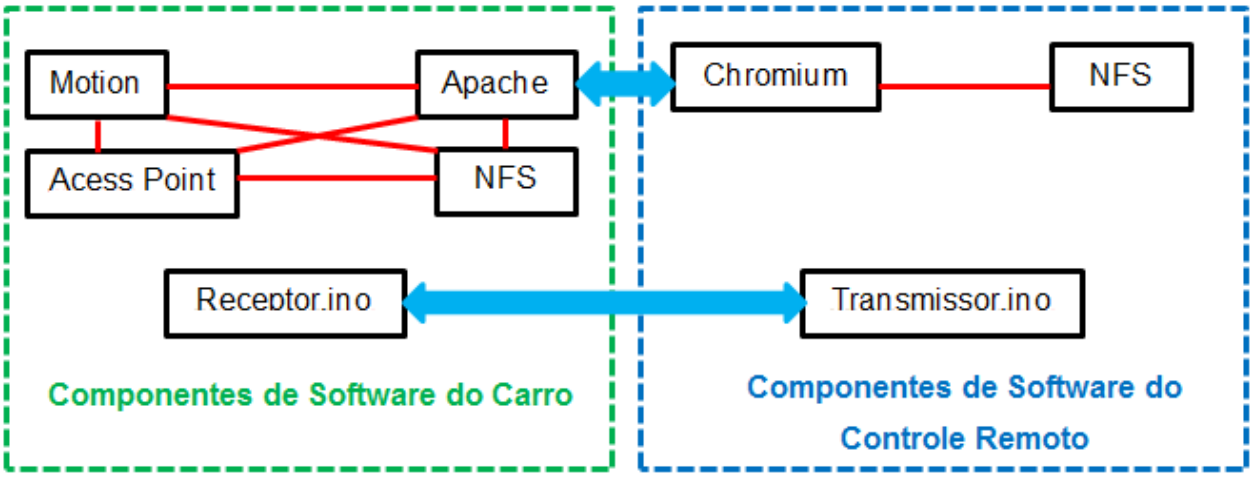

Figura 3.54 – Diagrama de Conexão entre os componentes de Software do projeto

O paradigma de Engenharia de Sofware utilizado em todo o projeto foi o Modelo Cascata, o qual requer uma abordagem sistemática, sequencial ao desenvolvimento do software. Ele abrange as seguintes atividades: Engenharia de Sistemas, Análise de Requisitos, Projeto, Codificação, Testes e Manutenção. O diagrama contendo a ordem destas atividades pode ser vista na Figura 3.55.

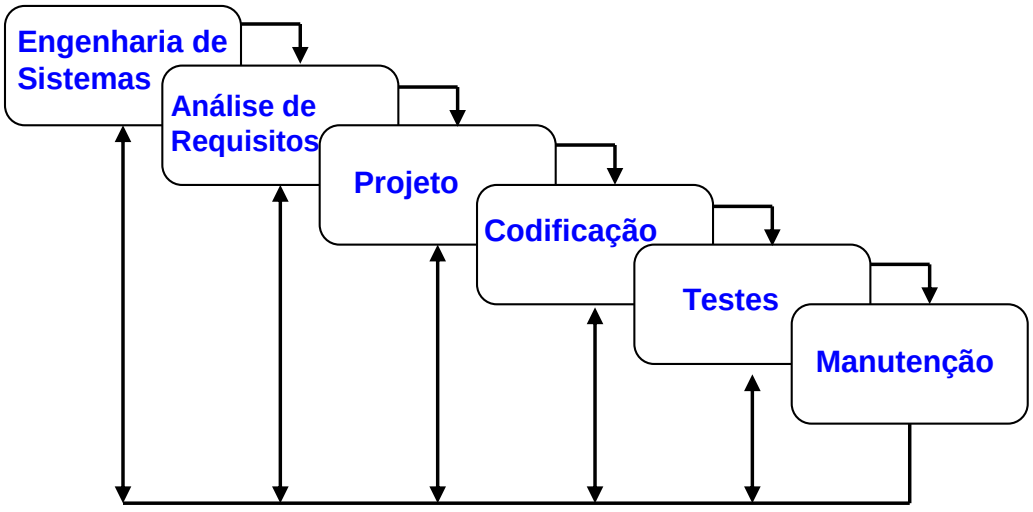

Figura 3.55 - Modelo Cascata de Engenharia de Software.

4 Resultados

Após a conclusão da montagem e configuração do carro e do controle remoto, foi observado o comportamento e desempenho de algumas funcionalidades. Nas subseções seguintes são mostrados e discutidos os resultados obtidos.

4.1 Consumo de Energia

Para realizar a medida de consumo de energia tanto por parte do controle remoto quanto por parte do carro foi utilizado um amperímetro de corrente contínua com ambos em operação.

A partir do valor da corrente consumida pelo controle remoto e pelo carro e sabendo a capacidade de carga das baterias utilizadas, podemos estimar a autonomia de ambos.

Os consumos de corrente do controle remoto e do carro parado e em velocidade máxima assim como a autonomia máxima de cada um destes casos podem ser vistos na Tabela 4.1.

Tabela 4.1 - Consumo e Tempo de Autonomia do Controle Remoto e do Carro

	Controle Remoto	**Carro Parado**	**Carro em Velocidade Máxima em Terreno Plano**
Consumo de Corrente	0,7 A	0,6 A	3,0 A
Tempo de Autonomia	1 hora e 41 minutos	3 horas e 40 minutos	44 minutos

Para estimar a autonomia da bateria com os motores em diferentes velocidades, foi traçado um gráfico mostrado na Figura 4.1.

4.2 Velocidade do carro

Para realizar a medida de velocidade média atingida pelo carro foi feita a medição de uma distância percorrida em um terreno plano em um determinado tempo e dividiu-se a distância percorrida pelo tempo. Como a velocidade do carro pode ser variada em função do nível de ajuste desejado, foi traçado um gráfico que mostra a velocidade do carro em terreno plano em função do *duty cicle* utilizado no sinal PWM que controla a velocidade do motor. Este gráfico pode ser visto na Figura 4.2.

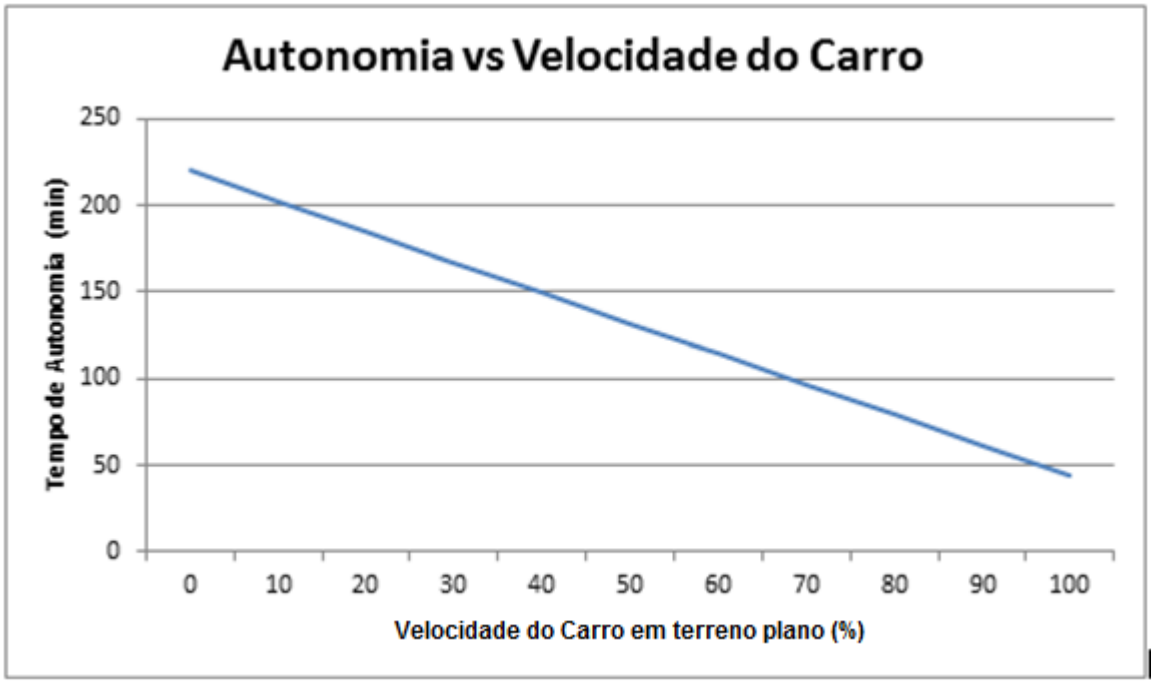

Figura 4.1 - Gráfico de autonomia vs velocidade do carro em terreno plano

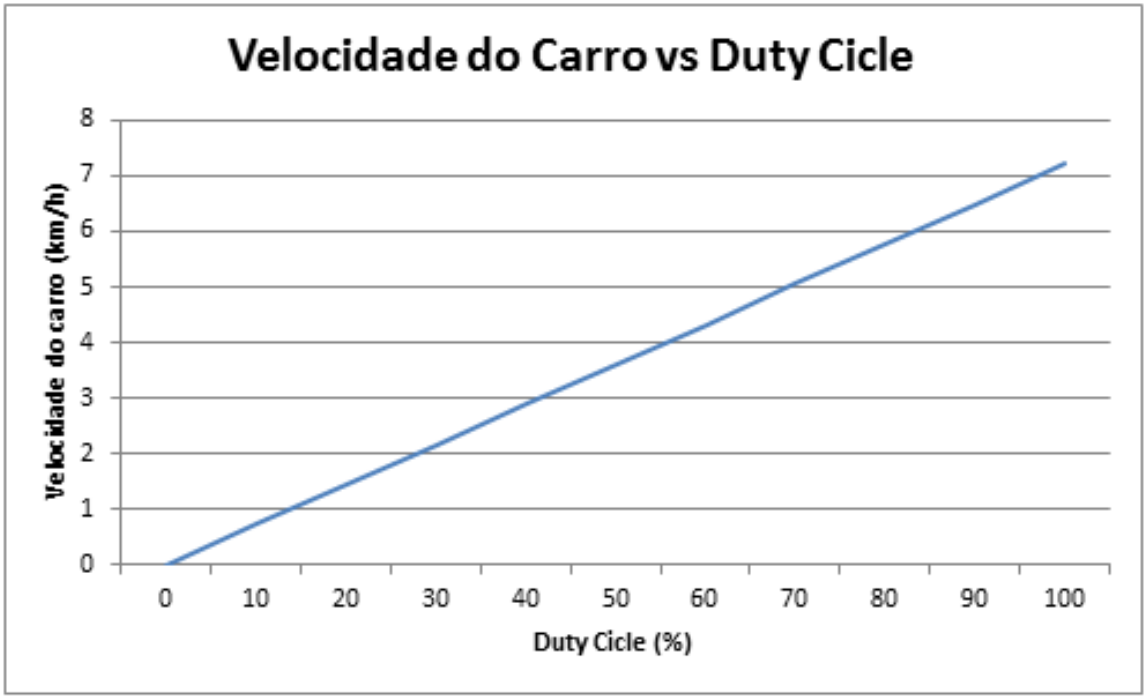

Figura 4.2 - Gráfico de Velocidade do Carro em terreno plano vs *Duty Cicle*

A velocidade máxima atingida pelo carro foi de 7,2 km/h, inferior à velocidade projetada de 9,8 km/h. Esta variação provavelmente se deu devido ao fato de o motor utilizado ser um motor remanufaturado, o que pode ter reduzido o seu desempenho. Outro

fator que pode ter influenciado na velocidade se consiste no fato de o motor não manter sua velocidade máxima quando um carga é submetida a ele.

4.3 Alcance da Transmissão sem Fio

Após a realização de testes de alcance através do distanciamento do carro em relação ao controle até o ponto em que os sinais não eram transmitidos entre o carro e o controle remoto para os dois tipos de transmissão sem fio presentes no trabalho, foi possível a inserção dos dados contido na Tabela 4.2. Os valores presentes na tabela são valores aproximados e refletem a ordem de grandeza da distância real.

Tabela 4.2 - Alcance da Transmissão sem Fio

	Alcance Aproximado
Transmissão WiFi	200 m
Transmissão RF	300 m

A partir dos dados da Tabela 4.2, podemos concluir que uma distância segura para a operação do carro é uma distância inferior a 200 metros entre o controle remoto e o carro.

4.4 Taxa de Transmissão

Para realizar a medida da taxa de transmissão de dados entre a Raspberry Pi[3] do carro e do controle remoto foi utilizado o software “iptraf”. A partir dele, é possível observar que a taxa de transmissão de dados entre os dois se dá a uma velocidade média de 510 kb/s. Um printscreen da tela do software em execução pode ser visto na Figura 4.3.

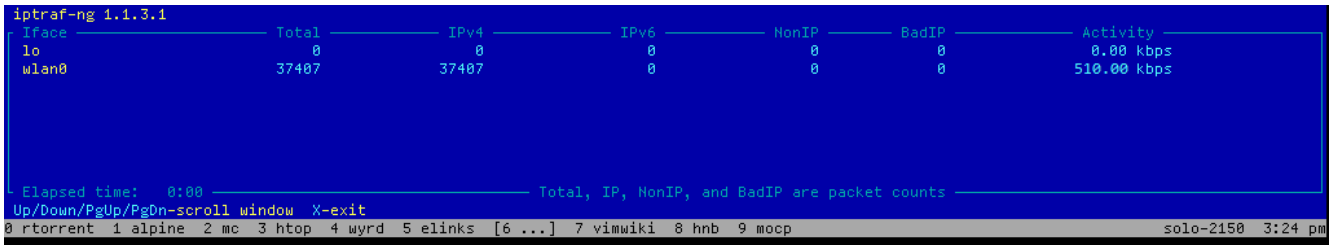

Figura 4.3 - Software “iptraf” em execução

4.5 Taxa de Amostragem da Câmera

Após a realização de uma inspeção teste, analisou-se o vídeo gerado pela câmera e constatou-se que a taxa de quadros por segundo é igual a 4, ou seja, a taxa de amostragem de imagens que o sistema produz é igual a 4 fps.

4.6 Processamento e Memória

Para realizar a medida do nível de processamento e de memória utilizados nas Raspberries Pi[3] do controle remoto e do carro, foi utilizado o comando top. As Figuras 4.4 e 4.5 mostram os níveis de processamento e memória utilizados pelo controle remoto e pelo carro respectivamente.

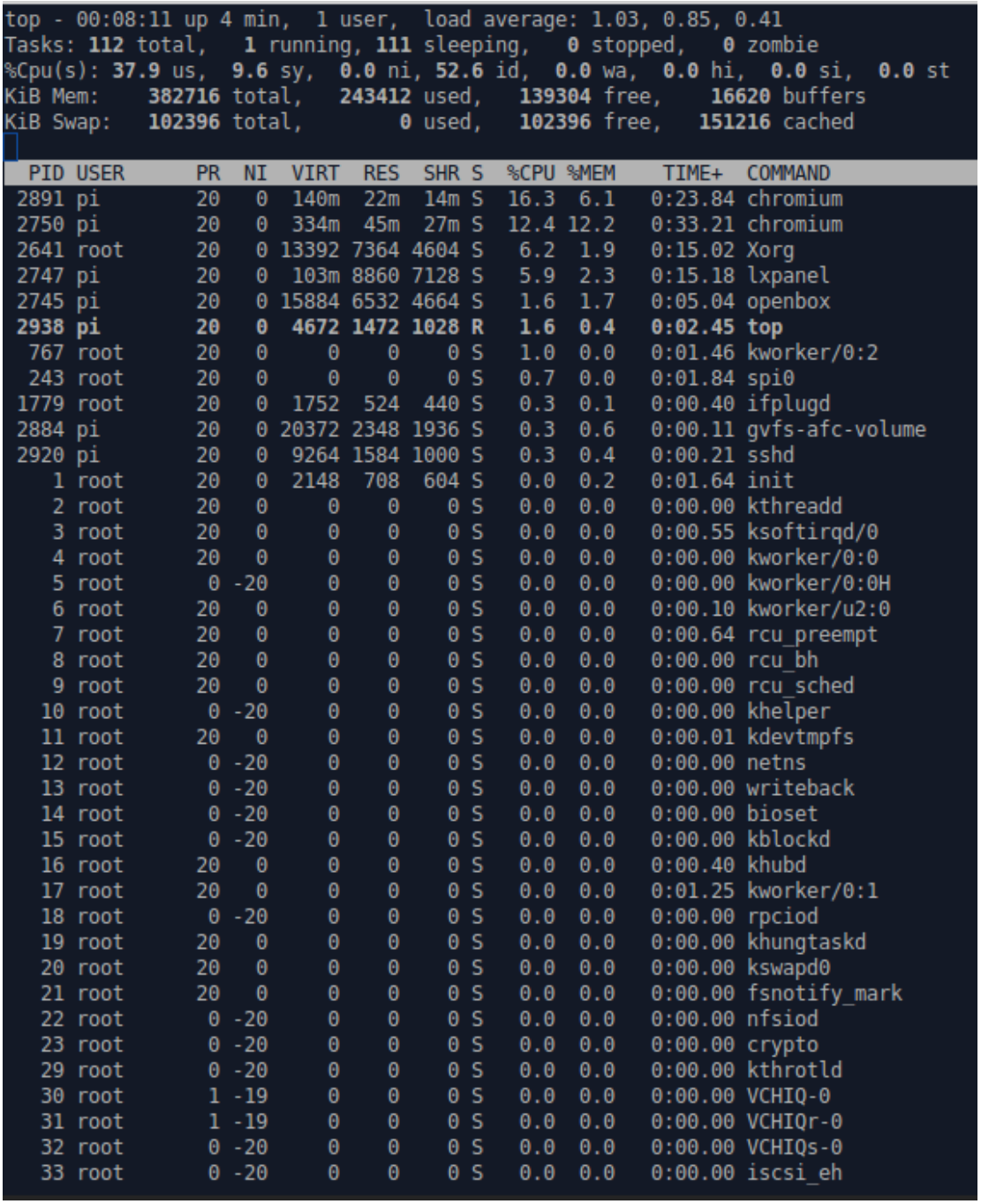

Figura 4.4 – Processos executados pela Raspberry Pi[3] do Controle Remoto

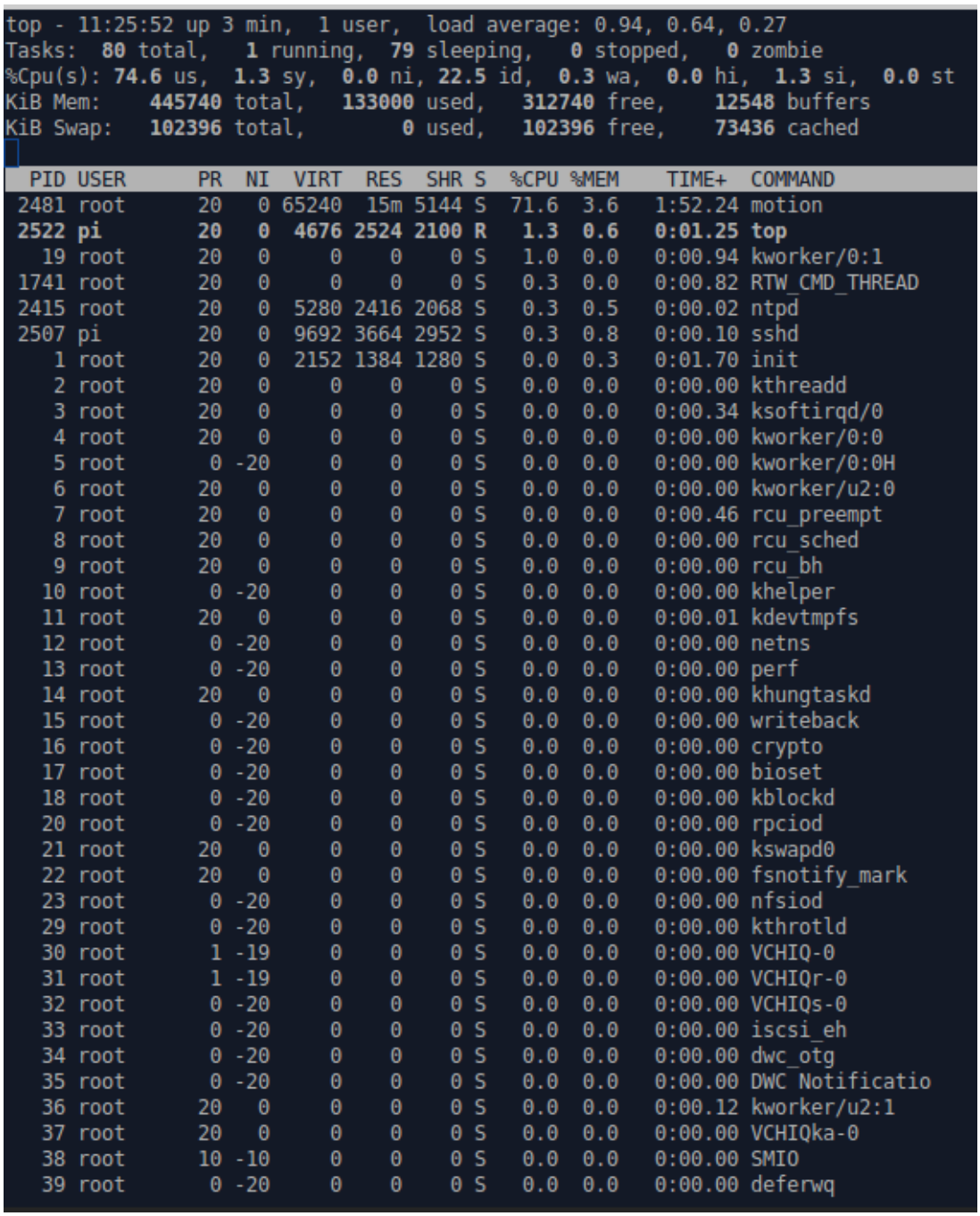

Figura 4.5 – Processos executados pela Raspberry Pi[3] do Carro

Podemos perceber que o nível de CPU utilizado pela Raspberry Pi[3] do carro é relativamente alto, porém a memória utilizada pela mesma é bastante baixa. Já na Raspberry Pi[3] do controle remoto, podemos perceber que o nível de CPU utilizado é relativamente baixo e o nível de memória utilizado é relativamente alto se o compararmos com o nível utilizado pela Raspberry Pi[3] do carro.

4.7 Custo

Um dos objetivos do projeto era o baixo custo. O custo para elaborar todo o projeto foi de cerca de R$ 400,00. Este valor é um valor relativamente baixo se comparado com plataformas disponíveis no mercado para executarem os demais objetivos presentes neste projeto.

5 Conclusão

Os acionamentos de motores e servomotores a partir de leituras analógicas e digitais em um controle remoto assim como a captura e a transmissão e exibição de imagens através de uma rede sem fio podem parecer uma funcionalidade bastante simples, porém dependendo de como é feita a implementação, estas tarefas podem se tornar bastante complexas. Para implementar essas funcionalidade, foi necessário projetar uma interface, construir um controle remoto e uma plataforma móvel, além de implementar todas etapas de integração dos componentes do sistema.

Durante o desenvolvimento deste projeto foi possível avaliar a eficiência de algumas das escolhas feitas para realizar a implementação de cada etapa necessária para conclusão do projeto. Algumas destas escolhas e suas implicações são apresentadas e discutidas nos próximos parágrafos.

A utilização de rodas pequenas é uma alternativa bastante barata e que facilita o controle do carro. Porém, em locais em que o terreno é bastante irregular, estas rodas podem se tornar ineficientes, o que exigiria o uso de rodas maiores ou esteiras. Esta alternativa pode aumentar o custo do projeto, mas, dependendo da aplicação, é uma mudança necessária.

O protocolo de comunicação entre as duas Raspberries Pi[3] utilizando rede sem fio mostrou-se possível e bastante flexível. A utilização de outras tecnologias como uma alternativa para estabelecer a comunicação poderia trazer vantagens relacionadas com a distância máxima de alcance e velocidade de comunicação, porém a escolha adotada foi capaz de transmitir dados na frequência necessária.

A utilização de plataformas de hardware e software *open source*[1] mostrou-se bastante eficiente devido à possuir uma comunidade ativa que compartilha e disponibiliza soluções para a maioria dos problemas encontrados durante o projeto.

A utilização de Linux Embarcado mostrou-se uma solução apropriada por se tratar de uma alternativa personalizável e com a facilidade para a expansão das funcionalidades oferecidas pela sua utilização na plataforma móvel e no controle remoto.

O software utilizado para fazer a captura das imagens pela câmera e disponibilização destas em uma porta IP apresentou uma taxa de quadros por segundo que pode ser menor do que o necessário dependendo do tipo detecção desejada. Embora ele cumpra o seu papel, a taxa de quadros por segundo transmitidas quando se escolhe uma boa resolução para a imagem não é tão grande quanto desejado. Em algumas aplicações na qual é necessário fazer a captura de movimentações rápidas no ambiente, este software não é uma boa escolha. Em situações na qual este requisito não é exigido, o software utilizado se mostra eficiente devido a sua facilidade de instalação e configuração.

Foi possível perceber também que existe um grande número de alternativas diferentes para solucionar os mesmos problemas. Este leque de possibilidades ressalta a importância da elaboração de estudos comparativos entre as alternativas para descobrir quais as vantagens e desvantagens entre cada uma delas e tomar decisões de projeto mais coerentes com o objetivo de alcançar uma solução com alta qualidade e eficiência.

O sistema se mostrou capaz de realizar inspeções de locais de difícil acesso, visto que, tanto a câmera quanto o display utilizados possuem uma resolução mediana. A facilidade para a mobilidade do carro também é um fator que torna a inspeção capaz de ser efetuada.

Após a realização dessas considerações pode-se afirmar que foi possível alcançar o objetivo proposto neste projeto, aprofundando e desenvolvendo os conhecimentos necessários para a implementação de um sistema dependente da integração entre software, hardware e componentes mecânicos. Realizar essa integração de tecnologias diferentes e desenvolver de novas tecnologias faz parte da missão de um engenheiro de computação. A realização de projetos como esse é uma ótima maneira de aprofundar-se nesse universo de conhecimento de uma forma prática, didática e funcional.

5.1 Trabalhos Futuros

O projeto desenvolvido mostrou-se bastante funcional e atingiu os objetivos propostos, porém há algumas melhorias que poderiam ser implementadas. A seguir são citadas algumas destas possíveis melhorias:

- Utilização de rodas maiores para possibilitar o trânsito do carro em superfícies irregulares;
- Utilização de baterias com maior capacidade de carga para permitir uma autonomia maior ao carro e ao controle remoto;
- Transferir o controle dos motores e dos servomotores para a Raspberry Pi[3] presente no carro, o que eliminaria a necessidade do Arduino[2] no carro;
- Utilização de um software de captura e transmissão de imagem na rede mais eficiente do que o software “Motion” utilizado, o que poderia permitir uma taxa de maior de quadros na captura das imagens. O MJPG-streamer poderia ser uma solução melhor;
- Incluir a utilização de sensores para evitar que a plataforma venha a se colidir com algum objeto, o que poderia danificar a mesma;
- Desenvolvimento de uma interface gráfica que permita uma maior portabilidade para outras plataformas;

- Desenvolvimento de uma proteção externa tanto para carro quanto para o controle remoto para aumentar suas resistências a choques mecânicos e melhoramento do design dos mesmos.

Referências Bibliográficas:

[1] Opensource.org, “Welcome to The Open Source Initiative | Open Source Initiative”, 2015. Disponível em: <http://opensource.org> Acesso em 25 de Maio de 2015.

[2] M. Banzi, “Getting started with Arduino”. Beijing: Make:Books / O'Reilly, 2009.

[3] E. Upton and G. Halfacree, “Raspberry Pi user guide”. Chichester, West Sussex, UK: Wiley, 2012.

[4] Raspberry Pi, “Raspberry Pi – Teach, Learn, and Make with Raspberry Pi”, 2015. Disponível em: <https://www.raspberrypi.org> Acesso em 03 de Maio de 2015.

[5]D. Seal, “ARM architecture reference manual”. Harlow: Addison-Wesley, 2001.

[6] M. Richardson, “Getting started with BeagleBone”. Sebastopol, [Calif.]: Maker Media, 2013.

[7] Raspbian.org, “FrontPage – Raspbian”, 2015. Disponível em: <http://www.raspbian.org> Acesso em 06 de Maio de 2015.

[8] M. Krafft, “The Debian system”. San Francisco: No Starch Press, 2005.

[9] Techterms.com, “Access Point Definition”, 2015. Disponível em: <http://techterms.com/definition/accesspoint> Acesso em 05 de Maio de 2015.

[10] Lavrsen.dk, 'WebHome < Motion < Foswiki', 2015. Disponível em: <http://www.lavrsen.dk/foswiki/bin/view/Motion/WebHome> Acesso em 16 de Maio de 2015.

[11] D. Group, “Welcome! – The Apache HTTP Server Project”, Httpd.apache.org, 2015. Disponível em: <htttp://httpd.apache.org> Acesso em 30 de Maio de 2015.

[12] Arduino.cc, “Arduino - ArduinoBoardNano”, 2015. Disponível em: <http://www.arduino.cc/en/Main/ArduinoBoardNano> Acesso em 02 de Maio de 2015.

[13] Arduino.cc, “Arduino - ArduinoBoardUno”, 2015. Disponível em: <http://www.arduino.cc/en/Main/ArduinoBoardUno> Acesso em 02 de Maio de 2015.

[14] Batteryuniversity.com, “Advantages & Limitations of the Lithium-ion Battery - Battery University”, 2015. Disponível em: <http://batteryuniversity.com/learn/article/is_lithium_ion_the_ideal_battery> Acesso em 03 de Maio de 2015.

[15] Maximintegrated.com, “DC-DC Converter Tutorial - Tutorial – Maxim”, 2015. Disponível em: < http://www.maximintegrated.com/en/app-notes/index.mvp/id/2031> Acesso em 03 de Maio de 2015.

[16] GitHub, “maniacbug/RF24”, 2013. Disponível em: <https://github.com/maniacbug/RF24> Acesso em 06 de Maio de 2015.

[17] Vivaolinux.com.br, “Configurando um servidor NFS em 4 passos [Artigo]”, 2015. Disponível em: <http://www.vivaolinux.com.br/artigo/Configurando-um-servidor-NFS-em-4-passos>. Acesso em 06 de Maio de 2015.

[18] Raspberry Pi, “Raspberry Pi Downloads - Software for the Raspberry Pi”, 2015. Disponível em:<https://www.raspberrypi.org/downloads> Acesso em 16 de Maio de 2015.

[19] Learn.adafruit.com, 'What you'll need | Setting up a Raspberry Pi as a WiFi access point | Adafruit Learning System', 2015. Disponível em: <https://learn.adafruit.com/setting-up-a-raspberry-pi-as-a-WiFi-access-point/what-youll-need> Acesso em 16 de Maio de 2015.

Apêndice A – Códigos Fonte executados nos Arduinos

Código A.1 – Código Arduino do Controle Remoto

```
#include <SPI.h>
#include "nRF24L01.h"
#include "RF24.h"

int comando[7];
int chave=4;

RF24 radio(9,10);

const uint64_t pipe = 0xE8E8F0F0E1LL;

void setup(void)
{
  pinMode(4, INPUT);
  radio.begin();
  radio.openWritingPipe(pipe);
}

void loop(void)
{
  comando[0] = analogRead(A2);      //movimentacao carro frente/re
  comando[1] = analogRead(A3);      //movimentacao carro direita/esquerda
  comando[2] = analogRead(A5);      //movimentacao camera cima/baixo
  comando[3] = analogRead(A6);      //movimentacao camera direita/esquerda
  comando[4] = digitalRead(chave); //movimentacao tipo carro/tanque
  comando[5] = analogRead(A1);      //velocidade do motor esquerdo
  comando[6] = analogRead(A0);      //velocidade do motor direito

  radio.write(comando, sizeof(comando)); //Escreve comandos no rádio
}
```

Código A.2 – Código Arduino do Carro

```
#include <SPI.h>
#include "nRF24L01.h"
#include "RF24.h"
#include <Servo.h>

Servo camera_yaw;
Servo camera_pitch;
Servo direcao;
float camera_yaw_angle=90.0f;
float camera_pitch_angle=90.0f;
float direcao_angle=90.0f;
int motor11=6;
int motor12=A0;
int motor21=5;
int motor22=A1;
int vel_motor1;
int vel_motor2;
int tempo_sem_sinal=0;
int comando[7];
//comando[0]=movimentacao carro frente/re
//comando[1]=movimentacao carro direita/esquerda
```

```
//comando[2]=movimentacao camera cima/baixo
//comando[3]=movimentacao camera direita/esquerda
//comando[4]=movimentacao tipo carro/tanque
//comando[5]=velocidade motor da esquerda
//comando[6]=velocidade motor da direita

RF24 radio(9,10);
const uint64_t pipe = 0xE8E8F0F0E1LL;

void setup(void)
{
  pinMode(motor11, OUTPUT);
  pinMode(motor12, OUTPUT);
  pinMode(motor21, OUTPUT);
  pinMode(motor22, OUTPUT);
  radio.begin();
  radio.openReadingPipe(1,pipe);
  radio.startListening();
  camera_yaw.attach(4);
  camera_pitch.attach(7);
  direcao.attach(3);
  camera_yaw.write((int)camera_yaw_angle);
  camera_pitch.write((int)camera_pitch_angle);
  direcao.write((int)direcao_angle);
}

void loop(void)
{
  if ( radio.available() )
  {
    bool done = false;
    while (!done)
    {
      done = radio.read(comando, sizeof(comando));
    }

    tempo_sem_sinal=0;

      if(comando[4]==0)//Movimentacao do tipo carro
    {
      //Controlando a movimentaço frente/re
      if(comando[0]>823) //Carro para frente
      {
         vel_motor1=(1023-comando[6])/4;
         analogWrite(motor11,vel_motor1);
         digitalWrite(motor12,HIGH);
         vel_motor2=comando[5]/4;
         analogWrite(motor21,vel_motor2);
         digitalWrite(motor22,LOW);
      }
      else if((comando[0]<200)&&(comando[0]>-1)) //Carro para traz
      {
         vel_motor1=comando[6]/4;
         analogWrite(motor11,vel_motor1);
         digitalWrite(motor12,LOW);
         vel_motor2=(1023-comando[5])/4;
         analogWrite(motor21,vel_motor2);
         digitalWrite(motor22,HIGH);
      }
      else //Carro parado
      {
```

```
          digitalWrite(motor11,LOW);
          digitalWrite(motor12,LOW);
          digitalWrite(motor21,LOW);
          digitalWrite(motor22,LOW);
      }

      //Controlando a direcao
      if(comando[1]>823) //Carro para direita
      {
        direcao_angle= 100.0f;
      }
      else if((comando[1]<100)&&(comando[1]>-1)) //Carro para esquerda
      {
        direcao_angle= 80.0f;
      }
      else //Carro em linha reta
      {
        direcao_angle= 90.0f;
      }
    }

    else //movimentacao do tipo tanque
    {
      direcao_angle=90.0f;
      if(comando[1]>823) //Carro circula direita
      {
          vel_motor1=comando[6]/4;
          analogWrite(motor11,vel_motor1);
          digitalWrite(motor12,LOW);
          vel_motor2=comando[5]/4;
          analogWrite(motor21,vel_motor2);
          digitalWrite(motor22,LOW);
      }
      else if((comando[1]<100)&&(comando[1]>-1)) //Carro circula esquerda
      {
          vel_motor1=(1023-comando[6])/4;
          analogWrite(motor11,vel_motor1);
          digitalWrite(motor12,HIGH);
          vel_motor2=(1023-comando[5])/4;
          analogWrite(motor21,vel_motor2);
          digitalWrite(motor22,HIGH);
      }
      else if(comando[0]>823) //Carro para frente
      {
          vel_motor1=(1023-comando[6])/4;
          analogWrite(motor11,vel_motor1);
          digitalWrite(motor12,HIGH);
          vel_motor2=comando[5]/4;
          analogWrite(motor21,vel_motor2);
          digitalWrite(motor22,LOW);
      }
      else if((comando[0]<200)&&(comando[0]>-1)) //Carro para traz
      {
          vel_motor1=comando[6]/4;
          analogWrite(motor11,vel_motor1);
          digitalWrite(motor12,LOW);
          vel_motor2=(1023-comando[5])/4;
          analogWrite(motor21,vel_motor2);
          digitalWrite(motor22,HIGH);
      }
    }
```

```
    //Controlando yaw da camera
    if(comando[2]>823) //Rotaciona camera sentido horario
    {
      if(camera_yaw_angle<180.2)
        camera_yaw_angle+=0.2;
    }
    else if((comando[2]<200)&&(comando[2]>-1)) //Rotaciona camera sentido
anti-horario
    {
      if(camera_yaw_angle>-0.2)
        camera_yaw_angle-=0.2;
    }

    //Controlando pitch da camera
    if(comando[3]>823) //Rotaciona camera para cima
    {
      if(camera_pitch_angle>-0.2)
        camera_pitch_angle-=0.2;
    }
    else if((comando[3]<200)&&(comando[3]>-1)) //Rotaciona camera para
baixo
    {
      if(camera_pitch_angle<180.2)
        camera_pitch_angle+=0.2;
    }
  }
  else
  {
      if(tempo_sem_sinal==1023) //Verifica se o sinal já sumiu faz tempo
      {                         //Em caso positivo, pára os motores
       digitalWrite(motor11,LOW);
       digitalWrite(motor12,LOW);
       digitalWrite(motor21,LOW);
       digitalWrite(motor22,LOW);
      }
      else
            tempo_sem_sinal++;
  }

  //Seta os servo motores para a posicao correta
  camera_yaw.write((int)camera_yaw_angle);
  camera_pitch.write((int)camera_pitch_angle);
  direcao.write((int)direcao_angle);
}
```

Apêndice B – Instalação do Sistema Presente nas Raspberrys Pi

Apêndice B.1 – Instalação do Sistema Operacional no Controle Remoto

- Passo 1: Inserir o cartão de memória no notebook.
- Passo 2: Abrir o terminal e digitar o comando: df -h . Este comando irá listar todos os dispositivos montados.
- Passo 3: Descobrir qual dos dispositivos montados é o referente ao cartão de memória e desmontar ele com o comando umount /dev/sdd1, onde sdd1 deve ser substituído pela unidade referente ao cartão.
- Passo 4: Acessar a pasta a qual o arquivo de imagem do sistema operacional está salvo e executar o comando: dd bs=4M if= TFT320TP-Raspberry-140921-V3.0.img of=/dev/sdd, onde sdd é a unidade referente ao cartão de memória.
- Passo 5: Após finalizar a gravação da imagem descrita no Passo 4, basta remover o cartão SD e inseri-lo na Raspberry Pi[3] e conectar um cabo de rede com acesso à internet, um teclado USB e um cabo HDMI ligado a um monitor na Raspberry Pi[3]. Após isto, devemos alimentar a Raspberry Pi[3] com uma fonte de 5 V para ligarmos a Raspberry Pi[3].
- Passo 6: A partir deste passo todos os passos são executados direto na Raspberry Pi[3] sem o auxílio do notebook. Ao realizar o primeiro boot, abrirá o “Rasp Config”, que pode ser visto na Figura B.1. Nele devemos configurar os seguintes itens:
    - Setar a opção Expand File System
    - Change User Password. Neste trabalho, o password foi alterado para “tcc”.
    - Overclock. Neste trabalho, o ovlerclock escolhido foi o de 800 MHz sem sobre tensão.
    - Em Advanced Options, foi setado o Acesso via ssh.
- Passo 7: Reinicie a Raspberry Pi[3].
- Passo 8: Logue com o usuário “pi” e o password “tcc”.
- Passo 9: Instalação das atualizações do SO: com a Raspberry Pi[3] conectada na rede, execute o comando: sudo apt-get update. Logo em seguida o comando sudo apt-get upgrade.
- Passo 10: Reinicie a Raspberry Pi[3] utilizando o comando sudo reboot.

Figura B.1 - “Rasp Config”.

Apêndice B.2 – Configuração da Rede WiFi no Controle Remoto

Ligar a Raspberry Pi e executar o seguinte comando para modificar o arquivo /etc/wpa_supplicant/wpa_supplicant.conf:

```
sudo nano /etc/wpa_supplicant/wpa_supplicant.conf
```

Deve-se adicionar as seguintes linhas no arquivo:

```
network={
        ssid="TCC"
        scan_ssid=1
        psk="tccfernando2015"
        key_mgmt=WPA-PSK
        id_str="home"
        priority=2
}
```

Deve-se agora executar o seguinte comando para modificar o arquivo /etc/network/interfaces:

```
sudo nano /etc/network/interfaces
```

Deve-se adicionar as seguintes linhas no arquivo:

```
iface wlan0 inet manual
wpa-roam /etc/wpa_supplicant/wpa_supplicant.conf

iface home inet static
address 192.168.42.2
netmask 255.255.255.0
gateway 192.168.42.1

iface default inet dhcp
```

Apêndice B.3 – Configuração do Sistema de Arquivos no Controle Remoto

- Passo 1: Editar o arquivo /etc/exports para definir a pasta a ser compartilhada e permissões de acesso. Para isso deve-se executar o comando: `sudo nano /etc/exports` e em seguida adicionar a seguinte linha no arquivo:

```
# Path do diretório        IP do cliente  Permissões de acesso
/usr/pi/motion_videos/   192.168.42.1  (rw,no_root_squash,sync)
```

Para mais informações sobre os tipos de permissões, acessar [17].

- Passo 2: Editar o arquivo /etc/hosts.deny para definir a segurança do compartilhamento. Para isso deve-se executar o comando: `sudo nano /etc/hosts.deny` e em seguida adicionar as seguintes linhas ao final do arquivo:

```
portmap: ALL
lockd: ALL
mountd: ALL
rquotad: ALL
```

- Passo 3: Editar o arquivo /etc/hosts.allow para definir o IP ou faixa de IPs que poderão "concorrer" aos compartilhamentos, ou seja, especificar quem terá acesso a cada serviço. Para isso deve-se executar o comando: `sudo nano /etc/hosts.allow` e em seguida adicionar as seguintes linhas ao final do arquivo:

```
portmap: 192.168.42.1
lockd: 192.168.42.1
rquotad: 192.168.42.1
mountd: 192.168.42.1
statd: 192.168.42.1
```

- Passo 4: Para a conclusão da configuração do servidor NFS, deve-se preparar o sistema para inicializar o serviço NFS no boot. Para isso, deve-se digitar os seguintes comandos:

```
cd /etc/rc.d
chmod a+x rc.portmap
chmod a+x rc.nfsd
```

- Passo 5: Deve-se então agora reiniciar o SO. Para isso, basta executar o comando `sudo reboot`

Apêndice B.4 – Instalação do Sistema Operacional no Carro

- Passo 1: Inserir o cartão de memória no notebook.
- Passo 2: Abrir o terminal e digitar o comando: df -h . Este comando irá listar todos os dispositivos montados.
- Passo 3: Descobrir qual dos dispositivos montados é o referente ao cartão de memória e desmontar ele com o comando umount /dev/sdd1, onde sdd1 deve ser substituído pela unidade referente ao cartão.
- Passo 4: Acessar a pasta a qual o arquivo de imagem do sistema operacional está salvo e executar o comando: dd bs=4M if= 2015-05-05-raspbian-wheezy.img of=/dev/sdd, onde sdd é a unidade referente ao cartão de memória.
- Passo 5: Após finalizar a gravação da imagem descrita no Passo 4, basta remover o cartão SD e inseri-lo na Raspberry Pi[3] e conectar um cabo de rede com acesso à internet, um teclado USB e um cabo HDMI ligado a um monitor na Raspberry Pi[3]. Após isto, devemos alimentar a Raspberry Pi[3] com uma fonte de 5 V para ligarmos a Raspberry Pi[3].
- Passo 6: A partir deste passo todos os passos são executados direto na Raspberry Pi[3] sem o auxílio do notebook . Ao realizar o primeiro boot, abrirá o Rasp Config, que pode ser visto na Figura B.1. Nele devemos configurar os seguintes itens:
    - Setar a opção Expand File System
    - Change User Password. Neste trabalho, o password foi alterado para “tcc”.
    - Overclock. Neste trabalho, o ovlerclock escolhido foi o de 800 MHz sem sobre tensão.
    - Em Advanced Options, foi setado o Acesso via ssh.
- Passo 7: Reinicie a Raspberry Pi[3].
- Passo 8: Logue com o usuário “pi” e o password “tcc”.
- Passo 9: Instalação das atualizações do SO: com a Raspberry Pi[3] conectada na rede, execute o comando: sudo apt-get update. Logo em seguida o comando sudo apt-get upgrade.
- Passo 10: Reinicie a Raspberry Pi[3] utilizando o comando sudo reboot.

Apêndice B.5 – Configuração da Rede WiFi no Carro

- Passo 1: Instalação do “hostpad” e do “isc-dhcp-server”. Para isso, deve-se executar o seguinte comando: sudo apt-get install hostapd isc-dhcp-server.

- Passo 2: Configurar o servidor DHCP alterando do arquivo /etc/dhcp/dhcpd.conf. Para isso, deve-se executar o seguinte comando: sudo nano /etc/dhcp/dhcpd.conf. Deve-se substituir as seguintes linhas

```
option domain-name "example.org";
option domain-name-servers ns1.example.org, ns2.example.org;
```

por:

```
#option domain-name "example.org";
#option domain-name-servers ns1.example.org, ns2.example.org;
```

de forma que estas linhas fiquem comentadas. Deve-se também substituir as seguintes linhas:

```
# If this DHCP server is the official DHCP server for the local
# network, the authoritative directive should be uncommented.
#authoritative;
```

por:

```
# If this DHCP server is the official DHCP server for the local
# network, the authoritative directive should be uncommented.
authoritative;
```

de forma a descomentar a linha “authoritative”.

Por fim deve-se adicionar as seguintes linhas ao fim do arquivo:

```
subnet 192.168.42.0 netmask 255.255.255.0 {
        range 192.168.42.10 192.168.42.50;
        option broadcast-address 192.168.42.255;
        option routers 192.168.42.1;
        default-lease-time 600;
        max-lease-time 7200;
        option domain-name "TCC";
        option domain-name-servers 8.8.8.8, 8.8.4.4;
}
```

-Passo 3: Alterar o arquivo /etc/default/isc-dhcp-server. Para isso, deve-se executar o comando sudo nano /etc/default/isc-dhcp-server. Deve-se alterar a seguinte linha:

```
INTERFACES=""
```

por:

```
INTERFACES="wlan0"
```

- Passo 4: Definir um IP estático alterando o arquivo /etc/network/interfaces. Para isso, deve-se executar o comando sudo nano /etc/network/interfaces. Deve-se adicionar as seguintes linhas ao fim do arquivo:

```
iface wlan0 inet static
  address 192.168.42.1
  netmask 255.255.255.0
```

- Passo 5: Definir um IP estático para o adaptador wireless USB. Para isso, deve-se executar o seguinte comando: sudo ifconfig wlan0 192.168.42.1.

- Passo 6: Configurar o "*Acess Point*[9]". Para isso deve-se criar o arquivo /etc/hostapd/hostapd.conf através do comando sudo nano /etc/hostapd/hostapd.conf e colar as seguintes linhas no arquivo:

```
interface=wlan0
driver=rtl871xdrv
ssid=TCC
hw_mode=g
channel=6
macaddr_acl=0
auth_algs=1
ignore_broadcast_ssid=0
wpa=2
wpa_passphrase=tccfernando2015
wpa_key_mgmt=WPA-PSK
wpa_pairwise=TKIP
rsn_pairwise=CCMP
```

- Passo 7: É necessário indicar para a Raspberry onde está o arquivo de configuração do "hostpad". Para isso, deve-se alterar o arquivo /etc/default/hostapd através do comando sudo nano /etc/default/hostapd. Deve-se trocar a linha:

```
#DAEMON_CONF=""
```

por:

```
DAEMON_CONF="/etc/hostapd/hostapd.conf"
```

- Passo 8: Permitir o acesso a vários clientes se necessário. Para isso deve-se alterar o arquivo /etc/sysctl.conf através do comando sudo nano /etc/sysctl.conf. Deve-se adicionar a seguinte linha ao fim do arquivo.

```
net.ipv4.ip_forward=1
```

- Passo 9: Para ativar a alteração feita no Passo 8 deve-se executar o comando sudo sh -c "echo 1 > /proc/sys/net/ipv4/ip_forward".

- Passo 10: Para criar uma translação entre a rede cabeada contida na porta eth0 e a rede sem fio contida na porta wlan0, deve-se executar os seguintes comandos:

```
sudo iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
sudo iptables -A FORWARD -i eth0 -o wlan0 -m state --state RELATED,ESTABLISHED -j ACCEPT
sudo iptables -A FORWARD -i wlan0 -o eth0 -j ACCEPT
```

-Passo 11: Para que esta translação ocorra automaticamente ao reiniciar a Raspberry Pi[3], deve-se executar o comando sudo sh -c "iptables-save > /etc/iptables.ipv4.nat.

-Passo 12: Deve-se alterar o arquivo /etc/network/interfaces. Para isso, deve-se executar o comando sudo nano /etc/network/interfaces e adicionar a seguinte linha ao fim do arquivo:

```
up iptables-restore < /etc/iptables.ipv4.nat
```

- Passo 13: Deve-se atualizar o “hostpad”. Para isso deve-se fazer o download do arquivo de atualização através do seguinte comando: wget http://adafruit-download.s3.amazonaws.com/adafruit_hostapd_14128.zip.

- Passo 14: Descomprimir o arquivo de atualização do “hostpad” através do comando unzip adafruit_hostapd_14128.zip.

- Passo 15: Mover a versão antiga através do comando sudo mv /usr/sbin/hostapd /usr/sbin/hostapd.ORIG.

- Passo 16: Mover a versão nova através do comando sudo mv hostapd /usr/sbin.

- Passo 17: Atribuir permissão para o arquivo da versão nova através do comando sudo chmod 755 /usr/sbin/hostapd.

- Passo 18: Testar o funcionamento do “*Acess Point*[9]” através do comando sudo /usr/sbin/hostapd /etc/hostapd/hostapd.conf.

- Passo 19: Setar o serviço como um “deamon”, ou seja, o serviço é inicializado automaticamente quando o sistema é iniciado. Para isso, deve-se executar os seguintes comandos:

```
sudo service hostapd start
sudo service isc-dhcp-server start
```

```
sudo update-rc.d hostapd enable
sudo update-rc.d isc-dhcp-server enable
```

Apêndice B.6 – Configuração do Sistema de Arquivos no Carro

- Passo 1: Editar o arquivo /etc/hosts.deny para definir a segurança do compartilhamento. Para isso deve-se executar o comando: `sudo nano /etc/hosts.deny` e em seguida adicionar as seguintes linhas ao final do arquivo:

```
portmap: ALL
lockd: ALL
mountd: ALL
rquotad: ALL
```

- Passo 2: Editar o arquivo /etc/hosts.allow para definir o IP ou faixa de IPs que poderão "concorrer" aos compartilhamentos, ou seja, especifica quem terá acesso a cada serviço. Para isso devemos executar o comando: `sudo nano /etc/hosts.allow` e em seguida adicionar as seguintes linhas ao final do arquivo:

```
portmap: 192.168.42.2
lockd: 192.168.42.2
rquotad: 192.168.42.2
mountd: 192.168.42.2
statd: 192.168.42.2
```

- Passo 3: Editar o arquivo /etc/fstab para definir montagem automática das pastas compartilhadas. Para isso deve-se executar o comando `sudo nano /etc/fstab` e em seguida adicionar a seguinte linha ao arquivo:

```
# Caminho do servidor   Ponto de montagem  Tipo-FS Opções
192.168.42.2:/usr/pi/motion_videos_client /usr/pi/motion_videos nfs rw 0 0
```

Apêndice C – Arquivos de configuração “Motion”

Código C.1 – Arquivo motion.conf

```
# Rename this distribution example file to motion.conf
#
# This config file was generated by motion 3.2.12



############################################################
# Daemon
############################################################

# Start in daemon (background) mode and release terminal (default: off)
daemon on

# File to store the process ID, also called pid file. (default: not defined)
process_id_file /var/run/motion.pid

############################################################
# Basic Setup Mode
############################################################

# Start in Setup-Mode, daemon disabled. (default: off)
setup_mode off

############################################################
# Capture device options
############################################################

# Videodevice to be used for capturing  (default /dev/video0)
# for FreeBSD default is /dev/bktr0
videodevice /dev/video0

# v4l2_palette allows to choose preferable palette to be use by motion
# to capture from those supported by your videodevice. (default: 8)
# E.g. if your videodevice supports both V4L2_PIX_FMT_SBGGR8 and
# V4L2_PIX_FMT_MJPEG then motion will by default use V4L2_PIX_FMT_MJPEG.
```

```
# Setting v4l2_palette to 1 forces motion to use V4L2_PIX_FMT_SBGGR8
# instead.
#
# Values :
# V4L2_PIX_FMT_SN9C10X : 0  'S910'
# V4L2_PIX_FMT_SBGGR8  : 1  'BA81'
# V4L2_PIX_FMT_MJPEG   : 2  'MJPEG'
# V4L2_PIX_FMT_JPEG    : 3  'JPEG'
# V4L2_PIX_FMT_RGB24   : 4  'RGB3'
# V4L2_PIX_FMT_UYVY    : 5  'UYVY'
# V4L2_PIX_FMT_YUYV    : 6  'YUYV'
# V4L2_PIX_FMT_YUV422P : 7  '422P'
# V4L2_PIX_FMT_YUV420  : 8  'YU12'
v4l2_palette 8

# Tuner device to be used for capturing using tuner as source (default /dev/tuner0)
# This is ONLY used for FreeBSD. Leave it commented out for Linux
; tunerdevice /dev/tuner0

# The video input to be used (default: 8)
# Should normally be set to 0 or 1 for video/TV cards, and 8 for USB cameras
input 8

# The video norm to use (only for video capture and TV tuner cards)
# Values: 0 (PAL), 1 (NTSC), 2 (SECAM), 3 (PAL NC no colour). Default: 0 (PAL)
norm 0

# The frequency to set the tuner to (kHz) (only for TV tuner cards) (default: 0)
frequency 0

# Rotate image this number of degrees. The rotation affects all saved images as
# well as mpeg movies. Valid values: 0 (default = no rotation), 90, 180 and 270.
rotate 0

# Image width (pixels). Valid range: Camera dependent, default: 352
width 320
```

```
# Image height (pixels). Valid range: Camera dependent, default: 288
height 240

# Maximum number of frames to be captured per second.
# Valid range: 2-100. Default: 100 (almost no limit).
framerate 30

# Minimum time in seconds between capturing picture frames from the camera.
# Default: 0 = disabled - the capture rate is given by the camera framerate.
# This option is used when you want to capture images at a rate lower than 2 per second.
minimum_frame_time 0

# URL to use if you are using a network camera, size will be autodetected (incl http:// ftp:// or file:///)
# Must be a URL that returns single jpeg pictures or a raw mjpeg stream. Default: Not defined
; netcam_url value

# Username and password for network camera (only if required). Default: not defined
# Syntax is user:password
; netcam_userpass value

# The setting for keep-alive of network socket, should improve performance on compatible net cameras.
# 1.0:        The historical implementation using HTTP/1.0, closing the socket after each http request.
# keep_alive:  Use HTTP/1.0 requests with keep alive header to reuse the same connection.
# 1.1:        Use HTTP/1.1 requests that support keep alive as default.
# Default: 1.0
; netcam_http 1.0

# URL to use for a netcam proxy server, if required, e.g. "http://myproxy".
# If a port number other than 80 is needed, use "http://myproxy:1234".
# Default: not defined
; netcam_proxy value
```

```
# Set less strict jpeg checks for network cameras with a poor/buggy firmware.
# Default: off
netcam_tolerant_check off

# Let motion regulate the brightness of a video device (default: off).
# The auto_brightness feature uses the brightness option as its target value.
# If brightness is zero auto_brightness will adjust to average brightness value 128.
# Only recommended for cameras without auto brightness
auto_brightness off

# Set the initial brightness of a video device.
# If auto_brightness is enabled, this value defines the average brightness level
# which Motion will try and adjust to.
# Valid range 0-255, default 0 = disabled
brightness 0

# Set the contrast of a video device.
# Valid range 0-255, default 0 = disabled
contrast 0

# Set the saturation of a video device.
# Valid range 0-255, default 0 = disabled
saturation 0

# Set the hue of a video device (NTSC feature).
# Valid range 0-255, default 0 = disabled
hue 0


############################################################
# Round Robin (multiple inputs on same video device name)
############################################################

# Number of frames to capture in each roundrobin step (default: 1)
roundrobin_frames 1
```

```
# Number of frames to skip before each roundrobin step (default: 1)
roundrobin_skip 1

# Try to filter out noise generated by roundrobin (default: off)
switchfilter off



##############################################################
# Motion Detection Settings:
##############################################################

# Threshold for number of changed pixels in an image that
# triggers motion detection (default: 1500)
threshold 10

# Automatically tune the threshold down if possible (default: off)
threshold_tune off

# Noise threshold for the motion detection (default: 32)
noise_level 2

# Automatically tune the noise threshold (default: on)
noise_tune on

# Despeckle motion image using (e)rode or (d)ilate or (l)abel (Default: not defined)
# Recommended value is EedDl. Any combination (and number of) of E, e, d, and D is valid.
# (l)abeling must only be used once and the 'l' must be the last letter.
# Comment out to disable
despeckle EedDl

# Detect motion in predefined areas (1 - 9). Areas are numbered like that:  1 2 3
# A script (on_area_detected) is started immediately when motion is         4 5 6
# detected in one of the given areas, but only once during an event.        7 8 9
# One or more areas can be specified with this option. (Default: not defined)
; area_detect value
```

```
# PGM file to use as a sensitivity mask.
# Full path name to. (Default: not defined)
; mask_file value

# Dynamically create a mask file during operation (default: 0)
# Adjust speed of mask changes from 0 (off) to 10 (fast)
smart_mask_speed 0

# Ignore sudden massive light intensity changes given as a percentage of the picture
# area that changed intensity. Valid range: 0 - 100 , default: 0 = disabled
lightswitch 0

# Picture frames must contain motion at least the specified number of frames
# in a row before they are detected as true motion. At the default of 1, all
# motion is detected. Valid range: 1 to thousands, recommended 1-5
minimum_motion_frames 1

# Specifies the number of pre-captured (buffered) pictures from before motion
# was detected that will be output at motion detection.
# Recommended range: 0 to 5 (default: 0)
# Do not use large values! Large values will cause Motion to skip video frames and
# cause unsmooth mpegs. To smooth mpegs use larger values of post_capture instead.
pre_capture 0

# Number of frames to capture after motion is no longer detected (default: 0)
post_capture 0

# Gap is the seconds of no motion detection that triggers the end of an event
# An event is defined as a series of motion images taken within a short timeframe.
# Recommended value is 60 seconds (Default). The value 0 is allowed and disables
# events causing all Motion to be written to one single mpeg file and no pre_capture.
gap 60

# Maximum length in seconds of an mpeg movie
# When value is exceeded a new mpeg file is created. (Default: 0 = infinite)
```

```
max_mpeg_time 0

# Always save images even if there was no motion (default: off)
output_all off



############################################################
# Image File Output
############################################################

# Output 'normal' pictures when motion is detected (default: on)
# Valid values: on, off, first, best, center
# When set to 'first', only the first picture of an event is saved.
# Picture with most motion of an event is saved when set to 'best'.
# Picture with motion nearest center of picture is saved when set to 'center'.
# Can be used as preview shot for the corresponding movie.
output_normal off

# Output pictures with only the pixels moving object (ghost images) (default: off)
output_motion off

# The quality (in percent) to be used by the jpeg compression (default: 75)
quality 75

# Output ppm images instead of jpeg (default: off)
ppm off



############################################################
# FFMPEG related options
# Film (mpeg) file output, and deinterlacing of the video input
# The options movie_filename and timelapse_filename are also used
# by the ffmpeg feature
############################################################

# Use ffmpeg to encode mpeg movies in realtime (default: off)
```

```
ffmpeg_cap_new on

# Use ffmpeg to make movies with only the pixels moving
# object (ghost images) (default: off)
ffmpeg_cap_motion off

# Use ffmpeg to encode a timelapse movie
# Default value 0 = off - else save frame every Nth second
ffmpeg_timelapse 0

# The file rollover mode of the timelapse video
# Valid values: hourly, daily (default), weekly-sunday, weekly-monday, monthly, manual
ffmpeg_timelapse_mode daily

# Bitrate to be used by the ffmpeg encoder (default: 400000)
# This option is ignored if ffmpeg_variable_bitrate is not 0 (disabled)
ffmpeg_bps 500000

# Enables and defines variable bitrate for the ffmpeg encoder.
# ffmpeg_bps is ignored if variable bitrate is enabled.
# Valid values: 0 (default) = fixed bitrate defined by ffmpeg_bps,
# or the range 2 - 31 where 2 means best quality and 31 is worst.
ffmpeg_variable_bitrate 0

# Codec to used by ffmpeg for the video compression.
# Timelapse mpegs are always made in mpeg1 format independent from this option.
# Supported formats are: mpeg1 (ffmpeg-0.4.8 only), mpeg4 (default), and msmpeg4.
# mpeg1 - gives you files with extension .mpg
# mpeg4 or msmpeg4 - gives you files with extension .avi
# msmpeg4 is recommended for use with Windows Media Player because
# it requires no installation of codec on the Windows client.
# swf - gives you a flash film with extension .swf
# flv - gives you a flash video with extension .flv
# ffv1 - FF video codec 1 for Lossless Encoding ( experimental )
# mov - QuickTime ( testing )
ffmpeg_video_codec mpeg4
```

```
# Use ffmpeg to deinterlace video. Necessary if you use an analog camera
# and see horizontal combing on moving objects in video or pictures.
# (default: off)
ffmpeg_deinterlace off


############################################################
# Snapshots (Traditional Periodic Webcam File Output)
############################################################

# Make automated snapshot every N seconds (default: 0 = disabled)
snapshot_interval 0


############################################################
# Text Display
# %Y = year, %m = month, %d = date,
# %H = hour, %M = minute, %S = second, %T = HH:MM:SS,
# %v = event, %q = frame number, %t = thread (camera) number,
# %D = changed pixels, %N = noise level, \n = new line,
# %i and %J = width and height of motion area,
# %K and %L = X and Y coordinates of motion center
# %C = value defined by text_event - do not use with text_event!
# You can put quotation marks around the text to allow
# leading spaces
############################################################

# Locate and draw a box around the moving object.
# Valid values: on, off and preview (default: off)
# Set to 'preview' will only draw a box in preview_shot pictures.
locate off

# Draws the timestamp using same options as C function strftime(3)
# Default: %Y-%m-%d\n%T = date in ISO format and time in 24 hour clock
# Text is placed in lower right corner
```

```
text_right %Y-%m-%d\n%T-%q

# Draw a user defined text on the images using same options as C function strftime(3)
# Default: Not defined = no text
# Text is placed in lower left corner
; text_left CAMERA %t

# Draw the number of changed pixed on the images (default: off)
# Will normally be set to off except when you setup and adjust the motion settings
# Text is placed in upper right corner
text_changes off

# This option defines the value of the special event conversion specifier %C
# You can use any conversion specifier in this option except %C. Date and time
# values are from the timestamp of the first image in the current event.
# Default: %Y%m%d%H%M%S
# The idea is that %C can be used filenames and text_left/right for creating
# a unique identifier for each event.
text_event %Y%m%d%H%M%S

# Draw characters at twice normal size on images. (default: off)
text_double off



##############################################################
# Target Directories and filenames For Images And Films
# For the options snapshot_, jpeg_, mpeg_ and timelapse_filename
# you can use conversion specifiers
# %Y = year, %m = month, %d = date,
# %H = hour, %M = minute, %S = second,
# %v = event, %q = frame number, %t = thread (camera) number,
# %D = changed pixels, %N = noise level,
# %i and %J = width and height of motion area,
# %K and %L = X and Y coordinates of motion center
# %C = value defined by text_event
# Quotation marks round string are allowed.
```

```
############################################################

# Target base directory for pictures and films
# Recommended to use absolute path. (Default: current working directory)
target_dir /home/pi/motion_videos

# File path for snapshots (jpeg or ppm) relative to target_dir
# Default: %v-%Y%m%d%H%M%S-snapshot
# Default value is equivalent to legacy oldlayout option
# For Motion 3.0 compatible mode choose: %Y/%m/%d/%H/%M/%S-snapshot
# File extension .jpg or .ppm is automatically added so do not include this.
# Note: A symbolic link called lastsnap.jpg created in the target_dir will always
# point to the latest snapshot, unless snapshot_filename is exactly 'lastsnap'
snapshot_filename %v-%Y%m%d%H%M%S-snapshot

# File path for motion triggered images (jpeg or ppm) relative to target_dir
# Default: %v-%Y%m%d%H%M%S-%q
# Default value is equivalent to legacy oldlayout option
# For Motion 3.0 compatible mode choose: %Y/%m/%d/%H/%M/%S-%q
# File extension .jpg or .ppm is automatically added so do not include this
# Set to 'preview' together with best-preview feature enables special naming
# convention for preview shots. See motion guide for details
jpeg_filename %v-%Y%m%d%H%M%S-%q

# File path for motion triggered ffmpeg films (mpeg) relative to target_dir
# Default: %v-%Y%m%d%H%M%S
# Default value is equivalent to legacy oldlayout option
# For Motion 3.0 compatible mode choose: %Y/%m/%d/%H%M%S
# File extension .mpg or .avi is automatically added so do not include this
# This option was previously called ffmpeg_filename
movie_filename %v-%Y%m%d%H%M%S

# File path for timelapse mpegs relative to target_dir
# Default: %Y%m%d-timelapse
# Default value is near equivalent to legacy oldlayout option
# For Motion 3.0 compatible mode choose: %Y/%m/%d-timelapse
```

```
# File extension .mpg is automatically added so do not include this
timelapse_filename %Y%m%d-timelapse


############################################################
# Live Webcam Server
############################################################

# The mini-http server listens to this port for requests (default: 0 = disabled)
webcam_port 2015

# Quality of the jpeg (in percent) images produced (default: 50)
webcam_quality 50

# Output frames at 1 fps when no motion is detected and increase to the
# rate given by webcam_maxrate when motion is detected (default: off)
webcam_motion off

# Maximum framerate for webcam streams (default: 1)
webcam_maxrate 1

# Restrict webcam connections to localhost only (default: on)
webcam_localhost off

# Limits the number of images per connection (default: 0 = unlimited)
# Number can be defined by multiplying actual webcam rate by desired number of seconds
# Actual webcam rate is the smallest of the numbers framerate and webcam_maxrate
webcam_limit 0


############################################################
# HTTP Based Control
############################################################

# TCP/IP port for the http server to listen on (default: 0 = disabled)
control_port 8080
```

```
# Restrict control connections to localhost only (default: on)
control_localhost on

# Output for http server, select off to choose raw text plain (default: on)
control_html_output on

# Authentication for the http based control. Syntax username:password
# Default: not defined (Disabled)
; control_authentication username:password



############################################################
# Tracking (Pan/Tilt)
############################################################

# Type of tracker (0=none (default), 1=stepper, 2=iomojo, 3=pwc, 4=generic, 5=uvcvideo)
# The generic type enables the definition of motion center and motion size to
# be used with the conversion specifiers for options like on_motion_detected
track_type 0

# Enable auto tracking (default: off)
track_auto off

# Serial port of motor (default: none)
; track_port value

# Motor number for x-axis (default: 0)
track_motorx 0

# Motor number for y-axis (default: 0)
track_motory 0

# Maximum value on x-axis (default: 0)
track_maxx 0
```

```
# Maximum value on y-axis (default: 0)
track_maxy 0

# ID of an iomojo camera if used (default: 0)
track_iomojo_id 0

# Angle in degrees the camera moves per step on the X-axis
# with auto-track (default: 10)
# Currently only used with pwc type cameras
track_step_angle_x 10

# Angle in degrees the camera moves per step on the Y-axis
# with auto-track (default: 10)
# Currently only used with pwc type cameras
track_step_angle_y 10

# Delay to wait for after tracking movement as number
# of picture frames (default: 10)
track_move_wait 10

# Speed to set the motor to (stepper motor option) (default: 255)
track_speed 255

# Number of steps to make (stepper motor option) (default: 40)
track_stepsize 40



##############################################################
# External Commands, Warnings and Logging:
# You can use conversion specifiers for the on_xxxx commands
# %Y = year, %m = month, %d = date,
# %H = hour, %M = minute, %S = second,
# %v = event, %q = frame number, %t = thread (camera) number,
# %D = changed pixels, %N = noise level,
# %i and %J = width and height of motion area,
# %K and %L = X and Y coordinates of motion center
```

```
# %C = value defined by text_event
# %f = filename with full path
# %n = number indicating filetype
# Both %f and %n are only defined for on_picture_save,
# on_movie_start and on_movie_end
# Quotation marks round string are allowed.
############################################################

# Do not sound beeps when detecting motion (default: on)
# Note: Motion never beeps when running in daemon mode.
quiet on

# Command to be executed when an event starts. (default: none)
# An event starts at first motion detected after a period of no motion defined by gap
; on_event_start value

# Command to be executed when an event ends after a period of no motion
# (default: none). The period of no motion is defined by option gap.
; on_event_end value

# Command to be executed when a picture (.ppm|.jpg) is saved (default: none)
# To give the filename as an argument to a command append it with %f
; on_picture_save value

# Command to be executed when a motion frame is detected (default: none)
; on_motion_detected value

# Command to be executed when motion in a predefined area is detected
# Check option 'area_detect'.   (default: none)
; on_area_detected value

# Command to be executed when a movie file (.mpg|.avi) is created. (default: none)
# To give the filename as an argument to a command append it with %f
; on_movie_start value

# Command to be executed when a movie file (.mpg|.avi) is closed. (default: none)
```

```
# To give the filename as an argument to a command append it with %f
; on_movie_end value

# Command to be executed when a camera can't be opened or if it is lost
# NOTE: There is situations when motion doesn't detect a lost camera!
# It depends on the driver, some drivers don't detect a lost camera at all
# Some hang the motion thread. Some even hang the PC! (default: none)
; on_camera_lost value

############################################################
# Common Options For MySQL and PostgreSQL database features.
# Options require the MySQL/PostgreSQL options to be active also.
############################################################

# Log to the database when creating motion triggered image file  (default: on)
sql_log_image on

# Log to the database when creating a snapshot image file (default: on)
sql_log_snapshot on

# Log to the database when creating motion triggered mpeg file (default: off)
sql_log_mpeg off

# Log to the database when creating timelapse mpeg file (default: off)
sql_log_timelapse off

# SQL query string that is sent to the database
# Use same conversion specifiers has for text features
# Additional special conversion specifiers are
# %n = the number representing the file_type
# %f = filename with full path
# Default value:
# insert into security(camera, filename, frame, file_type, time_stamp, text_event) values('%t',
'%f', '%q', '%n', '%Y-%m-%d %T', '%C')
sql_query insert into security(camera, filename, frame, file_type, time_stamp,
event_time_stamp) values('%t', '%f', '%q', '%n', '%Y-%m-%d %T', '%C')
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
# Database Options For MySQL
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

# Mysql database to log to (default: not defined)
; mysql_db value

# The host on which the database is located (default: localhost)
; mysql_host value

# User account name for MySQL database (default: not defined)
; mysql_user value

# User password for MySQL database (default: not defined)
; mysql_password value


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
# Database Options For PostgreSQL
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

# PostgreSQL database to log to (default: not defined)
; pgsql_db value

# The host on which the database is located (default: localhost)
; pgsql_host value

# User account name for PostgreSQL database (default: not defined)
; pgsql_user value

# User password for PostgreSQL database (default: not defined)
; pgsql_password value

# Port on which the PostgreSQL database is located (default: 5432)
```

```
; pgsql_port 5432


############################################################
# Video Loopback Device (vloopback project)
############################################################

# Output images to a video4linux loopback device
# The value '-' means next available (default: not defined)
; video_pipe value

# Output motion images to a video4linux loopback device
# The value '-' means next available (default: not defined)
; motion_video_pipe value


############################################################
# Thread config files - One for each camera.
# Except if only one camera - You only need this config file.
# If you have more than one camera you MUST define one thread
# config file for each camera in addition to this config file.
############################################################

# Remember: If you have more than one camera you must have one
# thread file for each camera. E.g. 2 cameras requires 3 files:
# This motion.conf file AND thread1.conf and thread2.conf.
# Only put the options that are unique to each camera in the
# thread config files.
thread /etc/motion/thread1.conf
thread /etc/motion/thread2.conf
; thread /usr/local/etc/thread3.conf
; thread /usr/local/etc/thread4.conf
```

Código C.2 – Arquivo thread1.conf

```
# /usr/local/etc/thread1.conf
#
# This config file was generated by motion 3.2.12



############################################################
# Capture device options
############################################################

# Videodevice to be used for capturing  (default /dev/video0)
# for FreeBSD default is /dev/bktr0
videodevice /dev/video0

# The video input to be used (default: 8)
# Should normally be set to 1 for video/TV cards, and 8 for USB cameras
input 8

# Draw a user defined text on the images using same options as C function strftime(3)
# Default: Not defined = no text
# Text is placed in lower left corner
text_left CAMERA 1



############################################################
# Target Directories and filenames For Images And Films
# For the options snapshot_, jpeg_, mpeg_ and timelapse_filename
# you can use conversion specifiers
# %Y = year, %m = month, %d = date,
# %H = hour, %M = minute, %S = second,
# %v = event, %q = frame number, %t = thread (camera) number,
# %D = changed pixels, %N = noise level,
# %i and %J = width and height of motion area,
# %K and %L = X and Y coordinates of motion center
# %C = value defined by text_event
# Quotation marks round string are allowed.
```

```
############################################################

# Target base directory for pictures and films
# Recommended to use absolute patch. (Default: current working directory)
#target_dir /usr/local/apache2/htdocs/cam1
target_dir /home/pi/motion_videos

############################################################
# Live Webcam Server
############################################################

# The mini-http server listens to this port for requests (default: 0 = disabled)
#webcam_port 8081

# Command to be executed when a picture (.ppm|.jpg) is saved (default: none)
# The filename of the picture is appended as an argument for the command.
#on_picture_save /usr/local/motion-extras/camparse1.pl

# Command to be executed when a movie file (.mpg|.avi) is closed. (default: none)
# Filename of movie is appended as an argument for the command.
#on_movie_end /usr/local/motion-extras/mpegparse1.pl
```

Código C.3 – Arquivo thread2.conf

```
# /usr/local/etc/thread2.conf
#
# This config file was generated by motion 3.2.12



############################################################
# Capture device options
############################################################

# Videodevice to be used for capturing  (default /dev/video0)
# for FreeBSD default is /dev/bktr0
videodevice /dev/video0

# The video input to be used (default: 8)
# Should normally be set to 1 for video/TV cards, and 8 for USB cameras
input 8

# Draw a user defined text on the images using same options as C function strftime(3)
# Default: Not defined = no text
# Text is placed in lower left corner
text_left CAMERA 1



############################################################
# Target Directories and filenames For Images And Films
# For the options snapshot_, jpeg_, mpeg_ and timelapse_filename
# you can use conversion specifiers
# %Y = year, %m = month, %d = date,
# %H = hour, %M = minute, %S = second,
# %v = event, %q = frame number, %t = thread (camera) number,
# %D = changed pixels, %N = noise level,
# %i and %J = width and height of motion area,
# %K and %L = X and Y coordinates of motion center
# %C = value defined by text_event
```

```
# Quotation marks round string are allowed.
############################################################

# Target base directory for pictures and films
# Recommended to use absolute patch. (Default: current working directory)
target_dir /home/pi/motion_videos_client



############################################################
# Live Webcam Server
############################################################

# The mini-http server listens to this port for requests (default: 0 = disabled)
#webcam_port 8082

# Command to be executed when a picture (.ppm|.jpg) is saved (default: none)
# The filename of the picture is appended as an argument for the command.
#on_picture_save /usr/local/motion-extras/camparse2.pl

# Command to be executed when a movie file (.mpg|.avi) is closed. (default: none)
# Filename of movie is appended as an argument for the command.
#on_movie_end /usr/local/motion-extras/mpegparse2.pl
```