FACULDADE DE TECNOLOGIA DE SÃO BERNARDO DO CAMPO
“ADIB MOISÉS DIB”

ALEXANDRE NOBORU FUKUDA

SISTEMA DE ORIENTAÇÃO INERCIAL PARA O ROBÔ MÓVEL AUTÔNOMO EMMY III

São Bernardo do Campo – SP
Junho/2018

ALEXANDRE NOBORU FUKUDA

SISTEMA DE ORIENTAÇÃO INERCIAL PARA O ROBÔ MÓVEL AUTÔNOMO EMMY III

Trabalho de conclusão de curso apresentado à Faculdade de Tecnologia de São Bernardo do Campo “Adib Moisés Dib” como requisito parcial para a obtenção do título de tecnólogo em Automação Industrial.

Orientador: Prof. Dr Cláudio Rodrigo Torres.

São Bernardo do Campo – SP
Junho/2018

ALEXANDRE NOBORU FUKUDA

SISTEMA DE ORIENTAÇÃO INERCIAL PARA O ROBÔ MÓVEL AUTÔNOMO EMMY III

Trabalho de conclusão de curso apresentado à Faculdade de Tecnologia de São Bernardo do Campo “Adib Moisés Dib” como requisito parcial para a obtenção do título de tecnólogo em Automação Industrial.

Orientador: Prof. Dr Cláudio Rodrigo Torres.

Trabalho de Conclusão de Curso apresentado e aprovado em: / / 2018.

Banca examinadora:

_____

Prof. Dr. Cláudio Rodrigo Torres, FATEC SBC – Orientador

_____

Prof.

_____

Prof.

Agradeço aos professores: Dr. Wellington Batista de Sousa pela iniciativa e apoio na implantação da Iniciação Científica na Fatec SBC; ao Dr. Delcínio Ricci pelas dicas em relação a monografia; e ao meu orientador Dr. Cláudio Rodrigo Torres que, através da sua orientação, tornou-se possível a realização deste trabalho.

Agradeço ao meu irmão Minoru, pela ajuda e apoio dado durante todo o meu curso de graduação.

“Um robô não pode ferir um ser humano ou permanecer passivo deixando um ser humano exposto ao perigo.”

ISAAC ASIMOV

RESUMO

Este projeto propõe o desenvolvimento de um algoritmo de controle de movimentos para um robô móvel autônomo. Esse algoritmo trabalha em conjunto com um sensor inercial de medição rotacional, ou seja, um giroscópio, que fornece dados de velocidade angular. Apresenta-se ao longo da dissertação os conceitos teóricos sobre localização espacial, os sensores inerciais, um breve histórico da robótica móvel, destacando-se o robô móvel autônomo Emmy, robô desenvolvido no Brasil, baseado nos conceitos da lógica paraconsistente. O objetivo principal deste projeto é movimentar uma plataforma mecânica móvel, deslocando-se de uma posição inicial ao destino desejado num ambiente sem obstáculos. Através de um dispositivo móvel, fornecemos os dados das coordenadas das posições ao robô. O processamento é executado por um microcontrolador. Esta monografia apresenta todas as etapas do desenvolvimento do projeto, desde o algoritmo de controle, passando pela montagem da plataforma mecânica e do circuito eletrônico, até os testes de funcionamento do protótipo.

Palavras-chave: Robótica. Controle. Paraconsistente. Microcontrolador. Sensores.

ABSTRACT

This project proposes the development of a motion control algorithm for an autonomous mobile robot. This algorithm works in conjunction with an inertial rotational measurement sensor, ie a gyroscope, which provides angular velocity data. Theoretical concepts on spatial location, inertial sensors, a brief history of mobile robotics, stand out the autonomous mobile robot Emmy, a robot developed in Brazil, based on the concepts of paraconsistent logic. The main objective of this project is to move a mobile mechanical platform, moving from an initial position to the desired destination in an environment without obstacles. Through a mobile device, we provide the position coordinate data to the robot. Processing is performed by a microcontroller. This monograph presents all the steps of the project development, from the control algorithm, through the assembly of the mechanical platform and the electronic circuit,to the prototype tests.

Keywords: Robotics. Control. Paraconsistent. Microcontroller. Sensors.

SUMÁRIO

INTRODUÇÃO

A aplicação dos chamados robôs móveis vem ganhando cada vez mais espaço em diversos segmentos da indústria. Por exemplo, ela abrange um vasto campo de aplicações, desde robôs que ajudam na localização e desativação de artefatos explosivos, patrulhamento de áreas, exploração submarina ou espacial, até a inspeção da integridade de tubulações em instalações perigosas como ambientes que possuam calor excessivo, gases tóxicos ou radioatividade.

Contudo, tais aplicações exigem uma perfeita integração entre sensores, atuadores e sistema de controle, de maneira a permitir uma navegação segura através de comandos à distância e/ou de forma autônoma.

Um robô móvel autônomo deve ser capaz de se locomover num ambiente estruturado ou não. Dessa forma são necessários para a realização dos movimentos, dispositivos que forneçam dados de referência e posicionamento no espaço, dentre os quais incluem os sensores inerciais.

Através do desenvolvimento da microeletromecânica, aliado à produção em larga escala, os sensores inerciais atuais possuem tamanhos reduzidos, precisão e boa relação custo-benefício. Eles estão presentes numa gama de produtos que vão de telefones móveis a equipamentos aeroespaciais.

Diante desse fato, o objetivo principal do presente tema é o desenvolvimento de um Sistema de Orientação Inercial para o Robô Móvel Autônomo Emmy III, robô baseado nos conceitos da lógica paraconsistente. Este projeto abrange somente a parte do sistema de controle dos movimentos, baseados em sensores inerciais. O projeto compõe-se de um circuito eletrônico com um microcontrolador, sensores inerciais (acelerômetro e giroscópio), dispositivo de entrada de dados, monitor de cristal líquido, e por um algoritmo de controle implementado no microcontrolador. O sistema de controle proposto abrange as seguintes funções:

- recebimento dos dados de posições iniciais e finais fornecidos pelo usuário, baseado num ambiente definido por um plano cartesiano;
- processamento da trajetória, gerando uma sequência de códigos de movimentos;
- coleta dos dados angulares dos sensores inerciais;
- comando dos motores para a execução dos movimentos numa plataforma robótica móvel.

Para os testes de validação do sistema, está previsto o uso de uma plataforma robótica móvel, composto por motores, circuitos de potência, baterias, e rodas.

Este trabalho compõe-se de três capítulos:

Capítulo 1 Fundamentação teórica: apresenta as bases teóricas para o desenvolvimento do projeto.

Capítulo 2 Metodologia: descreve o caminho percorrido para a elaboração do projeto.

Capítulo 3 Desenvolvimento do projeto: descreve o desenvolvimento da construção do projeto.

E finalmente, as Considerações finais: descreve, em linhas gerais, objetivo e justificativa, conquistas alcançadas, pontos fortes e fracos, relações entre os fatos verificados e as teorias e possíveis sugestões para estudos futuros.

1 FUNDAMENTAÇÃO TEÓRICA

Este capítulo apresenta as teorias que sustentam o desenvolvimento do projeto Sistema de Orientação Inercial para o Robô Móvel Autônomo Emmy III.

1.1 Navegação inercial

Esta seção apresenta os conceitos de um sistema de orientação inercial.

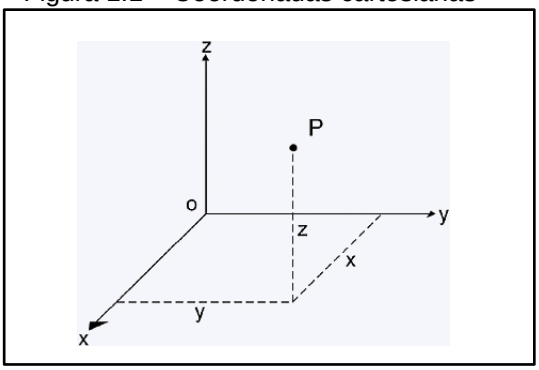
Matarić (2014) aponta que o termo navegação refere-se ao ato de um veículo se movimentar, de uma posição inicial até um destino, através de um planejamento do trajeto.

De acordo com Mori (2013), um sistema de navegação inercial é um conjunto que fornece medidas de movimentos lineares e/ou angulares através do processamento dos dados de um ou mais dispositivos inerciais. Esse sistema denomina-se INS (*Inertial Navigation System* – Sistema de navegação inercial).

1.1.1 Localização espacial e graus de liberdade

Mori (2013) descreve que um corpo físico se localiza no espaço tridimensional (3D) através do sistema de coordenadas cartesianas de eixos (x,y,z), conforme mostra a Figura 1.1.

Figura 1.1 – Coordenadas cartesianas

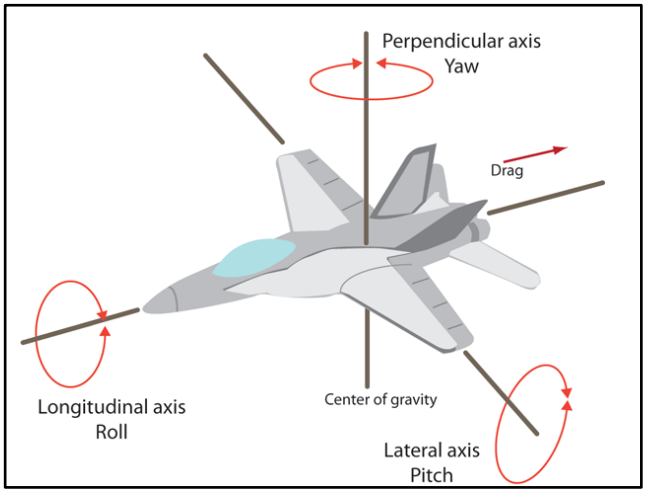


Fonte: www.cepa.if.usp.br, 2017

Matarić (2014) descreve *grau de liberdade* como o número de coordenadas necessárias para especificar o movimento de um sistema mecânico. No espaço tridimensional, tem-se no total seis graus de liberdade sendo:

- três de translação – movimentação ao longo dos eixos x,y,z, sem girar;
- três de rotação em torno dos eixos x,y, e z, e eles são chamados *rolagem, arfagem e guinada (roll, pitch, yaw),* Figura 1.2.

Figura 1.2 – Rotações em torno dos eixos de uma aeronave



Fonte: www.machinedesign.com, 2017

1.2 Sensores inerciais

Mori (2013) cita o uso de sensores inerciais num sistema de navegação inercial (INS). Fazem parte dessa classe de dispositivos os acelerômetros e os giroscópios. Eles são os responsáveis pelo fornecimento dos dados inerciais ao sistema. O conjunto de vários acelerômetros e giroscópios denomina-se IMU (*Inertial Measurement Unit* – Unidade de medição inercial).

1.2.1 Acelerômetros

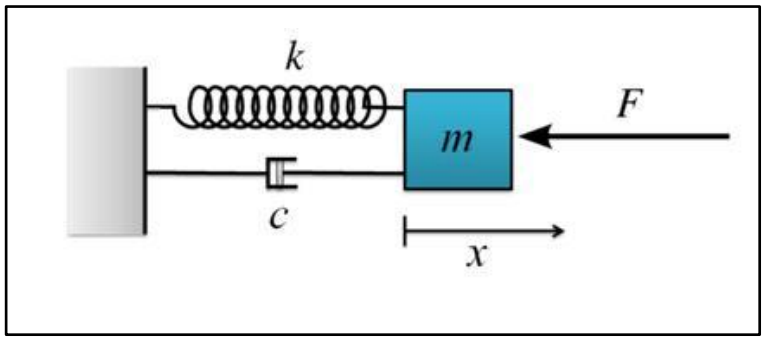
De acordo com Thomazini e Albuquerque (2007), os acelerômetros são dispositivos inerciais eletromecânicos que convertem a energia mecânica vinculada à aceleração de um corpo em sinais elétricos proporcionais, indicando a movimentação ao longo de um eixo de coordenadas. Dentre as aplicações do dispositivo citam: detecção de variações na velocidade de um corpo físico, medição da inclinação em máquinas e equipamentos, e o emprego nos sistemas de navegação de veículos aéreos, navais e espaciais.

Halliday, Resnick e Walker (2014) apontam a definição de força, segundo a Segunda Lei de Newton como:

$$\vec{F} = m \cdot \vec{a} \qquad [1.1]$$

A Figura 1.3 ilustra um modelo de acelerômetro mecânico tipo massa mola. O conjunto é composto por um corpo de massa *m,* afixado a uma base fixa através de uma mola e um sensor *c*, que registra variações de deslocamento.

Figura 1.3 – Acelerômetro tipo massa mola



Fonte: www.ni.com, 2017

Pela lei de Hooke, a força$\vec{F}$aplicada na mola é definida por:

$$\vec{F} = -k \cdot x \qquad [1.2]$$

onde *k* é a constante elástica da mola, e *x* é o deslocamento ao longo de um eixo (HALLIDAY; RESNICK; WALKER, 2014).

Relacionando as equações 1.1 e 1.2, tem-se*:*

$$x = a \cdot m / k \qquad [1.3]$$

ou seja, o deslocamento *x* corresponde ao produto da aceleração *a* e da massa *m* dividida pela constante elástica *k* (THOMAZINI; ALBUQUERQUE, 2007).

1.2.2 Giroscópios

Com o funcionamento análogo aos acelerômetros, os giroscópios são dispositivos inerciais que permitem medir a velocidade angular de um objeto em torno de um eixo de rotação (MORI, 2013). Os valores fornecidos são expressos em graus/s (INVENSENSE, 2013).

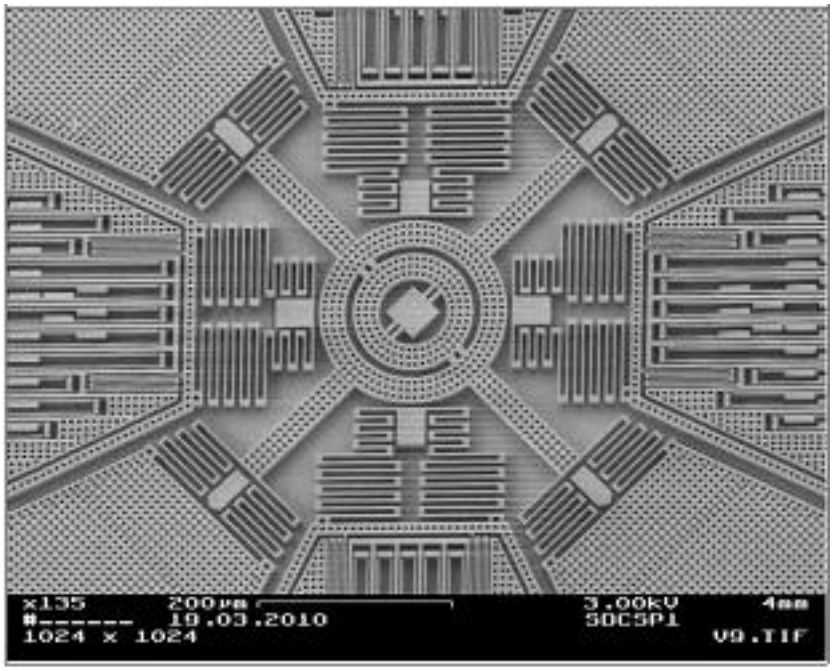
Segundo Halliday, Resnick e Walker (2014), a velocidade angular, representada por $\omega(t)$,resulta da derivada da posição angular θ em função do tempo. O inverso se obtém pela aplicação da integral da velocidade $\omega(t)$, ou seja:

$$\theta = \int \omega dt \qquad [1.4]$$

1.2.3 Dispositivos integrados

São dispositivos que integram o acelerômetro e o giroscópio num pequeno *chip* eletrônico. A tecnologia aplicada na fabricação desses sensores é denominada MEMS (*Micro Electro Mechanics Systems* – Sistemas Micro Eletromecânicos), que são dispositivos micromecânicos de silício, Figura 1.4 (MORI, 2013).

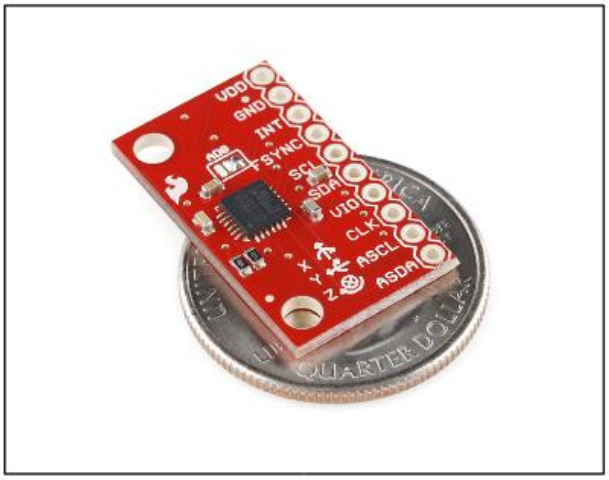
Figura 1.4 – Foto ampliada de um sensor MEMS



Fonte: www.community.arm.com, 2017

A Figura 1.5 apresenta um exemplo de dispositivo inercial integrado do fabricante InvenSense.

Figura 1.5 - *Chip* com acelerômetro e giroscópio InvenSense MPU 6050



Fonte: www.sparkfun.com, 2017

1.3 Robôs e robótica

Esta seção apresenta as definições de robô e robótica.

Rosário (2009) aponta que a origem da palavra “robô” vem do tcheco *robotnik*, que significa “servo”. Ela foi citada no início da década de 1920 por Karel Capek, dramaturgo de origem tcheco, na obra teatral de sua autoria *Robôs Universais de Rossum (RUR)*. Essa obra apresenta a criação de seres autômatos para executar serviços pesados, Figura 1.6.

Figura 1.6 – Cena da peça teatral *Robôs Universais de Rossum (RUR)*

Fonte: www.robohub.org, 2017

Atualmente, de acordo com Matarić (2014, p.19), define-se um “robô” como “[...] um sistema autônomo que existe no mundo físico, pode sentir o seu ambiente e pode agir sobre ele para alcançar alguns objetivos”.

De acordo com Rosário (2009), define-se a “robótica” como a área que, com o apoio das engenharias (mecânica e elétrica) e da tecnologia da informação, engloba o estudo e o desenvolvimento de robôs nas mais variadas aplicações.

1.4 Componentes de um robô

Esta seção apresenta os componentes de um robô.

Os principais componentes de um robô, segundo Matarić (2014), são:

- estrutura física;
- sensores;
- mecanismos efetuadores e dispositivos de atuação;
- sistema de controle ou controladores.

1.4.1 Estrutura física

Corresponde ao corpo do robô, e através dele o robô permite realizar tarefas interagindo com o mundo real, Figura 1.7.

Figura 1.7 – Estrutura física de um robô

Fonte: www.apod.nasa.gov, 2017

1.4.2 Sensores

Thomazini e Albuquerque (2007) descrevem os sensores como componentes com sensibilidade a formas energéticas do ambiente, convertendo-os para informações relacionadas a uma grandeza a ser medida (pressão, temperatura, etc.), Figura 1.8.

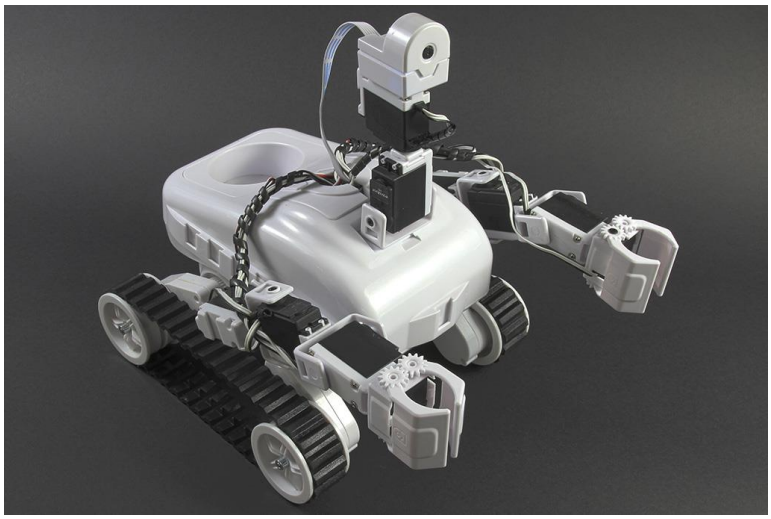
Figura 1.8 – Exemplo de sensores de proximidade

Fonte: www.automacao3r.com.br, 2017

1.4.3 Mecanismos efetuadores e dispositivos de atuação

De acordo com Matarić (2014), os efetuadores realizam tarefas físicas como movimentação (por rodas, esteiras) e manipulação de objetos (garras mecânicas). Os atuadores são dispositivos motores (elétricos, pneumáticos, hidráulicos) que executam o trabalho dos efetuadores, Figura 1.9.

Figura 1.9 – Robô com esteiras e garras mecânicas



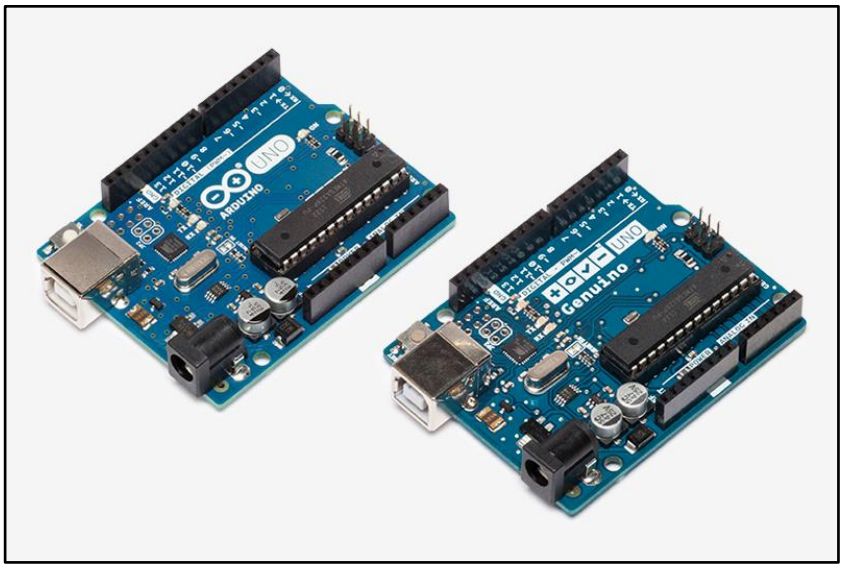
Fonte: www.ez-robot.com, 2017

1.4.4 Sistema de controle ou controladores

O sistema de controle ou controladores, descrito em Matarić (2014), é composto por dispositivos eletrônicos de processamento (computadores) e programas de controle. Ele tem como finalidade o comando do robô, recebendo informações dos sensores, tomando decisões e controlando os efetuadores/atuadores para executar ações desejadas de forma autônoma.

Um dispositivo de processamento muito utilizado nos controladores atuais é o microcontrolador. Williams (2015) define um microcontrolador como um computador completo num único *chip*, composto por uma unidade central de processamento, memória, comunicação serial e portas de entradas e/ou saídas, Figura 1.10.

Figura 1.10 – Placas Arduino com microcontrolador ATMEGA 328



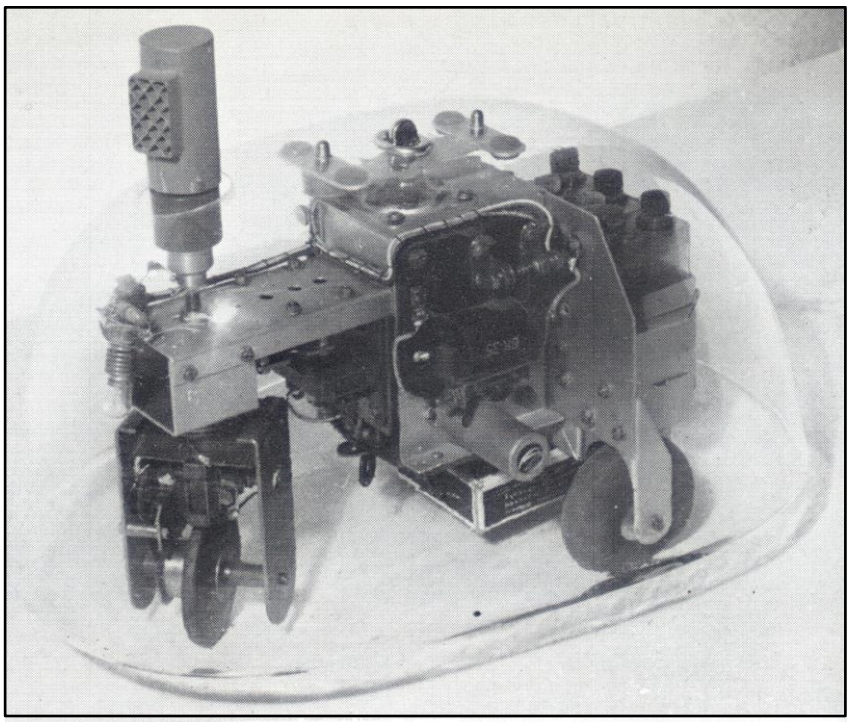
Fonte: www.arduino.cc, 2017

1.5 Robôs móveis

Esta seção apresenta os primeiros robôs móveis autônomos.

Matarić (2014) aponta que o primeiro robô autônomo foi desenvolvido na década de 1940, pelo pesquisador do campo da neurociência William Grey Walter (1910-1977). Ele construiu o robô para auxiliar no estudo do funcionamento dos sistemas cerebrais, fazendo uma correlação entre os sistemas biológicos e as máquinas. Esse robô era composto de uma plataforma mecânica com motores, rodas, bateria, sensor de contato, sensor fotoelétrico, e controlador eletrônico analógico. Possuía uma cobertura plástica no formato de uma tartaruga, e era controlado usando os princípios da cibernética[1], Figura 1.11.

Figura 1.11 – Robô de W. Grey Walter

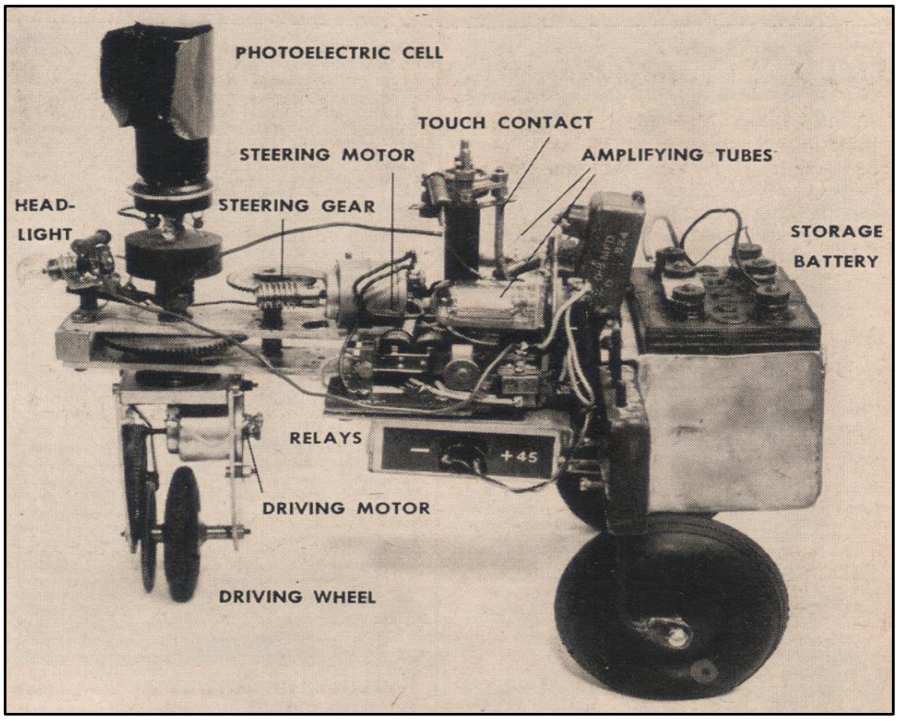


Fonte: www.cyberneticzoo.com, 2017

A Figura 1.12 apresenta em detalhes os componentes do robô de William Grey Walter.

[1] Cibernética – ciência que integra a neurociência, a biologia e engenharia. Do grego *kybernetes- "o que regula o movimento"* (MATARIĆ, 2014).

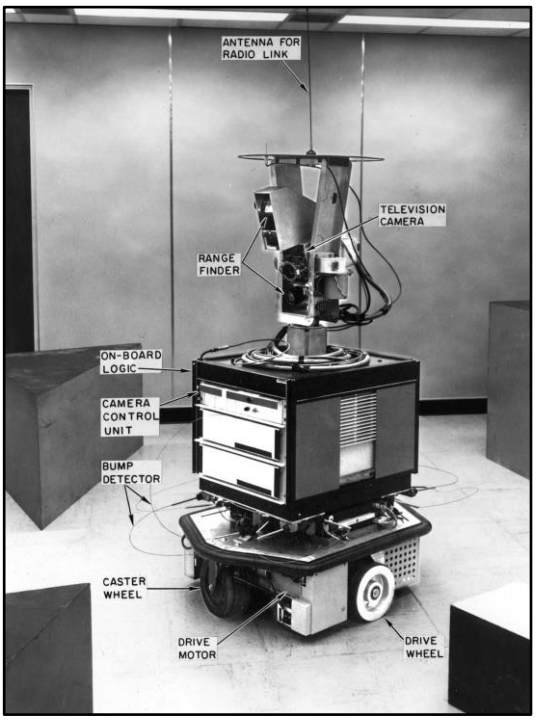
Figura 1.12 – Componentes do robô de W. Grey Walter



Fonte: www.cyberneticzoo.com, 2017

Com o advento da Inteligência Artificial (IA), foi desenvolvido no Stanford Research Institute (SRI), Califórnia (EUA), no final dos anos 1960, o robô *Shakey* (MATARIĆ, 2014), Figura 1.13.

Figura 1.13– O robô Shakey



Fonte: www.ai.sri.com/shakey, 2017

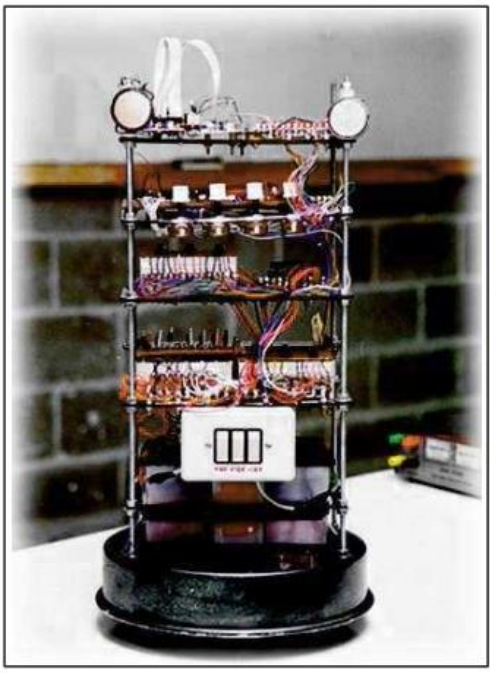
1.6 O robô autônomo Emmy

Torres (2010) destaca que há três versões sobre o robô móvel autônomo Emmy, sendo eles I, II e III, cujo sistema de controle tem como base a lógica paraconsistente[2] anotada evidencial Eт.

1.6.1 Robô Emmy I

Da Silva Filho; Torres e Abe (2006) destacam que o Emmy I foi o primeiro robô do mundo a funcionar com um controlador lógico paraconsistente. É um robô móvel autônomo capaz de desviar de obstáculos em um ambiente não estruturado. É constituído de uma estrutura de alumínio de formato circular de 30 cm de diâmetro e 60 cm de altura, projetado em módulos sobrepostos separados por função de cada um deles no sistema de controle, conforme ilustra a Figura 1.14, (TORRES, 2010).

Figura 1.14 – Robô Emmy I



Fonte: TORRES, 2010, p.71

[2] lógica paraconsistente – conceito de lógica que permite trabalhar com contradições de forma não trivial. Desenvolvido em 1948 pelo lógico polonês Stanislaw Jaskowski (1906 -1965) e pelo lógico brasileiro Newton Carneiro Affonso da Costa em 1954 (1929 -), que de forma independente apresentaram as primeiras ideias da Lógica Paraconsistente (TORRES, 2010).

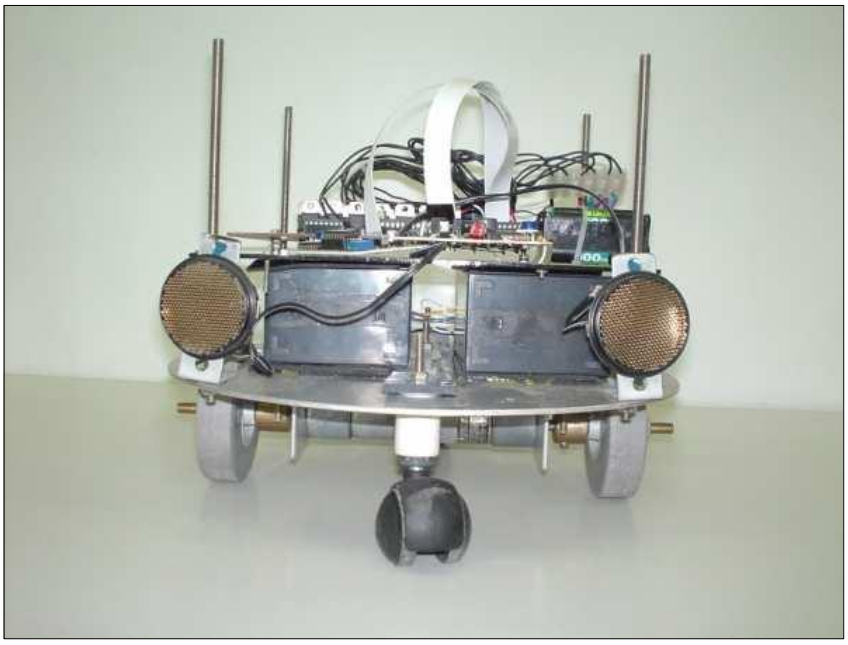
O robô Emmy I reconhece o ambiente ao seu redor por meio de seus sensores de ultrassom, e gera para o sistema de controle uma tensão que pode variar de 0 a 5 volts, dependendo da presença ou não de um obstáculo em cada sensor.

O sistema de controle do robô Emmy I utiliza a lógica paraconsistente para tratar as informações recebidas através de seus sensores e assim tomar uma ação para desviar do obstáculo.

1.6.2 Robô Emmy II

Segundo Torres (2010), o robô Emmy II é um aperfeiçoamento do robô Emmy I. Possui, basicamente, as mesmas características que o seu antecessor, mas sua estrutura mecânica é mais compacta, Figura 1.15.

Figura 1.15 – Robô Emmy II



Fonte: TORRES, 2010, p.77

1.6.3 Robô Emmy III

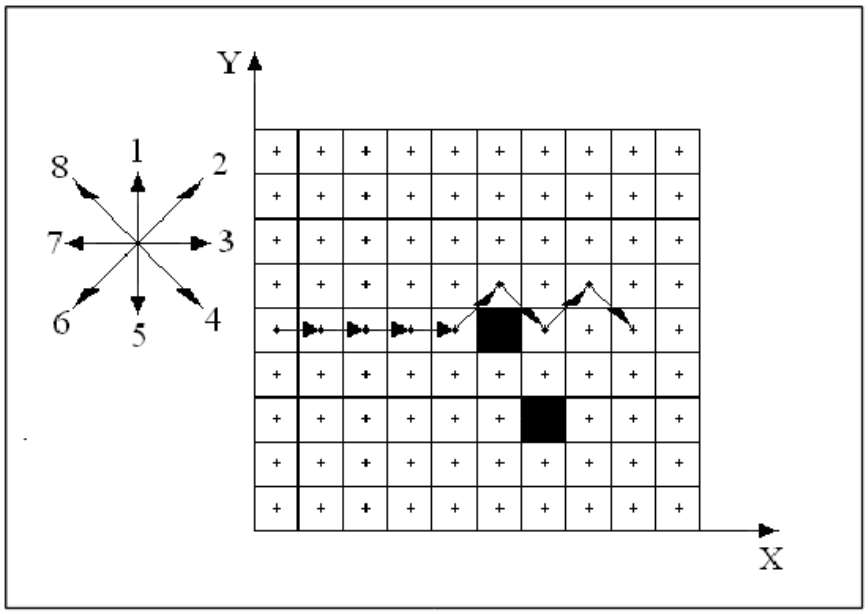
De acordo com Torres (2010), o robô Emmy III corresponde a um robô móvel autônomo com um sistema de controle e navegação. Seu funcionamento está baseado no emprego da lógica paraconsistente, sendo que esse sistema permite que o robô encontre um destino pré-determinado e se localize em um ambiente não estruturado, podendo, por exemplo, desviar de obstáculos.

Esse sistema é composto por um banco de dados e três subsistemas: subsistema de sensoriamento, subsistema mecânico e subsistema de planejamento.

- subsistema de sensoriamento: responsável pela manipulação de dados dos sensores, e pelo envio ao banco de dados as coordenadas que estão obstruídas.
- subsistema mecânico: é a estrutura física do robô com os componentes de movimentação.
- subsistema de planejamento: responsável pela leitura do banco de dados, processar e fornecer uma sequência de ações que a estrutura física deve realizar, para sair de uma posição origem (coordenadas Xo,Yo) e alcançar a posição destino (coordenadas Xd,Yd).

O subsistema de planejamento define oito tipos diferentes de movimentos. Cada movimento corresponde a um número de 1 a 8, com as direções separadas entre si em 45º. Assim, o subsistema de planejamento deve informar ao subsistema mecânico uma sequência numérica, e cada número é interpretado como um movimento diferente. O subsistema de planejamento mapeia o ambiente na forma de um plano cartesiano XY em torno do robô, dividindo-o em células, Figura 1.16.

Figura 1.16 - Exemplo de sequência gerada pelo subsistema de planejamento



Fonte: TORRES, 2010, p. 135

1.7 Algoritmo de planejamento de trajetória

Torres (2010) apresenta um algoritmo simplificado do subsistema de planejamento, enfatizando-se que funciona apenas num ambiente sem obstáculos. A modelagem do algoritmo baseia-se nos conceitos das redes de Petri.

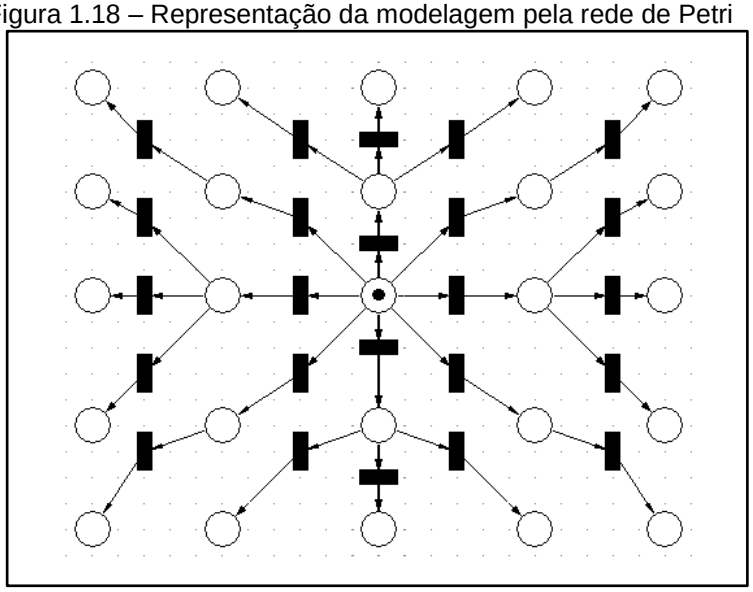
Uma rede de Petri define-se como um grafo orientado composto pelos símbolos representativos: posições ou lugares, transições e arco orientado. As posições são conectadas a outras posições através de transições. A Figura 1.17 mostra os símbolos representativos da rede de Petri.

Figura 1.17 – Símbolos representativos da rede de Petri

Fonte: TORRES, 2010, p. 108

Na modelagem do subsistema de planejamento, as posições da rede de Petri são as células do ambiente em torno do robô e as transições são os movimentos possíveis a realizar. A Figura 1.18 apresenta a representação da modelagem, onde a posição que contém marca é a célula onde se encontra o robô.

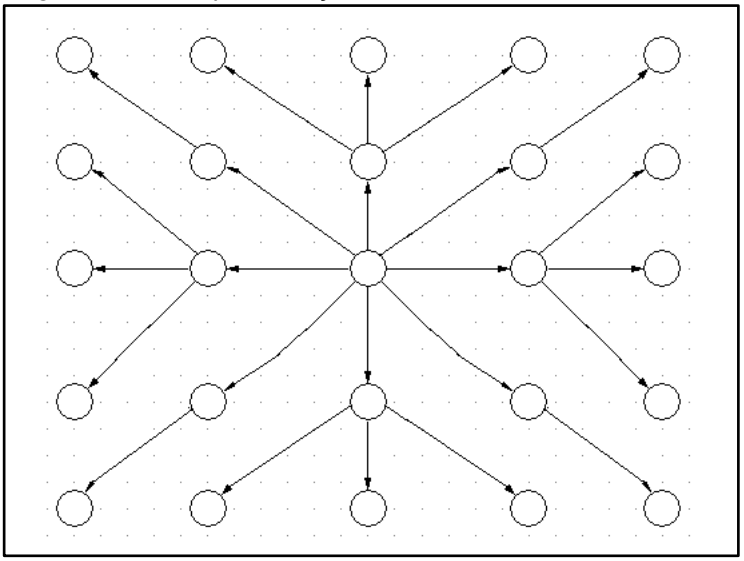
Figura 1.18 – Representação da modelagem pela rede de Petri



Fonte: TORRES, 2010, p. 110

Um planejador gera uma sequência de estados para que um sistema se desloque de um estado inicial para um estado final. Pela Figura 1.19, observa-se que o planejador tem muitos caminhos a experimentar até encontrar um que ligue o estado inicial ao final desejado.

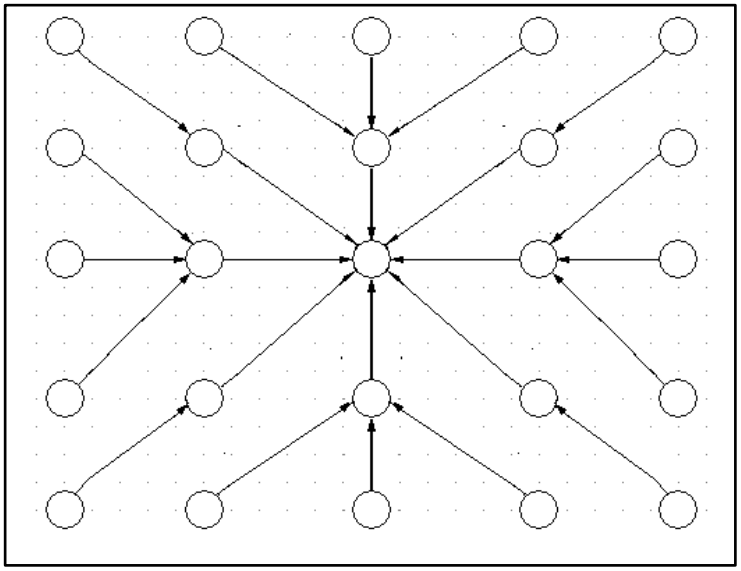
Figura 1.19 – Representação dos estados da rede de Petri



Fonte: TORRES, 2010, p. 111

Invertendo-se os sentidos dos arcos, de modo que o planejador gere a sequência de ações a partir da posição final até a posição inicial, obtemos um único caminho, Figura 1.20.

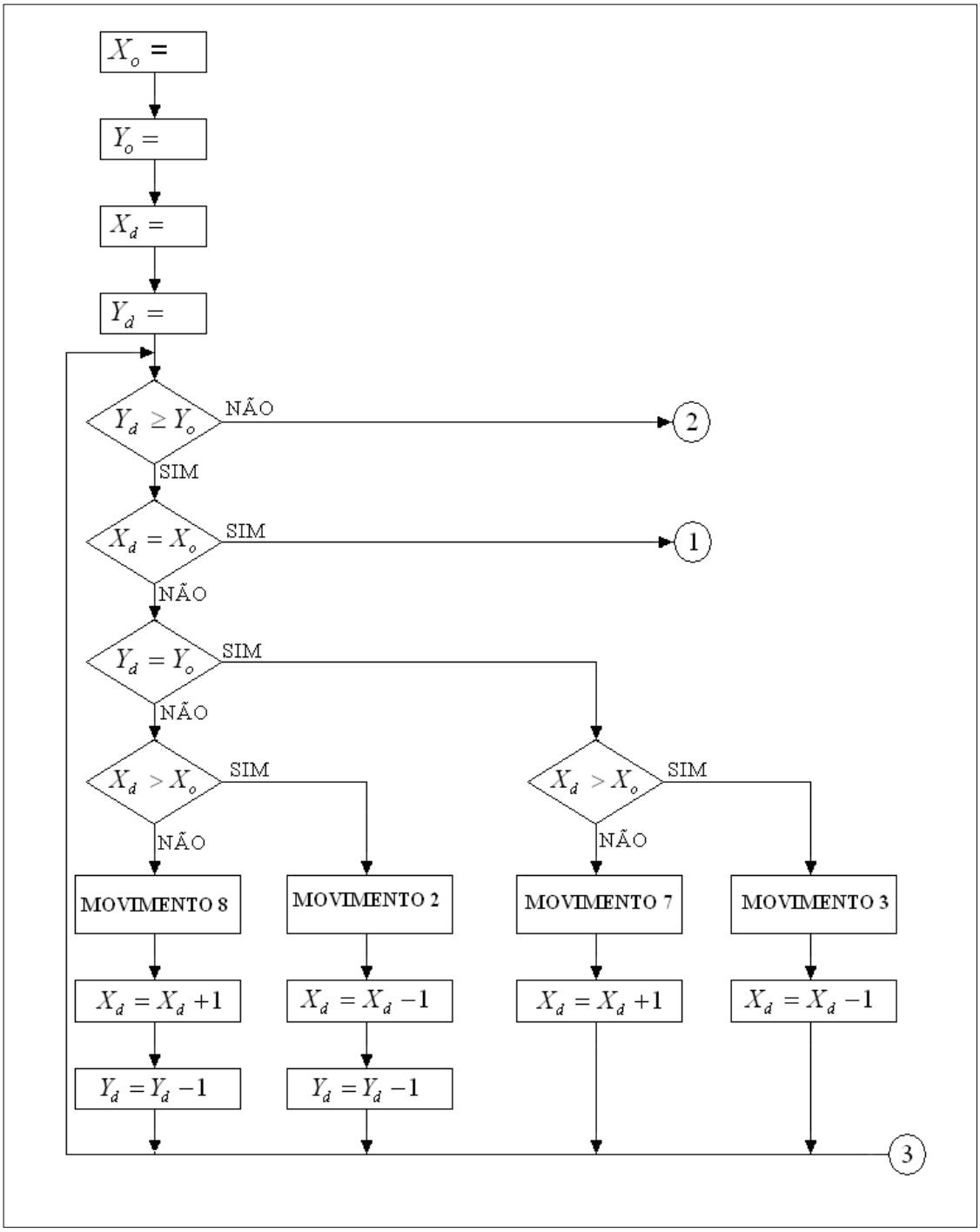
Figura 1.20 – Representação do planejador



Fonte: TORRES, 2010, p. 111

Apresenta-se a parte A do algoritmo na Figura 1.21.

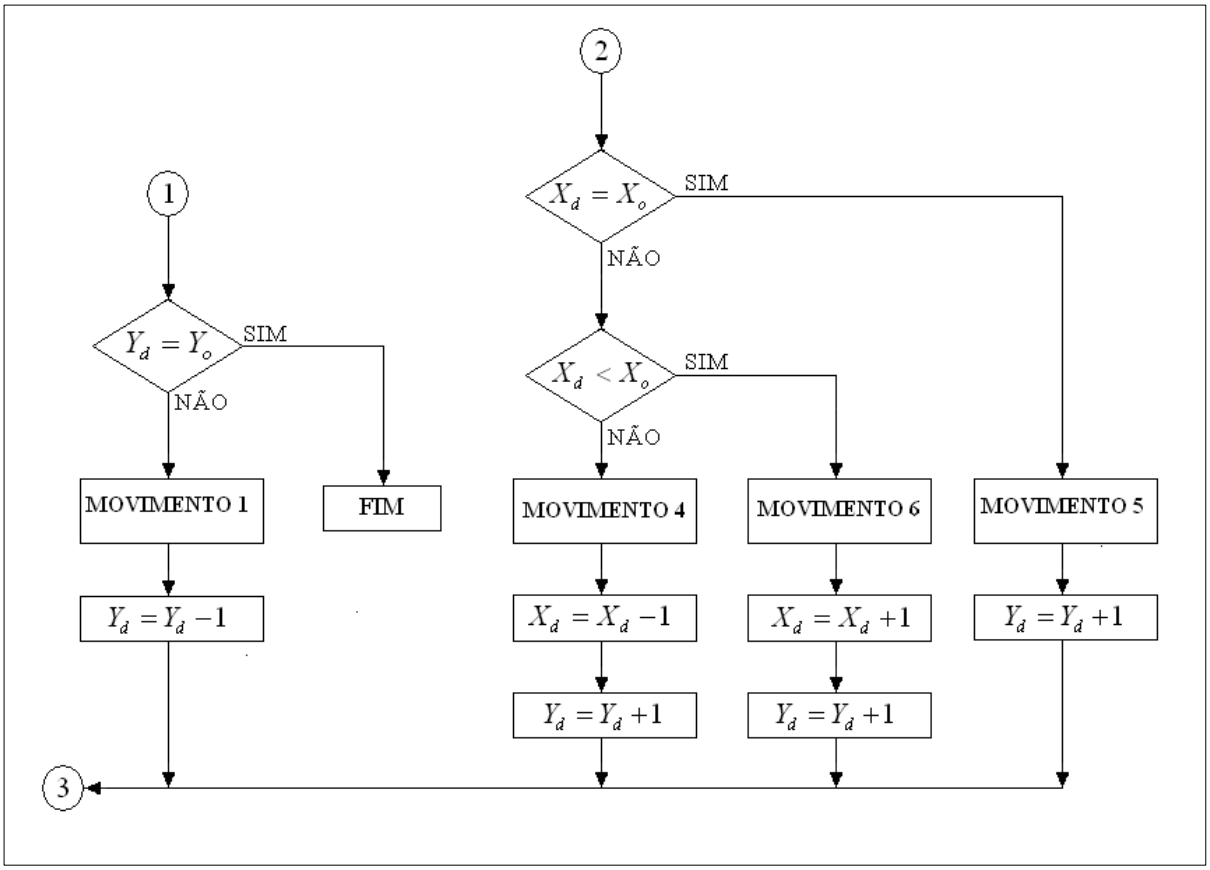
Figura 1.21– Algoritmo simplificado do subsistema de planejamento (parte A)



Fonte: TORRES, 2010, p.113

Apresenta-se na Figura 1.22 a parte B do algoritmo:

Figura 1.22 - Algoritmo simplificado do subsistema de planejamento (parte B)



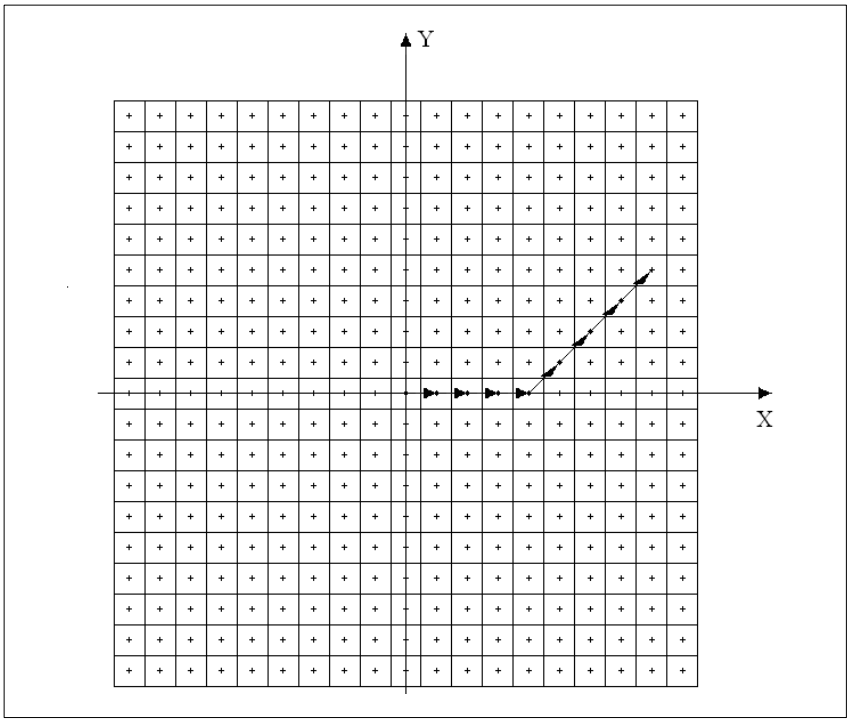
Fonte: TORRES, 2010, p.114

No algoritmo, (Xo,Yo) representa a coordenada da célula origem onde se encontra o robô, e (Xd,Yd) representa a coordenada da célula destino. Este algoritmo gera uma sequência numérica de códigos, descrito na seção 1.6.3. Esses códigos são enviados ao subsistema mecânico para a execução dos movimentos.

Segundo Torres (2010), a sequência numérica gerada pelo algoritmo corresponde a ações para que o robô saia do destino e se alcance a origem. Portanto, o último código gerado deve ser o primeiro a ser executado pela plataforma mecânica.

Uma simulação do algoritmo simplificado do subsistema de planejamento, com a coordenada de origem (0,0) e destino (8,4) é ilustrada na Figura 1.23.

Figura 1.23 – Trajetória gerada com origem (0,0) e destino (8,4)



Fonte: TORRES, 2010, p. 124

Na Figura 1.23 observa-se a sequência numérica gerada pelo algoritmo (2, 2, 2, 2, 3, 3, 3, 3). O último código gerado deve ser o primeiro a ser executado pelo robô, assim, o primeiro movimento deve ser o código 3.

2 METODOLOGIA

O presente capítulo apresenta os métodos e técnicas que são utilizadas para a realização do projeto intitulado Sistema de orientação inercial para o robô móvel autônomo Emmy III. Trata-se de uma pesquisa aplicada que é desenvolvida nas dependências da FATEC – SBC e na residência do autor do projeto.

Severino (2007) aponta que a metodologia é o caminho percorrido para descrever passo a passo as etapas da construção do projeto. São elas:

- tema-problema e justificativa;
- levantamento bibliográfico sobre o tema;
- leitura e documentação dessa bibliografia após seleção;
- construção lógica do trabalho;
- redação do texto.

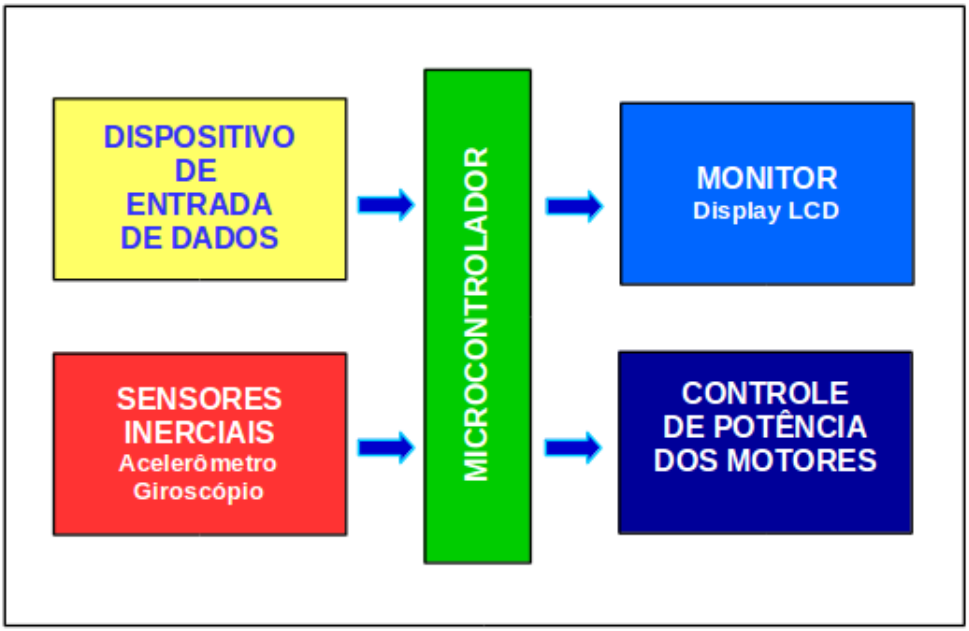
2.1 O tema-problema e justificativa

Apresentadas as explanações na introdução, decide-se construir um sistema eletrônico de orientação baseado em sensores inerciais com o desenvolvimento do programa de controle, para o projeto cujo tema-problema intitula-se Sistema de orientação inercial para o robô móvel autônomo Emmy III, com o objetivo de auxiliar no processamento e execução dos movimentos rotacionais e retilíneos.

2.2 Diagrama e modelo do projeto

O sistema de controle é integrado num circuito eletrônico. A Figura 2.1 representa o diagrama de blocos do sistema proposto neste projeto:

Figura 2.1 – Diagrama de blocos do sistema



Fonte: Autoria própria, 2018

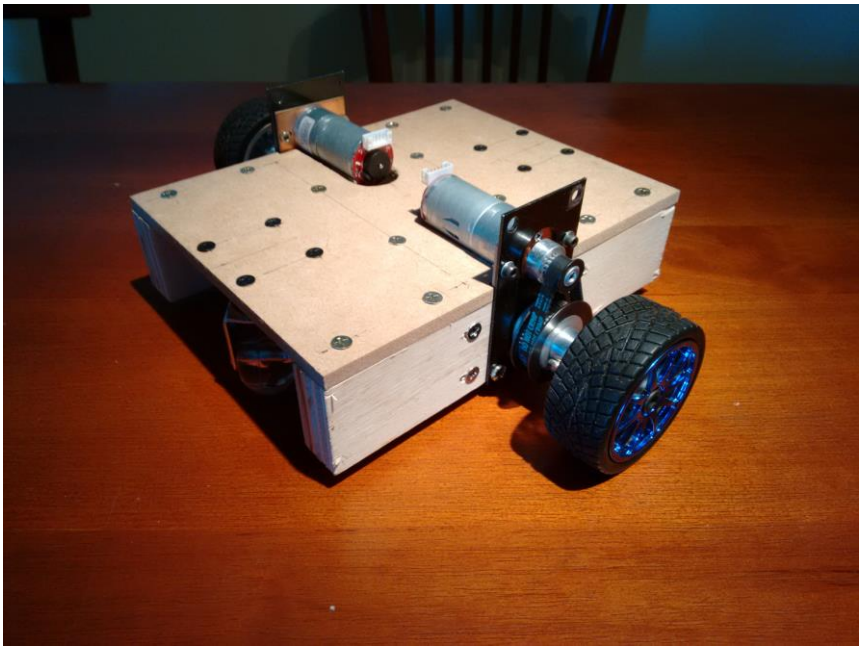
Componentes do sistema:

- dispositivo de entrada de dados: responsável pela aquisição dos dados das coordenadas de origem e destino do robô, composto por um módulo de comunicação serial via rádio padrão Bluetooth. Esse dispositivo se conecta a um telefone móvel (*smartphone*) com um programa aplicativo de comunicação serial, e através dele fornecemos os dados das coordenadas e recebemos as mensagens do robô;
- sensores inerciais: composto por um módulo com acelerômetro e giroscópio;
- microcontrolador: componente responsável pela coleta e processamento dos dados;
- monitor: composto por um visor de cristal líquido;
- controle de potência dos motores: circuito de potência para os motores.

Complementando, tem-se o desenvolvimento do programa de controle a ser implementado no microcontrolador.

A seguir, apresenta-se um modelo da plataforma mecânica do robô, para os testes de validação do sistema de controle, conforme Figura 2.2.

Figura 2.2 – Modelo de plataforma mecânica



Fonte: Autoria própria, 2018

2.3 Etapas teóricas e práticas para o desenvolvimento do projeto

Após a delimitação do tema-problema, justificativa, apresentação do diagrama de blocos e um modelo da plataforma mecânica do robô para os testes de validação do sistema de controle, seguem as etapas que sustentam o desenvolvimento do projeto:

Primeira etapa: levantamento da bibliografia. A pesquisa foi realizada na biblioteca da FATEC SBC, em *sites* especializados, publicações, trabalhos acadêmicos e manuais específicos.

Segunda etapa: após o levantamento bibliográfico, faz-se uma seleção dessas bibliografias e constrói-se o capítulo da Fundamentação teórica.

Terceira etapa: levantamento dos materiais a serem usados na construção do projeto, pesquisas em sites e empresas especializadas do ramo. Viabilidade econômica. Apresenta-se na tabela 2.1 a lista de materiais com os respectivos preços.

Tabela 2.1 – Lista de Materiais

MATERIAL	**PREÇO (R$)**
Módulo MPU-6050	25,00
Microcontrolador Atmel ATMEGA328p	48,00
Display OLED	39,90
Baterias 12V – 1,3A (2 unidades)	90,00
Módulo Bluetooth HC-06	30,00
Conversor de nível lógico	8,50
Motor de passo (2 unidades)	150,00
Controlador para motor de passo TB6560 (2 unidades)	80,00
Chave interruptora	5,00
Porta fusível	5,00
Fusível 7A	0,30
Mini protoboard	9,00
Regulador de tensão 3,3V LD33	2,50
Fios e cabos	10,00
Rodas de tração diâmetro 82mm (2 unidades)	82,00
Rodas de apoio (2 unidades)	10,00
Conjunto de eixos, mancais, polias e correias MXL (2 unid.)	60,00
Placas de madeira MDF 3mm e 6mm para o chassi	40,00
Cantoneiras L (4 unidades)	2,00
Braço suporte L (4 unidades)	4,00
Parafusos e porcas	12,00
Dobradiças (4 unidades)	2,00
Tintas spray (preto, prata e azul claro) (4 unidades)	60,00
Total	775,20

Fonte: Autoria própria, 2018

Quarta etapa: desenvolvimento do projeto, contemplando os algoritmos de controle e as montagens do circuito eletrônico e da plataforma mecânica.

Quinta etapa: elaboração das considerações finais.

3 DESENVOLVIMENTO DO PROJETO

Neste capítulo encontra-se passo a passo o desenvolvimento e construção do projeto que se intitula Sistema de orientação inercial para o robô móvel autônomo Emmy III.

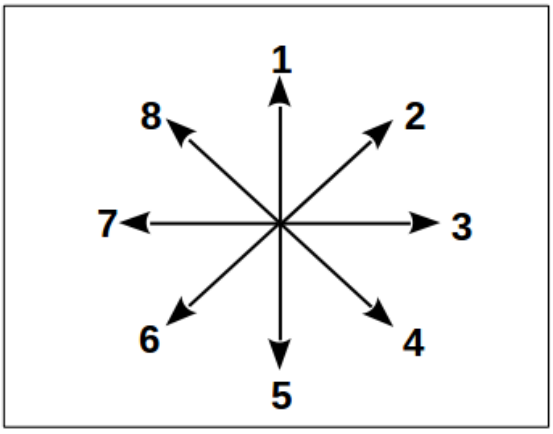
O desenvolvimento e construção do projeto são amparados nos seguintes tópicos:

- algoritmos de movimentos rotacionais e retilíneos;
- algoritmo do sistema de controle;
- programação do microcontrolador em linguagem C;
- montagem da plataforma mecânica;
- montagem do circuito eletrônico;
- testes de funcionamento;
- obstáculos e soluções.

3.1 Algoritmos de movimentos rotacionais e retilíneos

Dando início ao desenvolvimento do projeto, primeiramente, faz-se um diagrama dos oito movimentos rotacionais, que estão separados entre si em 45°, que representam os possíveis movimentos do robô, de acordo com a teoria de (TORRES, 2010), conforme ilustra a Figura 3.1.

Figura 3.1 – Diagrama dos movimentos

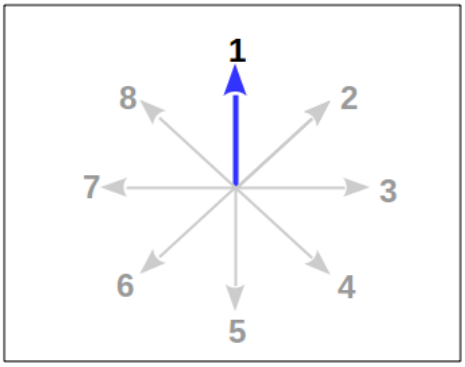


Fonte: Autoria própria, 2018

Para a realização desses movimentos, analisa-se a dinâmica da execução. Considera-se como referência a posição inicial 1, tem-se as seguintes situações:

- Movimento de 1 para 1: não há rotação, passo 0, conforme mostra a Figura 3.2;

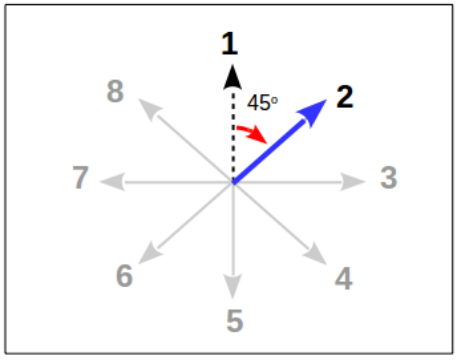
Figura 3.2 – Movimento 1 para 1, passo 0



Fonte: Autoria própria, 2018

- Movimento de 1 para 2: 45º sentido horário, 1 passo de 45º, conforme mostra a Figura 3.3;

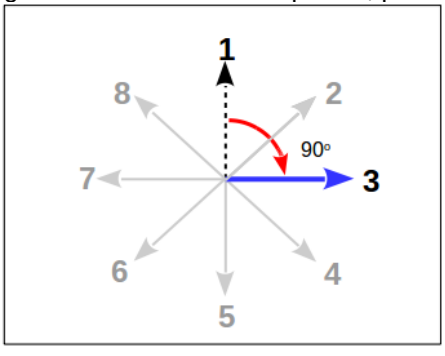
Figura 3.3 – Movimento 1 para 2, passo 1



Fonte: Autoria própria, 2018

- Movimento de 1 para 3: 90º sentido horário, 2 passos de 45º, conforme mostra a Figura 3.4;

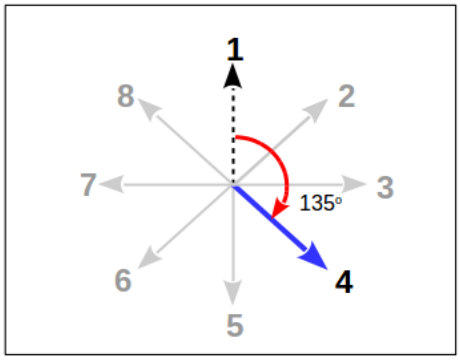
Figura 3.4 – Movimento 1 para 3, passo 2



Fonte: Autoria própria, 2018

- Movimento de 1 para 4: 135º sentido horário, 3 passos de 45º, conforme mostra a Figura 3.5;

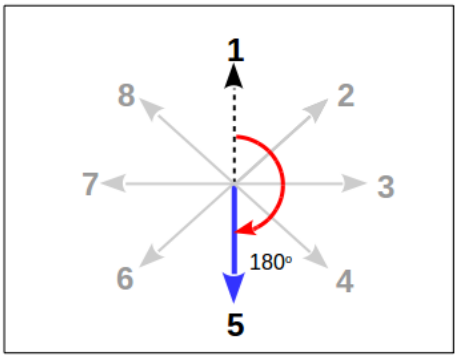
Figura 3.5 – Movimento 1 para 4, passo 3



Fonte: Autoria própria, 2018

- Movimento de 1 para 5: 180º sentido horário, 4 passos de 45º, conforme mostra a Figura 3.6;

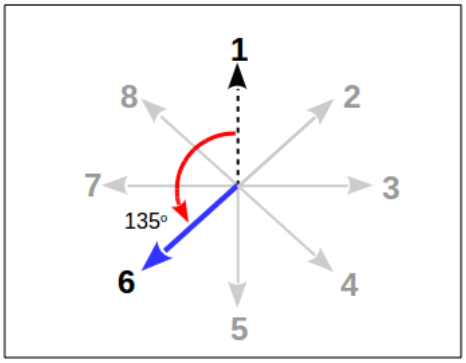
Figura 3.6 – Movimento 1 para 5, passo 4



Fonte: Autoria própria, 2018

- Movimento de 1 para 6: 135º sentido anti-horário, 3 passos de 45º, conforme mostra a Figura 3.7;

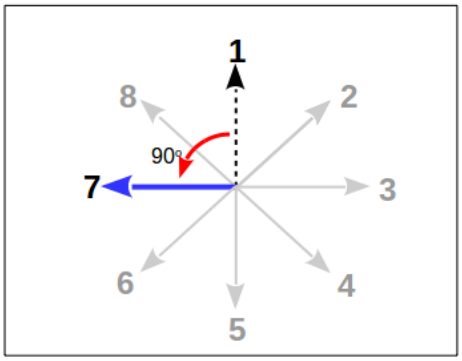
Figura 3.7- Movimento 1 para 6, passo -3



Fonte: Autoria própria, 2018

- Movimento de 1 para 7: 90º sentido anti-horário, 2 passos de 45º, conforme mostra a Figura 3.8;

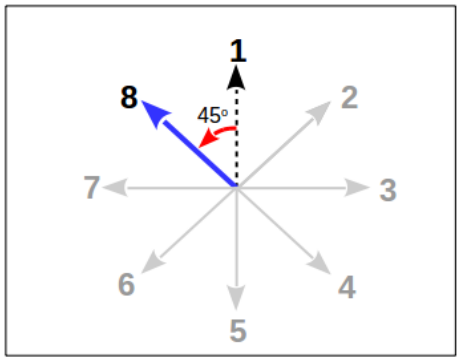
Figura 3.8 – Movimento 1 para 7, passo -2



Fonte: Autoria própria, 2018

- Movimento de 1 para 8: 45º sentido anti-horário, 1 passo de 45º, conforme mostra a Figura 3.9;

Figura 3.9 – Movimento 1 para 8, passo -1



Fonte: Autoria própria, 2018

Com base na análise dos movimentos, é elaborado uma tabela de passos de 45º, onde CÓDIGO ATUAL é o número da posição origem anterior ao movimento, e CÓDIGO RECEBIDO é o número enviado pelo subsistema de planejamento, Tabela 3.1.

Tabela 3.1 – Tabela de passos de 45º

CÓDIGO ATUAL	CÓDIGO RECEBIDO 1	2	3	4	5	6	7	8
1	0	+1	+2	+3	+4	-3	-2	-1
2	-1	0	+1	+2	+3	+4	-3	-2
3	-2	-1	0	+1	+2	+3	+4	-3
4	-3	-2	-1	0	+1	+2	+3	+4
5	-4	-3	-2	-1	0	+1	+2	+3
6	+3	+4	-3	-2	-1	0	+1	+2
7	+2	+3	+4	-3	-2	-1	0	+1
8	+1	+2	+3	+4	-3	-2	-1	0

Fonte: Autoria própria, 2018

O sinal do passo representa: (+) sentido horário, (-) sentido anti-horário. O zero significa que não há rotação.

Para o algoritmo de controle, define-se os passos através da fórmula:

$$PASSOS = COD\_REC - COD\_ATL \quad [3.1]$$

onde: COD_REC – Código recebido do subsistema de planejamento, e
COD_ATL – Código da direção atual.

Exemplos:

1) Direção atual = 1
   Código recebido = 3

Pela fórmula [3.1], temos:

PASSOS = 3 – 1

PASSOS = 2

Significa que a rotação será no sentido horário, com dois passos de 45º, ou seja, 90º.

2) Direção atual = 5

Código recebido = 2

Pela fórmula [3.1], temos:

PASSOS = 2 – 5

PASSOS = -3

Neste caso, a rotação será no sentido anti-horário, com três passos de 45º, ou 135º.

Pela fórmula [3.1], há casos onde PASSOS poderá ser maior que quatro, ou seja, o movimento rotacional será maior que 180º. Para realizar a rotação no modo mais rápido, define-se as seguintes condições:

- Se PASSOS for maior que 4:

$$PASSOS = PASSOS - 8 \qquad [3.2]$$

- Se PASSOS for menor que -3:

$$PASSOS = PASSOS + 8 \qquad [3.3]$$

Exemplo:

Direção atual = 1

Código recebido = 7

Pela fórmula [3.1], temos:

PASSOS = 7 – 1

PASSOS = 6 (maior que 4)

No caso, o movimento rotacional será horário, com seis passos de 45º, ou 270º. Para otimizar o movimento, aplicamos a fórmula [3.2]:

PASSOS = 6 – 8

PASSOS = -2

Neste caso, o movimento rotacional será anti-horário, com dois passos de 45º, ou 90º.

Para o algoritmo de controle, foram definidos os valores dos ângulos relacionados com os passos, conforme mostra a Tabela 3.2:

Tabela 3.2 – Valores dos ângulos de rotação

Número de Passos	Ângulo de Rotação
+4	+180º
+3	+135º
+2	+90º
+1	+45º
0	0
-1	-45º
-2	-90º
-3	-135º

Fonte: Autoria própria, 2018

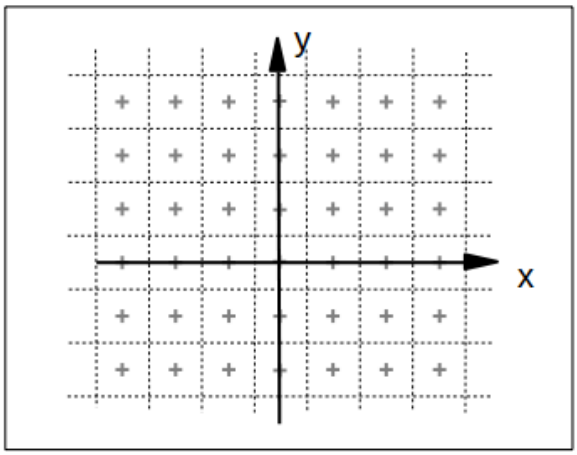
Define-se ângulo de rotação (ANG_ROT) pela fórmula:

$$ANG\_ROT = PASSOS \times 45^{o} \quad [3.4]$$

Na execução dos movimentos rotacionais, os valores angulares são comparados com a leitura do sistema inercial.

Após o processamento e a execução do movimento rotacional, ocorre o movimento retilíneo. De acordo com Torres (2010), o ambiente é dividido em células definidas por um plano cartesiano xy, Figura 3.10.

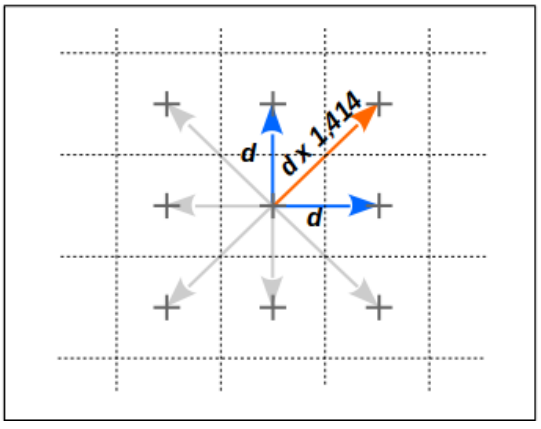
Figura 3.10 – Plano cartesiano xy



Fonte: Autoria própria, 2018

Cada célula é identificado pelo par de coordenadas (x,y). A distância entre as células, tanto na direção horizontal como na vertical, é definido como ***d***, com valor variável de acordo com a escala desejada. Nas direções diagonais define-se a distância entre células como ***d*** multiplicado por 1,414, Figura 3.11.

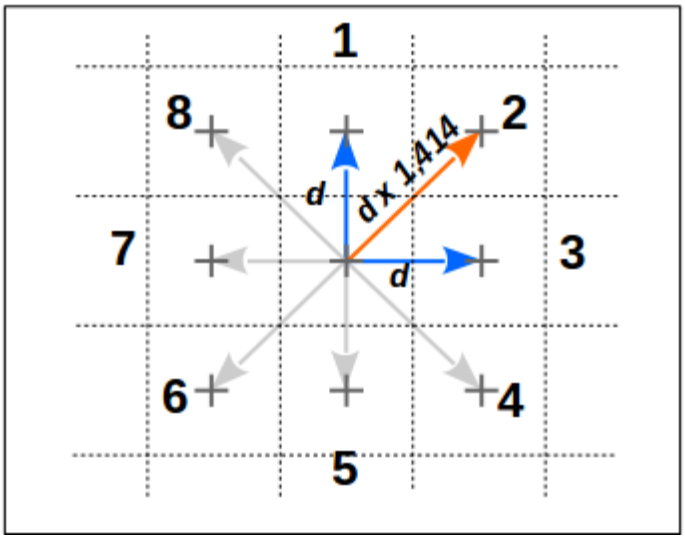
Figura 3.11 – Distâncias entre células



Fonte: Autoria própria, 2018

No algoritmo de controle, define-se a variável com o valor do movimento retilíneo como MOV_RETA. Com os códigos dos movimentos enviados pelo subsistema de planejamento, executa-se o deslocamento de uma célula a outra de acordo com as condições a seguir, Figura 3.12:

Figura 3.12 – Distâncias entre células



Fonte: Autoria própria, 2018

- Para os códigos ímpares (1, 3, 5 e 7), a distância entre células é ***d***, ou:

$$MOV\_RETA = d \qquad [3.5]$$

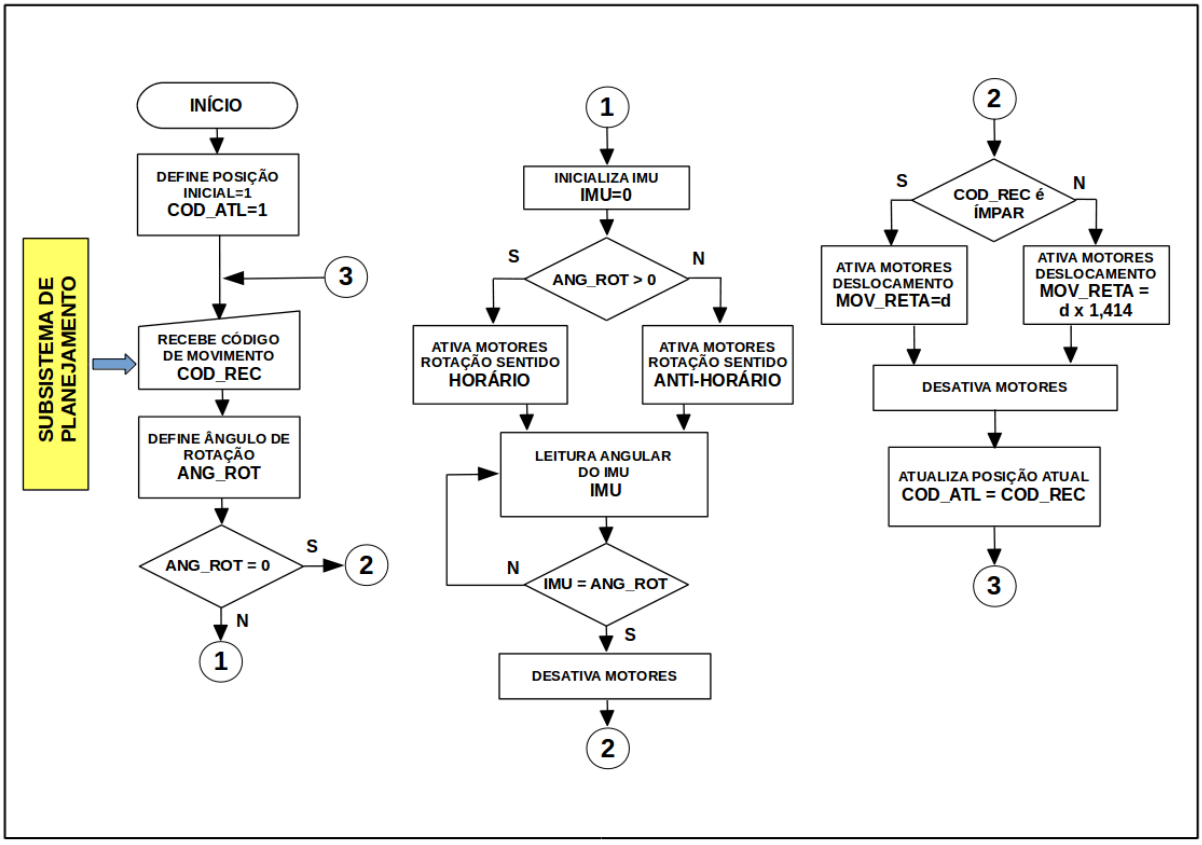
- Para os códigos pares (2, 4, 6 e 8), a distância entre células é ***d x 1,414***, ou:

$$MOV\_RETA = d \times 1{,}414 \qquad [3.6]$$

3.2 Algoritmo do sistema de controle

O funcionamento do sistema de controle é representado no fluxograma abaixo, Figura 3.13.

Figura 3.13 – Fluxograma do sistema de controle



Fonte: Autoria própria, 2018

A simulação do algoritmo se dá da seguinte maneira:

- Inicia-se com a posição inicial 1: *COD_ATL=1*;
- Aguarda o recebimento do código do subsistema de planejamento: *COD_REC*;
- Define ângulo de rotação - fórmula [1.4] : *ANG_ROT*;
- *ANG_ROT* = ZERO ?: se afirmativo vai para rotina 2 – Movimento retilíneo, caso contrário vai para rotina 1 – Movimento rotacional;

Rotina 1 – Movimento rotacional

No algoritmo, IMU corresponde ao sistema dos sensores inerciais, fornecendo o valor do ângulo de rotação.

- Inicialização: IMU com ângulo = 0º;
- *ANG_ROT* > ZERO ? : se afirmativo, ativa motores com rotação sentido HORÁRIO, caso contrário sentido ANTI-HORÁRIO;
- Executa a leitura angular do IMU;
- Se ângulo do IMU = ANG_ROT : desativa os motores e segue para rotina 2, caso contrário, volta a leitura do IMU;

Rotina 2 – Movimento retilíneo

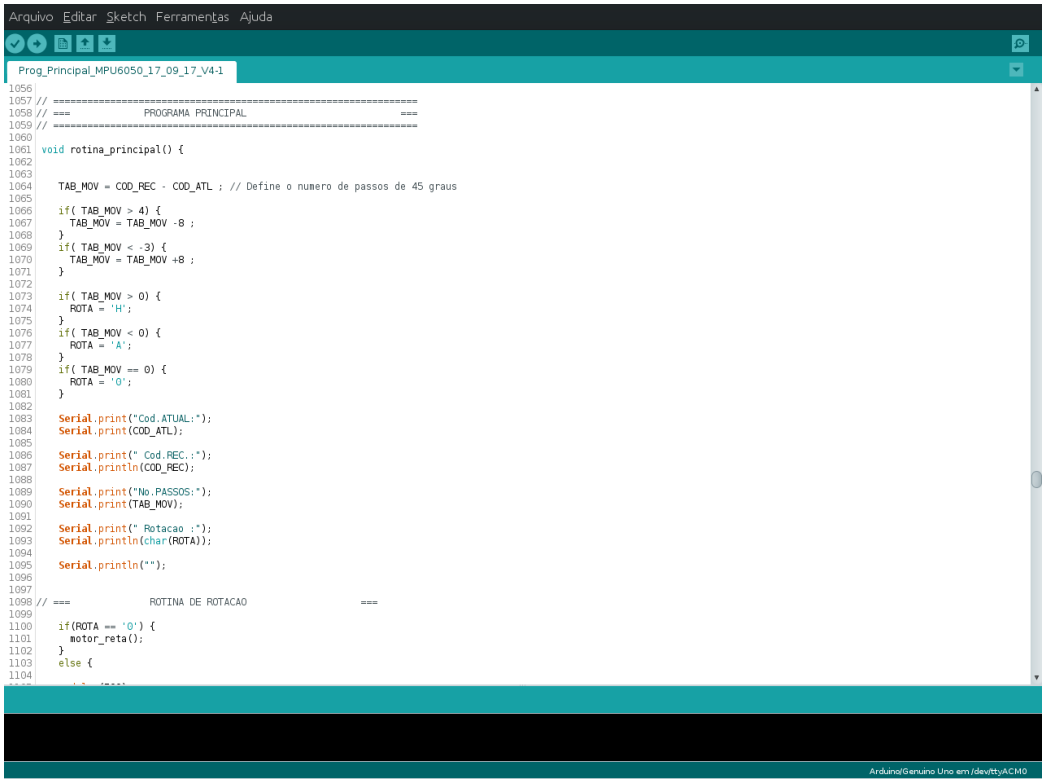
- *COD_REC* é ÍMPAR ? : se afirmativo, ativa os motores e o deslocamento retilíneo será *MOV_RETA* = d, caso contrário, o deslocamento será *MOV_RETA* = d x 1,414;
- Desativa os motores;
- Atualiza a posição atual *COD_ATL* recebendo o valor de *COD_REC*;
- Reinicia o processamento no ponto 3.

3.3 Programação do microcontrolador

Nesta seção apresenta-se a codificação dos algoritmos e o processo de gravação no microcontrolador.

O algoritmo é codificado em linguagem C através de um programa de desenvolvimento para o sistema Arduino instalado num computador pessoal (PC), conforme mostra a Figura 3.14. Esse programa compõe-se de: editor de texto, compilador para a conversão da linguagem C para linguagem de máquina (código binário), e comandos de comunicação entre o PC e o microcontrolador. O código fonte do programa é apresentado no Apêndice A.

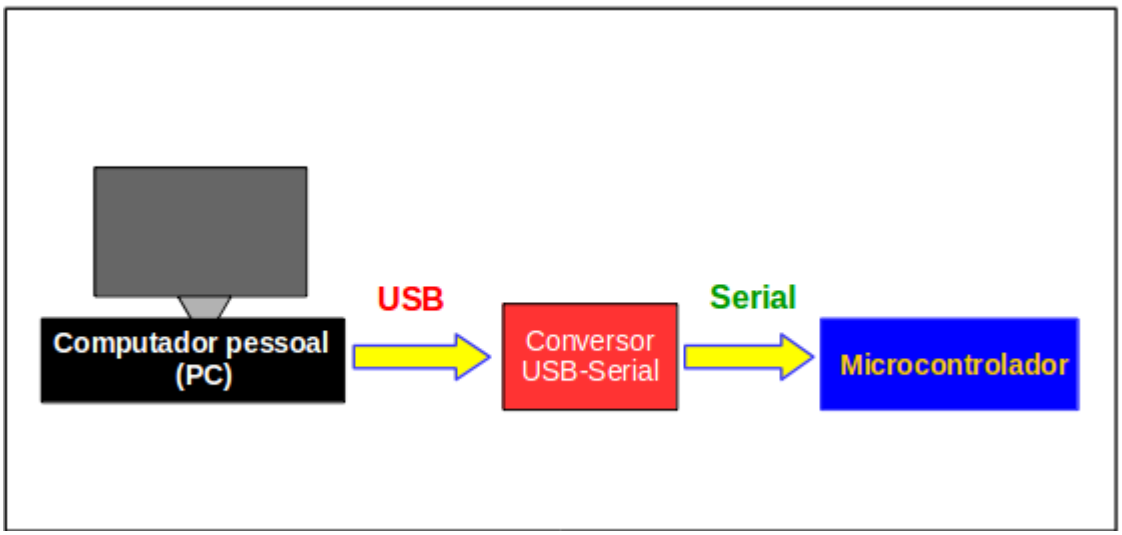
Figura 3.14 – Tela do programa de desenvolvimento



Fonte: Autoria própria, 2018

O microcontrolador conecta-se ao computador PC através de um conversor padrão USB para sinais seriais. O programa em C editado no PC converte-se em códigos binários e estes são gravados na memória do microcontrolador. A Figura 3.15 ilustra o processo de gravação do algoritmo no microcontrolador.

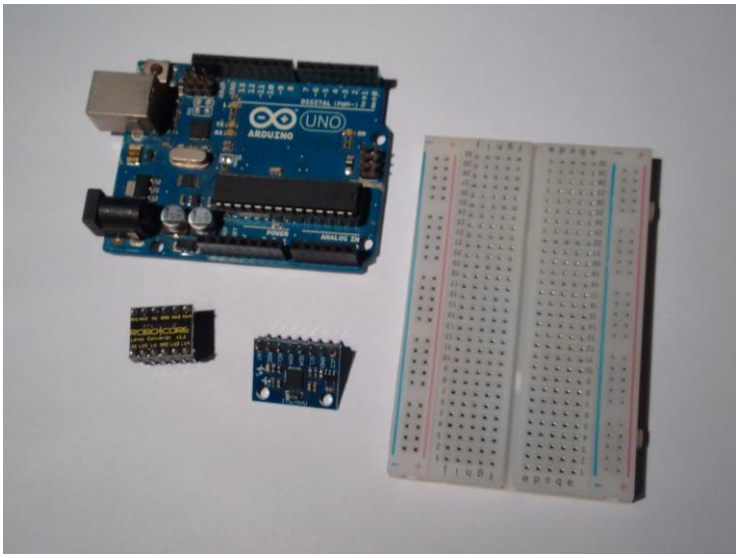
Figura 3.15 – Diagrama de conexões do processo de gravação



Fonte: Autoria própria, 2018

Para o desenvolvimento do programa, faz-se necessário um circuito com os seguintes componentes: uma placa Arduino Uno, módulo MPU-6050, módulo conversor de nível lógico, e placa de prototipagem eletrônica (*protoboard*) com fios *jumper,* Figura 3.16.

Figura 3.16 – Componentes utilizados para o desenvolvimento do programa



Fonte: Autoria própria, 2018

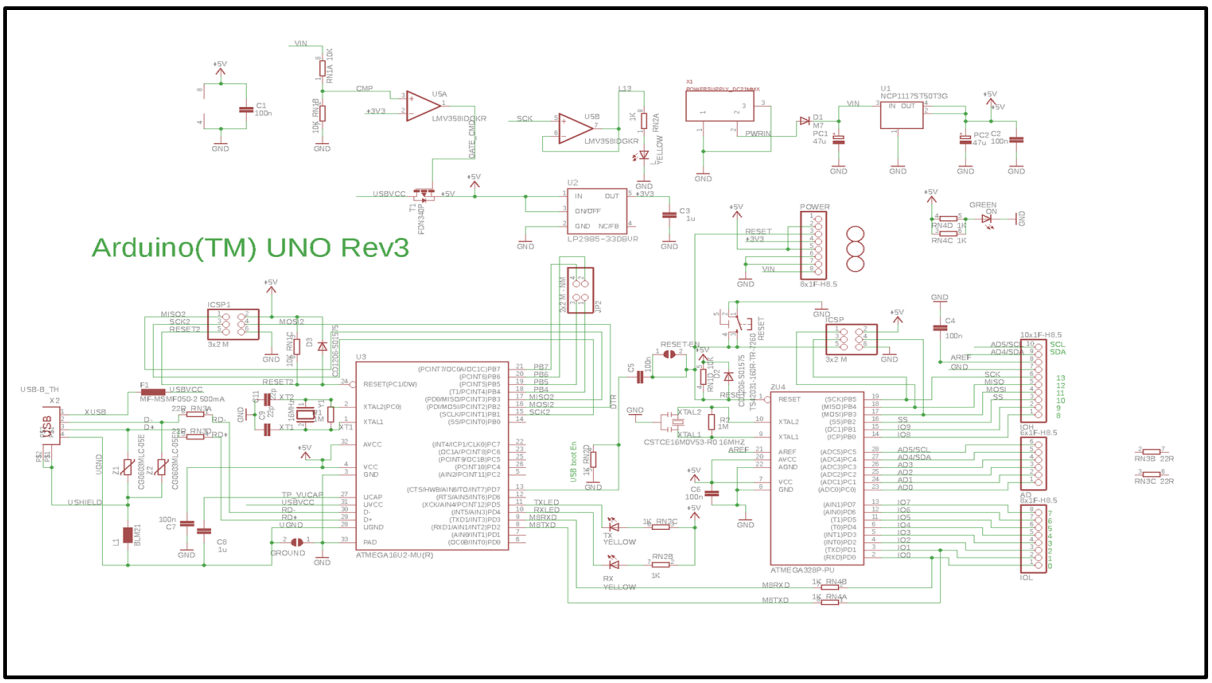
O Arduino Uno é uma placa composta por um microcontrolador ATMEGA 328p do fabricante Atmel, uma interface de comunicação com conversor USB-serial, reguladores de tensão (5V e 3,3V), além de pinos de entradas e saídas para conexões com dispositivos eletrônicos, com níveis lógicos de 5V, Figura 3.17.

Figura 3.17 – Placa Arduino Uno

Fonte: Foto de arquivo pessoal, 2018

O diagrama eletrônico da placa apresenta-se na Figura 3.18.

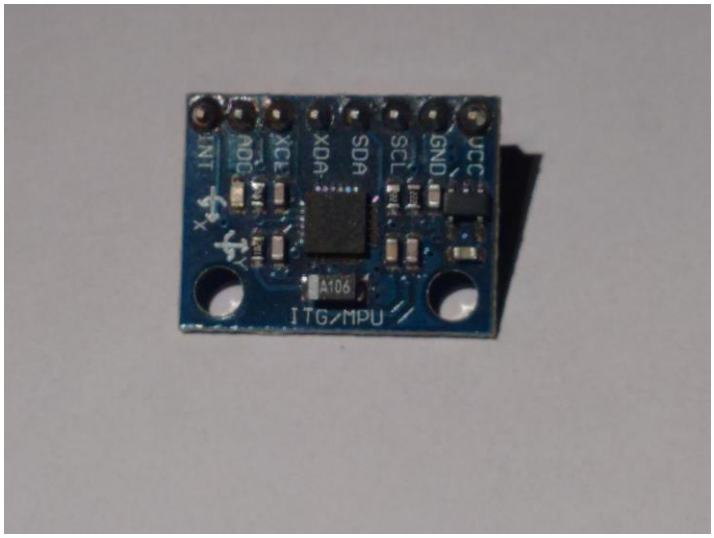
Figura 3.18 – Diagrama eletrônico do Arduino Uno



Fonte: www.arduino.cc, 2018

O módulo MPU-6050 é o sensor inercial integrado com acelerômetro e giroscópio do fabricante Invensense, Figura 3.19. Os dados são fornecidos no formato serial, através do protocolo de comunicação denominado I2C, com níveis lógicos de 3,3V.

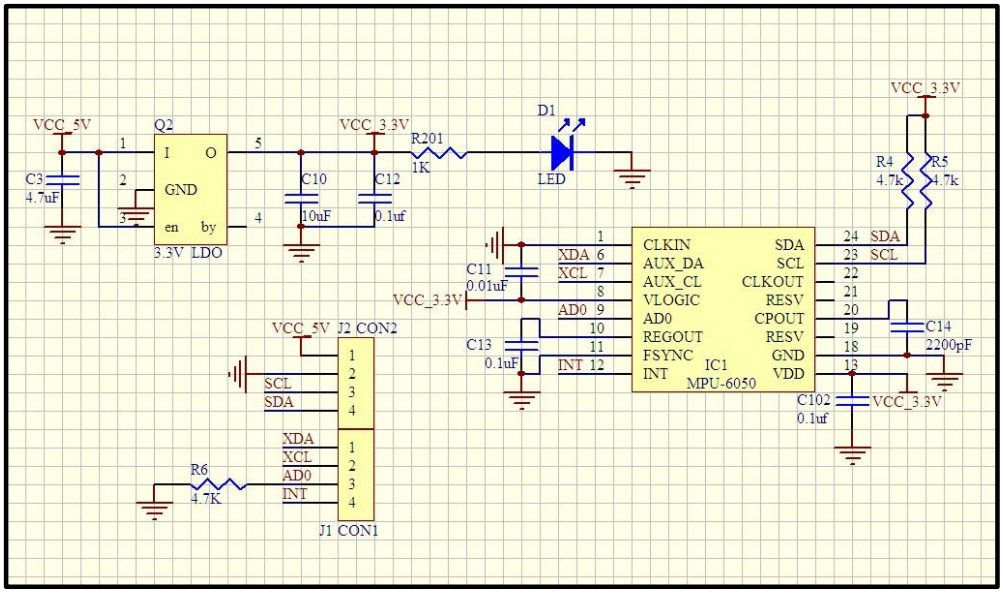
Figura 3.19 – Módulo MPU-6050



Fonte: Foto de arquivo pessoal, 2018

O diagrama eletrônico do módulo apresenta-se na Figura 3.20.

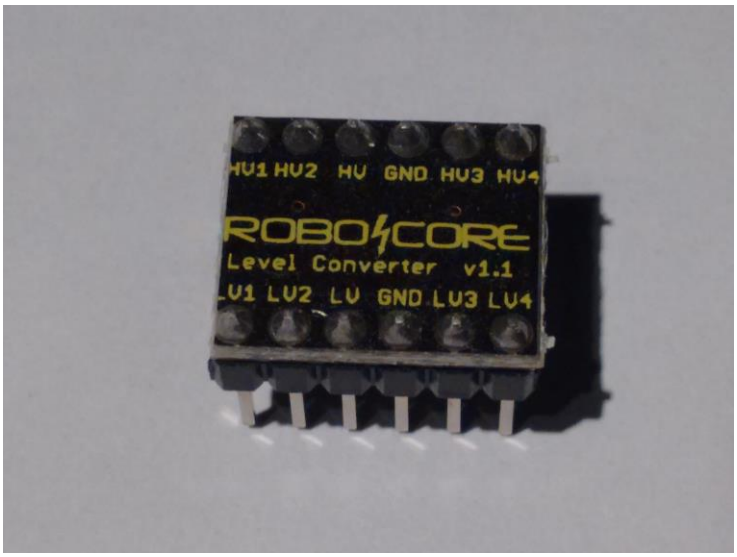
Figura 3.20 – Diagrama do módulo MPU-6050



Fonte: www.playground.arduino.cc/main/MPU-6050, 2018

O módulo conversor de nível lógico tem a função de converter sinais lógicos de 5V para 3,3V e vice-versa. O componente utilizado possui quatro canais de conversão, e é fabricado pela Robocore, Figura 3.21.

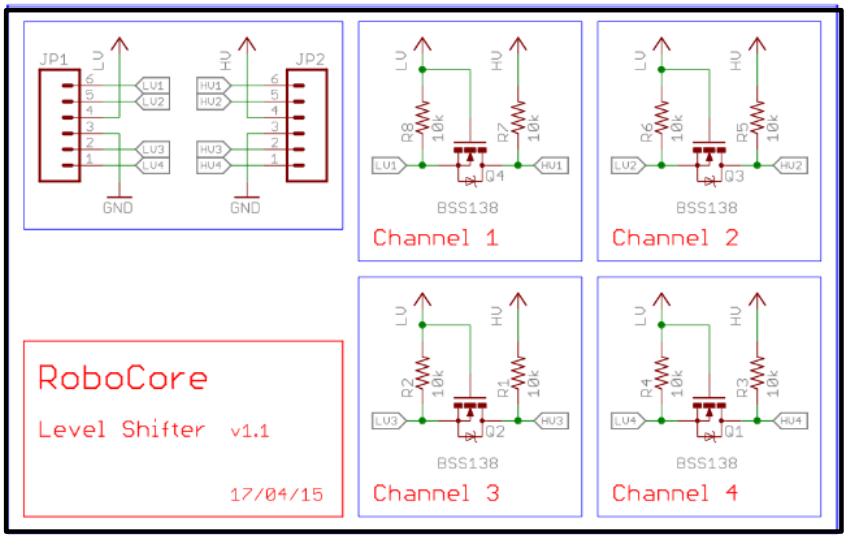
Figura 3.21- Conversor de nível lógico



Fonte: Foto de arquivo pessoal, 2018

No presente projeto, o componente faz a conversão de níveis lógicos entre o Arduino e o módulo MPU-6050. Apresenta-se na Figura 3.22 o diagrama eletrônico.

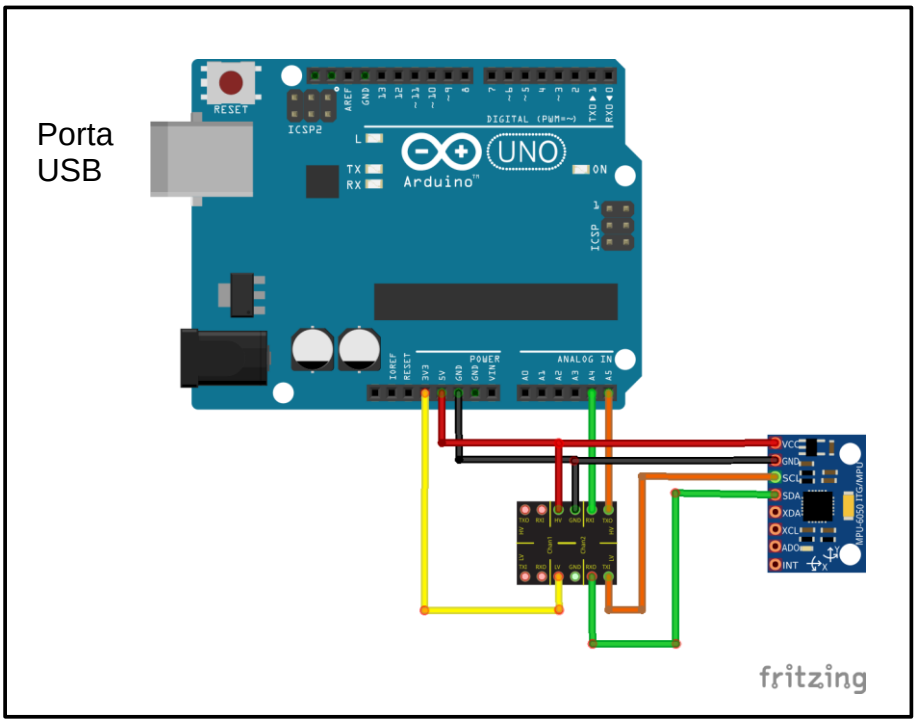
Figura 3.22 – Diagrama do conversor



Fonte: www.robocore.net, 2018

O esquema de conexões do conjunto é apresentado na Figura 3.23.

Figura 3.23 – Esquema de conexões do conjunto



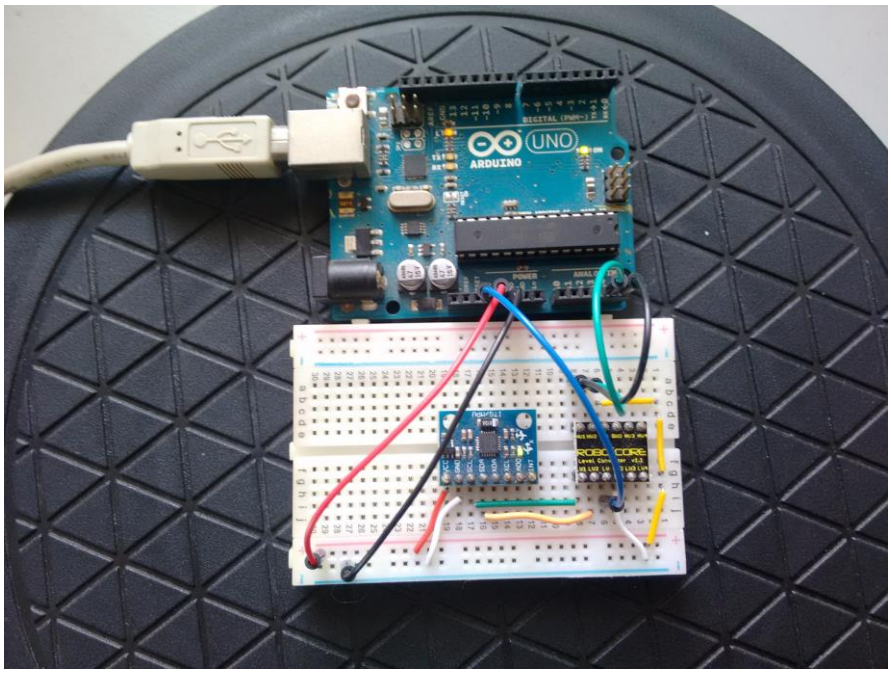
Fonte: Autoria própria, 2018

No Arduino Uno, as portas de comunicação I2C são os pinos A4 e A5 correspondendo, respectivamente, os sinais SDA (dados) e SCL (clock) do I2C. Esses pinos são conectados ao módulo conversor de nível lógico, convertendo os 5V do lado do Arduino para 3,3V do lado do MPU-6050.

A porta USB é o meio de comunicação com o computador pessoal (PC), e através dela realiza-se a transferência do programa desenvolvido no PC para o microcontrolador.

Os componentes integrados são apresentados na Figura 3.24, conforme o esquema da Figura 3.23. O conjunto está sobre uma base circular giratória, com a finalidade de testar as posições angulares.

Figura 3.24 – Circuito protótipo para o desenvolvimento do programa



Fonte: Autoria própria, 2018

3.4 Montagem da plataforma mecânica

Nesta seção apresenta-se os procedimentos para a montagem da plataforma mecânica do robô. Os desenhos técnicos dos componentes encontram-se no Apêndice B.

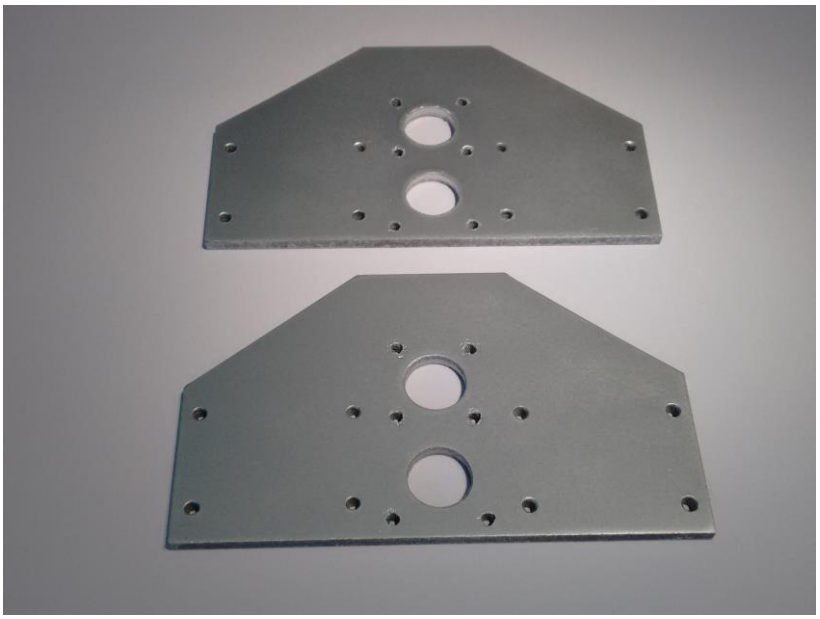
A plataforma mecânica compõe-se de:

- chassi estrutural - confeccionado com chapas de madeira MDF e placas de madeira compensado;
- sistema de tração - com eixos de aço, polias padrão MXL, correias, motores de passo e rodas.

3.4.1 Chassi estrutural

Apresenta-se a seguir a sequência de montagem do chassi. A figura 3.25 mostra as placas que representam as laterais do robô. O material é MDF com 6mm de espessura.

Figura 3.25 – Laterais do robô



Fonte: Autoria própria, 2018

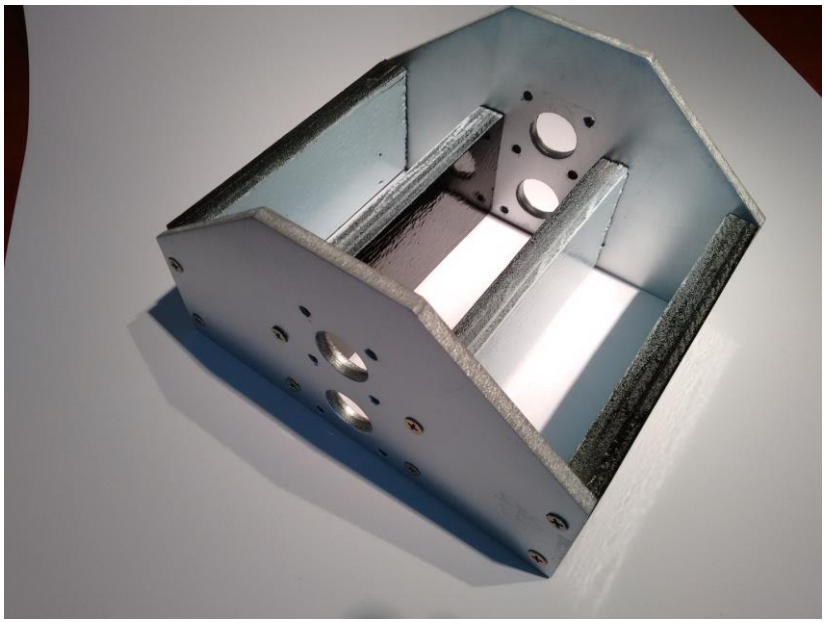
A Figura 3.26 mostra as placas de madeira compensado de 15mm de espessura, e formam as travessas que unem as duas placas laterais.

Figura 3.26 – Placas de compensado para travessas

Fonte: Autoria própria, 2018

A Figura 3.27 mostra a montagem das travessas com as laterais afixadas.

Figura 3.27 – Montagem das travessas com as laterais



Fonte: Autoria própria, 2018

3.4.2 Sistema de tração

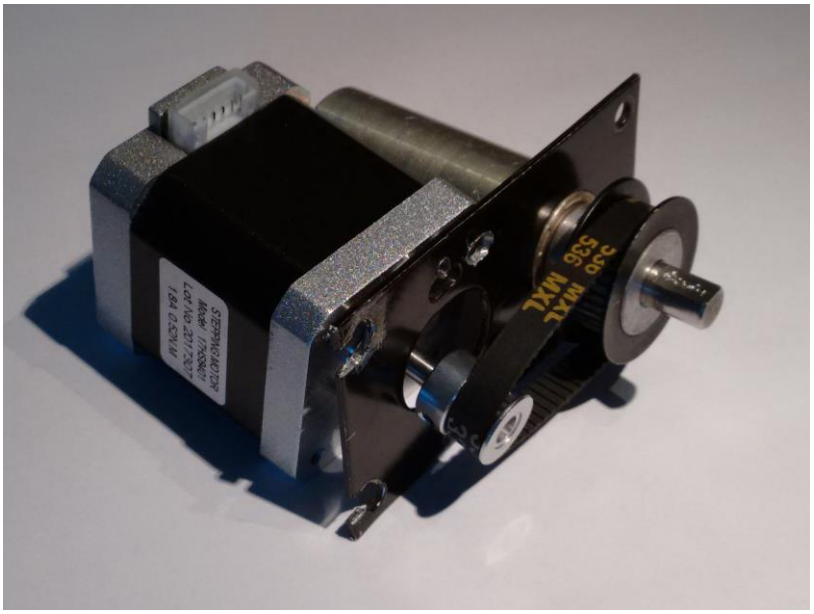
O sistema de tração compõe-se de duas peças formados por: eixos de aço com diâmetro 8mm, mancais com rolamentos, polias padrão MXL de 36 dentes, conforme mostra a Figura 3.28.

Figura 3.28 – Mancais com rolamentos, eixos de aço e polias MXL

Fonte: Foto de arquivo pessoal, 2018

A movimentação do mecanismo é efetuada através de motores de passo, instalados com uma polia padrão MXL de 16 dentes e uma correia, Figura 3.29.

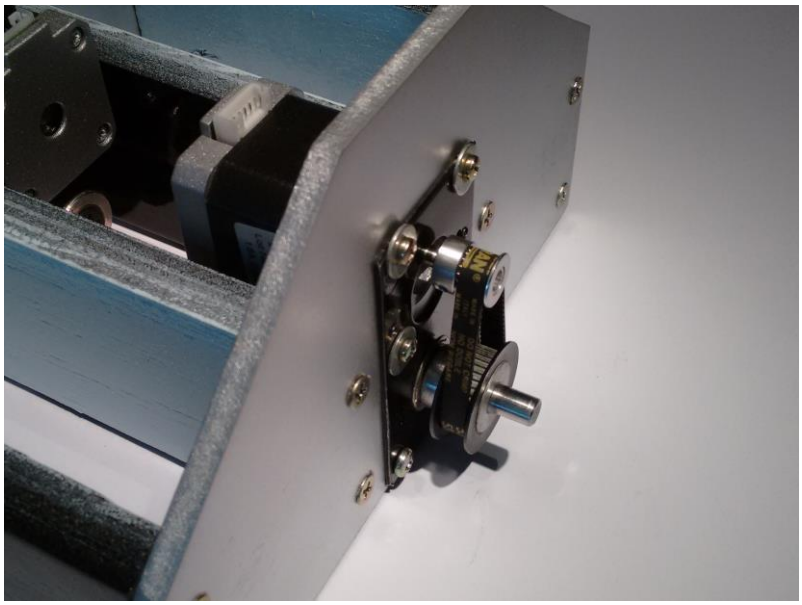
Figura 3.29 – Aspecto de montagem do motor de passo no mecanismo



Fonte: Autoria própria, 2018

A montagem do conjunto motor de passo e do mecanismo no chassi apresenta-se na Figura 3.30.

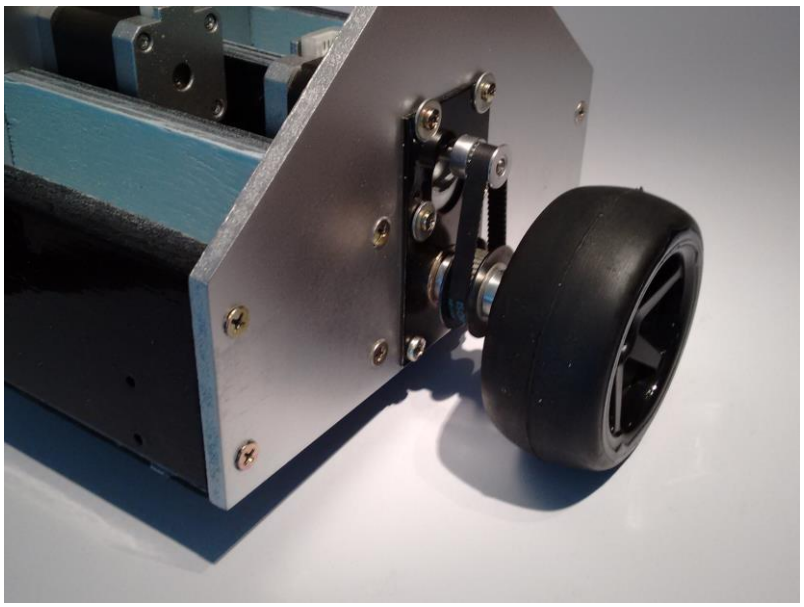
Figura 3.30 – Mecanismo instalado no chassi



Fonte: Autoria própria, 2018

O passo seguinte corresponde a instalação das rodas. As rodas são de plástico e borracha, diâmetro de 82mm, com um cubo central de alumínio para encaixe em eixo de 8mm de diâmetro. A Figura 3.31 mostra o detalhe da instalação.

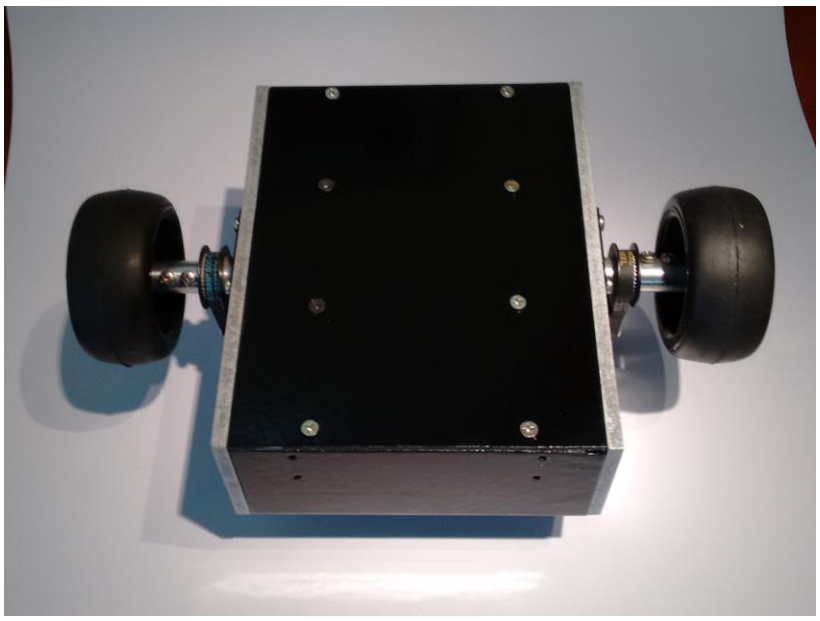
Figura 3.31 – Instalação das rodas



Fonte: Autoria própria, 2018

Para fechar a parte inferior do robô, instalou-se uma placa de madeira MDF de 3mm de espessura, conforme apresenta a Figura 3.32.

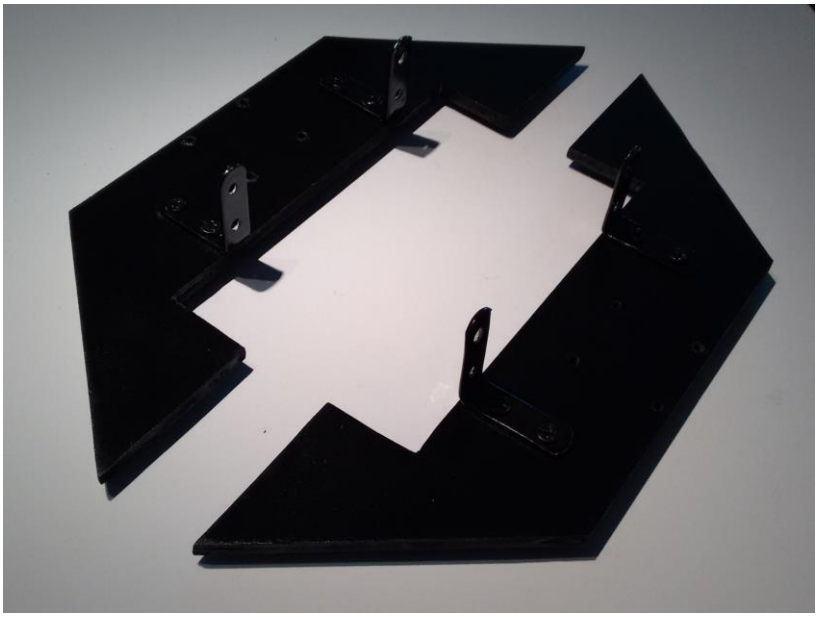
Figura 3.32 – Placa inferior do robô



Fonte: Autoria própria, 2018

Para a instalação das rodas de apoio, foram cortadas duas placas de MDF de 6mm de espessura, conforme apresenta a Figura 3.33.

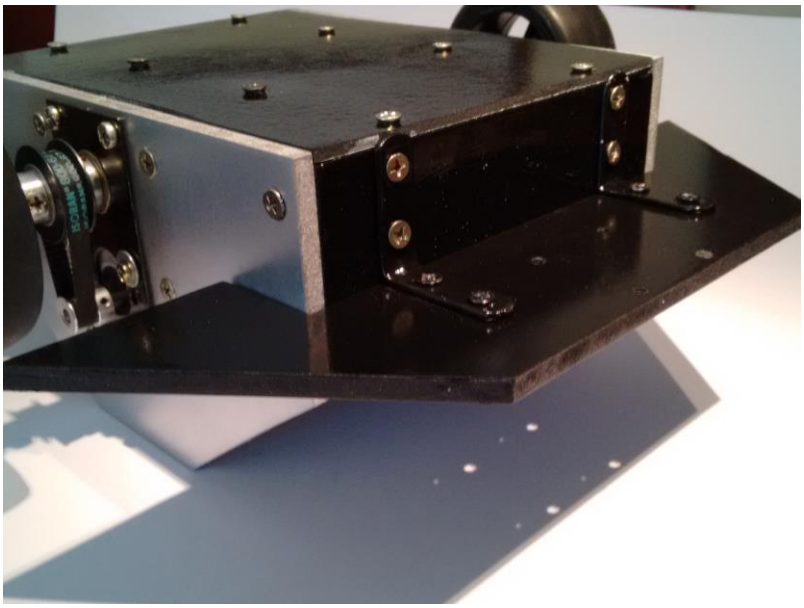
Figura 3.33 – Placas para as rodas de apoio



Fonte: Autoria própria, 2018

As placas são afixadas no chassi através de braços em L, conforme apresenta a Figura 3.34.

Figura 3.34 – Fixação das placas para as rodas de apoio



Fonte: Autoria própria, 2018

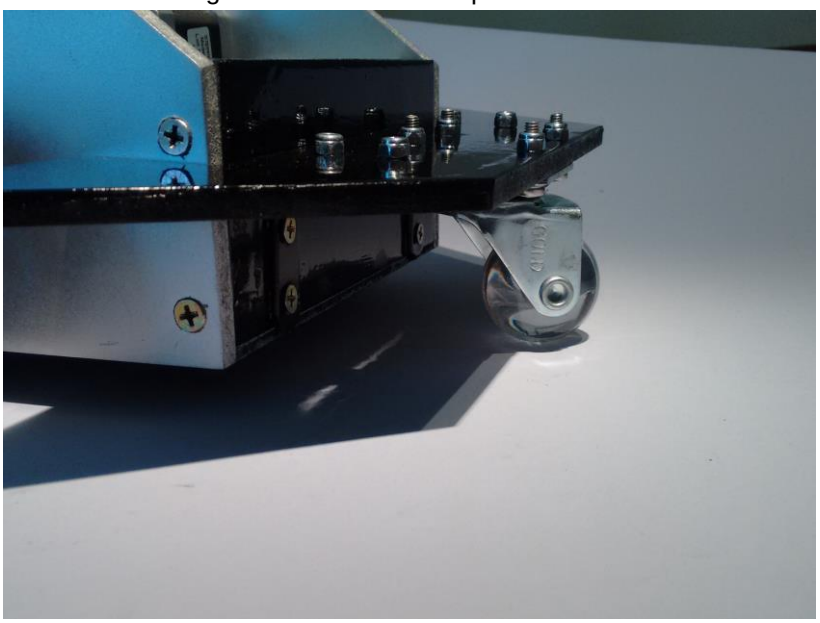
As rodas de apoio são instaladas nas placas, conforme a Figura 3.35.

Figura 3.35 – Fixação das rodas de apoio

Fonte: Autoria própria, 2018

A Figura 3.36 mostra a roda de apoio na posição de movimentação.

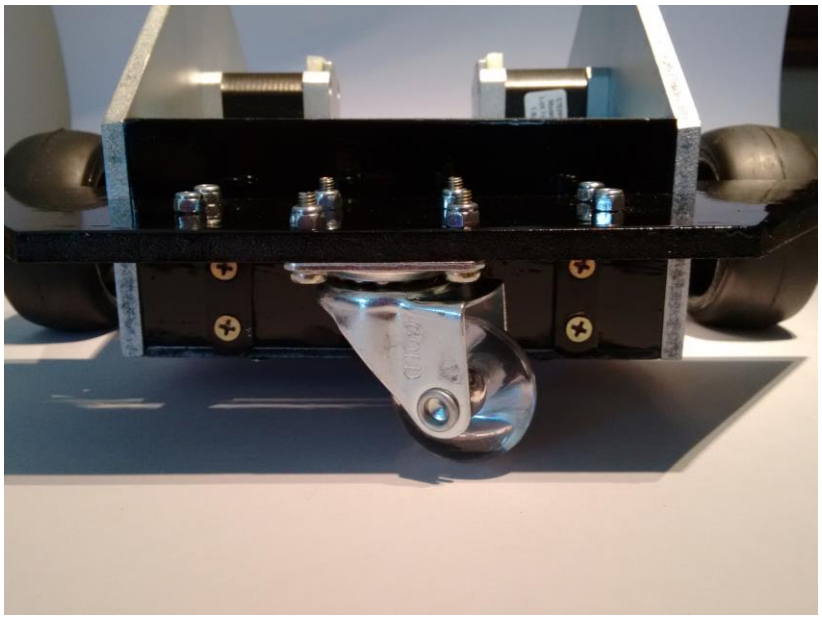
Figura 3.36 – Roda de apoio instalado



Fonte: Autoria própria, 2018

A Figura 3.37 apresenta outra vista da montagem.

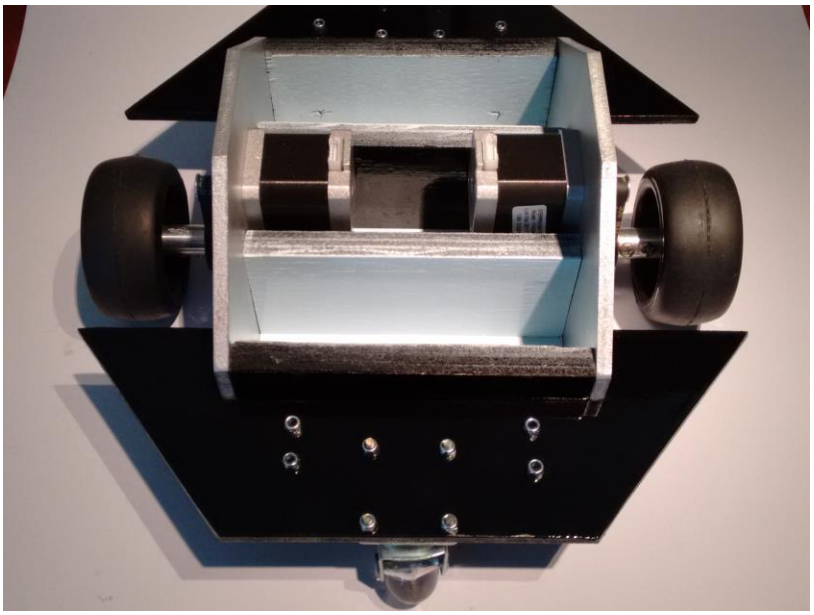
Figura 3.37 – Roda de apoio instalado



Fonte: Autoria própria, 2018

A Figura 3.38 apresenta a vista superior do chassi montado.

Figura 3.38 – Vista superior do chassi



Fonte: Autoria própria, 2018

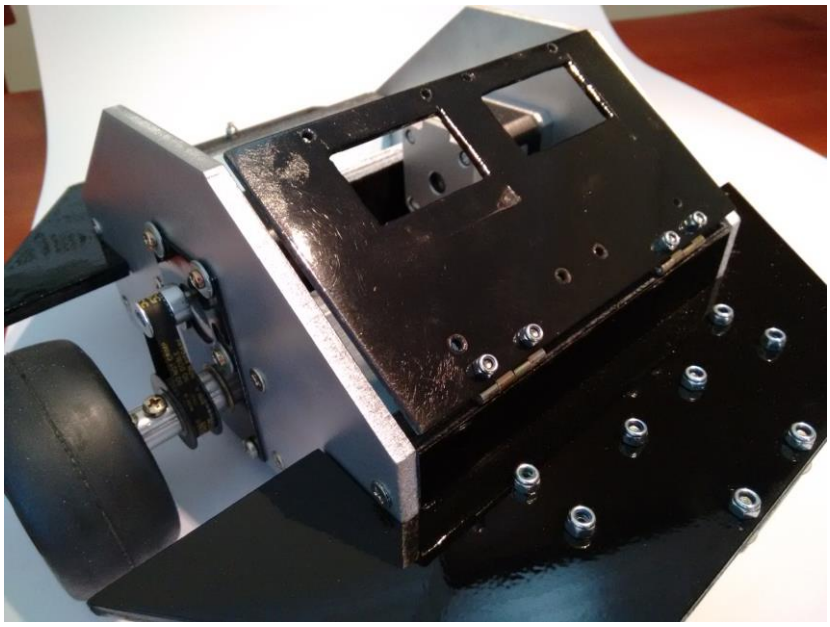
Para a instalação das placas de controle dos motores de passo, instalou-se uma placa de MDF com espessura de 3mm, afixada com duas dobradiças, conforme apresenta a Figura 3.39.

Figura 3.39 – Suporte para os controladores dos motores

Fonte: Autoria própria, 2018

A Figura 3.40 apresenta o suporte na posição fechada.

Figura 3.40 – Suporte para os controladores dos motores



Fonte: Autoria própria, 2018

3.5 Montagem do circuito eletrônico

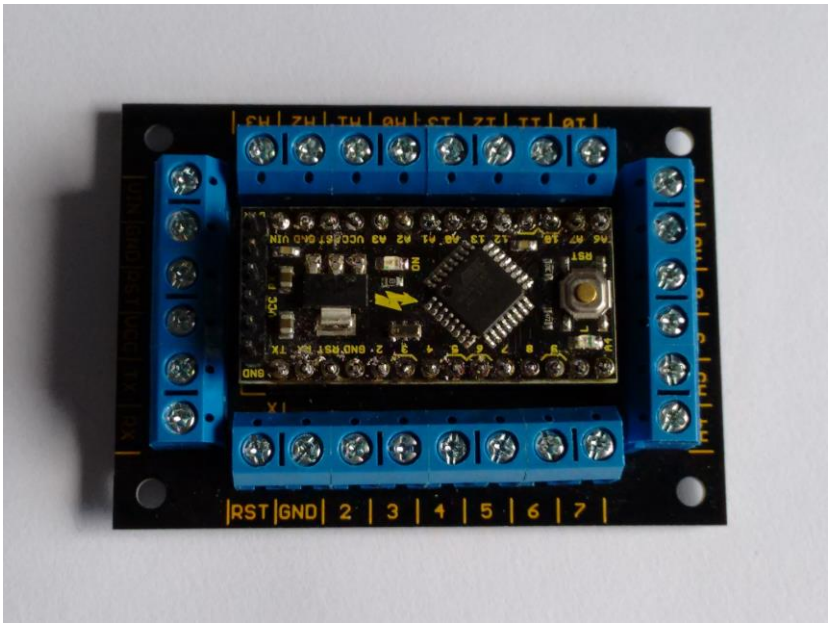
Nesta seção apresenta-se os procedimentos para a montagem e instalação do circuito eletrônico.

O circuito eletrônico compõe-se de:

- microcontrolador Atmega m328p;
- sensor inercial MPU6050;
- conversor de nível lógico;
- regulador de tensão para 3,3V LD33;
- display OLED de 0,96”;
- módulo transceptor padrão Bluetooth HC-06;
- controladores para motores de passo TB6560;
- baterias de 12V-1,3A.

Neste projeto utilizou-se um módulo do fabricante Robocore, com o microcontrolador Atmega m328p e com bornes de terminais para as conexões, conforme mostra a Figura 3.41. Esse módulo é compatível com o Arduino Uno utilizado no desenvolvimento do programa.

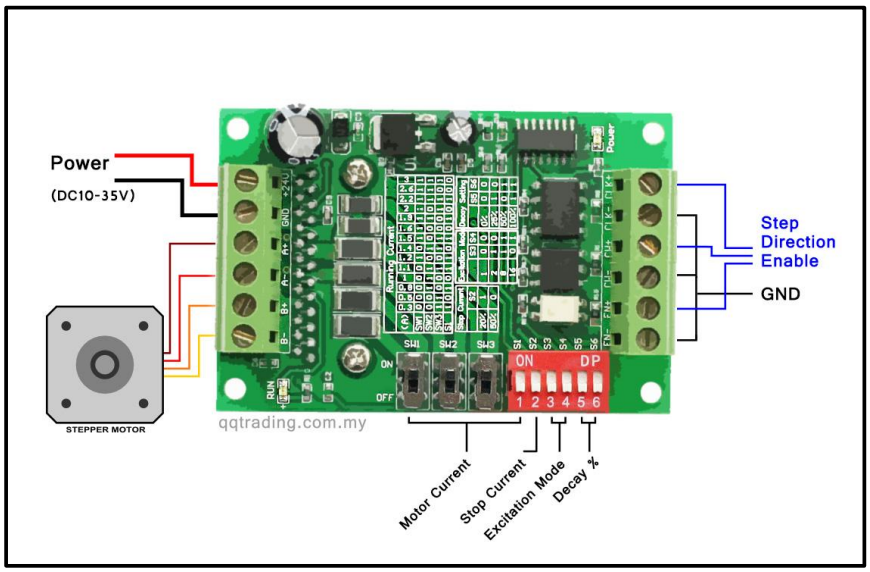
Figura 3.41 – Módulo microcontrolador Robocore



Fonte: Foto de arquivo pessoal, 2018

Um motor de passo necessita de um controlador para a movimentação. No presente projeto utilizou-se um controlador TB6560, conforme mostra a Figura 3.42.

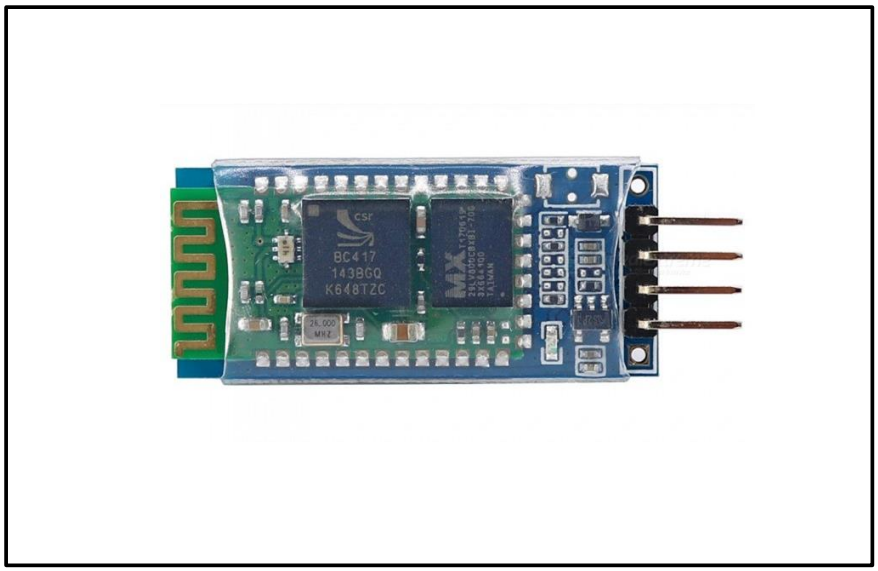
Figura 3.42 – Controlador de motor de passo TB6560



Fonte: www.cb-electronics.com, 2018

A Figura 3.43 mostra o módulo HC-06, que é o componente para a comunicação serial via rádio, padrão Bluetooth. Através dele torna-se possível a conexão com um dispositivo móvel (*smartphone*).

Figura 3.43 – Módulo HC-06



Fonte: www.dx.com , 2018

O display OLED tem a finalidade de apresentar as mensagens do processamento ao operador. A Figura 3.44 apresenta o módulo.

Figura 3.44 – Display OLED

Fonte: https://pt.aliexpress.com , 2018

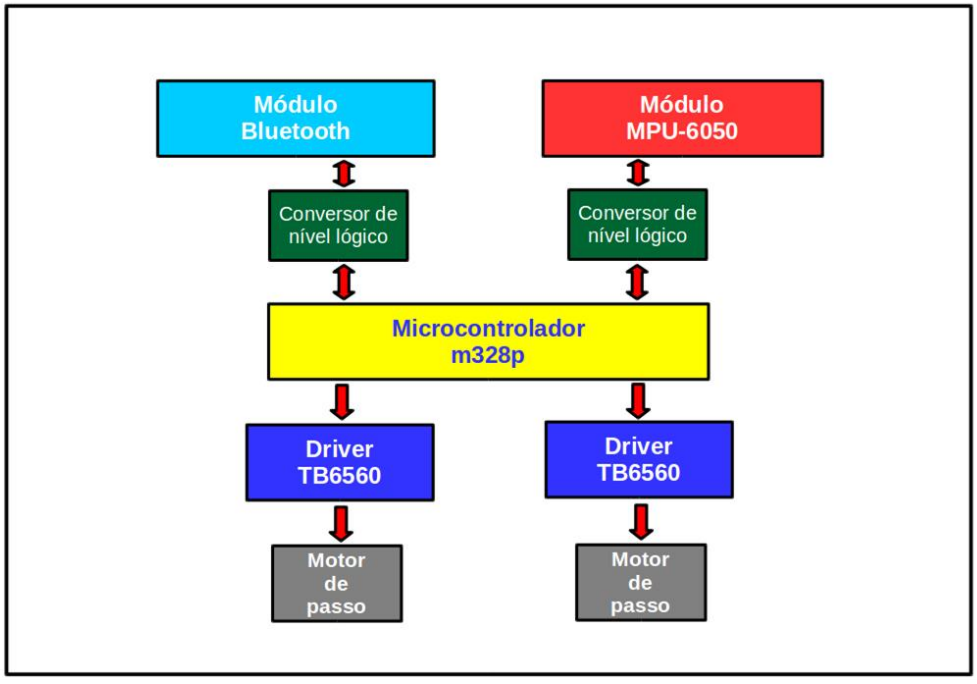
A alimentação do sistema é fornecida através de duas baterias do tipo chumbo-ácido, com tensão de 12V e capacidade de 1,3A cada, conforme mostra a Figura 3.45.

Figura 3.45 – Baterias de 12V

Fonte: Foto de arquivo pessoal, 2018

A Figura 3.46 apresenta o diagrama de blocos da integração dos componentes do projeto.

Figura 3.46 – Diagrama de blocos dos componentes



Fonte: Autoria própria, 2018

Apresenta-se a seguir a sequência de montagem e instalação dos componentes do circuito eletrônico.

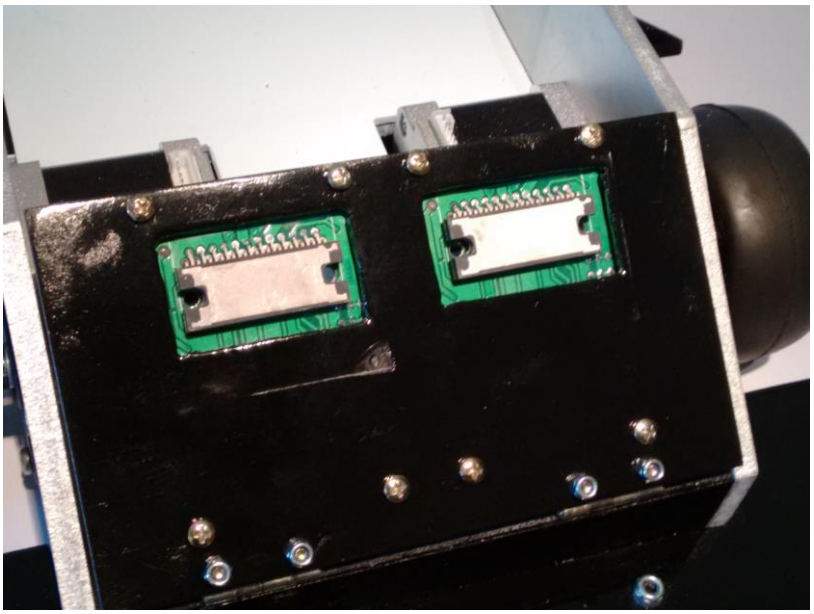
A Figura 3.47 apresenta a instalação dos controladores TB6560.

Figura 3.47 – Instalação dos controladores TB6560

Fonte: Autoria própria, 2018

A Figura 3.48 apresenta a outra face da placa suporte, destacando os circuitos integrados TB6560 sem os dissipadores de calor.

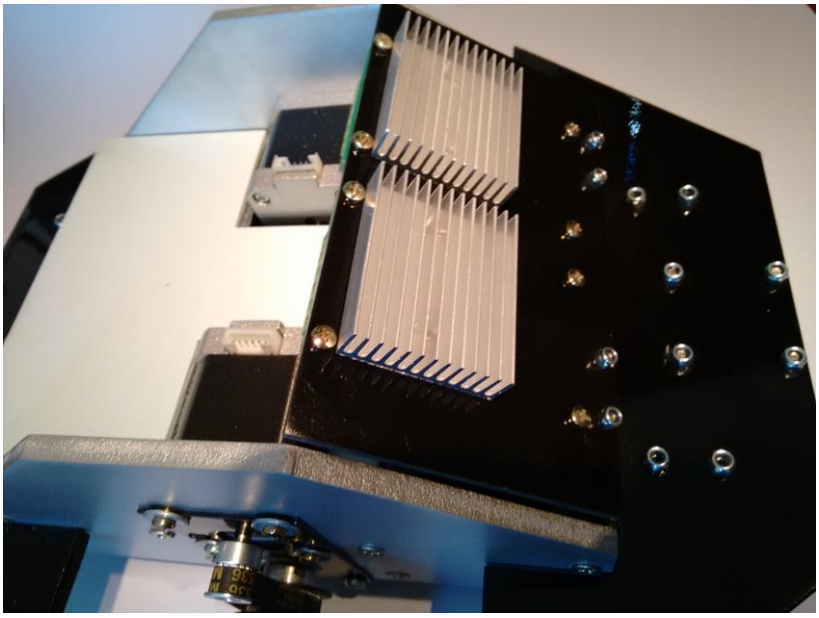
Figura 3.48 – Instalação dos controladores TB6560



Fonte: Autoria própria, 2018

A Figura 3.49 apresenta a instalação dos dissipadores de calor nos controladores.

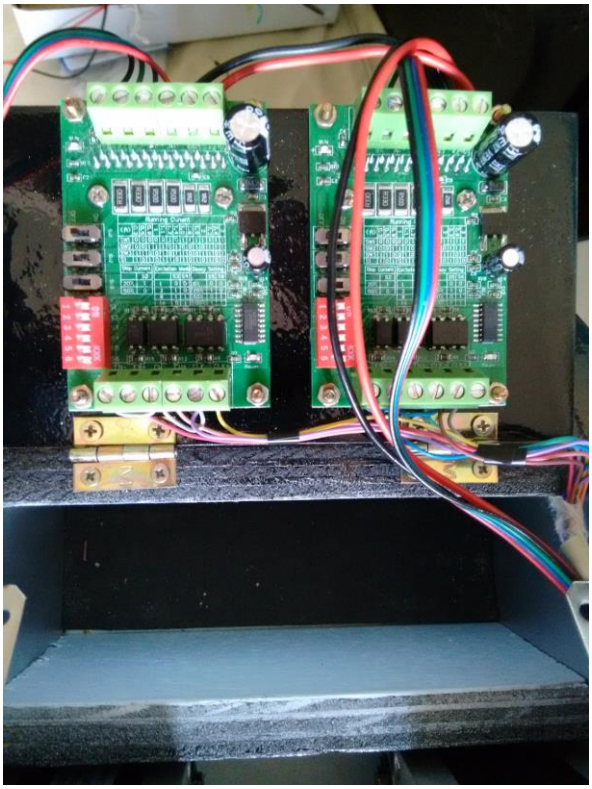
Figura 3.49 – Instalação dos dissipadores de calor



Fonte: Autoria própria, 2018

A Figura 3.50 mostra a fiação elétrica dos controladores.

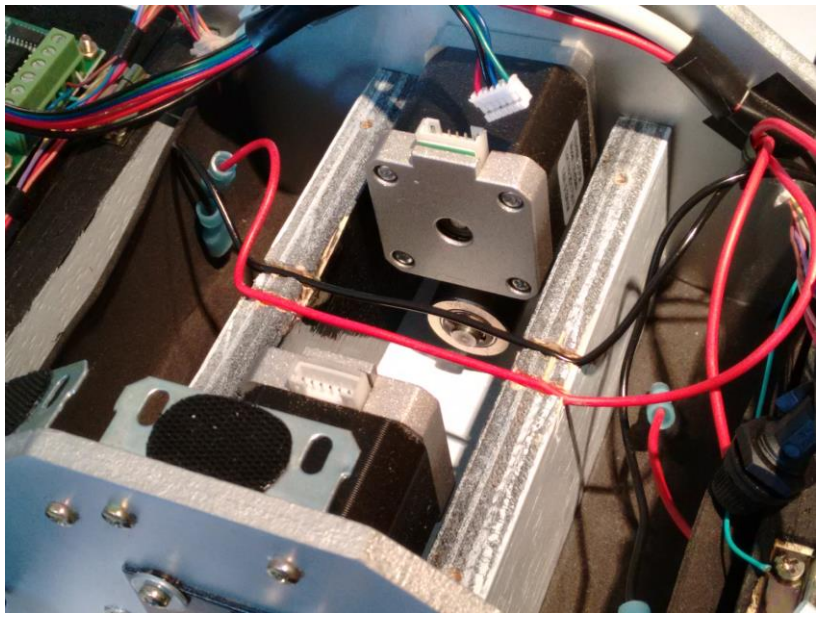
Figura 3.50 – Fiação elétrica dos controladores



Fonte: Autoria própria, 2018

A Figura 3.51 mostra a instalação dos cabos de alimentação destinados ás baterias de 12V.

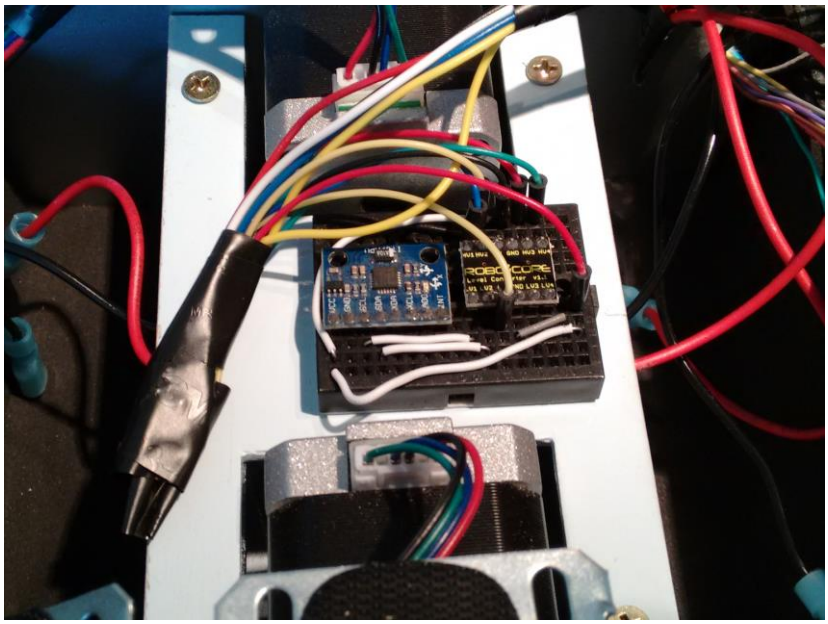
Figura 3.51 – Fiação elétrica dos cabos de alimentação



Fonte: Autoria própria, 2018

A Figura 3.52 mostra a instalação da placa central de MDF com 3mm de espessura, e sobre ela a *protoboard* com o sensor MPU-6050 e o conversor de nível lógico. A posição do sensor MPU-6050 corresponde ao centro de rotação do robô.

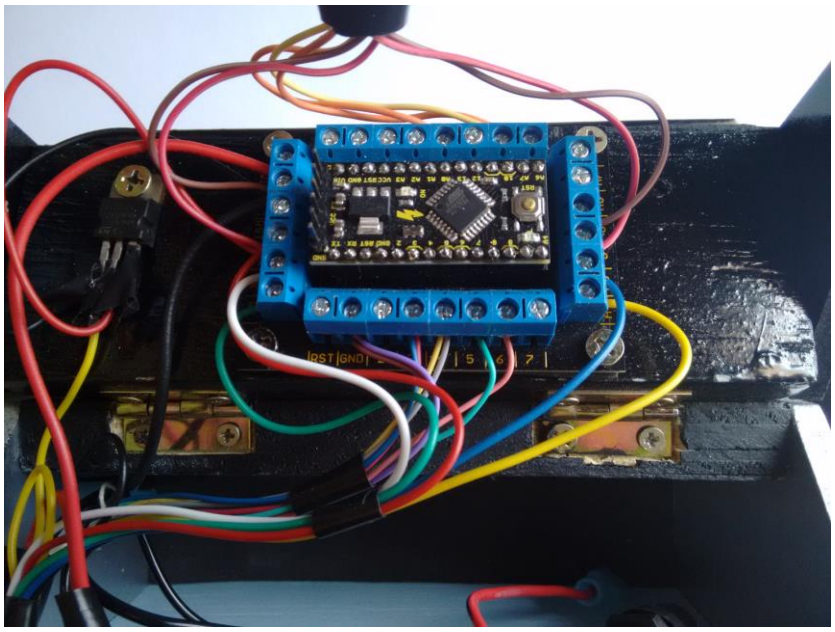
Figura 3.52 – Placa central com o *protoboard*



Fonte: Autoria própria, 2018

A Figura 3.53 mostra a instalação da placa traseira de madeira compensada, fixada por duas dobradiças, com a placa do microcontrolador e o regulador de tensão LD33.

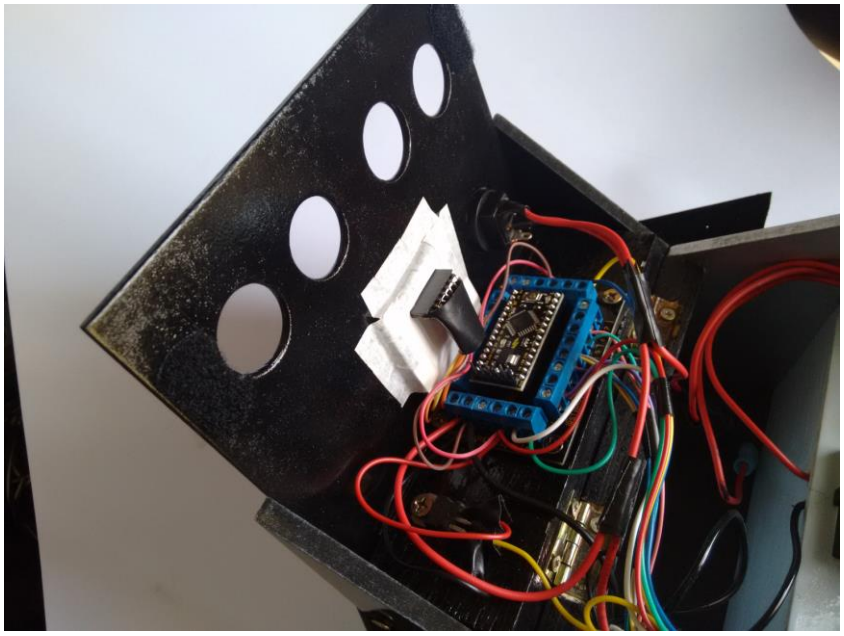
Figura 3.53 – Placa traseira com o microcontrolador e o regulador de tensão



Fonte: Autoria própria, 2018

O painel superior é instalado com o display OLED e a chave interruptora geral, conforme apresenta a Figura 3.54.

Figura 3.54 – Painel superior com o display e a chave interruptora



Fonte: Autoria própria, 2018

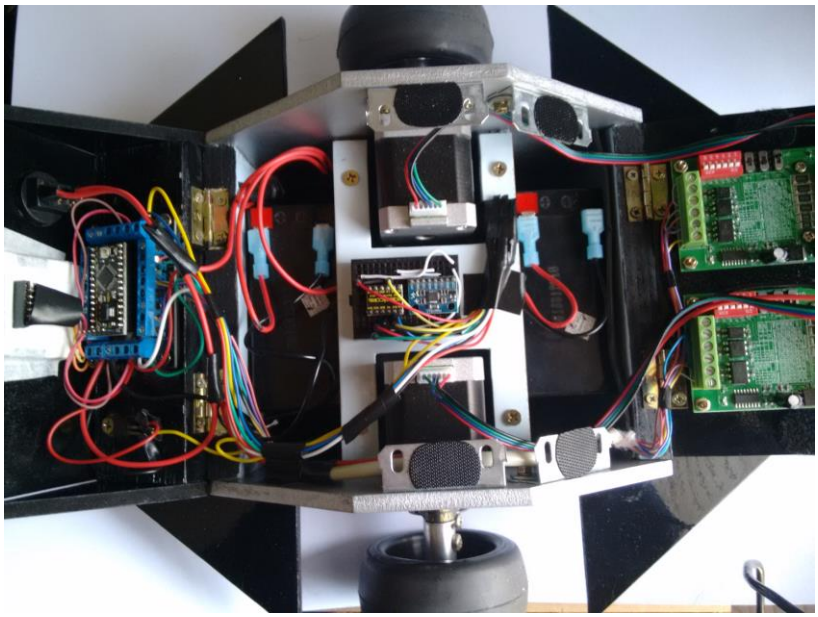
Apresenta-se na Figura 3.55 o detalhe do display OLED.

Figura 3.55 – Detalhe do display OLED

Fonte: Autoria própria, 2018

A Figura 3.56 apresenta a vista superior do conjunto montado, com as duas baterias de 12V instaladas nos compartimentos.

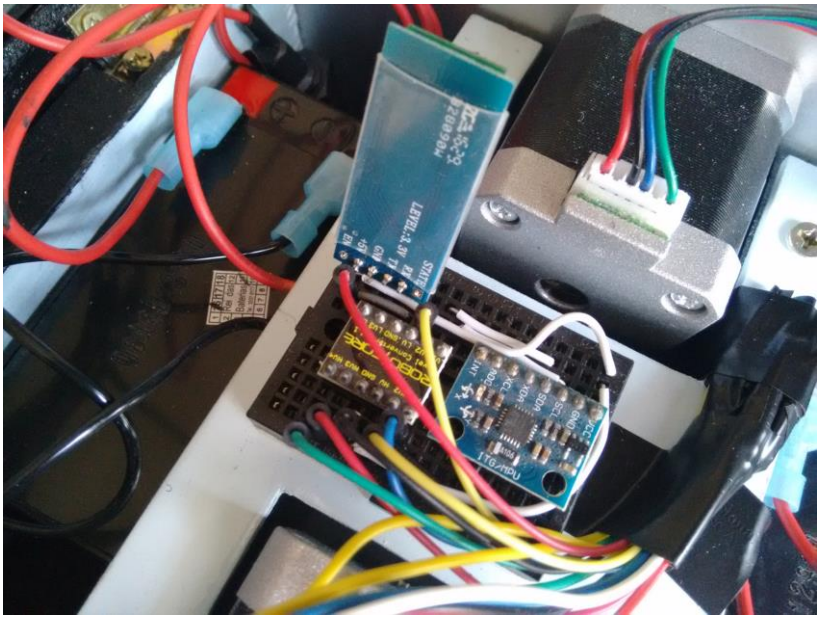
Figura 3.56 – Vista superior do conjunto montado



Fonte: Autoria própria, 2018

A Figura 3.57 apresenta o módulo Bluetooth HC-06 instalado no *protoboard.*

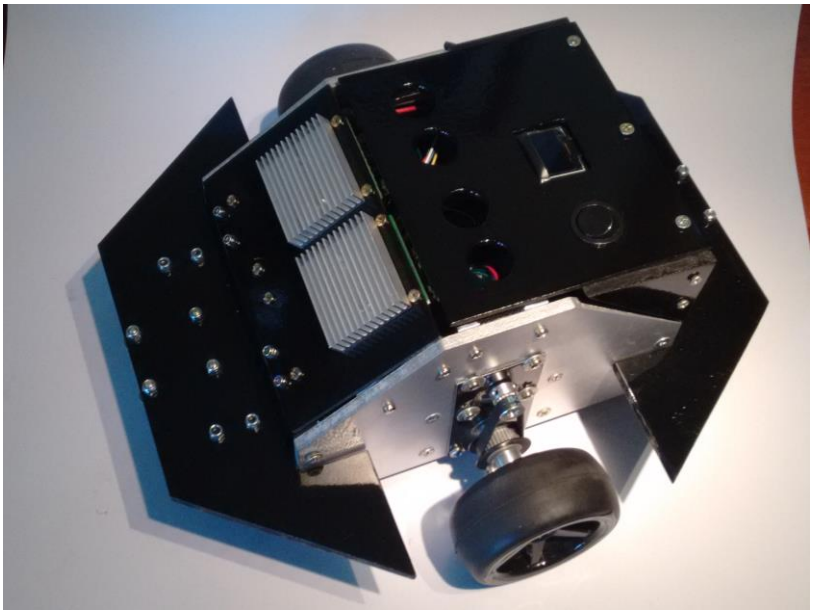
Figura 3.57 – Módulo Bluetooth HC-06 no *protoboard*



Fonte: Autoria própria, 2018

A Figura 3.58 apresenta o robô com a montagem finalizada.

Figura 3.58 – Robô finalizado



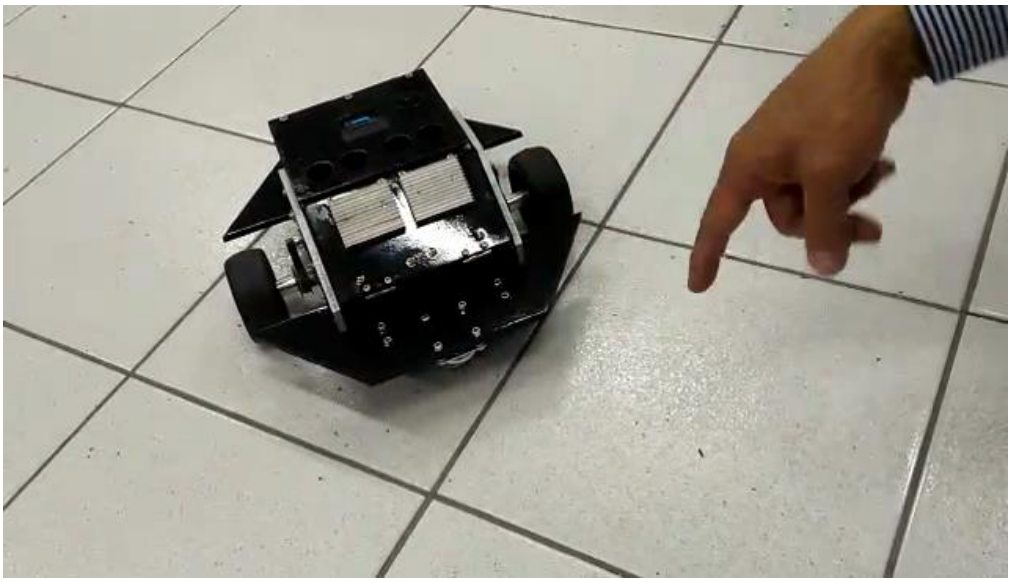
Fonte: Autoria própria, 2018

3.6 Testes e funcionamento

Os testes foram realizados nas dependências da Fatec SBC e na residência do autor.

Adota-se como referência de distância (o parâmetro *d* do algoritmo) a medida entre as placas do piso da Fatec SBC, ou seja, 31,5 cm, conforme mostra a Figura 3.59.

Figura 3.59 – Robô no piso da Fatec SBC



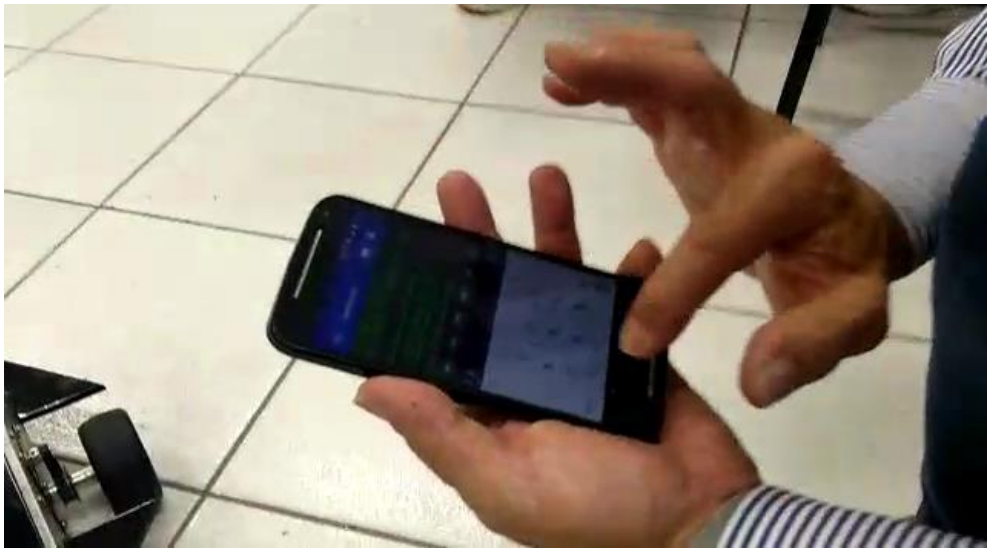
Fonte: Autoria própria, 2018

Para o deslocamento dessa distância *d,* considera-se o conjunto motor com o mecanismo de tração. O motor de passo realiza uma volta completa em seu eixo com 1600 pulsos, ajustado no controlador TB6560. A roda possui o diâmetro de 82mm. A polia conectada ao eixo do motor possui 16 dentes e a polia conectada no eixo da roda tem 36 dentes. Com essa configuração, obtemos uma redução de 1:2,25.

Considerando-se o comprimento da circunferência como pi (3,1416) multiplicado pelo diâmetro, temos: comprimento = 3,1416 x 82, ou comprimento = 257,61mm. Para a roda realizar uma volta, o motor deve receber 3600 pulsos, ou seja: 1600 pulsos x 2,25. Com esse valor, para deslocar 315mm (31,5 cm), são necessários 4402 pulsos. Esse valor é teórico, pois nos testes práticos encontrou-se o valor de 4530, que está implementado no programa.

De acordo com o algoritmo desenvolvido, o robô recebe os dados das coordenadas de origem e destino para a execução da trajetória. Esses dados são recebidos através da comunicação serial do microcontrolador. Para essa operação, utiliza-se o módulo HC-06 para conectar-se com um dispositivo móvel, ou *smartphone*, com um programa aplicativo que funciona como terminal serial padrão Bluetooth. A Figura 3.60 apresenta o uso do dispositivo móvel durante o funcionamento.

Figura 3.60 – Dispositivo móvel com o programa terminal serial



Fonte: Autoria própria, 2018

O procedimento inicial para a operação é a conexão, ou pareamento, do dispositivo móvel com o robô. Ativa-se a função Bluetooth do *smartphone* e o programa aplicativo de terminal serial. Na tela do *smartphone* aparece a mensagem “Entre com as coordenadas”. Digita-se as coordenadas no formato Xo,Yo,Xd,Yd e aperta Enter. Após isso, inicia-se o processamento do algoritmo e a movimentação do robô. Na finalização da trajetória, aparece a mensagem “TRAJETÓRIA EXECUTADA”, e em seguida, o programa está pronto para receber dados de uma nova trajetória.

CONSIDERAÇÕES FINAIS

O trabalho intitulado Sistema de orientação inercial para o robô móvel autônomo Emmy III, tem como objetivo o uso de sensores inerciais como auxílio na orientação, em conjunto com um algoritmo de movimentos. Ele é capaz de realizar uma trajetória num ambiente sem obstáculos, após o fornecimento dos dados das coordenadas de início e final do trajeto.

No projeto inicial, foi escolhido motores com caixa de redução. Porém, devido à pouca precisão nos movimentos, decidiu-se pela troca por motores de passo. A vantagem desse tipo de motor está na movimentação de forma mais precisa. A desvantagem está no baixo torque em altas rotações.

Nos movimentos rotativos, verificou-se algumas imprecisões nos movimentos. Fatores como diferenças nas dimensões dos componentes mecânicos, a precisão do sensor inercial utilizado e ondulações no piso, podem influenciar nos movimentos.

No sentido mais amplo, os testes realizados no protótipo validaram o algoritmo desenvolvido.

Uma das aplicações para esse sistema seria o controle de veículos autônomos industriais, também chamados de *AGV – Automated guided vehicle*, ou veículo guiado automaticamente.

Como possíveis implementações futuras, podemos citar o uso de outros sensores como bússola eletrônica, sistemas de posicionamento global (GPS), tornando o conjunto mais preciso nos movimentos.

REFERÊNCIAS

DA SILVA FILHO, J. I.; TORRES, C. R.; ABE, J. M. Robô Móvel Autônomo Emmy: Uma Aplicação Eficiente da Lógica Paraconsistente Anotada. **Seleção documental**, ISSN 1809-0648. Santos – São Paulo: Editora ParaLogike, ano 1, n.3, p. 19-26, Julho-Setembro. 2006.

HALLIDAY, D.; RESNICK, R.; WALKER, J. **Fundamentos de física**, volume 1: mecânica. 9. ed. Rio de Janeiro: LTC, 2014.

INVENSENSE **MPU-6000 and MPU-6050 Product specification revision 3.4**, 2013. Disponível em <https://store.invensense.com/datasheets/invensense/MPU-6050_DataSheet_V3%204.pdf>. Acesso em: 01 set. 2017.

MATARIĆ, M. J. **Introdução à robótica**. 1. ed. São Paulo: Unesp Blucher, 2014.

MORI, A.M. **Uso do Sistema inercial para apoiar a navegação autônoma**. 2013. 180 p. Dissertação (Mestrado) – Escola Politécnica da Universidade de São Paulo – USP. São Paulo. 2013.

ROSÁRIO, J. M. **Princípios de mecatrônica**. 4. ed. São Paulo: Prentice Hall, 2009.

SEVERINO, A.J. **Metodologia do trabalho científico**. 23. ed. São Paulo: Cortez, 2007.

THOMAZINI, D.; ALBUQUERQUE, P. U. B. **Sensores Industriais:** fundamentos e aplicações. 3. ed. São Paulo: Érica, 2007.

TORRES, C. R. **Sistema inteligente baseado na lógica paraconsistente anotada evidencial Eᴛ para controle e navegação de robôs móveis autônomos em um ambiente não estruturado**. 2010. 193 p. Tese (Doutorado) - Universidade Federal de Itajubá - UNIFEI, Itajubá, 2010.

WILLIAMS, E. **Make: AVR Programming**. 3. ed. San Francisco: Maker Media, 2015.

APÊNDICE A – PROGRAMA IMPLEMENTADO NO MICROCONTROLADOR

Nesta seção apresenta-se o código-fonte do programa implementado no microcontrolador.

```
/*
 * Projeto - TCC FATEC SBC
 *
 * Programa principal
 *
 * Alexandre Noboru Fukuda
 *
 * 09/09/2017 - versão m328p
 *
 * 15/05/18 - VERSAO OLED SPI U8X8
 */

// MPU-6050 Accelerometer + Gyro
// -----------------------------
//
// By arduino.cc user "Krodal".
// June 2012
// Open Source / Public Domain
//
// Using Arduino 1.0.1
// It will not work with an older version,
// since Wire.endTransmission() uses a parameter
// to hold or release the I2C bus.
//
// Documentation:
// - The InvenSense documents:
//   - "MPU-6000 and MPU-6050 Product Specification",
//     PS-MPU-6000A.pdf
//   - "MPU-6000 and MPU-6050 Register Map and Descriptions",
//     RM-MPU-6000A.pdf or RS-MPU-6000A.pdf
//   - "MPU-6000/MPU-6050 9-Axis Evaluation Board User Guide"
//     AN-MPU-6000EVB.pdf
//
// The accuracy is 16-bits.
//
// Temperature sensor from -40 to +85 degrees Celsius
//   340 per degrees, -512 at 35 degrees.
//
// At power-up, all registers are zero, except these two:
//      Register 0x6B (PWR_MGMT_2) = 0x40  (I read zero).
//      Register 0x75 (WHO_AM_I)   = 0x68.
//
//


#include <Arduino.h>
#include <U8x8lib.h>

#include <SPI.h>
#include <Wire.h>

U8X8_SSD1306_128X64_NONAME_4W_SW_SPI u8x8(/* clock=*/ 13, /* data=*/ 11, /* cs=*/ 10, /* dc=*/ 9, /* reset=*/ 8);

// The name of the sensor is "MPU-6050".
// For program code, I omit the '-',
// therefor I use the name "MPU6050....".


// Register names according to the datasheet.
// According to the InvenSense document
// "MPU-6000 and MPU-6050 Register Map
// and Descriptions Revision 3.2", there are no registers
// at 0x02 ... 0x18, but according other information
// the registers in that unknown area are for gain
// and offsets.
//
```

```
#define MPU6050_ACCEL_XOUT_H       0x3B   // R
#define MPU6050_ACCEL_XOUT_L       0x3C   // R
#define MPU6050_ACCEL_YOUT_H       0x3D   // R
#define MPU6050_ACCEL_YOUT_L       0x3E   // R
#define MPU6050_ACCEL_ZOUT_H       0x3F   // R
#define MPU6050_ACCEL_ZOUT_L       0x40   // R

#define MPU6050_PWR_MGMT_1         0x6B   // R/W
#define MPU6050_PWR_MGMT_2         0x6C   // R/W

#define MPU6050_WHO_AM_I           0x75   // R


// Default I2C address for the MPU-6050 is 0x68.
// But only if the AD0 pin is low.
// Some sensor boards have AD0 high, and the
// I2C address thus becomes 0x69.
#define MPU6050_I2C_ADDRESS 0x68


// Declaring an union for the registers and the axis values.
// The byte order does not match the byte order of
// the compiler and AVR chip.
// The AVR chip (on the Arduino board) has the Low Byte
// at the lower address.
// But the MPU-6050 has a different order: High Byte at
// lower address, so that has to be corrected.
// The register part "reg" is only used internally,
// and are swapped in code.
typedef union accel_t_gyro_union
{
  struct
  {
    uint8_t x_accel_h;
    uint8_t x_accel_l;
    uint8_t y_accel_h;
    uint8_t y_accel_l;
    uint8_t z_accel_h;
    uint8_t z_accel_l;
    uint8_t t_h;
    uint8_t t_l;
    uint8_t x_gyro_h;
    uint8_t x_gyro_l;
    uint8_t y_gyro_h;
    uint8_t y_gyro_l;
    uint8_t z_gyro_h;
    uint8_t z_gyro_l;
  } reg;
  struct
  {
    int x_accel;
    int y_accel;
    int z_accel;
    int temperature;
    int x_gyro;
    int y_gyro;
    int z_gyro;
  } value;
};

// Use the following global variables and access functions to help store the overall
// rotation angle of the sensor
unsigned long last_read_time;
float         last_x_angle;  // These are the filtered angles
float         last_y_angle;
float         last_z_angle;
float         last_gyro_x_angle;  // Store the gyro angles to compare drift
float         last_gyro_y_angle;
float         last_gyro_z_angle;

void set_last_read_angle_data(unsigned long time, float x, float y, float z, float x_gyro, float y_gyro, float z_gyro) {
  last_read_time = time;
  last_x_angle = x;
  last_y_angle = y;
  last_z_angle = z;
  last_gyro_x_angle = x_gyro;
  last_gyro_y_angle = y_gyro;
```

```
  last_gyro_z_angle = z_gyro;
}

inline unsigned long get_last_time() {return last_read_time;}
inline float get_last_x_angle() {return last_x_angle;}
inline float get_last_y_angle() {return last_y_angle;}
inline float get_last_z_angle() {return last_z_angle;}
inline float get_last_gyro_x_angle() {return last_gyro_x_angle;}
inline float get_last_gyro_y_angle() {return last_gyro_y_angle;}
inline float get_last_gyro_z_angle() {return last_gyro_z_angle;}

//  Use the following global variables and access functions
//  to calibrate the acceleration sensor
float    base_x_accel;
float    base_y_accel;
float    base_z_accel;

float    base_x_gyro;
float    base_y_gyro;
float    base_z_gyro;

//int calibra = 0;

int read_gyro_accel_vals(uint8_t* accel_t_gyro_ptr) {
  // Read the raw values.
  // Read 14 bytes at once,
  // containing acceleration, temperature and gyro.
  // With the default settings of the MPU-6050,
  // there is no filter enabled, and the values
  // are not very stable.  Returns the error value

  accel_t_gyro_union* accel_t_gyro = (accel_t_gyro_union *) accel_t_gyro_ptr;

  int error = MPU6050_read (MPU6050_ACCEL_XOUT_H, (uint8_t *) accel_t_gyro, sizeof(*accel_t_gyro));

  // Swap all high and low bytes.
  // After this, the registers values are swapped,
  // so the structure name like x_accel_l does no
  // longer contain the lower byte.
  uint8_t swap;
  #define SWAP(x,y) swap = x; x = y; y = swap

  SWAP ((*accel_t_gyro).reg.x_accel_h, (*accel_t_gyro).reg.x_accel_l);
  SWAP ((*accel_t_gyro).reg.y_accel_h, (*accel_t_gyro).reg.y_accel_l);
  SWAP ((*accel_t_gyro).reg.z_accel_h, (*accel_t_gyro).reg.z_accel_l);
  SWAP ((*accel_t_gyro).reg.t_h, (*accel_t_gyro).reg.t_l);
  SWAP ((*accel_t_gyro).reg.x_gyro_h, (*accel_t_gyro).reg.x_gyro_l);
  SWAP ((*accel_t_gyro).reg.y_gyro_h, (*accel_t_gyro).reg.y_gyro_l);
  SWAP ((*accel_t_gyro).reg.z_gyro_h, (*accel_t_gyro).reg.z_gyro_l);

  return error;
}

// The sensor should be motionless on a horizontal surface
//  while calibration is happening
void calibrate_sensors() {
  int                 num_readings = 20;
  float               x_accel = 0;
  float               y_accel = 0;
  float               z_accel = 0;
  float               x_gyro = 0;
  float               y_gyro = 0;
  float               z_gyro = 0;
  accel_t_gyro_union    accel_t_gyro;


  // Discard the first set of values read from the IMU
  read_gyro_accel_vals((uint8_t *) &accel_t_gyro);

  // Read and average the raw values from the IMU
  for (int i = 0; i < num_readings; i++) {
    read_gyro_accel_vals((uint8_t *) &accel_t_gyro);
    x_accel += accel_t_gyro.value.x_accel;
    y_accel += accel_t_gyro.value.y_accel;
    z_accel += accel_t_gyro.value.z_accel;
    x_gyro += accel_t_gyro.value.x_gyro;
    y_gyro += accel_t_gyro.value.y_gyro;
```

```
    z_gyro += accel_t_gyro.value.z_gyro;

    delay(60);//60
  }

  x_accel /= num_readings;
  y_accel /= num_readings;
  z_accel /= num_readings;
  x_gyro /= num_readings;
  y_gyro /= num_readings;
  z_gyro /= num_readings;

  // Store the raw calibration values globally
  base_x_accel = x_accel;
  base_y_accel = y_accel;
  base_z_accel = z_accel;
  base_x_gyro = x_gyro;
  base_y_gyro = y_gyro;
  base_z_gyro = z_gyro;

  delay(100);
}

///////////////
// Pin list //
/////////////

const int connect_led_pin = 13;  // pin used for connect status LED. Doubles up as Calibration failed

bool blinkState = false;

float heading;
unsigned long microsNow;

int COD_ATL=1; // Codigo Atual (Código inicial =1)
int COD_REC;   // Codigo Recebido do Subsistema de Planejamento
int PASSOS;    // Passos de 45 graus
int ANG_ROT;   // Ângulo de rotação
int ROTA;      // Sentido de rotacao
int MOV_RETA;  // Variável para movimento retilíneo
int it;        // Contador inicial de Setup
int t;         // Delay do clock TB6560
int i;         // Contador de clock

int t1=700;
int t2=400;
//int td=1000;
int d = 4530;   // Distância d
//int pdiag = 6406;
int tp; // contador de tempo

// Pinos de controle do driver TB6560

const int ena = 3; // Enable TB6560
const int clk = 4; // Clock TB6560
const int dir1 = 5; // Direção 1 TB6560
const int dir2 = 6; // Direção 2 TB6560

unsigned long microsPerReading, microsPrevious;
float accelScale, gyroScale;
int xo,yo,xd,yd,fim=0,Mov;  // Coordenadas e variáveis para rotina de planejamento
String Trajeto, CodPlan;

////////////////////////////////

void setup() {

  int error;
  uint8_t c;

  Serial.begin(9600);

    pinMode( ena, OUTPUT);
    pinMode( clk, OUTPUT);
    pinMode( dir1, OUTPUT);
    pinMode( dir2, OUTPUT);
```

```
    digitalWrite( 3, LOW); // enable
    digitalWrite( clk, LOW);  //clk
    digitalWrite( dir1, LOW);  //dir1
    digitalWrite( dir2, LOW);  //dir2

  // OLED - Tela inicial

    u8x8.begin();
    u8x8.setPowerSave(0);
   // u8x8.setFont(u8x8_font_chroma48medium8_r);
    u8x8.setFont(u8x8_font_amstrad_cpc_extended_f);
    u8x8.draw1x2String(4,0,"FATEC-SBC");
    u8x8.setCursor(0,3);
    u8x8.print("ALEXANDRE NOBORU");
    u8x8.setCursor(5,4);
    u8x8.print("FUKUDA");
    u8x8.setCursor(6,6);
    u8x8.print("2018");

    delay(3000);

  // Initialize the 'Wire' class for the I2C-bus.
  Wire.begin();

  // default at power-up:
  //    Gyro at 250 degrees second
  //    Acceleration at 2g
  //    Clock source at internal 8MHz
  //    The device is in sleep mode.
  //

  error = MPU6050_read (MPU6050_WHO_AM_I, &c, 1);

  // According to the datasheet, the 'sleep' bit
  // should read a '1'. But I read a '0'.
  // That bit has to be cleared, since the sensor
  // is in sleep mode at power-up. Even if the
  // bit reads '0'.
  error = MPU6050_read (MPU6050_PWR_MGMT_2, &c, 1);

  // Clear the 'sleep' bit to start the sensor.
  MPU6050_write_reg (MPU6050_PWR_MGMT_1, 0);

  //Initialize the angles
  calibrate_sensors();
  set_last_read_angle_data(millis(), 0, 0, 0, 0, 0, 0);
  delay(200);

}


void loop() {


// ===========================================
// ===  ENTRADA DE COORDENADAS Xo,Yo,Xd,Yd  ===
// ===========================================

  u8x8.clear();
//  u8x8.setFont(u8x8_font_chroma48medium8_r);
 /*
  u8x8.setCursor(3,3);
  u8x8.print("AGUARDANDO");
  u8x8.setCursor(3,4);
  u8x8.print("COORDENADAS");
*/

  u8x8.draw1x2String(3,2,"AGUARDANDO");
  u8x8.draw1x2String(3,4,"COORDENADAS");

  Serial.println("Entre com as coordenadas");

   while (Serial.available() == 0) {
   }

    xo = Serial.parseInt();
    yo = Serial.parseInt();
```

```
  xd = Serial.parseInt();
  yd = Serial.parseInt();

  if (Serial.read() == '\n') {

    Serial.print(xo); Serial.print(" ");
    Serial.print(yo); Serial.print(" ");
    Serial.print(xd); Serial.print(" ");
    Serial.println(yd);
  }

    u8x8.clear();
   // u8x8.setFont(u8x8_font_chroma48medium8_r);
    u8x8.setFont(u8x8_font_amstrad_cpc_extended_f);

    u8x8.setCursor(0,0);
    u8x8.print("INICIO (");
    u8x8.print(xo);  u8x8.print(",");
    u8x8.print(yo);  u8x8.print(")");
    u8x8.setCursor(0,1);
    u8x8.print("FINAL  (");
    u8x8.print(xd);  u8x8.print(",");
    u8x8.print(yd);  u8x8.print(")");

fim = 0;

  do {
    SisPlan();
    Trajeto += String(Mov);
  //  Serial.print(Mov);
  //  Serial.print(".");
  } while (fim != 1);

int comp = Trajeto.length()-2;

// Execucao dos codigos processados

 digitalWrite( 3, LOW); // enable

for (int i=comp; i>=0; i--) {

  CodPlan = Trajeto.charAt(i);

  Serial.println(" ");

  if(CodPlan == "1" ) {
    COD_REC = 1;
    rotina_principal();
  }
  if(CodPlan == "2" ){
    COD_REC = 2;
    rotina_principal();
  }
   if(CodPlan == "3" ){
    COD_REC = 3;
    rotina_principal();
  }
   if(CodPlan == "4" ){
    COD_REC = 4;
    rotina_principal();
  }
   if(CodPlan == "5" ){
    COD_REC = 5;
    rotina_principal();
  }
   if(CodPlan == "6" ){
    COD_REC = 6;
    rotina_principal();
  }
   if(CodPlan == "7" ){
    COD_REC = 7;
    rotina_principal();
  }
   if(CodPlan == "8" ){
    COD_REC = 8;
    rotina_principal();
  }
```

```
  }

  Trajeto="" ;

    Serial.println(" ");
    Serial.println("*****************************");
    Serial.println("    TRAJETORIA EXECUTADA");
    Serial.println("*****************************");

    u8x8.draw1x2String(1,3,"            ");
    u8x8.draw1x2String(1,5,"            ");

    u8x8.draw1x2String(2,3,"TRAJET\xd3RIA");
    u8x8.draw1x2String(2,5,"EXECUTADA");

  delay(1000);

}

// ================================================================
// ===            PROGRAMA PRINCIPAL                          ===
// ================================================================

 void rotina_principal() {

    PASSOS = COD_REC - COD_ATL ; // Define o numero de passos de 45 graus

    if( PASSOS > 4) {
      PASSOS = PASSOS -8 ;
    }
    if( PASSOS < -3) {
      PASSOS = PASSOS +8 ;
    }

    if( PASSOS > 0) {
      ROTA = 'H';
    }
    if( PASSOS < 0) {
      ROTA = 'A';
    }
    if( PASSOS == 0) {
      ROTA = '0';
    }

    Serial.print("Cod.ATUAL:");
    Serial.print(COD_ATL);

    Serial.print(" Cod.REC.:");
    Serial.println(COD_REC);

    Serial.print("No.ANG_ROT:");
    Serial.print(PASSOS);

    Serial.print(" Rotacao :");
    Serial.println(char(ROTA));
    Serial.println("");


// ================================================================
//                ROTINA DOS MOVIMENTOS DE ROTAÇÃO
// ================================================================

    if(ROTA == '0') {
      motor_reta();
    }
    else {

      delay(200);

      t=330;  tp=0;

    Serial.println("Executando Rotacao ");

      u8x8.setCursor(3,3);
      u8x8.print("EXECUTANDO");
      u8x8.setCursor(4,4);
      u8x8.print("ROTACAO");
```

```
      u8x8.setCursor(1,6);
      u8x8.print("ANG.ROT:");

    ANG_ROT = PASSOS * 45;

    motor_rotacao();

     set_last_read_angle_data(millis(), 0, 0, 0, 0, 0, 0);

     leitura_inercial();

     Serial.print("ANGULO:");
     Serial.println(ANG_ROT);

    //leitura_inercial();

   // delay(200);

      do {
      tp++;
      if(tp>10) {
        leitura_inercial();
        tp=0;
      }

      execClock();

      setVelRota();

      // OLED - VALOR DO ÂNGULO DE ROTAÇÃO
/*
      u8x8.setCursor(9,6);

      if(heading >=0) {
        if(heading < 10) {
          u8x8.print("    ");
        }
        if(heading>=10 && heading<100) {
          u8x8.print("   ");
        }
        if(heading>=100) {
          u8x8.print("  ");
        }
      }
      else {
        if(heading>-10 && heading <0) {
          u8x8.print("   ");
        }
        if(heading>-100 && heading<=-10) {
          u8x8.print("  ");
        }
        if(heading<=-100) {
          u8x8.print(" ");
        }
      }

      u8x8.print(int(heading));
      u8x8.print("  ");
*/

      } while (ANG_ROT != heading);


      Serial.println("ROTAÇÃO EXECUTADO ");

      u8x8.setCursor(3,3);
      u8x8.print("          ");
      u8x8.setCursor(4,4);
      u8x8.print("       ");

      delay(300);
      motor_reta();

      }

       COD_ATL = COD_REC;
}
```

```
// ================================================================
// ===      FIM DO PROGRAMA PRINCIPAL                          ===
// ================================================================



// ================================================================
//          ROTINA DE CONTROLE DOS MOTORES - ROTACAO
// ================================================================

void motor_rotacao() {


/************************** HORÁRIO ****************************/
//
                if (ROTA == 'H') {

                  digitalWrite(dir1,LOW );    digitalWrite(dir2,LOW );
    delay(50);
                }

/************************** ANTI HORÁRIO ***********************/
//
   else if (ROTA =='A') {

    digitalWrite(dir1,HIGH);    digitalWrite(dir2,HIGH);
    delay(50);
   }


}


// ================================================================
//          ROTINA DE CLOCK DO MOTOR DE PASSO
// ================================================================

void execClock() {


    digitalWrite( clk, HIGH);
    delayMicroseconds(t);
    digitalWrite( clk, LOW);
    delayMicroseconds(t);

}

// ================================================================
//          ROTINA DE CONTROLE DOS MOTORES - RETA
// ================================================================

void motor_reta() {

 digitalWrite( 3, LOW); // enable
 t = t2;

 if(COD_REC == 1 || COD_REC == 3 || COD_REC == 5 || COD_REC == 7) {

   MOV_RETA = d; // MOV_RETA

 }
 else {

   MOV_RETA = d * 1.414;  // MOV_RETA x 1,414
 }

 //
   digitalWrite( dir1, HIGH); // dir1
   digitalWrite( dir2, LOW ); // dir2


   for(i=1; i<=MOV_RETA; i++) {

    if((i<400)||(i>=(MOV_RETA-300))) {
     acel();
    }
```

```
      else {
        t=t2;
      }

     execClock();

    }

  Serial.println("MOVIMENTO RETILÍNEO EXECUTADO");

}

void acel()
{
  if (i<400) {

   t = t1 - i*1.1;

  }
  else {
    t = t2 + i*1.1 - (MOV_RETA -300);
  }
}

// ================================================================
// ===     ROTINA DE VELOCIDADE DA ROTACAO - 23/04/17          ===
// ===     Adaptação para m328p - 09/09/2017
// ================================================================

void setVelRota() {

 // ANG_ROT = PASSOS * 45;

  if (ANG_ROT < 0) {
    if (heading <= ANG_ROT+15) {

     t = 1465;
    }
  }
  if (ANG_ROT > 0) {
    if (heading >= ANG_ROT-15) {

     t = 1465;
    }
  }

}


// ================================================================
// ===              ROTINA - LEITURA INERCIAL                  ===
// ================================================================

void leitura_inercial() {

  int error;
  double dT;
  accel_t_gyro_union accel_t_gyro;

  // Read the raw values.
  error = read_gyro_accel_vals((uint8_t*) &accel_t_gyro);

  // Get the time of reading for rotation computations
  unsigned long t_now = millis();

  // Convert gyro values to degrees/sec
  float FS_SEL = 131;
 /*
  float gyro_x = (accel_t_gyro.value.x_gyro - base_x_gyro)/FS_SEL;
  float gyro_y = (accel_t_gyro.value.y_gyro - base_y_gyro)/FS_SEL;
  float gyro_z = (accel_t_gyro.value.z_gyro - base_z_gyro)/FS_SEL;
*/

  float gyro_x = 0;
  float gyro_y = 0;
  float gyro_z = (accel_t_gyro.value.z_gyro - base_z_gyro)/FS_SEL;
```

```
  // Get raw acceleration values
  //float G_CONVERT = 16384;
 /*
  float accel_x = accel_t_gyro.value.x_accel;
  float accel_y = accel_t_gyro.value.y_accel;
  float accel_z = accel_t_gyro.value.z_accel;
  */
  float accel_x = 0;
  float accel_y = 0;
  float accel_z = accel_t_gyro.value.z_accel;

  // Get angle values from accelerometer

  float RADIANS_TO_DEGREES = 180/3.14159265358979323846264338327950288;
 /*
  float accel_angle_y = atan(-1*accel_x/sqrt(pow(accel_y,2) + pow(accel_z,2)))*RADIANS_TO_DEGREES;
  float accel_angle_x = atan(accel_y/sqrt(pow(accel_x,2) + pow(accel_z,2)))*RADIANS_TO_DEGREES;

  float accel_angle_z = 0;
  */
  // Compute the (filtered) gyro angles
  float dt =(t_now - get_last_time())/1000.0;
/*
  float gyro_angle_x = gyro_x*dt + get_last_x_angle();
  float gyro_angle_y = gyro_y*dt + get_last_y_angle();
  float gyro_angle_z = gyro_z*dt + get_last_z_angle();
*/
  float gyro_angle_x = 0;
  float gyro_angle_y = 0;
  float gyro_angle_z = gyro_z*dt + get_last_z_angle();

  // Compute the drifting gyro angles
 /*
  float unfiltered_gyro_angle_x = gyro_x*dt + get_last_gyro_x_angle();
  float unfiltered_gyro_angle_y = gyro_y*dt + get_last_gyro_y_angle();
  float unfiltered_gyro_angle_z = gyro_z*dt + get_last_gyro_z_angle();
*/
  float unfiltered_gyro_angle_x = 0;
  float unfiltered_gyro_angle_y = 0;
  float unfiltered_gyro_angle_z = gyro_z*dt + get_last_gyro_z_angle();

  // Apply the complementary filter to figure out the change in angle - choice of alpha is
  // estimated now.  Alpha depends on the sampling rate...
 // float alpha = 0.968;
  // float alpha = 0.96;
  //float alpha = 0.96;
 /*
  float angle_x = alpha*gyro_angle_x + (1.0 - alpha)*accel_angle_x;
  float angle_y = alpha*gyro_angle_y + (1.0 - alpha)*accel_angle_y;
  float angle_z = gyro_angle_z;  //Accelerometer doesn't give z-angle
*/

  float angle_x = 0;
  float angle_y = 0;
  float angle_z = gyro_angle_z;  //Accelerometer doesn't give z-angle

  // Update the saved data with the latest values
  set_last_read_angle_data(t_now, angle_x, angle_y, angle_z, unfiltered_gyro_angle_x, unfiltered_gyro_angle_y,
unfiltered_gyro_angle_z);

  // ÂNGULO DE ROTAÇÃO

  heading = int(-angle_z);

      u8x8.setCursor(9,6);

      if(heading >=0) {
        if(heading < 10) {
          u8x8.print("    ");
        }
        if(heading>=10 && heading<100) {
          u8x8.print("   ");
        }
        if(heading>=100) {
          u8x8.print("  ");
        }
      }
```

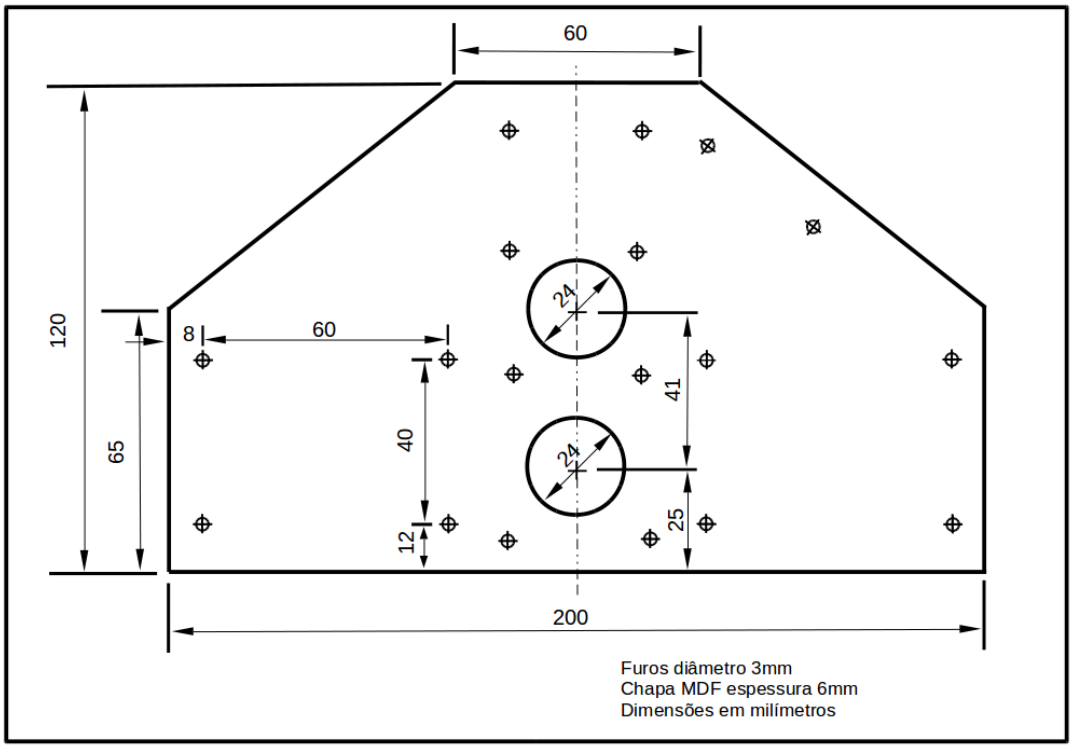
```
      else {
        if(heading>-10 && heading <0) {
          u8x8.print("   ");
        }
        if(heading>-100 && heading<=-10) {
          u8x8.print("  ");
        }
        if(heading<=-100) {
          u8x8.print(" ");
        }
      }

      u8x8.print(int(heading));
      u8x8.print("  ");

}


// ================================================================
// ===          ROTINA DE PLANEJAMENTO DE TRAJETÓRIA          ===
// ================================================================

void SisPlan() {

  // Primeiro IF
  if(yd >= yo) {

    // Segundo IF
    if(xd == xo) {

      // rotina 1

      if(yd == yo) {
        fim = 1;
      }
      else {
        Mov = 1;
        yd = yd - 1;
      }
    }

      // Terceiro IF
    else if(yd == yo) {
        if(xd > xo) {
          Mov = 3;
          xd = xd - 1;
        }
        else {
          Mov = 7;
          xd = xd + 1;
        }
      }

      // Quarto IF
    else if(xd > xo) {
        Mov = 2;
        xd = xd - 1;
        yd = yd - 1;
      }
      else {
        Mov = 8;
        xd = xd + 1;
        yd = yd - 1;
      }
  }
  else {

    // rotina 2

    if(xd == xo) {
      Mov = 5;
      yd = yd + 1;
    }
    else {
      if(xd < xo) {
        Mov = 6;
        xd = xd + 1;
```

```
      yd = yd + 1;
     }
     else {
      Mov = 4;
      xd = xd - 1;
      yd = yd + 1;
     }
    }
   }
}


// --------------------------------------------------------
// MPU6050_read
//
// This is a common function to read multiple bytes
// from an I2C device.
//
// It uses the boolean parameter for Wire.endTransMission()
// to be able to hold or release the I2C-bus.
// This is implemented in Arduino 1.0.1.
//
// Only this function is used to read.
// There is no function for a single byte.
//
int MPU6050_read(int start, uint8_t *buffer, int size)
{
  int i, n, error;

  Wire.beginTransmission(MPU6050_I2C_ADDRESS);
  n = Wire.write(start);
  if (n != 1)
    return (-10);

  n = Wire.endTransmission(false);    // hold the I2C-bus
  if (n != 0)
    return (n);

  // Third parameter is true: relase I2C-bus after data is read.
  Wire.requestFrom(MPU6050_I2C_ADDRESS, size, true);
  i = 0;
  while(Wire.available() && i<size)
  {
    buffer[i++]=Wire.read();
  }
  if ( i != size)
    return (-11);

  return (0);  // return : no error
}


// --------------------------------------------------------
// MPU6050_write
//
// This is a common function to write multiple bytes to an I2C device.
//
// If only a single register is written,
// use the function MPU_6050_write_reg().
//
// Parameters:
//   start : Start address, use a define for the register
//   pData : A pointer to the data to write.
//   size  : The number of bytes to write.
//
// If only a single register is written, a pointer
// to the data has to be used, and the size is
// a single byte:
//   int data = 0;        // the data to write
//   MPU6050_write (MPU6050_PWR_MGMT_1, &c, 1);
//
int MPU6050_write(int start, const uint8_t *pData, int size)
{
  int n, error;

  Wire.beginTransmission(MPU6050_I2C_ADDRESS);
  n = Wire.write(start);        // write the start address
```

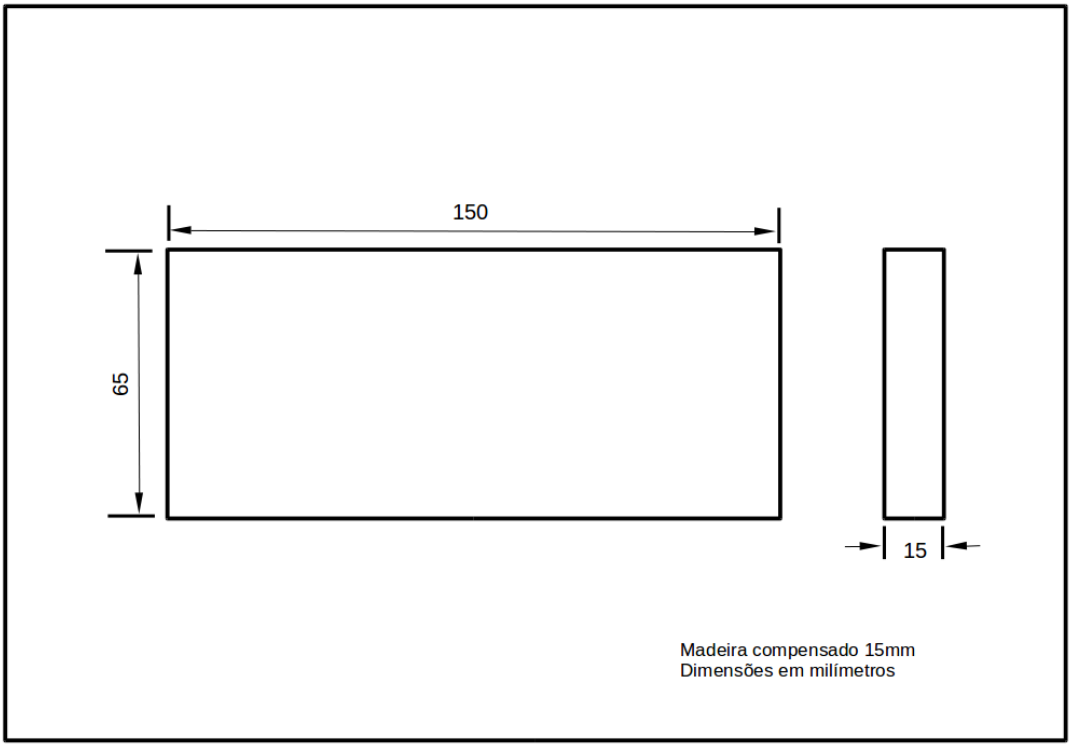
```
  if (n != 1)
    return (-20);

  n = Wire.write(pData, size);  // write data bytes
  if (n != size)
    return (-21);

  error = Wire.endTransmission(true); // release the I2C-bus
  if (error != 0)
    return (error);

  return (0);         // return : no error
}

// ----------------------------------------------------------
// MPU6050_write_reg
//
// An extra function to write a single register.
// It is just a wrapper around the MPU_6050_write()
// function, and it is only a convenient function
// to make it easier to write a single register.
//
int MPU6050_write_reg(int reg, uint8_t data)
{
  int error;

  error = MPU6050_write(reg, &data, 1);

  return (error);
}
```

APÊNDICE B – DESENHOS TÉCNICOS DA ESTRUTURA MECÂNICA

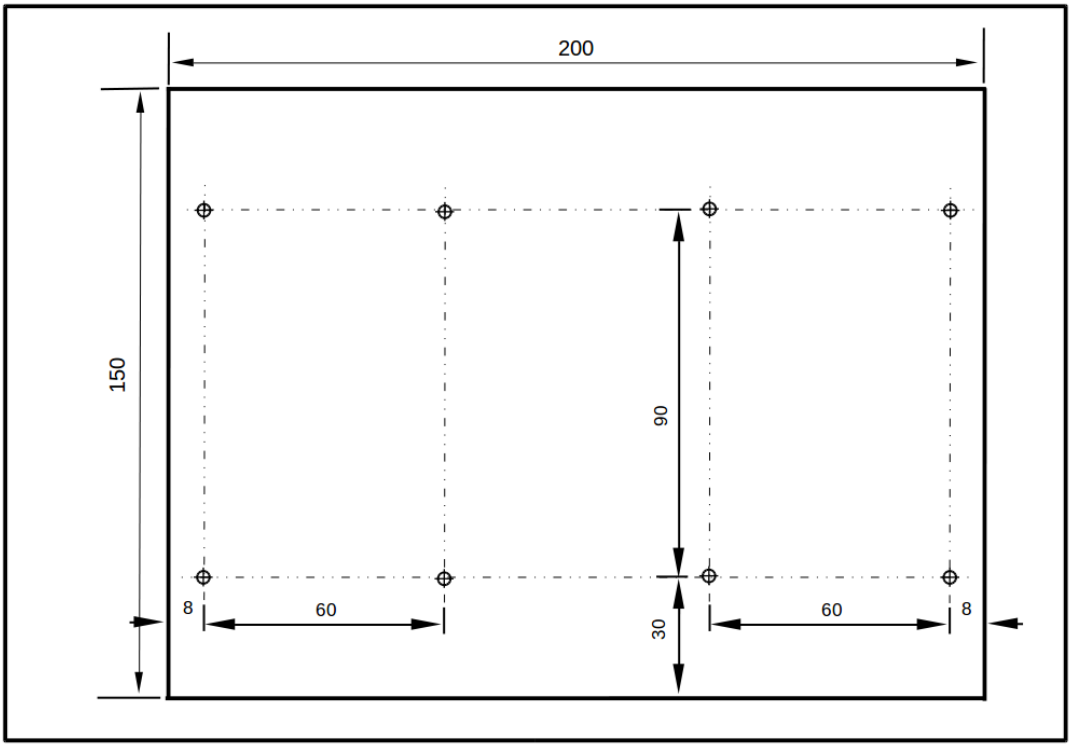
Placa lateral



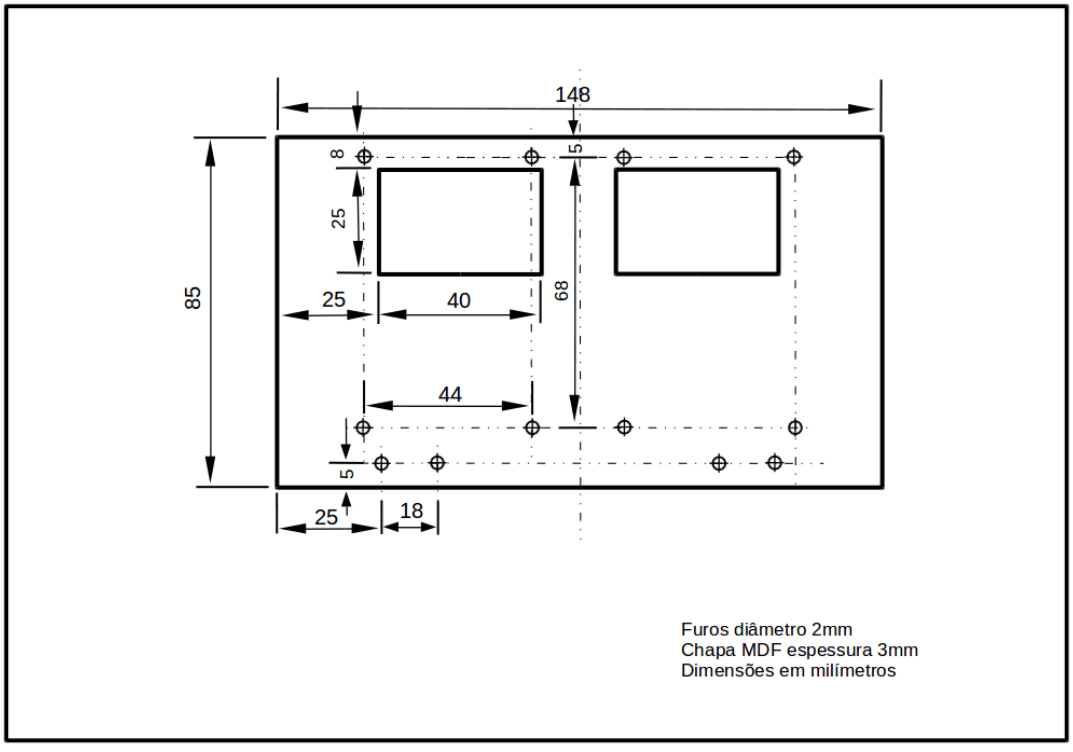
Travessa



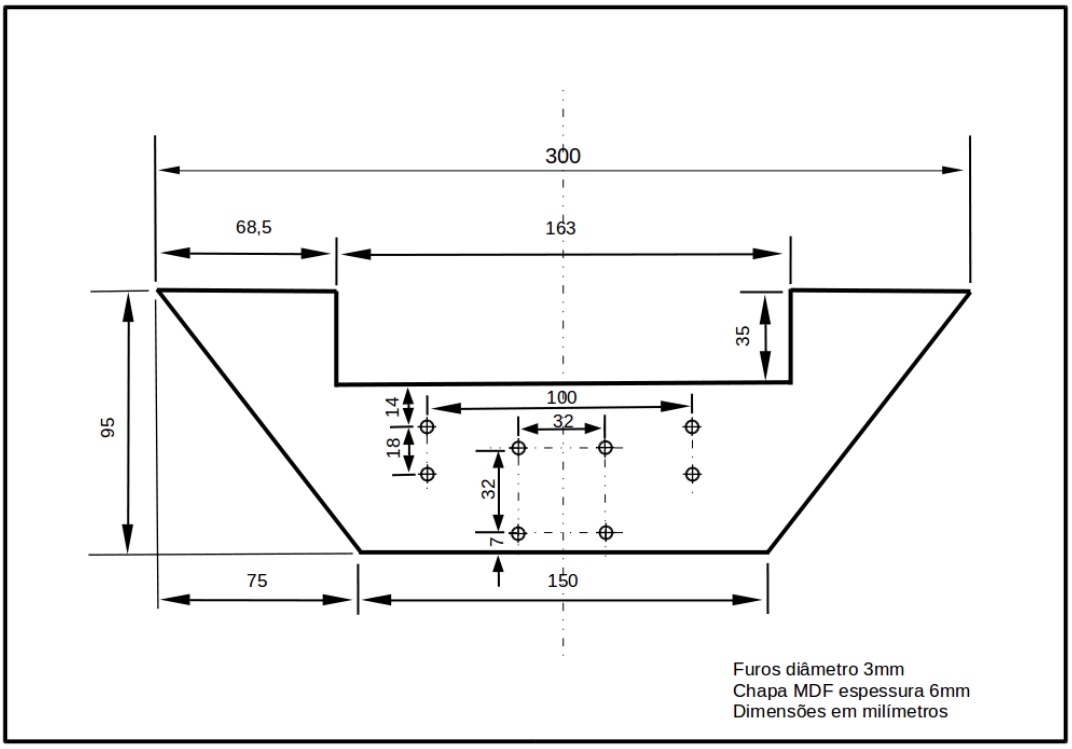
Placa inferior



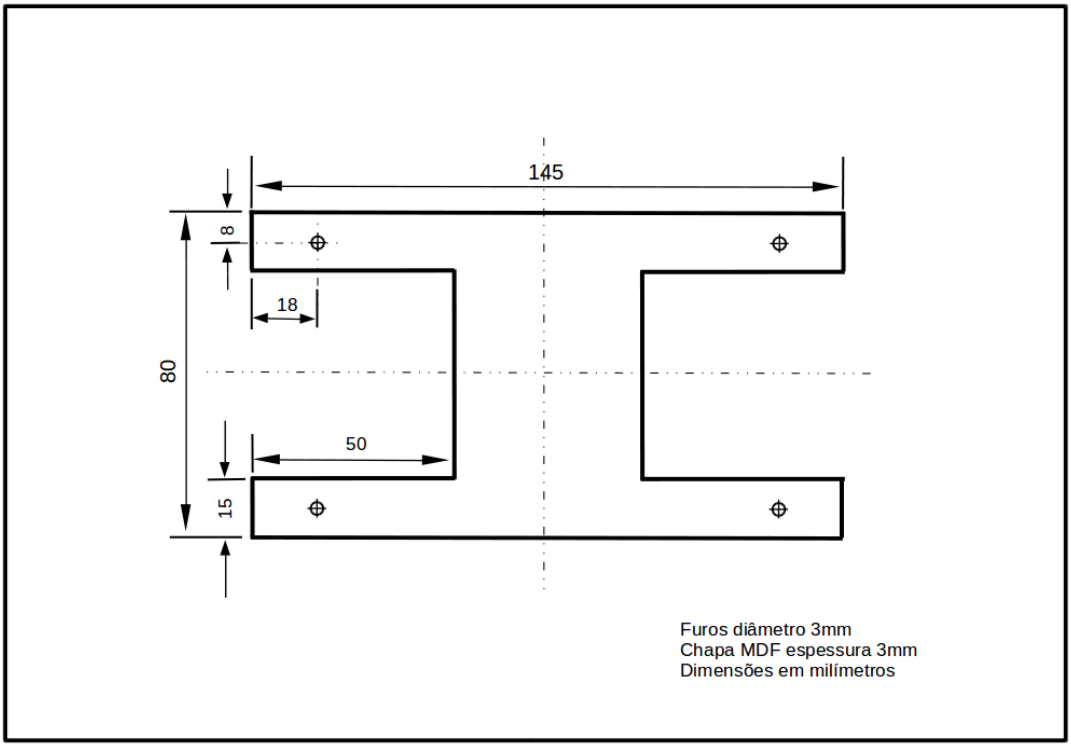
Suporte dos drivers



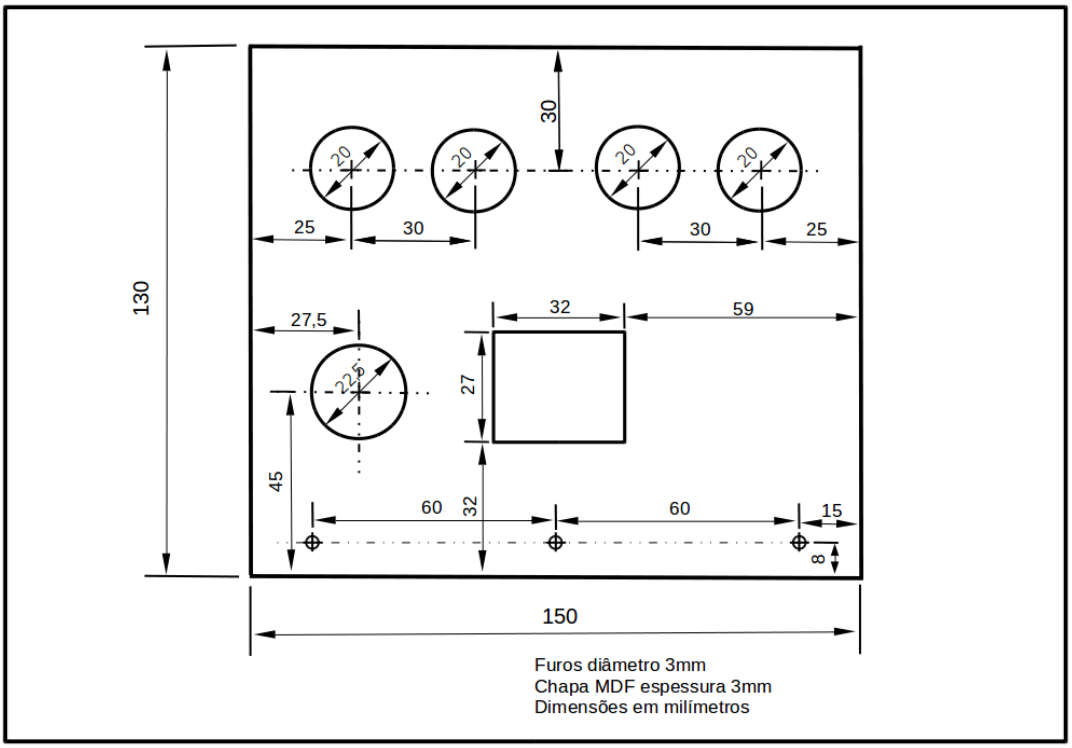
Suporte das rodas de apoio



Placa central



Painel superior



Travessa painel superior