UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
DEPARTAMENTO ACADÊMICO DE ELETRÔNICA
DEPARTAMENTO ACADÊMICO DE INFORMÁTICA
CURSO DE ENGENHARIA DE COMPUTAÇÃO

LUIS GUILHERME MACHADO CAMARGO
PEDRO ALBERTO DE BORBA
RICARDO FARAH
STEFAN CAMPANA FUCHS
TELMO FRIESEN

MAPEAMENTO DE AMBIENTES COM O ROBÔ BELLATOR

TERCEIRO ENTREGÁVEL

CURITIBA

2013

LUIS GUILHERME MACHADO CAMARGO
PEDRO ALBERTO DE BORBA
RICARDO FARAH
STEFAN CAMPANA FUCHS
TELMO FRIESEN

MAPEAMENTO DE AMBIENTES COM O ROBÔ BELLATOR

Terceiro entregável apresentado à Unidade Curricular de Oficina de Integração 3 do Curso de Engenharia de Computação da Universidade Tecnológica Federal do Paraná como requisito parcial para aprovação.

CURITIBA

2013

SUMÁRIO

1 TERCEIRO ENTREGÁVEL - 10/04/2013

Conforme estabelecido na relação de *deliverables* do documento de análise tecnológica, este terceiro entregável consiste nos seguintes itens:

1. Diagramas de casos de uso e de classes (estação base).
2. Diagrama de casos de uso (software embarcado).
3. Diagrama de fluxo de dados (software embarcado).
4. Diagrama em blocos (hardware).
5. Explicação detalhada de cada bloco (hardware).
6. Diagrama elétrico/eletrônico (hardware).

Além dos itens obrigatórios dos entregáveis, estão presentes neste documento os seguintes adicionais:

1. Requisitos funcionais.
2. Descrição das classes da estação base.
3. Diagrama de classes do sistema embarcado.
4. Descrição das classes do sistema embarcado.
5. Diagramas de estados.
6. Diagramas de sequência.
7. Diagrama em blocos do hardware.
8. Diagrama elétrico/eletrônico do hardware.

2 MODELAGEM UML

2.1 REQUISITOS FUNCIONAIS

1. A estação base deve mostrar na interface gráfica um mapa 2D (atualizado automaticamente) representando o robô e os obstáculos detectados por ele. Representado pelo requisito funcional: **"Estação base mostra mapa 2D do robô e dos obstáculos detectados – RF1"**.
2. O usuátio pode salvar o mapa 2D no disco rígido. Representado pelo requisito funcional: **"O usuário pode salvar o mapa – RF2"**.
3. O usuário pode carregar o mapa 2D do disco rígido. Representado pelo requisito funcional: **"O usuário pode carregar o mapa – RF3"**.
4. A estação base deve mostrar na interface gráfica a imagem captada pela *webcam* do robô. Representado pelo requisito funcional: **"Estação base mostra a imagem captada pela webcam – RF4"**.
5. O usuário pode movimentar o robô, controlando a velocidade de suas rodas remotamente pelo teclado da estação base. Representado pelo requisito funcional: **"O usuário pode movimentar o robô – RF5"**.
6. A estação base deve ser capaz de estabelecer conexão com o robô, informando o usuário caso a conexão ocorra com sucesso ou não. Representado pelo requisito funcional **"O usuário pode estabelecer a conexão entre o robô e a estação base – RF6"**.

2.2 REQUISITOS NÃO FUNCIONAIS

1. A imagem transmitida pela câmera do robô deve ser colorida. Representado pelo requisito não funcional: **"O robô deve enviar vídeo em imagem colorida para a estação base - RNF1"**.
2. O robô deve transmitir as imagens de sua câmera em tempo real. Representado pelo requisito não funcional: **"O robô deve transmitir os dados de vídeo captados pela**

câmera em tempo real - RNF2”.

2.3 CASOS DE USO IDENTIFICADOS

1. Visualização de mapa 2D na interface gráfica segundo os dados lidos dos sensores do robô. Representado pelo caso de uso: **“Mostrar mapa - UC1”**.
2. Gravação do mapa em um arquivo no disco rígido. Representado pelo caso de uso: **“Salvar mapa - UC2”**.
3. Leitura do mapa de um arquivo do disco rígido. Representado pelo caso de uso: **“Carregar mapa - UC3”**.
4. Leitura de informações dos sensores do robô. Representado pelo caso de uso: **“Ler amostras dos sensores - UC4”**.
5. Visualização de imagens da *webcam* do robô. Representado pelo caso de uso: **“Visualizar imagens da câmera - UC5”**.
6. Alteração pelo usuário da velocidade das rodas do robô. Representado pelo caso de uso: **“Alterar velocidade das rodas - UC6”**.
7. Solicitação de estabelecimento de conexão com o robô. **“Estabelecer conexão - UC7”**.
8. Consulta à documentação do robô pelo usuário. Representado pelo caso de uso: **“Consultar documentação do robô - UC8”**.

Foram produzidos três Diagramas de Casos de Uso (Figuras 1, 2 e 3) com base nos casos de uso apresentados. O primeiro diagrama representa o *software* da estação base, e o segundo e o terceiro representam o sistema embarcado (TS-7260 e placa de baixo nível, respectivamente).

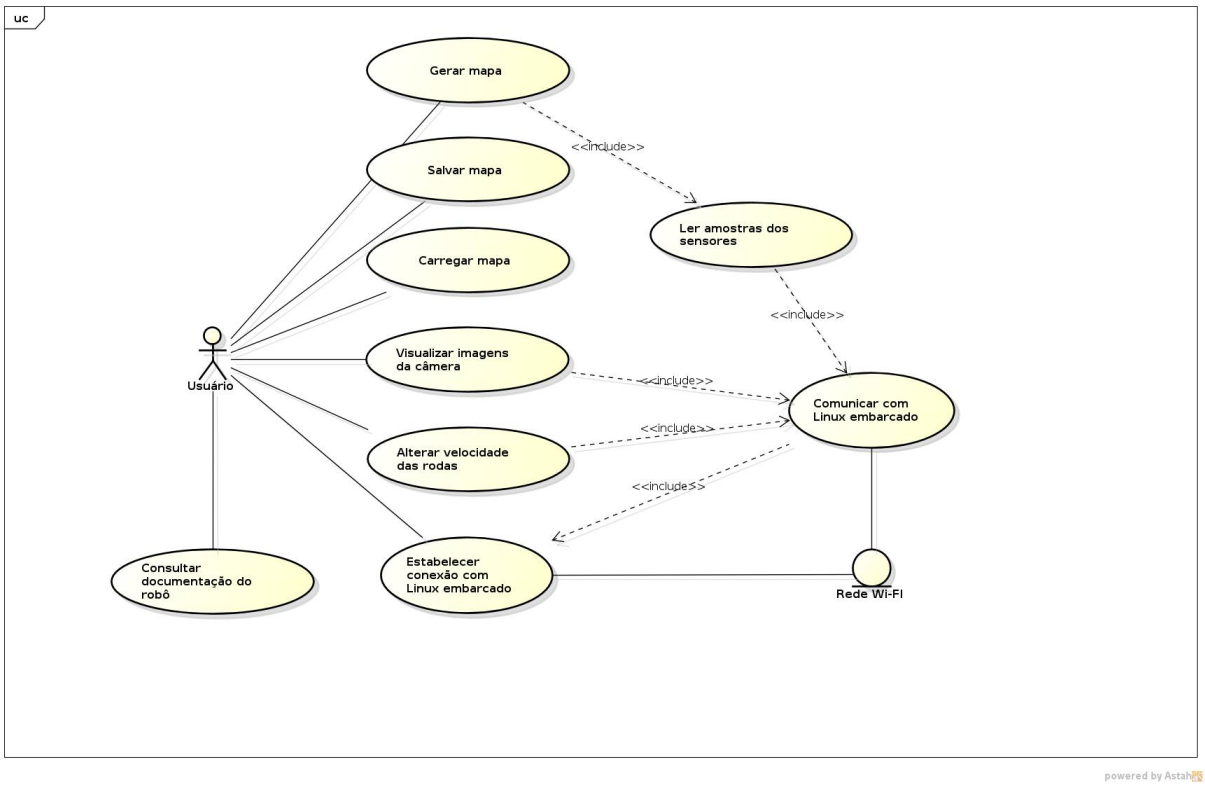

Figura 1: Diagrama de casos de uso do *software* da estação base.

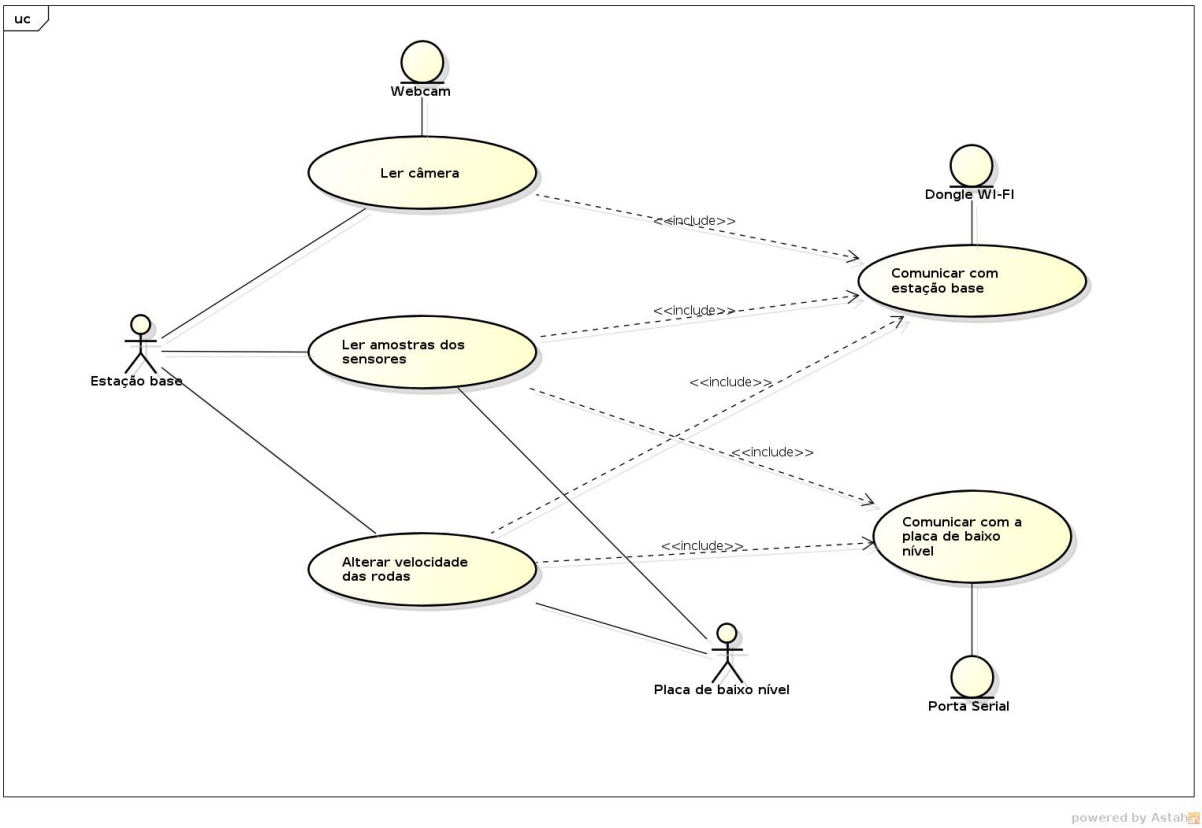

Figura 2: Diagrama de casos de uso do *software* para a placa TS-7260.

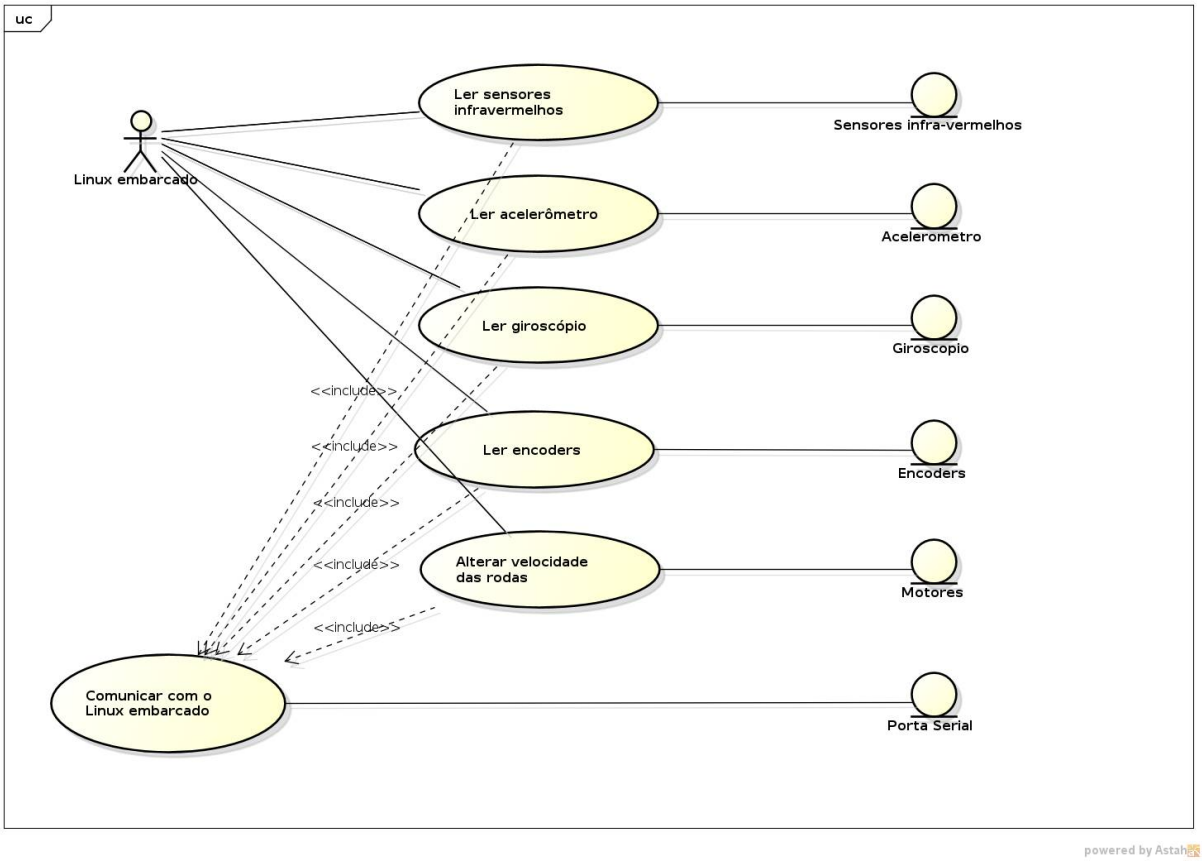

Figura 3: Diagrama de casos de uso do *software* para a placa de baixo nível.

2.4 DIAGRAMA DE CLASSES DA ESTAÇÃO BASE

A Figura 4 mostra o diagrama de classes da estação base.

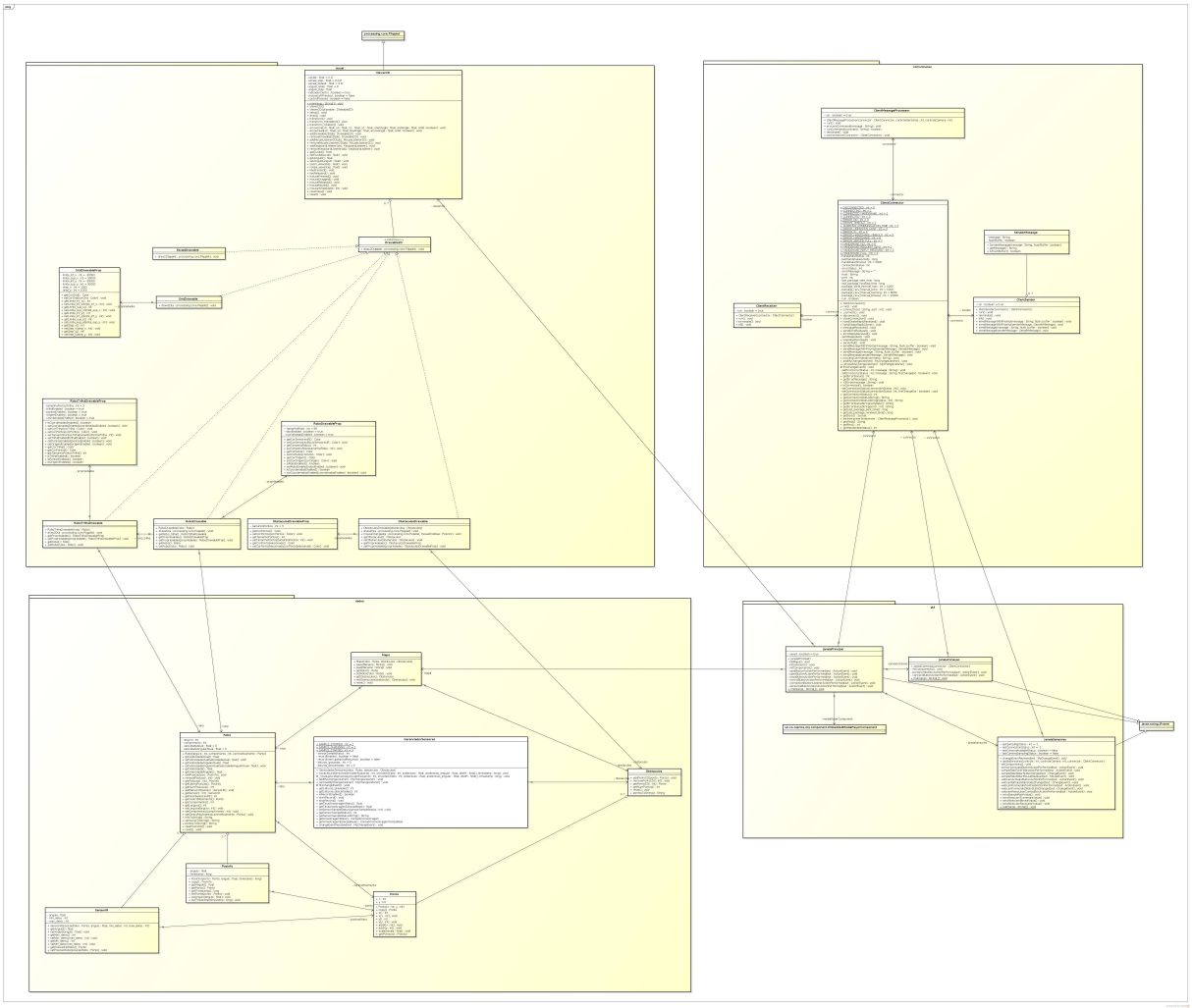

Figura 4: Diagrama de classes da estação base

2.4.1 Descrição das classes da estação base

O *software* da estação base do robô foi dividido em cinco pacotes: *visual*, *dados*, *comunicao*, e *gui*. A seguir há uma descrição de cada pacote e das suas respectivas classes.

2.4.2 Pacote *visual*

Este pacote consiste de toda a parte visual da estação base e conta com as seguintes classes: Viewer2D, Drawable2D, EscalaDrawable, RoboDrawable, RoboTrilhaDrawable, ObstaculosDrawable, EscalaDrawableProp, RoboDrawableProp, RoboTrilhaDrawableProp e ObstaculosDrawableProp. Na Tabela 1 estão descritas as classes deste pacote.

Tabela 1: Pacote *visual*

Classe	**Descrição**
Viewer2D	Responsável por exibir os objetos Drawable2D. Possui recursos de pan, zoom e rotate.
Drawable2D	Representa genericamente objetos 2D que podem ser desenhados em um Viewer2D.
EscalaDrawable	Responsável por desenhar uma escala gráfica no mapa.
RoboDrawable	Responsável por desenhar o robô no mapa.
RoboTrilhaDrawable	Responsável por desenhar a trilha percorrida pelo robô no mapa.
ObstaculosDrawable	Responsável por desenhar os pontos de cada obstáculo no mapa.
EscalaDrawableProp	Contém as propriedades visuais de desenho da escala.
RoboDrawableProp	Contém as propriedades visuais de desenho do robô
RoboTrilhaDrawableProp	Contém as propriedades visuais de desenho da trilha do robô.
ObstaculosDrawableProp	Contém as propriedades visuais de desenho dos obstáculos.

2.4.3 Pacote *dados*

Este pacote consiste de toda a parte da estação base que processa e armazena as informações essenciais do robô e do mapa. Conta com as seguintes classes: Mapa, Obstaculos, Robo, ControleSensores, Posinfo, SensorIR e Ponto. Na Tabela 2 estão descritas as classes deste pacote.

2.4.4 Pacote *comunicacao*

Este pacote consiste em toda a parte de comunicação da estação base com o robô e conta com as seguintes classes: ClientMessageProcessor ClientConnection, ClientReceiver, ClientSender e Message. Na Tabela 3 estão descritas as classes deste pacote.

É importante ressaltar que o protocolo TCP requer obrigatoriamente a especificação de um cliente e de um servidor para estabelecimento de uma conexão. Nas implementações desse protocolo em diversas linguagens (como Java e C++) existem tipos de *socket* distintos para cliente e servidor. Na criação de um *socket* de servidor, há obrigatoriamente a atribuição

Tabela 2: Pacote *dados*

Classe	**Descrição**
Mapa	Responsável por representar o mapa. Armazena as informações essenciais do robô e dos obstáculos detectados.
Obstaculos	Responsável por conter os obstáculos detectados pelo robô.
Robo	Responsável por representar o robô, este contêm largura, comprimento e centro de movimento (ponto central entre as duas rodas).
GerenciadorSensores	Responsável por atualizar a posição do robô e dos pontos que representam os obstáculos, de acordo com as leituras feitas pelos sensores.
Posinfo	Responsável por conter as informações de uma posição do robô.
SensorIR	Responsável por representar um sensor IR do robô.
Ponto	Representa um ponto de cordenadas cartesianas (x,y).
GerenciadorCamera	Responsável por gerenciar o status da câmera e o recebimento de imagens.

de uma porta de escuta, na qual o servidor aguarda que um cliente efetue uma requisição de conexão. Não é possível, ao menos nas implementações atuais do TCP, estabelecer conexão entre dois *sockets* de cliente ou entre dois *sockets* de servidor. Como neste projeto, o robô proverá serviços à estação base (envio de imagens da câmera, envio de leituras de sensores, além de prover a posibilidade de comando dos motores) o robô foi escolhido como servidor e a estação base como cliente. Enfatiza-se que o paradigma cliente-servidor não implica de forma alguma que a comunicação seja unidirecional. Pelo contrário, o envio de pacotes pode ser feito bidirecionalmente após uma conexão TCP ser estabelecida, sem nenhuma restrição quanto a isso.

2.4.5 Pacote *gui*

Este pacote consiste em toda a interface gráfica do sistema e conta com as seguintes classes: JanelaConexao, JanelaPrincipal e JanelaSensores. Na Tabela 4 estão descritas as classes deste pacote.

Tabela 3: Pacote *comunicacao*

Classe	**Descrição**
ClientMessageProcessor	Thread responsável pelo processamento de mensagens recebidas de um host de conexão.
ClientConnector	Thread responsável por efetuar a gerência da conexão do cliente (estação base) com o servidor (robô).
ClientReceiver	Thread responsável por receber mensagens de um host de uma conexão.
ClientSender	Thread responsável por enviar mensagens ao host de uma conexão.
SenderMessage	Contém uma mensagem que pode ser enviada por um Sender.

Tabela 4: Pacote *gui*

Classe	**Descrição**
JanelaConexao	Janela com as informações e configurações da conexão com o Bellator.
JanelaPrincipal	Janela principal da interface gráfica da estação base.
JanelaSensores	Janela de configuração dos sensores.

2.5 DIAGRAMA DE CLASSES DO SISTEMA EMBARCADO

A Figura 5 mostra o diagrama de classes do sistema embarcado (placa TS-7260).

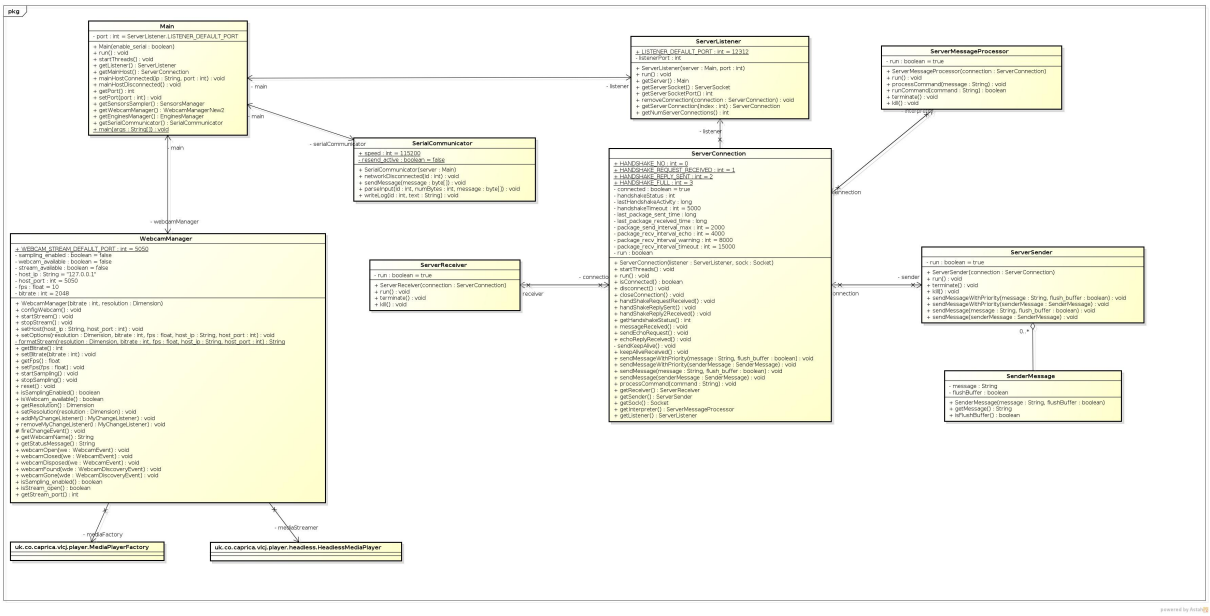

Figura 5: Diagrama de classes do sistema embarcado (placa TS-7260).

Tabela 5: Descrição das classes do sistema embarcado (placa TS-7260)

Classe	**Descrição**
Main	Classe principal do robô.
SensorsSampler	Thread responsável por requisitar amostras dos sensores da plca de baixo nível em intervalos de tempo previamente programados.
ServerMessageProcessor	Thread responsável por realizar o processamento de mensagens recebidas de um host de conexão.
ServerListener	Thread responsável por escutar requisições de conexão.
ServerSender	Thread responsável por enviar mensagens ao host de uma conexão.
SenderMessage	Contém uma mensagem que pode ser enviada por um Sender.
ServerReceiver	Thread responsável por receber mensagens de um host de uma conexão.
SerialCommunicator	Responsável por gerenciar a comunicação via porta serial entre a TS-7260 e a LPC2103.
WebcamManager	Responsável por gerenciar a abertura e o fechamento da stream de imagens da webcam, além de configurar opções como resolução e taxa de frames.

2.6 PROTOCOLO DE COMUNICAÇÃO

Esta seção detalha o protocolo de comunicação estabelecido entre a estação base, a placa TS-7260 (sistema com linux embarcado) e a placa LPC2103 (sistema embarcado de baixo nível).

O protocolo desenvolvido para a comunicação (via Wi-Fi) entre a estação base e a TS-7260 visa utilizar o TCP como camada de transporte. Como foi explicitado anteriormente na seção 2.4.4, o robô foi escolhido como servidor da conexão, e a estação base como cliente. Para haver confirmação da conexão entre os dois, optou-se por criar um protocolo de *handshake* semelhante ao existente no TCP, de 3 passos: requisição, resposta, confirmação. A requisição é feita pelo cliente no início da conexão, a resposta é dada pelo servidor em seguida. Posteriormente, o cliente envia uma mensagem de confirmação, e a conexão é totalmente estabelecida. O uso do *handshake* contribui para tanto a estação base como o sistema embarcado confirmarem que estão conectados um ao outro e não a um servidor/cliente qualquer.

Para possibilitar que o tráfego de mensagens possa ser feito de forma rápida, reduzindo atrasos, o envio e recebimento de mensagens é feito de forma assíncrona. Essa escolha foi feita tendo em vista que, em uma comunicação totalmente síncrona, um programa (ou thread) é bloqueado ao chamar uma função de recebimento ou envio, até que efetivamente seja completa a transação. Supondo que só houvesse uma thread gerenciando a conexão, o programa não poderia enviar ou receber dados ao mesmo tempo em *full-duplex*, mas somente *half-duplex* (somente enviar ou somente receber).

A solução desenvolvida para possibilitar a comunicação assíncrona foi o uso de 4 threads (tanto na estação base quanto no sistema embarcado) para gerenciar os diversos aspectos envolvidos nela. A primeira *thread* (a gerenciadora principal de conexão) é a responsável por estabelecer e manter a conexão, além de gerenciar os potenciais erros que possam ocorrer (tempo excessivo sem comunicação e fechamento de socket). Para a estação base, o diagrama de estados dessa *thread* está representado na Figura 6, e para o sistema embarcado na Figura 7.

A segunda *thread* tem a função de gerenciar o envio de mensagens. O programa principal, ao necessitar enviar uma mensagem, faz uma requisição a essa *thread* que insere a mensagem em uma fila de envio. O programa principal não fica bloqueado, dessa forma, pois não necessita aguardar a mensagem ser completamente enviada, podendo efetuar outras tarefas. O diagrama de estados dessa *thread* está presente na Figura 8.

A terceira *thread* gerencia o recebimento de mensagens. Seu funcionamento é relativamente simples: ela possui um loop, no qual aguarda até alguma mensagem ser recebida.

Quando ocorre o recebimento de alguma mensagem, ela é encaminhada para a quarta *thread* (cuja explicação está a seguir) que processa o conteúdo dela e executa as operações que são necessárias para cada tipo de mensagem. Dessa forma, novas mensagens podem ser recebidas rapidamente, pois o receptor não fica bloqueado realizando o processamento das informações recebidas. O diagrama de estados dessa terceira *thread* está presente na Figura 9.

A quarta *thread*, como já exposto, é a responsável por processar mensagens recebidas e realizar as operações que são necessárias para cada tipo de mensagem, o que depende da codificação exposta na seção 2.6.1. Ela possui uma fila, na qual são inseridas as mensagens a serem processadas. Dessa forma a *thread* receptora não necessita ficar bloqueada aguardando o término do processamento. O diagrama de estados dessa quarta *thread* está presente na Figura 10.

2.6.1 Codificação das mensagens

- Mensagens do TS-7260 para o LPC2103 (via porta serial)
  - **SYNC**
    (byte) END_CMD
    Quando o microcontrolador LPC2103 recebe esta mensagem, responde com as leituras mais recentes dos encoders, de cada sensor de distância, do acelerômetro e do giroscópio (enviando uma mensagem SENSORS, explicada abaixo).
  - **ENGINES**
    (byte) vel_roda_esquerda
    (byte) vel_roda_direita
    (byte) END_CMD
    Ao receber este comando, o microcontrolador utiliza os valores para definir o nível de PWM para as rodas do robô. Os valores de velocidade são representados por um byte cada, nos quais o bit mais significativo indica o sentido de rotação da roda (1 para frente e 0 para trás) e os restantes a intensidade do PWM.
- Mensagens do LPC2103 para a TS-7260 (via porta serial)
  - **SENSORS**
    (byte) encoder_esq_H, *(byte) encoder_esq_L*,
    (byte) encoder_dir_H, *(byte) encoder_dir_L*,
    (byte) IR1, *(byte) IR2*, *(byte) IR3*, *(byte) IR4*, *(byte) IR5*,

(byte) AX_H, *(byte) AX_L*,
(byte) AY_H, *(byte) AY_L*,
(byte) AZ_H, *(byte) AZ_L*,
(byte) GX_H, *(byte) GX_L*,
(byte) GY_H, *(byte) GY_L*,
(byte) GZ_H, *(byte) GZ_L*,
(byte) TIMESTAMP_H, *(byte) TIMESTAMP_L*
(byte) END_CMD
Representa a leitura de todos os sensores (encoders, infra-vermelhos, acelerômetro e giroscópio).

Os 4 primeiros bytes são os valores das leituras dos encoders esquerdo e direíto (cada um com um byte alto e um baixo). Os valores das leituras dos encoders representam a diferença entre a contagem atual a contagem anterior.

Nos próximos 5 bytes, as leituras do sensores ópticos são enviadas em sequência. As distâncias que os sensores ópticos são capazes de mensurar são divididos em valores discretos de 0 a 255 (MARIN et al., 2012).

Após isso, os 12 bytes que se seguem representam as leituras do acelerômetro e do giroscópio. Os bytes que começam com 'A' representam a leitura de cada um dos eixos do acelerômetro. Aqueles que começam com 'G' representam a leitura de cada um dos eixos do giroscópio.

O timestamp (valor alto e baixo) é um contador de 16 bits que é incrementado entre cada amostre e zera automaticamente quando chega ao valor máximo, usado para determinar o instante em que foi feita a leitura dos dados. Como a amostragem dos sensores na placa de baixo nível será efetuada em intervalos fixos, a informação do contador do timestamp pode ser utilizada para obter informações de tempo de cada amostra.

- Mensagens bidirecionais entre estação base e TS-7260 (via Wi-Fi):

  – **ECHO_REQUEST**
  (byte) END_CMD
  Requisição de ping.

  – **ECHO_REPLY**
  (byte) END_CMD
  Resposta de ping.

  - **DISCONNECT**

    (byte) END_CMD

    Solicitação de desconexão.

- Mensagens da estação base para a TS-7260 (via Wi-Fi):

  - **HANDSHAKE_REQUEST**

    (byte) END_CMD

    Solicitação de handshake.

  - **HANDSHAKE_CONFIRMATION**

    (byte) END_CMD

    Confirmação de handshake.

  - **SENSORS_START**

    (byte) END_CMD

    Solicitação de início da amostragem dos sensores.

  - **SENSORS_STOP**

    (byte) END_CMD

    Solicitação de parada da amostragem dos sensores.

  - **SENSORS_RATE**

    (float) Nova taxa de amostragem

    (byte) END_CMD

    Solicitação de mudança da taxa de amostragem dos sensores (amostras/s).

  - **WEBCAM_START**

    (byte) END_CMD

    Solicitação de início da amostragem da webcam.

  - **WEBCAM_STOP**

    (byte) END_CMD

    Solicitação de parada da amostragem da webcam.

  - **WEBCAM_RATE**

    (float) Nova taxa de quadros

    (byte) END_CMD

    Solicitação de mudança da taxa de quadros da webcam.

  - **WEBCAM_RESOLUTION**

    (int) Largura em pixels

    (int) Altura em pixels

(byte) END_CMD

Solicitação de mudança da resolução da webcam.

– **ENGINES**

(byte) vel_roda_esquerda

(byte) vel_roda_direita

(byte) END_CMD

Solicitação de mudança da velocidade dos motores.

- Mensagens da TS-7260 para a estação base (via Wi-Fi):

  – **HANDSHAKE_REPLY**

  (byte) END_CMD

  Resposta de handshake.

  – **SENSORS**

  (byte) encoder1_H, *(byte) encoder1_L*,
  (byte) encoder2_H, *(byte) encoder2_L*,
  (byte) IR1, *(byte) IR2*, *(byte) IR3*, *(byte) IR4*, *(byte) IR5*,
  (byte) AX_H, *(byte) AX_L*,
  (byte) AY_H, *(byte) AY_L*,
  (byte) AZ_H, *(byte) AZ_L*,
  (byte) GX_H, *(byte) GX_L*,
  (byte) GY_H, *(byte) GY_L*,
  (byte) GZ_H, *(byte) GZ_L*,
  (byte) TIMESTAMP_H, *(byte) TIMESTAMP_L*
  (byte) END_CMD

  Possui a mesma funcionalidade e parâmetros que a mensagem SENSORS enviada da LPC2103 para a TS-7260.

  – **SENSORS_STATUS**

  (boolean) Status da amostragem [on - off]

  (float) Taxa de amostragem

  (byte) END_CMD

  Informações de status dos sensores. Usado na interface gráfica para confirmar o recebimento de comandos de mudança de taxa de amostragem e início/parada da amostragem.

– **WEBCAM_STATUS**
(float) Taxa de quadros
(int) Largura em pixels
(int) Altura em pixels
(boolean) Status da stream [on - off]
(int) Porta da stream
(byte) END_CMD
Informações de status da webcam. Usado na interface gráfica para confirmar o recebimento de comandos relativos à webcam, e para que a estação base tenha conhecimento do status da stream da webcam.

– **ENGINES_STATUS**
(byte) vel_roda_esquerda
(byte) vel_roda_direita
Informações sobre as velocidades programadas dos motores. Usado na interface gráfica para confirmar o recebimento de comandos de movimentação efetuados pelo usuário.

2.6.2 Diagramas de estados

Nesta seção estão expostos os diagramas de estados do protocolo de comunicação.

Um aspecto importante a ressaltar é que, nas *threads* de envio (Figura 8) e de processamento de mensagens (Figura 10), pode haver adição assíncrona de elementos na fila. Ou seja, quando é feita a verificação do número de elementos presentes na fila (como representado nos diagramas), tem-se em vista que elementos podem ter sido adicionados a qualquer instante. Obviamente, no ponto de vista da implementação, existem as seções críticas que devem ser devidamente gerenciadas para evitar condições de disputa e outros problemas de concorrência. Porém, as seções críticas se resumem aos acessos à fila somente, o que reduz consideravelmente a complexidade do processo.

Na Figura 11 está exposto o diagrama de estados da *thread* do Linux embarcado que é responsável por realizar a amostragem dos sensores em intervalos fixos de tempo. Ela realiza a amostragem enviando periodicamente – quando programada – comandos SYNC (vide seção 2.6.1) para a placa de baixo nível. Vale ressaltar que para melhor explicar este processo, na Figuras 16 e 17 da próxima seção está exposto um diagrama de sequência que demonstra a amostragem dos sensores.

Nas Figuras 12 e 13 estão presentes os diagramas de estados da captura e recebimento

de imagens da webcam. Foi utilizada a biblioteca externa *libVLC* (VIDEOLAN, 2013) – a componente de baixo nível do *player* de mídia VLC – tanto na estação base como no Linux embarcado para efetuar o processo de transmissão e visualização de imagens. No Linux embarcado, quando um comando de início de webcam é dado pelo usuário, uma *stream* HTTP de imagens é aberta pela *libVLC*, e a estação base é posteriormente notificada sobre o fato. Na estação base, quando ocorre a notificação de que a stream foi aberta, a componente de *player* da *libVLC* da janela principal é ativada (conectando dessa forma, na stream HTTP de imagens).

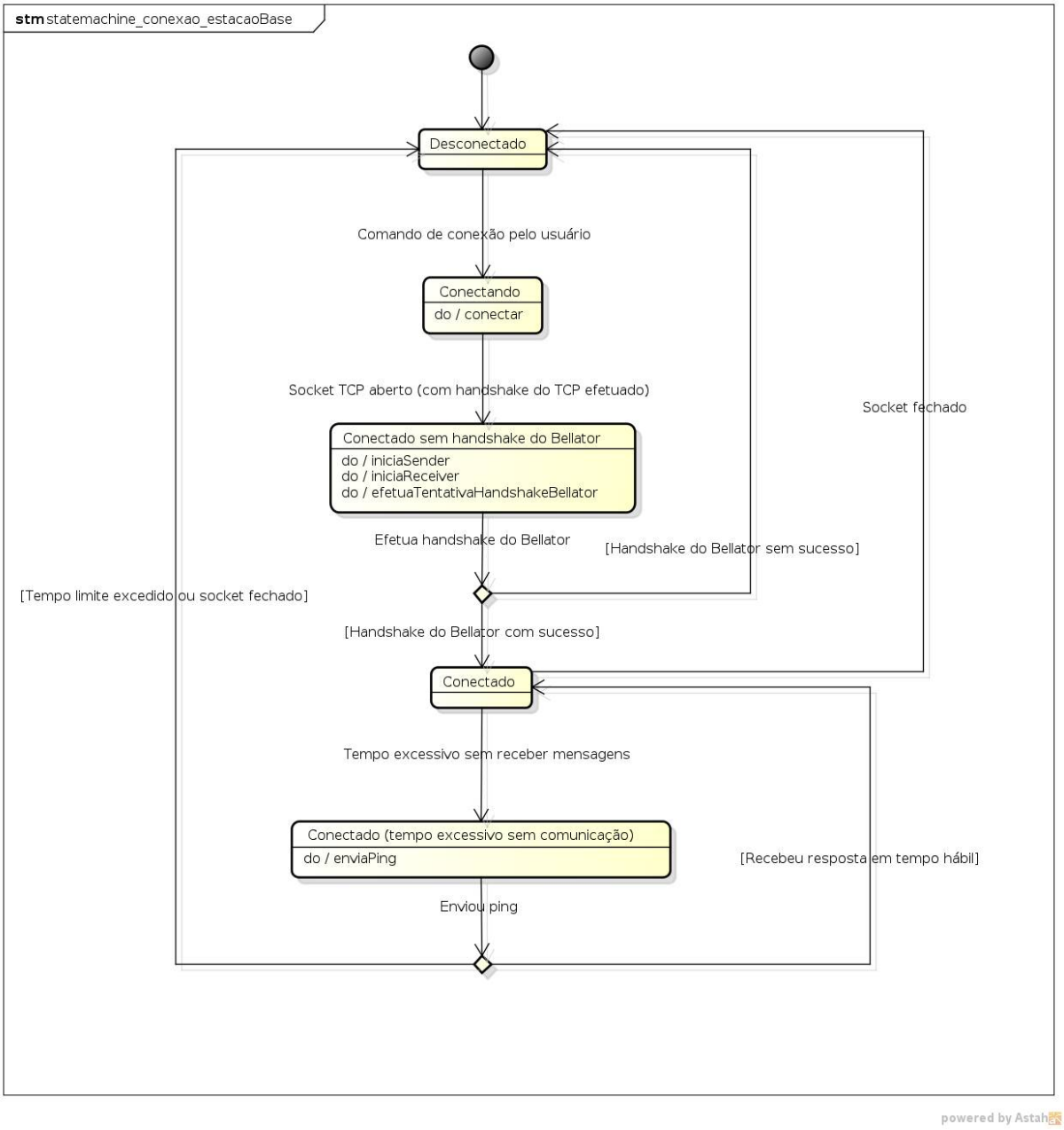

Figura 6: Diagrama estados da *thread* principal da conexão da estação base.

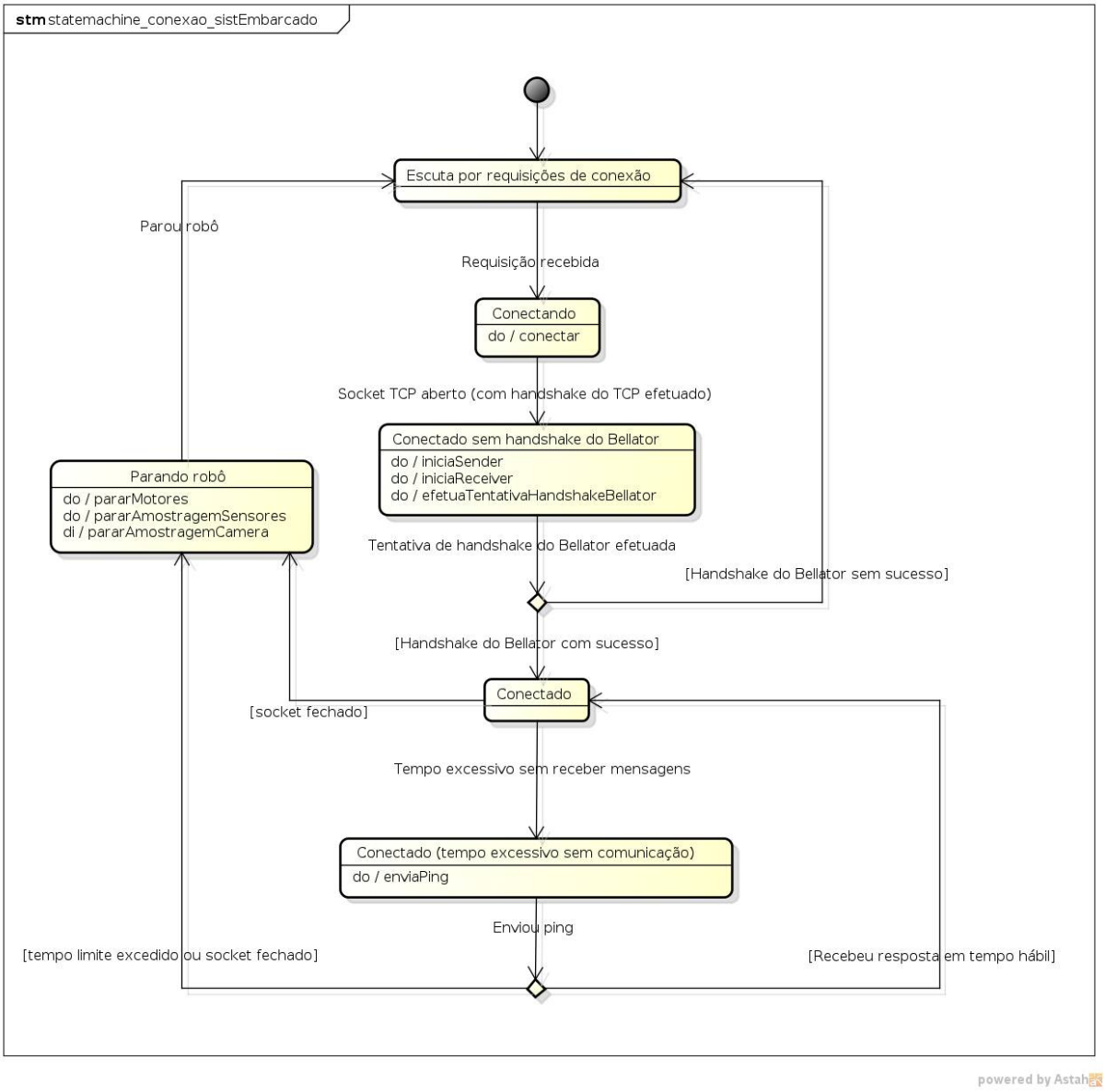

Figura 7: Diagrama de estados da *thread* principal da conexão do sistema embarcado.

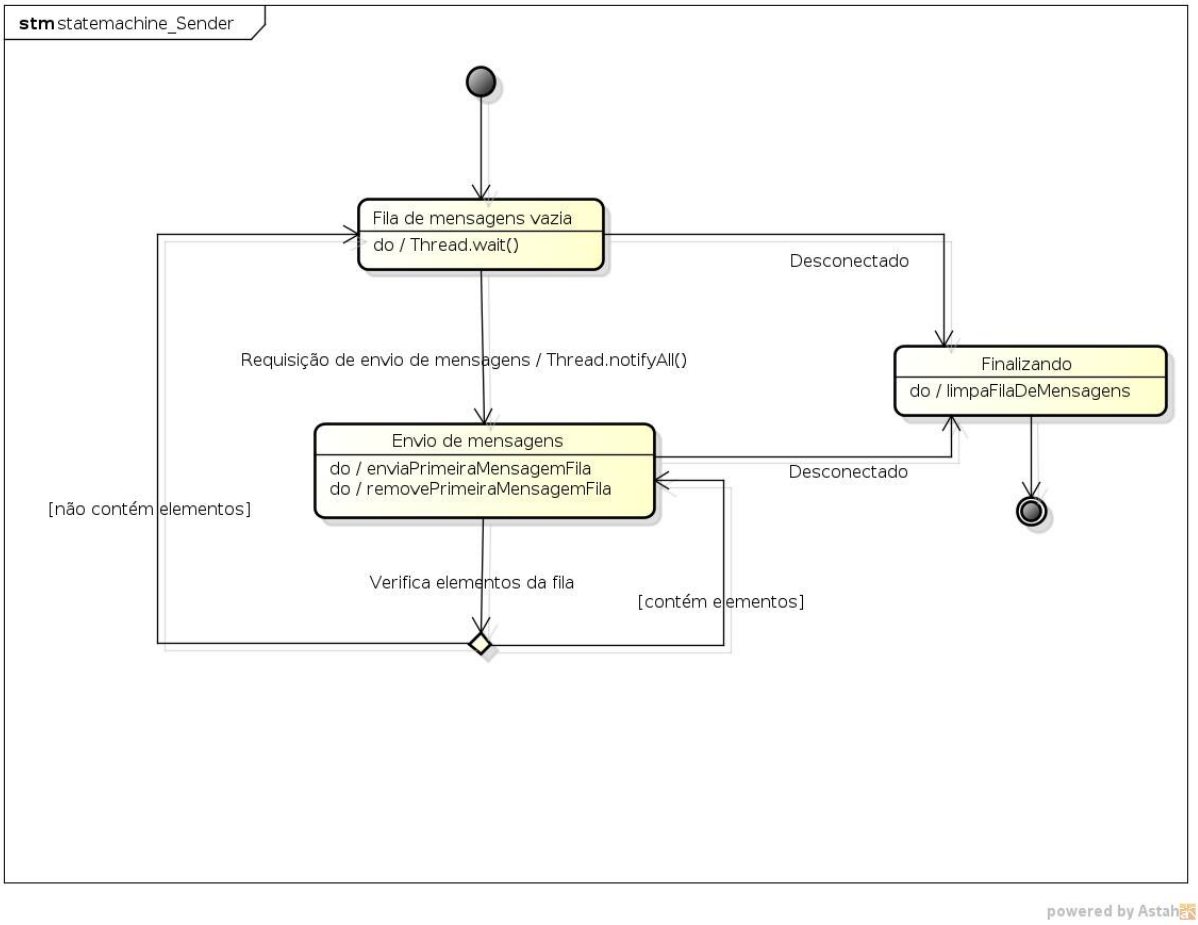

Figura 8: Diagrama de estados da *thread* que envia mensagens (igual para estação base e sistema embarcado).

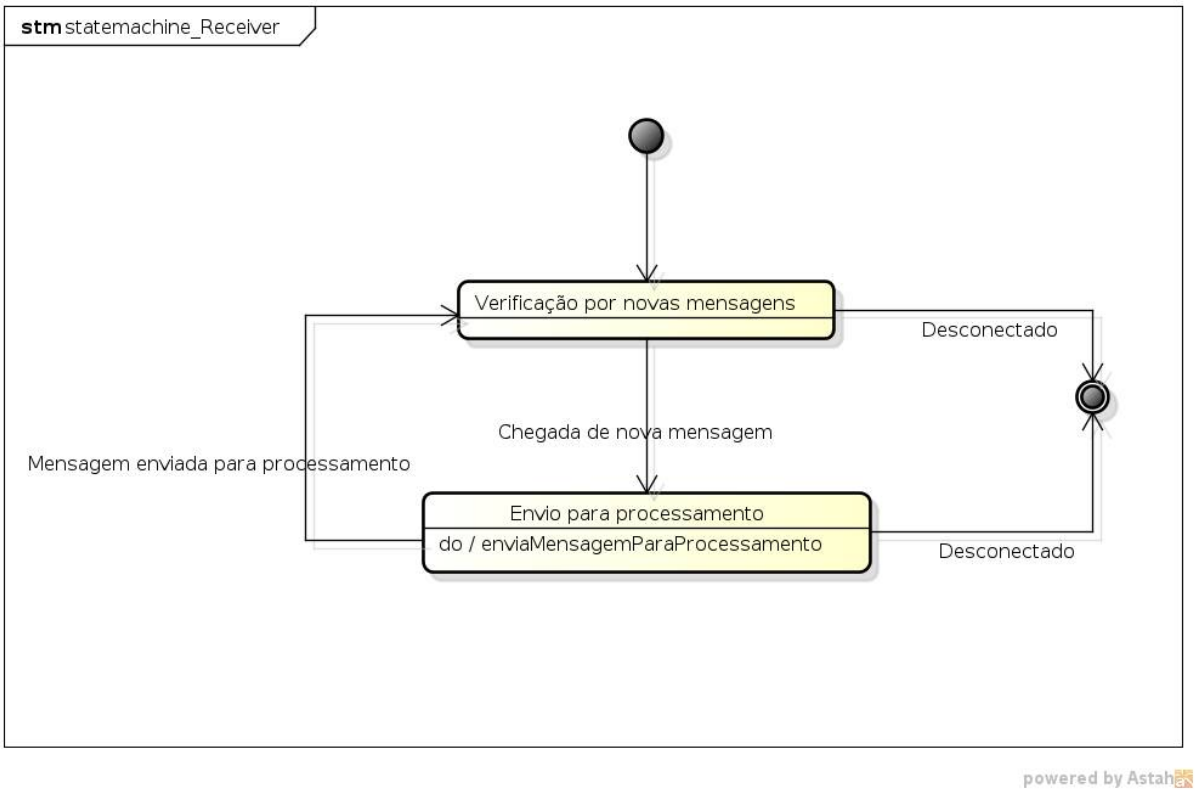

Figura 9: Diagrama de estados da *thread* receptora de mensagens (igual para estação base e sistema embarcado).

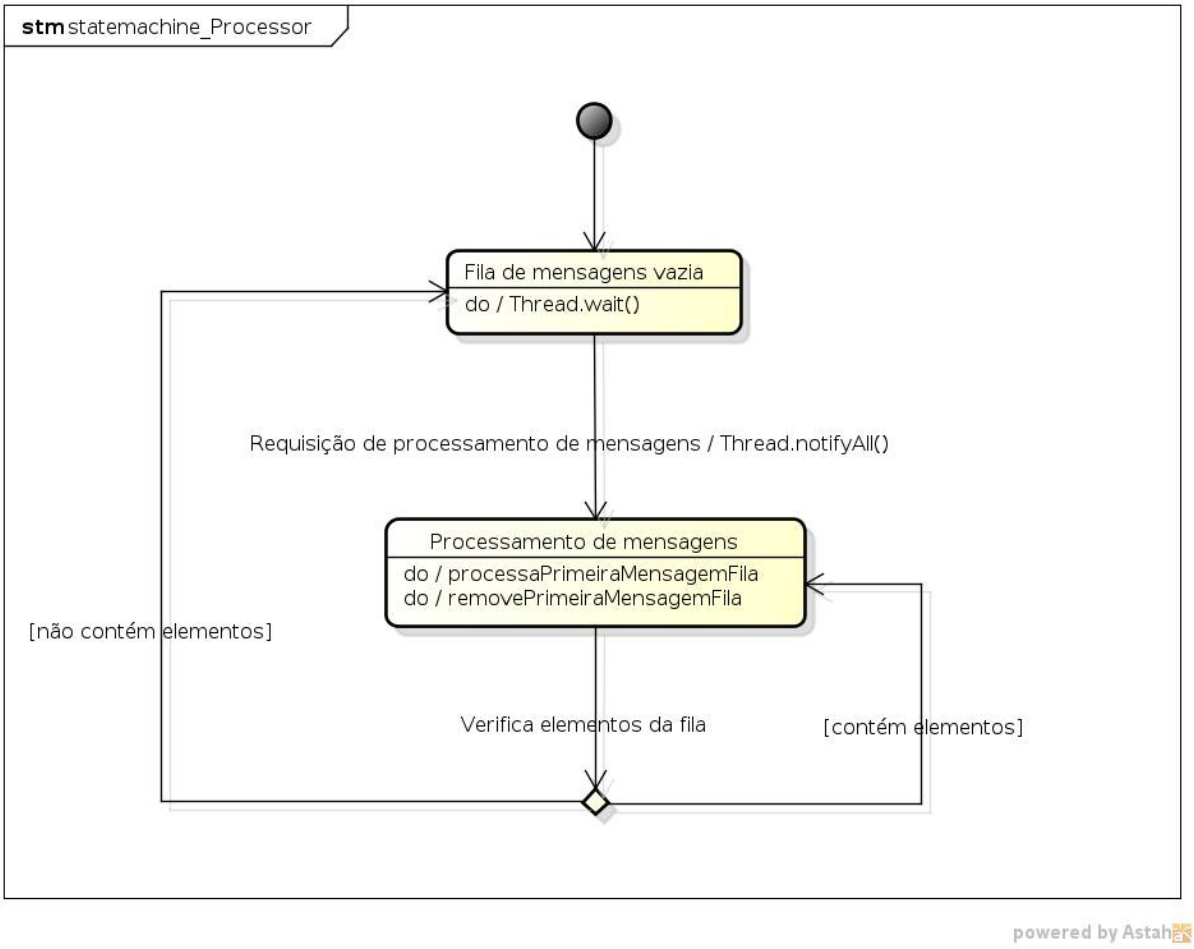

Figura 10: Diagrama de estados da *thread* que processa mensagens recebidas (igual para estação base e sistema embarcado).

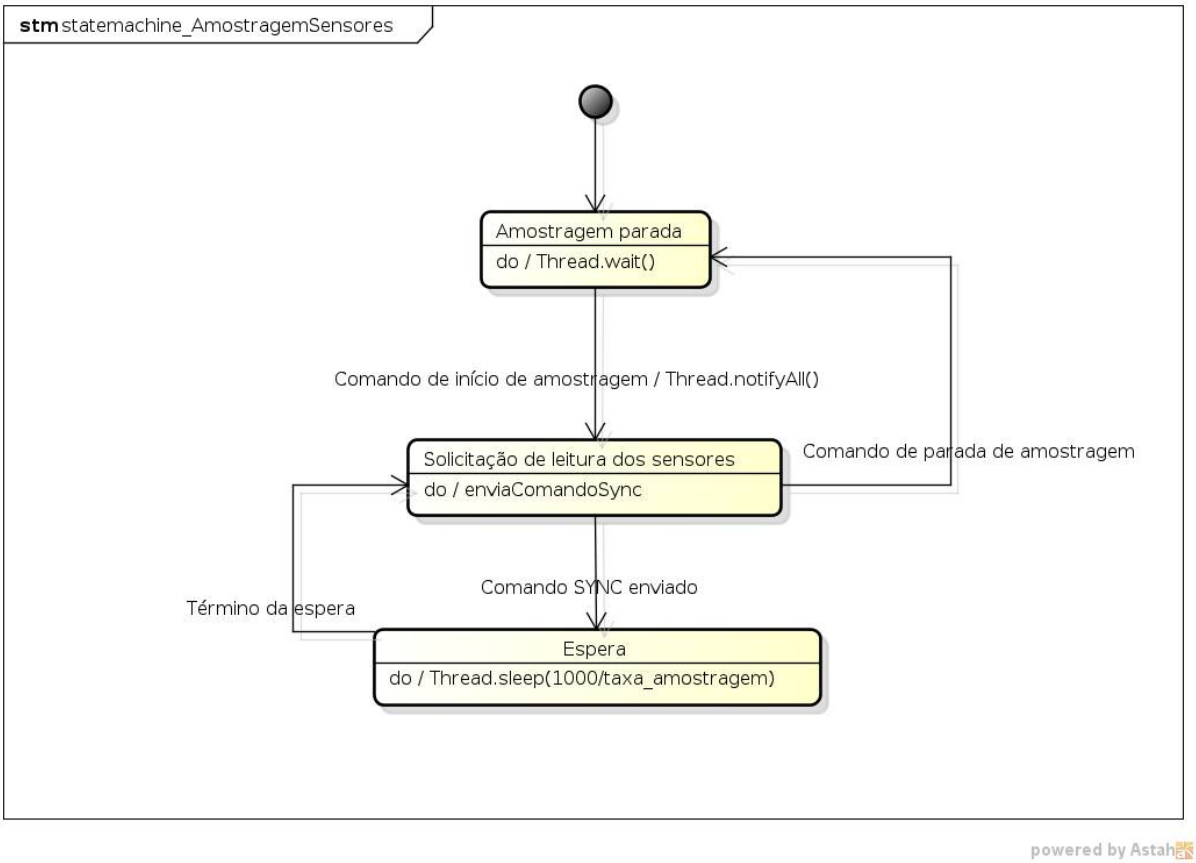

Figura 11: Diagrama de estados da *thread* responsável por efetuar a amostragem dos sensores (sistema embarcado).

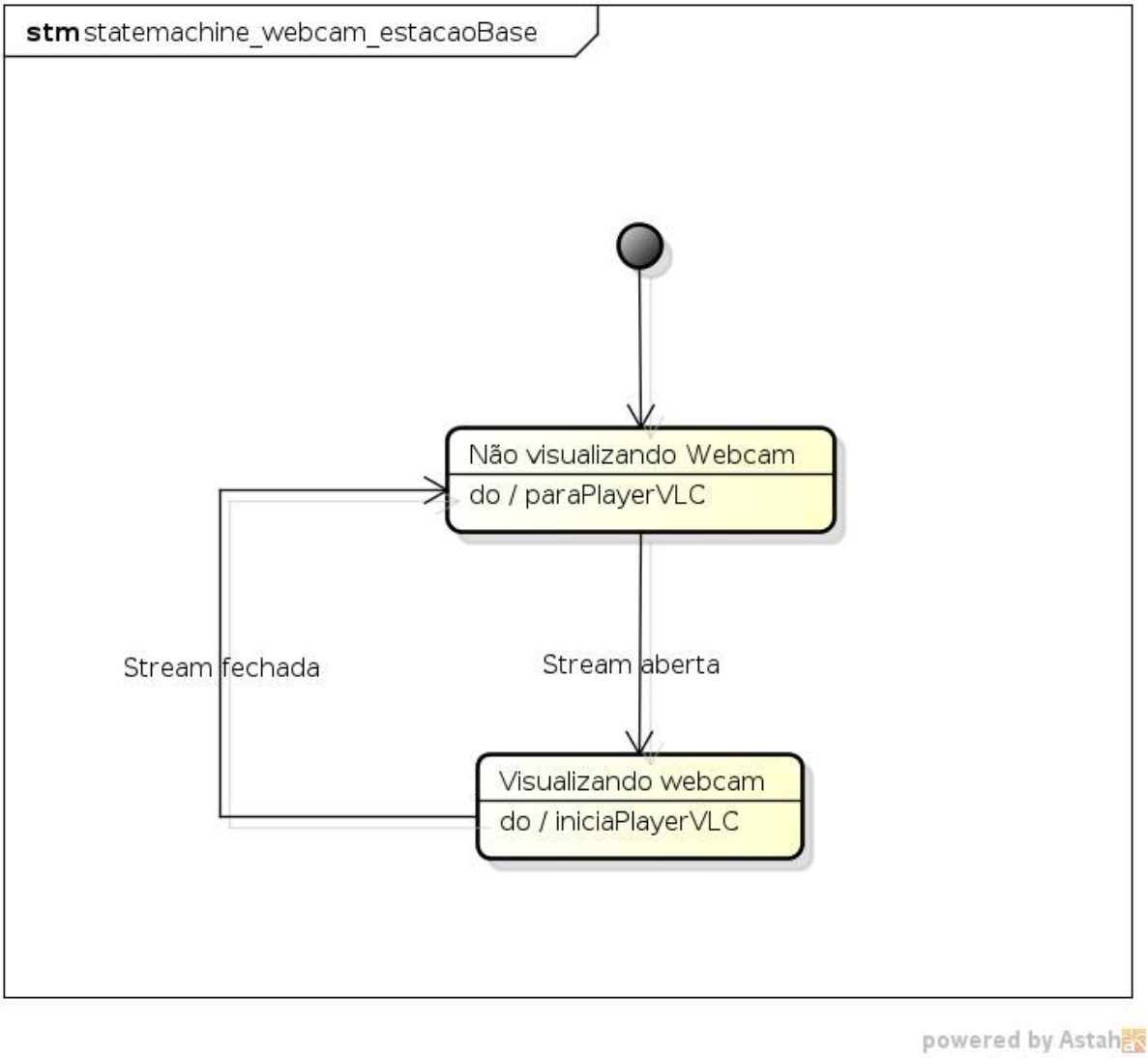

Figura 12: Diagrama de estados do visualização de imagens da *webcam* (estação base).

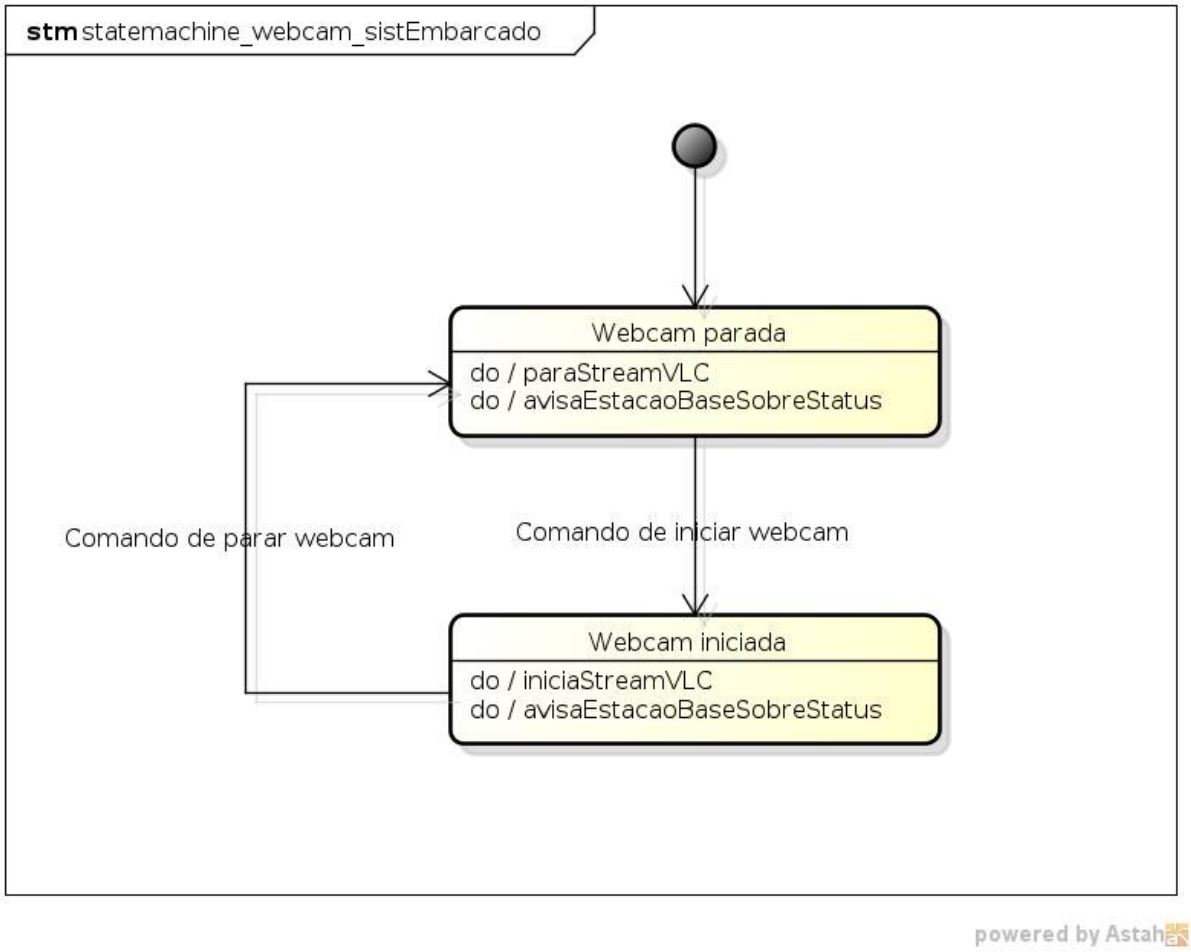

Figura 13: Diagrama de estados do envio de imagens da *webcam* (sistema embarcado).

2.6.3 Diagramas de sequência

Nessa seção os diagramas de sequência de comandos dos motores, de mensagens de amostras dos sensores e da ativação da webcam. Os diagramas das Figuras 14 e 15 representam como um comando de mudança de velocidade das rodas dado pelo usuário chega até a placa de baixo nível. Os das Figuras 16 e 17 demonstram a sequência dos dados de leituras dos sensores que saem da placa de baixo nível e chegam até o usuário. Os diagramas das Figuras 18 e 19 demonstram um comando de ativação da webcam dado pelo usuário, como ele chega até o Linux embarcado e como posteriormente o usuário recebe as imagens da webcam.

Vale ressaltar que as chamadas assíncronas, ou seja, que não bloqueiam a execução da *thread* chamadora, foram representadas também nestes diagramas.

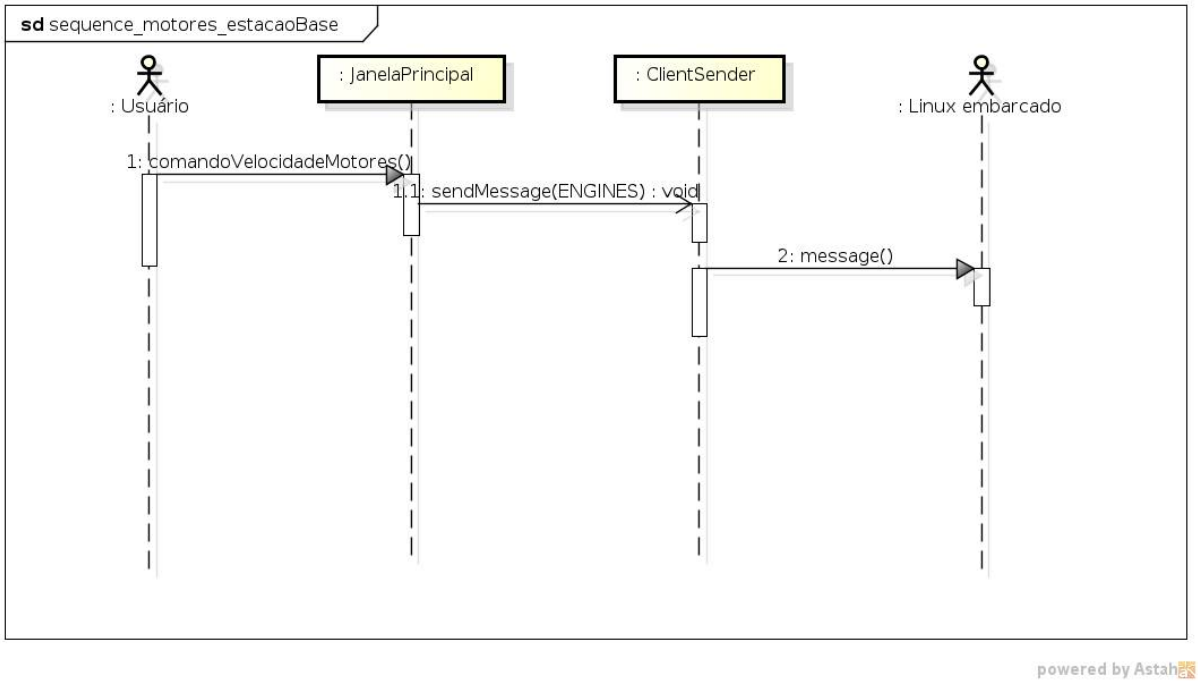

Figura 14: Diagrama de sequência de comando para mudança de velocidade dos motores (representa comando dado pelo usuário).

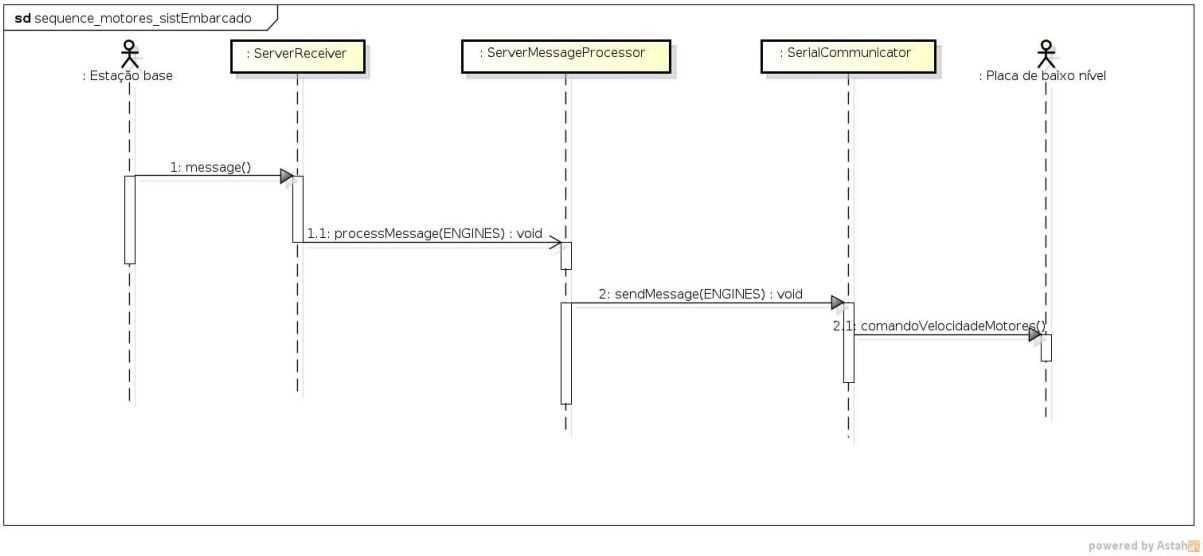

Figura 15: Diagrama de sequência de comando para mudança de velocidade dos motores (representa mensagem chegando no sistema embarcado).

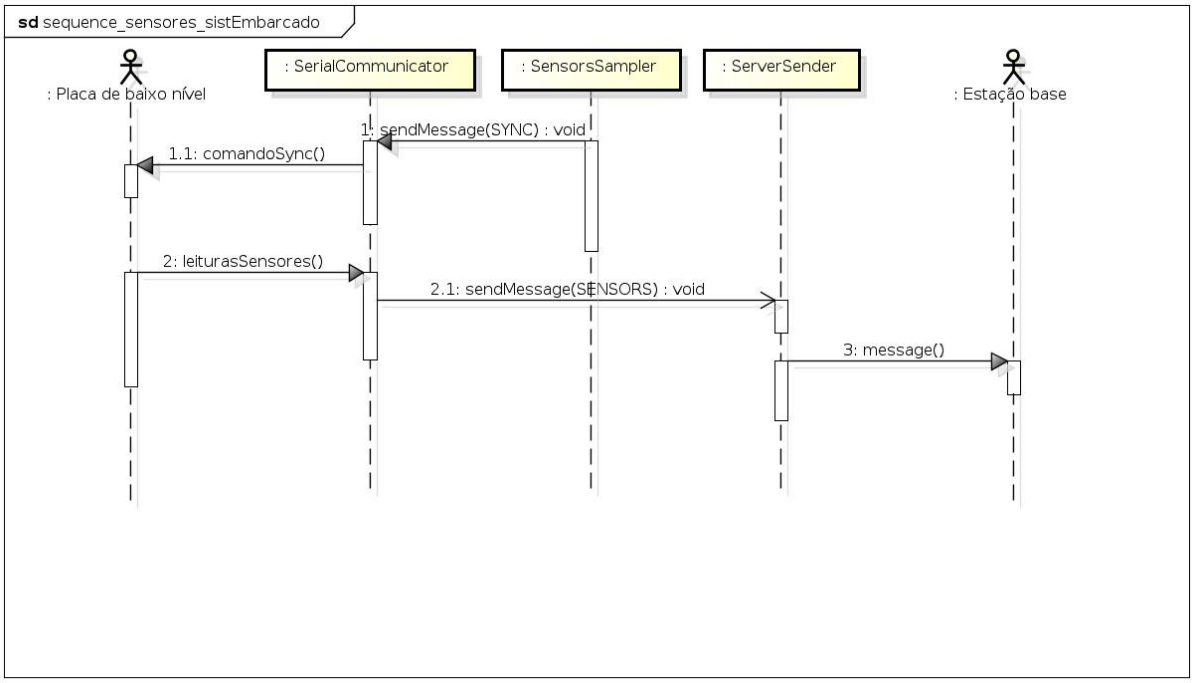

Figura 16: Diagrama de sequência da amostragem dos sensores (representa amostras saindo do sistema embarcado).

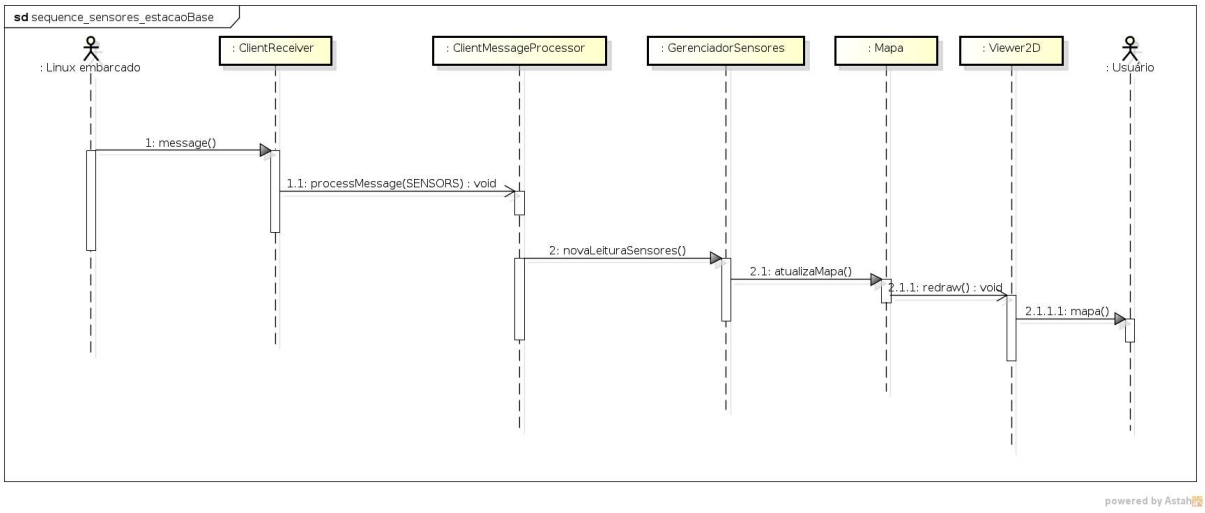

Figura 17: Diagrama de sequência da amostragem dos sensores (representa mensagem chegando na estação base).

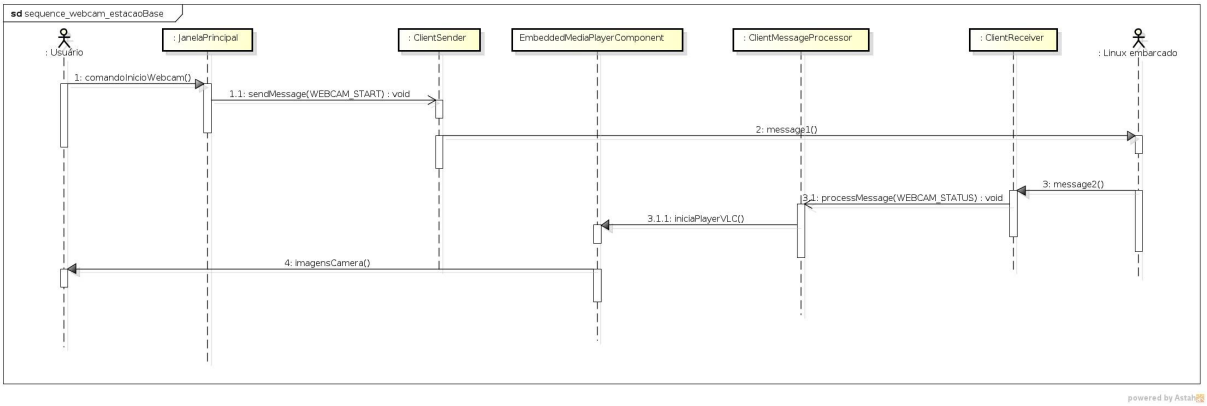

Figura 18: Diagrama de sequência de comando para ativação da webcam (representa comando dado pelo usuário e o *player* da *libVLC* sendo ativado posteriormente).

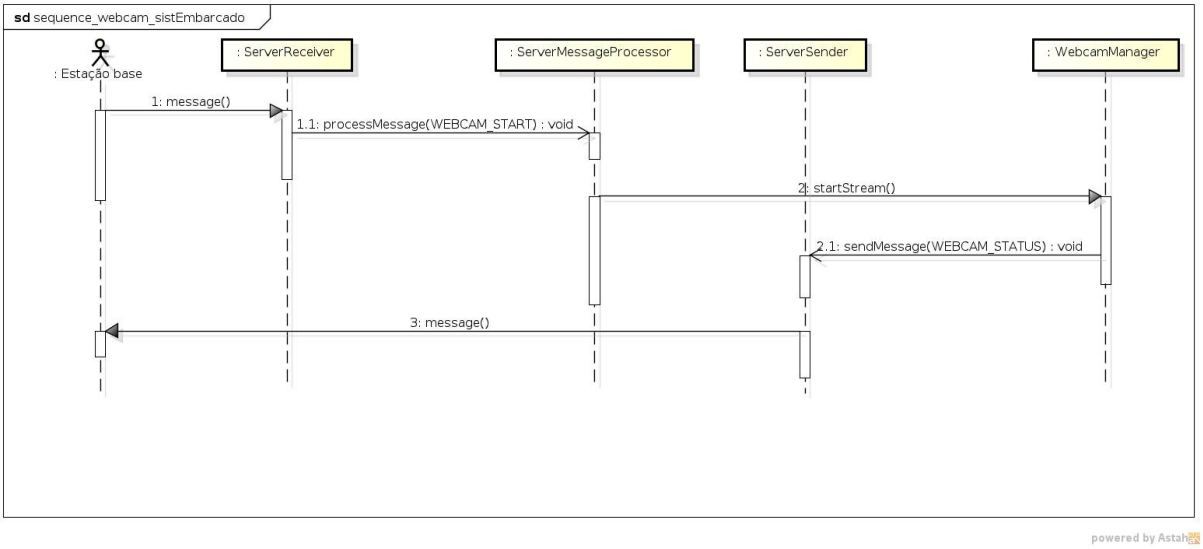

Figura 19: Diagrama de sequência de comando para ativação da webcam (representa mensagem de ativação da webcam chegando no sistema embarcado e estação base sendo posteriormente notificada sobre o novo status).

3 DIAGRAMAS DO HARDWARE

3.1 DIAGRAMA DE BLOCOS

Na figura 20 mostra-se o diagrama de blocos do sistema embarcado e suas conexões com o restante do robô. A seguir está também uma descrição para cada um dos blocos da placa de circuito impresso do sistema embarcado.

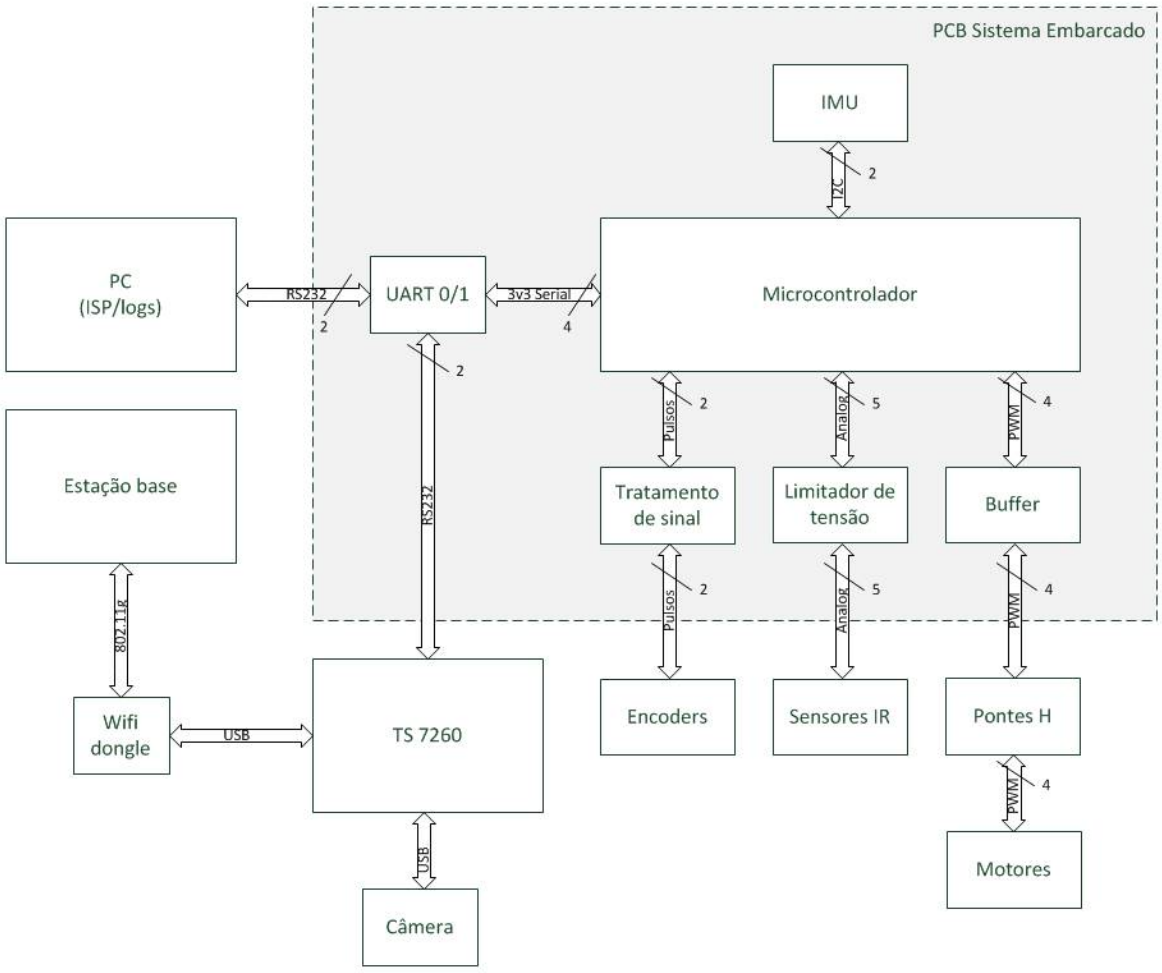

Figura 20: Diagrama de blocos do hardware

1. Microcontrolador: Este bloco fará a leitura dos sensores: encoders, infra-vermelhos, ace-

lerômetro e giroscópios. Além disso possui a implementação do protocolo de comunicação para interação com o linux embarcado da placa TS-7260.

2. UART 0/1: Responsável por ajustar os níveis de tensão para comunicação serial no padrão RS-232 com a placa TS-7260.
3. Buffer: Responsável por fornecer corrente e elevar os níveis de tensão de saída do microcontrolador de 3,3V para 5,0V. Esse buffer é conectado às pontes H já existentes no robô.
4. IMU: possui o acelerômetro e o giroscópio e se comunicará com o microcontrolador por meio do protocolo I2C.
5. Limitador de tensão: Necessário pois os sinais de saída dos sensores de infravermelho que já existem no robô não estão limitados em 5V, podendo a saída ultrapassar 5,0V e danificar o microcontrolador.
6. Tratamento de sinal: Composto por um filtro RC passa baixas e um schmitt trigger para remover qualquer falha que possa ocorrer na geração dos pulsos no encoder.

3.2 DIAGRAMA ELÉTRICO/ELETRÔNICO

Na figura 21 mostra-se o diagrama de elétrico eletrônico do sistema embarcado. Cada bloco da figura 20 corresponde a alguns componentes do diagrama elétrico eletrônico. A seguir detalha-se um pouco mais cada bloco.

1. Microcontrolador: Composto pelo microcontrolador LCP2103 da NXP que possui arquitetura ARM. Ele dispõe de duas interfaces seriais, conversor analógico digital com 8 canais, interface i2c, entradas de captura e interrupção, saídas de PWM entre outras funções que não serão utilizadas nesse projeto.
2. UART 0/1: Constituído por um chip max3232 que opera em níveis de tensão CMOS e que gera os níveis adequados para o padrão RS-232 utilizando alguns capacitores.
3. Buffer: Constituído por um chip 74HC244, é responsável por fornecer corrente e elevar os níveis de tensão de saída do microcontrolador de 3,3V para 5,0V.
4. IMU: Composto pela placa de desenvolvimento MPU-6050, que possui um chip com o mesmo nome, MPU-6050, e circuitos RC auxiliares necessários para o funcionamento do MPU-6050.
5. Limitador de tensão: Constituído de um resistor com baixo valor, 270 ohms, e um diodo Zener polarizado reversamente e com tensão de ruptura de 4.3V. Quando a tensão de

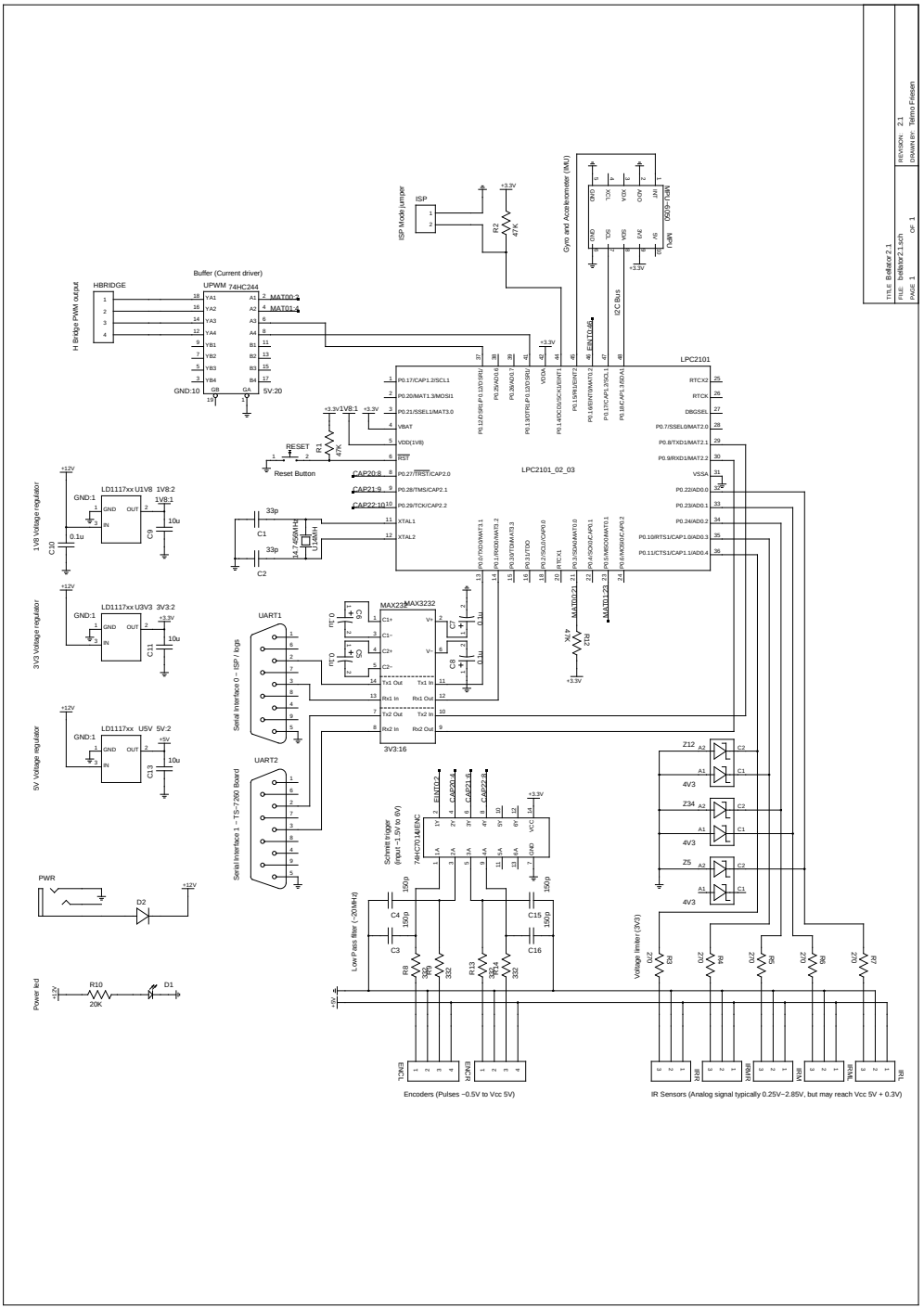

Figura 21: Diagrama elétrico/eletrônico.

entrada ultrapassar 4.3V o diodo passa a conduzir e mantém a tensão de 4.3V no resistor. O datasheet do microcontrolador sugere que se mantenha a impedância da carga menor que 40kohms, logo a adição de um resistor de 270 ohms pode ser desconsiderado com relação ao erro que possa causar na leitura do conversor. Um resistor de 270 ohms leva a uma corrente de 3.7mA quando a saída do sensor for 5.3V, que é o valor máximo previsto no datasheet.

6. Tratamento de sinal: Composto por um filtro RC passa baixas e um chip 74HC7014. As frequências acima de 20MHz são atenuadas no sinal do encoder. Esse valor foi calculado com base na forma de onda da saída especificada no datasheet do encoder. Para tanto utilizam-se resistores de 332 ohms e capacitores de 150pF.

REFERÊNCIAS

MARIN, A. J.; BORGES, J. C. N.; WERGRZN, Y. A. **Desenvolvimento de robô móvel e análise qualitativa de algoritmos de navegação fuzzy**. Curitiba, 2012.

VIDEOLAN. 2013. Disponível em: <http://www.videolan.org/vlc/>.