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

SEGUNDO 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

Segundo entregável apresentada 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 SEGUNDO ENTREGÁVEL - 27/03/2013

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

1. Versão inicial dos diagramas de casos de uso (software embarcado).
2. Versão inicial do diagrama de fluxo de dados (software embarcado).
3. Versão inicial dos diagramas de estados (sistema de comunicação).
4. Versão inicial da descrição das mensagens e codificações dos comandos (sistema de comunicação).
5. Versão inicial do diagrama elétrico/eletrônico (hardware).

Além disso, estão presentes neste documento os seguintes itens adicionais:

1. Versão atual dos requisitos funcionais.
2. Versão atual do diagrama de casos de uso da estação base.
3. Versão atual do diagrama de classes da estação base.
4. Versão atual da descrição das classes da estação base.
5. Versão atual do diagrama em blocos 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).

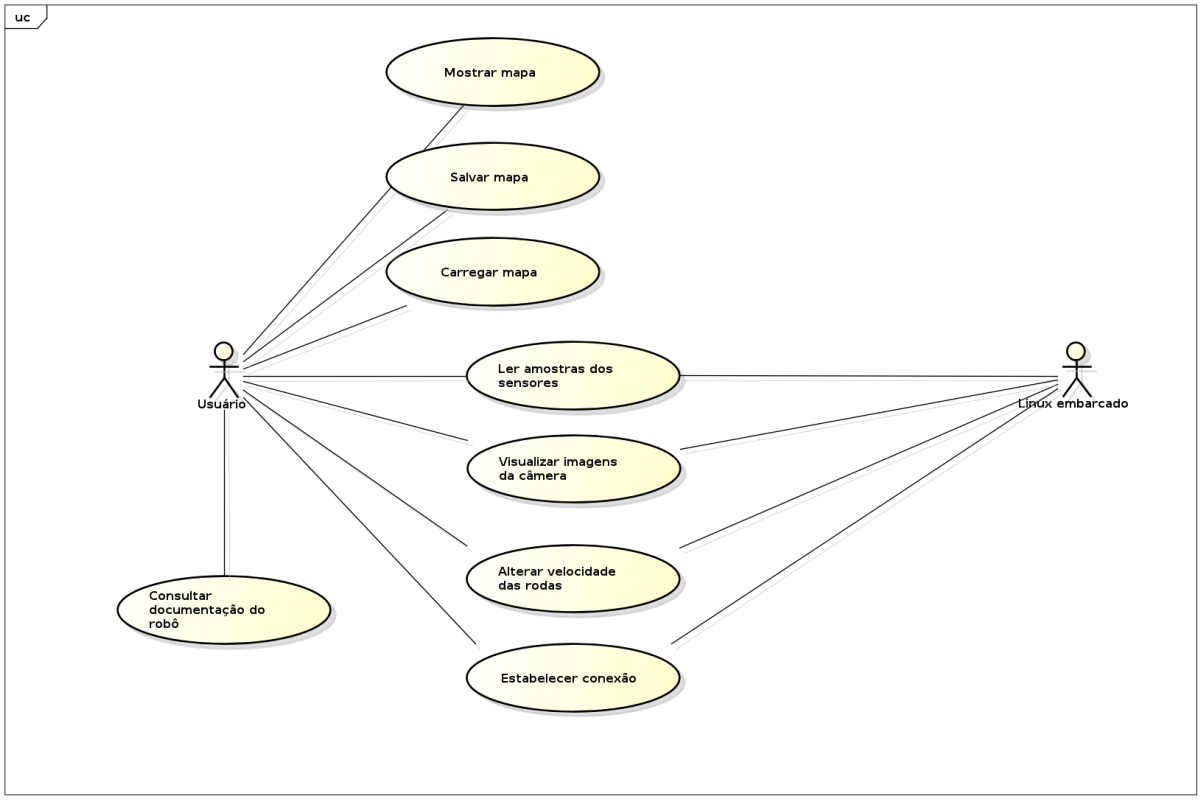

Figura 1: Diagrama de casos de uso do *software* da estação base.

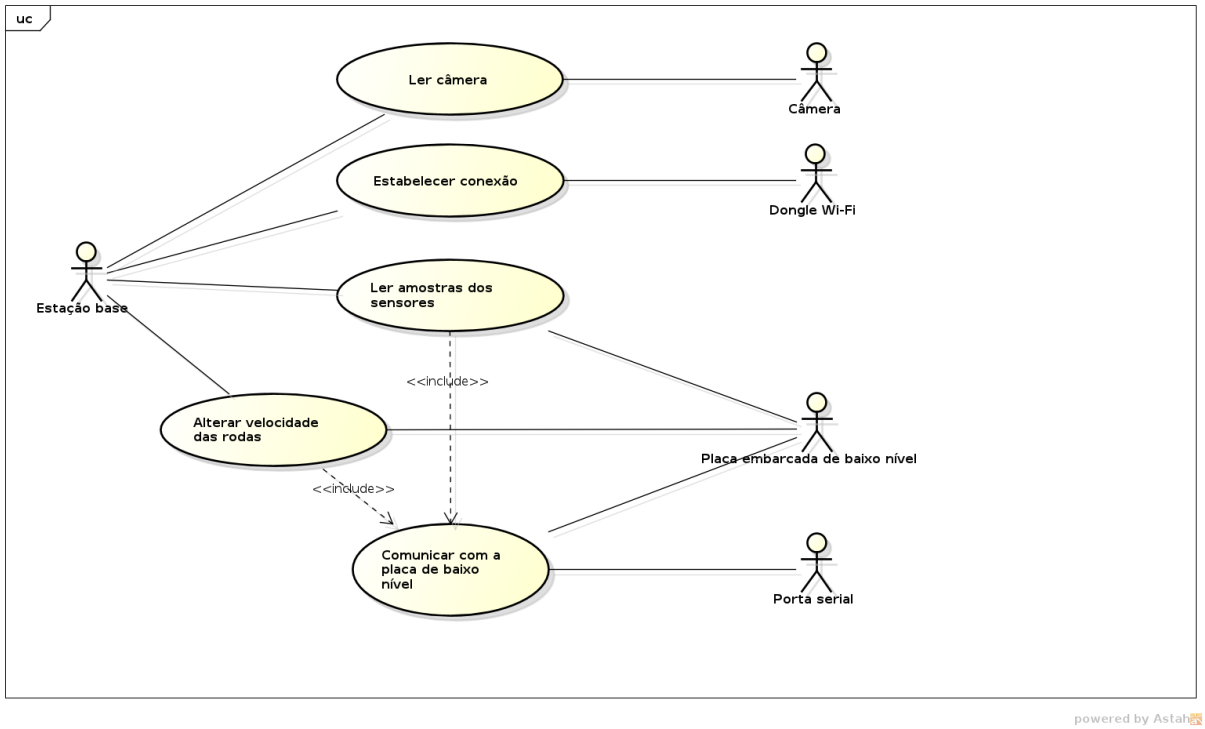

Figura 2: Diagrama de casos de uso do *software* para a placa TS-7260.

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.

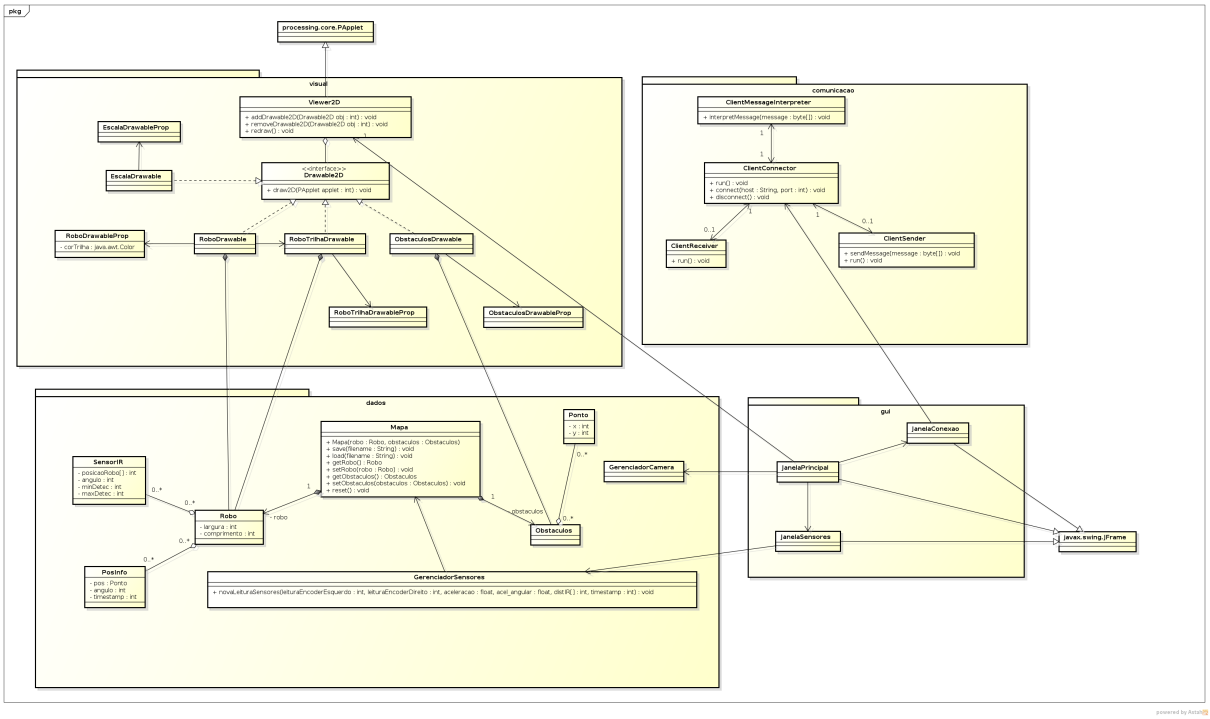

Figura 4: Diagrama de classes

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.

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.

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.

2.4.4 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 5 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 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

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.

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.

2.5 FLUXO DE DADOS

A Figura 5 mostra o diagrama de fluxo de dados.

Tabela 3: Pacote *comunicacao*

Classe	Descrição
ClientMessageProcessor	Responsável pela interpretação dos comandos recebidos pelo cliente.
ClientConnector	Responsável por efetuar a gerência da conexão do cliente (estação base) com o servidor (robô).
ClientReceiver	Responsável por receber mensagens de um host de uma conexão.
ClientSender	Responsável por enviar mensagens ao host de uma conexão.

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.

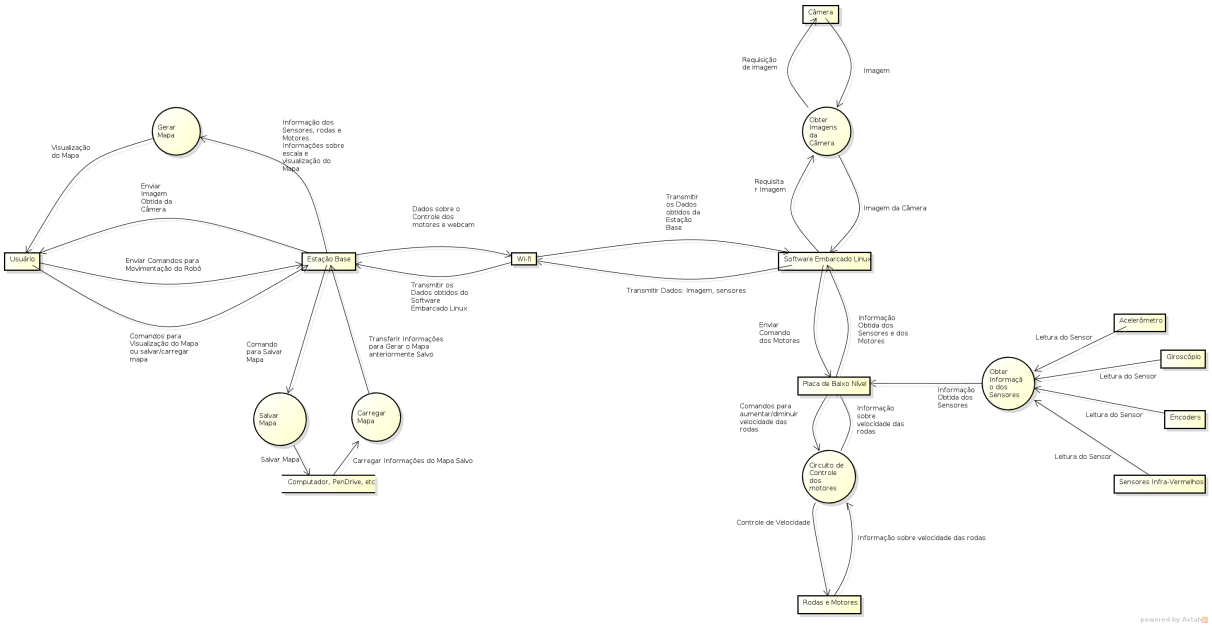

Figura 5: Diagrama de fluxo de dados.

2.6 DIAGRAMA DE CLASSES DO SISTEMA EMBARCADO

A Figura 6 mostra o diagrama de classes do sistema embarcado.

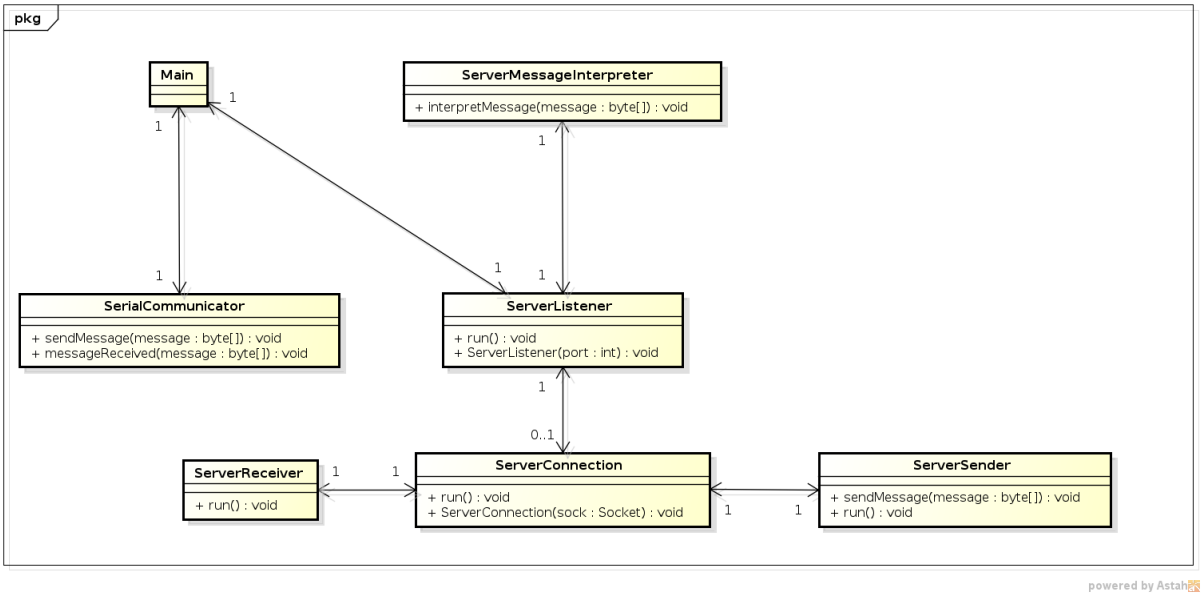

Figura 6: Diagrama de classes

2.6.1 Descrição das classes do software para o sistema embarcado (TS-7260)

Tabela 5: Pacote *comunicacao*

Classe	**Descrição**
Main	Classe principal do robô.
ServerMessageProcessor	Responsável pela interpretação dos mensagens recebidas pelo servidor (robô).
ServerListener	Responsável por escutar as novas conexões de clientes.
ServerSender	Responsável por enviar mensagens ao host de uma conexão.
ServerReceiver	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.

2.7 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 possibilitar que o tráfego de mensagens pudesse 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). No caso da estação base, os diagramas de estados dessa *thread* estão representados na Figura 7. Para o sistema embarcado há a representação da Figura 8.

A segunda *thread* tem a função de gerenciar o envio de mensagens. O programa, ao necessitar enviar uma mensagem, faz uma requisição a essa *thread* que insere-a 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 sequência dessa *thread* está presente na Figura 9.

A terceira *thread* gerencia o recebimento de mensagens. Seu funcionamento é muito simples: ela possui um loop, no qual aguarda até alguma mensagem ser recebida. Quando ocorre o recebimento de alguma mensagem, ela a encaminha para a quarta thread (explicada abaixo) 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 sequência dessa *thread* está presente na Figura 10.

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 2.7.3. Ela possui uma fila, na qual são inseridas as mensagens a serem processadas. Dessa forma o receptor não necessita ficar bloqueado aguardando o término do processamento. O diagrama de sequência dessa *thread* está presente na Figura 11.

2.7.1 Diagramas de estados

Nesta seção estão expostos os diagramas de estados do protocolo de comunicação nas Figuras 7, 8, 9, 10 e 11.

Um aspecto importante a ressaltar é que, nas *threads* de envio (Figura 9) e de processamento de mensagens (Figura 11), 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 a basicamente aos acessos à fila somente, o que reduz consideravelmente a complexidade do processo.

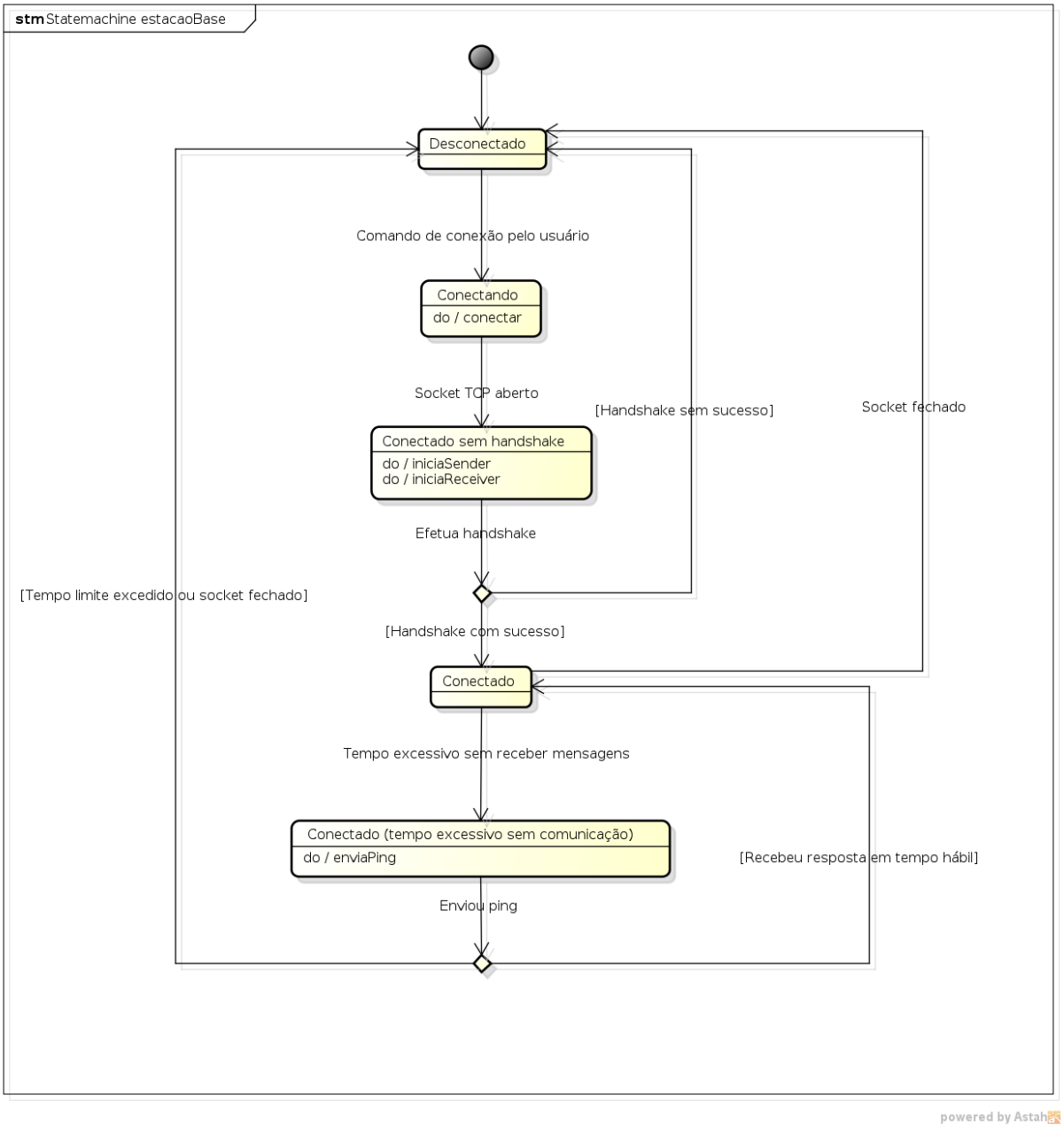

Figura 7: Diagrama estados da *thread* principal da conexão da estação base.

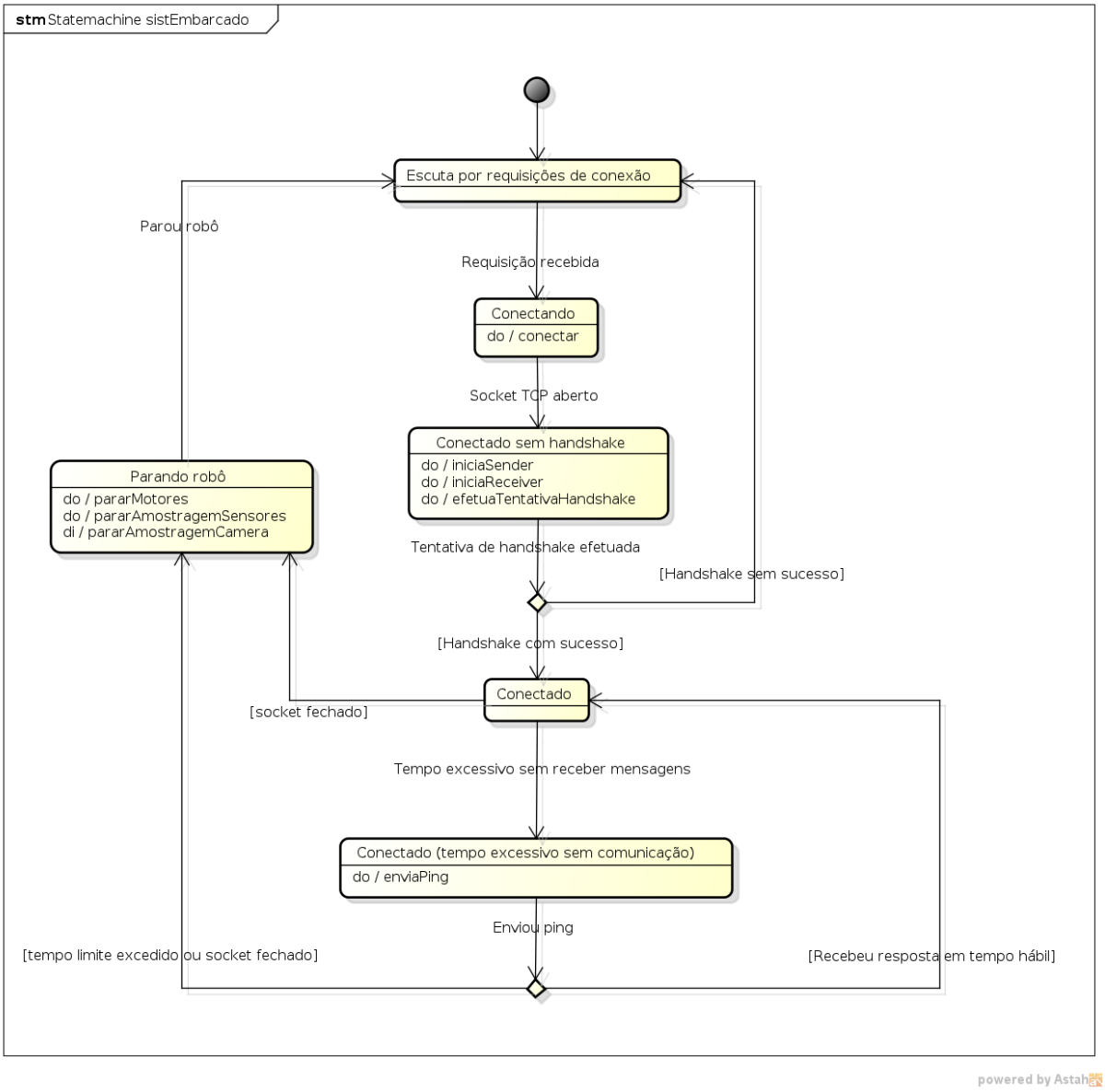

Figura 8: Diagrama de estados da *thread* principal da conexão do sistema embarcado.

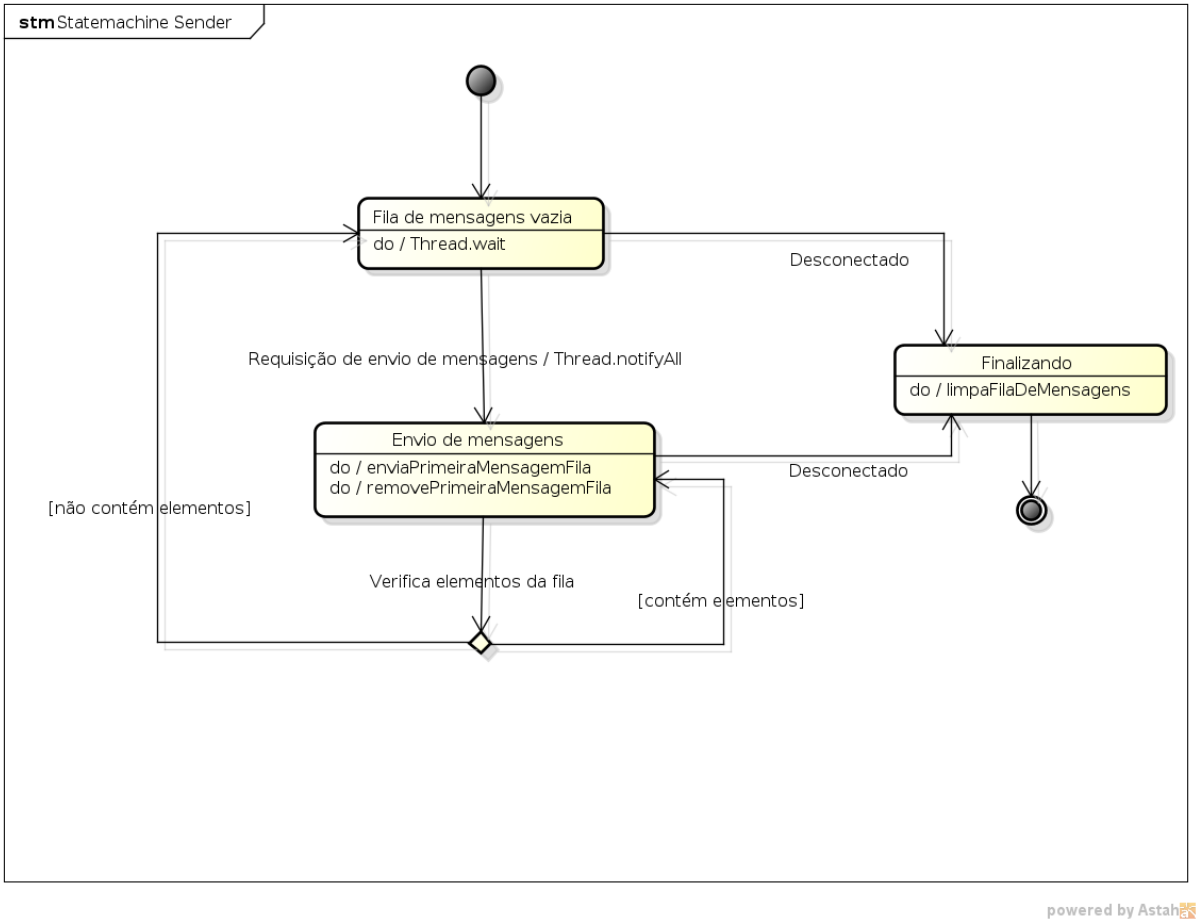

Figura 9: Diagrama de estados da *thread* que envia mensagens (igual para estação base e sistema embarcado).

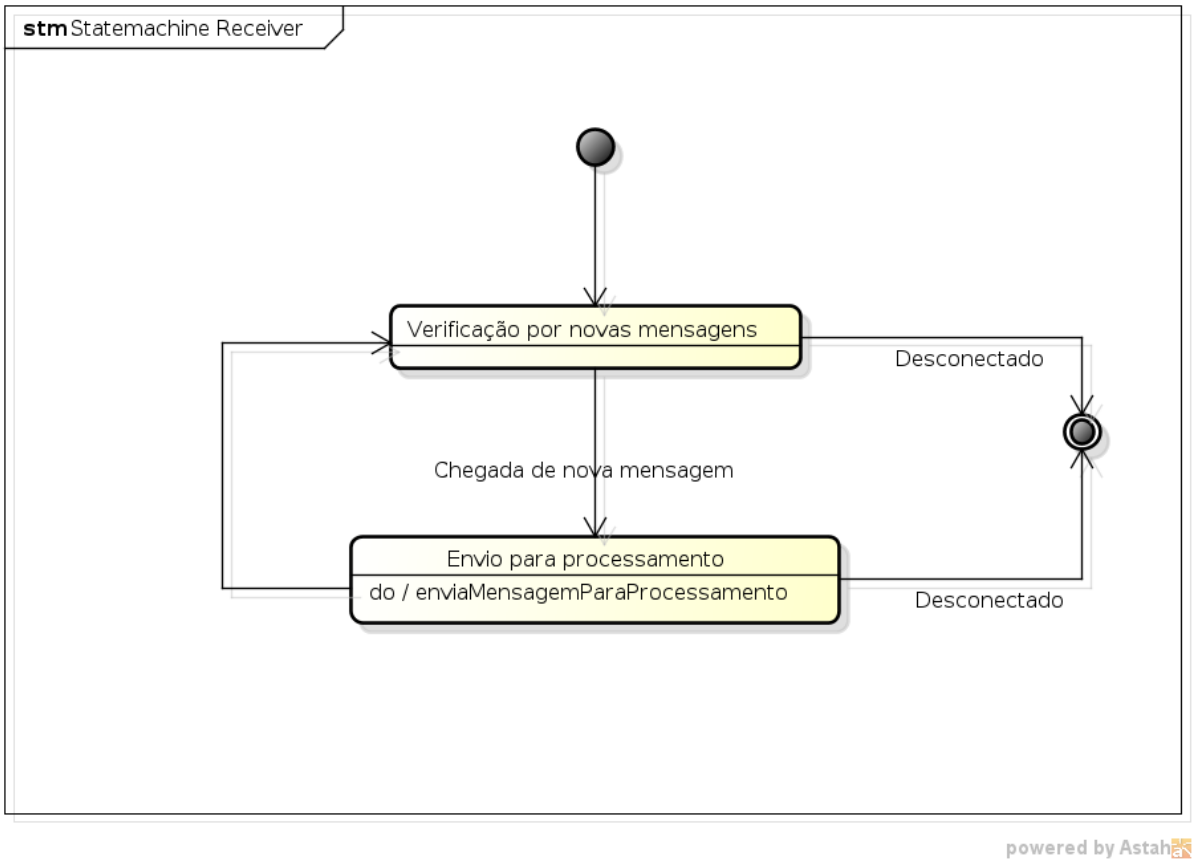

Figura 10: Diagrama de estados da *thread* receptora de mensagens (igual para estação base e sistema embarcado).

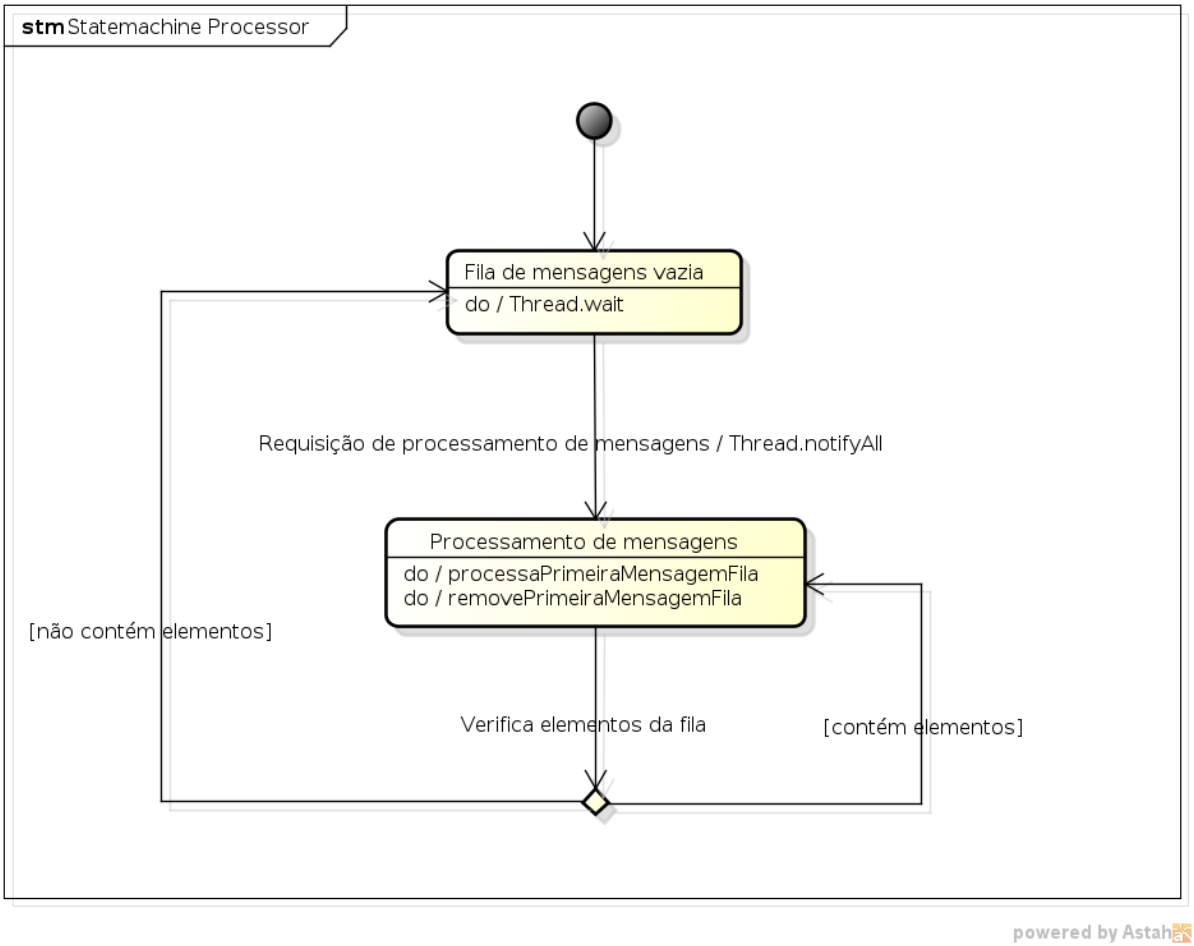

Figura 11: Diagrama de estados da *thread* que processa mensagens recebidas (igual para estação base e sistema embarcado).

2.7.2 Diagramas de sequência

Nessa seção estão presentes as versões iniciais de dois diagramas de sequência. O primeiro (Figura 12) representa como um comando de mudança de velocidade das rodas dado pelo usuário chega até a placa de baixo nível. O segundo (Figura 13) demonstra os dados de leituras dos sensores que saem da placa de baixo nível e chegam até o usuário. Vale ressaltar que nos diagramas foram representadas também as chamadas que são assíncronas, ou seja, que não bloqueiam a execução da *thread* chamadora.

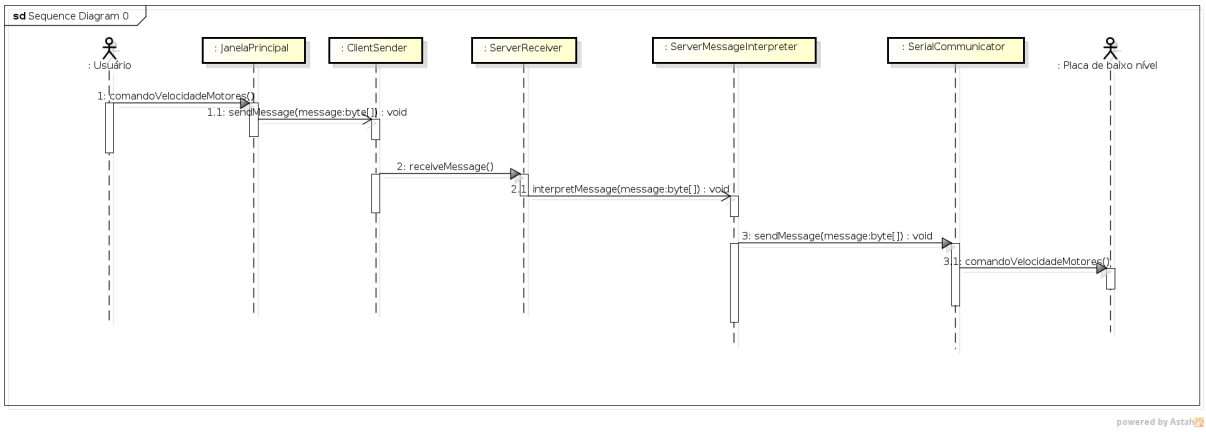

Figura 12: Diagrama de sequência de um comando (dado pelo usuário) de mudança de velocidade dos motores.

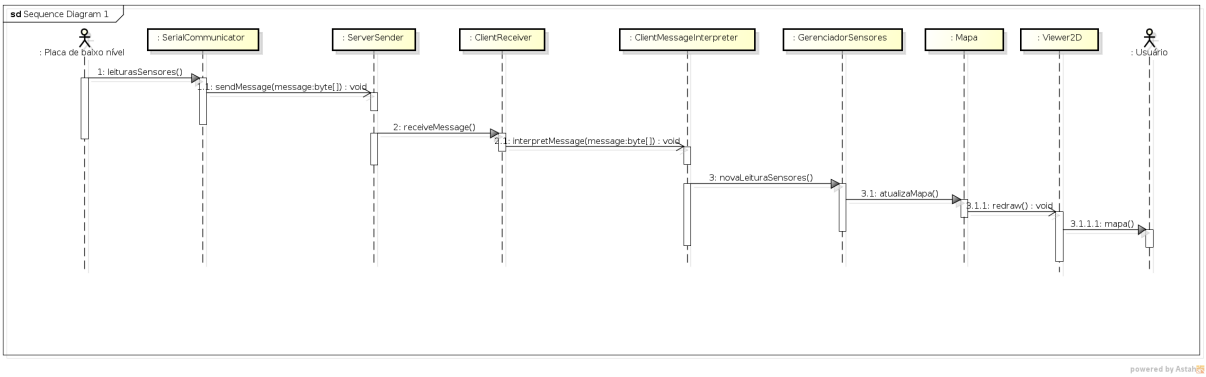

Figura 13: Diagrama de sequência de uma leitura dos sensores vinda da placa de baixo nível.

2.7.3 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 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
    (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 é um contador de 0 a 255 (que zera automaticamente quando chega ao valor máximo), usado para controlar a perda e ordem dos dados.

- 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).
  - **SENSORS_STATUS_REQUEST**
    (byte) END_CMD
    Requisição de status da amostragem dos sensores. Usado na interface gráfica para atualizar as informações sobre os sensores.
  - **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.

  - **WEBCAM_STATUS_REQUEST**

    (byte) END_CMD

    Solicitação de informações sobre status da webcam. Usado na interface gráfica para atualizar as informações sobre a webcam.

  - **ENGINES**

    (byte) vel_roda_esquerda

    (byte) vel_roda_direita

    (byte) END_CMD

    Solicitação de mudança da velocidade dos motores.

  - **ENGINES_STATUS_REQUEST**

    (byte) END_CMD

    Solicitação de status dos motores. Usado na interface gráfica para confirmar o recebimento de comandos de movimentação efetuados pelo usuário.

- 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*,
(long) TIMESTAMP_UNIX
(byte) END_CMD

Possui os mesmos parâmetros da mensagem SENSORS enviada da LPC2103 para a TS-7260, com exceção do timestamp, que é trocado por um timestamp UNIX em milissegundos (que representa a hora do recebimento das leituras na TS-7260). Essa informação de tempo é utilizada pela estação base para efetuar os cálculos de posicionamento do robô.

– **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

– **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.

3 DIAGRAMA DE BLOCOS DO HARDWARE

3.1 DIAGRAMA DE BLOCOS

Na figura 14 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.

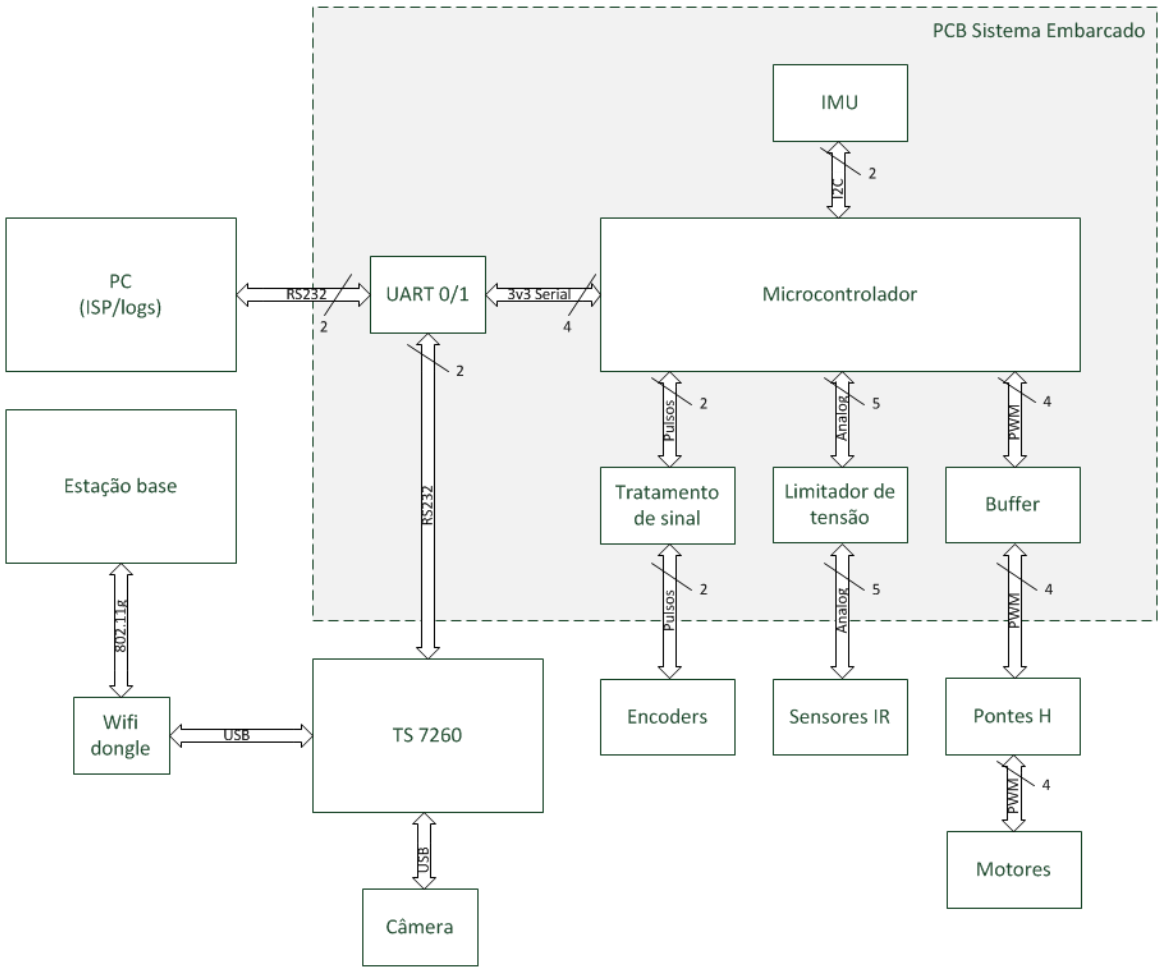

Figura 14: Diagrama de blocos do hardware

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. A frequência de corte do filtro pode ser obtida pela velocidade máxima que o robô pode atingir, que foi suposta em 1 m/s (como apresentado nos requisitos de hardware).

3.2 DIAGRAMA ELÉTRICO/ELETRÔNICO

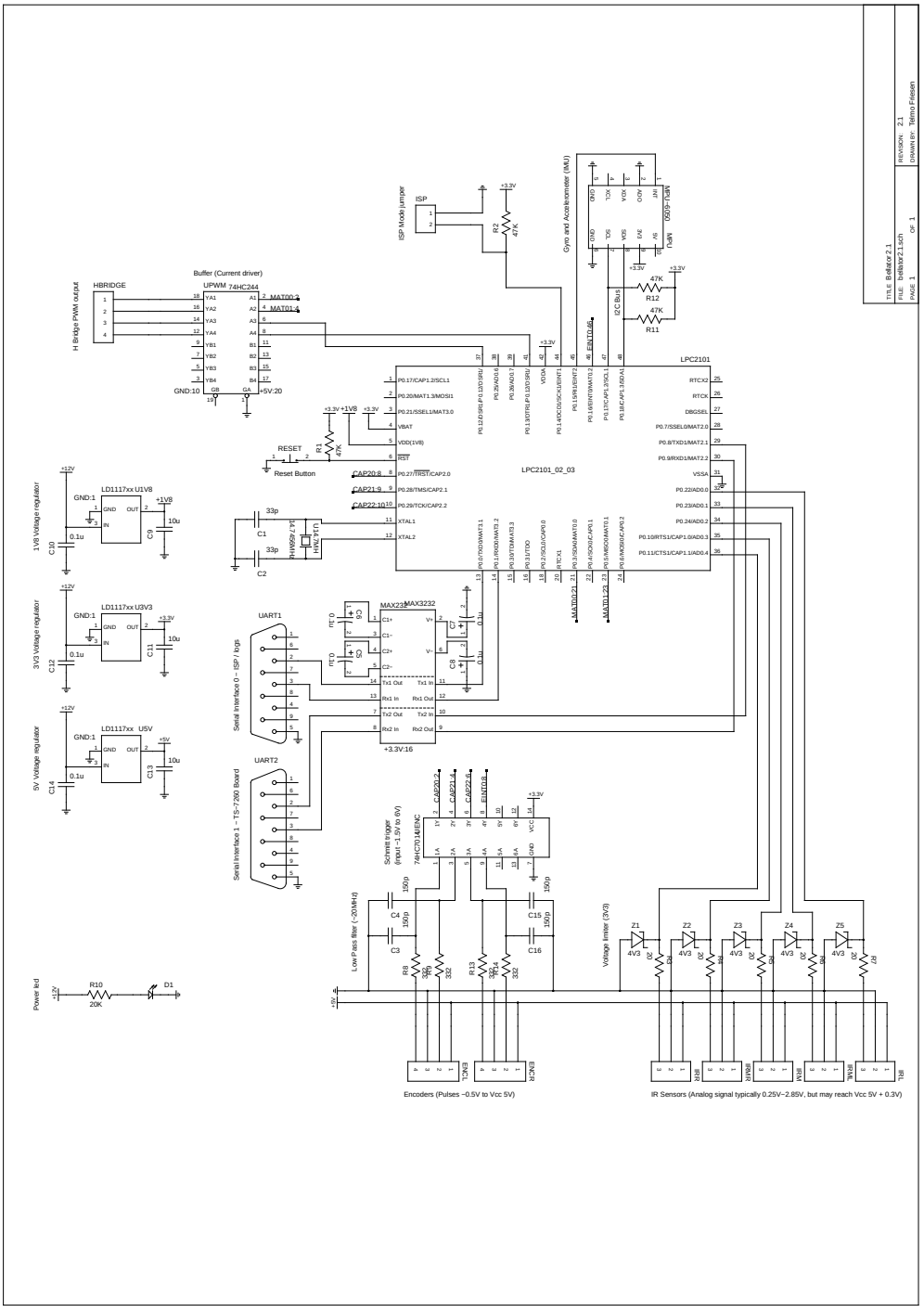

Figura 15: Diagrama elétrico/eletrônico.

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.