Relatório – Etapa 3: Prototipagem e Ajustes

1) Resumo executivo

Nesta etapa, construímos e exercitamos o protótipo funcional do sistema embarcado TinyML para monitorar o estado de um contêiner (ex.: parado, subindo, descendo), exibindo a classe no OLED e publicando o resultado via MQTT. A arquitetura foi organizada em tarefas FreeRTOS (aquisição, inferência, exibição, conectividade), com comunicação por filas/semaforos; o driver do display SSD1306 foi integrado; a base do cliente MQTT foi preparada; e definimos a lógica de baixo consumo quando o sistema detecta “Parado”. Esta etapa também consolidou o plano de testes, medições e próximos ajustes.

Classificação de movimentos (parado, movimento, curva, impacto) usando MPU6050, com telemetria via MQTT, display de status, e gerenciamento de energia.

---

2) Arquitetura do sistema

2.1 Hardware

- **Placa**: BitDogLab (RP2040 / Pico W).
- **Acelerômetro**: MPU6500/MPU6050 via I2C.
- **Display**: OLED SSD1306 128×64 I2C (endereço típico 0x3C).
- **Conectividade**: Wi-Fi do CYW43 (Pico W) para MQTT.
- **Energia**: operação com bateria LiPo (meta de autonomia em RNF).

2.2 Software e organização (FreeRTOS + Pico SDK)

- **Build/CMake**: o projeto compila com Pico SDK + FreeRTOS. O `CMakeLists.txt` inclui `freertos_kernel`, `pico_cyw43_arch_lwip_sys_freertos` e módulos do projeto (`mpu6050.c`, `display_ssd1306.c`, `mqtt_client.c`, `tinyml_engine.c`, etc.).
- **Configuração do RTOS**: `FreeRTOSConfig.h` habilita *tickless idle* e o gancho de *idle* (`configUSE_IDLE_HOOK 1`), base para economia de energia.
- **Tarefas principais**:
    - **mpu_task**: lê o acelerômetro e envia janelas para a fila da ML.
    - **ml_task**: executa pré-processamento + inferência (TinyML) e envia a classe para a fila do display e para publicação MQTT.
    - **display_task**: recebe a classe e atualiza o SSD1306.
    - **wifi_task**: gerencia conexão Wi-Fi.

- **mqtt_task**: publica resultados no broker após o Wi-Fi sinalizar disponibilidade.
- **system_monitor/power_manager**: monitora saúde/estado e aplica políticas de energia.

2.3 Comunicação entre tarefas

- **Semáforo** entre `wifi_task` → `mqtt_task`: MQTT só inicia/publica depois do Wi-Fi estar conectado (evita tentativas em vão e desperdício).
- **Fila** entre `ml_task` → `display_task`: desacopla inferência (rápida) da atualização do display (I2C, mais lenta) e serializa acesso ao OLED.

---

3) Implementações e estado atual

3.1 FreeRTOS – tarefas concorrentes

- **Status**: **implementado/parcialmente integrado**.
- **Evidências**:
  - *Build*: `CMakeLists.txt` linka `freertos_kernel` e a *arch* do CYW43 com FreeRTOS.
  - *Config*: `FreeRTOSConfig.h` com *tickless idle* e *idle hook* ativos, que dão base para economia de energia.
- **Detalhes de projeto**:
  - Prioridades: aquisição (alta) → ML (média/alta) → display (baixa/média).
  - Pilhas dimensionadas para evitar *stack overflow* em ML e MQTT.
  - Fila de resultados ML: tamanho ≥ 4 janelas; mensagens com `{classe, confiança, timestamp}`.
  - Semáforo binário: Wi-Fi **dá** (give) quando conectado; MQTT **espera** (take) antes de publicar.

3.2 Aquisição de dados – MPU6500 (I2C, ±2g)

- **Status**: **implementado** (driver referenciado no build) e validado em protótipo.
- **Config recomendada**:
  - Endereço I2C: 0x68 (AD0=0) ou 0x69 (AD0=1).
  - **Escala ±2g**: registrar `ACCEL_CONFIG` (0x1C) com `AFS_SEL=0`.
  - Filtro/DLPF: `CONFIG` (0x1A) ajustado para reduzir ruído (ex.: 44–94 Hz de *bandwidth* dependendo do *sample rate*).

    - Taxa de amostragem: 50–100 Hz (ex.: `SMPLRT_DIV` para compatibilizar com o *clock* base do sensor).

- **Boas práticas aplicadas**:
    - **Calibração inicial** (offsets) com o contêiner parado;
    - Cálculo de **magnitude** e normalização;
    - **Janelamento** de N amostras (ex.: 1–2 s) para a ML.

3.3 Display – SSD1306 (classe detectada)

- **Status**: **implementado** (driver presente).
- **Fluxo**:
    1. `display_init()` configura I2C e o SSD1306;
    2. `display_show_class("Parado" | "Subindo" | "Descendo" | …)` atualiza o *framebuffer*;
    3. `display_update()` envia para o OLED.
- **Projeto de UX**: texto centralizado, ícone/indicador de Wi-Fi, e (opcional) confiança em %.

3.4 MQTT – conexão e publicação (QoS 2)

- **Status**: **base pronta / em ajustes**.
    - O módulo `mqtt_client.c` está estruturado; a pilha de rede é a do **lwIP** integrada ao CYW43 sob FreeRTOS (conforme o *toolchain* linkado no CMake).
    - O requisito da Etapa 3 pede **Paho MQTT** com **QoS 2**. Planejamos duas opções:
        1. **Manter lwIP MQTT** para estabilizar a fase (mais simples para Pico W) e,
        2. **Migrar para Paho Embedded-C** garantindo QoS 2 e *callbacks* de confirmação (**PUBREC/PUBREL/PUBCOMP**).
- **Tópicos**: `containers/<id>/estado` (telemetria), `containers/<id>/debug` (logs opcionais).
- **Reconexão**: retentativa exponencial e *keep alive* de 30–60 s.

3.5 TinyML – pipeline e motor de inferência

- **Status**: **parcial**.
    - O módulo `tinyml_engine.c/h` já expõe uma API (`tinyml_engine_init`, `tinyml_engine_classify`).

- **Durante a prototipagem**, está ativo um **classificador heurístico *mock***: ele calcula a magnitude do acelerômetro na janela e usa limiares para rotular (ex.: *Parado* abaixo de um limiar, *Movimento* acima etc.). Isso permite validar o fluxo end-to-end (tarefas, filas, display, MQTT) **antes** da integração do modelo final.

- **Próximo passo**: coletar dados + treinar no **Edge Impulse**, exportar o *C++ library* (ou CMSIS-NN/TFLM) e encaixar no `tinyml_engine.c`, mantendo a mesma assinatura da API.

3.6 Lógica de baixo consumo (sleep, redução de taxa e Wi-Fi)

- **Status**: **em andamento**.

  - Com **tickless idle** e *idle hook* habilitados no `FreeRTOSConfig.h`, o RP2040 entra em espera quando não há tarefas prontas (base para economia).

  - Política: se a classe **"Parado"** persistir por T segundos, a **mpu_task** reduz a taxa de amostragem (ex.: 10–20 Hz), e a conectividade entra em modo econômico (desassoc/desliga rádio temporariamente). Ao detectar vibração novamente, tudo volta ao modo normal.

---

4) Processo TinyML (Edge Impulse)

1. **Coleta de dados**

   - Casos planejados: *Parado, Subindo (elevação), Descendo, Vibração/Andando* (ajuste final conforme cenário real).

   - Captura em janelas com *label* e metadados (temperatura/posição se disponível).

2. **Pré-processamento no Edge Impulse**

   - Filtros (passa-baixa) e **features**: RMS, variação, picos e espectro (FFT), magnitude.

   - *Windowing* com sobreposição para não perder eventos.

3. **Desenho do modelo**

   - Opções: **KNN/Lite** (baixo custo), **SVM** linear, ou **CNN 1D** leve (com TFLM).

   - Métrica-alvo: **acurácia ≥ 80%** e **latência < 200 ms**

4. **Treino e avaliação**

   - *K-fold* ou *hold-out*, *confusion matrix*, curva **precision/recall** por classe.

   - *Ablation* de features para simplificar sem perder acurácia.

5. **Conversão/Exportação**

   - Export **C++ library** para integração; validar RAM/Flash no RP2040.

6. **Integração e inferência no firmware**
    - Substituir o *mock* por `run_classifier()` (ou equivalente da lib exportada) dentro da `ml_task`.
    - Validar tempo de inferência (ms), consumo e robustez.

---

5) Plano de testes da prototipagem

5.1 Testes unitários

- **Aquisição**: verificar frequência real (ex.: 50/60/100 Hz) e ausência de *missed samples*.
- **Pré-processamento**: conferir janela, normalização e features (comparar com Edge Impulse).
- **Display**: garantir atualização correta da classe e legibilidade.
- **MQTT**: publicar com QoS 2 e conferir **PUBREC/PUBREL/PUBCOMP** no broker (Mosquitto).

5.2 Testes de integração

- Fluxo **end-to-end**: MPU → ML → fila → Display e MQTT.
- Reconexão Wi-Fi e robustez a *timeouts*.
- Múltiplos ciclos de “Parado → Movimento → Parado” validando o modo de economia.

5.3 Testes de campo

- Roteiros práticos:
    1. Contêiner **parado** por ≥ 60 s → deve reduzir taxa e rádio.
    2. **Subida/descida** (inclinação) → deve classificar corretamente e publicar.
    3. **Vibração/andando** → manter taxa normal e telemetria contínua.
- Medidas de **acurácia** por classe e **latência** (amostra→OLED/MQTT).

---

6) Métricas e consumo (o que medir e como registrar)

Observação: abaixo estão **campos que vamos preencher com números reais** após rodar o protocolo de testes.

Acurácia global: ____ %

- **Acurácia por classe**:
    - Parado: ____ %
    - Subindo: ____ %
    - Descendo: ____ %
    - Vibração/Andando: ____ %
- **Latência média (janela → decisão)**: ____ ms
- **Tempo de publicação MQTT (decisão → broker)**: ____ ms
- **Consumo** (corrente média):
    - Modo **ativo** (Wi-Fi on, amostragem normal): ____ mA
    - Modo **economia** (classe *Parado*, taxa reduzida, rádio off/eco): ____ mA
    - **Ganho estimado de autonomia**: ____ %
- **Estabilidade**: horas contínuas sem *reset* / *watchdog*: ____ h

---

7) Desafios encontrados e correções

1. **Integração FreeRTOS e Pico W**
    - Ajustes no CMake para linkar `freertos_kernel` e a *arch* correta do CYW43 sob FreeRTOS.
    - Inclusão do `FreeRTOSConfig.h` no *include path* para silenciar *warnings* dos headers.
2. **Driver do MPU6500**
    - Filtro e escala ajustados (±2g) para maximizar separabilidade entre “Parado” e “Movimento”.
    - Calibração inicial para remover *bias* de gravidade.
3. **Display SSD1306**
    - *Frame buffer* e atualização centralizada por *queue* para evitar *tearing* e travamentos na I2C.
4. **MQTT (QoS 2)**
    - Base com lwIP estabilizada; **pendente** migrar/confirmar Paho Embedded-C e fechar o fluxo de **QoS 2** com *callbacks* de confirmação.
5. **Energia**
    - **Tickless idle** ativado; política de redução de taxa a partir de “Parado” implementada.

- Próximo: medir consumo real e avaliar desligamento temporal do rádio.

---

8) Próximos passos (até a entrega final)

- **MQTT/Paho**: integrar Paho Embedded-C com **QoS 2** e reconexão robusta.
- **Modelo Edge Impulse**: coletar dados reais, treinar e **substituir o mock** no `tinyml_engine`.
- **Telemetria**: payload JSON padronizado, *timestamps* e IDs de dispositivo.
- **Consumo**: medir com USB meter; registrar **mA** nos dois modos e estimar autonomia.
- **Documentação**: fotos/vídeo do protótipo em funcionamento, README atualizado com números.

---

9) Evidências do protótipo

- **Código-fonte**:
  1. `CMakeLists.txt` com FreeRTOS + CYW43 (Wi-Fi) integrados.
  2. `FreeRTOSConfig.h` com tickless e idle hook (base de economia).
  3. `display_ssd1306.[ch]` (exibição da classe).
  4. `mqtt_client.c` (base para publicação).
  5. `tinyml_engine.[ch]` (API pronta; *mock* de inferência funcional).
- **Demonstração esperada**:
  1. Ligar o sistema → Wi-Fi conecta;
  2. Mexer o conjunto (simular *subindo/descendo*) → OLED mostra a classe;
  3. Publicação MQTT no broker;
  4. Parar o sistema por T s → entra em modo econômico; ao mover novamente, volta ao normal.

---

10) Conclusão

A prototipagem confirmou a **viabilidade** do fluxo completo (aquisição → ML → exibição → MQTT) e estabeleceu a base da **otimização de energia**. Falta **substituir o classificador mock** pelo modelo treinado no Edge Impulse e **consolidar QoS 2 com Paho MQTT** para cumprir integralmente a especificação da Etapa 3. O plano de testes e métricas está definido; com as próximas medições, fechamos o relatório final com números de acurácia, latência e consumo.