**Descrição Técnica – Comunicação sem fios com Arduino UNO R4 WiFi, HuskyLens e Simulação de Controlo de Carro**

**1. Introdução**

Este documento descreve a fase de testes de um sistema embarcado baseado no Arduino UNO R4 WiFi, com vista ao desenvolvimento de um carro autónomo que integra reconhecimento visual com a câmara HuskyLens e controlo de movimento com o driver de motores L298P. Embora o veículo físico ainda não esteja montado, a estrutura lógica, as comunicações e os testes de simulação estão a ser desenvolvidos e validados em bancada.

**2. Objetivo**

* Estabelecer uma comunicação Wi-Fi funcional entre o Arduino UNO R4 WiFi e um cliente externo (navegador web).
* Ligar e comunicar com a HuskyLens via UART, simulando decisões baseadas em reconhecimento visual.
* Simular a ativação dos motores através de LEDs ligados aos pinos digitais do Arduino (substituindo temporariamente os motores reais).
* Preparar a infraestrutura base para controlo remoto do sistema, sem necessidade de ligação física por cabo USB após upload do firmware.

**3. Componentes utilizados**

| **Componente** | **Função** |
| --- | --- |
| Arduino UNO R4 WiFi | Microcontrolador principal com conectividade Wi-Fi |
| HuskyLens | Sensor de visão artificial com reconhecimento de cor |
| Driver L298P | Controlador de motores (ligação opcional nesta fase) |
| LED vermelho | Simulação do motor esquerdo |
| LED verde | Simulação do motor direito |
| Resistências 220Ω | Para limitar corrente dos LEDs |
| Jumpers / Breadboard | Montagem do circuito em bancada |

**4. Arquitetura do sistema**

**4.1. Comunicação com a HuskyLens**

* Utiliza a interface UART (Serial) através da biblioteca SoftwareSerial.
* Pinos sugeridos: D8 (RX), D9 (TX).
* Modo de operação da HuskyLens: Reconhecimento de Cor ou Tag ID.
* O Arduino envia comandos e lê as respostas para tomar decisões.

**4.2. Comunicação Wi-Fi**

* A placa UNO R4 WiFi usa o chip ESP32-S3 como coprocessador de conectividade.
* Utiliza a biblioteca WiFiS3.h para configurar uma ligação de rede local (Access Point ou Cliente) e disponibilizar um Web Server HTTP.
* O utilizador interage com o sistema através de um navegador, enviando comandos como:
  + /start
  + /stop
  + /status

**4.3. Simulação de controlo de motores**

* LEDs conectados aos pinos D5 e D6 representam os motores esquerdo e direito, respetivamente.
* A lógica do sistema acende/apaga os LEDs conforme os comandos recebidos ou decisões baseadas nos dados da HuskyLens.

**5. Fluxo de funcionamento**

1. O utilizador acede a uma página web hospedada pelo Arduino via Wi-Fi.
2. Seleciona um comando (start, stop, status) através de um botão ou URL.
3. O Arduino interpreta esse comando e executa uma ação:
   * No modo start, inicia a lógica de deteção da HuskyLens.
   * Lê um ID de cor ou tag.
   * Liga LEDs para simular motores com base nessa deteção.
4. O sistema responde com uma mensagem no navegador (feedback).
5. Tudo é testado em bancada, sem necessidade de estrutura mecânica do carro.

**6. Vantagens desta abordagem**

* Permite testar e validar toda a lógica de controlo **antes da montagem física** do robô.
* Explora o potencial do **Arduino UNO R4 WiFi** como nó IoT de controlo.
* Estabelece uma base sólida para futuras expansões:
  + Motores reais
  + Lógica de navegação
  + Registo de dados
  + Aprendizagem automática

**7. Próximos passos**

* Ligar fisicamente os motores ao L298P e testar com alimentação externa.
* Montar a estrutura física do carro.
* Expandir a lógica de controlo com base em distâncias, ângulos ou múltiplas tags.
* Implementar uma interface gráfica mais rica (ex: dashboard com estado e telemetria).
* Iniciar integração com ROS 2 (numa fase posterior).

**📡 Comunicação com o Arduino UNO R4 WiFi**

**✅ 1. Comunicação via Wi-Fi**

O Arduino UNO R4 WiFi tem um **co-processador ESP32-S3** que permite comunicações sem fios **via Wi-Fi (e opcionalmente Bluetooth LE)**. Existem **duas formas principais** de comunicação com o Arduino usando **Wi-Fi**:

**🟦 Opção A – Web Server no Arduino (HTTP Server)**

**📌 Como funciona:**

* O Arduino cria um **servidor web** (local ou ponto de acesso).
* O utilizador envia **pedidos HTTP** (ex: /start, /stop) a partir de um browser, app ou outro dispositivo.
* O Arduino interpreta o pedido e executa ações (ex: liga motor, lê sensor).

Uma imagem com texto, captura de ecrã, Tipo de letra, branco

Os conteúdos gerados por IA podem estar incorretos.

**✅ Vantagens:**

* Simples de implementar.
* Compatível com qualquer dispositivo (PC, telemóvel, tablet).
* Interface fácil: podes criar páginas HTML com botões.
* Não precisa de software no lado do cliente (apenas browser).

**❌ Desvantagens:**

* Baseado em requisições (não é comunicação contínua).
* Lento para aplicações em tempo real (latência entre pedido e resposta).
* Apenas funciona numa direção por padrão: cliente → Arduino (mas pode enviar resposta).

**🟨 Opção B – Cliente Socket / TCP ou UDP (Comunicação contínua)**

**📌 Como funciona:**

* O Arduino liga-se como **cliente** a um **servidor remoto** (ou vice-versa).
* Trocamos dados **em tempo real** (ex: comandos, estados, telemetria).
* Usa protocolos TCP (fiável) ou UDP (rápido, mas sem garantias).

**🔧 Exemplo prático:**

* Um programa Python no PC envia comandos TCP para o Arduino.
* O Arduino responde imediatamente com o estado do sistema.

**✅ Vantagens:**

* Comunicação **em tempo real** (baixa latência).
* Permite envio e receção contínuos de dados.
* Ideal para sistemas de controlo ou robôs com feedback frequente.

**❌ Desvantagens:**

* Mais complexo de programar.
* Requer software no lado do cliente (ex: app ou script em Python).
* Pode ser bloqueado por firewalls ou routers.

**🔷 2. Comunicação via Bluetooth – É possível no UNO R4 WiFi?**

**❗Resposta curta: Sim, mas com limitações.**

O ESP32-S3 suporta **Bluetooth Low Energy (BLE)**, mas:

* O **Bluetooth não está diretamente acessível na IDE Arduino tradicional** ainda (em julho 2025).
* É possível usar BLE com bibliotecas **alternativas** e via firmware personalizado no ESP32-S3.
* A funcionalidade ainda está **em desenvolvimento nas bibliotecas Arduino oficiais** para o UNO R4 WiFi.

**🔧 Exige:**

* Firmware customizado no ESP32-S3 (ou suporte oficial da ArduinoBLE.h, que ainda está instável).
* Gestão de comunicação RA4M1 ↔ ESP32-S3 (via comandos UART internos).
* App ou cliente BLE no telemóvel (ex: nRF Connect, apps personalizadas).

**✅ Vantagens do Bluetooth:**

* Funciona **sem rede Wi-Fi** (útil para ambientes fechados).
* **Menor consumo de energia**.
* Comunicação ponto-a-ponto direta com o telemóvel.

**❌ Desvantagens do Bluetooth:**

* BLE é mais limitado que o Bluetooth clássico.
* Dificuldade de configuração no UNO R4 WiFi (ainda pouco suportado).
* Menor largura de banda e alcance.
* Mais difícil de programar que HTTP.

**📘 Registo de Desenvolvimento – AzorCAR (Etapas 1 a 3)**

**✅ Etapas já concluídas**

**🔹 Etapa 1 – Web Server com controlo de LEDs**

* Implementado um **servidor web no Arduino UNO R4 WiFi** (ESP32-S3) com a biblioteca WiFiS3.h.
* Criada uma **interface web minimalista** acessível via browser (em PC ou telemóvel na mesma rede).
* Quatro LEDs foram usados para simular o estado do robô:
  + **Verde** – sistema ON
  + **Vermelho** – modo PARAR
  + **Amarelo** – motor ESQUERDO
  + **Azul** – motor DIREITO
* Comandos HTTP como /start e /stop ligam e desligam os LEDs.
* Código foi documentado com comentários pedagógicos, ideal para alunos.

**🔹 Etapa 2 – Comunicação com HuskyLens (Color Recognition)**

* Estabelecida comunicação entre o Arduino e a **HuskyLens** usando SoftwareSerial nos pinos 8 (RX) e 9 (TX).
* A HuskyLens foi configurada no modo **Color Recognition** com três cores treinadas:
  + ID 1 – virar à esquerda (amarelo)
  + ID 2 – virar à direita (azul)
  + ID 3 – parar (vermelho)
* Testes mostraram reconhecimento fiável das cores.
* No monitor serial, começaram a surgir os IDs detetados, provando a comunicação estável.

**🔹 Etapa 3 – Integração Web + HuskyLens (Modo Visão)**

* Juntou-se o Web Server com a lógica de decisão da HuskyLens.
* A interface web permite agora **ligar/desligar o modo visão** remotamente (via /start ou /stop).
* Quando o modo está **ativo**, o Arduino:
  + Lê os IDs da HuskyLens continuamente
  + Acende os LEDs correspondentes
  + Mostra no Serial Monitor o ID e a decisão (esq, dir, parar)
* Adicionada lógica de **funcionamento offline**:
  + Se não houver Wi-Fi, o sistema ativa automaticamente o modo visão local.
  + Isto permite que os testes continuem mesmo sem router ou rede.

**🔮 Possibilidades futuras com ROS 2**

**🔸 Comunicação Serial com ROS 2 (via USB)**

* O Arduino UNO R4 pode ser ligado ao Raspberry Pi 5 via **USB (UART)**.
* O Arduino envia dados (Serial.println("ID: 2")), e o Raspberry corre um nó Python em ROS 2 que:
  + Lê os dados
  + Publica num tópico (ex: /comando\_azorcar)
* Ideal para transmitir decisões do Arduino para o sistema ROS.

**🔸 Enviar comandos do ROS para o Arduino**

* No futuro, o Raspberry pode enviar comandos de volta (por exemplo, /cmd\_vel) e o Arduino atua localmente nos motores.
* Permite um ciclo completo de controlo: visão no Arduino → decisão ROS → ação nos motores.

**🔸 Comunicação via Wi-Fi**

* O Arduino pode atuar como **cliente Wi-Fi** e enviar dados para um **servidor Flask** no Raspberry Pi.
* Ou receber comandos remotos por socket TCP.
* Alternativa útil se não houver cabo USB disponível, embora mais sensível a falhas de rede.

**🔸 micro-ROS (opcional, avançado)**

* Como o Arduino UNO R4 WiFi tem ESP32-S3, é tecnicamente capaz de correr **micro-ROS**.
* Requer desenvolvimento com o **ESP-IDF** e não é suportado no Arduino IDE diretamente.
* Apenas recomendável para projetos ROS mais avançados e tempo disponível.

**🧭 Resumo técnico atual**

* ✔️ Interface Web funcional com Wi-Fi e comandos remotos.
* ✔️ Comunicação estável com HuskyLens (modo reconhecimento de cor).
* ✔️ Simulação clara com LEDs para cada estado.
* ✔️ Capacidade de funcionar offline.
* 🚀 Pronto para integrar com ROS 2 via USB Serial (com nó Python).

**🎯 Seguimento de Objeto com Servo e HuskyLens**

Nesta fase do projeto AzorCAR, foi implementado um sistema de seguimento horizontal de objetos utilizando a câmara HuskyLens em modo *Object Tracking* e um servo motor ligado ao Arduino UNO R4 WiFi.

**📸 Princípio de funcionamento**

A HuskyLens deteta e rastreia um objeto em tempo real, fornecendo as coordenadas X e Y do centro do objeto detetado. Apenas o valor X é utilizado, uma vez que o objetivo é controlar a rotação horizontal da câmara montada num servo motor.

**🧭 Adaptação do Referencial (HuskyLens)**

A imagem da HuskyLens tem resolução:

* X: 0 a 320 (esquerda → direita)
* Y: 0 a 240 (cima → baixo)

Para converter para um sistema cartesiano com origem no centro da imagem:

Uma imagem com texto, captura de ecrã, Tipo de letra, branco

Os conteúdos gerados por IA podem estar incorretos.

⚠️ A inversão de Y (120 - y) é feita porque na imagem da HuskyLens o valor **aumenta de cima para baixo**, o que é contrário ao referencial matemático habitual.

**🧠 Exemplo de novo sistema de coordenadas**

| **Posição na imagem** | **Coordenadas Cartesiano adaptadas** |
| --- | --- |
| Canto superior esquerdo | (-160, +120) |
| Centro da imagem | (0, 0) |
| Canto inferior direito | (+160, -120) |

**🔧 Aplicação prática no servo**

Para controlar o servo com este novo referencial:

Uma imagem com texto, Tipo de letra, captura de ecrã, branco

Os conteúdos gerados por IA podem estar incorretos.

**🧪 Benefícios desta abordagem:**

* Pensamento coerente com o plano cartesiano (X/Y)
* Facilita mapeamentos com lógica de IA ou planeamento
* Simplifica integrações futuras com ROS e RViz

**📘 Descrição – Seguimento de Objeto com Servo (Eixo X)**

Este módulo implementa um sistema de **seguimento horizontal automático de objetos** com base na posição detetada pela câmara **HuskyLens** em modo *Object Tracking*. Um **servo motor ligado à base da câmara** gira para a esquerda ou direita, de modo a manter o objeto sempre centrado na imagem.

**⚙️ Componentes envolvidos:**

* Arduino UNO R4 WiFi
* HuskyLens (modo *Object Tracking*)
* Servo motor SG90 (ou similar)
* Fonte externa de 5V para alimentar o servo
* SoftwareSerial para comunicação UART com a HuskyLens

**🧭 Lógica de funcionamento:**

1. **Inicialização**:
   * O servo é ligado ao pino D10 e posicionado a **90° (posição central)** assim que o sistema arranca.
   * A HuskyLens é inicializada na porta UART virtual (D8, D9) com taxa de 9600 bps.
2. **Referencial cartesiano centrado**:
   * A imagem da HuskyLens (320x240 px) é convertida para um **sistema cartesiano com origem no centro**:
     + X: -160 (esquerda) → 0 (centro) → +160 (direita)
     + Y: -120 (baixo) → 0 (centro) → +120 (cima)
   * Esta abordagem permite raciocínio espacial e futura integração com planeamento em ROS.
3. **Processamento contínuo**:
   * Em cada iteração, o Arduino:
     + Solicita uma leitura à HuskyLens.
     + Se um objeto estiver a ser seguido:
       - Calcula a coordenada X centrada.
       - Converte esse valor para um ângulo de servo entre 0° e 180°, usando a função map() e constrain().
       - Move o servo horizontal para alinhar a câmara com o objeto.
     + Se **não detetar nenhum objeto**, o servo **mantém a posição anterior** (sem movimentos bruscos nem recentragem automática).

**📐 Mapeamento do movimento:**

| **Coordenada X (cartesiana)** | **Posição na imagem** | **Ângulo do servo** |
| --- | --- | --- |
| -160 | Extrema esquerda | 0° |
| 0 | Centro | 90° |
| +160 | Extrema direita | 180° |

**🧠 Preparação para o futuro:**

* O código já está **estruturado para adicionar um segundo servo motor no eixo Y**, possibilitando seguimento vertical.
* Basta declarar um segundo servo (servoY) e ativar o mapeamento correspondente para o eixo Y, com base em yCartesiano.

**✅ Resumo técnico:**

* Uso da biblioteca Servo.h para controlo PWM.
* Porta UART virtual criada com SoftwareSerial.
* Comunicação a 9600 bps com a HuskyLens.
* Conversão matemática para alinhar referencial da imagem com servo.
* Simples, robusto e altamente pedagógico.