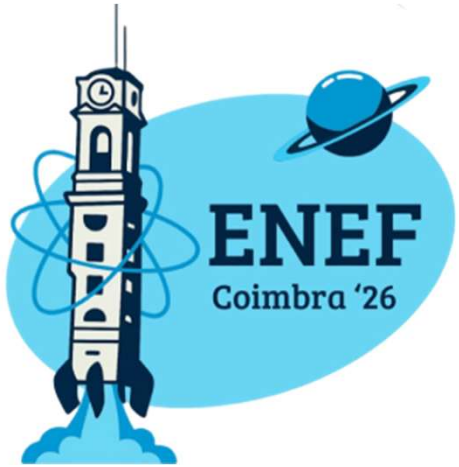


Workshop de Robótica

Introdução à Navegação Autónoma e Planeamento — do Wall-Following ao A*

Luís Garrote, PhD
Eduardo Borges, MSc

Human-Centered Mobile Robotics Lab
Institute of Systems and Robotics – University of Coimbra

INSTITUTO DE SISTEMAS E ROBÓTICA
UNIVERSIDADE DE COIMBRA

1 2 9 0
FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE D COIMBRA

CONTEXTO

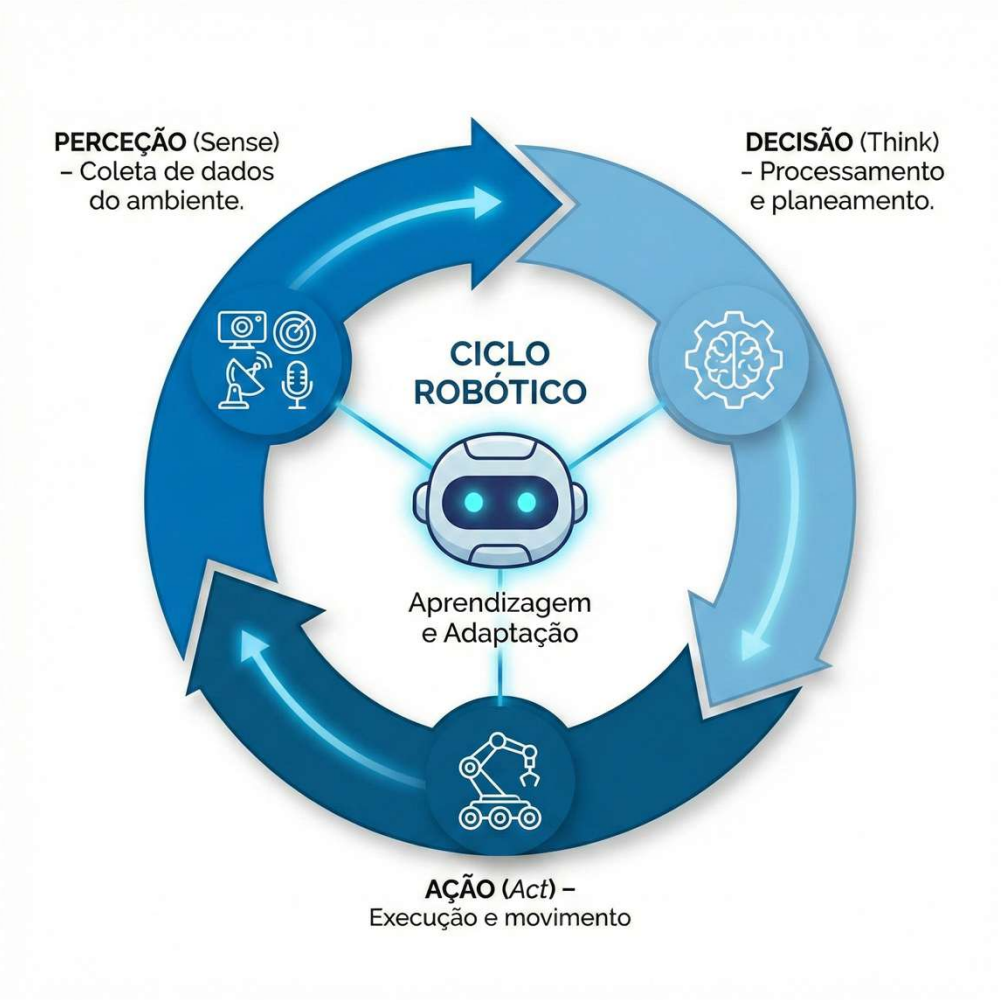
O que é um robô?

ISO 8373:2012 – Mecanismo acionado programável em dois ou mais eixos com um grau de autonomia, movendo-se dentro do seu ambiente, de forma a realizar tarefas pretendidas.

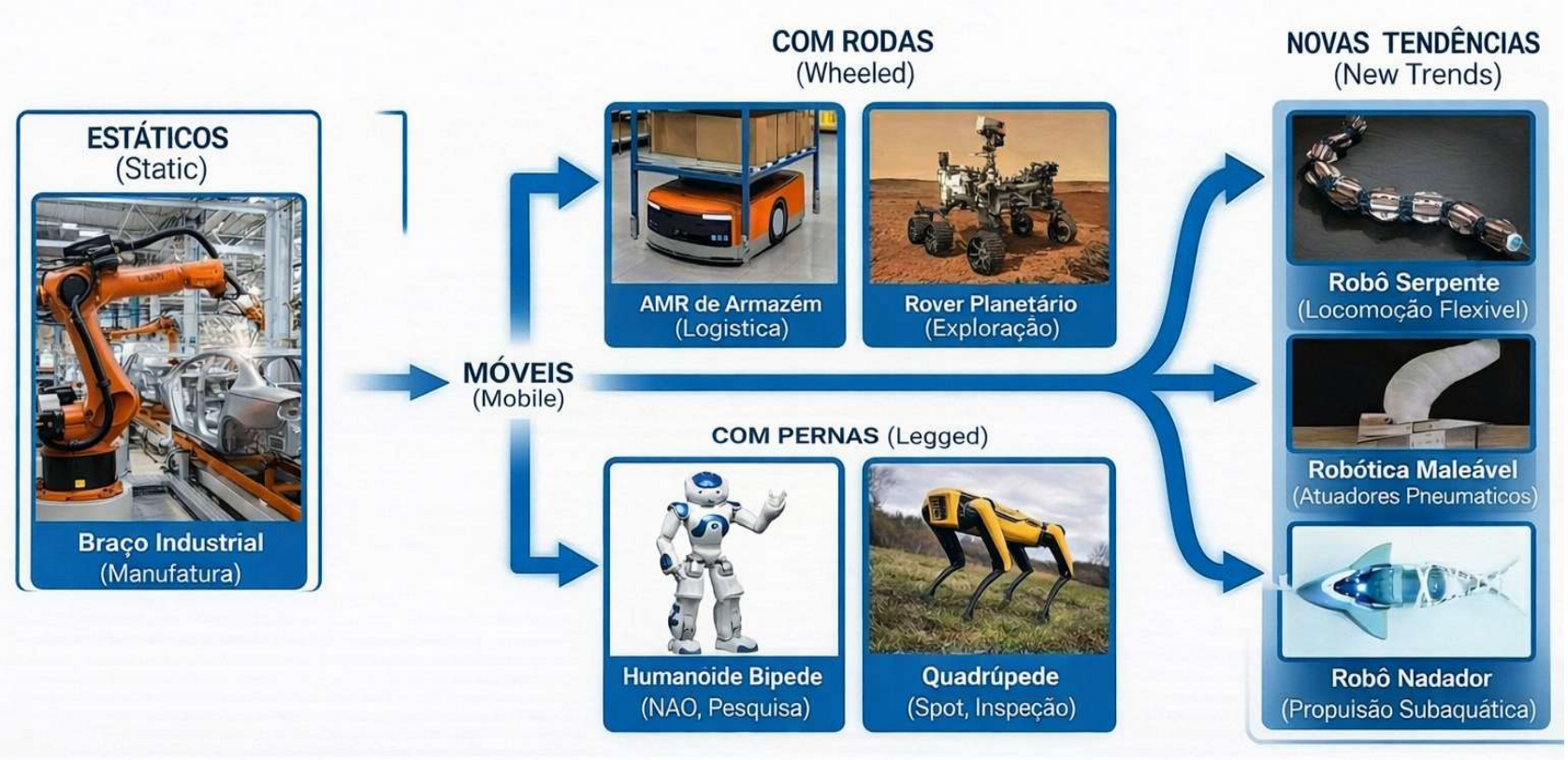
ISO 8373:2021 – Mecanismo acionado programado com um grau de autonomia para realizar locomoção, manipulação ou posicionamento.

Composto por:

- **Sensores:** Transdutores de grandezas físicas para digitais (LIDAR, Câmaras, Encoders).
- **Atuadores:** Motores DC, Servos (conversão de energia elétrica em mecânica).
- **Controlador:** O "cérebro" que processa a informação.



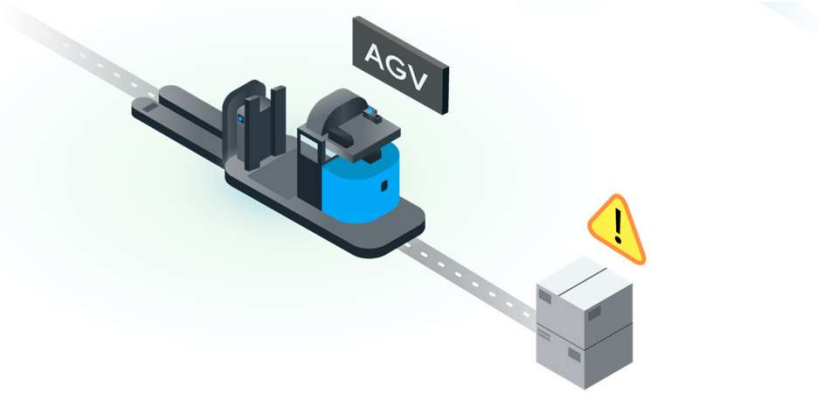
Quais os tipos de robôs mais comuns?



Robótica móvel – AGV vs AMR

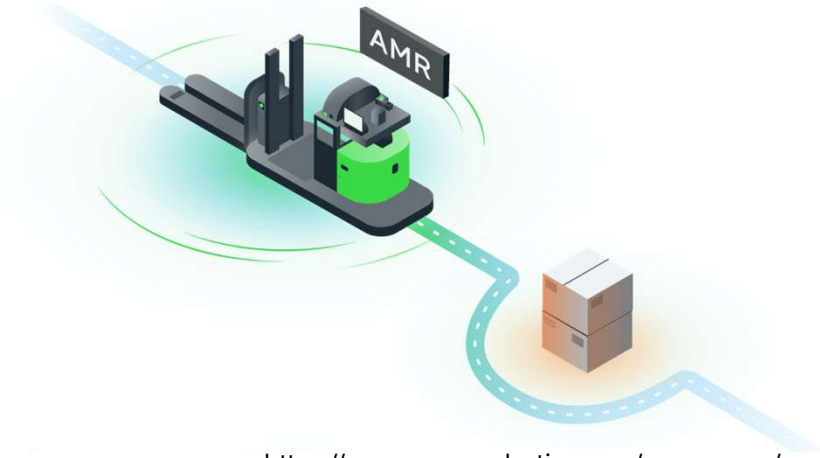
Automated Guided Vehicle (AGV)

- Seguem um caminho pré-determinado (fitas magnéticas ou fios).
- Inflexíveis – difíceis de ajustar.
- Custo elevado de instalação.



Autonomous Mobile Robot (AMR)

- Operam de forma independente e flexível.
- Capazes de navegar em ambientes dinâmicos.
- Navegam e exploram o ambiente sem dependerem de um caminho pré-definido.

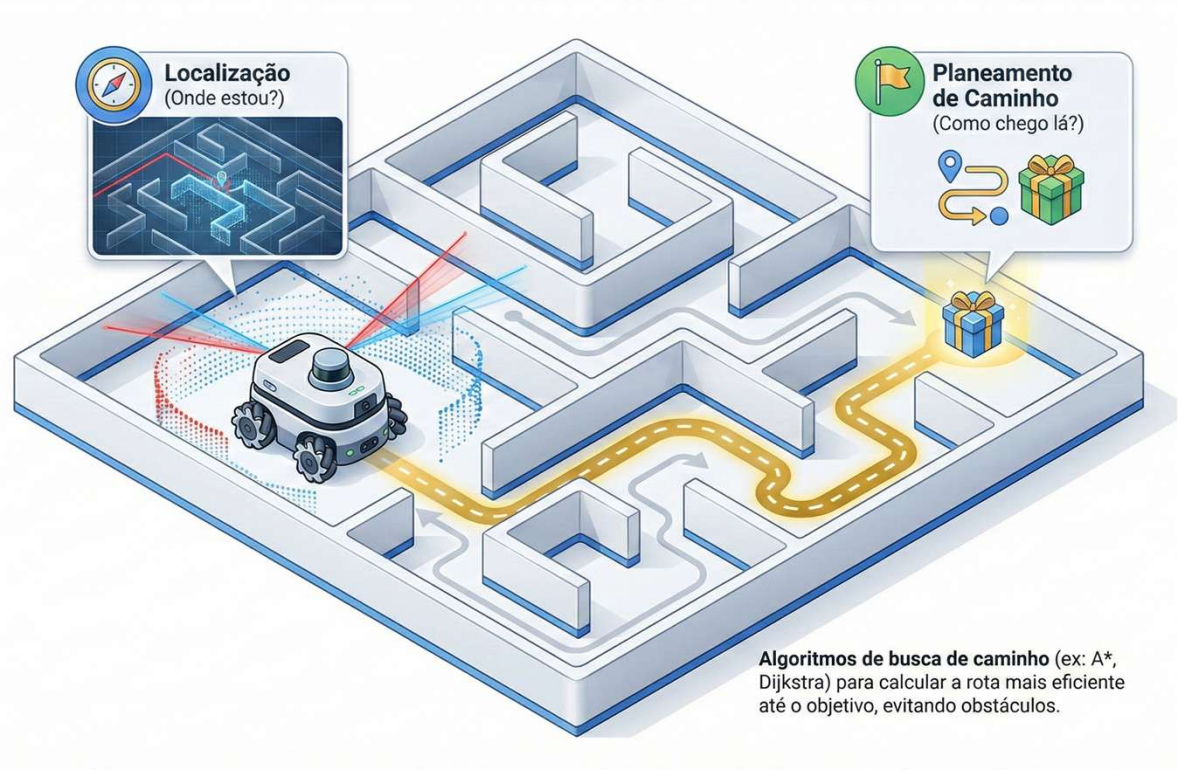


https://www.vecnarobotics.com/amr-vs-agv/

Robótica móvel – Navegação

As 3 Perguntas da Navegação:

1. Onde estou? (Localização/SLAM).
2. Onde quero ir? (Objetivo).
3. Como chego lá? (Planeamento de Caminho e Controlo de Movimento).



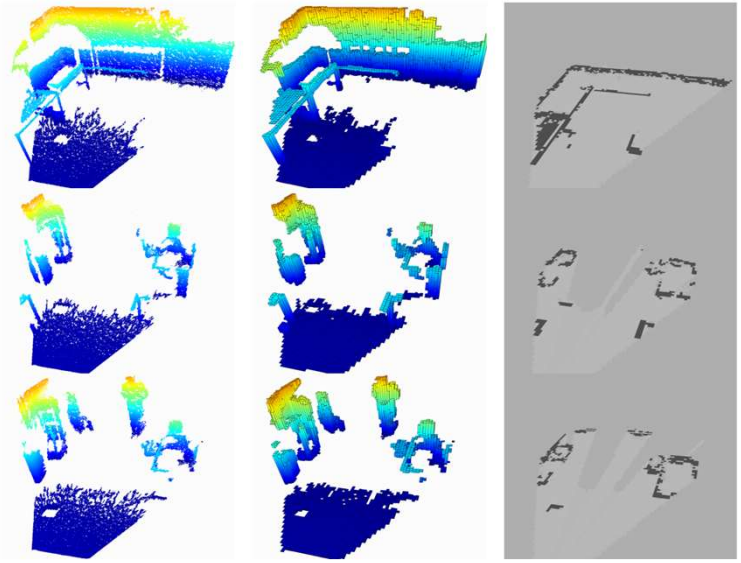
Perceção do Ambiente

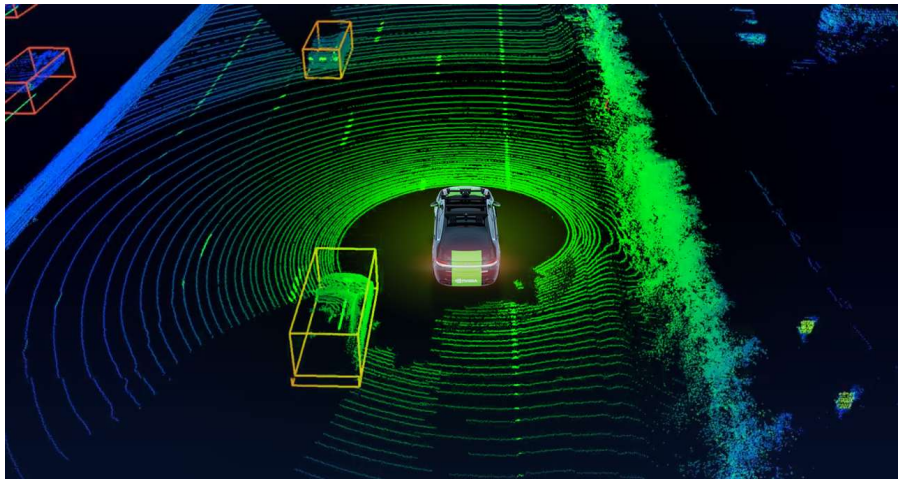
Sensores

- RGB-D Cameras
- LiDAR
- Odometria + IMU
- RADAR

Representaçõ

- Point Cloud
- 2D/2.5D Grid
- 3D Grid
- Graph
- OctoMaps

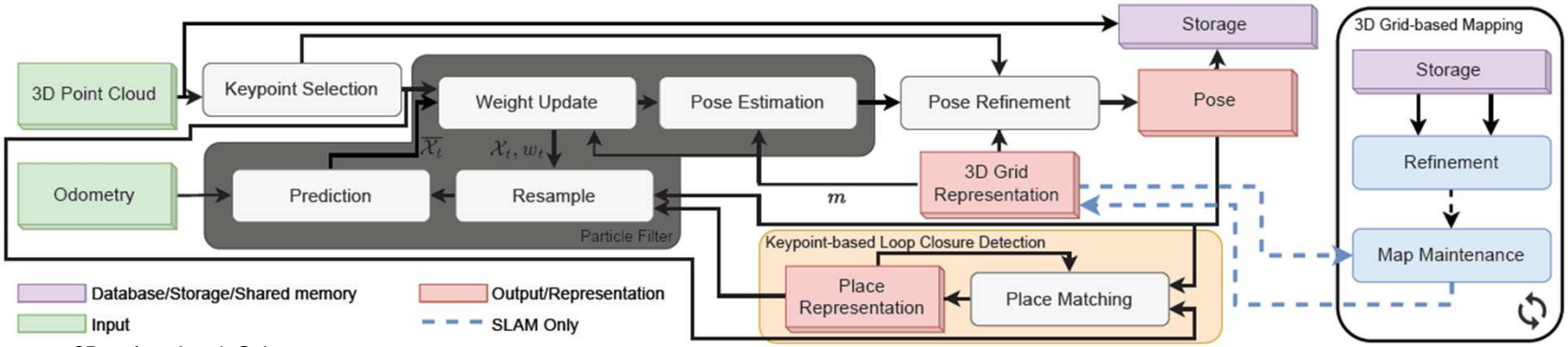




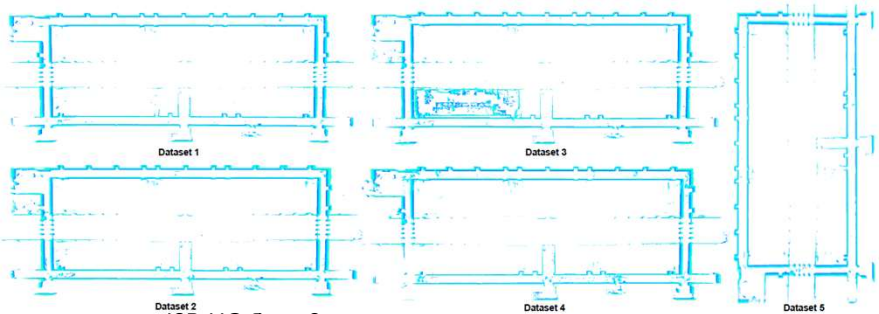
https://blogs.nvidia.com/blog/lidar-sensor-nvidia-drive/

Investigação em Curso (SLAM)

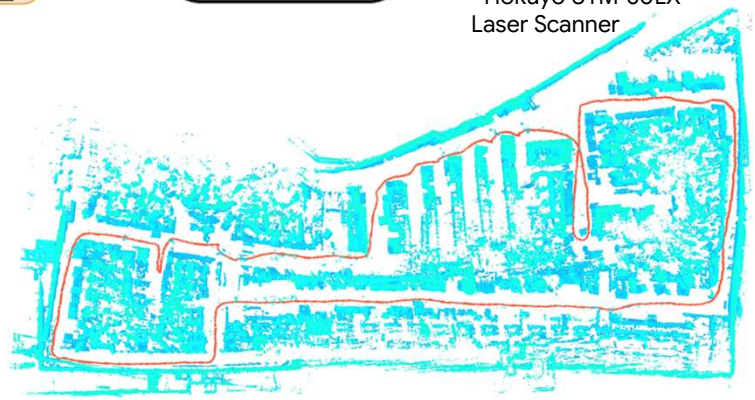
3D grid-based framework designed to function seamlessly as SLAM or as a robust standalone localization system, with refinement and loop closure detection stages.



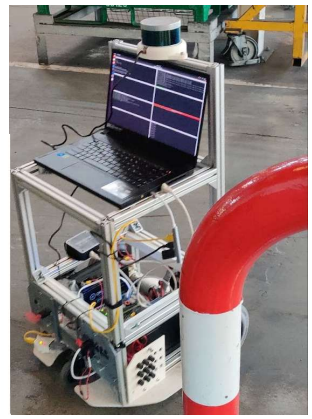
Inputs: 3D point cloud; Odometry
Main modules: AMCL-like particle filter Pose Estimation; Pose Refinement Stage; Loop Closure Detection



ISR-UC floor 0



Preliminary results - PSA Stellantis Mangualde

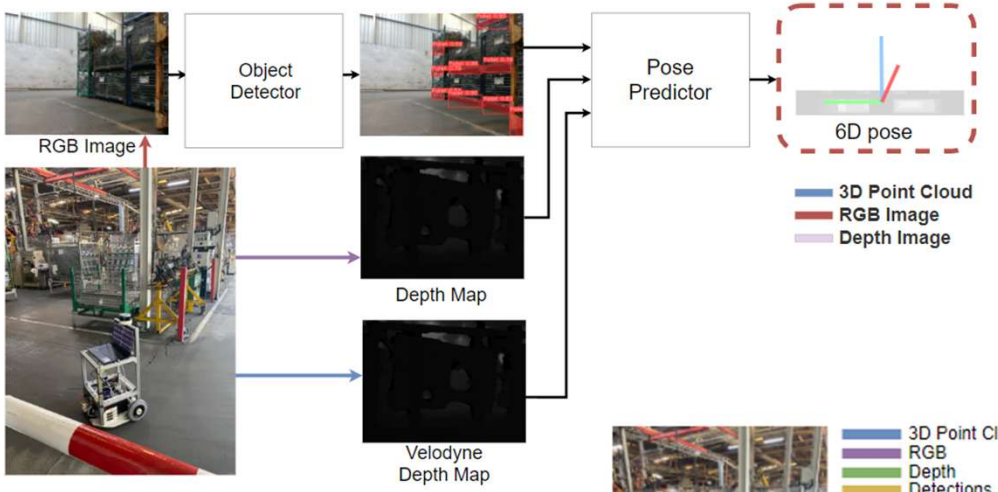


InterBot Equipped with:
- Velodyne VLP16
- Intel Realsense D435
- Hokuyo UTM-30LX Laser Scanner

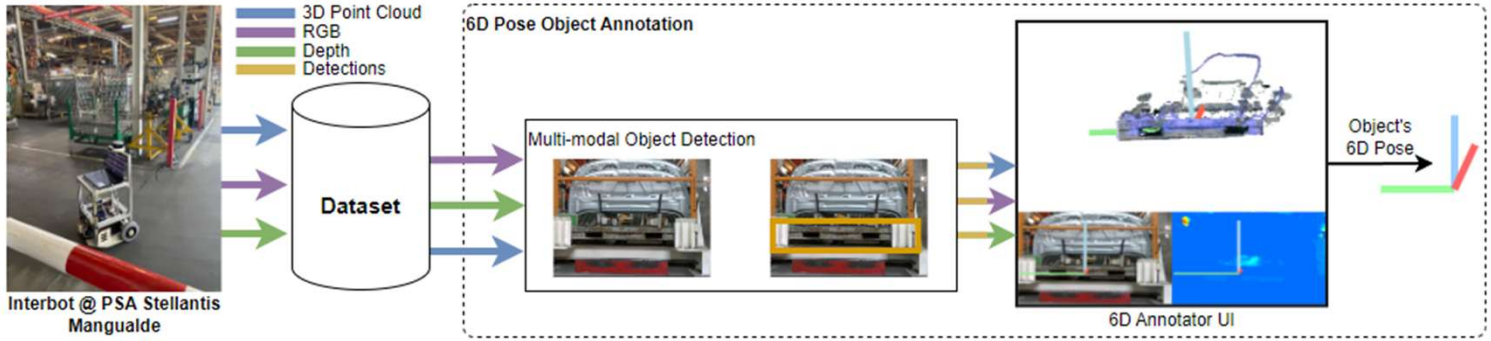
[1] R. Cruz, L. Garrote, A. Lopes and U. J. Nunes, "Modular software architecture for human-robot interaction applied to the InterBot mobile robot," *2018 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)*, Torres Vedras, Portugal, 2018
[2] L. Garrote, U. Reverendo and U. J. Nunes, "Exploiting 3D Grids for Indoor SLAM in Featureless Scenarios," 2024 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC), Paredes de Coura, Portugal, 2024

Investigação em Curso (Percepção)

Detection of objects of interest in industrial environments and pose estimation, with applications in box and pallet picking.



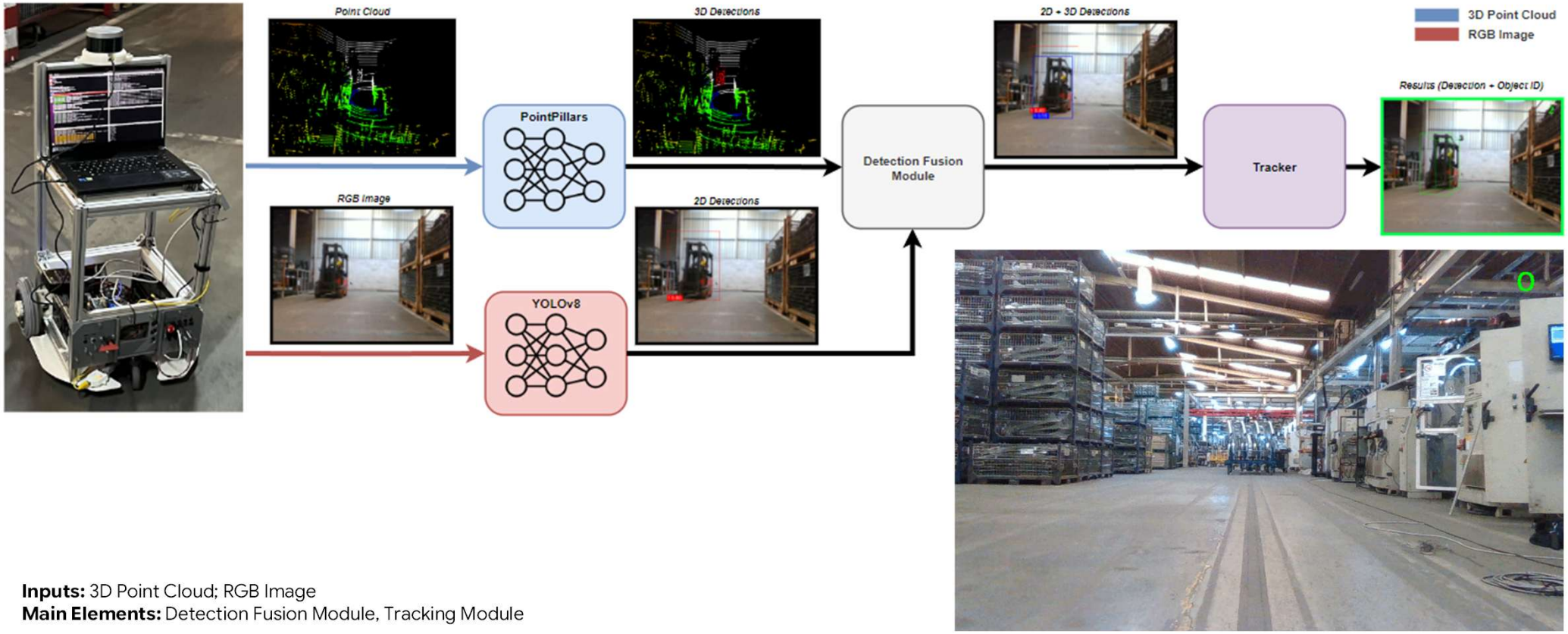
Datasets:



Inputs: 3D Point Cloud; RGB-D Image
Main Elements: Object Detector; Pose Predictor

Investigação em Curso (Percepção - Tracking)

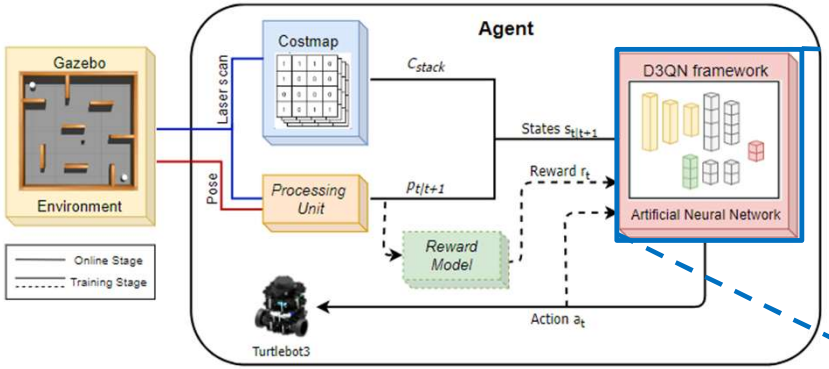
A deep learning-based system for 3D object detection and tracking, designed to optimize the movement and prevent collisions of AMRs in challenging industrial environments.

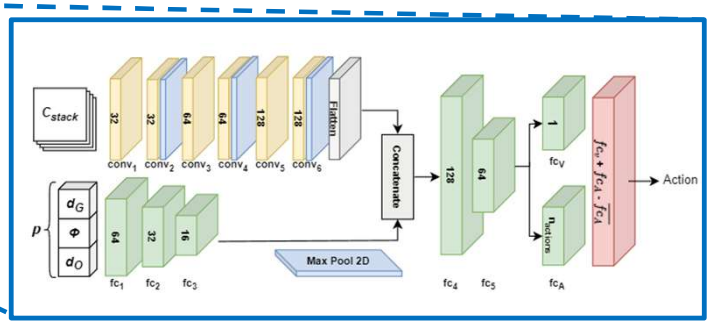


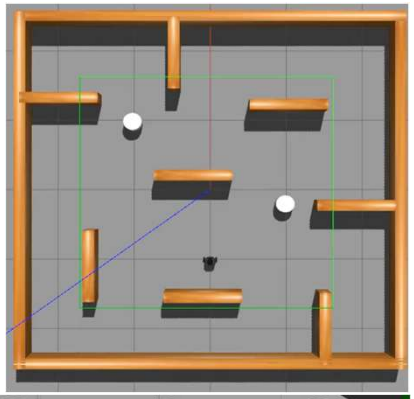
Inputs: 3D Point Cloud; RGB Image
Main Elements: Detection Fusion Module, Tracking Module

Investigação em Curso (Navegação)

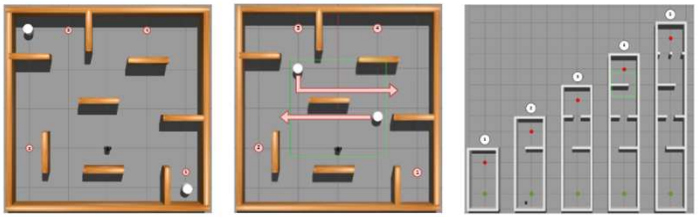
Deep Reinforcement Learning approaches to solve robot motion planning in environments populated by both static and dynamic obstacles by exploiting DeepRL frameworks.



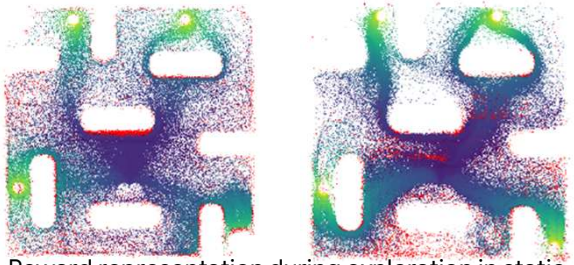




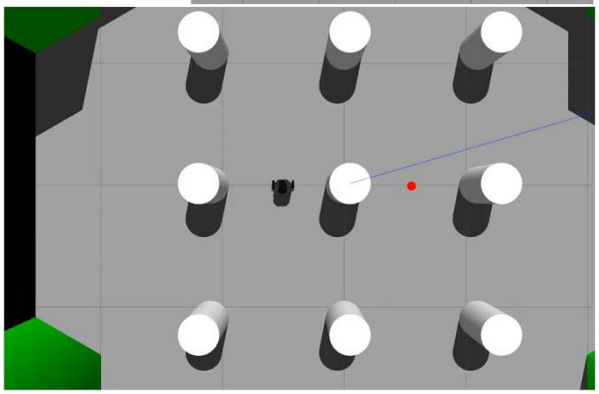
Inputs: Odometry; Point cloud 2D; 2D Goal
Main Elements: Costmap-based state representation; Prioritized Experience Replay; Reward Propagation and Curriculum/Transfer Learning.



Training: Static vs Dynamic vs Curriculum learning scenarios



Reward representation during exploration in static and dynamic scenarios

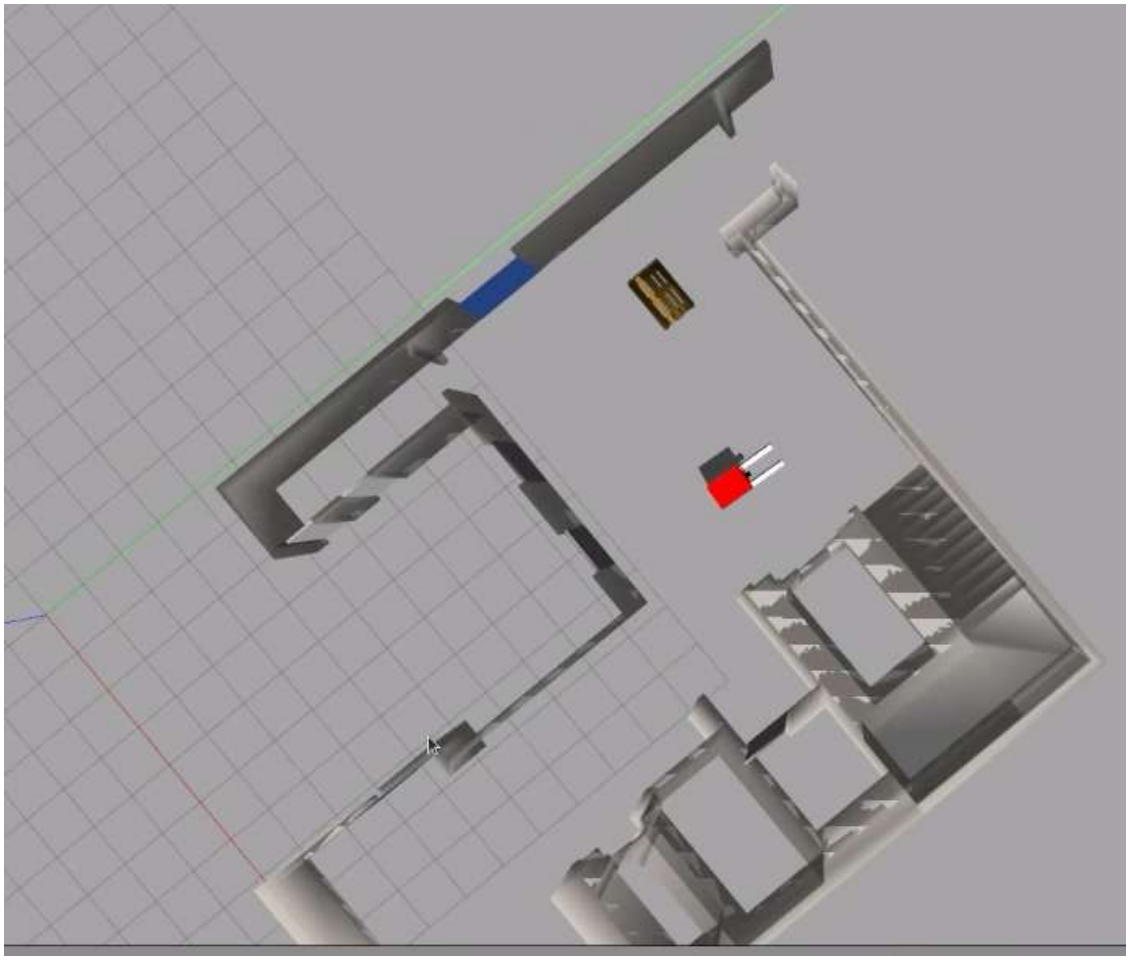


[1] Gonçalves, G., Palaio, D., Garrote, L., Nunes, U.J., "DeepRL-Based Robot Local Motion Planning in Unknown Dynamic Indoor Environments", Robot 2023: Sixth Iberian Robotics Conference. ROBOT 2023.
[2] L. Garrote, J. Perdiz and U. J. Nunes, "Costmap-based Local Motion Planning using Deep Reinforcement Learning," 2023 32nd IEEE International Conference on Robot and Human Interactive Communication (RO-MAN), Busan, Korea, Republic of, 2023

Investigação em Curso (Navegação)

Neste contexto, o planeamento local precisa de objetivos. Para obter modelos com maior generalização, podem ser utilizados os Large Language Models (LLMs) e os Vision-Language Models (VLMs) para guiar a exploração do ambiente.

O objetivo é usar as suas capacidades de decisão e perceção para orientar a seleção de metas, promovendo uma cobertura mais ampla do ambiente e evitando que o robot volte a visitar de forma desnecessárias às mesmas áreas.

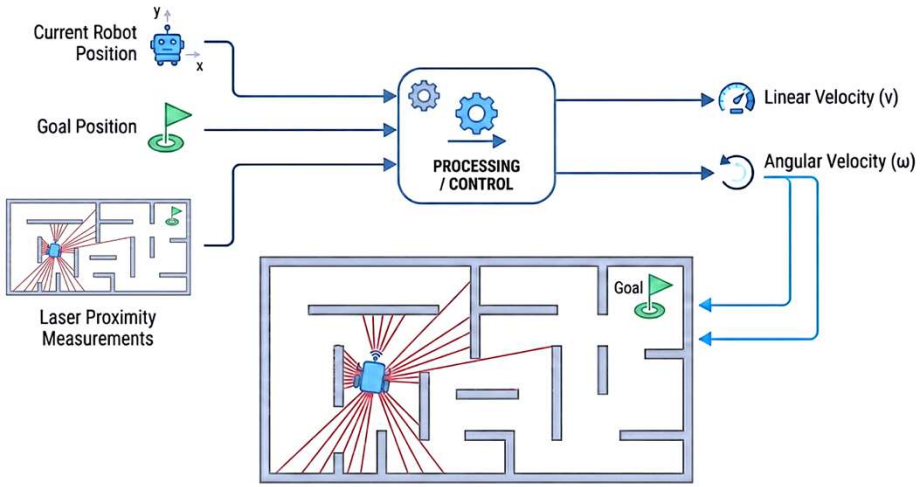


ALGORITMOS E LÓGICA

Exploração

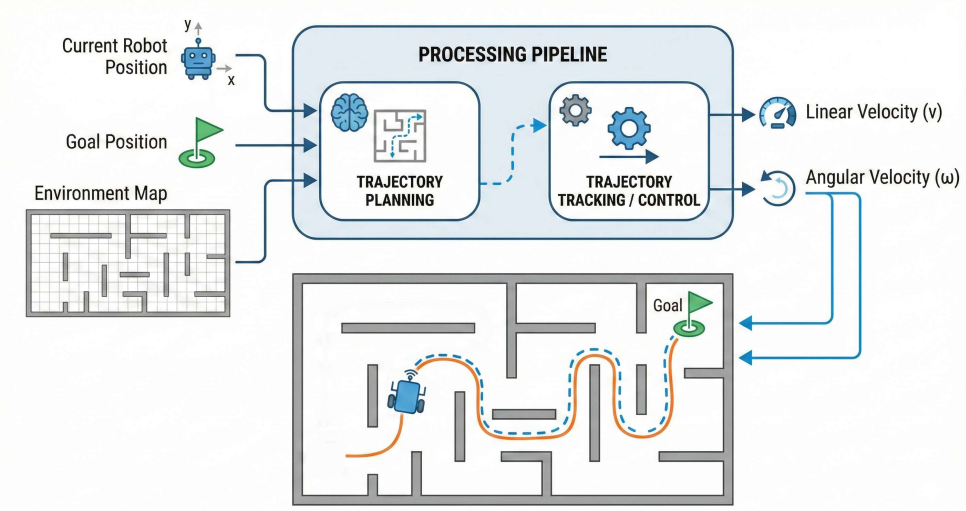
Sem o conhecimento do mapa:

- O robô tem que chegar ao objetivo seguindo paredes, e tomar decisões em zonas de incerteza. Pode ao longo dessa exploração criar a representação do ambiente e garantir que não revisita as mesmas regiões múltiplas vezes.
- Podem ser usados algoritmos de *coverage* para planear a exploração.



Com o conhecimento do mapa:

- Podem ser usados planeadores para criar um plano global e depois usar técnicas de seguimento de trajetórias (Path Following) para garantir que a missão do robot é executada com sucesso.



Algoritmos – Wall Following

Conceito:

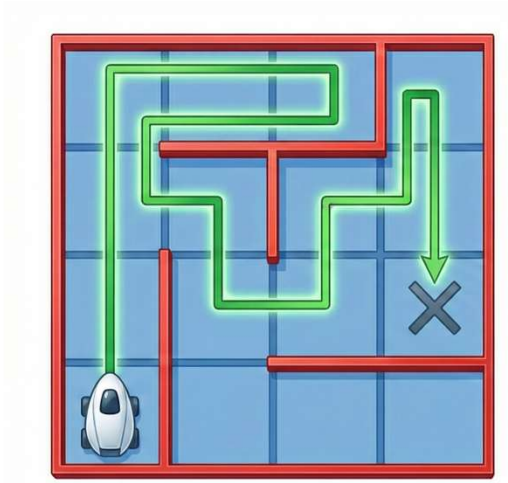
- Comportamento baseado em regras simples (Se/Então). Não há "memória" nem mapa global.

Exemplo (Mão Esquerda):

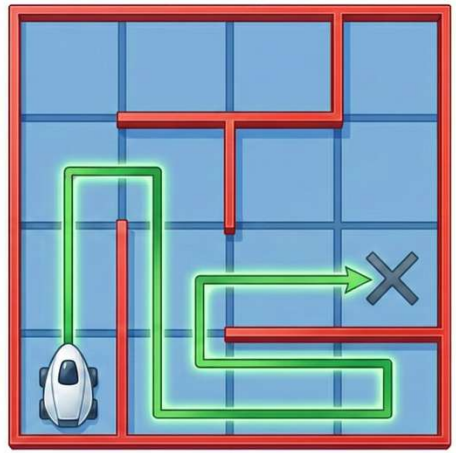
- Se (frente livre) E (esquerda perto da parede) -> Avança.
- Se (frente bloqueada) -> Roda à direita.
- Se (esquerda longe da parede) -> Aproxima-se da esquerda.

Limitação:

- Loops infinitos, mínimos locais (o robô pode ficar preso num *loop*).



(a) Left-hand rule.



(b) Right-hand rule.

Algoritmos – Bug2

Conceito:

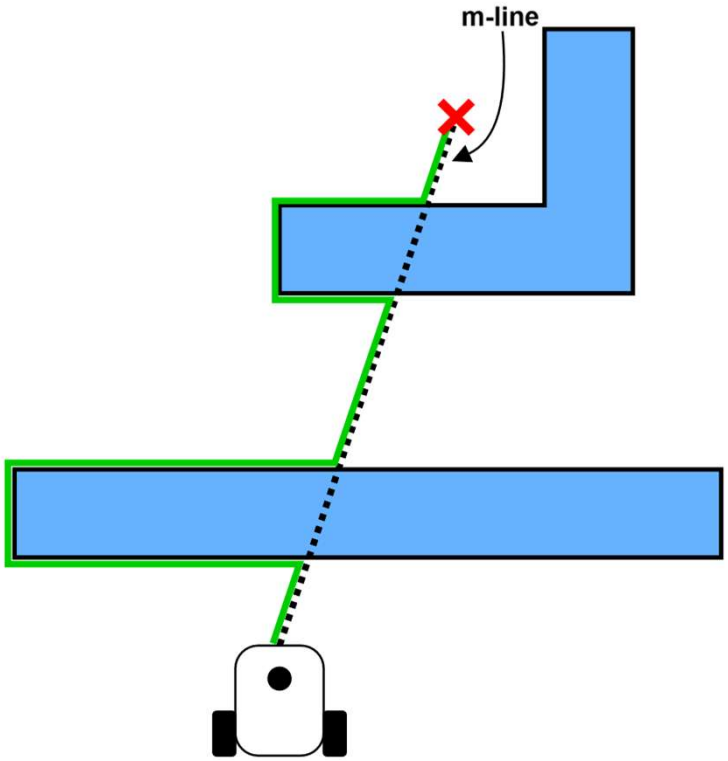
- Um algoritmo de navegação híbrido: combina comportamento reativo (contornar obstáculos) com um objetivo global fixo.
- Assume que o robô conhece a sua posição e a coordenada do objetivo, mas não conhece o mapa dos obstáculos.

A Linha M (M-Line):

- Uma linha reta imaginária traçada diretamente do Ponto de Partida ao Objetivo.
- Funciona como um "trilho magnético": o robô tenta manter-se sempre sobre esta linha.

O Comportamento (Regras de Estado):

1. Avançar pela M-Line até encontrar um obstáculo.
2. Contornar a parede do obstáculo (esquerda ou direita).
3. Ao cruzar novamente a M-Line, continuar em direção ao objectivo.



Planeamento Global

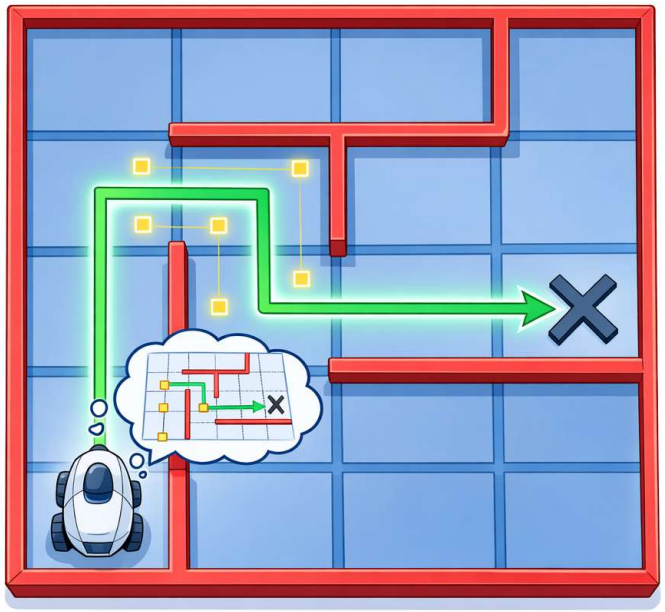
Considerando que o robô conhece o mapa, podemos tentar uma abordagem que não requer a exploração do ambiente.

Este passo assume que o robô tem um mapa obtido por SLAM e que de alguma forma existe um caminho sem colisões que liga a posição inicial e final do robô.

Este tipo de navegação (comum no Robot Operating System - ROS) requer um planeador global e um planeador local.

Existem vários tipos e famílias de planeadores mas normalmente são usados:

- Planeadores baseados em graph/grid (A*,D*, ...)
- Planeadores baseados em amostragem (RRT,RRT*, PRM,...)



Algoritmos – A*

Conceito:

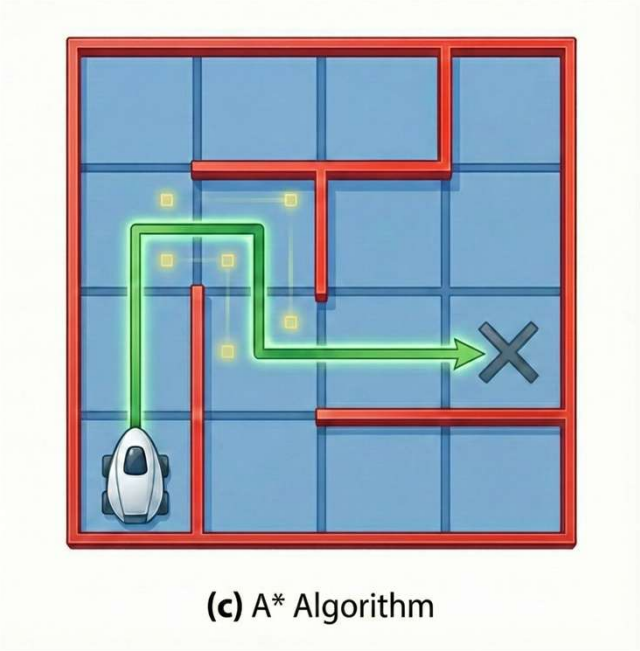
- O algoritmo de busca mais famoso para *pathfinding*.
- Ao contrário de algoritmos "cegos" que exploram uniformemente em todas as direções, o A* usa uma "bússola" para focar a exploração na direção do objetivo.

A Equação: $f(n) = g(n) + h(n)$

- $g(n)$: Custo real do início até ao nó atual (distância percorrida).
- $h(n)$: Heurística (estimativa do custo do nó atual até ao objetivo).

A Heurística:

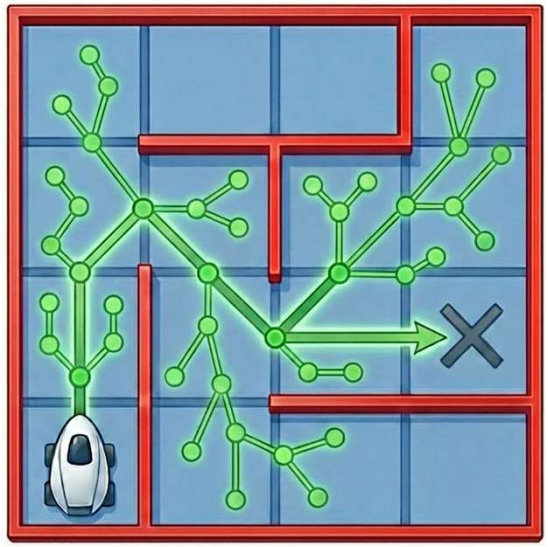
- Euclidiana: Distância em linha reta (hipotenusa).
- Manhattan: Soma dos catetos (ideal para grelhas).



(c) A* Algorithm

https://en.wikipedia.org/wiki/A*_search_algorithm

Algoritmos – RRT/RRT*



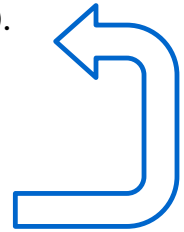
(e) RRT Algorithm

Conceito:

- Ao contrário do A*, que usa uma heurística na direção do objetivo, o RRT (Rapidly-exploring Random Tree) explora o espaço de forma aleatória e incremental, construindo uma árvore que rapidamente cobre o ambiente.
- O algoritmo seleciona pontos aleatórios no espaço e expande a árvore em direção a esses pontos, permitindo encontrar caminhos mesmo em espaços contínuos e com obstáculos complexos.
- O RRT* é uma versão otimizada que melhora continuamente os caminhos encontrados, aproximando-se da solução ótima.

Como Funciona:

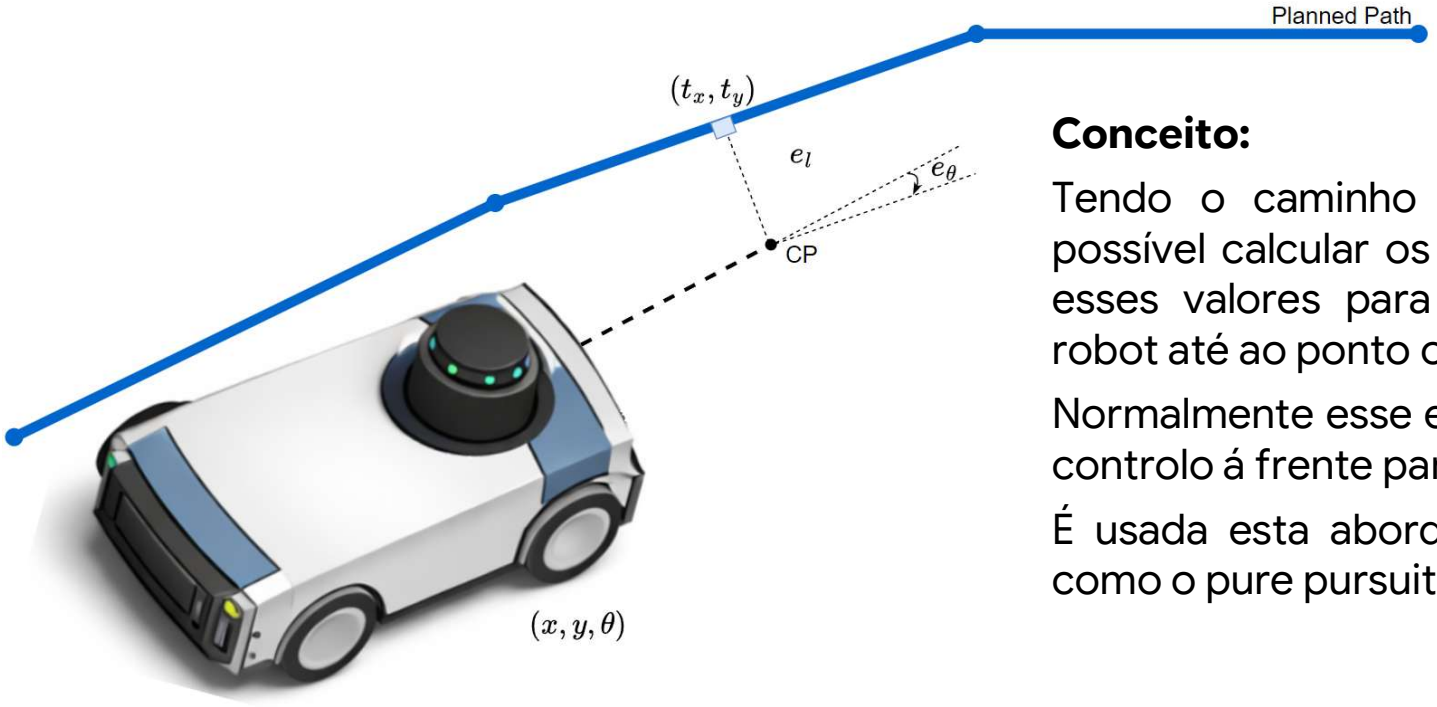
- Começa no nó inicial (posição inicial do robot).
- Amostra um ponto aleatório no espaço.
- Expande a árvore em direção a esse ponto.
- Evita colisões com obstáculos.
- Repete até alcançar o objetivo.



Ciclo até encontrar uma solução válida ou máximo número de nós

https://en.wikipedia.org/wiki/Rapidly_exploring_random_tree

Seguimento de Caminho



Conceito:

Tendo o caminho planeado pelo planeador, é possível calcular os erros lateral e angular e usar esses valores para guiar o comportamento do robot até ao ponto objetivo.

Normalmente esse erro é calculado num ponto de controlo á frente para garantir estabilidade.

É usada esta abordagem em métodos clássicos como o pure pursuit, Kanayama, entre outros.

Caso seja complicado o calculo dos erros, pode-se usar uma abordagem mais direta, de velocidade linear constante e velocidade angular dada pelo erro angular entre a posição do robot e o objetivo no caminho:

$$\theta_d = \text{atan2}(t_y - y,\ t_x - x) \qquad e_\theta = \text{wrap}(\theta_d - \theta) \qquad \omega = k\, e_\theta$$

A velocidade linear pode depois ser reduzida ao aproximar do ponto objetivo.

IMPLEMENTAÇÃO

Setup

Considerando um robot diferencial onde o comando de controlo é dado por (v,w) temos que calcular as velodidades da roda esquerda e direita como:

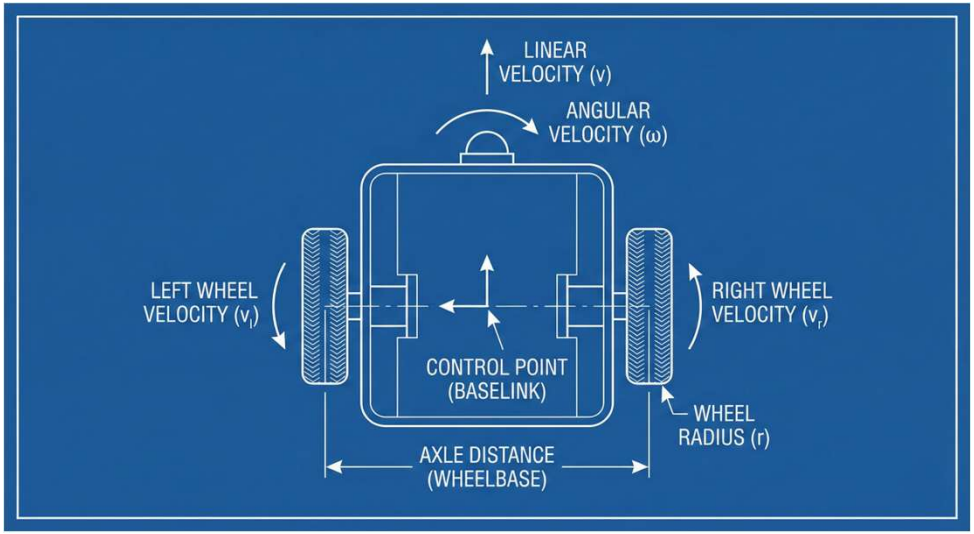
$$v_l = v - \frac{L}{2}\omega, \quad v_r = v + \frac{L}{2}\omega$$

L é a distancia entre eixos.

A cinematica direta do robot diferencial é dada por (update discreto):

$$\begin{bmatrix} x_{k+1} \\ y_{k+1} \\ \theta_{k+1} \end{bmatrix} = \begin{bmatrix} x_k \\ y_k \\ \theta_k \end{bmatrix} + \Delta t \begin{bmatrix} \frac{v_r^* + v_l^*}{2} \cos\theta_k \\ \frac{v_r^* + v_l^*}{2} \sin\theta_k \\ \frac{v_r^* - v_l^*}{L} \end{bmatrix}$$

Neste contexto * representa as estimativas do movimento.



Instruções do Google Colab

Componente pratica deste workshop vai ser feita no Google Colab.

Codigo está disponivel em:

https://github.com/luisgarrote/Minimouse

```
!git clone https://github.com/luisgarrote/Minimouse.git
%cd Minimouse
!pip install -e .
```

```
import minimouse
from google.colab import output
output.enable_custom_widget_manager()
disp=minimouse.Display()
```

```
disp.show()
```

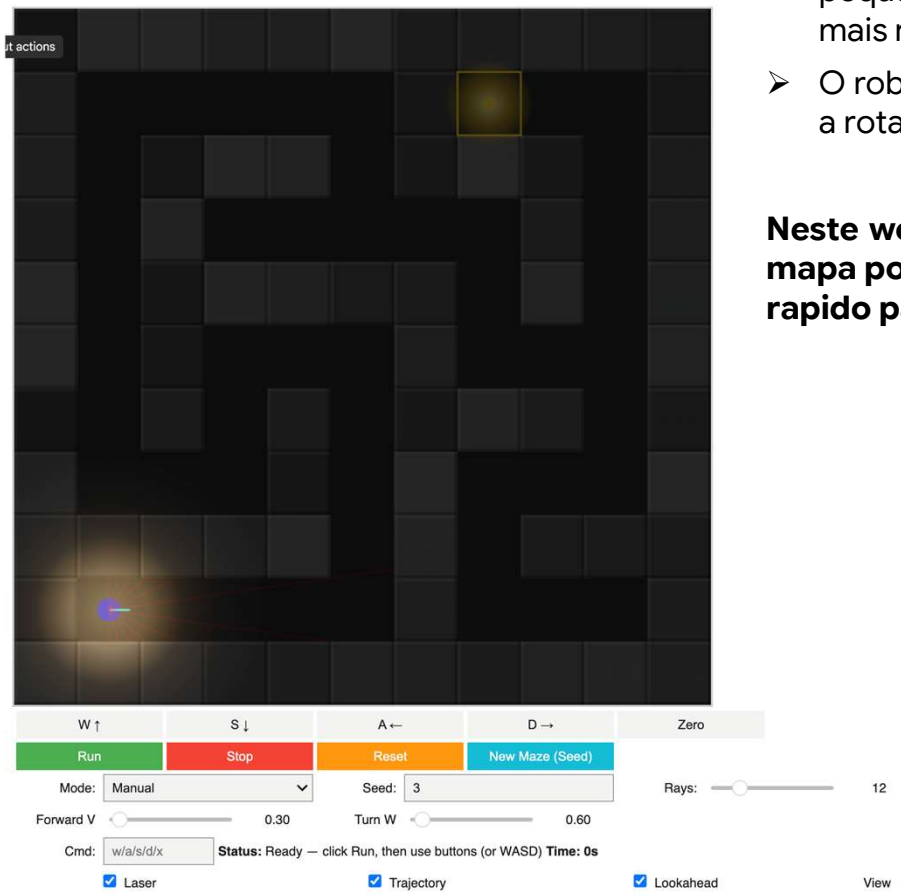
Google Colab não permite acesso ao input do utilizador e tem alguns problemas com simulação em tempo real.

É necessario ter scripts activos para que o javascript nas widgets/controlos funcione.

Testado no Google Chrome

Interface

Qual é o ambiente e porquê?

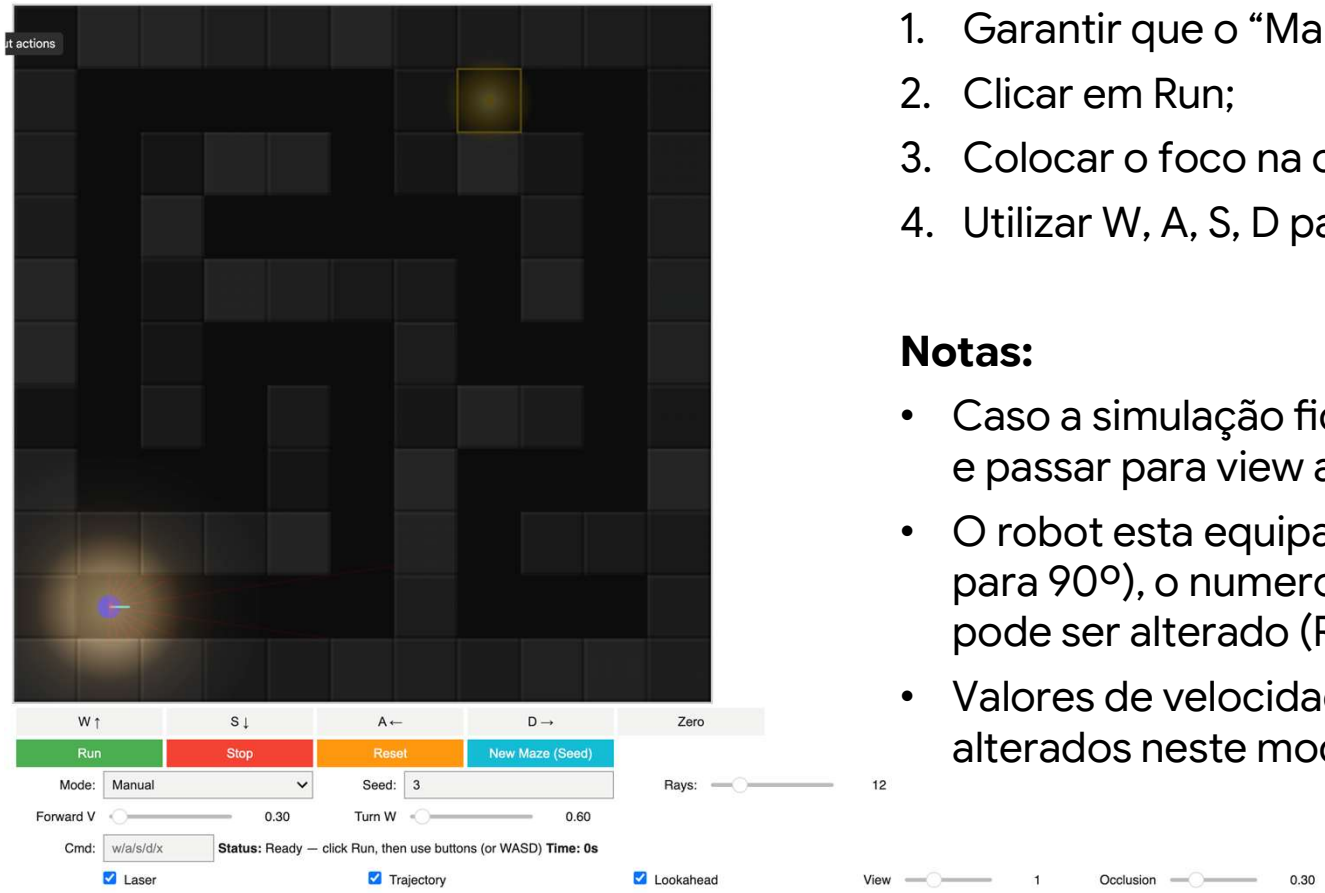


- O Micromouse é uma competição classica de robótica (existe desde 1977) em que pequenos robôs autónomos (“ratos”) têm de encontrar e percorrer o caminho mais rápido até ao centro de um labirinto.
- O robô começa sem conhecer o percurso e precisa de explorar, mapear e calcular a rota ótima.

Neste workshop simplificamos o problema, localização do robot é perfeita e o mapa pode ser acedido, o foco fica apenas em tentar encontrar o caminho mais rapido para o objectivo.

https://www.youtube.com/watch?v=urhMXrpEmv4

Interface



Passos a seguir para testar:

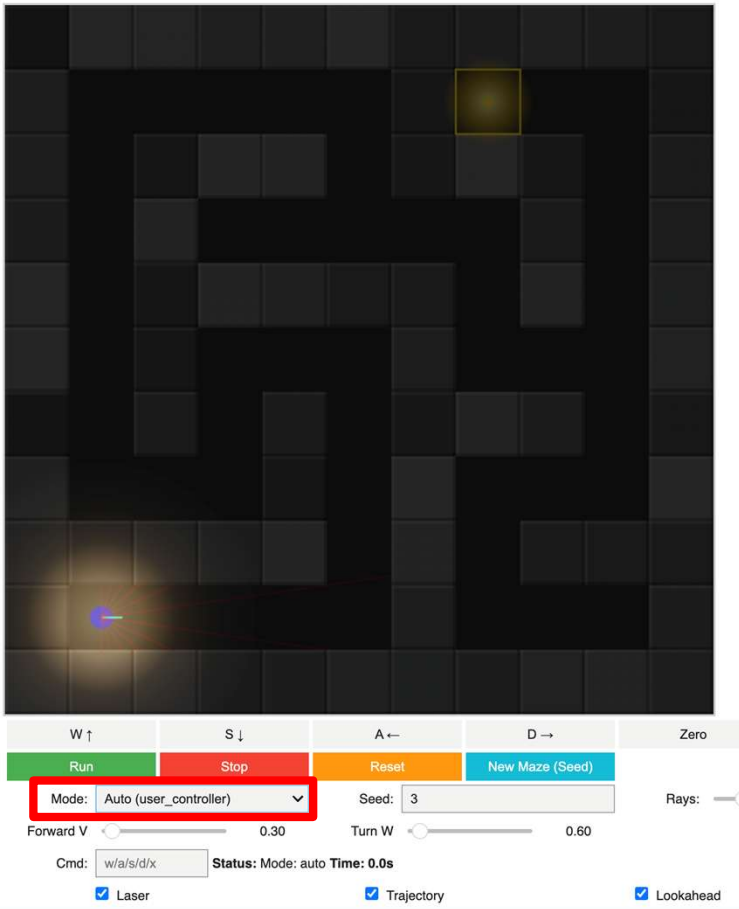
1. Garantir que o “Manual Mode” está ativo;
2. Clicar em Run;
3. Colocar o foco na caixa de texto Cmd;
4. Utilizar W, A, S, D para controlar o robot.

Notas:

- Caso a simulação fique lenta, fazer uncheck das flags e passar para view a 1 pode remover esse problema;
- O robot esta equipado com um laser 2D de 180º (-90+ para 90º), o numero de pontos gerados nesse espaço pode ser alterado (Rays).
- Valores de velocidade linear e angular podem ser alterados neste modo (Forward V e Turn W).

Interface

Como criar um controlador:




```python
def reset():
    """Utiilizar caso seja necessario limpar o estado de variaveis"""
    pass

def user_controller(obs):
    """
        Define o controlador do robot.
        Inputs: pose, laser, goal_cell.
    """
    x, y, th = obs["pose"]
    laser = obs["laser"]
    gx_cell, gy_cell = map(int, obs["goal_cell"])
    # Convert goal cell -> world coords (env provides this)
    gx, gy = disp.env._cell_center_world(gx_cell, gy_cell)
    pos = (float(x), float(y))
    goal = (float(gx), float(gy))
    # ------------------------------
    # Laser sectors (IMPORTANT: +angles = left, -angles = right)
    # ------------------------------
    n = len(laser)
    angles = np.linspace(-math.pi/2, math.pi/2, n)

    front_mask = np.abs(angles) < math.radians(10)
    left_mask  = angles >  math.radians(35)
    right_mask = angles < -math.radians(35)

    front = float(np.min(laser[front_mask])) if np.any(front_mask) else disp.env.max_range
    left  = float(np.min(laser[left_mask]))  if np.any(left_mask)  else disp.env.max_range
    right = float(np.min(laser[right_mask])) if np.any(right_mask) else disp.env.max_range
    # goal stop tolerance: env considers done at ~0.45*cell_size radius
    goal_stop = 0.45 * disp.env.cell_size

    d_goal = dist(pos, goal)
    if d_goal < goal_stop:   # chegou ao objectivo
        return 0.0, 0.0

    ################################################
    #       TODO.   Define (v, w)
    ################################################
    # Convert (v, w) -> (vl, vr)
    # ------------------------------
    L = disp.env.wheel_base
    vl = v - 0.5 * L * w
    vr = v + 0.5 * L * w
    # Clip wheel speeds
    max_wheel = 6.0 * disp.env.cell_size
    vl = float(np.clip(vl, -max_wheel, +max_wheel))
    vr = float(np.clip(vr, -max_wheel, +max_wheel))
    return vl, vr

disp.callbacks["user_controller"]=user_controller
disp.callbacks["reset_hook"]=reset
```

O mapa do ambiente esta disponivel na variavel **disp.env.occ**

```
[[1 1 1 1 1 1 1 1 1 1]
 [1 0 0 0 0 0 1 0 0 1]
 [1 1 1 1 1 0 1 0 1 1]
 [1 0 0 0 1 0 1 0 0 1]
 [1 0 1 0 1 0 1 1 1 1]
 [1 0 1 0 0 0 1 0 0 1]
 [1 0 1 1 1 1 1 0 1 1]
 [1 0 1 0 0 0 0 0 1 1]
 [1 0 1 1 1 0 1 1 1 1]
 [1 0 0 0 0 0 1 0 0 1]
 [1 1 1 1 1 1 1 1 1 1]]
```

Acesso a posição, leituras do laser e objectivo

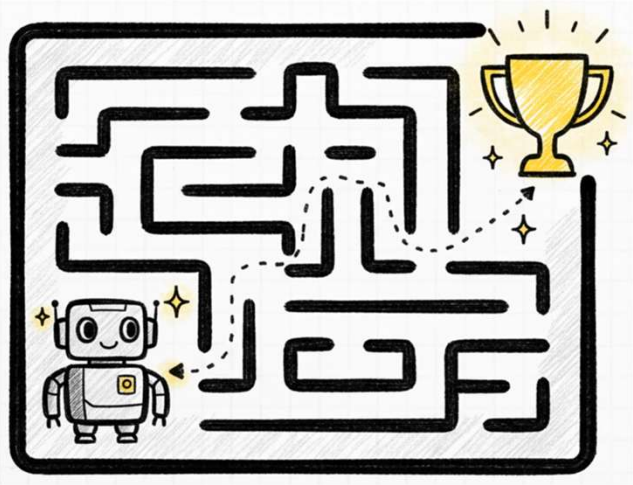
Mapeamento entre leituras e ângulos para obter menor leitura a esquerda, direita ou frente do robot.

Condição de paragem (faz stop ao contador).

Conversão entre (v,w) e (vl,vr), representa o comando a enviar para o cada uma das rodas

Registar as funções no dicionário de callbacks para que o simulador consiga encontrar as novas implementações

Desafio Prático



CHALLENGE: Chegar o mais rápido possível ao objectivo!

Regras:

1. Cada participante deve usar o ambiente do Google Colab (com a seed a 3);
2. Podem ser usadas outras bibliotecas ou ser pedida ajuda de LLMs (ChatGPT, Gemini, etc);
3. Apenas podem ser modificadas as funções user_*controller* e *reset*;
4. A métrica de avaliação será o tempo até chegar ao objetivo;
5. Primeiro critério de desempate será o desempenho em labirintos construídos para novas *seeds* (a indicar depois);
6. Segundo critério de desempate será o desempenho num ambiente mais complexo.