Neural A* Search path planning algorithm in Duckiebots for autonomous driving

Lidia Fargueta Pelufo
lidia.fargueta@fau.de
Chair of Multimedia Communications and Signal Processing

Friedrich-Alexander-Universität
Technische Fakultät

Index

- Software work
- Hardware work

Last work done

SOFTWARE understanding

- Studied how Neural A* repository Works.
- Study the database that they are using.
- Implementation: generate a Duckietown circuit and predict the path with Neural A*.

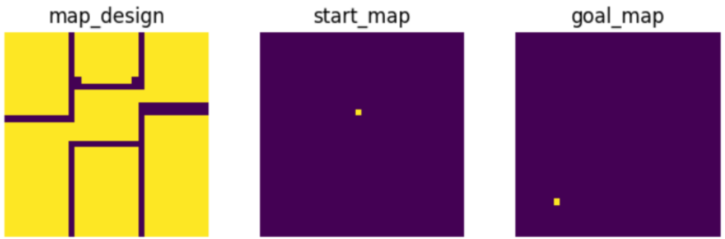
HARDWARE understanding

- Understand how the robot is moving, what kind of commands we should give it to move.
- Found a project that tracks the real-time position coordinates of the robot in the room

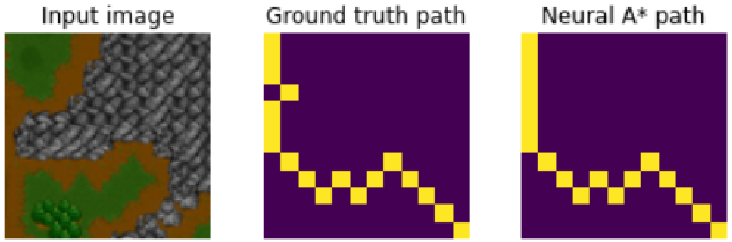
SOFTWARE

Neural A* repository

- Compare Neural A* with Vanilla A*
- Database (2 types):



Binary images



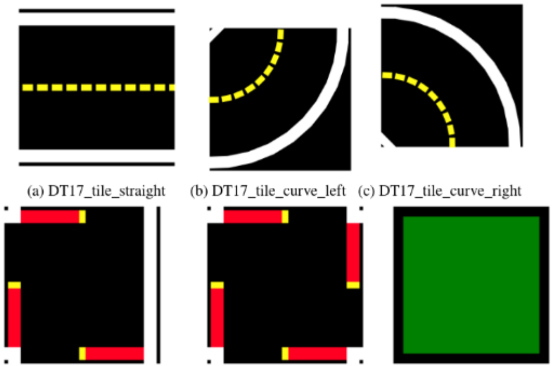
Raw image (Warcraft database)

How for Duckietown?

- Duckietown/map-utils:
  - Generator.py
    - output.yaml
    - adj-matrix.npz
  - Path-planning.py

Output.yaml

```yaml
tiles:
- [empty, empty,        empty,        empty,        empty,       empty,        empty,       empty,       empty]
- [empty, empty,        empty,        curve_right/N, straight/W, 3way_right/E, straight/W,  curve_left/N, empty]
- [empty, empty,        curve_right/N, curve_left/E, empty,      straight/N,   empty,       straight/N,  empty]
- [empty, curve_right/N, curve_left/E, empty,        empty,      3way_right/N, straight/W,  curve_left/E, empty]
- [empty, straight/N,   empty,        empty,        empty,       straight/N,   empty,       empty,       empty]
- [empty, curve_right/W, straight/W,  curve_left/N, empty,       straight/N,   empty,       empty,       empty]
- [empty, empty,        empty,        curve_right/W, straight/W, curve_left/E, empty,       empty,       empty]
- [empty, empty,        empty,        empty,        empty,       empty,        empty,       empty,       empty]
```



(a) DT17_tile_straight (b) DT17_tile_curve_left (c) DT17_tile_curve_right

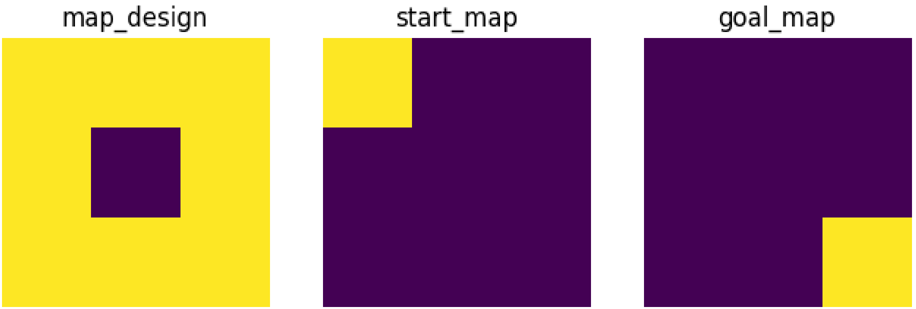
(d) DT17_tile_three_way_center (e) DT17_tile_four_way_center (f) DT17_tile_empty

```
0 0 2-2-3-2-2
    |   |   |
0 2-2 0 2 0 2
  |     |   |
2-2 0 0 3-2-2
|       |
2 0 0 0 2 0 0
|       |
2-2-2 0 2 0 0
    |   |
0 0 2-2-2 0 0
```

Terminal representation

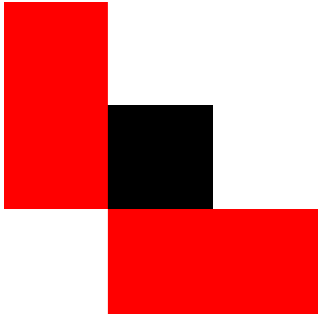
Idea...

Represent Duckietown environment as a binary image, as we have the adjacent matrix:

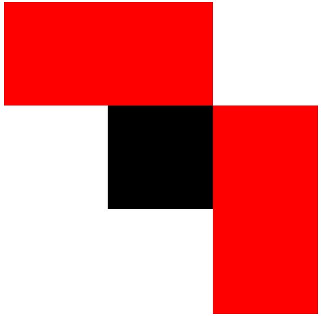


Idea…

Neural A*



Vanilla A*

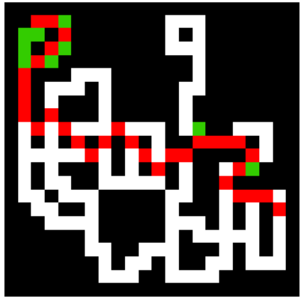


- Problems to solve:
  - Evaluation of neighbours (only 4)
  - Directional graph
- DATABASE IDEA FOR NEURAL A*:
  - We can create our training database with data-utils repository on loop or…
  - It is not necessary, it works good using the given trained weights
- Other idea: Work with raw images? More or less complex?

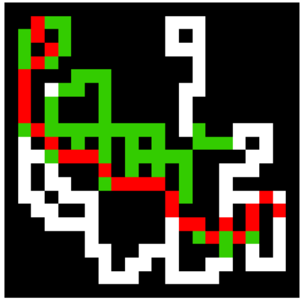
Idea…

- More complex roads

Neural A*

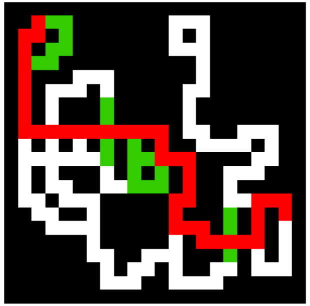


Vanilla A*

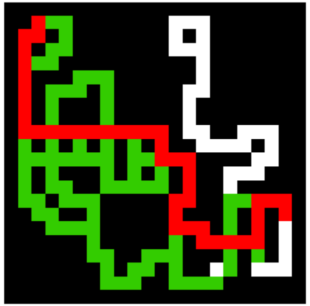


- Problem with the neighbours solved

Neural A*



Vanilla A*



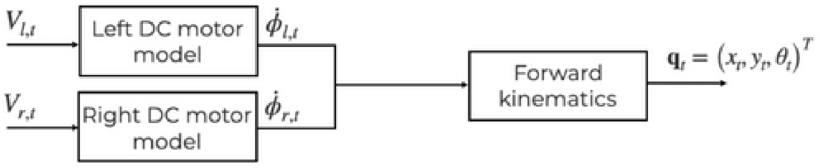
Last idea…

```
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 S 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 1 1 1 0
0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 X 0
0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 1 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

- There is a duckietown repository for running the implemented path-planning algorithm and generate a .yaml file representing the path. **Future work** is to research if there is a way of using this file in simulation for the robot to do the path.

HARDWARE

Odometry



$$\dot\phi_t = \left(R\frac{b}{K_i} + K_b\right)^{-1} V_t \qquad \dot{\mathbf{q}}_t = \frac{R}{2}\begin{bmatrix} cos\theta & 0 \\ sin\theta & 0 \\ 0 & 1 \end{bmatrix}\begin{bmatrix} 1 & 1 \\ \frac{1}{L} & -\frac{1}{L} \end{bmatrix}\begin{bmatrix} \dot\phi_r \\ \dot\phi_l \end{bmatrix}$$

$$\boxed{gain = R\frac{b}{K_i} \qquad trim = K_b}$$

Parameters for calibration of the wheels

- Odometry problem (determine position of the robot):

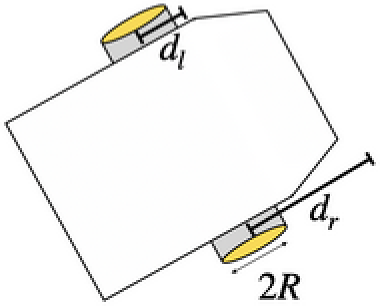
$$\begin{cases} x_{k+1} = x_k + \Delta x_k \\ y_{k+1} = y_k + \Delta y_k \\ \theta_{k+1} = \theta_k + \Delta \theta_k \end{cases}$$

1. Determine rotation of each wheel

   $\Delta\varphi_k = N_k * \alpha$, where $\alpha = \frac{2\pi}{N_{tot}}$

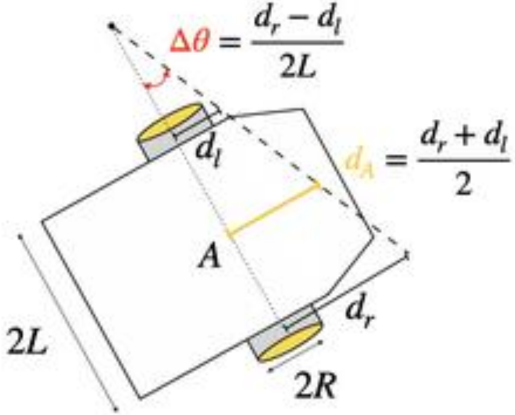
Odometry

2. Total distance tralled by each wheel:



$$d_{l\backslash r,k} = R * \Delta\varphi_{l\backslash r,k}$$

3. Rotation and distance travelled by the robot:

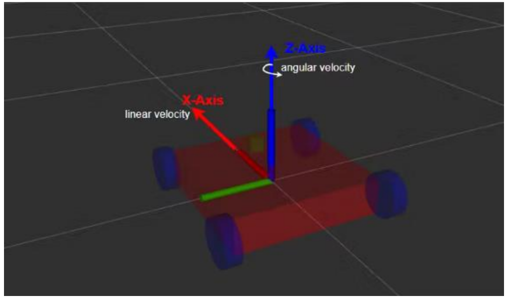


Odometry

4. Expressing the robot motion in the world reference frame:

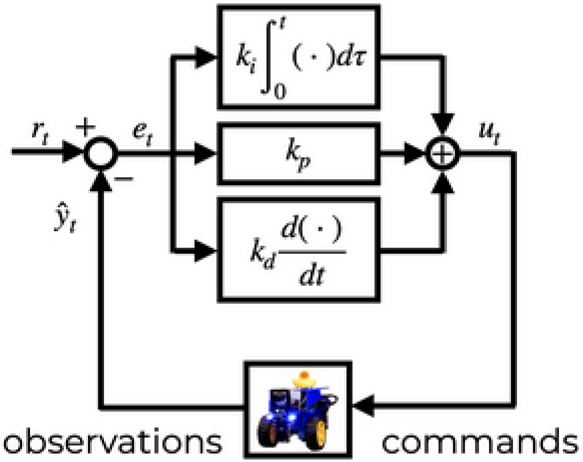
$$\Delta x_k = d_{A,k} \cos(\theta_k)$$
$$\Delta y_k = d_{A,k} \sin(\theta_k)$$

Once we know the real-time position, according to this information we calculate the linear and angular velocities of the robot:



Odometry – PID Heading Control

$$e_t = r_t - y_t \rightarrow 0 \Rightarrow y_t \rightarrow r_t$$



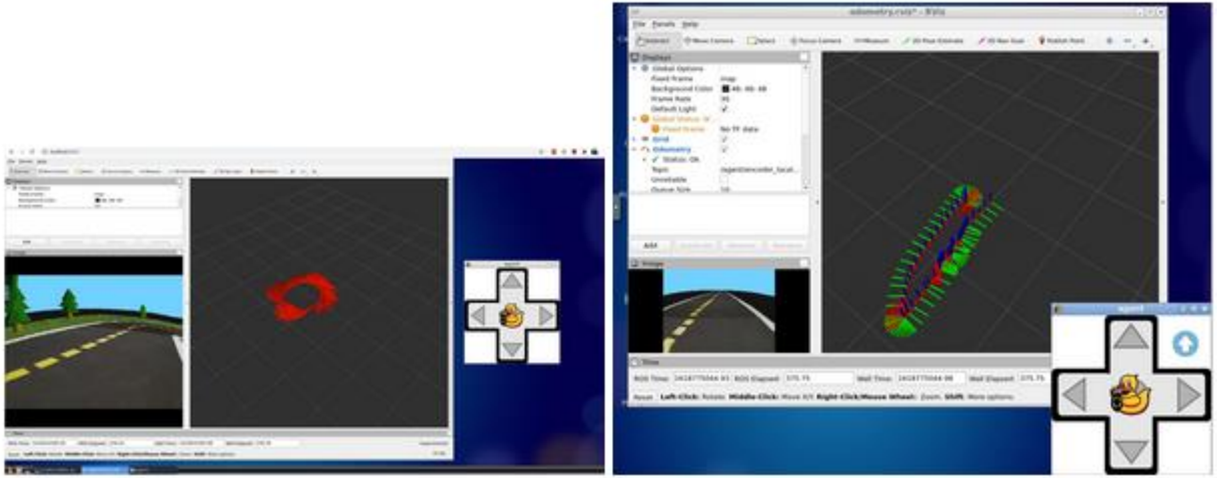
A PID control loop.

- We assume constant linear speed (v) and compute the angular speed (ω).

$$u_t = k_p e(t) + k_i \int_0^t e(\tau) d\tau + k_d \frac{de_t}{dt}$$

We need to adjust the 3 constants correctly.

Odometry and PID



We can test everything in the Exercise “Modelling and Control” from the repository ([GitHub - duckietown/duckietown-lx: Duckietown Learning Experiences](https://github.com/duckietown/duckietown-lx)). In future, learn how to regulate both velocities for out application (path-planning)