Training a Pathfinding Wheeled Robot in Webots Using LiDAR on Linux

---

1. Setup Webots on Linux

Install Webots using Snap:

sudo snap install webots --classic

---

2. Create the Webots World

a. Open Webots and Create a New World

- Go to `File` > `New World`
- Save it as `wheeled_pathfinder.wbt`

b. Add Ground and Obstacles

- Use `Solid` objects for walls and obstacles
- Add a `RectangleArena` or `Floor` for the navigation area

c. Add a Wheeled Robot

- Use a prebuilt robot like `Pioneer 3-AT` or design a custom one
- Attach the following devices:
  - `Lidar`
  - `DifferentialWheels`
  - Optionally, `Camera`, `Compass`, `GPS` for navigation and debugging

---

3. Configure the LiDAR

In the robot's configuration, add:

```
Lidar {
  name "lidar"
  horizontalResolution 512
  numberOfLayers 1
  fieldOfView 3.14
  minRange 0.1
  maxRange 5.0
  rotationFrequency 10
}
```

---

4. C Controller Setup

a. Folder Structure

```
my_project/
├── controllers/
│   └── pathfinder/
│       ├── pathfinder.c
│       └── Makefile
├── worlds/
│   └── wheeled_pathfinder.wbt
```

b. Makefile

```
TARGET = pathfinder
CC = gcc
CFLAGS = -Wall -Wextra -O2
WEBOTS_INC = $(shell webots --include-path)
WEBOTS_LIB = $(shell webots --lib-path)

pathfinder: pathfinder.c
	$(CC) $(CFLAGS) -I$(WEBOTS_INC) -L$(WEBOTS_LIB) -o pathfinder pathfinder.c
-lController

clean:
	rm -f pathfinder
```

5. C Code (pathfinder.c)

```
#include <webots/robot.h>
#include <webots/motor.h>
#include <webots/lidar.h>

#define TIME_STEP 32
#define LIDAR_NAME "lidar"
#define LEFT_MOTOR "left wheel motor"
#define RIGHT_MOTOR "right wheel motor"
#define OBSTACLE_THRESHOLD 0.6

int main() {
  wb_robot_init();

  WbDeviceTag left_motor = wb_robot_get_device(LEFT_MOTOR);
  WbDeviceTag right_motor = wb_robot_get_device(RIGHT_MOTOR);
  wb_motor_set_position(left_motor, INFINITY);
  wb_motor_set_position(right_motor, INFINITY);

  wb_motor_set_velocity(left_motor, 0.0);
  wb_motor_set_velocity(right_motor, 0.0);

  WbDeviceTag lidar = wb_robot_get_device(LIDAR_NAME);
  wb_lidar_enable(lidar, TIME_STEP);
  wb_lidar_enable_point_cloud(lidar); // Optional

  int lidar_width = wb_lidar_get_horizontal_resolution(lidar);

  while (wb_robot_step(TIME_STEP) != -1) {
    const float *lidar_values = wb_lidar_get_range_image(lidar);

    bool obstacle_left = false;
    bool obstacle_right = false;

    for (int i = 0; i < lidar_width; ++i) {
      float value = lidar_values[i];

      if (value < OBSTACLE_THRESHOLD) {
        if (i < lidar_width / 2)
          obstacle_left = true;
        else
          obstacle_right = true;
```

```
    }
  }

  double left_speed = 3.0;
  double right_speed = 3.0;

  if (obstacle_left) {
    left_speed = 1.0;
    right_speed = -1.0;
  } else if (obstacle_right) {
    left_speed = -1.0;
    right_speed = 1.0;
  }

  wb_motor_set_velocity(left_motor, left_speed);
  wb_motor_set_velocity(right_motor, right_speed);
 }

 wb_robot_cleanup();
 return 0;
}
```

---

6. Compile and Run

In the controller directory:

```
make
```

Then in Webots:

- Assign the controller to the robot
- Run the simulation

---

7. Optional: Training for Pathfinding

To go beyond simple obstacle avoidance:

- **Use Reinforcement Learning (RL):**
  - Export LIDAR/GPS data
  - Train with external tools (e.g., OpenAI Gym bridge)
  - Integrate trained policy back into Webots controller

---

8. Tips for Enhancement

- Add GPS for absolute positioning
- Implement A* or Dijkstra for global path planning
- Use LiDAR-based SLAM for mapping and localization

Title: Reinforcement Learning-Based Pathfinding in Webots from Scratch

---

Objective

To train a TurtleBot3 robot using LiDAR in a circular Webots arena to navigate from a random start location to a goal, while avoiding dynamically placed obstacles using Reinforcement Learning (RL). The model will be trained visually inside Webots and exported for reuse.

---

1. Webots Simulation Setup

a. Create a New World

1. Launch Webots.
2. Go to `File > New World`.
3. Save the world as `rl_arena.wbt`.

b. Add Arena and Obstacles

1. Insert a `Solid` floor for the arena and scale it circularly.
2. Insert a `TurtleBot3 Burger` and name it `ROBOT` using the DEF field.
3. Add 5 `Cube` obstacles and name them `CUBE0`, `CUBE1`, ..., `CUBE4`.
4. Add a small `Solid` sphere as the destination and name it `GOAL`.

c. Attach Devices to Robot

Ensure your TurtleBot3 has the following:

- LiDAR (name: `lidar`)

- Left and Right wheel motors (name: `left wheel motor`, `right wheel motor`)

---

2. Create Python Controller

a. Folder Structure

```
project_folder/
├── controllers/
│   └── rl_controller/
│       ├── pathfinder.py
├── my_rl_env.py
├── train.py
├── models/
```

b. Install Required Python Packages

```
pip install numpy gym stable-baselines3
```

---

3. Custom Gym Environment: `my_rl_env.py`

```python
import gym
from gym import spaces
import numpy as np
from controller import Supervisor

class WebotsEnv(gym.Env):
    def __init__(self):
        super(WebotsEnv, self).__init__()
        self.robot = Supervisor()
        self.time_step = int(self.robot.getBasicTimeStep())

        # Devices
        self.lidar = self.robot.getDevice('lidar')
        self.left_motor = self.robot.getDevice('left wheel motor')
        self.right_motor = self.robot.getDevice('right wheel motor')

        self.lidar.enable(self.time_step)
        self.left_motor.setPosition(float('inf'))
        self.right_motor.setPosition(float('inf'))

        # Spaces
```

```
        self.action_space = spaces.Discrete(3)  # 0: forward, 1: left, 2: right
        self.observation_space = spaces.Box(low=0.0, high=5.0, shape=(512,), dtype=np.float32)

        # Nodes
        self.robot_node = self.robot.getFromDef("ROBOT")
        self.goal_node = self.robot.getFromDef("GOAL")
        self.cubes = [self.robot.getFromDef(f"CUBE{i}") for i in range(5)]

    def step(self, action):
        speeds = [(3, 3), (2, -2), (-2, 2)]
        left_speed, right_speed = speeds[action]

        self.left_motor.setVelocity(left_speed)
        self.right_motor.setVelocity(right_speed)

        self.robot.step(self.time_step)
        obs = self.lidar.getRangeImage()
        reward, done = self.compute_reward()
        return np.array(obs, dtype=np.float32), reward, done, {}

    def reset(self):
        self.randomize_positions()
        self.robot.step(self.time_step)
        obs = self.lidar.getRangeImage()
        return np.array(obs, dtype=np.float32)

    def compute_reward(self):
        position = self.robot_node.getField("translation").getSFVec3f()
        goal = self.goal_node.getField("translation").getSFVec3f()
        dist = np.linalg.norm(np.array(position) - np.array(goal))
        return (10.0, True) if dist < 0.3 else (-0.01, False)

    def randomize_positions(self):
        import random
        self.goal_node.getField("translation").setSFVec3f([random.uniform(-1, 1), 0.0, random.uniform(-1, 1)])
        self.robot_node.getField("translation").setSFVec3f([random.uniform(-1, 1), 0.0, random.uniform(-1, 1)])
        for cube in self.cubes:
            cube.getField("translation").setSFVec3f([random.uniform(-1, 1), 0.0, random.uniform(-1, 1)])
```

---

4. Training Script: `train.py`

```
from stable_baselines3 import PPO
from my_rl_env import WebotsEnv

env = WebotsEnv()
model = PPO("MlpPolicy", env, verbose=1)
model.learn(total_timesteps=10000)
model.save("models/ppo_pathfinder")
```

---

5. Run Training

1. Launch Webots and load `rl_arena.wbt`.
2. Set the controller of TurtleBot3 to `pathfinder.py`.
3. Run the Webots simulation.
4. In a terminal, run:

```
python3 train.py
```

---

6. Test Trained Model

```
from stable_baselines3 import PPO
from my_rl_env import WebotsEnv

env = WebotsEnv()
model = PPO.load("models/ppo_pathfinder")
obs = env.reset()
done = False
while not done:
    action, _ = model.predict(obs)
    obs, reward, done, _ = env.step(action)
```

---

7. Summary

This guide helps you set up a Webots simulation and train a TurtleBot3 robot using reinforcement learning to reach a dynamic goal while avoiding randomly placed obstacles. The trained model is saved and the training process is visualized within Webots itself.

Title: Reinforcement Learning-Based Pathfinding in Webots using VS Code on Linux

---

Objective

To train a TurtleBot3 robot using LiDAR in a circular Webots arena to navigate from a random start location to a goal, while avoiding dynamically placed obstacles using Reinforcement Learning (RL). The model will be trained visually inside Webots and exported for reuse. The entire development will be done using Python 3 and Visual Studio Code (VS Code) on a Linux system.

---

1. Full Setup Guide

a. Install Webots on Linux

1. Download the latest AppImage from: https://cyberbotics.com/

Make it executable:

chmod +x Webots-*.AppImage

./Webots-*.AppImage

2.
3. Follow the installation steps and allow Webots to add itself to your system path.

b. Install VS Code

Install via terminal:

sudo snap install code --classic

1.

Launch it with:

code

2.

c. Set Up Python 3 Environment

Ensure Python 3 is installed:

```
python3 --version
```

1.

Install pip and dependencies:

```
sudo apt update
sudo apt install python3-pip
pip3 install numpy gym stable-baselines3
```

2.

Install the Webots Python controller interface:

```
pip3 install controller
```

3.

d. VS Code Setup

1. Open VS Code.
2. Install the Python extension from Microsoft (search "Python" in Extensions).
3. Open the folder where you'll build your project (e.g., `webots_rl_project/`).
4. Create and organize files as per the structure below.

---

2. Project Structure

```
webots_rl_project/
├── controllers/
│   └── rl_controller/
│       └── pathfinder.py
```

├── my_rl_env.py

├── train.py

├── log.txt

├── models/

---

3. Webots Simulation Setup

a. Create the Arena

1. Open Webots.
2. File > New World > Save as `rl_arena.wbt`.
3. From the Scene Tree, right-click > Add > Solid > set shape to a flat cylinder or circular platform as the arena base.

b. Add TurtleBot3

1. Drag and drop `TurtleBot3 Burger` from the Robot window.
2. In the properties, rename it as `ROBOT` using the `DEF` field.

c. Add Obstacles

1. Add five cubes from the `Solid` section.
2. Name them as `CUBE0`, `CUBE1`, ..., `CUBE4` in the `DEF` field.

d. Add Goal Marker

1. Add a sphere or small colored cube to represent the goal.
2. Name it `GOAL` in the `DEF` field.

e. Add and Configure Devices

1. On the `ROBOT`, make sure to add:
    - A `Lidar` sensor named `lidar` (with horizontal resolution ≥ 512).
    - Two motors named `left wheel motor` and `right wheel motor`.
2. Enable Supervisor mode for the controller in the robot's settings.

f. Save the World

1. Save your progress (`File > Save World`).

---

4. Custom Gym Environment: `my_rl_env.py`

[...unchanged content remains here...]

---

5. Training Script: `train.py`

[...unchanged content remains here...]

---

6. Running the Setup in VS Code

1. Open your `webots_rl_project/` folder in VS Code.
2. In Webots:
    - Open `rl_arena.wbt`
    - Set the robot controller to `pathfinder.py` from the dropdown.

In VS Code Terminal, run:

python3 train.py

3.

4. Observe Webots for live training animation.

5. Check `log.txt` for rewards, actions, and episode summary.

---

7. Testing the Trained Model

[...unchanged content remains here...]

---

8. Summary

This guide provides a full setup from scratch for developing and training a reinforcement learning agent using Webots and Python inside VS Code on Linux. Visual training is observed inside Webots, while detailed logs are stored in `log.txt`, and the final model is saved to disk.