Exploring the V-REP simulation environment

ELA406 - Mobile Robotics

Ulrik Åkesson
Master's Programme in Robotics
Academy of Innovation, Design, and Engineering, Mälardalen University, Sweden
Email: uan13001@student.mdh.se

***Abstract*—As a part of a project in the course Mobile Robotics, a robot has been simulated using V-REP. The simulated robot uses a differential drive and is implemented to follow a line and avoid obstacles. Two tasks which required vision and proximity sensors to be implemented in V-REP. The simulation was controlled using code written in Python by using the remote API support in V-REP.**

1. Introduction

In the field of robotics simulation is a powerful tool and there exist a multitude of different tools with different strenghts and weekneses. A couple of examples are Gazebo [1], Open Dynamics Engine (ODE) [2], and Virtual Robotics Experimentation Platform (V-REP) [3]. Where according to a study from 2014 by S. Ivaldi *et. al.* Gazebo is the most commonly used tool, ODE is the most well known, and V-REP has the highest user satisfaction [4].

1.1. Background

As a project in the course ELA406 - Mobile Robotics one possible task was to explore a simulation environment and implementing a line following robot with obstacle avoidance capabilities. Based on the suggestion from the teacher the choice of simulation environment landed on V-REP.

Further supporting that V-REP is an interesting simulation environment to use for this project is a comparison between Gazebo and V-REP which L. Nogueira did in 2014 in which it was concluded that V-REP is a more user-friendly and easy to use than Gazebo as well as being more feature rich [5].

2. V-REP

V-REP is as mentioned in the introduction a simulation environment. It is developed by Coppelia Robotics and is a licensed software. However, it is available for free under an educational license for students, teachers, professors, and hobbyists as well as schools and universities. V-REP availible on the three major operating systems, GNU/Linux, Windows, and macOS but for this project, only the Windows version have been used.

2.1. Programming languages

The primary programming language used for scripting in V-REP is Lua and scripts for the robot can be created in the application without any other software. However, V-REP supports other programming languages via its Remote API framework [6]. These programming languages are C/C++, Java, Matlab, Octave, and Python. In this project all the code is written in Python 3.

2.2. Working with V-REP

Working with V-REP is a subjectively easy experience. This is due to the extensive and well-written documentation and tutorials made available by Coppelia Robotics [7] and the video tutorials made available on YouTube by Nikolai K. [8]. Creating and modifying a scene or a robot is done in the application, and a lot can be done by dragging and dropping objects into the scene, but there is also possible to make adjustments with finer control via menus.

In V-REP there are a lot of models of different types of robots available, both mobile and non-mobile and a lot of them are models of real world robots such as e-puck [9] and Baxter [10]. There are also a lot of models related to the environment, such as furniture, plants, and persons to mention a few. These models allow an experiment to be set up faster, and more time can be devoted to simulation and testing rather than creating a robot model or building a world for the simulation to take place in. In figure 1 it can be seen what V-REP looks like when a new scene is first created.

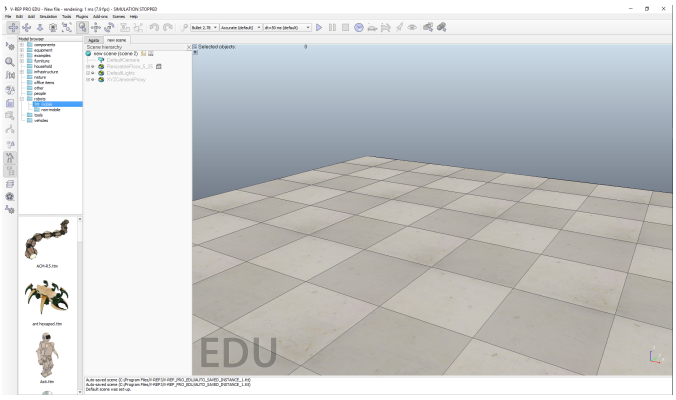

Fig. 1: In this figure a new scene have been opened in V-REP. The model browser to the left contains a lot of different useful models of robots, furniture, and people to mention a few.

3. Implementation

For this project, line following and obstacle avoidance were implemented on a differential drive robot. The base of the robot was created from scratch by following the instructions in Coppelias tutorial and then appropriate sensors where added based on the environment the robot were to operate in as well as on the tasks of following a line and avoiding obstacles. The robot can be seen in three different views in figures 2, 3, and 4.

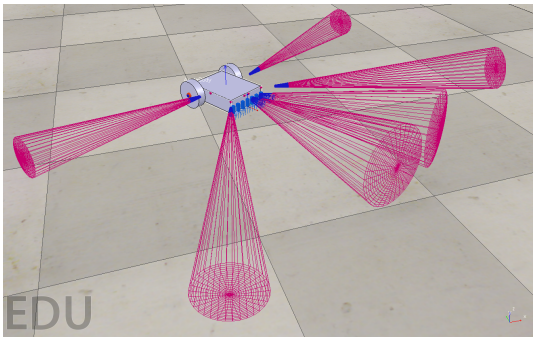

Fig. 2: The differential drive robot created for this project.

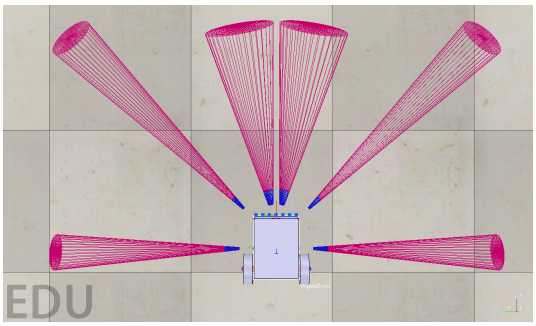

Fig. 3: The differential drive robot created for this project. The sensors seen in the figure from bottom left and clockwise are defined as **Edge**, **Corner**, **Front**, **Front**, **Corner**, **Edge**.

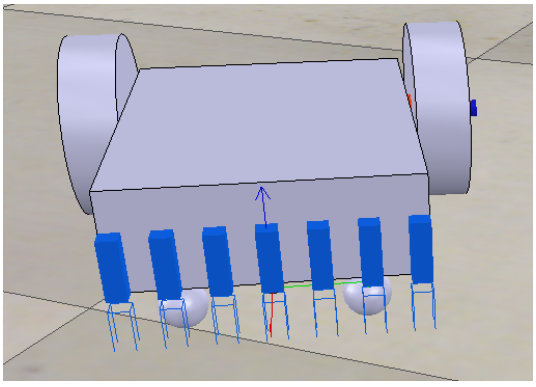

Fig. 4: The differential drive robot created for this project. The sensors seen in the figure are the intensity sensors used to detect the line.

3.1. Actuators

Since this is a differential drive robot only 2 actuators are needed, more specifically a motor for each of the two wheels. In V-REP that is represented by revolute joints.

3.2. Sensors

Two types of sensors are used, the first type being a color/intensity sensor that is used to identify and follow a line. The other type is ultrasonic distance sensors which are used to detect obstacles. A total of seven intensity sensors are used represented in V-REP as vision sensors and a total of 5 ultrasonic sensors are used which are represented by proximity sensors in V-REP. The vision sensors are configured to be orthographic with a resolution of 1x1 pixels and the proximity sensors are configured to have a maximum range between 0.6 and 0.8 meters depending on the position and use of the sensor.

3.3. Controller

The controller as previously mentioned includes two behaviors, one to follow a line and one to avoid obstacles. The code is structured in such a way that after initialization it transitions into the line follower state where it stays until an obstacle is detected, at which point it transitions to the obstacle avoidance state where it stays until the robot is back on the line. In figure 5 this behavior is abstracted in the form of a state machine.

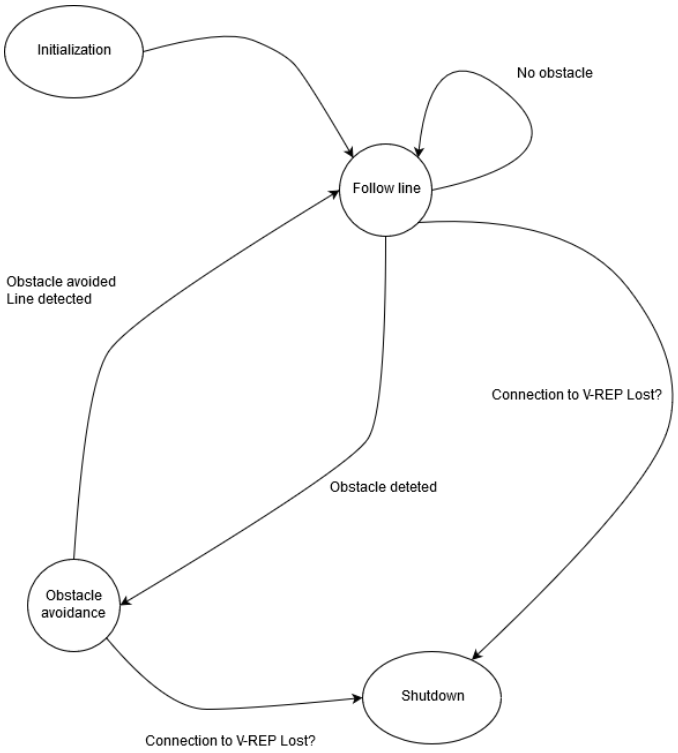

Fig. 5: The state machine which describes the transitions between the two states of the controller.

The sequence of which the code flows over each iteration is abstracted as a block diagram in figure 6.

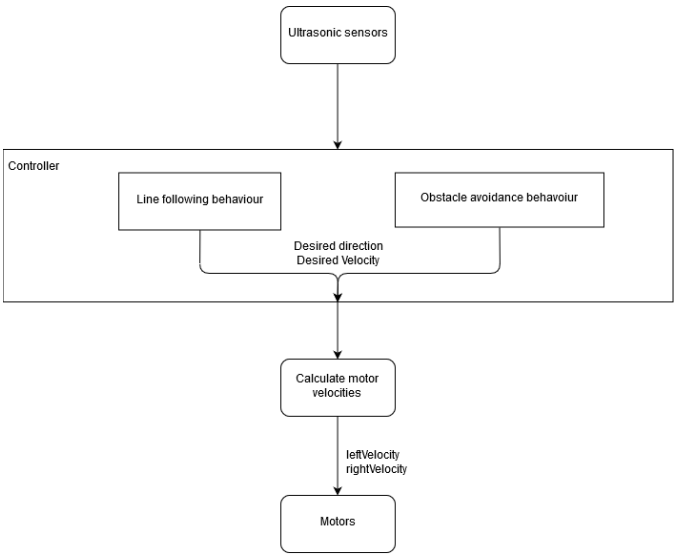

Fig. 6: Block diagram abstraction of the controller.

The line follower is implemented around the intensity sensors which is used to detect the line. When the line is detected the error is calculated based on the distance between where the line is and the center of the robot. This error is used in a proportional controller to calculate the desired direction and the desired velocity of the robot. The desired direction is calculated by a proportional controller as seen in equation 1 and the desired velocity is calculated by the inversely proportional controller as seen in equation 2. This setup with the two controllers means that the bigger the error the more the robot steers and the slower it moves, and if the error is small it will move faster and steer less. The gains for the controllers are tuned to the robot and is set to $gainP_{steering} = 5$ and $gainP_{velocity} = 1.2$. $velocity_{max} = 2$ The algorithm in a more general form is described in algorithm 1.

$$direction_{desired} = gainP_{steering} * error \tag{1}$$

$$velocity_{desired} = velocity_{max} - |gainP_{velocity} * error| \tag{2}$$

The obstacle avoidance is implemented with the help of the distance sensors and is written to be able to avoid the obstacles in the simulation environment. The obstacle avoidance algorithm takes into the account of the last desired direction of the robot so that when it detects an obstacle it can turn away from the obstacle towards where it came from which is assumed to be an area free from obstacles since no sensors were triggered there earlier. The obstacle avoidance algorithm can be seen in algorithm 2.

Data: *Boolean* detected line
Read intensity sensors;
if *Line detected* **then**
  Calculate error;
end
Calculate desired direction;
if *Desired direction is out of range* **then**
  Coerce desired direction;
end
Calculate desired velocity;
if *Desired velocity is out of range* **then**
  Coerce desired velocity;
end

Algorithm 1: Line follower behaviour for the line following differential drive robot. The intensity sensors can be seen in figure 4.

Data: *Boolean* detected obstacle
if *Front sensor detect obstacle* **then**
  Enter obstacle avoidance behaviour;
  while *Corner sensor doesn't detect obstacle* **do**
    Turn away from obstacle;
  end
  while *Edge sensor don't detect obstacle* **do**
    Drive straight ahead;
  end
  while *Corner sensor don't detect obstacle* **do**
    Turn toward obstacle;
  end
  while *Edge sensor don't detect obstacle* **do**
    Drive straight ahead;
  end
  while *Edge sensor detect obstacle* **do**
    Drive straight ahead;
  end
  while *Line not detected* **do**
    Drive forward and turn toward the line;
  end
end

Algorithm 2: Obstacle avoidance behaviour for the line following differential drive robot. The sensors can be seen in figure 3 and from bottom left and clockwise are defined as **Edge**, **Corner**, **Front**, **Front**, **Corner**, **Edge**.

3.4. Simulation scene

The scene implemented for this project can be seen in figure 7 and is designed to test the robots ability to follow the line and to avoid obstacles and return to the line. The track consist of two parts, one easier with more straight parts and less sharp turns, and one more challenging part with sharper turns and barely any straight parts. The obstacles are either cylinders or rectangle placed arbitrarily along the path.

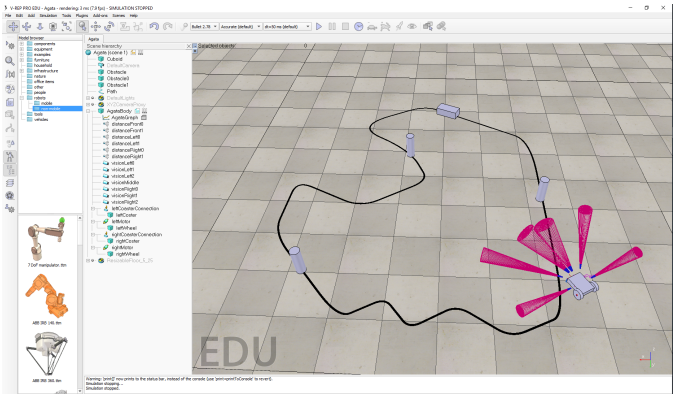

Fig. 7: The scene used for the simulations in this project

4. Results

In V-REP it is possible to attach a graph element to the robot which will plot the robot's path as a 3D-graph and that is what will be used to visualize the result of the controllers ability to follow the line and avoid obstacles. In figure 8 it can be seen that the robot closely follows that path without any big deviance. This part of the track is easier due to long straight sections and no sharp turns. In figure 9 the path is harder with more and sharper turns, and it can be seen that the purple 3D-graph does not follow the line as closely. In figure 10 obstacles have been added to the path. It can be seen that the robot avoids the obstacles and finds its way back to the path.

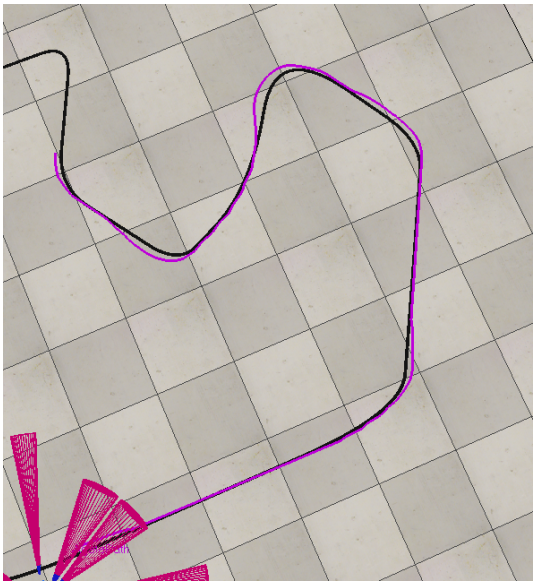

Fig. 8: The robots path on the easier part of the track.

Fig. 9: The robots path on the harder path of the track.

Fig. 10: The robots path when avoiding obstacles.

5. Discussion

Using V-REP has been a quite easy experience due to very good documentation and the well written and/or explained tutorials available. The ease of with the scene was set up and the robot created meant that more time could be put into writing the scripts and improve the behaviours.

It also means that it would possible to more rapidly test different types of robots or control algorithms, which may be slowed down if the simulation environment is slower and more tedious to use. The support for other programming languages than Lua via the remote API makes it more accessible and

versatile since a language that is more familiar or perhaps better suited to the actual application can be used.

REFERENCES

[1] O. S. R. Foundation, "Gazebo," http://gazebosim.org/, accessed 2018-08-29.

[2] R. Smith, "ODE: Open Dynamics Engine," https://www.ode.org/, accessed 2018-08-29.

[3] C. Robotics, "V-REP: Virtual Robotics Experimentation Platform," http://www.coppeliarobotics.com/, accessed 2018-08-29.

[4] S. Ivaldi, V. Padois, and F. Nori, "Tools for dynamics simulation of robots: a survey based on user feedback," *CoRR*, vol. abs/1402.7050, 2014. [Online]. Available: http://arxiv.org/abs/1402.7050

[5] L. Nogueira, "Comparative analysis between gazebo and v-rep robotic simulators," *Seminario Interno de Cognicao Artificial-SICA*, vol. 2014, p. 5, 2014.

[6] C. Robotics, "V-REP: Remote API Overview," http://www.coppeliarobotics.com/helpFiles/en/remoteApiOverview.htm, accessed 2018-08-29.

[7] ——, "V-REP: Tutorials," http://www.coppeliarobotics.com/helpFiles/en/tutorials.htm, accessed 2018-08-29.

[8] N. K., "Nikolai K: Tutorials," https://www.youtube.com/playlist?list=PL38P7Q24q4XA7c0uNj0kO4or-bKhFYdIg, accessed 2018-08-29.

[9] E. P. F. de Lausanne, "e-puck education robot," http://www.e-puck.org/, accessed 2018-08-29.

[10] R. Robotics, "Baxter," https://www.rethinkrobotics.com/baxter/, accessed 2018-08-29.