Awesomizing the P2DX

AP2DX

Wadie Assal 6398693 Jasper Timmer 5995140 Maarten de Waard 5894883
Maarten Inja 5872464

June 24, 2011

Contents

1 Preface

As part of the course Software Engineering and Distributed Applications, of the University of Amsterdam, we are asked to program a distributed application for a virtual robot. The team consists of students with different profiles: three are from Artificial Intelligence, one from Computer Science and ICT at the Amsterdam University of Applied Sciences and one from the pre-master Software Engineering. This melange is both a chance to create something good from different viewpoints as well as a point of attention, because none of us has followed the bachelor Computer Science at the University of Amsterdam. TODO.

2 Introduction

For the client a robot controller that can autonomously map the yellow arena had to be build. This was attempted in less than four weeks. After two and three weeks there were milestones to show the client the progress.

2.1 USARSim

As simulator USARSim is used. USARSim (Unified System for Automation and Robot Simulation) is an application to simulate the real robot arena's of the IEEE, based on the Unreal engine. The robot should be able to navigate through the 'Yellow Arena'. This is a map with a few static obstacles.

2.2 Structure of this document

This document describes the setup of the development environment and the architecture of the program. In the appendices there is more technical information.

3 Goals

With the client there is agreed to build the following features in the robot-controller:

- Loosely coupled modules based on network communication
- Robot should be safe, i.e. stop for obstacles
- Robot should be able to drive autonomously through the environment
- Robot should be able to create a map of the environment

3.1 Milestones

At every agreed milestone there was a deliverable plannened. This deliverable can contain software and or documentation.

The first milestone was agreed to be:

- Drive: The program should be able to direct the robot through the environment but not yet be able to follow lines or walls.
- Avoid collision: The robot should be able to avoid collision with objects and walls. It will stop, turn to a random angle, and drive on. This way it will cover most of the area without colliding.
- Experiment and content of final report: A draft of the final report, containing a description of the experiment and its contents.

What was actually delivered at this milestone was a program that could only spawn a robot, and Javadoc describing the API of the application. This was partly because writing tests took a lot of time and the baseclass was not yet ready. More about the baseclass later.

As second milestone there should be:

- Avoid obstacles: The robot will be able to avoid the obstacles that cross its path, in stead of stopping and turning a random corner.
- Navigate: The robot will be able to navigate through the room.
- Mapper: A class that creates a map of the room out of the sensor data. In the time of milestone two it does not have to be able to create an entire map and be very accurate, but it will be able to make some implementation
- Improved Sensor: The sensor class will be improved to be able to make an accurate map
- Improved Reflexes: The reflex class needs to be able to use some sensor data to be able to avoid objects appropriately.

The deliverable for this milestone also did not meet the agreed requirements. TODO.

The final delivable should meet all the goals. TODO.

4 Architecture

4.1 Introduction

AP2DX is written in the Java programming language. This was choosen as it is advertised to be reliable and fast by the company Flowtraders. This is important for AP2DX, because it should be a safe robot controller, that stops in time and does not harm anyone on it's path. The architecture of AP2DX is based on Object Oriented Programmin (OOP). To not repeat the same code again and again for every module, a baseclass was constructed. This class could do things as: read a config file, write to a logfile, accept incoming connections, start outgoing connections based on the config file, handle incoming messages and send responses. The baseclass was designed to be flexible for the needs of every module. Also, the program is heavily multithreaded, to open and check connections and to do the business logic and send messages. As for the messages: a base messageclass was build, to facilitate the communication between USARSim and AP2DX, and between modules of AP2DX. As standard for config file and communication, JSON (Javascript Object Notation) was used.

TODO: Class diagram etc.

4.2 USARSim

The simulator exists of a virtual enviroment where different maps and different robots can be loaded. These robots have different sizes and different features. As example of features of a robot, there are different kinds of sensors and different kinds of wheels. For AP2DX, the robot P2DX was used. This robot is a three-wheel, rear swivel wheel robot. The two frontwheels can be controlled independently of eachother, so it is possible to turn almost in place. As sensors are available on the P2DX:

- 8 Sonar distance sensors
- 1 Laser distance sensor
- Odometry
- Internal Navigation System
- Camera

For the purpose of AP2DX only sonar and laser sensors are used.

TODO: Picture.

4.3 Coordinator

The first module a message on a journey from the USARSim to AP2DX meets, is the AP2DX Coordinator. The Coordinator is programmed to spawn the robot and translate and relay traffic between the simulator and the AP2DX System. That is all it really does. It has a connection with the Sensor module to relay the incoming sensor data and the Motor module to receive commands for the simulator.

4.4 Sensor

When a message is translated by the coordinator the journey continues to the AP2DX Sensor module. This module creates a visualisation of sonar and laser sensor-data in a user interface window, so it is possible to see what the robot sees. Then the message is cloned: she is sent to the Mapper module and the Reflex module.

4.5 Mapper

In the mapper module all the sensor information comes together and here is where the information is going to make sense. The sensor information combined with the position of the robot the mapper module will construct a map. This map is then send to the planner module. If there are any irregularities with the position of the robot it will be corrected here and send to the planner.

4.6 Planner

The planner module is the brain of the software. It directs the robot to unexplored places and solves problems with obstacles. It will make its decisions based on the map send from the mapper module and the obstacle information send from the reflex module. The planner is also the place where the position of the robot is saved. The position is periodically sent to the mapper so the postion can be corrected by the mapper if needed.

4.7 Reflex

The robot is controlled by the reflex module through the motor module. The reflex module will take the responsibility for unexpected situations that could not be foreseen by the planner module. For example low obstacles or unexpected falling object in the path of the robot. The reflex module will stop the robot so the arising situation can be handled gracefully by the planner module. The information needed to make a new route is send to the planner module from the reflex module.

4.8 Motor

The motor modules main function is to provide an abstract layer for the reflex module to control the robot. This will disguise the complexities of controlling the robot. For example if you want to control the robot on a low level, you will have to send a message to drive forward, calculate if the desired distance is covered and then stop the wheel taking into account sliding of the robot because of the momentum. Through the motor module you can control the robot by sending a drive forward message with the desired distance that needs to be covered. The module itself will calculate the distance and stop the robot on the correct place.

4.9 Base Class

OBSOLETE? here we will discuss the Base Class

4.10 Communication

here we will discuss how our separate modules communicate with eachother and with USARSim

5 Testing

In this section, we will explain the way we tested our program, using the testserver with Jenkins

6 Experiment

Here we will explain the experiment and how we did the tests. (can be more than one experiment).

7 Results

Here we will discuss the results from our experiment and explain what it means

8 Future Work

In this section, we discuss what future work could improve our program and what kind of uses we can think of for our program.