A Robotics Environment for Software Engineering Courses

Stephan Göbel, Ruben Jubeh, Simon-Lennert Raesch
[sgoe|ruben|lra]@cs.uni-kassel.de
University of Kassel, Software Engineering
Wilhelmshöher Allee 73
34121 Kassel, Germany

Abstract

The initial idea of using Lego Mindstorms Robots for student courses had soon to be expanded to a simulation environment as the user base in students grew larger and the need for parallel development and testing arose. An easy to use and easy to set up means of providing positioning data led to the creation of an indoor positioning system so that new users can adapt quickly and successfully, as sensors on the actual robots are difficult to configure and hard to interpret in an environmental context. A global positioning system shared among robots can make local sensors obsolete and still deliver more precise information than currently available sensors, also providing the base necessary for the robots to effectively work on shared tasks as a group. Further more, a simulator for robots programmed with Fujaba and Java which was developed along the way can be used by many developers simultaneously and lets them evaluate their code in a simple way, while close to real-world results.

Introduction

Figure 1: Robot vehicle

To use robotics within our lectures and courses aimed to visualize and physically represent object oriented software algorithms, which is even more intuitive than just looking at in-memory object structures, as shown in (I. Diethelm, L. Geiger, A. Zündorf 2002). A basic framework was developed in 2008 (Jubeh 2008), based upon the LeJOS framework[1]. The first robot solved the Towers-of-Hanoi-Game, see (I. Diethelm, L. Geiger, A. Zündorf 2003). Since the last four student terms, we realized more application scenarios, see (Van Gorp et al. 2009).

Using robotics in Software Engineering courses and projects is very attractive for students, as programs and algorithms physically manifest and interact with real-world objects. But using robotics also has it's drawbacks: simple actions like moving the robot to certain places is an advanced task, since one needs to control the actuators (motors) and also verify through sensors, that the action succeeded. For example, using light sensors for orientation, as the Hanoi Robot did, is unreliable. Generally, interpreting the real world state by various sensors is complex and often error-prone. Low level hardware interfacing software is difficult to develop and test. Mainly because the "Software Engineering with Robots"-Course at Kassel University was so popular that more students enrolled to that course than we had robotic kits, we decided to also build a simulator around our framework. This simulator allows to develop and test the actual robot control code independently of the hardware at a higher level of abstraction.

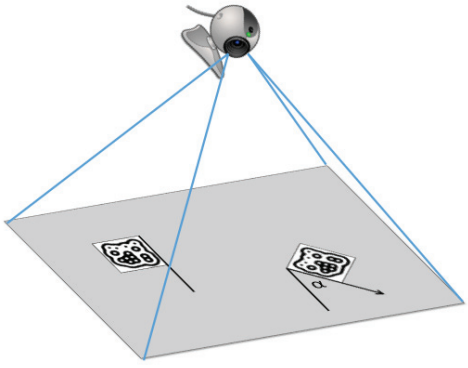

Figure 2: Webcam tracking

The robot vehicles currently used are depicted in figure 1.

[1] http://lejos.sourceforge.net/

Two big main wheels allow differential steering and turning on the spot. On top of the robot, there's a marker, which is used to track the robot optically. The robot doesn't use any directly attached sensors, which simplifies programming it a lot. A webcam is tracking the markers, recognizing positions and forwarding the information via network broadcast to the robot control software instances. By using this optical global-view tracking approach, each robot individually sees it's own position and the position of other marked objects. The framework also supports a view distance, objects tracked too far away are artificially hidden. The tracking is the only sensory input source used in the programming model. This approach simplifies the simulator a lot, because we don't have to simulate any distance metering sensors like ultrasonic or laser/light sensors. Only position data has to be generated out of the movement simulation component.

Positioning

Our positioning component, called FIPS (Fiducial Indoor Positioning System), is based on the reacTIVision[2] framework, which uses image recognition techniques to detect predefined optical markers via webcam. These markers are called *fiducials*. Each symbol has an ID which can be detected along with the rotation and, of course, the absolute position. Especially having the exact rotation of the robot helps a lot for precisely navigating it, a feature that other in- or outdoor position systems can only achieve by utilizing a magnetic field sensor, which is unreliable even in indoor scenarios.

Simulator

The main motivation for the simulator is to allow transparent development of scenarios and control software without the need for actual robot hardware. It features a simple 3D graphical viewer and simulates navigational movements and rotation in 2D space. Robot movements are translated in x, y coordinates and a heading value. The *navigator* is the only used actuator in all scenarios presented here, which abstracts the underlying two wheel driving motors. Additionally, there's an integration of a simple 2D geometric/physics engine for collision handling between robots and other geometric shapes. To close the simulation loop, the 2D positions of all objects have to be fed back to the FIPS, which is the only sensory interface the robot control software uses. Some fuzzing transformations identified in real-world tests are applied to the coordinates to mimic realistic behavior.

Scenarios

During a term project, students have to implement the autonomous robot vehicle control software and parts of the application scenario as software for the simulator. At the beginning of each project the scenario was a textual description of what the robot vehicles had to achieve under given rules. Later on, the scenario was implemented in actual code setting up the robots etc. In case of a simulation, all the environment has to be simulated as well. So, a scenario offers basically two operation modes, controlling real robots in a real environment, or simulating robots within a virtual environment.

The most popular application scenario, *Cat-and-Mouse*, consists of two robots, one playing the role of a cat (the hunter), the other playing a mouse (the prey). The cat periodically scans for the mouse and tries to catch it by driving to it's position. The mouse moves randomly around, until there is a cheese object (simply a box with a fiducial marker) being placed in the field, which instantly attracts the mouse's attention. What makes the scenario so attractive is that people can interact with both robots by placing the cheese somewhere, which enforces a reaction of the mouse and might help to escape or being trapped.

Workshop Proposal

We would like to organize a hands-on programming event, extending the *Cat-and-Mouse* scenario with more interactivity and adding a third robot/role, e.g. a Dog. The FIPS system and a robot for each team member is provided. We would like to discuss the desired robot behavior first, and then implement it using Java programming in Eclipse. Team members should be familiar with general Java programming, no advanced skills are required. The tutors, who are also framework developers, can quickly help with programming questions and pitfalls.

Summary

Our system requires nothing more than a webcam and some printed sheets of fiducials. This makes this system very applicable for our department, as we are software- but not hardware experts. Furthermore, installation costs are extremely low. The highly reduced complexity compared to other indoor positioning systems allows a quick and easy setup of the system on other sites. Students can even rebuild it at home. Therefore, we would like to present our project and discuss further ideas with the participants of the workshop, introduce our development framework and ideally organize a hands-on programming session.

References

I. Diethelm, L. Geiger, A. Zündorf. 2002. UML im Unterricht: Systematische objektorientierte Problemlösung mit Hilfe von Szenarien am Beispiel der Türme von Hanoi. *Erster Workshop der GI-Fachgruppe Didaktik der Informatik, Bommerholz, Germany*.

I. Diethelm, L. Geiger, A. Zündorf. 2003. Fujaba goes Mindstorms. *Objektorientierte Modellierung zum Anfassen; in Informatik und Schule (INFOS) 2003, München, Germany*.

Jubeh, R. 2008. Simple robotics with Fujaba. In *Fujaba Days*. Technische Universität Dresden.

Van Gorp, P.; Jubeh, R.; Grusie, B.; and Keller, A. 2009. Fujaba hits the Wall(-e) – Beta working paper 294, Eindhoven University of Technology. http://beta.ieis.tue.nl/node/1487.

[2] http://www.reactivision.org/