VIRTUAL-IUB:
the 2006 IUB Virtual Robots team

Yashodan Nevatia, Mentar Mahmudi, Stefan Markov,
Ravi Rathnam, Todor Stoyanov, and Stefano Carpin

School of Engineering and Science
International University Bremen – Germany
s.carpin@iu-bremen.de

Abstract. The VIRTUAL-IUB team aims at the development of a fully autonomous team of agents that explore and map an unknown virtual environment. Each robot is controlled by a simple behavior based architecture. The team is heterogeneous in order to exploit the different mobility abilities offered by wheeled and tracked platforms. Explicit communication between the agents happens only to perform selected tasks, like map merging. This description paper offers some details about the software architecture used.

1 Introduction

The Virtual Robots competition runs under the Robocup Rescue Simulation League umbrella with the goal of providing a sound bridge towards the Robocup Rescue Real Robots League [1][2]. For this first competition, our efforts in developing a team of autonomous Virtual Robots that operate in an unknown environment have been mainly directed towards mapping and exploration. These are among the basic capabilities that a rescue robot should exhibit. In particular, we have focussed our efforts towards the integration of a freely available mapping software that has been developed and tested on real robots [3]. Our goal is to prove that a seamless migration of code between simulated agents and real robots is possible. Figure 1 shows an example of map built by the robot while exploring the yellow arena. Robots in the team at the moment operate mainly in a selfish way, i.e. at the moment they do not coordinate their exploration efforts. Cooperation happens only when robots actively merge their partial maps.

2 Virtual agents

Among the many different robots offered within USARSim, we have chosen to focus our attention on the P2DX model and the Talon (see figure 2).

Both models are equipped with frontal range scanners and sonars, INUs, RFID and victim sensors. Cameras are mounted on the robots, but at the moment are not used to identify victims. They are instead used only to gather

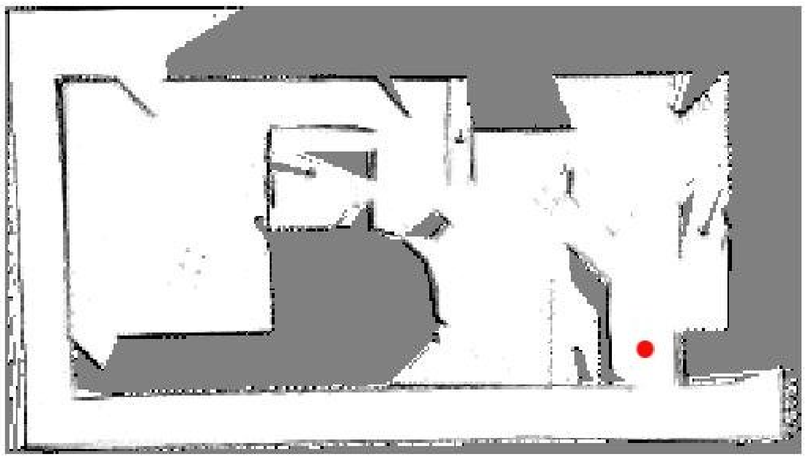

Fig. 1. A map produced by the robot while exploring the virtual yellow arena. The red dot indicates the robot

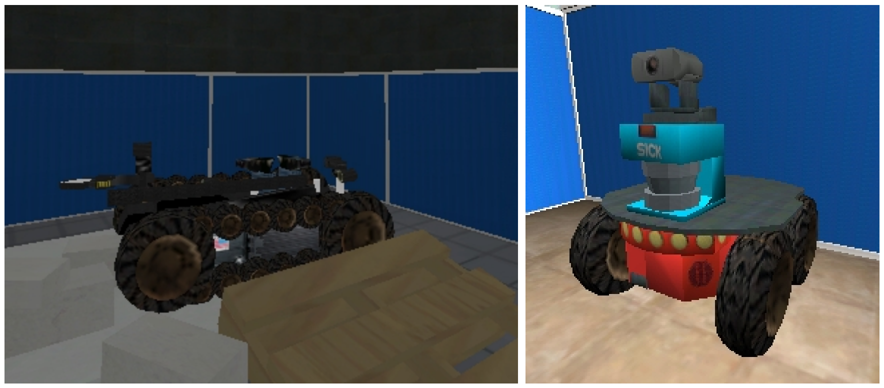

Fig. 2. The Talon robot (left) and the P2DX robot (right)

additional information when victims are detected with the victim sensor. In order to control robots in USARSim, we have created suitable Player [4]. drivers. The reason to use two different robot platforms is in the different mobility capabilities offered. The P2DX is a relatively small wheeled platform ideal to explore wide flat areas. The Talon instead is a much bigger platform equipped with tracks and therefore capable of overcoming rough terrains, ruble piles, step fields and so on. On the other hand the track arrangement is such that one issue the same motion commands both the P2DX and to the Talon, therefore the same software can run on both of them without modifications.

3 Software structure

Both wheeled and tracked robots are controlled by the same software. The *selfish* software component structure, i.e. the one non-cooperative, is depicted in figure 3. The function of each module is described in the following.

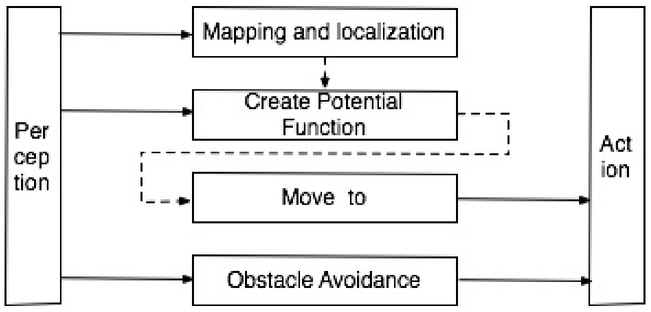

Fig. 3. The software structure controlling each robot

Perception: this module polls the sensors at a fixed frequency and provides access methods to deliver data to the other modules. All other modules have no direct access to robot sensors

Action: the *Action* module is module in charge of issuing motion commands to the platform. Only the *Obstacle Avoidance* and the *Move To* modules can send commands to the *Action* module. In case discrepant commands are sent at the same time, the *Action* module performs the arbitration by giving higher priority to *Obstacle Avoidance.*

Mapping and Localization: the *Mapping and Localization* module processes inputs coming from the laser range scanner, the INU and the odometry sensor in order to build a metric map. This module integrates the software described in [3]. The *Mapping and Localization* module provides methods to access the map (used by the *Create Potential Function* module, and to retrieve the best pose estimation (used also by the *Move To* module).

Create Potential Function: this module is responsible to decide where the robot will move next. This decision is encoded into a potential function having a global minima at the target point that is passed to the *Move To* module, hence the name. The input to this module is the map and pose produced by the *Mapping and Localization* block, and the input coming from the victim sensor. In case no victim is detected the module runs the frontier based exploration algorithm [5] and detects one frontier to reach in order to better explore the environment. In case one victim is detected, the module instead selects the victim location as target position. In both cases the module then builds a potential function with a global minima at the target point

Move To: the Move To module processes the potential function provided by the *Create Potential Function* module and the pose estimation given by *Mapping and Localization* and decides how to move the robot following a gradient descent. The use of a potential function coupled with the pose estimation allows to correct possible motion errors that could emerge while moving (so called *feedback plan*), as opposed to the use of an open loop path planner.

Obstacle Avoidance : this module processes input coming form the range scanner and from the sonars and is triggered only when an obstacle closer than a certain lower threshold is detected. In this case it sends to the *Action* module a command aimed to move the robot away from the obstacle.

The software is written in C++ and each module is implemented as an independent thread (we use the *zthread* library for an agile implementation of threads in C++).

3.1 Cooperation

The cooperative part is still under development and builds over the software we developed for the Virtual Robots demo held in Osaka during Robocup 2005. The idea is that when robots meet, they stop the *selfish* behavior and localize each other in the respective maps. This can be done using the camera, or even mounting and RFID tag on the robots. Once the relative positions are known, map merging is a straightforward process. Map inconsistencies, i.e. regions believed to be free by one robot and occupied by the other, are handled in a defensive way, declaring them occupied. If this is not the case, such errors will be easily recovered by the mapping software once resumed. After the merging is completed, robots resume their selfish behavior and the merging process is inhibited for a certain time window, in order to prevent repeated map merging between the same couple of robots.

References

1. S. Carpin, M. Lewis, J. Wang, S. Balakirski, and C. Scrapper. Bridging the gap between simulation and reality in urban search and rescue. In *Robocup 2006: Robot Soccer World Cup X*, LNCS.
2. S. Carpin, J. Wang, M. Lewis, A. Birk, and A. Jacoff. High fidelity tools for rescue robotics: results and perspectives. In *Robocup 2005: Robot Soccer World Cup IX*, LNCS, pages 301–311, 2006.
3. S. Grisetti, C. Stachniss, and W. Burgard. Improving grid-based slam with rao-blackwellized particle filters by adaptive proposals and selective resampling. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 2432 – 2437, 2005.
4. R.T. Vaughan, B.P. Gerkey, and A. Howard. On device abstractions for portable, reusable robot code. In *Proceedings of the IEEE/RSJ IROS*, pages 2421–2427, 2003.
5. B. Yamauchi. A frontier-based approach for autonomous exploration. In *International Symposium on Computational Intelligence in Robotics and Automation*, pages 146–151, 1997.