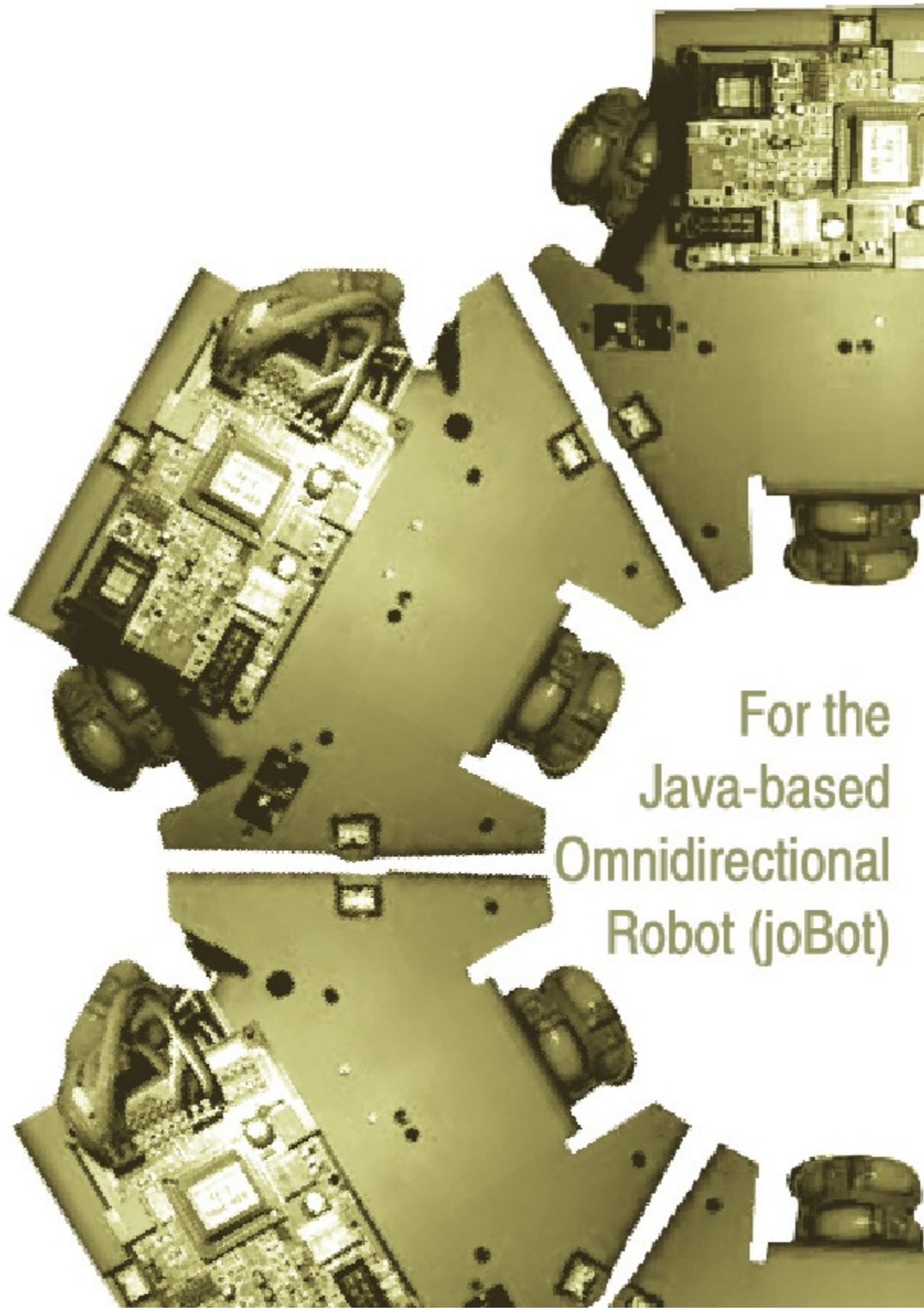




MULTI MOTIONS
interactive animatronics

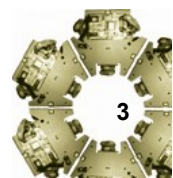
USER MANUAL



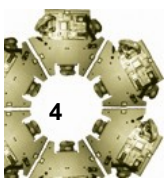
For the
Java-based
Omnidirectional
Robot (joBot)

Table of Contents

Summary of changes.....	5
Chapter 1 The Origins of joBot.....	8
1.1 How the project started.....	8
1.2 Enter muVium.....	8
1.3 Developing joBot.....	9
1.4 About the Simulator and joBot Platform.....	9
Chapter 2 Introduction.....	10
2.1 Goals.....	10
2.2 the muVium software.....	11
2.3 Installing the software.....	11
2.3.1 <i>Installing stand-alone</i>	11
2.3.2 <i>Installing in Eclipse</i>	13
2.4 Testing the software.....	15
2.4.1 <i>Testing the installed simulator</i>	15
2.4.2 <i>Testing the muVium IDE</i>	16
Chapter 3 QuickStart Guide.....	18
3.1 Known issues.....	18
3.2 Before development.....	18
3.3 Running demo programs.....	18
3.3.1 <i>Demo BallLover</i>	19
3.3.2 <i>Demo Maze Follower</i>	19
3.3.3 <i>Demo Multi Agent</i>	20
3.3.4 <i>UVM Demo</i>	20
3.4 Developing for joBot.....	20
3.5 Running a joBot simulation.....	20
3.6 Running an application on joBot.....	21
3.7 Assignments.....	21
Chapter 4 The joBot Simulator.....	22
4.1 Agents.....	22
4.2 Bodies.....	23
4.2.1 <i>Standard joBot</i>	24
4.2.2 <i>LightBot</i>	24
4.2.3 <i>RoboCup Jr joBot or RCJoBot</i>	24
4.2.4 <i>joBot Junior robot</i>	24
Chapter 5 Running the Simulator.....	25
5.1 MenuBar.....	26
5.1.1 <i>Simulation</i>	26
5.1.2 <i>View</i>	27
5.1.3 <i>Insert</i>	28
5.1.4 <i>Output</i>	29
5.1.5 <i>Play / Pause button</i>	29
5.1.6 <i>Speedslider</i>	29
5.2 Speed.....	29
5.3 RobotGUI.....	30
5.3.1 <i>Agent selection</i>	30
5.3.2 <i>Manipulating Agents</i>	31
5.3.3 <i>The Web Service Interface</i>	32
5.3.4 <i>The DIP switches interface</i>	33
5.3.5 <i>Audiograph</i>	33



5.3.6 Mouse Sensor.....	34
5.4 Moving Agents around.....	34
5.4.1 Stopping or inspecting a joBot.....	34
5.5 Other Simulation Environments.....	36
5.5.1 RoboCup Junior fields.....	36
5.5.2 The Rescue field.....	37
5.5.3 The Dance floor.....	38
5.5.4 The Maze field.....	39
5.5.5 Making and following Mazes.....	39
5.5.6 Multi Agent systems.....	39
Chapter 6 joBot Hardware and Software.....	41
6.1 joBot hardware.....	41
6.1.1 JPB Board.....	41
6.1.2 IR Distance Sensors.....	42
6.1.3 Servo motors.....	42
6.1.4 Power Switch.....	42
6.1.5 Communications Port.....	43
6.1.6 Battery holders.....	43
6.2 Using Servos.....	43
6.3 Omnidirectional Drive.....	43
6.4 joBot Software.....	44
6.4.1 Web Service interface.....	44
6.4.2 Drive function.....	45
6.4.3 Vector Drive function.....	46
6.4.4 State Machines.....	47
6.4.5 UVMAgent sleep	47
6.5 Servos.....	48
6.6 DIP switches.....	49
6.7 Sensors.....	49
6.8 LEDs.....	50
6.9 I/O ports.....	50
6.10 Optional extension units.....	51
6.10.1 Display Unit.....	51
6.10.2 Displaying data.....	51
6.10.3 Using the LEDs.....	51
6.10.4 Sounding the beeper.....	51
6.10.5 Using the Remote Control.....	52
6.11 Additional peripherals.....	52
Chapter 7 Example Programs	53
7.1 Available demo programs.....	53
Chapter 8 The Development architecture.....	54
8.1 Compiling and Running the software.....	54
8.1.1 Writing for the muVium architecture.....	54
8.1.2 Checking the muVium documentation.....	55
8.1.3 Creating a new muVium project.....	55
8.1.4 Compiling the program.....	58
8.1.5 Establishing contact with the processor.....	59
8.1.6 Downloading the program to joBot.....	60
8.1.7 Running and testing.....	60
Glossary.....	62
Links.....	66



Summary of changes

This project is the result of several workshops on the 'Body Hopper' project at the University of Amsterdam. The following people did participate and developed most of the initial version of the software for the simulator.

The first version of the Body Hopper software was developed by:

- Markus Heukelom
- Ork de Rooij
- Paul Ruinard
- Edwin v/d Thiel
- Jasper van Woudenberg

The second version that also worked with the robot platform was developed by:

- Tim van Kasteren
- Tim van Oosterhout
- Brammert Ottens
- Gerben de Vries
- Joost Winter

More recent projects have added the Maze facility and the local agents.

The current version, dedicated to joBot and RoboCup Junior has been developed based on the existing software by a team consisting of:

- Bastiaan de Groot
- Seth van der Maas
- Maarten Tromp

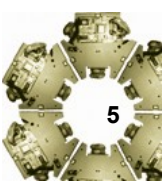
The more recent additions of the joBot Junior robot and the RoboCup Junior playing fields has been done by the following people:

- Louis Fontaine
- Arjan Nieuwenhuizen

The mechanical design of joBot was done by Marcel Binkhorst from Binkhorst Design. The robot was designed with possible applications for RoboCup Jr. in mind, which requires the diameter to be maximum 18 cm. For RoboCup jar however the standard IR sensors may not be used and should be replaced by either bumper sensors or ultrasonic sensors.

The much simpler joBot Junior robot was designed by Wytze Hoekstra as a graduation project while at the InHolland school in Haarlem.

A close cooperation with James Caska from muVium made it possible for the UPB board to become Java aware and it can now be programmed in Java. This allows us to develop software for the joBot inside the Eclipse IDE and then test it on a



simulated version of the robot. When all software works fine, it may be compiled and programmed into the 'real' JPB and the joBot may run your Java program locally.

James at muVium and other people are working hard on a muVium plug-in for Eclipse, so that all development may be also done directly from the Eclipse environment.

This project is also used in teaching a course on Data Structures and Algorithms and the Maze generation and shortest path finding algorithms in this simulator are the main results of that course. The manual is used by the students in that course to get an idea about the problems associated with building software projects that are slightly larger than basic software development courses.

February-March 2006, this project is used as a practicum for the course Software Process of the Master Software Engineering at the Universiteit van Amsterdam. During this six week practicum 36 students, divided in teams of developers, architects, interaction designers and marketeers, did the following:

- Designed and partly implemented a new architecture for the simulator software.
- Designed and implemented an improved user interface for the simulator.
- Provided regression and user tests for part of the simulator software.
- Improved the correlation between the simulator and the real robots.
- Redesigned and implemented the existing RoboCup Jr. website, with the aid of a content management system.
- Redesigned the layout of the existing manuals and updated the content.
- Wrote a marketingplan for the joBot project.

The following students were part of this team:

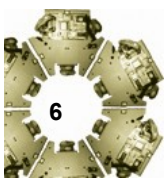
Rianne Aziz
Raymond Backus
Paul Bakker
Menno Bredenoord
Ray Burgemeestre
Jan Derricks
Riaz Gaffar

Bart den Haak
Sebastiaan Herman
Bram van Hoof
Sabrina Jim
Jermaine Jong
Richard Kettelerij
Sannie Kwakman

Reinier L'Abée
Sven Langenhuizen
Arnold Lankamp
Yuka Lutfala
Jacob Ooms
Youri Op 't Roodt
Maarten Pater

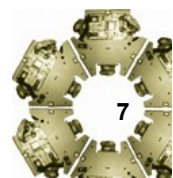
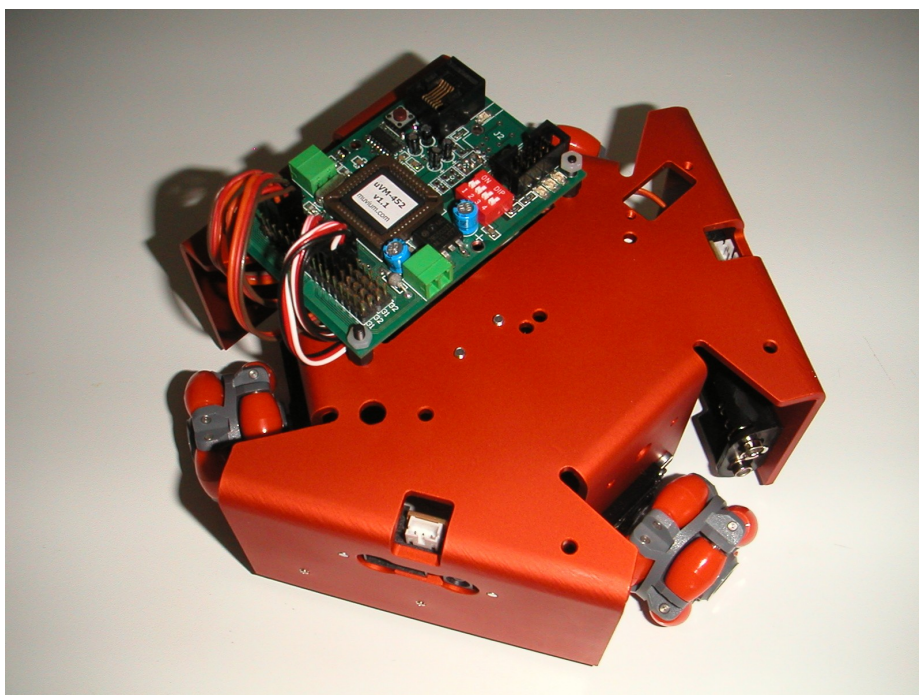
Tim Prijn
Jeroen Quakernaat
Julien Rentrop
Robin Rijnbeek
Paul Speijers
Ron Valkering
Daniel Vranken

Bart Vreeken
Ernout van der Waard
Bas Warmerdam
René Wiegers
Jacco van Willegen
Jonathan Witkamp



Status

This manual reflects a work in progress. The joBot software and associated hardware are constantly under development in student projects and as a result the manual will undergo frequent changes. Be sure to regularly check the MultiMotions site for new versions or consult the source from where you obtained this version of the system.



Chapter 1 The Origins of joBot

1.1 *How the project started*

The Java Omnidirectional Robot (joBot) is a project that found its roots in a number of 'Project AI' classes of the section Intelligent Autonomous Systems of the University of Amsterdam.

During such projects a team of students work together to solve a non-trivial problem in AI, robotics or other intelligent systems. The first project was started as the 'Body Hopper' project in which the aim was to develop an intelligent agent that could live both inside a personal computer and in a dedicated robotic hardware frame.

The agent should be able to travel from a PC to the robot platform and back or even from robot body to robot body. This moving between robot platforms was dubbed 'body hopping'.

During the past years several groups of students participated in the project and developed a simulation framework that allows the development of agents that can inhabit both the virtual world and the real world inside a robot frame. While at first we only had PC based agents, we later developed an agent that also ran on the Palm Pilot Robot Kit (PPRK), which required the availability of a Palm Pilot.

Hopping from platform to platform proved difficult using Java. A dynamic class loader is normally required and only a few JVMs existed that allowed this. The size of such programs was also limited on devices like a Palm. Thus in the first instances of the system, agents were written in a simple language, developed in Forth, and allowed simple interpretation of a number of basic commands. This program was easy to beam to the Palm using standard protocols.

As the project developed, however, the limitations of this approach became clear and the original desire to write the agent in pure Java became stronger.

1.2 *Enter muVium*

The hardware that was used for several of the robots that were developed during this period was all PIC based systems, running on MultiMotions' Universal Processor Bus(UPB) board. James Caska the founder of muVium had been working on a Just In Time (JIT) compiler for Java to allow small Java programs to run on standard PIC controllers.

This proved a much better basis for program development, not in the least because of the commitment of James to developing a tightly coupled system including the muVium compiler and the simulation environment of the virtual breadboard.

So the idea was born to use this approach to develop a simulator that would allow development of Java agents that could run both inside the simulator and on the UPB boards of MultiMotions which was then dubbed the Java Processor Board (JPB).



The current system is the result of this close cooperation and allows students to fully develop, debug and test their agents in a simulation environment and lets the resulting PIC code to be transferred to the robot base using nothing more than a simple serial connection between the PC and the JPB board.

1.3 *Developing joBot*

The omnidirectional drive has become very popular over the past years. Not only did the PPRK have such a drive, more and more robots in the RoboCup tournament are equipped with this way of providing locomotion to robots.

Then Marcel Binkhorst came along. A mechanical engineer with a strong interest in Artificial Intelligence (AI) and robotics was very interested in the omnidirectional drive and designed a new framework that would also allow the robot to be used as a platform for RoboCup Junior.

So from this work and the next generation of AI students the new simulator was developed, allowing the joBot to be used both in a general environment using obstacles or a maze, and in a standard RoboCup Junior environment. This approach allows participants in RoboCup Junior to develop agents and strategies and run them on a PC. When their software seems to work fine, it can be transferred to a joBot of their own design, equipped with additional sensors and actuators and participate in the real competition.

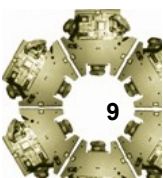
The first series of joBot platforms was built with the help of the engineering department of the AMC, where the workshop did all the metal work on the first batch of joBots. A group of medical information students (MIK) participated in the development of the maze facilities for the simulator.

1.4 *About the Simulator and joBot Platform*

The simulator allows a muVium based Java program to be run inside a simulated environment. Developing for a virtual platform dramatically cuts down the development and debug cycle. Owing to the close integration between the muVium software, the JPB, the joBot platform and the simulator, developing agents (the 'brains' of a robot) or even mobile agents for autonomous robots has become much simpler.

This manual describes all components that are required to develop a fully functional software and hardware agent. It covers the following subjects:

- Basic requirements
- Developing, testing and debugging an agent
- Using the simulator
- Developing an agent for joBot
- Running the agent in joBot
- Installing the required software



Chapter 2 Introduction

This document is a description of the software and hardware architecture of the joBot Omnidirectional Robot based on the Java Processing Board (JPB).

It is a design document and describes the functionality that is currently under development. All available prototypes are being tested with this software. As soon as new versions become available and new insights have developed, this will be included in future versions of this document.

All software is provided free of charge and in its current status and may be downloaded freely from the MultiMotions site. The software is developed in Java using the open source Eclipse IDE.

All JPB software is generated in Java, which will work both at the PC, using a Java test environment and on the JPB.

The main purpose of this project is to provide an environment that permits easy experimentation with real-time autonomous robotics systems in a language that allows speedy development.

2.1 Goals

The various components, developed by MultiMotions to build robot systems, are based on a uniform hardware- and software architecture.

The JPB is part of a family of components, aimed at the development of a variety of autonomous systems. The hardware is based on the Universal Processor Bus architecture, used in MultiMotions UPB and JPB boards. This stackable architecture provides a 32-pin connector, that allows stackable piggy-back boards to be used. The JPB board is a powerful board that, based on the muVium compiler, is able to run a compiled version of Java on a PIC controller.

The software architecture is based on the following goals:

- To allow inexperienced developers, using a standard language and an easy-to-use interface to develop their own robot application.
- To allow experienced developers to set up a running robot control application, while allowing full access to all levels of the system.
- To allow development without having to purchase additional development software.
- To allow the use of standard languages, compilers and debuggers during the development of an autonomous robotics application.

These goals are realized with an interactive development environment that permits the development and debugging of Java software on the PC and the Java based hardware platform, specifically designed for autonomous systems.



2.2 *the muVium software*

The muVium compiler is a special JIT compiler, developed by James Caska of the muVium company. It allows the use of Java on embedded controllers, like the PIC controllers, that are used in MultiMotions JPB boards.

The muVium approach consists of a special version of the PIC controller which contains a bootloader that allows the device to be programmed without the usual development system that is required for other systems.

The procedure to use muVium on the joBot and other JPB boards is generally as follows:

1. Develop a program in Java using Eclipse or any other development environment and compile it into a set of standard Java .class files.
2. Test the program in the simulator, where the joBot may be controlled using Java instructions.
3. When the program is OK, use the muVium IDE and create a new muVium project of the selected .class files.
4. Compile the module using the online muVium compiler. For this access to the Internet is required, since the compiler is an online Web-service that sends the compiled module back.
5. Now program the JPB board using the muVium IDE and the compiled module.

The muVium IDE is provided within the jobotsim.zip file, so it will be installed together with the simulator. More information about muVium and the muVium software can be found at www.muVium.com

2.3 *Installing the software*

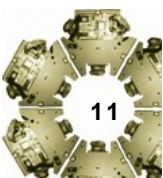
Unpack the JobotSim.zip file from the CD or the www.multimotions.com site into a directory with a name like JobotSim.

You may directly run the simulator from this directory as a stand-alone program or it may be included as a project in the Eclipse IDE and can be run from there. The usual way of running the simulator however is from the Eclipse IDE. This is more convenient because all software development will also be done with Eclipse.

2.3.1 *Installing stand-alone*

The program is a Java program and therefore does not need to be installed as a Windows program. A file joBot.bat is provided that includes all necessary path settings. When installing into a different directory than the default contained in the joBot.bat file, all path references in this file need to be updated accordingly.

In most cases this way of installing is not used and testing is done from the Eclipse environment. This is the most convenient way as described below.



2.3.2 Installing in Eclipse

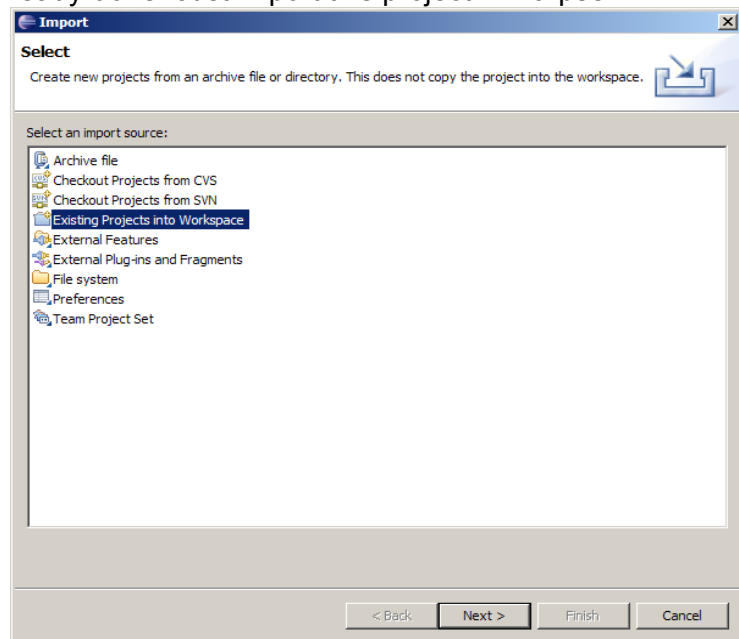
When you plan to modify or extend the simulator you will need to install it as an Eclipse project. When not using Eclipse, use your favorite IDE instead. When using Eclipse, please note the following:

The Eclipse project configuration files are already included. If you use those all configuration of Eclipse is already done. Just import this project in Eclipse:

File > Import...

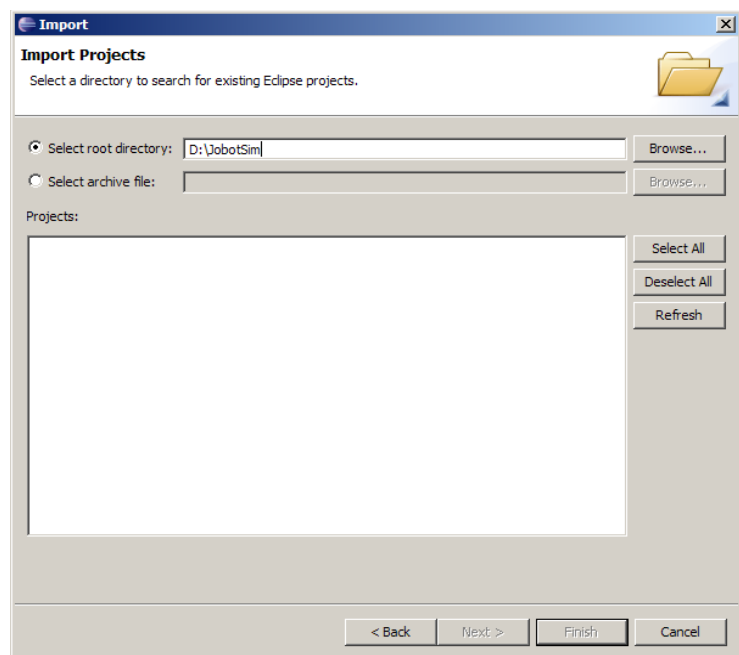
Select: 'Existing Projects into Workspace'

Press: Next



Use the browse button to select the directory in which you installed the software

Press: Finish



If you don't use the predefined Eclipse project, you may define a project yourself in Eclipse or another IDE. You then need to know the following:

- Required libraries:
 - JRE System Library [JDK 1.4.X]
 - comm.jar – Standard sun communications library
 - Junit.jar – Library with Java Unit test software
 - openVBB.jar – muVium simulation software
 - uvm.jar – muVium support software
 - wnonstandard.jar – library for shortest path algorithms to generate a maze
 - wutil.jar – general java utility functions
- The muVium compiler assumes that Java version 1.3 byte code is generated. You will need to set the Java Compiler option `-target 1.3`.
- The muVium compiler assumes that the Java byte code is compiled with the `-g` option to include debug information.
- The class which holds the 'main' method for the simulator is:
javaBot/Simulator

See Eclipse manual for details on how to use this information.

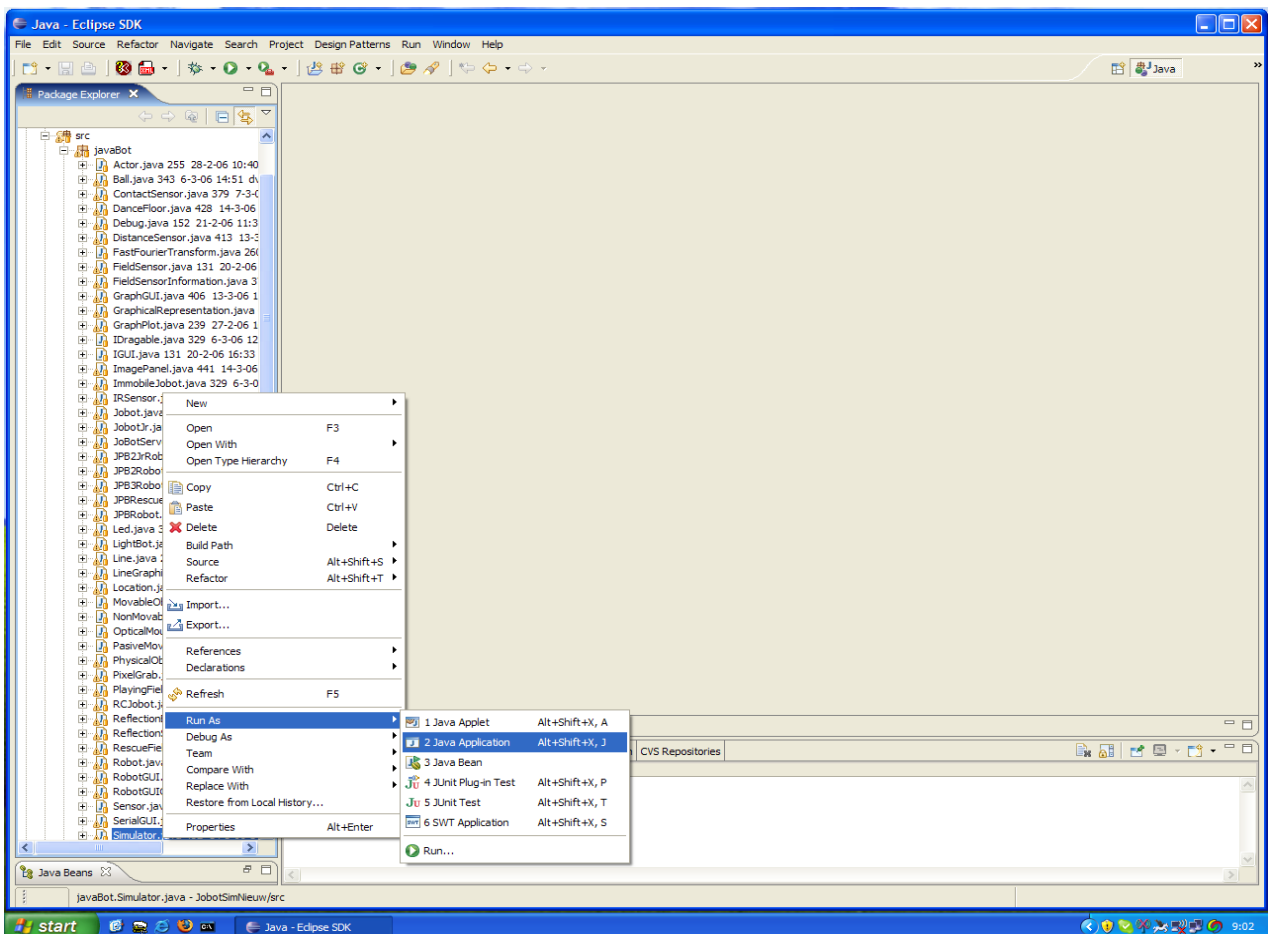


2.4 Testing the software

2.4.1 Testing the installed simulator

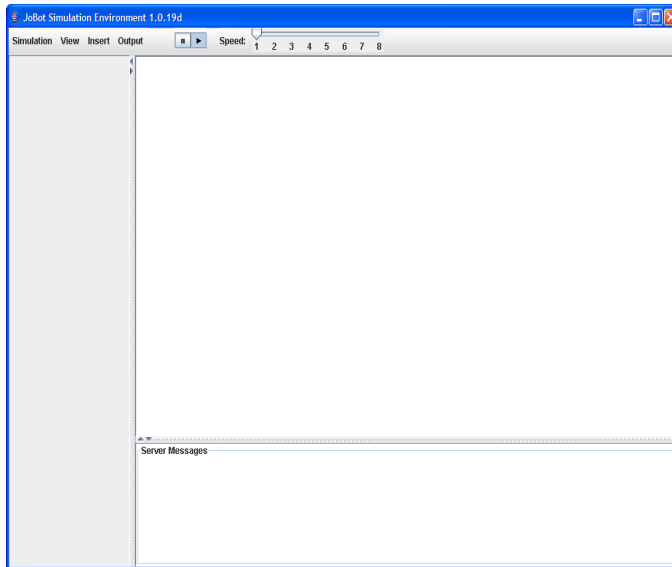
When running the stand-alone version of the simulator, start joBotSim.bat

When running from inside Eclipse, select the joBot project in the Package Explorer. Now select from the 'src' folder the JavaBot package and in it the 'simulator.java' file. Right click on it, and in the Pop-up Menu select 'Run As' and 'Java Application'.



The next time you can directly select the 'Run' button; Eclipse will remember the most recently running program.

When the program starts, the following screen appears:



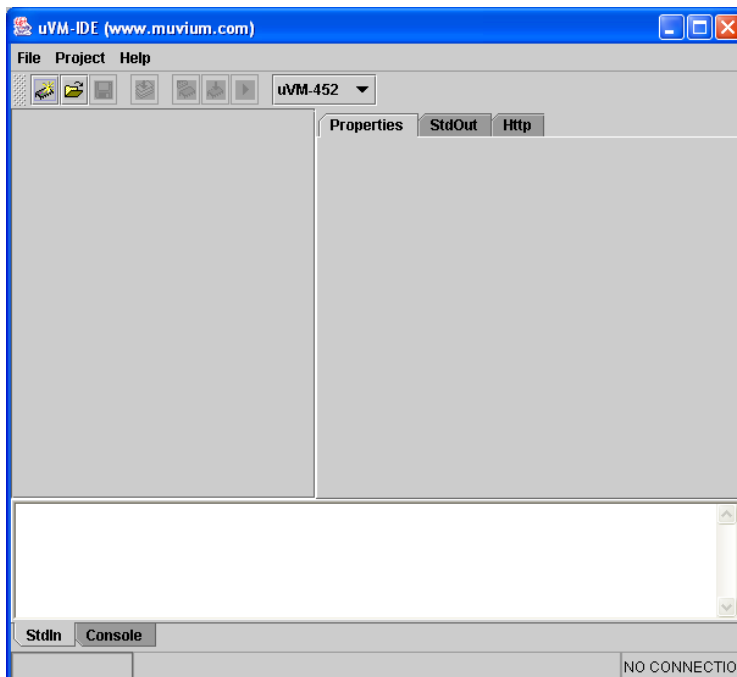
When you see this screen the Virtual World is running successfully.

When this works, the installation was successful and you can start programming the joBot.

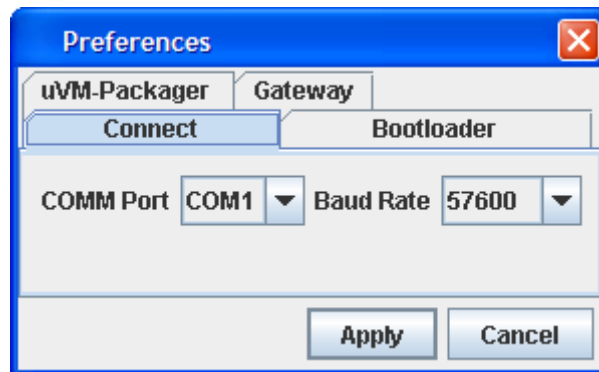
2.4.2 Testing the muVium IDE

You can use the file UVMIDE.bat to start the muVium IDE. It is recommended to place a shortcut to this .bat file on the desktop, so you can start the IDE quickly.

When you start the IDE it will look something like this:



Now select from File > Options... and you will get a small screen if all is well:



This screen allows you to select the COMM port and Baud Rate that you will be using. If you do not see this screen appearing, there is something wrong with the serial interface. There probably are currently no serial ports on your machine

Chapter 3 QuickStart Guide

In this section we briefly describe the steps you will need to go through in order to use joBot and the simulator. Subsequent sections of this manual describe the procedures in more detail. Even more detail about the hardware can be found on the website of MultiMotions: www.multimotions.com

It is assumed that you will be using a Java development environment. In this manual we recommend Eclipse, but any other environment will do.

Please be aware that several of the facilities are still under development.

3.1 Known issues

- Although the simulator lets you change from one simulation to another by changing fields and or inserting and removing objects and robots, this might produce unpredictable results. A better approach is to select Simulation > New from the menu, or start a new instance of the simulator for each simulation.
- Although you can choose both simulator-only agents as UVMAgents for all robots, each type of agent requires its own type of robot. See paragraph 5.1.3 for details.
- Although you can choose all simulator-only agents for all robots, they won't all work as expected. For instance the BallLover agent expects a special IRsensor and Reflection sensor configuration that only the LightBot can provide.

3.2 Before development

Before you can start with the system you need to have installed Java version 1.4.x, a suitable development environment like Eclipse, the muVium software and the joBot simulator.

This may already have been done for you on your machine. If not, you will need to download the software and install and test it. See the installation section on how to do this.

Please be aware that in order to compile and run on the joBot platform, you will need direct access to the Internet, because the muVium compiler is a web-service. If you are only using the simulator the muVium compiler and internet-access is not required.

3.3 Running demo programs

When the joBot simulator has been installed, the included demo programs may be run. Demos show the two types of agents: simulator-only agents and UVMAgents. Agents use the simulator to execute a program for the joBot. UVMAgents will run



both in the simulator and on a real joBot. Whereas simulator-only agents only live in the simulator. See paragraph 2.4.1 on how to start the simulator.

To get an overview of the possibilities perform the steps in the next paragraphs.

3.3.1 Demo BallLover

This agent finds a ball and follows it. When you move the ball around the robot will follow it

1. Start the simulator
2. From the menu select Insert > Robot > LightBot
3. From the menu select Insert > Object > Ball
4. In the RobotGUI select in the first dropdown list['Select a (sim) agent'] the BallLover agent.
5. Move the ball around and see the robot follow, when the ball is in reach of its sensors.

3.3.2 Demo Maze Follower

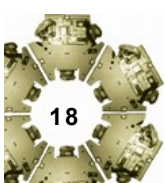
There are two demos for the maze follower. The first one has internal knowledge about the maze and follows the shortest path. This agent could never run as a UVMAgent. The second one does not use world knowledge and has to find its way using the walls. This agent is a so-called wall follower and could also be written as a UVMAgent.

The simulator-only agent knows about the maze and uses this information to follow the shortest path.

1. Start the simulator (Run javaBot.Simulator.java as Java Application).
2. From the menu select Command > Show Maze
3. From the menu select Insert > Scenario > MazePath Follower
4. See how the agent follows the indicated shortest path.

The second agent uses its Reflection sensors to follow the path.

1. Start the simulator (Run javaBot.Simulator.java as Java Application).
2. From the menu select Command > Show Maze
3. From the menu select Insert > Robot > joBot
4. In the RobotGUI select in the first dropdown list['Select a (sim) agent'] the Mazeagent.
5. Watch how unintelligent the joBot is trying to find a way out.



3.3.3 Demo Multi Agent

These agents will flock around balls and follow them around

1. Start the simulator (Run javaBot.Simulator.java as Java Application)
2. From the menu select Insert > Scenario > Balls and Followers
3. Manipulate the balls and see what the joBots do.
4. You may turn off the sensor lines in the Commands menu.

3.3.4 UVMDemo

This is the main demo program in which behaviors for the real and simulated joBot are programmed. This program is described in more detail in the separate UVMDemo manual.

This UVMAgent shows several behaviors, selectable by the DIP switches.

1. Start the simulator (Run javaBot.Simulator.java as Java Application)
2. From the menu select Insert > Object > Ball
3. From the menu select Insert > Robot Demo > UVMDemo
4. At the RobotGUI select DIP switches 1 and 3
5. When blocking one of the Reflection sensors with the ball by dragging it with the mouse, watch how the joBot will get away from the ball

This same demo program is loaded on the standard joBot to experiment with. See the separate UVMDemo manual for detailed information about UVMDemo.

3.4 *Developing for joBot*

Development for joBot is done by writing small Java programs, we call agents, and testing them inside the simulator as shown before. Please note that these programs generally are very small and use a subset of the full Java language. The program is developed with an IDE like Eclipse and compiled into a .class file. This is done automatically by Eclipse when you save your file.

Using the simulator you may now test your Java program and see if the joBot does what you had in mind it would do.

If all goes well, you may then compile and load your program into the flash memory of the joBot's JPB processor board.

3.5 *Running a joBot simulation*

Once you have created your agent, you start the simulator and create a joBot. Then from the GUI select your agent from the second selection box.



The first drop-down list contains all simulator-only agents. (All the classes in the package `javaBot.agents` which are an extension of the `javaBot.agents.JoBotAgent` class.) These agents will only run inside the simulation environment. The second drop-down list contains `UVM`Agents. (All the classes in the package `javaBot.UVM` which are an extension of the `com.muvium.UVMRunnable` class.) Once an agent is selected it will start running inside the simulated `joBot` just as it would run on the real `joBot`.

You keep changing your software until it does what you had in mind. If it works correctly you may load your program into a real `joBot`.

3.6 *Running an application on joBot*

The `joBot` needs to be connected to a serial port on your computer. By default `COM1` is used for this. Details are specified inside the file `joBot.xml`.

You now need to compile the java byte-code into machine-code that will be loaded into the flash memory on the `JPB` board. This is done with the `muVium` IDE as described in chapter 8.

3.7 *Assignments*

There are several assignments that you can take. Use this manual and the `UVMdemo` agent (`javaBot.UVM.UVMDemo.java`) as an example of what you can do. Please note the restrictions on the size of programs and the Java facilities for `muVium`.

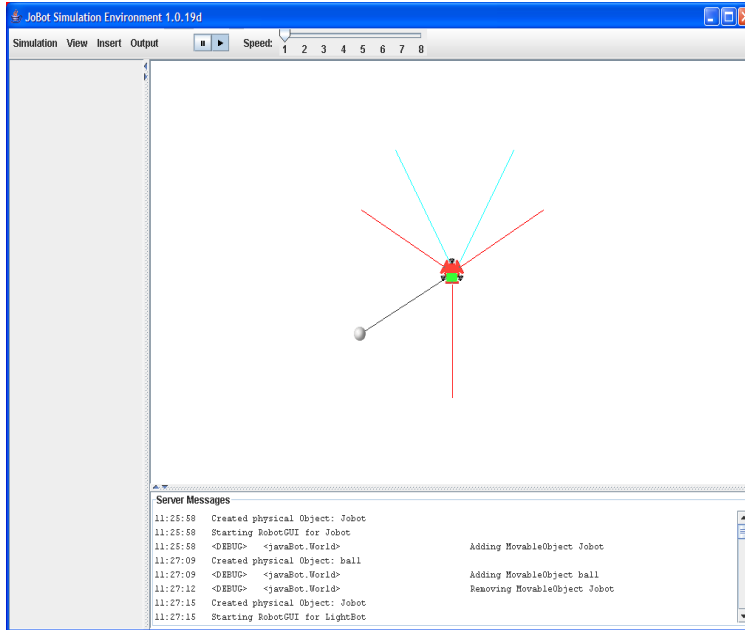
The following assignments may be tried, but you may also design your own application:

1. Take the existing 'FleeBehavior' code and modify it such that the robot does not flee from an object but moves toward it. When this is working, make sure that the robot does not get closer than about 10 cm to the robot and stops there.
2. Make a simple wall follower to find the exit in a maze. The example uses just one Reflection sensor, you may create one that is more intelligent and used all sensors for instance.
3. Follow a moving object or move away from a moving object.
4. Dribble with a ball using just the Reflection sensors of `joBot`.



Chapter 4 The joBot Simulator

The joBot Simulator is the main program. It simulates a JPB based robot and lets you develop and test software for it.



Within the joBot environment we distinguish two basis entities, the agent (brain) and its robot platform (body). Although currently targeted at joBot in principle an agent could inhabit a variety of platforms. Some additional platforms may be developed in the future while users of the system may develop their own platforms as well.

Apart from the simulation environment, the full source code of the simulator is available for those who wish to extend the capabilities of the system. If you do so, please let us know so that others may share the results of your effort and can use it as the basis for further development. All software is open source, free of charge and adaptable by everyone. The software is not permitted to be used in commercial systems.

4.1 Agents

An agent is a robot's brain. It will need a host in which the brain can live. Inside the simulator this is the PC the software is running on. Once running in a joBot, the JPB and muVium controller becomes the host for the agent.

The joBot simulator allows two types of agents:

1. **Simulator-only agents.** Such agents are not designed to run on a robot platform and are depending on the simulator to become alive. They can be



used to develop multi-agent systems that are confined to the world of the joBot simulator.

2. **UVMagents.** These agents can live both inside the simulator and the JPB environment. A simple JPB board is sufficient for the agent to become alive. In principle many applications may be developed with this approach, using a variety of platforms of which joBot is just an instance.

The joBot simulator is targeted to the joBot hardware platform. Other kinds of platforms should be developed by the user if new platforms or features are desired. The source code of the simulator is provided to allow such developments.

There are several examples of both types of agents:

Simulator-only agents:

1. The Maze agent follows a maze inside the simulator. It has knowledge about its world that could not be provided as part of a joBot platform.
2. The Ball followers are a collection of agents that are attracted to balls and exhibit various behaviors. This multi-agent demo shows the interesting interactions that may occur between multiple agents.

UVMagents:

3. The UVMDemo agent displays a number of different behaviors that can be copied into your own agent.
4. The Wall follower may run inside the simulator, following a maze without having any explicit knowledge of the world. This allows this agent to perform the same behavior on a joBot platform.

4.2 Bodies

The body of an agent allows it to perform in the real world and to use real-world sensors to sense and act.

The joBot platform is equipped with a basic JPB board that allows modest Java programs to be running inside it. Because of the limitations of the PIC controller's size and speed not all Java constructions are supported on this platform. The hardware restrictions are described in chapter 8 "The development architecture". Other platforms may be developed with the provided simulation environment, as long as the muVium architecture is supported. Several other systems have been built with the same architecture and based on the simulation environment.

In this manual we concentrate on the joBot platform in its basic form. There are two versions of joBot. The red version is the basic version; the black version is the special RoboCup version which has more sensors and a different drive system than the standard version.



4.2.1 Standard joBot



The standard joBot comes with a muVium 452 processor, three servo motors and three Reflection sensors and one IRsensor. The JPB board contains a standard set of red, yellow and green LEDs and a 4 position DIP switch. Communication with the robot is done through a standard RS232 link.

4.2.2 LightBot

LightBot is a special version of the standard joBot, equipped with three IRsensors and one Reflection sensor.

4.2.3 RoboCup Jr joBot or RCJoBot



The Black version of joBot is developed specifically as a kit for the RoboCup Junior competition. In addition to all features of the basic version, it supports other sensors and actuators.

Please note that the black version is currently still under development and can only be used inside the simulation environment.

It may additionally be equipped with:

1. Faster motors that allow swift movement.
2. Optionally a muVium 6720 or muVium 6722 controller, allowing for much larger programs.
3. Ultrasonic distance sensors instead of the standard Reflection sensors.
RoboCup Jr robots may not emit IR light.
4. IRsensor and IR floor position sensor.
5. Electronic compass.
6. Wireless serial connection.



4.2.4 joBot Junior robot



The joBot Junior version is a much simpler design that does not use the omnidirectional drive, but instead uses a two-wheel differential drive engine. It is specifically designed with RoboCup Junior and it's three different games in mind.

This robot is developed specifically as a kit for the RoboCup Junior competition. In addition to all features of the basic version, it supports other sensors and actuators. It is designed as a simple and inexpensive kit for schools.

It may additionally be equipped with:

1. Faster motors that allow swift movement.
2. Optionally a muVium 6720 or muVium 6722 controller, allowing for much larger programs.
3. RoboCup Dribbler.
4. Electronic compass.

Chapter 5 Running the Simulator

The simulator is provided as a separate program. From the directory from where you installed the system, select the joBotSim program or run javaBot.simulator from within Eclipse. See the installation section on how to install the system.

When the system starts, you will see the world and a number of menus in it where you may select a number of functions. Standard the world will simulate an area of 500 x 250 centimeters.

After starting the simulator you may choose from six different environments that are selected by menu options:

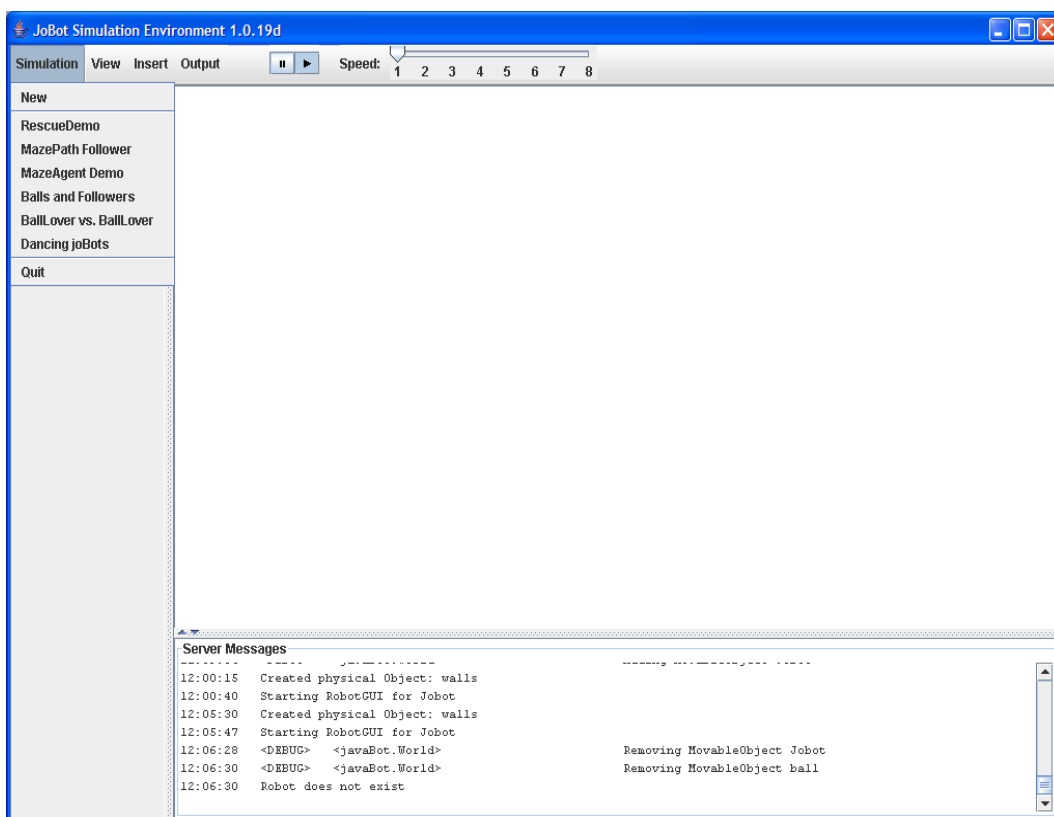
Standard	This is the standard interface showing a 20 x 10 grid of 25 cm squares
Single	This is the RoboCup Junior one-on-one environment of 119 x 87 cm.
Double	This is the RCJr two-on-two environment of 183 x 122 cm.
Rescue	This is showing the standard RoboCup Junior Rescue playing field of 120 x 100 cm .
Dance	This is showing a blank field, representing the dance area with the outer field borders. The real dance area is 10 x 5 meters.
Maze	This is showing a maze and the calculated shortest path trough the maze.

See chapter 5.5 for more detail about the various environments.

5.1 MenuBar

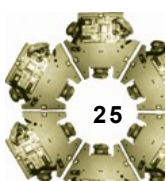
The simulator has four menu entries: Simulation, View, Insert and Output. Also there are two commands provided at the menubar. A pause /start button to stop the simulation temporarily and a slider with which you can slow down or speed up the simulation.

5.1.1 Simulation



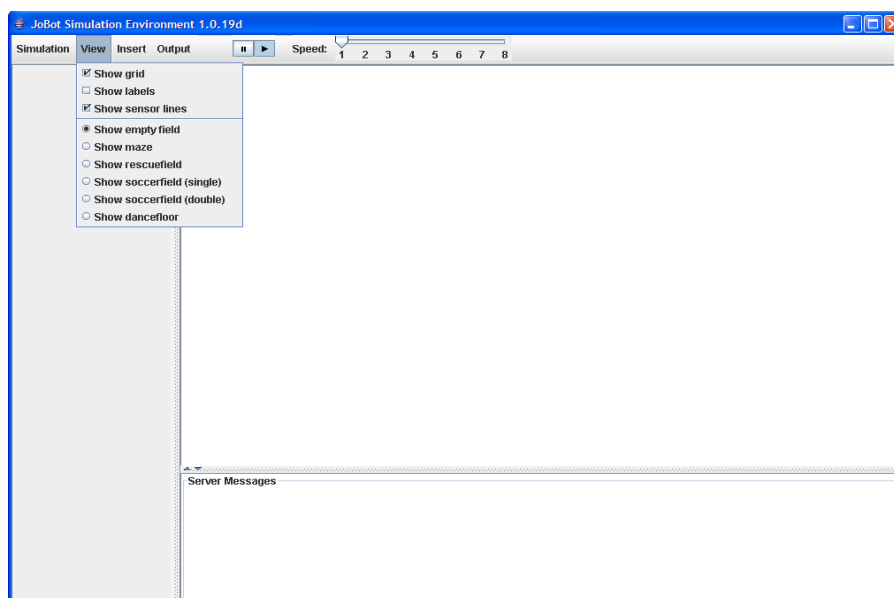
The Simulation menu is provided to start a new simulation or to show some predefined example simulations:

- **New** – Starts a new simulations with an empty standard field.
- **Rescue Demo** – A demo of a joBot that performs a rescue operation.
- **MazePathFollower** – Is the demo maze follower that uses internal knowledge to follow the shortest path.
- **Maze Agent Demo** – A demo agent that uses its Reflection sensors to find a way through the maze.
- **Ball and Followers** – A multi-agent demo that uses software agents only.
- **BallLover vs. BallLover** – A demo of two joBots trying to play soccer.



- **Dancing joBots** – Four joBots dancing on the dance area.
- **Quit** – Stops the simulator

5.1.2 View



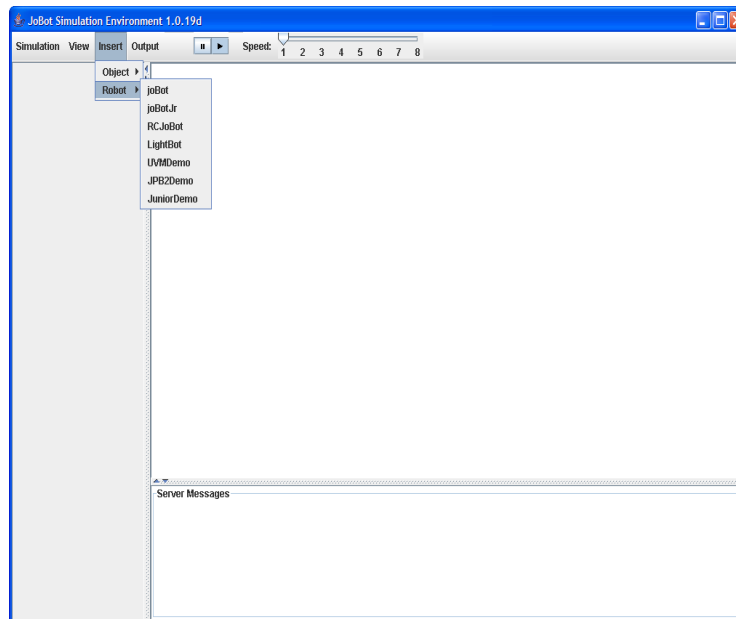
The View menu lets you change what is to be seen in the main panel of the simulator.

- **Show Grid** – Shows the grid on the background. Each grid field represents a distance of 25 cm. A joBot is 18 cm in diameter.
- **Show Labels** – Turns the name of the joBot on inside the simulation field.
- **Show Sensor Lines** – Displays the sensor lines. This is helpful in determining the field in which the sensors are active. Some sensors only show their field in the presence of an object. For instance the IR sensors are only shown when there is a ball in the field.

You can also choose from five possible backgrounds:

- **Show empty field** – This displays an empty field.
- **Show maze** – Generates and displays a maze and the calculated shortest path. This path is accessible to a joBot and may be used to follow the path. The dimensions of the maze depends on the dimensions of the last field shown.
- **Show soccerfield (single)**– This displays the gradient that is used by the joBot to determine its position.
- **Show soccerfield (double)**– This displays the gradient that is used by the joBot to determine its position.
- **Show rescue** – Shows the standard rescue field
- **Show dancefloor** – This shows the lines which form the boundaries of the dance area. In the simulator those lines are black. In the real world those lines are black and white. So if your agent is able to distinguish black lines, it will work properly both in the simulator as in the real world.

5.1.3 Insert



The Insert menu lets you insert objects and robots into the world:

The objects and robots are generated either in the center of the field or at a random location, depending on the type. For each robot a RobotGUI is displayed and from that several actions may be performed.

All objects and robots can be moved by dragging them with the mouse and rotated by dragging them while holding the ctrl-key.

- **Objects**

- **Ball** – Creates a ball.
- **Wall** – Creates a wall.
- **Victim** – Creates a Victim which can be used with the rescue game.

- **Robots**

- **joBot** – Creates a standard joBot.
- **joBot Jr** – Creates a two-wheeled joBot Junior robot.
- **RCjoBot** – Creates a black RoboCup Junior joBot. There is currently no hardware version available of this robot.
- **LightBot** – Creates a LightBot
- **UVM Demo** – A joBot with a UVM Agent.
- **JPB2 Demo** – A JPB2 joBot with a UVM Agent
- **Junior Demo** – A joBot Jr with a UVM Agent.

The first four robots can be equipped with a simulator-only agent via the RobotGUI. The latter three robots can be equipped with a (real) UVM Agent. For now this is standard the javaBot.UVM.UVMDemo class. This robots you can manipulate via the commandline and the DIP switches of the RobotGUI, but the results of selecting a new agent may be unpredictable. See the description of the RobotGUI at paragraph 5.3 for details.



5.1.4 Output

The Output menu lets you control the Server Messages window at the bottom of the screen. The simulator has three successive levels of messages from the Output menu you may choose which level you want to show in the Server Messages window, or you can turn all message showing off. By default the Server Messages window scrolls automatically, thus showing you the latest messages. Via the Output menu you can turn autoscrolling off. You can use this option when messages in a loop are presented to fast to be read.

5.1.5 Play / Pause button

You can use the Play / Pause button to stop and restart the simulation.

5.1.6 Speedslider

The speedslider can be used to speed up or slow down the robots. See the next paragraph for how this speed is determined and how this can affect the behaviour of an agent.

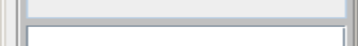

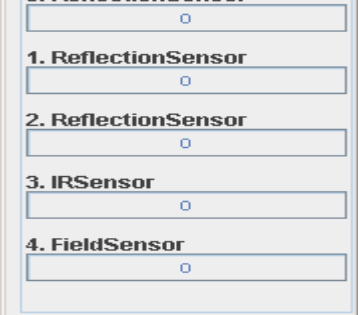
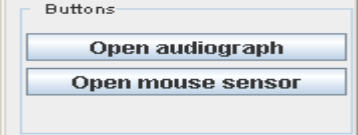
5.2 Speed

On a real robot the frequency of execution of the behaviour of a robot depends on the internal clock of the muVium controller. In the simulator this internal clock is simulated by a Timer provided by the Java GUI for the simulator-only agents and by a timer provided by muVium.openvbb for the UVMAgents. The latter is less influenced by other programs running on the PC and therefore it better resembles the real PIC clock. These timers are not affected by the speedslider. Therefore a agent may appear to not 'seeing' a ball, or running through a wall, when you use a speed setting higher then 1. To overcome this problem in UVMAgents, you have to increase the value of the `servicePeriod` parameter when calling the constructor of a `Behaviour` subclass. Also if, in a simulator-only agent, you use time to determine the position, as the example `MazeAgent` does, the output will not be the same for each setting of the speed slider.



5.3 RobotGUI

Each robot in the simulator has a RobotGUI from which sensor and actuator information can be read, the DIP switches can be set and that simulates the Web Service interface that is used for the additional RC unit. You can also use the RobotGUI to select an agent for the robot. The RobotGUI is opened when the robot is first created and you can reopen it with a context-menu option accessible by right-clicking the robot. The RobotGUI can be docked in the left panel of the simulator, or opened as a separate window. You can toggle between the two through the context-menu of the robot. Also two buttons are provided for opening separate windows for inspecting the sound sensor and the mouse sensor.

Agent selection	
DIP switches	
DIP switch readings	
Web Interface StatusLine	
Web Interface CommandLine	
LEDs	
Sensors	
servos	
Buttons	

5.3.1 Agent selection

The first thing that needs to be done is to select an agent type that needs to be associated with the joBot. When an agent is already running inside joBot it will be stopped and replaced by a new agent. For the joBot, joBot Jr., RCJoBot and Light

There are two agent types that may be selected: a simulator-only agent or an UVMAgent. Only the latter can be used on a real robot. The RobotGUI has two selection boxes, one for each type. The selection is done based on the .class files that are available in the system directory for the simulator. All the classes in the package javaBot.agents which are an extension of the javaBot.agents.JoBotAgent class appear in the first dropdownlist. These are the simulator-only agents which will only run inside the simulation environment. The second drop-down list contains all classes in the package javaBot.UVM which are an extension of the com.muvium.UVMRunnable class. The so-called UVMAgents which can run in the simulator as well as on the real robot.

For the joBot, joBot Jr., RCJoBot and LightBot robots, you can choose from the Simulator-only Agents provided in the upper selectionbox. These agents are not affected by the DIP switch settings and cannot be addressed by the Web Service Interface.

Every Agent is written for a specific robot, so not all the agents will work in every robot. For instance a standard joBot does not have the sensor configuration needed for the ballLover agent, so that agent will fail in that robot. This failure is shown in the messages in the Server Messages Panel.

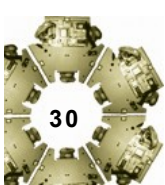
When a joBot has been created, it will be shown as a Dummy. After an agent type has been selected, the joBot will receive the name of the selected agent. When multiple agents are created, each will be given a number in addition to its name. As soon as an agent is selected, it will start performing and the simulator will show the action of the agent. The LEDs will blink if there is code in the agent that will activate this and the joBot will start moving or reacting to stimuli.

Depending on the type of agent selected, any previous agent will be removed and the new agent will be started.

A number of demo simulator-only agents are provided that demonstrate the various facilities of an agent that runs only in the simulator:

- **BallLover** – Is an agent that is attracted to a ball. It will find a ball and then move towards it.
- **MazeAgent** – Follows a maze when one is given, using only its sensors to find a way out.
- **DanceAgent** – Makes random movements.

For the UVMDemo, JPB2Demo and JuniorDemo robots you can choose an agent from the lower selectionbox. Currently this will always be javaBot.UVM.UVMDemo.class, the same one that is selected for this robots at time



of creation. In effect there is currently no selection possible for these robots. See the separate UVMDemo manual for more detail about this class.

5.3.2 Manipulating Agents

When an agent has been created, the combination of joBot and agent will start performing. Depending on the type of agent, actions will be executed and the simulator will display the effects on the screen. The simulated LEDs will light up in the RobotGUI and the values of all sensors and actuators are displayed continuously.

There are a number of ways in which the agent in the simulator can now be manipulated and be made to react to events or actions from the user.

First of all another object can be brought into the simulator like for instance a ball or another joBot. When a sensor detects an object, the simulator will show the value in the RobotGUI and the agent will react to this changed sensor value, if programmed to do so.

The DIP switches on the robot GUI can be used to signal to an UVMAgent that some action is required. It is entirely up to the programmer to define how these switches will be used. Each switch can be used independently to perform a certain action or the total value of the switches can be used as an indicator. The UVMdemo shows an example of the use of the DIP switches and use it to select one of several different behaviors.

Additionally, commands may be sent to the agent using the Web Service interface commandLine.

5.3.3 The Web Service Interface

Every UVMAgent is required to implement the muVium Web Service interface. With this interface simple commands can be sent to the agent and it can be used to retrieve data from the agent.

Commands are typed into the commandLine of the RobotGUI in the form of a command code and a maximum of 3 numeric parameters. This information is sent to the agent and it will respond with appropriate actions.

Currently the following commands are provided:

1. **Drive x y z** – Sends data to the X, Y and Z servo values and activates these. This allows the servos to be set by a command. Make sure the agent is not overwriting this information so a mode must be selected in which the agent is silent. In the joBot Junior version the Z motor is used to drive the (optional) dribbler.
2. **VectorDrive x y r** – Sends the X and Y direction of travel to the agent and the Rotation of the body. Once the command is received it is executed.
3. **Sensor x** – Returns the current value of sensor X. The value is shown in the StatusLine above the commandLine.



4. **Action x** – Sends an action command to the agent. This is a single byte code that may be interpreted by the agent in any way the programmer wants.
5. **Start** – Restarts execution of the program after issuing a stop command.
6. **Stop** – Stops execution of the program.

These commands are only executed by an UVMagent when the proper Web Service interface has been implemented. See the demo programs for examples on how to implement these features.

For more information about the Drive and VectorDrive settings see the “Hardware” section.

5.3.4 The DIP switches interface

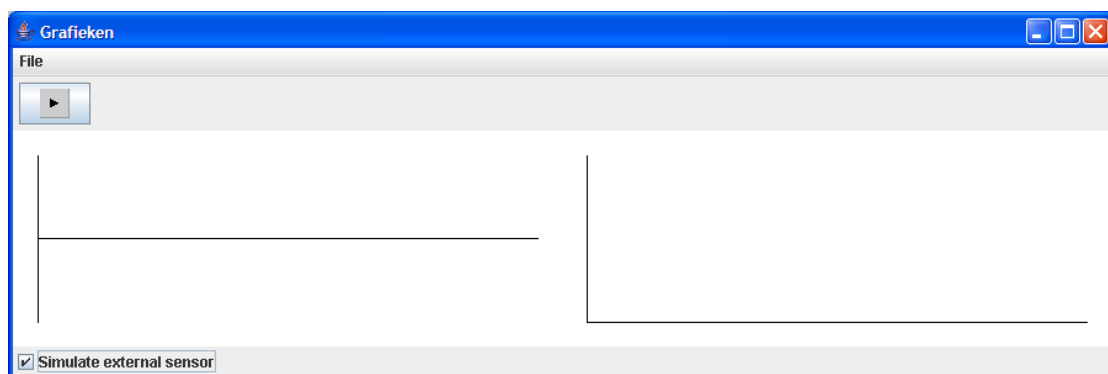
The JPB board has a DIP switch that may be read by the program. The value of the switch consists of a nibble (4 bits) and is used for mode settings or other facilities.

In the simulator, the position of the DIP switches is set by the four selection boxes above the commandLine.

The switches are numbered from left to right as 1,2,3,4. Any combination may be made, where the value of the switches is read with 1 being the low order bit and 4 the high order bit.

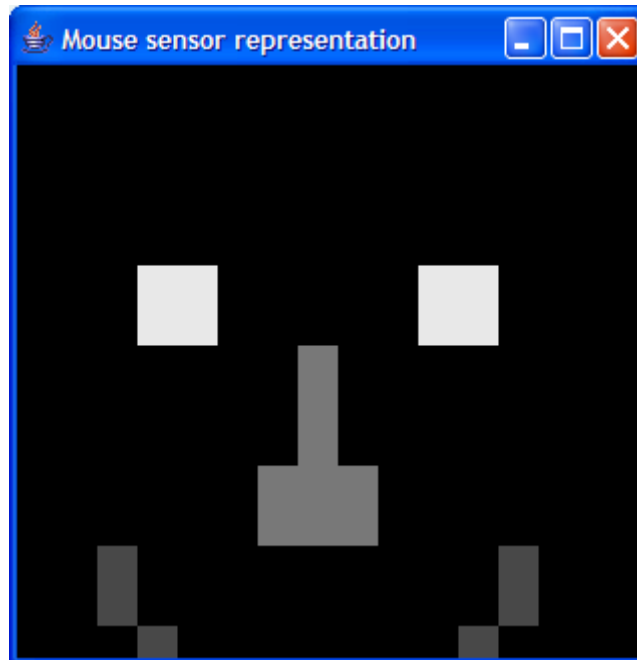
The decimal value represented can be read below the switches.

5.3.5 Audiograph



For inspection of the additional soundsensor a separate window can be opened from the RobotGUI. See sensor documentation on how to connect a real sensor to your computer. Alternatively you can simulate sounds from a file with a button provided in the audiograph.

5.3.6 Mouse Sensor



The Mouse Sensor window is used for:

5.4 Moving Agents around

joBots will move around under control of their agent. Sometimes, however it is desirable to place the agent at a certain location or make it face a certain direction. This is like picking up the real joBot and placing it somewhere on the floor.

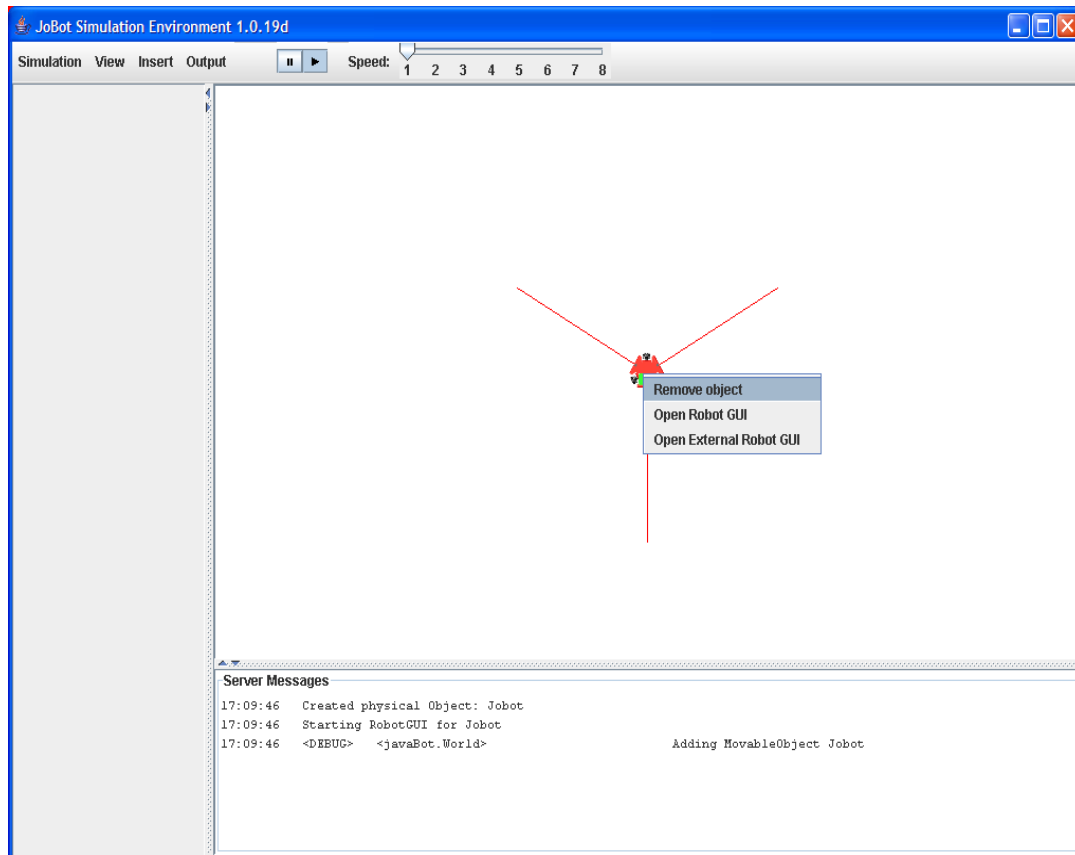
By selecting a joBot or ball or other object with the left mouse button, you may drag it to any position in the field. With a ball, notice that when a certain speed is given to the ball it will continue in the given direction when released.

When the ctrl-key is pressed, while pressing the left mouse button, the joBot may be rotated along its axis, thus making it face a certain direction.

Please be aware that any actions from the agent may interfere with this, since new movement commands may continue to be given by the agent. If this is undesirable, a function should be provided in the agent to stop all action temporarily.

5.4.1 Stopping or inspecting a joBot

When pressing the right mouse button while on an object, a pop-up window appears in which a selection can be made between removing the object or to show the robot's RobotGUI on the screen.



The pop-up menu shows three options, Remove Object, Open Robot GUI and Open External Robot GUI.

5.5 Other Simulation Environments

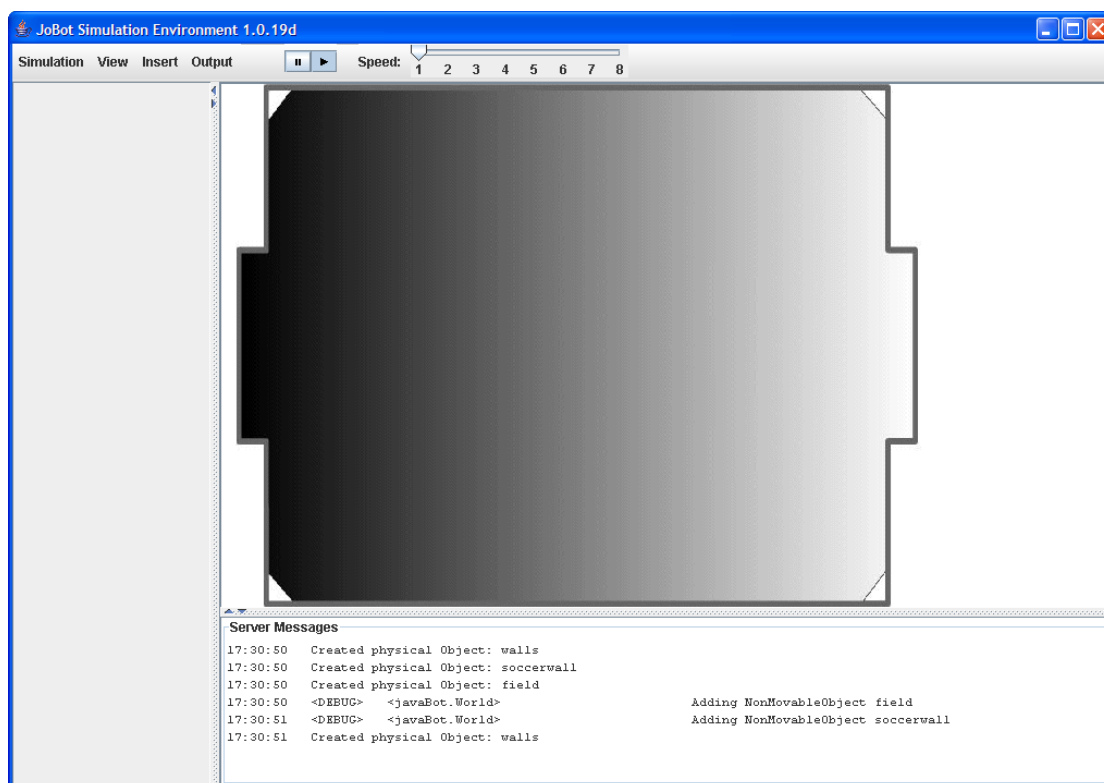
Up till now we have mainly discussed the default environment. There are however five additional environments that a joBot may be placed in:

The one-on-one RoboCup Junior field, the two-on-two field, the Rescue field, the Dance floor and the Maze field.

The official sizes of the field are scaled to fit the height of the mainpane of the simulator. The robots and other objects are scaled accordingly. The Maze uses the dimensions of the field last shown. If a field's width will not fit in the window, then it will be cut of at the right.

5.5.1 RoboCup Junior fields

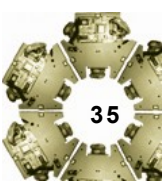
When selecting the SoccerField, the gradient appears that is used by the RCjoBot IR floor position sensor to determine it's position on the field.



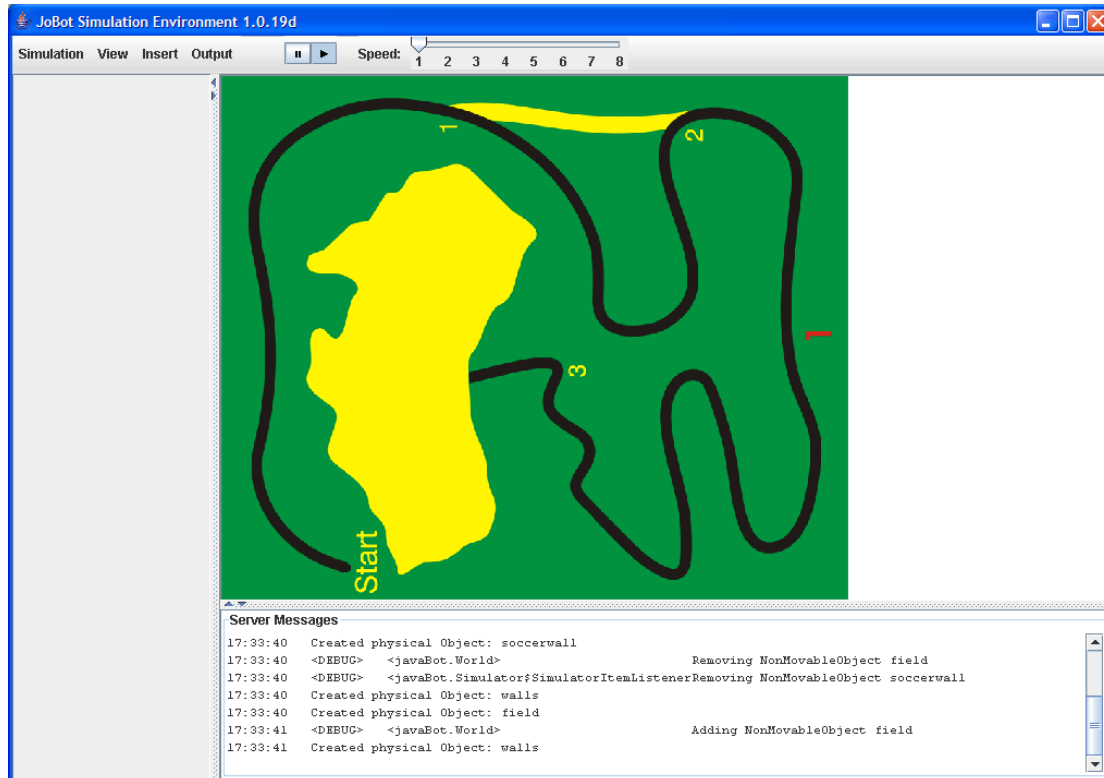
There are two field sizes, based on the RoboCup rules.

The one-by-one field is 119 x 87 cm.

The two-by-two field is 183 x 122 cm.

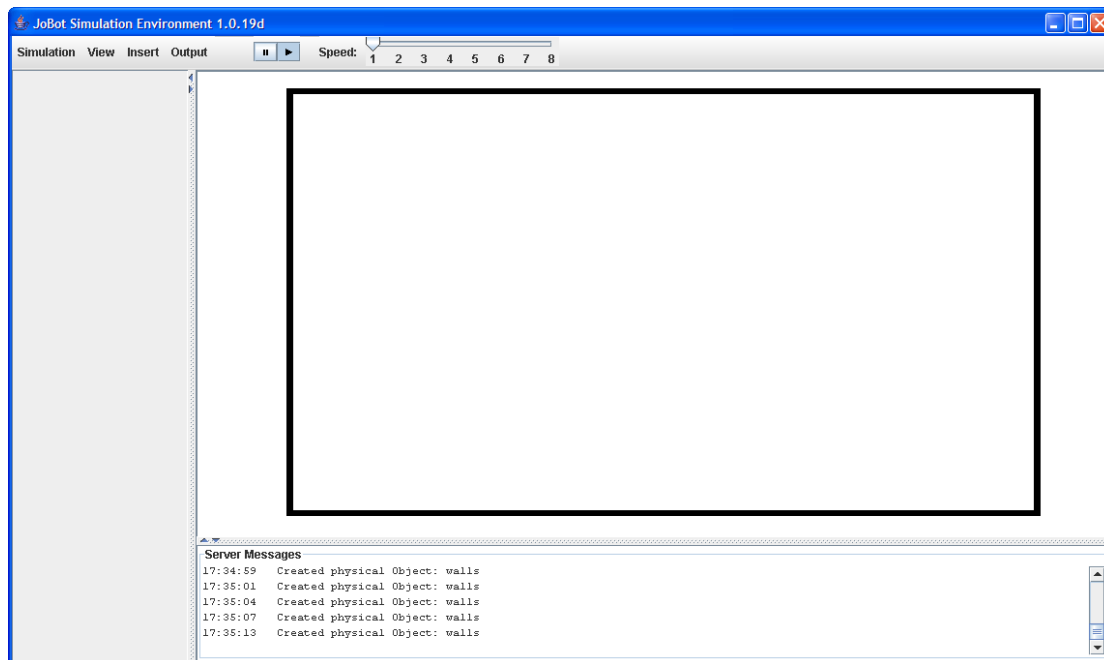


5.5.2 The Rescue field



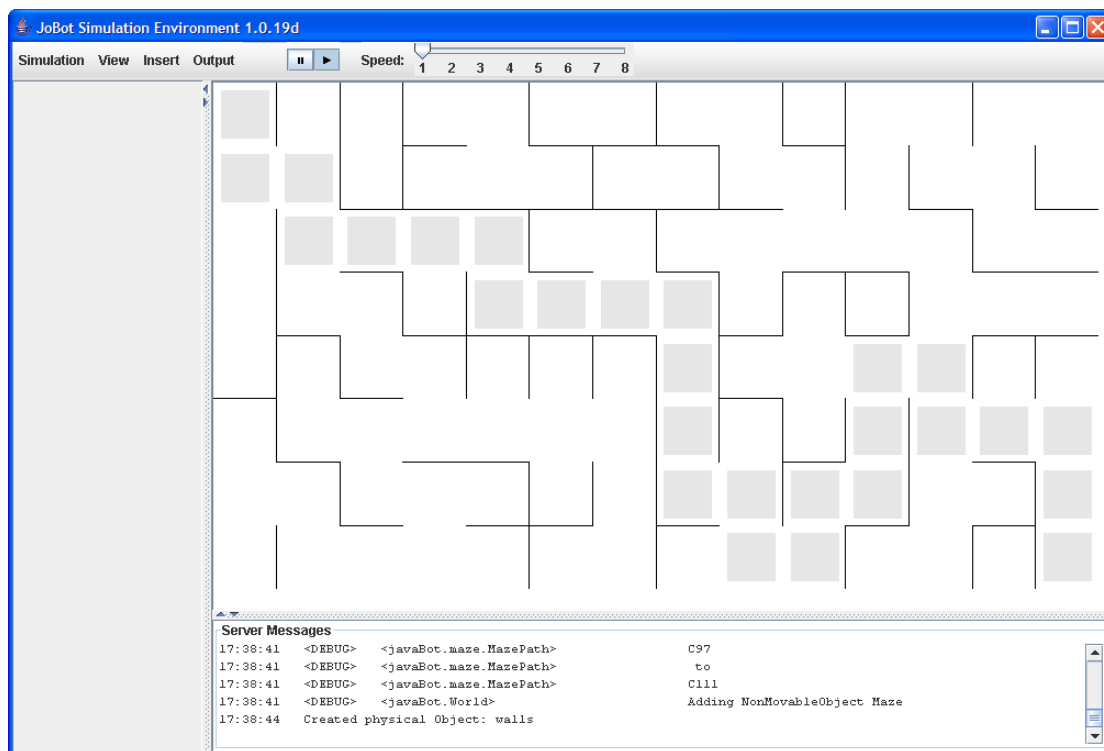
The rescue field provides a background for the Robocup junior rescue game. The size of this field is 120 x 100 cm. The purpose of this game is that a robot follows the black or yellow track and then pushes a victim out of the yellow field. Either straight or by following the track back again.

5.5.3 The Dance floor



On the Dancefloor there is a 10 x 5 meter square of black lines which can be detected by a Fieldsensor. According to the rules, the robots may not pass this lines.

5.5.4 The Maze field



When selecting a maze, a new randomized maze is shown and the simulator uses the walls to calculate the signals that are reflected from these walls.

5.5.5 Making and following Mazes

When the Show Maze command is selected, a maze is generated and shown. An internal structure is generated that keeps track of all the walls. These walls may then be used by the joBot's distance sensors to see if there is free space to move to. The default behavior of the joBot is to scan for obstacles and move away from them, just as the real robot would do too.

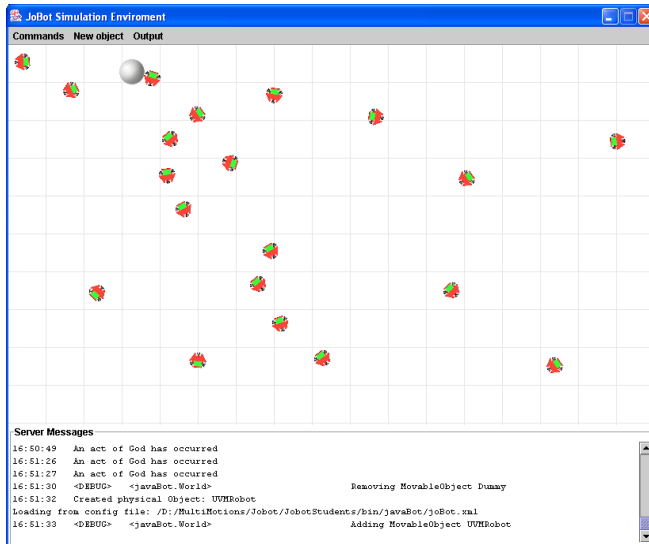
The 'Room' is divided into 20x10 Cells. Each cell is 25x25 cm. Each cell is surrounded by four walls and the joBot can 'see' these with its distance sensors. When an opening is detected the joBot may go through it.

5.5.6 Multi Agent systems

Instead of having one, two or four joBots operating in a field, many similar or different agents may act inside the simulated world. The multi-agent demo shows this capability.

In this particular case many ball-loving agents are created along with some balls and the world simulates the interactions between agents, balls and other agents.

The agent code, as before determines the behavior. Many interesting systems can be built with the simulator.

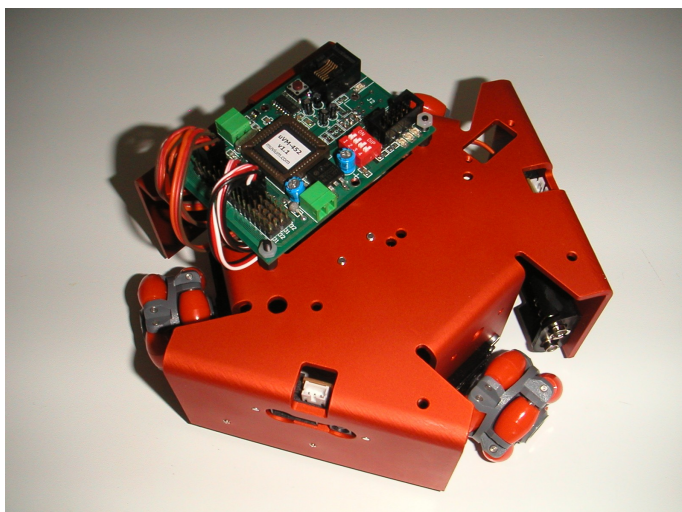


Chapter 6 joBot Hardware and Software

6.1 joBot hardware



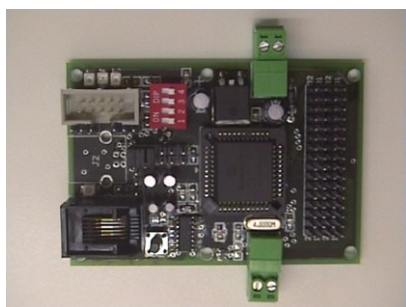
The joBot consists of a metal frame, motors, sensors, batteries and a JPB board. Each of these components will be described briefly underneath:



6.1.1 JPB Board

The JPB board consists of a muVium 452 controller, the JPB bus connectors, power connectors, a DIP switch and reset button.

The COM port is needed to communicate with joBot online and is required during programming. The processor may run in BOOTMODE or in execution mode. BOOTMODE is used to program the device. In all other situations the unit runs in execution mode.



Pressing the RESET button will restart the program. When being programmed, the RESET button is used to turn the processor into BOOTMODE.

The DIP switches are used by the program running in the processor to enter settings or to put the joBot into a different mode of operation.

6.1.2 IR Reflection Sensors

The IR Reflection sensors detect objects at a distance of up to 80 cm. Very close to the joBot the distance cannot be measured accurately. The distance is measured by triangulation, measuring the angle of reflection of a focused IR beam. This process makes the sensor virtually insensitive to colors, allowing a reliable distance estimation of an object.



Because the beam of the sensor is very narrow, you need to take this into account when writing software for joBot.

The distance is returned as an analog voltage level that is converted into a digital value by the internal AD converters.

The values run approximately from 0 to 128, resulting in a resolution of about 5 mm.

Be aware that at a distance of about 5 cm and less the resulting values are becoming unreliable.

6.1.3 Servo motors



The servo motors of joBot are modified RC servos that take a PWM signal as input. The pulse width determines the speed and direction of rotation. At speed 0 the servo stops or will turn very slightly, depending on the calibration of the servo.

At speed 100 it will run forward at maximum speed, at -100 it will turn backward at maximum speed.

Because the pulse width is generated under software control, it is not as accurate as when a hardware timer would be used. However since we need at least three servos on joBot the two built-in timers of the PIC controller are not sufficient to generate the required signals.

The result is that drift can occur and an exact zero position may sometimes not be realized. This causes very slow creeping movement of joBot.

6.1.4 Power Switch

The power switch to the right of the JPB board has three positions. The middle position is OFF. The backward position (towards the JPB board) is the programming position in which just the processor is powered but not the servos and sensors. The forward position powers the servos, the sensors and the processor. By switching joBot in the backward position all processor functions are turned on, without joBot being able to move. This is also useful for testing.

6.1.5 Communications Port

The communications port is used to program the device and is also used to transmit data between joBot and another computer. When using a separate interface, this

may also be done using a wireless connection such as an IrDA interface, a bluetooth interface or a wireless LAN interface.

Port speed and the port number are detected automatically on the JPB board but must be specified on the PC. This is done in the jobot.xml. When using the UVM IDE a separate options panel is provided to supply this information.

6.1.6 Battery holders

Power is provided by two different sets of batteries. A 9V battery provides power to the processor and two sets with each two 1.2V batteries provide power for the servos and sensors.

All batteries are rechargeable and can be replaced. Because of the construction of joBot the four AA batteries may be a bit more difficult to replace than the 9V battery.

A small tool is provided in the form of a shoehorn to aid in the replacement of the batteries closest to the servos.

6.2 Using Servos

The JPB and muVium architecture support the control of standard RC servos. These servos are controlled by sending a pulse width modulated (PWM) signal, where the width of the pulse determines the servo position.

In the joBot the servos are modified for continuous rotation thus controlling both speed and direction.

The muVium architecture supports two ways of controlling servos: PWM and Direct.

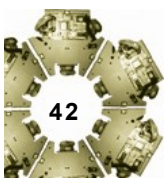
The **PWM** method has the highest resolution and uses the PIC pwm registers directly. Therefore only two servos may be controlled by this method.

The **Direct** method uses a software timer and therefore has a much lower resolution. Resolution depends on processor speed and the number of servos controlled. The JPB supports up to 8 servos and by targeting the ADC inputs for digital operation an additional 8 servos may be controlled. Making a total of max 16 servos. Of course each sensor needs a ADC input.

In joBot the direct method is used exclusively since at least three motors need to be controlled to allow the omnidirectional drive to control the robot.

6.3 Omnidirectional Drive

joBot is a Java Based Omnidirectional Robot, but what does that mean, omnidirectional? Omnidirectional means in all directions. Because of the special omnidirectional wheels that cannot only rotate like a normal wheel but have no friction in the sideways direction, driving three such wheels allows the robot to move in any direction.





The nice thing about this movement is that there is no need for the robot to have a 'front' since it can move in any direction.

Therefore the robot is equipped with three motors that can be controlled in a number of combinations to allow many different movements.

By selecting a combination of speed and direction from the three motors, the robot can be driven in any desired direction while at the same time controlling speed and

angle of rotation.

The joBot software contains two functions that allow the motors to be driven, Drive and vectorDrive.

Drive controls the speed and direction of every motor by a single parameter for each motor. Motors are numbered counterclockwise, starting with the front wheel, i.e. the wheel opposite the JPB board.

The vectorDrive command specifies three parameters also but here they mean the X and Y vector for the robot's direction and a third parameter that indicates the rotation angle.

6.4 joBot Software

In order to run software on joBot, the user program must adhere to a number of standards. The muVium API provides classes with which the joBot hardware can be controlled. In addition the class javaBot.UVM.JobotBaseController is provided which provides convenient methods to access the servos and sensors. The drive and vectordrive functions are discussed below. For more information of this class see the Javadoc.

Every UVMAgent must extend com.muvium.UVMRunnable in order to run on a muVium controller. This class implements the Runnable interface and so the method run() is used to start execution. It also needs to implement the AgentWebService in order to use the Web Service interface that is provided by the simulator:

```
public class UVMDemo extends UVMRunnable implements AgentWebService {}
```

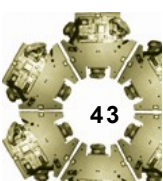
6.4.1 Web Service interface

The Web Service interface allows the program to accept commands from the RobotGUI in the simulator and execute them while having an online connection with the simulator. Also the additional RC device uses this interface. For this the following functions need to be implemented:

```
package javaBot.UVM;

import com.muvium.web.WSDL;

/**
```



```
* @author James Caska
*/
public interface AgentWebService extends WSDL
{

    public int getSensor(int sensor);

    public int getState();

    public void setState(int dip);

    public void vector(int vx, int vy, int omega);

    public void drive(int vx, int vy, int vz);

}
```

6.4.2 Drive function

This function is provided by the class `javaBot.UVM.JobotBaseController`:

```
drive (s0, s1, s2)
```

This function drives each of the three servos according to the setting of the corresponding parameter. The servos are numbered counterclockwise starting with the wheel opposite the JPB.

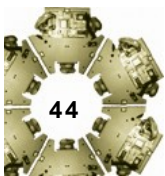
Normal rotation, as indicated by the arrows in the diagram, is indicated by a number between 1 and 100.

Reverse rotation is indicated by a number between -1 and -100.

0 stops rotation of the servo.

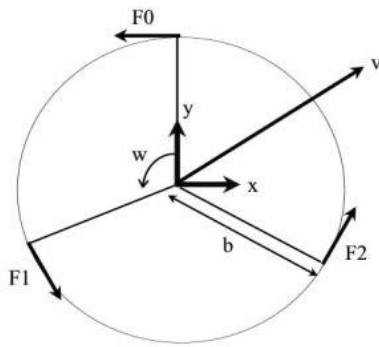
The number is an indication for the speed of the servo. Since the used servo servos are not linear, the software has to compensate for this and calculates the drive factor that has to be given to the servo. There is especially a difference between the forward and backward direction. -80 does not rotate as fast as 80 and can therefore create problems when trying to drive a straight line.

This needs to be compensated for in software and is depending on the used servo type.



6.4.3 Vector Drive function

This function also is provided by the class `javaBot.UVM.JobotBaseController`



With the drive function you have to specify exactly all motions for the three wheels. Although that is fine for fixed movements, when a trajectory needs to be calculated it is much simpler to allow the specification of a direction.

VectorDrive takes three parameters that allow the definition of direction speed and rotation of the robot:

```
vectorDrive (X, Y, Rot);
```

The X and Y coordinates specify a vector that in essence is a direction in standard Euclidian coordinates. Where the Y axes points in the direction of servo 0, the servo opposite the JPB board.

The numbers indicate the relative speed, so the following specifications are:

```
vectorDrive (0,0,0);           // Stand still
vectorDrive (100, 0, 0);      // Move right (positive X) at full speed
vectorDrive (-100, 0, 0);     // Move left (negative X)
vectorDrive (0, 100, 0);      // Move forward (positive Y) at full speed
vectorDrive (0, -100, 0);     // Move backward (negative Y)
vectorDrive (70, 30, 0);      // Move toward v full speed
vectorDrive (35, 15, 0);      // Move toward v approx. half speed
```

The third parameter specifies the rotation speed. At 0 the robot will maintain the current position. At 100, the robot will rotate along its own axis:

```
vectorDrive (0, 0, 100);      // Spin in same position
vectorDrive (0, 100, 50);     // Move forward rotating clockwise
```

6.4.4 State Machines

Robot control programs are best written in the form of a finite state machine or a number of nested stated machines.

The program runs a timed loop in which every cycle a full set of functions is performed. Such a cycle is called a heartbeat. Usually the red LED is flashed on and off during a heartbeat to show that the system is still alive.

The system keeps track of where it is in the process by keeping the current state in a state variable. When a certain event occurs, for instance when a sensor reaches a certain value, the internal state of the system is changed. That makes sure that during the next cycle another action is taken by the program.

Although you could program all statements sequentially and insert waiting loops at various places, this will not allow the program to be interrupted and change actions under external control. State machines have been designed to cope with the changing real-time aspects that are important to autonomous robots.

For an example study the code of the UVMdemo program where the heartbeat and finite state machine approach is used.

6.4.5 UVMAgent sleep

When the agent program is required to wait for a certain time, you can have the software run a loop, but in general it is much better to let the program wait a number of cycles using a sleep function.

A special version of the Java sleep function is available that allows the muVium controller to wait a number of microseconds, while at the same time allowing interrupts to be processed. In this way the processor can perform other functions while the main program waits for a certain time to expire. The code for this looks as follows:

```
public class MyUVMAgent extends UVMRunnable implements AgentWebService
{
    private void agentSleep(int delay)
    {
        long t = System.currentTimeMillis();
        while(System.currentTimeMillis() - t < delay){
            super.doEvents();
        }
        .....
    }
}
```



6.5 Servos

The servos on joBot are controlled with software driven pulse width modulator (PWM) that needs to get a value, indicating how fast the motor spins and in what direction.

The provided drive and vectorDrive functions perform all calculations to set the PWM signals. On a lower level however the system needs to get these values according to the parameters of the MultiServoController.

When the program starts up, this servocontroller needs to be created. The maximum pulse width which corresponds with the 100% signal differs per machine type and is therefore returned to the user when starting up. This maximum position is used in calculating the value that corresponds to 100%, 50% or any value in between. A value of 0 is the minimum value, equaling -100% or rotating backward full speed, while the middle position (the half value) corresponds to the motor standing still.

The basic commands are given underneath but when using the drive and vectorDrive functions the setting of the actual servo values are all taken care of automatically.

First you need to save the maximum value in order to do the calculations for the pulse width.

```
import com.muvium.UVMRunnable;
import com.muvium.apt.*;

public int max= 0; // Default is 156
private int maxHalf;
private int speed; //desired speed as a value between -100 and 100
private int[] servo={0,1,2}
protected MultiServoController controllerDirect;
controllerDirect= getPeripheralFactory().createMultiServoController(
    MultiServoController.IMPLEMENTATION_DIRECT);
controllerDirect.start();
max= controllerDirect.getMaxPosition();
maxHalf= max >> 1;
int pos = ((speed * maxHalf)/100) + maxHalf;
controllerDirect.setPosition(pos, servo[1]);
```

Please note that if the servo values are set too frequently, the system will hardly get a chance to execute the given commands and the servos will show erratic behavior. Although the servos themselves can handle 50 refresh cycles per second, the pwm generators cannot. So in general it is better to refresh with a frequency of less than 5 Hz.



6.6 DIP switches

The JPB board may be equipped with a standard DIP switch. If this is supplied, the switches may be used to set options for the program. A demo program is provided that shows how to read the switches. If the switches are not used, they should be set to the OFF position to prevent them to interfere with regular operation.

Because the switches use the weak pull-up option on the PIC controller, these pull-ups MUST be set for proper operation. If they are not set properly the inputs effectively float and may cause unpredictable behavior of all processes in the software.

The four switches are numbered from 1 to 4 and are read as a half byte (nibble).

So if switch 1 is ON, the value is read as 0x80, when 2 is ON the value is 0x40 etc.

The value is stored in the high order 4 bits of the PORTB register, so when reading the value, make sure to shift it 4 positions to the right before processing the value.

```
import com.muvium.io.PortIO;

private int dipSw;
PortIO.setProperty (PortIO.PORTB, PortIO.PROPERTY_PORT_PULLUP_ON);

PortIO.setTris (0xF0, PortIO.PORTB); // Set 0-3 as output and 4-7 as inputs
// Read the DIP switch register
dipSw = PortIO.getPort(PortIO.PORTB) >> 4;
```

6.7 Sensors

The standard joBot comes equipped with three InfraRed distance sensors that can measure a distance of up to 80 cm.

The sensors are powered by the same batteries as the motors because they draw too much current to be driven by the 5 V power directly.

Each sensor returns a voltage based on the distance sensed and this value is interpreted by one of the onboard analog to digital converters (ADC).

These converters have a resolution of 10 bits and are capable of delivering a value between 0 and 1023. The sensor, however, does not use the full scale and therefore returns values roughly between 0 and 128. Please note that although the value hardly ever gets bigger than 128 a full integer, consisting of two bytes is returned because the ADC inputs of a PIC controller are 10 bits wide.

Three of the eight ADC inputs are used and the remainder are free for other use. Some additional sensors are under development like a light sensitive sensor, a sound sensor, an analog compass and ultrasonic sensors.

```
import com.muvium.apr.ADCReader;

protected ADCReader adc;
protected int[] sensors= new int[3];

adc= getPeripheralFactory().createADCReader(0, ADCReader.READ_INT);
for (int i= 0; i < sensors.length; i++) {
```



```

        adc.setChannel(i);
        sensors[i]= adc.read();
    }

```

6.8 LEDs

The LEDs on the board may be used for various purposes. The red led is generally also used to show the heartbeat.

The JPB board supports three LEDs, located on the PortB port. If you use full port IO, PortB on the PIC controller uses bits 1-3 for the LEDs and bits 4-7 as input for the DIP switches.

The JPB-2 board supports four LEDs, located on the PortE port, using buts 4-7 as output. The definitions for the JPB-2 LEDs have been changed accordingly.

The pins must be set as outputs, unless the setTris has already been used like in the example with the DIP switches.

```

import com.muvium.io.PortIO;

private static final byte ledRED= (byte) 0x01;
private static final byte ledYEL= (byte) 0x02;
private static final byte ledGRN= (byte) 0x03;

PortIO.setPinAsOutput(ledRED, PortIO.PORTB);
PortIO.setPinAsOutput(ledYEL, PortIO.PORTB);
PortIO.setPinAsOutput(ledGRN, PortIO.PORTB);
PortIO.setOutputPinHigh(ledRED, PortIO.PORTB);
PortIO.setOutputPinHigh(ledYEL, PortIO.PORTB);
PortIO.setOutputPinHigh(ledGRN, PortIO.PORTB);

PortIO.toggleOutputPin(ledRED, PortIO.PORTB); // Heartbeat

```

Please note that the LEDs are used in inverted mode. So setting the output pin high means turning the LED off.

6.9 I/O ports

The I/O ports on the PIC controller may be read or written directly from the software. For this the PortIO is available which allows the setting of the direction of a port or its individual pins and allows the setting of pin values or an entire port, consisting of 8 bits.

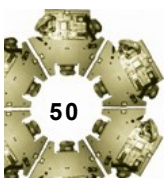
The LED routines above list most of the functions for this already. In addition to this there is full port IO in the form of setProperty, setTris, getPort and setPort:

- setProperty sets the PIC's internal pullups, which is required when using the DIP switches.
- setTris corresponds to the TRIS settings of the PIC controller, allowing the definition of the direction of each pin (0=output, 1=input).
- getPort reads a byte from the given port.
- setPort writes a byte to the given port.




```
import com.muvium.io.PortIO;

PortIO.setProperty (PortIO.PORTB, PortIO.PROPERTY_PORT_PULLUP_ON);
PortIO.setTris (0xF0, PortIO.PORTB); // Set 0-3 as output and 4-7 as inputs
dipSw = PortIO.getPort(PortIO.PORTB) >> 4;
PortIO.setPort(PortIO.PORTB, dipSw << 4); // Show switches in the LEDs
```



6.10 Optional extension units

6.10.1 Display Unit

As an option the joBot may be equipped with a display board that has a 2 x 12 character LCD screen, four LEDs, a beeper and an infrared remote control receiver.

This unit is connected to the JPB using the standard SPI interface with the JPB running in master mode and the Display Control Unit (DCU) in slave mode.



Some sample code is available to show how to use the display and the other peripherals on the DCU board.

Since the DCU is not permitted to interrupt the JPB, the program has to poll the DCU continually to see if some RC commands may have received. All this functionality is encapsulated in the sample code.

For more details check the online documentation of the DCU.

6.10.2 Displaying data

The most frequently used facility is to display data. The display acts like a standard terminal and accept strings of ASCII data, terminated by CRLF characters.

In addition it allows horizontal and vertical scrolling, which may also be controlled using a remote control unit.

Internally the display holds 2 x 40 characters of which 2 x 12 characters are visible. Horizontal scrolling will reveal the hidden part of the 2 x 40 character buffer.

If desired, the user may define up to eight special characters that may be loaded in the display unit under software control.

6.10.3 Using the LEDs

The four LEDs on the board may be used for various purposes. The red led is also used to show that a command has been received from the RC unit.

The fourth LED may be a bright blue led or could be used as an IR emitter, allowing the board to operate as an RC sender. There is no software onboard to generate the required signals however.

6.10.4 Sounding the beeper

The standard piezo beeper may be sounded by sending the <bell> character.

6.10.5 Using the Remote Control

The unit has a standard IR remote control receiver that works with any remote control device that sends the Philips RC5 code set.



Only the low order 6 bits of this code are used, thus limiting the total number of characters that may be sent to the unit.

A number of built-in commands and a special test mode may also be activated, using the remote control unit.

Currently the DCU option is only available on an experimental basis.

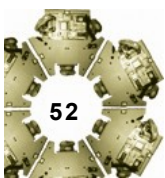
6.11 Additional peripherals

More peripherals are available or under development.

Please check the data sheets for other piggy-back modules for the JPB boards for more details.

Other units that are available are:

1. Advanced electronic compass using tilt sensors and CAN interface.
2. Simple Compass unit (Under development).
3. Web Server unit that allows direct connection to a network using an UTP connector. It supports many protocols and connects directly to the JPB.
4. Ultrasonic Range sensor. (Under development).
5. Intelligent Camera unit (Under development).
6. Light sensor.
7. Sound sensor (Under development).
8. Gas sensor (Under development).
9. Optical odometry sensor to detect robot movement (under consideration).



Chapter 7 Example Programs

An example program is provided to show the functionality of the robot. It performs the following functions:

UVMDemo This program contains most of the functionality of a UVMagent and selects functions by using the DIP switches. It contains the following subprograms:

- a. **Calibrate** This shows how to use the DIP switch on the board. It may be used to control option settings of the processor. It also shows the use of the standard LEDs that are available on the board.
- b. **Test** Simple program that drives the joBot in all four directions and makes it follow a square pattern. May be modified using the DIP switches.
- c. **Gyrate** Makes a gyrating movement, showing the omnidirectional capabilities of the joBot.
- d. **Flee** Demonstrates the robot going away from obstacles
- e. **Wall Hug** Simple wall follower that uses only one Reflection sensor

7.1 Available demo programs

A number of demonstration and test programs are available and under development to show how to use the various facilities. Several programs also contain JUnit test sets to verify the validity of the program. These may be run using the Eclipse IDE.

In principle all programs are aimed to be run both in the simulator and on the real robot. In the first version, the real and virtual robots may differ slightly. Eventually they should become completely transparent.

More detailed information about the UVMDemo program can be found in the UVMDemo manual.



Chapter 8 The Development architecture

8.1 *Compiling and Running the software*

Development of the Java software running on the JPB (but also the PC part) is done using the Eclipse IDE, using the Java plug-ins.

Once a module has been compiled, it needs to be loaded into the muVium controller. This is done in several steps:

1. Develop your program and test it in the Eclipse environment, or using the joBot simulator directly.
2. Compile the program using the muVium IDE. Later versions may integrate this process in the Eclipse environment.
3. Load the compiled program onto the JPB board.

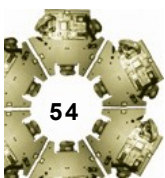
8.1.1 Writing for the muVium architecture

The muVium compiler takes the Java byte code in the .class files and translates that into machine language for the JPB's muVium controller.

Currently there are two versions of this processor, the muVium 452 and the muVium 6720 and muVium 6722. The latter one is standard on the joBot platform and allows larger programs to be written than the limited 16K space of the 452.

When writing programs for this family of processors, several things need to be kept in mind:

1. Not all Java functions have been implemented in the muVium architecture. The limited space in a small embedded processor dictates these limitations.
2. No extensive string handling facilities are available, for instance the + concatenation operator is not supported.
3. No floating point math is available on these embedded processors. All calculations are done in 8 or 16 bit integer format. So if precision is needed, you need to use scaled integer math instead.
4. The PIC controllers have a limited hardware stack. On the 452 for instance only about 6 local variables are allowed in a function, as a result of the 256 byte ram of that controller. The 6720 and 6722 allows for roughly 4k of ram and thus for larger programs and working space
5. muVium uses a concept, called the Abstract Peripheral Toolkit, which makes sure that general peripherals such as lamps, motors, sensors are defined in a standard way for different platforms. There are several libraries that come with muVium that encapsulate most of the peripherals that you might need. See chapter 7 for a description.



6. Make sure that your program derives from the base class `uvmRunnable`
7. `muVium` programs do not have a `main` but instead use the `run` method to start the process.

Violations of points 1 to 4 won't be signalled by the simulator, but the `muVium` compiler will generate errors or warnings.

Therefore you need to be very careful with the use of expensive nested function calls and printouts of debugging information since this cost a considerable amount of internal space.

Look at the examples, provided by `muVium` and by the `joBot` environment to see how to use the facilities.

When you compile using the `muVium` IDE you will get returned the % of usage on your processor so you can see how much space is still left.

8.1.2 Checking the `muVium` documentation

Because new versions of the `muVium` architecture become available frequently, please check the latest version of the `muVium` documentation. It does contain much more detailed information than can be included in this manual.

Also for good insight check the examples that are provided with the `muVium` architectures for explanations on the various facilities.

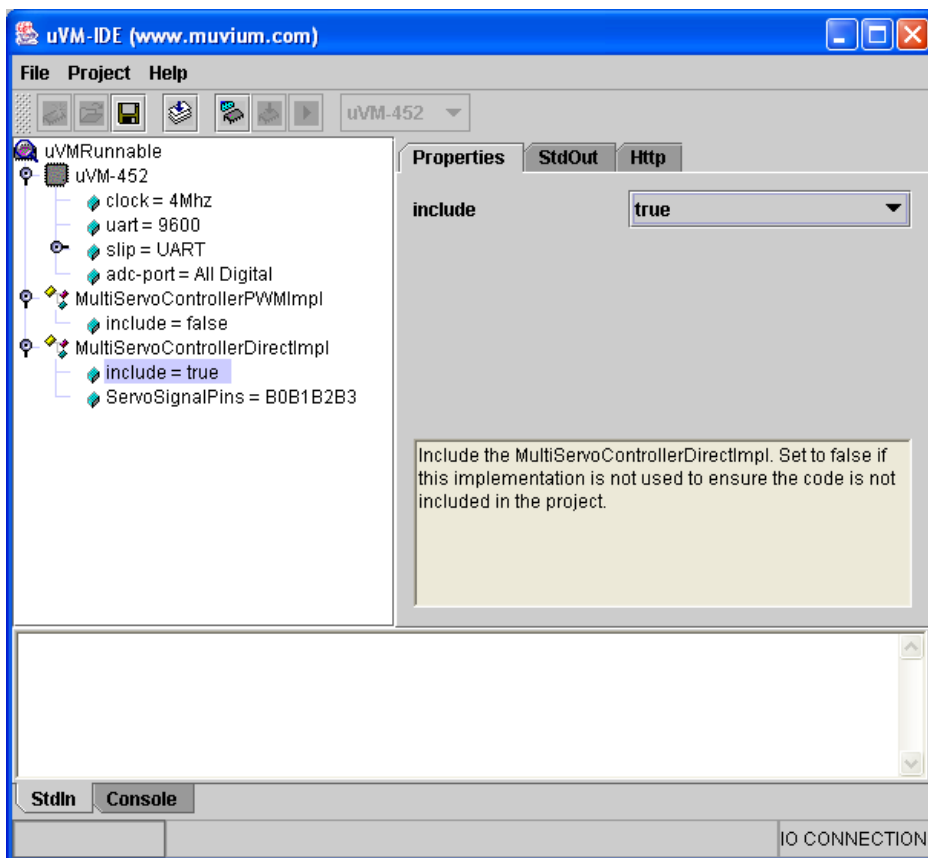
8.1.3 Creating a new `muVium` project

Once a program has been created and tested in the JPB test environment, it may be compiled and loaded onto the JPB. Start the IDE and first select the type of device you are using. For `joBot` this is the `uVM-452` for the `muVium 452` controller or `uVM-8720` and `uVm-6722` for the newer `muVium 6720` and `muVium 6722` controllers

Now select `File|New`. You will be asked to point to the path of the `.class` file that was created during compile in Eclipse.

Depending on the facilities that you have used in your program you now may have to select a number of devices and settings for the application. In the Test application you will see the following:

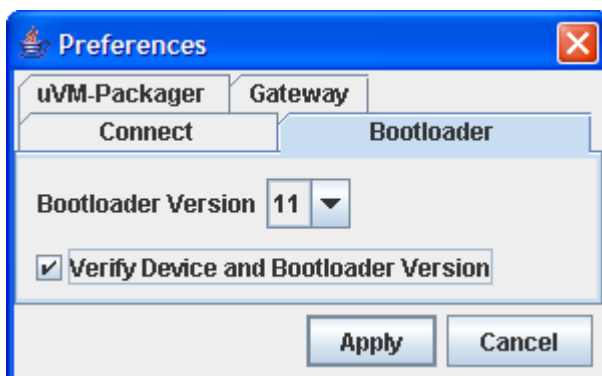




Change the settings of the uVM-452 and uVM-6720(uVM-8720) to clock = 20 MHz, uart = 57600 and adc-port = All Analog. For the uVM-6722(uVM-8720) clock = 40 MHz. The other settings are the same.

The set MultiServoControllerPWMImpI to false and MultiServoControllerDirectImpl to true. The ServoSignalPins are set to D0D1D2 for the three joBot motors.

Finally open the preferences window by selecting File > Options...



At the Bootloader tab choose the correct Bootloader Version: 11 for uVM-452 and 15 for the other controllers.

After that save the XML file that stores these settings and use it the next time when you want to open the project.

Alternatively you may use the provided XML file at ...\\src\\javaBot\\UVM\\joBot.xml
You than have to change the following lines to point to the class you want to compile:

```
FILENAME>D:\\MultiMotions\\Jobot\\JobotSim18\\bin\\javaBot\\UVM\\UVMDemo.class</UVMRUNNABLE-FILENAME><UVMRUNNABLE-CLASSFILE>javaBot.UVM.UVMDemo</UVMRUNNABLE-CLASSFILE>
```

Once you have done that, you can use File > open to read all the settings from file and arrive at the same screen as above. The complete contents of joBot.xml is listed below

```
<XMLConfig>
<DEVICE name="uVM-452">
  <PARAM name="clock" value="20Mhz">
    </PARAM>

  <PARAM name="uart" value="57600">
    </PARAM>

  <PARAM name="slip" value="UART">
    <PARAM name="IP Address" value="0.0.0.0">
      </PARAM>
      <PARAM name="Port" value="80">
        </PARAM>
        <PARAM name="Emulation" value="DIRECT">
          </PARAM>
          <PARAM name="Duplex" value="full">
            </PARAM>
            <PARAM name="packetsize" value="600">
              </PARAM>
            </PARAM>
          </PARAM>
        </PARAM>
      </PARAM>
    </PARAM>
  <PARAM name="adc-port" value="All Analog">
    </PARAM>
  </DEVICE>
  <CLASS name="javaBot.UVM.UVMDemo" />
  <CLASS name="java.lang.Exception" />
  <CLASS name="java.lang.Throwable" />
  <CLASS name="java.io.Serializable" />
  <CLASS name="java.lang.Object" />
  <CLASS name="java.lang.Thread" />
  <CLASS name="java.lang.Runnable" />
  <CLASS name="com.muvium.apt.PeriodicTimer" />
  <CLASS name="com.muvium.apt.PeripheralFactory" />
  <CLASS name="java.lang.System" />
  <CLASS name="com.muvium.io.PortIO" />
  <CLASS name="javaBot.UVM.WallHugBehavior" />
  <CLASS name="javaBot.UVM.FleeBehavior" />
  <CLASS name="javaBot.UVM.CuriousBehavior" />
  <CLASS name="javaBot.UVM.MapReaderBehavior" />
  <CLASS name="javaBot.UVM.CalibrateBehavior" />
  <CLASS name="com.muvium.driver.opticalMouse.Agilent2051" />
  <CLASS name="com.muvium.apt.Time" />
```



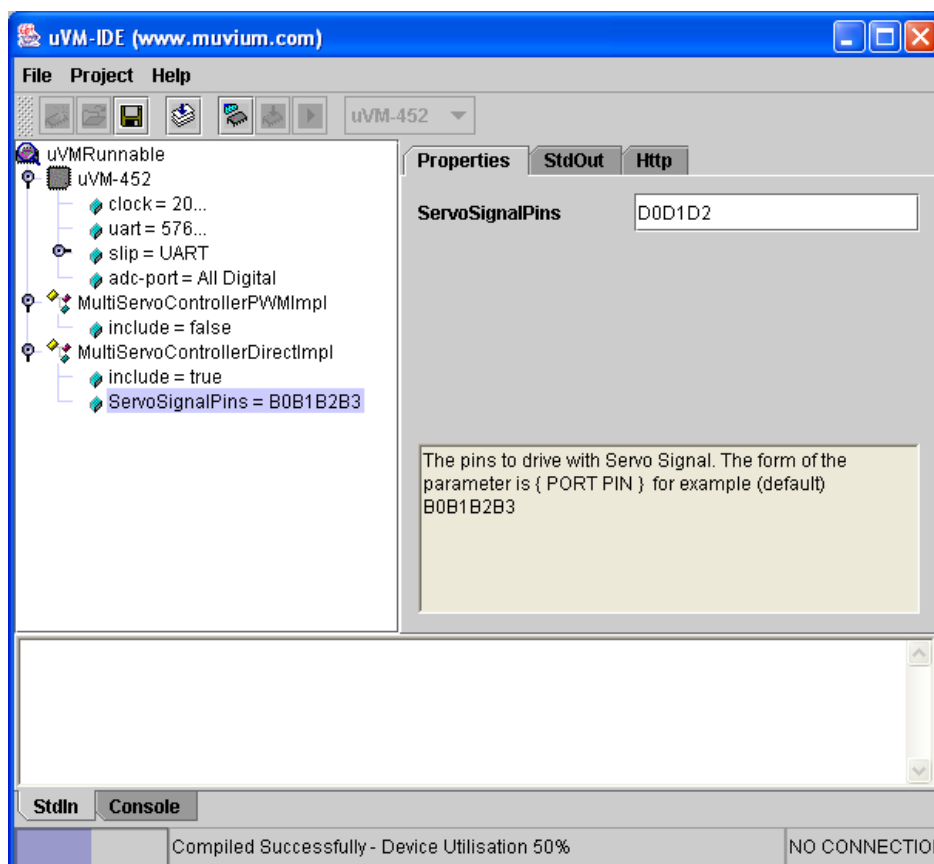
```
<CLASS name="com.muvium.aptMuvium.TimeImpl" />
<CLASS name="com.muvium.apt.SPIMaster" />
<CLASS name="com.muvium.io.PinIO" />
<CLASS name="com.muvium.apt.PeripheralFactoryImpl" />
<CLASS name="javaBot.UVM.Behavior" />
<CLASS name="javaBot.UVM.JobotBaseController" />
<CLASS name="com.muvium.apt.ADCReader" />
<CLASS name="com.muvium.apt.MultiServoController" />
<CLASS name="com.muvium.apt.PeripheralUnavailableException" />
<CLASS name="com.muvium.aptMuvium.MultiServoControllerPWMImpl">
  <PARAM name="include" value="false">
</PARAM>
</CLASS>
<CLASS name="com.muvium.aptMuvium.MultiServoControllerDirectImpl">
  <PARAM name="include" value="true">
</PARAM>
  <PARAM name="ServoSignalPins" value="D0D1D2" />
</CLASS>
<CLASS name="com.muvium.apt.TimerListener" />
<CLASS name="javaBot.UVM.AgentWebService" />
<CLASS name="com.muvium.web.WSDL" />
<CLASS name="com.muvium.apt.TimerEvent" />
<CLASS name="com.muvium.apt.EventObject" />
<CLASS name="javax.servlet.ServletResponse" />
<CLASS name="java.io.OutputStreamWriter" />
<CLASS name="java.io.PrintWriter" />
<CLASS name="java.io.Writer" />
<CLASS name="java.io.PrintStream" />
<CLASS name="java.io.FilterOutputStream" />
<CLASS name="java.io.OutputStream" />
<CLASS name="com.muvium.apt.DebuggingService" />
<CLASS name="java.lang.StringBuffer" />
<CLASS name="java.lang.StringIndexOutOfBoundsException" />
<CLASS name="java.lang.IndexOutOfBoundsException" />
<CLASS name="java.lang.RuntimeException" />
<CLASS name="java.lang.String" />
<CLASS name="com.muvium.io.UARTOutputStream" />
<CLASS name="com.muvium.io.UARTInputStream" />
<CLASS name="java.io.InputStream" />
<UVMRUNNABLE-
FILENAME>D:\MultiMotions\Jobot\JobotSim18\bin\javaBot\UVM\UVMDemo.class</UV
MRUNNABLE-FILENAME><UVMRUNNABLE-CLASSFILE>javaBot.UVM.UVMDemo</UVMRUNNABLE-
CLASSFILE>
<LAUNCHER>REAL</LAUNCHER>
<PACKAGER />
<GATEWAY-PORT />
<BOOTLOADER name=""><PARAM name="comm" value="" /><PARAM name="baud"
value="" /></BOOTLOADER>
</XMLConfig>
```

8.1.4 Compiling the program

After you have opened the XML file, you may now Select Project > package to compile the program. This is done automatically when you have an Internet connection.

After compilation the screen will look as follows:



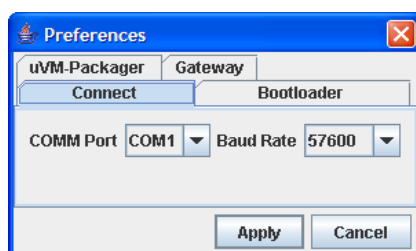


Notice that the compilation was successful and that the device is used for 50%. So the program may still grow. We are now ready to program the JPB board.

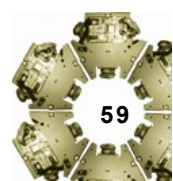
If your program doesn't comply with what is stated in paragraph 8.1.1, then an error message will be shown in the console window.

8.1.5 Establishing contact with the processor

Select now Project > Connect and Reboot. This depends also on the settings that you have to select first in the File > Options... menu:



Select the proper COM port and baud rate. The JPB board uses a standard serial port to communicate with the PC. The newer JPB-2 board allows the use of a serial port or a USB port. The USB port is implemented on the PC as a Virtual Serial Port,



thus maintaining compatibility in the software from both ends. The muVium controller uses the serial port, the PC uses a virtual serial port, even though the communication is handled by a full-speed USB 2.0 connection. USB High-speed is not supported on this device.

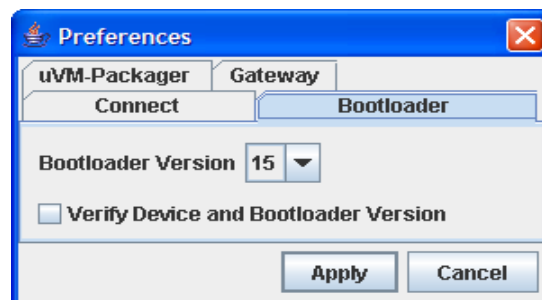
After that the system will wait for you to press the reset button on the JPB board, signaling to set the device in boot mode.

When that has been done successfully, program the device.

8.1.6 Downloading the program to joBot

Select Project > Upload and the device will be programmed. When finished, you may press Project > Run or simply reset the JPB to get the program started.

If you get errors while uploading then you might try again with the 'Verify Device and Bootloader Version' option unchecked.



8.1.7 Running and testing

Debugging facilities for muVium are not yet available. Currently you may use a number of facilities to debug your program:

Use the LED's on the joBot to give an indication of what the program does. The Test program blinks the red light to indicate a heartbeat, indicating the system is alive. When it stops, the program aborted.

When an exception occurs all LEDs are lighted.

You may also use the 4 DIP switches (if they have been installed on the board) to change settings at runtime.

Finally you may use the `System.out.println()` command to show output on the console of the IDE. However beware that this costs considerable overhead in the program and should be removed when the program is running fine.

We highly recommend the use of JUnit that is integrated as an option with Eclipse.

Some demo programs contain a test set that is used in Eclipse to test functionality of the system, before committing it to the JPB board.

Glossary

actuator Device with which a robot acts upon its environment, for instance a motor.

ADC Analog Digital Converter.

agent The software that acts as the brain of a robot

Baud Rate Designates the speed of a RS232 interface or serial port. Usually measured in bits per second. One Baud(1 signal per second) is not necessarily 1 bit per second as it is possible to code more than one bit in each signal.

breadboard Reusable solderless device used to build a (generally temporary) prototype of an electronic circuit.

byte code Code that is produced by the Java compiler and can be interpreted by a Java Virtual Machine(JVM).

code simulator emulator of the robot hardware.

COMM port Communications port synonym for Serial Port on a PC. This port implements the RS232 interface.

command interface Window that allows typed commands to be sent to the robot. These commands are then executed locally.

Compiler Computerprogram that translates text written in a computerlanguage(sourcecode) into another computerlanguage(objectcode). For instance javacode into java byte code or java byte code into assembly.

Controller see Microcontroller.

DCU Display Control Unit; Optional device that expands the JPB with a 2X12 LCD screen.

DIP switch An electric switch that is packaged in a standard dual in-line package(DIP). Designed to be used on a printed circuit board. Is used to customize the behavior of a electronic device

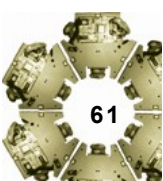
Directory Container on a filesystem in which files and other directories can be stored. On Windows system usually referred to as folder.

R Sensor Sensor with which the distance to an object can be measured.

drive Function in the joBot software that allows the speed of each motor to be set.

Eclipse Open source Integrated Development Environment(IDE) especially useful for development of Java programs.

flash memory A form of rewritable memory that holds its content without the need of a power supply. The PIC chips used for JPB have on-chip flash memory in which the program can be loaded.



finite state machine FSM or **finite automation** a model of behavior composed of states, transitions and actions.

IDE Integrated Development Environment. An environment which provides easy access to the tools needed for development and deployment of computer programs, such as for instance editors, compilers and debuggers.

IR Reflection sensor distance sensor which actively uses Infrared as a means to measure the distance to an object

IR sensor InfraRed detection sensor used to detect a active InfraRed emitting ball. This sensor detects an IR source in an approximate 60 degree wide sector in front of it.

IR floor position sensor InfraRed sensor that can be used to designated the place on a RoboCup Jr soccerfield or the boundary lines of a dance area.

Java Serial communication Java API needed to use a serial port on a PC.

JIT compiler just-in-time compilation (JIT), also known as dynamic translation, is a technique for improving the performance of bytecode-compiled programming systems, by translating bytecode into native machine code at runtime.

joBot Javabased **Omnidirectional roBot**; robot which uses a JPB.

joBot agent. Agent which can be used for a joBot, an agent is the brain of a robot.

joBot simulator Simulator which can be used on a PC to test the behaviour of agents used for joBot

JPB Java Processor Board; Board with a muVium controller, enabled to run Java programs.

JVM Java Virtual Machine. Platform dependent computer program which makes it possible to use the platform independent java byte code on diverse platforms. Either by interpreting the bytecode or by the use of a just-in-time(JIT) compiler.

LED Light Emitting Diode

Maze A labyrinth that can be generated in the simulator.

Microcontroller (MCU) A computer-on-a-chip used to control electronic devices. It is a type of microprocessor emphasizing self-sufficiency and cost-effectiveness, in contrast to a general-purpose microprocessor, the kind used in a PC. A typical microcontroller contains all the memory, peripherals and input/output interfaces needed, whereas a general purpose microprocessor requires additional chips to provide these functions.

MultiMotions Manufacturer of JPB board, joBot and robotic devices.

muVium The company that provides the tools needed to run a Java program on a PIC controller

muVium 452 controller muVium controller based on a PIC 18F452



muVium 6720 controller muVium controller based on a PIC 18F6720

muVium 6722 controller muVium controller based on a PIC 18F6720

muVium architecture The set of tools and API's developed by muVium which makes it possible to run a Java program on a PIC controller.

muVium controller PIC controller loaded with a bootstrap program that makes it possible to load programs into the on-chip flash memory through the RS232 interface. Thus enabling the loading of the program instructions into the PIC controller with a PC (via a serial port).

muVium compiler Compiler which translates Java byte code into PIC assembly language. This compiler runs as a WebService that takes a java class which is an extension of UVMRunnable and returns an file which can be loaded into the flashmemory of a PIC controller.

muVium IDE Integrated Development Environment for the tools developed by muVium.

muVium plug-in for Eclipse Extension for Eclipse which enhances Eclipse with the functionality provided by the muVium IDE

muVium project Software module that runs on a muVium controller.

muVium software Software that runs on a muVium controller.

muVium uVMDK Development Kit for muVium software

muVium Web Service interface Protocol that can be used to remotely control a JPB, for instance with an additional RemoteControl or Bluetooth Unit. Also the RobotGUI of the simulator uses this interface.

omnidirectional drive Device with such a set of wheels that it can go in any direction without turning.

Palm Pilot Handheld computer manufactured by the Palm company

PIC By MicroChip manufactured family of microprocessors. For joBot at this moment PIC18F452 and PIC18F6720 are being used. These processors have on-chip flash memory to hold the program instructions and on-chip RAM memory. The program instructions has to be loaded into the PIC controller via a special controller.

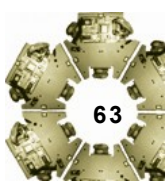
Platform combination of processor and operating system.

PPRK Palm Pilot Robot Kit.

Processor see Microcontroller.

PWM Pulse Width Modulated; Method to signal a servo to go to a certain position.

RC unit Additional unit to enable remote control of a JPB



RCjoBot joBot enhanced with features needed for RoboCup

RoboCup International competition for soccer playing autonomous robots

RoboCup Dribbler Device used in RoboCup which allows a robot to dribble with a ball. It consists of a rotating wheel that pushes the ball the beneath the robot's body.

RoboCup Junior International competition of autonomous robots in a number of games, especially for teams of students aging from 9 to 19.

robot mechanical device that makes its own decisions, recognizes and understands its environment, adapts to its environment, is able to move trough its environment and is able to communicate with people and other robots.

RobotGUI User Interface for a simulated robot in the simulator.

RS232 Standard protocol for serial binary data interconnection, commonly used in computer serial ports.

sensor Device with which a robot 'senses', becomes aware of its environment.

serial interface An interface on a computer system with which data is transferred in or out one bit at a time. (contrast parallel interface).

serial port Common name for a serial interface on a PC.

Servo motor A electric motor that usually includes a velocity and/or position feedback device.

the simulator Computer program with which robot behaviour can be simulated.

simulator-only agent Agent that can only exist inside the simulator.

SPI Serial Peripheral Interface; standard high-speed serial interface used in microcontrollers.

Ultrasonic Reflection sensor Alternative distance sensor that can be used when IR Reflection sensors aren't possible because of a active IR emitting Ball.

UPB board Universal Processor Bus; board manufactured by MultiMotions provided with a MicroChips PIC18Fxxx processor.

uVM 452 see muVium 452 controller.

uVM 6720 see muVium 6720 controller.

uVM 6722 see muVium 6722 controller.

UVMAgent Agent that can be used inside the simulator and for a muVium controller.

UVMdemo program Demonstration program for an UVMAgent implemented by javaBot.UVM.UVMDemo.java



vectorDrive Function in the joBot software that allows speed, direction and rotation of the joBot to be set.

virtual breadboard Computer program developed by muVium that simulates a breadboard and with which designs of electronic circuits can be tested.

Web Service interface see muVium Web Service interface.

Web-service A software system designed to support interoperable machine-to-machine interaction over a network, using a set of XML based protocols(SOAP, WSDL, UDDI).

world term used inside the simulator to designate the borders of the environment in which a simulated robot operates.

With thanks to [Wikipedia](#) for some of the definitions.

Links

www.multimotions.com

www.robocupjunior.nl

www.muvium.com

