DELIVERABLE D6.1

INITIAL ROBOTICS COMPONENTS AND SIMULATION ENVIRONMENT

Manuel Giuliani (FORTISS), Andre Gaschler (FORTISS), Markus Rickert (FORTISS)

Beneficiaries: FORTISS (lead)
Workpackage: WP6: Social Robotics and Embodiment
Description: This deliverable reports on the architecture and hardware components of the JAMES human-robot interaction system after the first project year. It contains an overview of the software that were used to program the robotics components and the communication between software modules. Additionally, it reports on the JAMES robot simulation environment and documents the usage of the simulator.

Version: Final
Nature: Report (R)
Dissemination level: Public (PU)
Pages: 13
Date: 2012-03-07
1 JAMES Human-Robot Interaction System

Figure 1a shows the JAMES human-robot interaction system. The robot setup consists of two industrial robot arms (Mitsubishi Melfa RV-6SL) with compliant hands (Meka H2), and an animatronic head (Philips iCat) which is capable of producing emotions and lip-synchronised speech. The robot is equipped with two stereo cameras (PointGrey Bumblebee) and two depth sensors (Microsoft Kinect), which also contain microphone arrays. The robot is surrounded by a sensor cage that allows developers to flexibly setup cameras and depth sensors. The robot is mounted on a table, which is part of a bar. The bar table can be reached by humans standing in front of the bar as well as the robot. It ensures that the robot can handover drinks to its human customers by either placing bottles on the bar or handing them over to the human directly. At the same time, the bar table also ensures that humans keep a safety distance from the robot arms.

1.1 Robot Architecture

In this deliverable, we shortly summarise the architecture of the JAMES robot in order for the reader to get a clear picture of the robot. Please refer to JAMES deliverable 7.1 for more details on the integrated system and the functions of the single software modules.

Figure 1b shows the architecture of the JAMES robot after the first project year. The robot recognises its environment—the real world—by using Visual Processing and Speech Recognition. Visual processing is capable of recognising human heads and hands, please refer to JAMES deliverable 1.1 for more details. For speech recognition, we are using the speech recogniser that is made available in Microsoft Kinect SDK\(^1\). At the time of the first JAMES system evaluation, the Kinect speech recogniser supports English speech recognition that can be constrained by a language model. The Parser processes the recognition results by speech recognition with a CCG [4]. It transforms the spoken utterance into a logical formula and sends it to the State Manager. The State Manager processes the visual and verbal information, i.e. it keeps track of the human agents that are in front of the robot’s cameras and what they said, and sends this information to the Planner. The Planner calculates the next actions the robot should execute and sends this information to the Output Planner, which then transforms these actions into multimodal output—speech, gestures, and robot actions—and sends it to the Talking-Head Controller and the Robot Motion Planner, respectively. Alternatively, output controller can send the multimodal output to the Robot Simulator, if the software should be tested without the actual robot hardware. Section 3 gives more details on this robot simulator.

\(^1\)http://www.microsoft.com/en-us/kinectforwindows/
1.2 Hardware Components

This section describes the new hardware components that were obtained specifically for the JAMES project. Descriptions for the older system components—robot arms and iCat talking-head—can be found in [3] and [1].

![Figure 2](image1.png)

**Figure 2:** New hardware of the JAMES robot. The Meka H2 compliant hands are capable of grasping and holding bottles, and of producing simple hand gestures. The PointGrey Bumblebee stereo camera is mounted at the optimal height to track heads of human customers.

**Figure 2** shows the two main new hardware components of the JAMES robot: the robot has two Meka H2 Compliant hands\(^2\) (Figure 2a and in the foreground of Figure 2b), which each have 5 series elastic actuators driving 12 joints. The underactuated fingers are built from dual durometer urethane. Fingers and thumb can be closed separately and the thumb can be abducted. The hands are strong enough to grasp and lift filled glass bottles and to hold them for about a minute before the motors overheat and go into safety shutdown. Due to the underactuated design of the fingers, the hands can grasp arbitrary objects as long as the object is smaller than the hands or has a handle and is not too heavy. Furthermore, the robot is equipped with two PointGrey Bumblebee stereo cameras\(^3\) (Figure 2b, camera is mounted directly below the iCat talking-head). The cameras have a resolution of 640x480 pixels and capture images at a frame rate of up to 48 frames per second, which makes the cameras an adequate hardware for recognition of gestures and facial expressions. The cameras are mounted directly below the head of the JAMES robot, for head and hand tracking of human customers, and above the head looking down on the whole bar scene from a view angle of approximately 45° to recognise groups of customers.

2 Software

In this section, we describe the software that were used to implement the robot arms (Section 2.1), the iCat talking head (Section 2.2) and the communication between the single components of the robot system (Section 2.3). This section is intended to provide a rough introduction to the software and pointers to further documentation and downloads.

---

\(^2\)http://mekabot.com/products/compliant-hand/

\(^3\)http://www.ptgrey.com/products/bumblebee2/bumblebee2_stereo_camera.asp
2.1 Robotics Library

To program the arms of the JAMES robot, we are using the Robotics Library (RL), which was implemented by Markus Rickert as part of his PhD thesis [2] and was used in the robotics projects JAST (Joint Action Science and Technology)\footnote{http://www6.in.tum.de/Main/ResearchJast}, JAHIR (Joint Action for Humans and Industrial Robots)\footnote{http://www6.in.tum.de/Main/ResearchJahir}, and EcceRobot\footnote{http://eccerobot.org/}. RL is available as platform-independent open source library for C++ and can be obtained at \url{http://roblib.sourceforge.net/} or by using a Launchpad repository in Ubuntu Linux.

![Figure 3: Overview of the robotics library, showing the single parts of the software.](image)

**Figure 3** shows the single parts of RL: math, mathematical basics for robotics such as vector and matrix operations based on the Eigen library\footnote{http://eigen.tuxfamily.org}; util, contains helper classes for timers, threads, mutexes, and semaphores; xml, XML wrapper for loading and saving XML representations of robot kinematics and scene descriptions; hal, the hardware abstraction layer that makes the hardware transparent to the programmer; kin, forward and inverse kinematics in Denavit-Hartenberg notation; mdl, alternative rigid body kinematics and dynamics in spatial vector algebra; sg, a scene graph abstraction, which acts as a wrapper for visualisation and collision detection; plan, implementations of common robot motion planning algorithms; ctrl, task-based robot control.

In the first year of JAMES, we were mainly using the hardware abstraction layer, the scene graph abstraction, and the kinematics to program the robot for the first system evaluation and the JAMES simulator. In the following section we give examples for using the scene graph and the kinematics.

**Scene Graph Abstraction and Kinematics**

![Figure 4: Connection of kinematics and dynamics representation of a robot to its geometric model in the Robotics Library.](image)

**Figure 4** shows the connection of scene graph and kinematics in the RL. A scene consists of several models, which in turn consist of single bodies. A body, which is specified in world coordinates, is defined by shapes, for
example a box, sphere or convex hull. For example, the JAMES robot consists of several bodies that represent
the robot’s parts, which are connected by joints. In the scene graph, models of different grades of complexity
can be used for collision detection and distance computation. Complex models can for example be reduced by
using the convex hull of the represented body to increase system performance. Figure 5 shows an example for
a scene definition in XML, which describes a model with name model1 and a body with name body1.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<r1sg
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="r1sg.xsd">
 <scene href="scene.wrl">
   <model name="model1">
     <body name="body1"/>
   </model>
 </scene>
</r1sg>
```

Figure 5: Example for an XML file describing a scene with one model.

The scene description references a geometry file, which is written in VRML (Virtual Reality Modelling Language).
The geometry file describes the bodies and shapes of a robot and its surrounding scene, as well as how the
shapes are aligned to each other. Figure 6 shows an example file written in VRML. In the scene representation,
body1 is described as shape in form of a box and the initial coordinate system of body1 is defined with the
attribute translation.

```vrml
#VRML V2.0 utf8
Transform {
  children [
    DEF model1 Transform {
      children [
        DEF body1 Transform {
          translation -1 0 0
          children [
            Shape {
              appearance Appearance {
                material Material {
                  diffuseColor 1.0 1.0 0.0
                }
              }
            }
          } size 1.5 1.5 1.5
        }
      ]
    }
  ]
}
```

Figure 6: Example for an VRML file describing the bodies and shapes of a given model.

The used kinematics describe the joint configurations of the robot. For that, the bodies of the robot are defined
in a tree structure so that they can be changed in unison. After linking bodies together in a tree, the coordinates
of the robot links can be transformed into world coordinates by using forward kinematics. The calculated values,
position and orientation, can then be assigned to a scene graph abstraction. To execute the calculated joint
control of the robot, it can be either sent to the real robot or the simulation environment by using the hardware
abstraction layer.

Figure 7 shows a 3D visualisation of the james scenario description, which displays the calculated coordinate
systems of the single links of which the robot is built. Figure 8 shows parts of the kinematics definition of the
Figure 7: Scene graph abstraction of the JAMES robot and bar showing the coordinates of the joints with link coordinate system.

JAMES robot in XML. It describes the position of the robot in the world coordinate system, which has its origin in the middle of the table that is staying in front of the robot. The XML tag `revolute` specifies the robot joints with a Denavit-Hartenberg matrix as well as the minimum and maximum values of the joint positions. Furthermore, in the tag `frame` the links of the tree that spans the robot bodies are defined.

### 2.2 Open Platform for Personal Robotics

The iCat talking-head was programmed by using the software Open Platform for Personal Robotics (OPPR)\(^8\), which was provided by the manufacturer Philips. OPPR provides the Dynamic Module Library (DML) that is used to program actions and behaviours for the iCat head. OPPR already comes with a set of animations for interaction scenarios, such as a greeting animation or a behaviour to display denial. To create own animations, OPPR contains an animation editor that provides an easy to use interface with which each individual servo of the iCat head can be controlled. The iCat head is not under development anymore, thus OPPR only runs on 32bit Windows machines. The JAMES programmers are part of an initiative that will make the OPPR software available as open source project so that it can also be ported to 64bit systems in the future.

For the first system evaluation, a set of new animations was implemented, nodding, head shaking, an animation signalling that the robot did not understand a verbal utterance by the human, as well as several facial expressions. These animations are tailored to the JAMES scenario, for example the general viewing direction of the iCat head needs to be set lower than in other scenarios, because the robot is taller than average humans. Furthermore, several methods have been implemented that run in parallel so that the head can for example talk and execute an animation at the same time to produce multimodal output. An overview of the iCat actions that have been implemented for the first version of the JAMES human-robot interaction system will be given at the end of the following section.

\(^8\)http://www.research.philips.com/technologies/projects/robotics/index.html
Figure 8: Scene graph abstraction of the JAMES robot and bar showing the coordinates of the joints with link coordinate system.

2.3 Internet Communications Engine

In the JAMES human-robot interaction system, communication between system components is realised by using the Internet Communications Engine\(^9\) (Ice). Ice is a middleware that runs on all common operating systems, including Windows, Linux and Mac OS, and supports a set of programming languages, for example Java, C++, C#, and Python. The usage of Ice is ideal for distributed projects such as JAMES, because the middleware leaves it open to the project partners to use their preferred operating system and programming language. Besides the supported operating systems and programming languages, Ice also offers a set of services, from which the JAMES project partners use the publish-subscribe service IceStorm, to implement a systemwide message broadcast from the single system components.

The interfaces between the system components are defined in an own definition language, the Specification Language for Ice (Slice). At this point, we show the Slice interface definitions for the robot arms and the iCat talking-head as examples for inter-module communication. Figure 9 shows the interface definition for the robot arms. It defines several exceptions that might occur during runtime as well as the types of drinks the robot can serve to its human customers. The method \texttt{give(Drink myDrink, string userId)} is used to tell the robot to hand

\(^9\)http://www.zeroc.com/
over a specific drink with id \textit{userId} to the user, which the robot gets from the state manager that was introduced in Section 1. Furthermore, the robot uses IceStorm to publish to other modules when it started and stopped executing this action by using the methods \textit{giveStarted()} and \textit{giveFinished()}. 

```java
module robot {
    exception OperationFailure {
        string details;
    }
    exception OperationAborted extends OperationFailure {
    }
    enum Drink {
        Coke, Juice, Water
    }
    interface RobotController {
        void give(Drink myDrink, string userId) throws OperationFailure;
        void idleMotion() throws OperationFailure;
    }
    interface RobotListener {
        void giveStarted(Drink myDrink, string userId);
        void giveFinished();
    }
}
```

**Figure 9:** Slice interface definition for the arms of the JAMES robot.

The interface of the iCat head, which is shown in Figure 10, defines the various expressions that the head can display. The iCat furthermore has several actions, which can be directly called by other software components: \textit{doExpression(Expression exp)}, execution of the given expression, \textit{lookAround()}, look around randomly in the bar area, \textit{lookAt(string id)}, look at agent or object with id \textit{id}, and \textit{say(string data)}, say the sentence that is specified by \textit{data}. The iCat also publishes when it started and stopped doing an expression or uttering a verbal expression. For this, it uses the methods \textit{expressionStarted(Expression exp)} and \textit{expressionFinished()}, or \textit{speechStarted(string speechStr)} and \textit{speechFinished()}, respectively.

```java
module iCat {
    enum Expression {
        Nod, NodTwice, ShakeHead, ShakeHeadTwice, Frown, Smile, SadFace, NotUnderstand
    }
    interface ICatController {
        void doExpression(Expression exp);
        void lookAround();
        void lookAt(string id);
        void say(string data);
    }
    interface ICatListener {
        void expressionStarted(Expression exp);
        void expressionFinished();
        void speechStarted(string speechStr);
        void speechFinished();
    }
}
```

**Figure 10:** Slice interface definition for the arms of the JAMES robot.
3 Simulation Environment

Since JAMES is a distributed project and not all project partners have a direct access to the robot hardware, a robot simulator was set up that can be used to develop and test software that communicates with the robot. The robot simulator could also be called a robot visualisation, because it mainly shows the movements of the robot. The simulator is running on a virtual Windows server and the communication of software components to the server is realised over Ice. This section, which is intended as a user documentation, describes how users can get access to the server (Section 3.2), how they can transfer their data onto the server (Section 3.1), and how they can run the simulator to test their own software components (Section 3.3).

3.1 User Accounts

The JAMES robot simulator is reachable under URL http://james.fortiss.org. Users can either log on to the software by using a remote desktop connection (rdpv5) to get a graphical user interface (GUI) to work on the server or they can connect to the server via Ice to test connections to the robot without getting a visual feedback.

JAMES developers who want to use the robot simulator can obtain a user account by writing an informal email to Manuel Giuliani (giuliani@fortiss.org) or Andre Gaschler (gaschler@fortiss.org). The simulation server is a Windows virtual machine and always online. For this reason, users are advised to follow a set of rules:

- Users need to pick a strong password containing at least one capital letter and at least one digit. The JAMES simulator has a gigabit internet connection, which makes the server a desirable target for hackers.
- Each user gets an own user account registered to the user’s last name. Users are responsible for all actions made by their accounts. Accounts should not be shared.
- The simulation is not meant to be used as a desktop machine because of its limitations in computing power and hard disk space.
- Users are advised to keep copies of their files stored at a separate place. The virtual machine of the simulation server may be reverted or reinstalled at any time.
- Users should use the common directory “C:\james” if they want to save files on the computer that need to be accessed by other users.
- The firewall of the server is very restrictive. If some program cannot connect to the server, the reason may be the firewall. For the Ice middleware, tcp ports 50000–50010 are open.
- Only two users may remotely connect to the server at the same time. However, disconnecting and reconnecting while leaving all programs running on the server is possible at all times. In busy times, users should disconnect their remote desktop connection to the server (no need to log off) when graphical view on the server is not needed. The number of Ice connections is not affected by this limitation; users can have their software running without a graphical view and work with the Ice middleware at all times.

3.2 Data Transfer

There are three ways to copy data files onto the server:

1. Direct copy and paste from a Windows PC to the remote desktop window. This is only recommended for non-software files.
2. SVN client. When logged in via remote desktop connection, user can use the software TortoiseSVN\textsuperscript{10}, which installed on the server, to access a subversion repository and download files. For that, users should

\textsuperscript{10}Information on TortoiseSVN and documentary can be found at http://tortoisesvn.tigris.org/
open a Windows Explorer and use the TortoiseSVN menu which opens by right-clicking in the folder in which the data should be copied into. This is recommended for all software.

3. SCP client. Similar to the SVN client, the software WinSCP\textsuperscript{11} is installed on the server. This method is useful for copying files from a (Linux) server.

3.3 Running the Simulation Server

The simulation server can be used in several stages during the software component development, testing and deployment. It basically supports two usage scenarios: first, the Ice registry runs on the simulation server and the software components that need to be tested point to the registry on the server, which is the preferred method to run simple interface tests. In the following, we refer to this method as remote master integration tests. Second, users can locally configure and run a master node. This scenario is preferred for integrative testing on the Ice infrastructure, testing multiple components, or testing locally installed hardware. We refer to this method as local master integration tests. The latter architecture allows tests closer to the real system, and may be necessary while development is proceeding. Both usage scenarios are outlined in the following paragraphs.

Remote Master Integration Tests

For a remote master integration, little or no software components run on the local machine of the user. The developer may install the software component to be tested directly on the simulation server, where many components are pre-installed and readily configured. Alternatively, developers can run their software component on their local computers and establish an Ice connection to the components running remotely on the simulation server.

As the robot component requires several numerical, hardware abstraction and robotics-related software libraries, as it was shown in Section 2.1, it is not straightforward to install on many systems. It is reasonable to run the robot simulation component remotely on the simulation server. The robot simulation component implements the exact functions of the real JAMES robot, with the exception that its output is visualised in a 3D view, as shown in Figure 11.

![Figure 11: 3D visualisation of the JAMES robot simulator.](image)

To use the visualisation, users should execute the following steps:

1. Establish a remote desktop connection with the simulation server.

\textsuperscript{11}Information on WinSCP and documentary can be found at http://winscp.net/eng/docs/start
2. Run the Ice master node, robot node, logging node and any other node required for the testing scenario. A startup command script to run these nodes can be found in C:\james\home\bin\run_simulation_server.cmd. This script runs all respective Ice grid nodes in command windows and finally opens an Ice grid GUI window.

3. The startup command scripts also opens an Ice grid GUI with which the robot visualisation and a dummy vision input module can be started. To log into the Ice grid, users should click on “File → Login...” and click “OK” on the appearing window with title “Login - IceGrid Admin”. Figure 12 shows the opened Ice grid GUI.

4. To start the robot server and the vision dummy input component, users need to left-click on the plus symbols of the Ice nodes named “robot” and “vision” and to right-click and choose option “Start” on the servers named “Robot” and “VisionDummy”.

5. After these steps, the robot server is waiting for other software components to call its Ice interfaces. A sample application to call robot commands can be found in the JAMES project SVN at https://svn.ecdf.ed.ac.uk/repo/inf/james-project/software/trunk/robot/src/client/. This very simple application will compile on most platforms and allows users to locally call all remote robot functions, in order to ensure that the Ice grid is correctly configured.

6. Tests using the robot simulation component may be performed at this point. The actual motion of the robot is visible in the remote 3D view, and allows visual inspection in order to avoid collisions or otherwise unwanted motion paths.

![Image of Ice grid GUI with running robot and vision dummy simulator.](image.png)

**Figure 12:** A running Ice grid GUI with running robot and vision dummy simulator.

### Local Master Integration Tests

While the development of the distributed system proceeds, tests on the integrated system itself become more and more necessary. For this, it is often cumbersome to install and run components on the simulation server; most notably, platform or hardware dependent components cannot be deployed on the virtual Windows machine. This section demonstrates another application of the simulation server that explains how to test data processing and planning together with computer vision and robot components. For this, the JAMES simulation server offers the computer vision dummy component that was programmed by FORTH, which can be used to publish artificial recognition results to the Ice grid.

To use the JAMES simulation server as extension to your local machine, users have to execute the following steps:
1. Configuration of local Ice setup to accept connections from the robot and vision node located on http://james.fortiss.org.

2. Establish a remote desktop connection to the simulation server.

3. Configuration of Ice setup on simulation server to connect to the Ice registry on the local computer by the user. The configuration files for the robot and vision node are stored in the folder C:/james/integrated-system/config/nodes/

4. To run the vision and robot node on the simulation server, users can use the command script that is stored in C:/james/home/bin/run_robot_vision_node.cmd

5. Users need to make sure that all remote nodes are online.

6. To run the dummy vision process and the robot-coach process manually in the robot node, users should use configuration of the Ice grid GUI that was presented in the previous paragraph.

References


