

# The iCub Humanoid Robot

by  
Semih Onay  
Supervised by  
Associated Professor Elena Battini Sönmez

Undergraduate Program in Computer Science  
Istanbul Bilgi University  
2015

## TABLE OF CONTENTS

<b>LIST OF FIGURES</b>	<b>2</b>
<b>LIST OF SYMBOLS/ABBREVIATIONS</b>	<b>3</b>
<b>1. ABSTRACT</b>	<b>4</b>
<b>2. INTRODUCTION</b>	<b>5</b>
<b>3. LITERATURE REVIEW</b>	<b>6</b>
<b>4. METHODOLOGY</b>	<b>7</b>
4.0.1. The iCub Cognitive Architecture . . . . .	7
4.0.2. The Embodiment . . . . .	8
4.0.3. Software Representation . . . . .	8
4.0.4. The YARP . . . . .	9
4.0.5. iCub Simulation Environment . . . . .	10
<b>5. CONCULUSIONS</b>	<b>11</b>
5.0.6. Speech Synthesis . . . . .	11
5.0.7. Object Grasping . . . . .	11
5.0.8. Image Capture . . . . .	11
5.0.9. Sound I/O . . . . .	11
APPENDIX A: Sample Code Snippets . . . . .	12
A.0.10. Turkish Synthesis Module . . . . .	12
A.0.11. Object Mover Module . . . . .	14
APPENDIX A: Screenshots . . . . .	15
<b>Bibliography</b>	<b>17</b>

**LIST OF FIGURES**

Figure 4.1. General View of iCub Cognitive Architecture . . . . .	7
Figure 4.2. iCub Internal Architecture . . . . .	9
Figure 4.3. YARP Command Line Interface . . . . .	10
Figure 4.4. iCub Simulation Environment . . . . .	10
Figure A.1. Full Body Movement . . . . .	15
Figure A.2. Action Recognizer . . . . .	15
Figure A.3. iSpeak Module . . . . .	16
Figure A.4. iCub Sample Body Postures . . . . .	16

**LIST OF SYMBOLS/ABBREVIATIONS**

YARP	Yet Another Robot Platform
ACE	The ADAPTIVE Communication Environment
ODE	Open Dynamics Library
OpenGL	Open Graphics Library
SDL	Simple DirectMedia Layer

## 1. ABSTRACT

Intelligent agents with human level of proficiency is the ultimate challenge for science and technology. Especially for the artificial intelligence field. Designing of such systems now eager to evolve and behave like humans. Recent discoveries in this field are focused on training communication skills with perception, classification and action which are modelled from humans. They are systems that are capable of receiving environmental activities(stimulus) and communication. This paper presents a cognitive humanoid robot that combines abilities such as object detection, image acquisition, language and movement. Aim is to understand how motor control abilities, visio-motor abilities and linguistics are designed and developed. Study includes robotic simulation applications. Cognitive modules will be developed by using application templates and documentation.

Keywords : cognitive science, humanoid robots,iCub

## 2. INTRODUCTION

The iCub is a humanoid robot that developed at Istituto Italiano di Tecnologia (IIT) as part of European project RobotCub and later adopted by more than 20 laboratories around the world. It has 53 motors that can move the head, arms and hands, midriff, and legs. It can see and hear, it has the sense of proprioception<sup>1</sup>. It's designed to aid studies of human cognition and artificial intelligence. Project members developed computer simulator to experiment new techniques.

Computer simulations are getting important in area robotics area. Simulations may not provide real complexity of the physical world and not reliable as real dynamics. The simulator of iCub is an easy way to test new algorithms and methods instead of dealing with complex configuration of iCub hardware. The simulator is designed to be accurate as real world psychics and dynamics. Development is based on directly from first prototype of simulation environment Webots.<sup>2</sup> It was expensive and had limited access to source code which made hard to modify source code in order to add some functionalities. Then iCub simulation created. Simulation environment uses to simulate body movement and collision detection algorithms to measure psychical interaction with the world. ODE<sup>3</sup> is used in wide range of projects like GAZEBO<sup>4</sup>. ODE is an open source physics engine for authoring tools, computer games, etc. It uses OpenGL<sup>5</sup> renderer and it has some disadvantages due to limitation of OpenGL engine computation efficiency on complex structures. iCub simulation is using OpenGL directly via SDL<sup>6</sup> which helps to render complex robot movements and computationally efficient simulation observations. The aim of this study is to experiment and understand how cognitive architecture of robotics designed and implemented.

---

<sup>1</sup>Body configuration and movement using accelerometers and gyroscopes

<sup>2</sup>Webots : <https://www.cyberbotics.com>

<sup>3</sup>Open Dynamics Engine : <http://www.ode.org>

<sup>4</sup>GAZEBO : <http://gazebosim.org>

<sup>5</sup>Open Graphics Library : <https://www.opengl.org>

<sup>6</sup>Simple DirectMedia Layer : <https://www.libsdl.org>

### 3. LITERATURE REVIEW

Development of simulator is described by abstraction of parts to handle complex instructions more precisely and efficient. Some other external software libraries are used to reduce required time to animate given parts of robots body from parameters. Abstractions made it easy to implement new methods, algorithms into a simulation environment. Understanding of these libraries are the key of creating new interfaces and methods to robot in virtual world. Action primitives are pre-defined inside a simulation environment to extend capability of creating new interfaces to virtual simulation world and can be changed or taught as different languages which helps to extend knowledge about languages.

The great challenges in robotics is to imitate human behaviour. Most of the humanoid robots have some perception to recognize their surroundings autonomously. Grasping an object in its environment, is one of the most difficult tasks for all kinds of robots. Many parameters must be provided for, the objects, surroundings, and the specification of the assignment. Taking these parameters into consideration, the ability to receive sensing information from the robot is crucial when implementing an efficient robotic grasp. The quality of the sensing information must also be taken into consideration, as signals may limit precision and can potentially be noisy. In recent years, there have been several models implemented to perform a grasping behavior. [V. Tikhanoff, A. Cangelosi, G. Metta Platform ]

- Knowledge based grasp action,
- Geometric grasp action,
- Sensor driven and learning based grasping.

## 4. METHODOLOGY

### 4.0.1. The iCub Cognitive Architecture

The architecture consists of modules and sub-modules. Their relations are similar to the human mental system. Gaze controlling, reaching, and locomotion establishes the simple goal oriented actions. Episodic and procedural memories are effects a simple version of an internal simulation to provide capabilities for prediction and reconstruction, as well as productive model construction by experienced affordances from environmental sensors. A simple process of homeostatic state is achived by the affective state that provides action selection and attention selection. These futures are already implemented in the simulation environment. Indivial components of the ar-

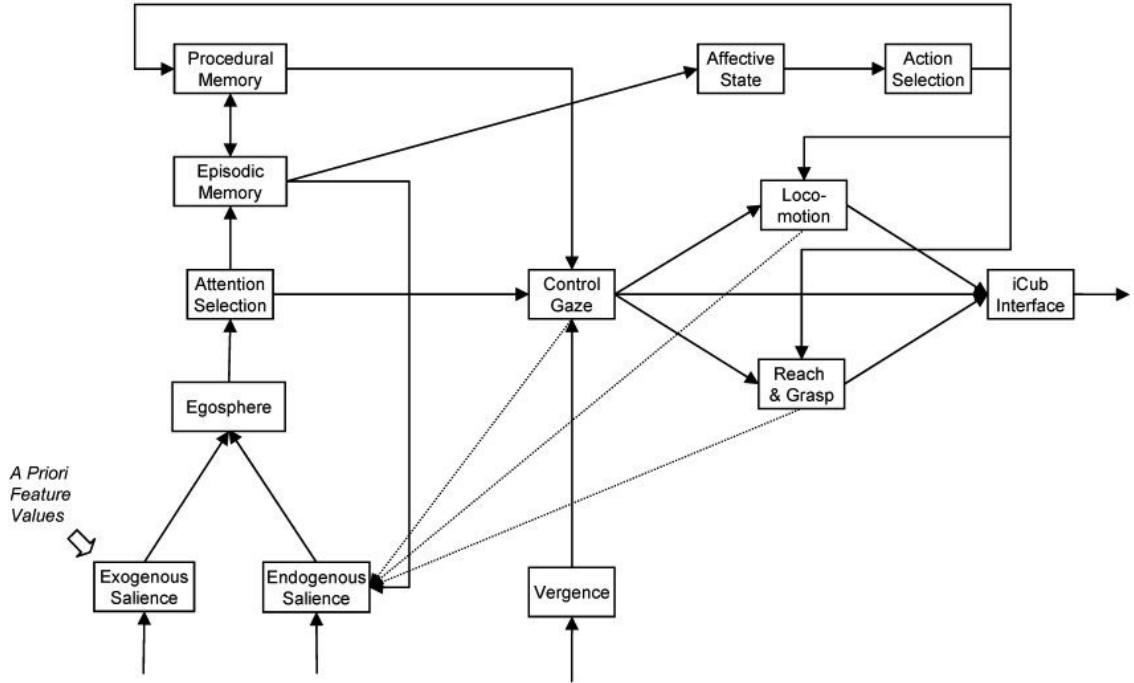


Figure 4.1. General View of iCub Cognitive Architecture

chitecture operates synchronously for a series of states representing cognitive behaviour that emerges from the interaction of separate parallel processes instead of being governed by some state machine as it like the most of cognitive architectures. In addition, motivations are encapsulated in the affective state of the system. They are designed explicitly to address curiosity and experimentation by using explorative motives which are triggered by exogenous and endogenous factors, respectively. Distinction between the exogenous and endogenous salience is represented by the need to include an attention system to consolidate both factors.

#### 4.0.2. The Embodiment

It is developed as an embodied agent by its physical components joints, actuators, and a many of sensors that are providing internal stimuli and external stimuli information about the body and its surroundings. Actuation is effected through a variety of DC motors and servo-motors. Sensory data provides joint position, velocity of movements, and torque, streamed video images from the head eye cameras, audio from the left and right ear microphones[**David Vernon** ]. Accessing to the motor control and sensor interface is provided through iCubInterface in the cognitive architecture in Fig. 5.1 This module is represented as an YARP software module iCubInterface. That interface turns relevant data into actions.

#### 4.0.3. Software Representation

Software implementation is shown in Figure 5.2 has been realized as a complete software system consist of a YARP and iCub modules. All modules are connected internally and communicates through YARP ports and they can be called from a application or from command line interface with required paramaters. These modules are:

- salience
- endogenousSalience
- egoSphere
- attentionSelection
- episodicMemory
- proceduralMemory
- affectiveState
- actionSelection
- controlGaze2
- logPolarTransform
- cameraCalib

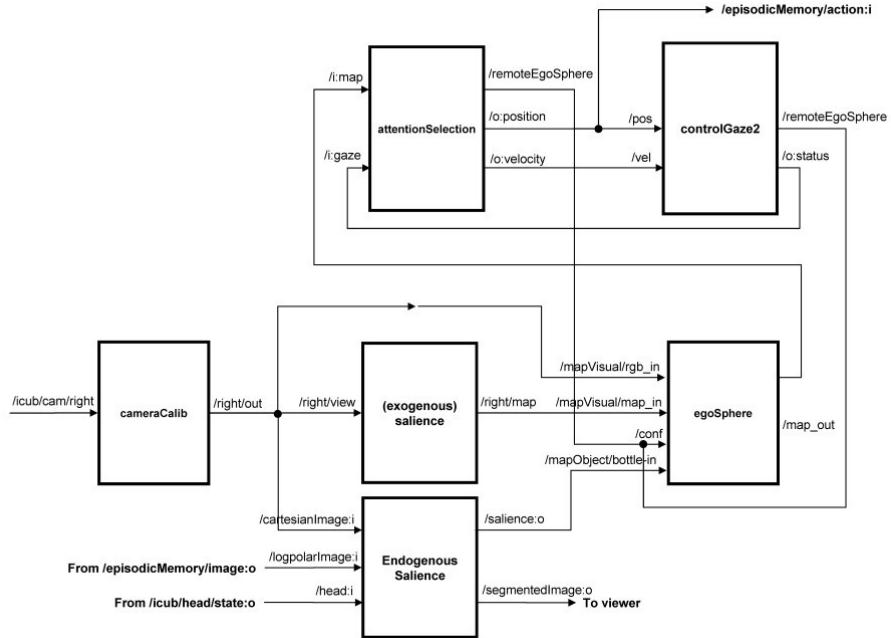


Figure 4.2. iCub Internal Architecture

#### 4.0.4. The YARP

It is a set of open source libraries that supports modularity by using abstraction method in softwares to handle common difficulties in robotics area which are known as modularity algorithms, hardware interfaces and OS platforms. To deal with OS specific builds, requires to use cross-platform build tools such as CMake<sup>7</sup> and ACE<sup>8</sup>. YARP is providing platform independence. First abstraction can be described as a protocols. Main YARP protocol manages interprocess communications in operating systems. It can deliver process messages of any size across the network by using different protocols. Second abstraction is about hardware communications. The method is to define interface for class of devices to fold native coded APIs. Changes in hardware requires changes in API<sup>9</sup> calls via linking suitable libraries to encapsulate hardware dependency problems. These abstractions combined to use remote device drives where that can be accessed across the network like a parallel processing. The purpose of YARP ports are to move data from threads to threads over the processes. Flow of the data can be configured and observed from command-line at real time. Port can receive or send data from any other port. Connections between ports can be modified easily with using different protocols such as TCP<sup>10</sup> and UDP<sup>11</sup>. The choice depends on quality of message transmission or response time. Using TCP is for reliability and UDP is for speed with effect on unreliable transmissions. As it is seen in Figure 5.3 it registers a name server over localhost and assigns root node to branching iCub components.

<sup>7</sup>CMake : <http://www.cmake.org>

<sup>8</sup>The ADAPTIVE Communication Environment : <http://www.cs.wustl.edu/~schmidt/ACE.html>

<sup>9</sup>API : Application Programming Interface

<sup>10</sup>Transmission Control Protocol

<sup>11</sup>UDP : User Datagram Protocol

```

\ \ \ \ \ \ \ \ \ \ \ \ 
\ \ \ \ \ \ \ \ \ \ \ \ 
Call with --help for information on available options
Options can be set on command line or in /Users/Semih/Library/Application Support/yarp/config
/yarpserver.conf
Using port database: :memory:
Using subscription database: :memory:
IP address: default
Port number: 10000
yarp: Port /root active at tcp://192.168.1.131:10000

Registering name server with itself:
* register "/root" tcp "192.168.1.131" 10000
+ set "/root" ips "127.0.0.1" "192.168.1.131"
+ set "/root" process 28689
* register fallback mcast "224.2.1.1" 10000
+ set fallback ips "127.0.0.1" "192.168.1.131"
+ set fallback process 28689
Name server can be browsed at http://192.168.1.131:10000/

Ok. Ready!

```

Figure 4.3. YARP Command Line Interface

#### 4.0.5. iCub Simulation Environment

The computer simulation model of the iCub allows to create realistic scenarios in where robot can interact with a virtual world and physical limitations, interactions that occur between the virtual world is simulated using open source library ODE to provide accurate simulation of body dynamics. Simulation surroundings can be manipulated by a single configuration file which contains ; textures, objects, default arm positions, video configurations and calibration settings. In order to place some objects in the environment, it needs to be spawned by using the command line interface. Environment supports direct frame streaming through yarpview application.

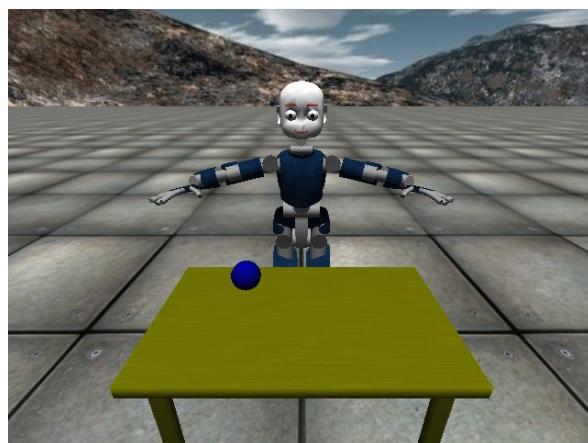


Figure 4.4. iCub Simulation Environment

## 5. CONCULUSIONS

### 5.0.6. Speech Synthesis

Speech module has ability to speak given text inputs only in English language. A new Module developed from application template with open source text to speech library. iCub now can speak Turkish language. Module can be extended to synthesis of another languages. eSpeak wrapper can provide various of language sound database. Speech Recognition module exists but it needs some Windows system libraries.

### 5.0.7. Object Grasping

Module tested with example data sets. It can grasp an object from table. It requires that other modules support. Required modules must be running before the grasping module. It needs to be trained from a neural network file.

### 5.0.8. Image Capture

An application developed from templates to capture actual action sequences from head camera. Images can be streamed as a video.

### 5.0.9. Sound I/O

Small module developed to send sound commands from computer microphone. Module documentation was not enough descriptive to create new audio ports for speech recognition. Some applications were not complied at all.

All implementations are avaible on [GitHub](#).

## APPENDIX A: Sample Code Snippets

### A.0.10. Turkish Synthesis Module

```

class iSpeak : protected BufferedPort<Bottle>,
public     RateThread
{
    string name;
    string package;
    string package_options;
    deque<Bottle> buffer;
    Mutex mutex;

    bool speaking;
    MouthHandler mouth;

    // gets text sequence
    void speak(const string &phrase)
    {
        string command("echo\u00a0\"");
        command+=phrase;
        command+="\u00a0|";
        command+=package;
        command+="\u00a0";

        // Check for speech synthesis library
        if (package=="espeak")
            command+= "-v\u00a0turkish\u00a0--stdin";

        // Get text input from CLI
        command+=package_options;
        //int ret=system(command.c_str());
    }
}

```

```
}

void run()
{
    string phrase;
    double time;
    bool onlyMouth=false;
    int rate=(int)mouth.getRate();
    bool resetRate=false;
    double duration=-1.0;

    mutex.lock();
    // protecting also the access of size() function
    if (buffer.size()>0)
    {
        // allocate space from yarp Bottle
        Bottle request=buffer.front();
        buffer.pop_front();

        if (request.size()>0)
        {
            if (request.get(0).isString())
            {
                phrase=request.get(0).asString().c_str();
                speaking=true;
            }
            else if (request.get(0).isDouble() || request.get(0).isInt())
            {
                time=request.get(0).asDouble();
                speaking=true;
                onlyMouth=true;
            }
        }
    }
}
```

}

}

#### A.0.11. Object Mover Module

```
public:objectMoverThread(ResourceFinder &_rf) : rf(_rf) {
    virtual bool loadParams() {
        name = rf.check("name", Value("objectMover")).asString().c_str();
        neckTT = rf.check("necktt", Value(2.0)).asDouble();
        eyeTT = rf.check("eyett", Value(1.2)).asDouble();
        trajTime = rf.check("trajtime", Value(4.0), "Solver\u20d7trajectory
time").asDouble();
        maxPitch = rf.check("maxpitch", Value(30.0), "Torso\u20d7max\u20d7pitch").asDouble();

        //get which arm to use. default to left if they didnt pass in left
        armname = rf.check("arm", Value("left"), "arm\u20d7name").asString().c_st
        if (armname == "right") {
            armInUse = true;
        }
        else {
            armInUse = false;
        }
        //get robot name to use (icubSim is for simulation)
        robotname = rf.check("robot", Value("icubSim"), "robot\u20d7name").asString();
    }
}
```

## APPENDIX A: Screenshots

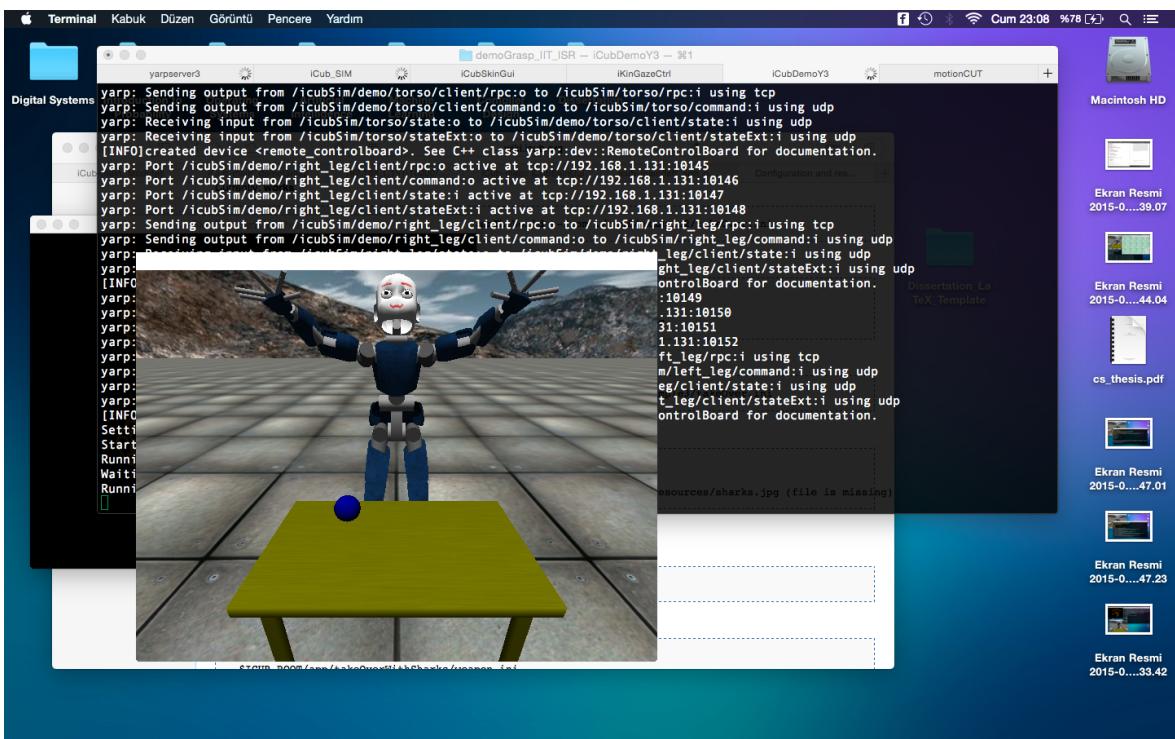


Figure A.1. Full Body Movement

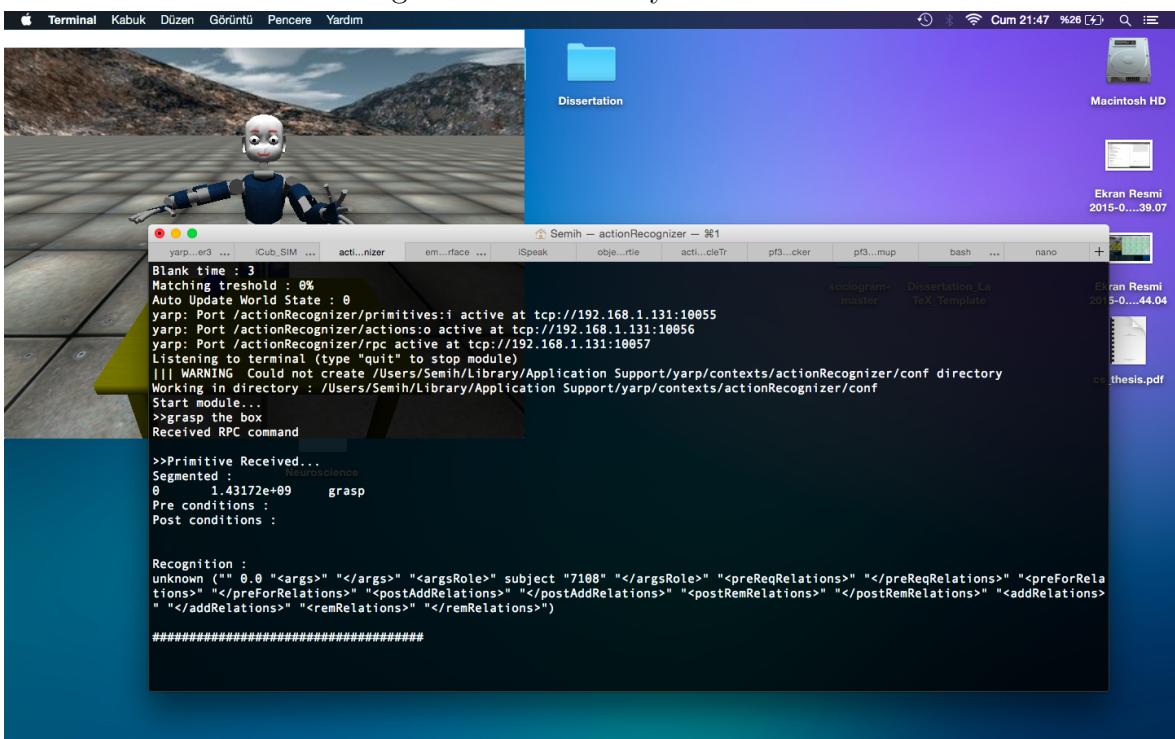


Figure A.2. Action Recognizer

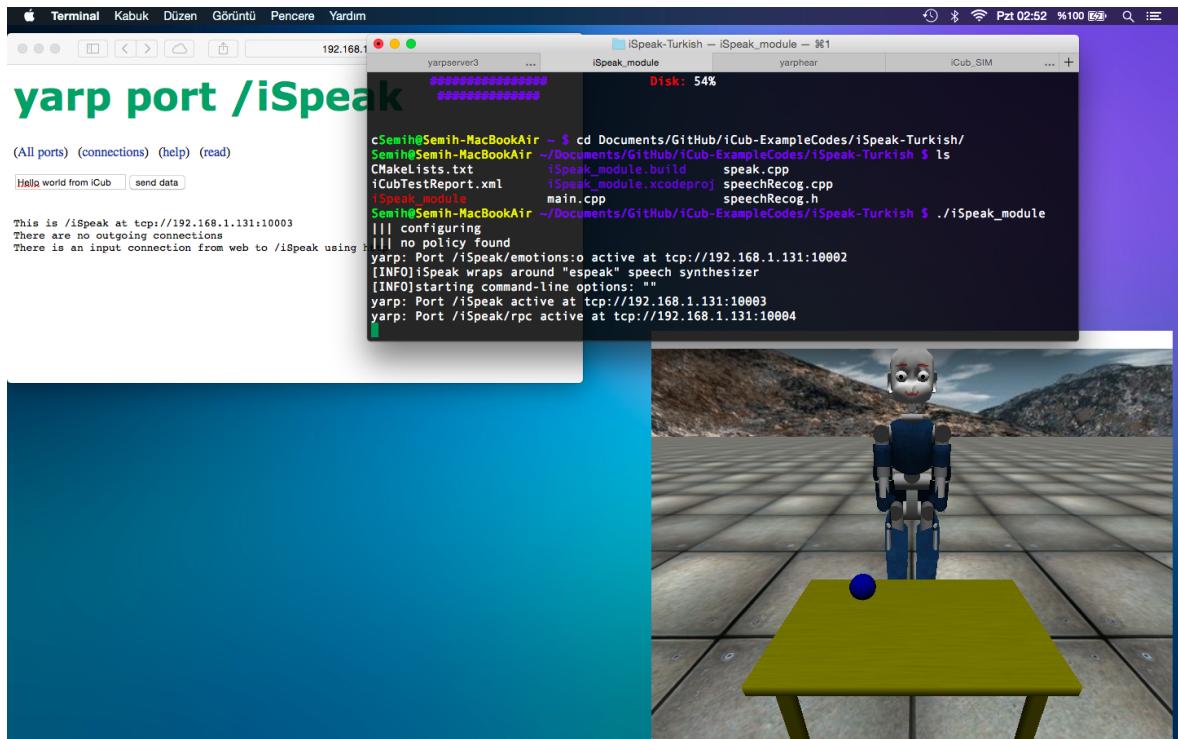


Figure A.3. iSpeak Module

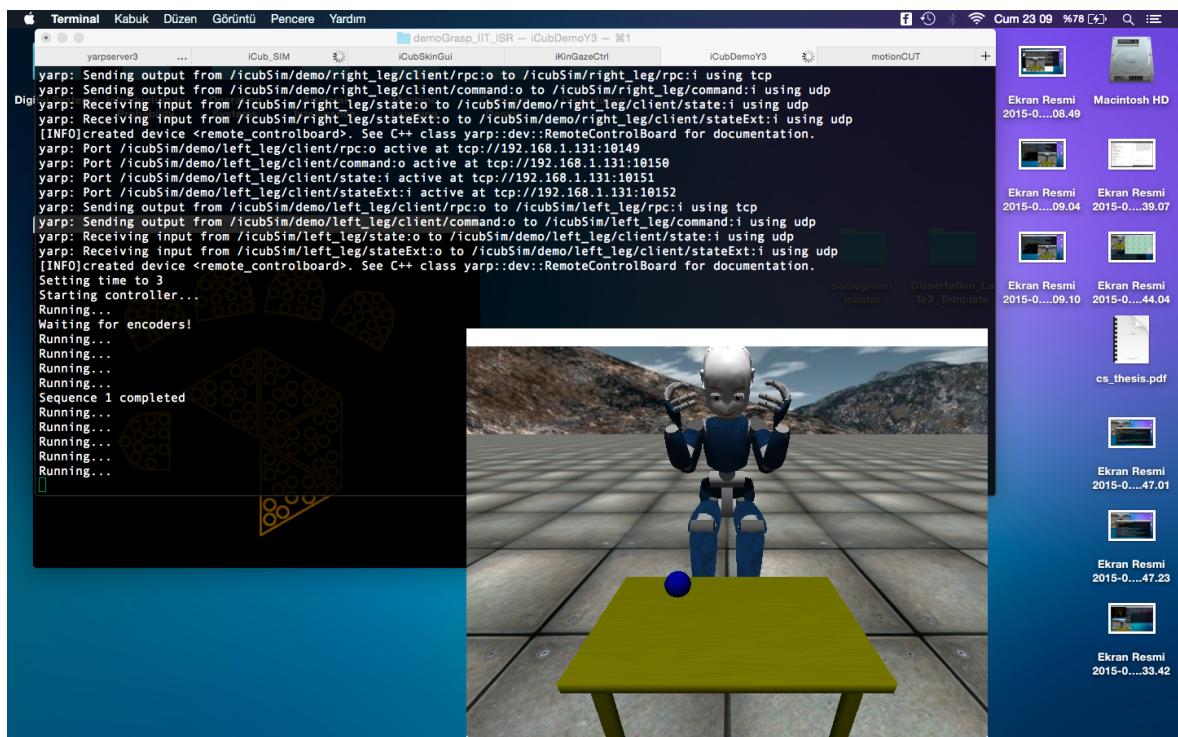


Figure A.4. iCub Sample Body Postures

## Bibliography

**1. Webots: Robot simulation software**

<https://www.cyberbotics.com>

**2. ODE: Open Dynamics Engine**

<http://www.ode.org>

**3. Gazebo: Open Source Simulation Environment**

<http://gazebosim.org>

**4. YARP: Yet Another Robot Platform**

<http://wiki.icub.org/yarp/>

**5. CMake: Cross-Platform Open Source Build System**

<http://www.cmake.org>

**6. ACE: The ADAPTIVE Communication Environment**

<http://www.cs.wustl.edu/~schmidt/ACE.html>

**7. iCub: An Open Source Cognitive Humanoid Robotic Platform**

<http://www.icub.org>

**8. David Vernon The iCub [online]. URL <http://www.referenceforbusiness.com/biography/A-E/Ballmer-Steve-1956.html>. Accessed 18 March 2015.**

**9. Speech and Action Integration in Humanoid Robots: Simulation Experiments with iCub, V. Tikhanoff, A. Cangelosi, G. Metta Platform**

**10. The iCub humanoid robot: an open platform for research in embodied cognition, Giorgio Metta Giulio Sandini, David Vernon, Lorenzo Natale Francesco Nori**