

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

LIST OF FIGURES	II
LIST OF SYMBOLS/ABBREVIATIONS	III
1. ABSTRACT	IV
2. ÖZET	V
3. INTRODUCTION	VI
4. LITERATURE REVIEW	VII
5. METHODOLOGY	VIII
5.0.1. The YARP Arcitechture	VIII
5.0.2. The iCub Cognitive Architecture	IX
5.0.3. iCub Simulation Environment	XI
5.0.4. Mechanics of iCub	XI
6. RESULTS	XIII
7. CONCLUSION	XIV
7.0.5. Object Representation	XIV
APPENDIX A: Sample Code Snippet	XV
APPENDIX A: Screenshots	XVI
Bibliography	XVII

LIST OF FIGURES

Figure 5.1. Cognitive Architecture of iCub	VIII
Figure 5.2. YARP Command Line Interface	IX
Figure 5.3. Cognitive Architecture of iCub	X
Figure 5.4. iCub Simulation Environment	XI
Figure 5.5.	XII

LIST OF SYMBOLS/ABBREVIATIONS

YARP	Yet Another Robot Platform
DA	Description of abbreviation

1. ABSTRACT

Robot are evolving to the edge of the our knowledge. The software and hardware produced for them are disappears without leaving any signs. Exploring how to make robots are stable and long-lasting, without yielding our ability of changing sensors, hardwares and networks. Advancing over two fronts, software and hardware. For some time, we have been developing and using the YARP robot software architecture, which helps organize communication between sensors, processors, and actuators so that loose coupling is encouraged, making gradual system evolution much easier. YARP includes a model of communication that is transport-neutral, so that data flow is decoupled from the details of underlying networks and protocols in use. Importantly for the long term, YARP is designed to play well with other architectures. Device drivers written for YARP can be ripped out and used without any “middleware.” On the network, basic interoperation is possible with a few lines of code in any language with a socket library, and maximally efficient interoperation can be achieved by following documented protocols. These features are not normally the first things that end-users look for when starting a project, but they are crucial for longevity. We emphasize the strategic utility of the Free Software social contract to software development for small communities with idiosyncratic requirements. We also work to expand our community by releasing the design of our ICub humanoid under a free and open license, and funding development using this platform.

2. ÖZET

3. 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 proprioception¹. 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 physics and dynamics. Development is based on directly from first prototype of simulation environment Webots.² 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 physical interaction with the world. ODE³ is used in wide range of projects like GAZEBO⁴. ODE is an open source physics engine for authoring tools, computer games, etc. It uses OpenGL⁵ renderer and it has some disadvantages due to limitation of OpenGL engine computation efficiency on complex structures. iCub simulation is using OpenGL directly via SDL⁶ which helps to render complex robot movements and computation efficient simulation observations. Simulator is free and available to anyone who interested in robotics and learning about basics of robotics.

¹Body configuration and movement using accelerometers and gyroscopes

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

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

⁴GAZEBO : <http://gazebo-sim.org>

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

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

4. 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.

5. METHODOLOGY

5.0.1. The YARP Arcitechture

The computer simulation model of the iCub robot. Simulator allows to create realistic scenarios in where robot can interact with a virtual world and physical limitations and interactions that occur between the virtual world is simulated using open source library which is [ODE] (Open Dynamics Engine) to provide accurate simulation of body dynamics. Top of the [YARP]. It is a set of open source libraries that

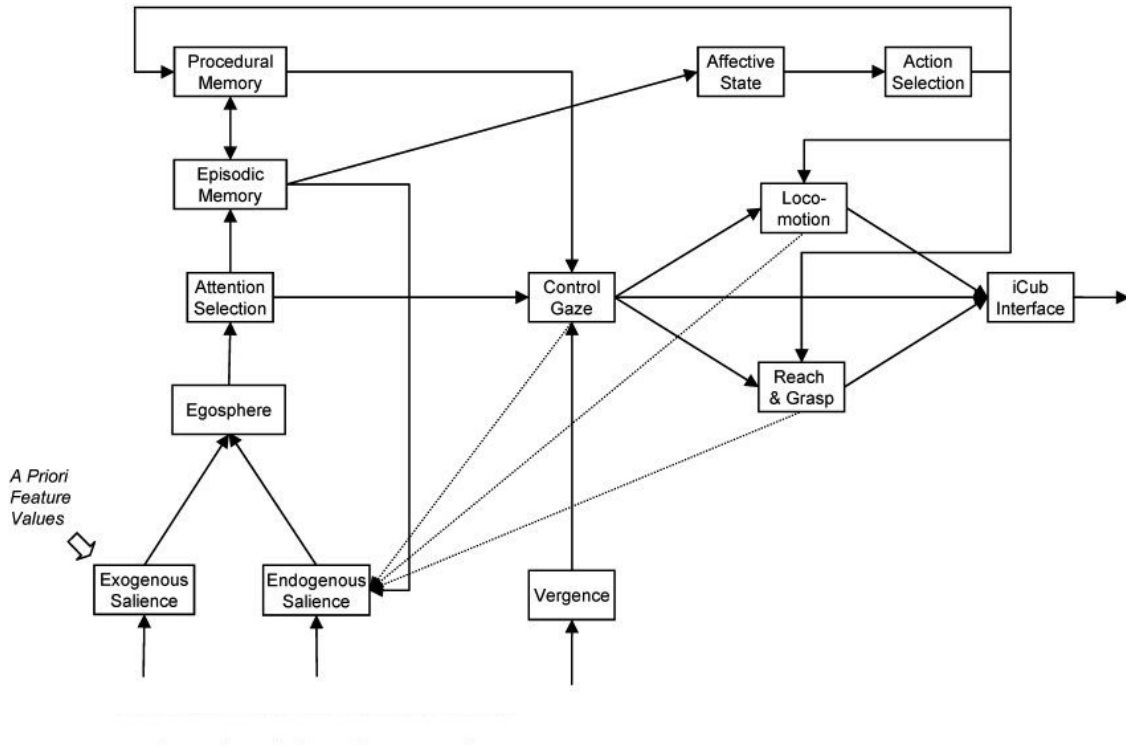
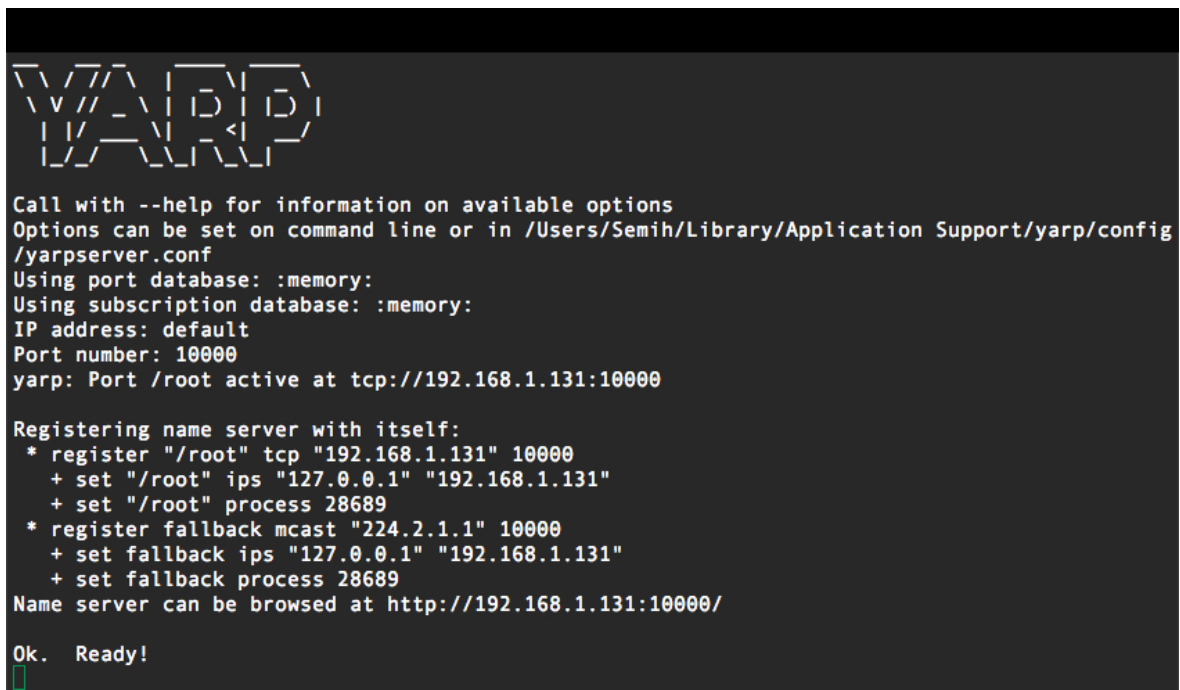


Figure 5.1. Cognitive Architecture of iCub

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] and [ACE]. 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 calls via linking suitable libraries to encapsulate hardware dependency problems. These two abstraction 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(Transmission Control Protocol) and UDP(User Datagram Protocol). The choice of protocol is depends on quality of message transmission or response time. Using TCP is for reliability and UDP is for speed with effect on unreliable transmissions.



```

YARP

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 5.2. YARP Command Line Interface

5.0.2. The iCub Cognitive Architecture

The design and realization of any cognitive architecture is a long-term project. The iCub cognitive architecture is no different. Inevitably, the realization happens in stages. In what follows, we describe a cognitive architecture which follows a significant subset of the roadmap guidelines to a greater or lesser extent. The goal of this preliminary cognitive architecture is to integrate some of the phylogenetic capabilities identified in Chap. 6 in a way that is meaningful for both neurophysiology and developmental psychology. In other words, the current goal is to build a minimal but faithful functioning system as a proof of principle. Sect. 7.3 discusses the degree to which each guideline has been followed and Chap. 8 considers the challenges posed by a complete implementation of the roadmap guidelines. The basis of the main conclu-

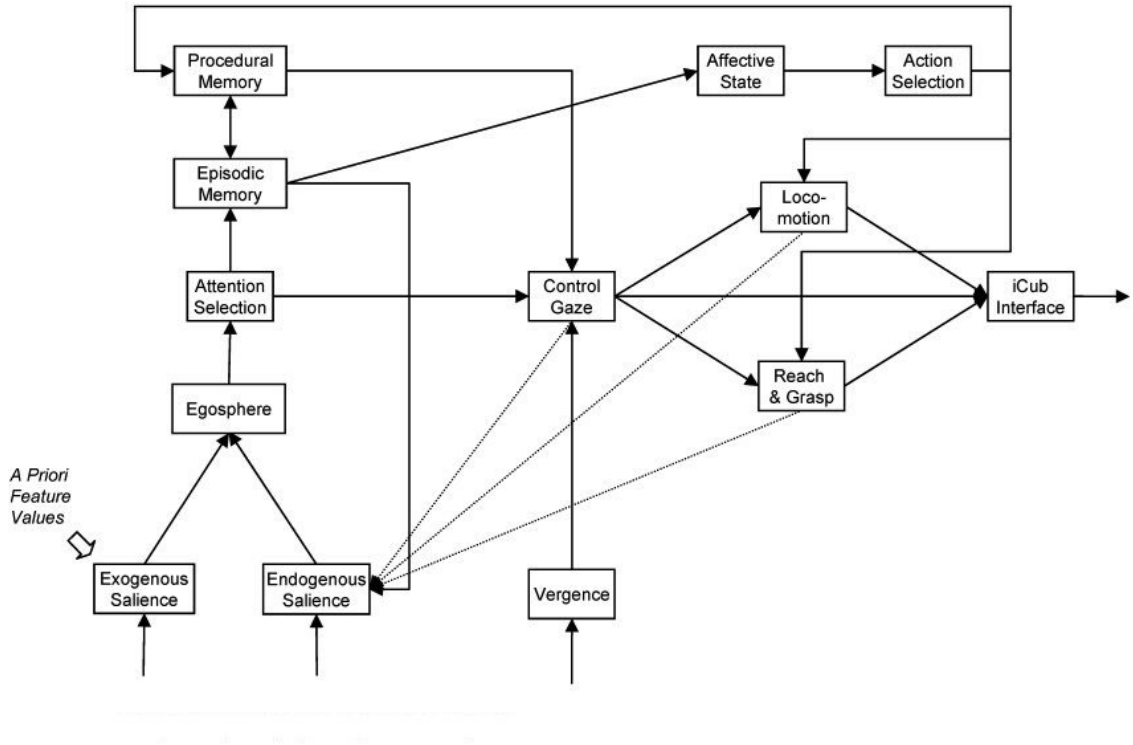


Figure 5.3. Cognitive Architecture of iCub

sions, decided to research on some capabilities. Gaze control, reaching, and locomotion constitute the initial simple goal-directed actions. Episodic and procedural memories are included to effect a simplified version of internal simulation in order to provide capabilities for prediction and reconstruction, as well as generative model construction boot- strapped by learned affordances. In addition, motivations encapsulated in the system's affective state are made explicit so that they address curiosity and experimentation, both explorative mo- tives, triggered by exogenous and endogenous factors, respectively. This distinction between the exogenous and the endogenous is reflected by the need to include an attention system to incorporate both factors.

5.0.3. iCub Simulation Environment



Figure 5.4. iCub Simulation Environment

5.0.4. Mechanics of iCub

[David Vernon, *The iCub*]The kinematic specifications of the body of the iCub including the definition of the number of DOF and their actual locations as well as the actual size of the limbs and torso were based on ergonomic data and x-ray images. The possibility of achieving certain motor tasks is favored by a suitable kinematics and, in particular, this translates into the determination of the range of movement and the number of controllable joints (where clearly replicating the human body in detail is impossible with current technology). Kinematics is also influenced by the overall size of the robot which was imposed a priori. The size is that of a 3.5 years old child (approximately 100cm high). This size can be achieved with current technology. QRIO1 is an example of a robot of an even smaller size although with less degrees of freedom. In particular, our task specifications, especially manipulation, require at least the same

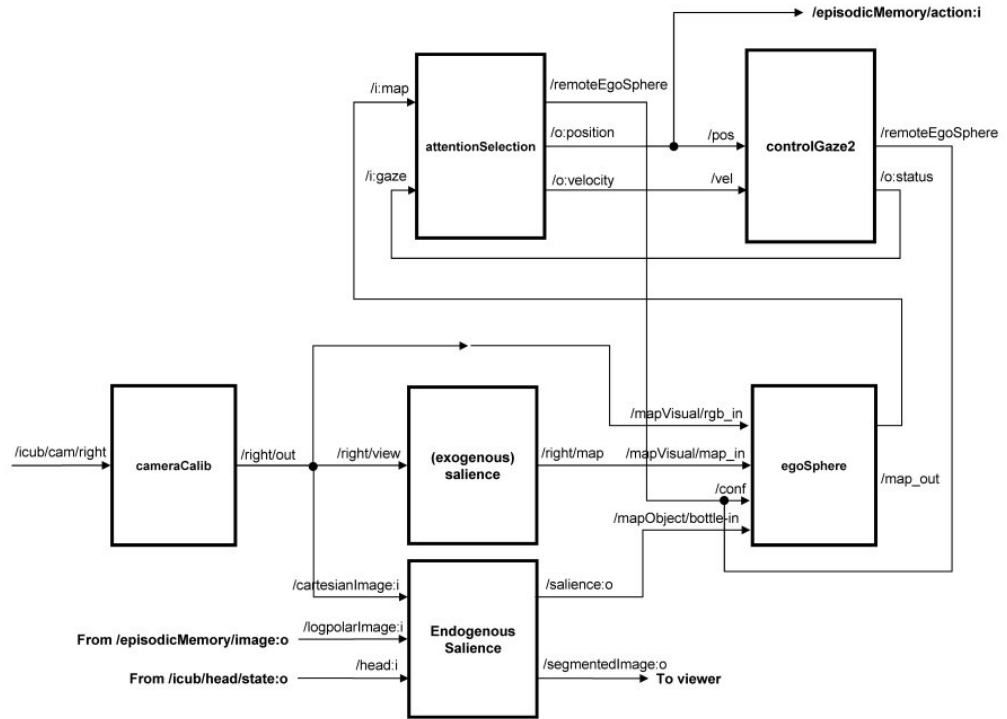


Figure 5.5.

6. RESULTS

AŞSLFKASF SF S F ASF sa f sf s F sdf asd g asdg

7. CONCLUSION

Prediction, Reconstruction, and Action: Learning Affordances Every action entails a prediction about how the perceptual world will change as a consequence of that action. This is the goal of the action and it is what differentiates an action from a simple movement or sequence of movements. Equivalently, every pair of perceptions is intrinsically linked or associated with an action. So, if we think of a perception-action-perception triplet of associations (P_i, A, P_j) , we can effect simple prediction, reconstruction (or explanation), and action as associative recall by presenting (P_i, A, \tilde{a}) , (\tilde{n}, A, P_j) , or (P_i, \tilde{a}, P_j) , respectively, to the iCub's Procedural Memory and by associatively recalling the missing element. This triplet-based representation, is conceptually identical to the stand-alone iCub framework for learning object affordances [256, 257, 258]. What differs is the manner in which the association network is constructed. In the latter framework, learning is accomplished by autonomous exploration using elementary actions that allow the iCub to experiment with its environment and to develop and understanding the relationships between actions, objects, and action outcomes, modelling these relationships using a Bayesian network. Affordances are represented by a triplet (O, A, E) , where O is an object, A is an action performed on that object, and E is the effect of that action. is the predictive aspect of to integrate this affordance learning technique with the Procedural Memory and Episodic Memory components in the iCub cognitive architecture. A significant problem remains, however. In the existing framework, the actions that the robot uses to experiment with and explore the object are assumed to exist as predefined primitive manipulation movements, such as push, tap, and grasp. Clearly, we require a more flexible approach in which the action's movements can be generated and adjusted adaptively as a consequence of the outcome of the action.

7.0.5. Object Representation

At present, there is no explicit concept of objecthood in the iCub cognitive architecture in the sense of Guideline 15, and there are no visual processes which identifies such objects. As we noted in the previous chapter, the implementation of this capability is far from trivial but, in principle, it doesn't pose a major challenge using conventional computer vision techniques based on motion segmentation and boundary detection. However, it would be instructive to investigate a more active approach based on the characteristics of overt attention, since attention mirrors interpretation and it plays such a significant part in cognition. Arguably, and consistent with Guideline 15, parts of a visual scene assume objecthood when they present a persistent and stable pattern of salience. This stable pattern of salience can be encapsulated by a repeatable localized eye gaze scan path pattern and represented by a given $(P, A_i, P_i \dots A_j, P_c)$ clique within the network of associations in the procedural memory. Object detection and recognition then becomes a matter of associative clique retrieval based on all or part of the clique.

APPENDIX A: Sample Code Snippet

```

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_trajectory
time").asDouble();
maxPitch = rf.check("maxpitch",Value(30.0),"Torso_max_pitch").asD

//get which arm to use. default to left if they didnt pass in left
armname = rf.check("arm", Value("left"),"arm_name").asString().c_
if (armname == "right") {
armInUse = true;
}
else {
armInUse = false;
}

//get which robot target to use
robotname = rf.check("robot", Value("icub"),"robot_name").asString
}

```


APPENDIX A: Screenshots

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://gazebo-sim.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 21 March 2007.