Aquila: An Open-Source GPU-Accelerated Toolkit for Cognitive Robotics Research

Martin Peniak, Anthony Morse, Christopher Larcombe, Salomon Ramirez-Contla and Angelo Cangelosi

Abstract—This paper presents a novel open-source software Aquila developed as a part of the iTalk and RobotDoC projects. This software provides many different tools and biologically inspired systems that are useful for cognitive robotics research. Aquila addresses the need for high-performance robot control by adopting the latest parallel processing paradigm based on the NVidia CUDA technology. The software philosophy, implementation, functionalities and performance are described together with three practical examples of selected modules.

I. INTRODUCTION

Recent approaches that attempt to understand the nature of cognition have shifted their focus from emphasising formal operations on symbols to a rather different approach where cognition is seen as an embodied or situated activity and therefore largely determined by the physical form of an embodied system [1][2][3]. Artificial intelligence, developmental psychology, neuroscience, and dynamical systems theory have directly inspired a completely novel approach called developmental robotics, which is a highly interdisciplinary subfield of robotics also known as epigenetic or ontogenetic robotics [4][5][6][7].

Building artificial cognitive systems not only advances the current state of the art in the field of artificial intelligence but also provides us with insights into many different aspects of human capabilities. In addition, by developing cognitive models inspired by empirical observations it is possible to reveal problems and loopholes in our current models of the brain and cognition. Artificial cognitive systems based on this approach need to undergo an autonomous and gradual mental development from "infancy" to "adulthood". Interaction with their environments in an autonomous manner with little or no human intervention is necessary where no tasks or goals are pre-defined. This aids the process of achieving a good level of environmental openness where an artificial cognitive system is able to cope with different and previously unexpected environments. Another important aspect of the systems based on developmental robotics is that they are able to learn from their previous experience and use it to assist the acquisition of new skills. Developmental robotics attempts to understand how the control system's organisation of a single robot develops through various experiences over time.

Embodiment plays a significant part for achieving iTalk’s goals to develop autonomous artificial embodied agents capable of acquiring complex behavioural, cognitive and linguistic skills through individual and social learning. For example, it has been demonstrated that research in action and language learning in natural and artificial cognitive systems can directly benefit from this approach, inspired by the developmental systems and phenomena studied in children, allowing re-enactment of gradual process that have the potential to bootstrap various cognitive capabilities and integrate them into a unified interactive cognitive system [5][8][6].

The main theoretical hypothesis is based on the assumption that the parallel development of action, conceptualisation and social interactions permits the bootstrapping of language capabilities, which leads to the enhancement of cognitive development [9]. In other words, it is assumed that a complete cognitive system capable of sensorimotor coordination, object manipulation, imitation, goals understanding and others is required to allow the emergence of rich communication skills. This emerging ability to communicate is hoped to further bootstrap the development of cognition enabling more complex integration and transfer of knowledge, sensorimotor skills, construction of action categories, social learning, the acquisition of grounded conceptual representations as well as the development of grammatical structure of language.

The developmental robotics approach to action and language learning is consistent with recent brain-inspired approaches to mental development since computational neuroscience considers the neural development constrains on embodiment, as well as on cognition [10][11][12]. It is not surprising that an overwhelming number of studies from various fields suggest that actions and language are closely integrated together. This highlights the importance of embodiment as well as supports the hypothesis of usage based language as opposed to classical explanations of language development assuming an extensive language-specific cognitive hardwiring.

This integration requires complex and computationally intensive algorithms running in parallel while controlling high-performance systems. The processing requirements are increasing with every added feature and it is not uncommon that at the end of the software development stage a particular system is unable to cope with fast-response robot-control tasks. This is very likely when a system requires applying filters to millions of pixels from a robot’s cameras,

This work was supported by the EU Integrating Project - ITALK (214886) within the FP7 ICT programme - Cognitive Systems, Interaction and Robotics.
running large-size neural networks with millions of synaptic connections and using multiple self-organising maps while controlling the robot in the real-time.

In 2003, the energy consumption and heat-dissipation problems slowed down the progress and virtually all processor vendors have changed their strategy to manufacture chips with multiple cores, which had a strong impact on the software developer community [13]. In the meanwhile, manufacturers have been looking into new technologies that would increase the number of transistors per wafer. However, reducing these dimensions comes at its price since the current leakage becomes a problem [14] and the bizarre effects of quantum mechanics [15] have to be taken into account.

Since 2003, semiconductor industry have been divided into multicore and manycore design trajectories [16]. Manycore design aims to increase the processing power by increasing the number of cores in a processor. This number was doubling with each semiconductor process generation starting with dual-core chips and reaching hyper-threaded hexa-core systems. Manycore system is fundamentally different with regards to its design philosophy. While CPUs are optimised for processing of sequential code and feature sophisticated control logic and large cache memories, GPUs design philosophy emerged from the fast growing video industry where massive numbers of floating point operations are required to render every single frame. As a result, a GPU chip has most of its area dedicated to processing of the floating point operations and features only tiny cache memories.

In 2006, NVidia released GeForce 8800 GPU, which was capable of mapping separate programmable graphics processes to an array of GPUs, which paved the way to first general purpose computing using parallel GPU processors. GPGPU was an intermediate step where graphics card programmers had to use an OpenGL or DirectX API to implement their programs. Using GPGPU technique many different applications have achieved dramatic speed improvements. For example, Kruger and Westermann developed a framework for solving linear algebra[17], Harris and colleagues designed a cloud dynamics simulation based on partial differential equations[18], Rodrigues and colleagues implemented molecular dynamics simulation [19] and Nyland and colleagues N-body simulation [20].

However, this required the expert knowledge of these APIs and, in addition, all calculations needed to represent their inputs as textures while their outputs would be represented as a set of pixels generated through raster operations. This has changed during the development of Tesla GPU architecture when NVidia researchers realised that its potential could be much higher if one could think of GPUs like individual programmable processors. As a result, in 2007 NVidia released the CUDA (Compute Unified Device Architecure) programming model that was designed to support mutual CPU/GPU application execution.

Parallel computing using CUDA and NVidia cards is being increasingly taken up by industry and academies. Many commercial and research applications have migrated from using standard processors only to a collaborative CPU/GPU use where each architecture does what is best at. In general, most of these applications can achieve tremendous speed-ups in performance, which is anything between 1.3x to 2,600x [21]. Since quantum computing is still in its infancy and CPUs are approaching the processing limits constrained by the physical laws, it seems that parallel computing using GPU devices is the next paradigm that is yet to become fully recognised and widely used.

CUDA has been employed in a wide variety of applications, however, only a handful of these have any relevance for the developmental robotics domain. Some research that has been done on neural networks and visual processing showed promising results (e.g. [22][23]) but this field requires further investigation in applying the CUDA technology.

This paper presents a novel software named Aquila, developed as a part of the iTalk and RobotDoC projects, that makes use of the latest parallel processing paradigm based on the CUDA technology. The next section, that provides an overview of Aquila’s philosophy, implementation, functionalities and performance, is followed by three different examples of how some its modules can be used.

II. OVERVIEW

Aquila is an open-source project that was inspired by the recent advancements in supercomputing making use of GPU enabled devices such as those based on NVidia Fermi architecture resulting in a dramatic performance increase, which is nowadays a crucial aspect of any software running complex tasks.

The development of Aquila was driven by the increasing need for an open-source high-performance modular software that would provide not only a convenient interaction with the iCub humanoid robot and its simulator (see section II-A) but also multiple tools and bio-inspired systems (see section II-B) that are useful for cognitive robotics research.

Source code that implements functions running on CPUs is mostly written in C++ programming language. Parallel functions running on GPU cards are written in CUDA-C, which is an extension to C programming language providing access to the virtual instruction set and memory in CUDA-capable GPU cards. Aquila is based on freely available multi-platform libraries and currently compiles on Linux and Mac operating systems. The graphical user interface (GUI) is clearly structured and divided into several modules that can be easily changed using tabs at the top of the application (see Figure 1). Fast and efficient interaction with the iCub is facilitated through the integration of many useful modules in a simple and intuitive GUI based on Qt libraries. The communication with the iCub humanoid robot or its simulator is provided by the YARP protocol [24][25]. Image processing is implemented through OpenCV\textsuperscript{1} libraries as well as native functions. The subsequent rendering to the GUI is realised through the native use of OpenGL where the incoming images are mapped into textures and sent to

\textsuperscript{1}In September 2010, OpenCV announced GPU support [26]
Qt’s QGLWidget. 2D and 3D visualisations are based on QwtPlot and QwtPlot3D libraries respectively. Some of the modules use speech recognition, which is provided by the Julius engine.

![Image of Aquila](image_url)

Fig. 1. Top part of the image shows multiple tabs that are used for switching between different modules. All the other controls that are below the tabs are part of the Terminal module, which is the initial default module. The top part of the image shows system information, initialised zero-force control and face expressions interfaces. The bottom part provides information about iCub’s joint control modes that are currently used.

### A. iCub Humanoid Robot and Simulator

The iCub (www.icub.org) is a small humanoid robot that is approximately 105cm high, weights around 20.3kg and its design was inspired by the embodied cognition hypothesis. This unique robotic platform with 53 degrees of freedom (12 for the legs, 3 for the torso, 32 for the arms and six for the head) was designed by the RobotCub Consortium [27], which involves several European universities and it is now widely used by the iTalk project and few others. The iCub platform design is strictly following open-source philosophy and therefore its hardware design, software as well as documentation are released under general public license (GPL). RobotCub name is partially an acronym where **Cub** stands for Cognitive Universal Body and the initial funding for this project was 8.5 million from Unit E5 (Cognitive Systems and Robotics) of the European Commission’s Seventh Framework Programme.

Tikhonoff et al. have developed an open-source simulated model of the iCub platform [28]. This simulator has been widely adopted as a functional tool within the developmental robotics community, as it allows researchers to develop, test and evaluate their models and theories without requiring access to a physical robot.

### B. Modules

Aquila currently provides a number of modules most of which support CPU and GPU execution. Below is a brief description of each module and its functionalities. For a more detailed information about their implementation and use see Aquila manual (section V).

**Terminal** - provides relevant information about the external systems that Aquila interacts with, displays status messages about the presence of GPU devices, Julius speech recognition system, YARP server, the iCub robot and its simulator. **Terminal** provides way to set face expressions of the iCub either by using low or hi-level module and regardless whether a user is connected to the real robot or the simulator. This module integrates with the force control system developed by Italian Institute of Technology [29] and is able to start/stop compliant mode on iCub as well as change several parameters such as joints stiffness, damping and offsets. In addition, **Terminal** displays colour visualisation of the iCub’s joint control modes (e.g. position, velocity, impedance), provides different options such as lunching external applications or modifying Aquila’s global settings.

**Sequence Recorder** - provides a simple and convenient way of recording, saving and replaying sensorimotor sequences. This module uses the Zero Force Control interface to enable running iCub robot in a compliant mode while recording sequences.

**ESN Kinematics** - implements echo state networks.

**Modi Experiment** - runs attention system, speech module and provides a simple way to demonstrate the modi experiment (see section III-C).

**U-Shaped Curves in Development** - runs and visualises multiple self-organising maps that are connected by Hebian weights. This biologically inspired model provides one explanation of why children are better at recognising phonemes when they are 8 months old then their performance gets worse before it improves again later [30].

**Multiple Timescales Recurrent Neural Network** - trains complex continuous time recurrent neural networks with multiple timescales using the backpropagation through time algorithm. The module executes neural network control systems on the iCub robot or the simulator as well as visualises many parameters (see section III-B).

**Self-Organising Maps** - trains, saves, loads and visualises self-organising maps, which are very useful and powerful computational tools capable of preserving the topological relations in multi-dimensional data.

**Abstraction-Reaction Accumulator** - explores a novel approach to synthesising complex adaptive behaviour and non-task-specific control systems.

**Vision** - renders internally as well externally video streams from the iCub robot, its simulator as well as from up to two connected cameras. It allows user to take screenshots, record individual frames and videos, maximise and minimise individual viewports and apply various image processing filters.

**CUDA** - displays relevant information about CUDA devices found on the system during Aquila initialisation. This information can be very useful in aiding the process of setting GPU execution parameters (e.g. number of threads) that have dramatic influence on the performance.
C. Performance Benchmarking

The performance benchmarking was done in order to demonstrate the potential of using GPU devices. It is important to note that these are only preliminary comparisons of unoptimised GPU and CPU code. The performance can be farther increased for both architectures, however, the reason for demonstrating the GPU-CPU differences is to give a general feeling of the extent to which GPUs outperforms CPUs when code is not extensively optimised.

The multiple timescales recurrent neural network (MTRNN) system benchmark was done for the backpropagation through time (BPTT) algorithm (Table I), single forward pass through the network (Table II) as well as for self-organising maps (SOM) training (Table III).

These tests were performed on MTRNN and SOM modules using different parameters to show how both architectures scale when the amount of processing increases.

![Fig. 2. system setup used for benchmarking consisting of 8 x 2.67GHz hyperthreaded processors, 16GB RAM, 1 x GeForce GTX470 and 3 x Tesla c1060 GPU cards.](image)

<table>
<thead>
<tr>
<th>number of neurons</th>
<th>CPU time</th>
<th>GPU time</th>
<th>speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>336</td>
<td>11.14min</td>
<td>0.90min</td>
<td>12.31x</td>
</tr>
<tr>
<td>1104</td>
<td>171.41min</td>
<td>3.83min</td>
<td>44.64x</td>
</tr>
</tbody>
</table>

**TABLE I**

<table>
<thead>
<tr>
<th>number of neurons</th>
<th>CPU time</th>
<th>GPU time</th>
<th>speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>336</td>
<td>30ms</td>
<td>0.4ms</td>
<td>75x</td>
</tr>
<tr>
<td>1104</td>
<td>370ms</td>
<td>1ms</td>
<td>370x</td>
</tr>
<tr>
<td>4176</td>
<td>5022ms</td>
<td>5ms</td>
<td>1004x</td>
</tr>
</tbody>
</table>

**TABLE II**

<table>
<thead>
<tr>
<th>number of neurons</th>
<th>CPU time</th>
<th>GPU time</th>
<th>speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>64</td>
<td>0.22sec</td>
<td>0.20sec</td>
<td>1.14x</td>
</tr>
<tr>
<td>256</td>
<td>1.75sec</td>
<td>0.45sec</td>
<td>3.88x</td>
</tr>
<tr>
<td>1024</td>
<td>14.2sec</td>
<td>1.15sec</td>
<td>12.36x</td>
</tr>
<tr>
<td>4096</td>
<td>126.17sec</td>
<td>3.20sec</td>
<td>39.37x</td>
</tr>
</tbody>
</table>

**TABLE III**

The benchmarking results of unoptimised code show that achieved speedups vary from 1.14x to 1004x. It is clear that the performance of GPU devices stands out soon after a large amount of data needs to be processed. A good example of such scaling can be seen in the MTRNN forward pass times that vary from 75x to 1004x speedups.

III. Examples

This section provides brief descriptions of selected modules, their underlying systems as well as preliminary experimental results.

A. Abstraction-Reaction Accumulator

The Abstraction-Reaction Accumulator (ARA) is an experimental adaptive control system, inspired by early cybernetic work [31]. The model explores a novel approach to synthesising complex adaptive behaviour and non-task-specific control systems. The system is structured such that a growing repertoire of adaptive behaviours is observed to emerge when coupled to an appropriate environment. Through a gradual process of cumulative learning, behavioural reactions are associated with perceptual abstractions. The current Aquila implementation is specific to the embodiment of the iCub humanoid robot: abstractions are defined as low-dimensional states dependent on high-dimensional sensory data obtained from the physical or simulated iCub, such as joint encoder values or eye camera images; reactions are defined as states in the iCub robot that determine or describe observable behaviours, such as postures, joint velocities or joint positions. A shared vision module is used to obtain the raw camera images, while another shared module (iCubControl) is used to obtain proprioceptive joint-position data, and to move individual joints on the iCub robot.

Intermittent feedback from the environment, provided through the Aquila GUI, modulate parameters in the system such that certain reactions (components of overt behaviour) are more likely to occur in the presence of certain abstractions (sensory states). This feedback simulates perturbation to essential variables, which quantify the appropriateness of the state of specific groups of joints on the robot, referred to as synergies. Each multidimensional synergy has its own essential variable, and can take one of several different states at any one time. The number of synergies and a mapping between each joint and synergy is also specified by the user in the GUI. Any joint on the robot can be mapped and controlled by the ARA. When a perturbation to an essential variable occurs, a parameter (dimension of variation) corresponding to the present reaction and present state of abstraction will be modulated, such that future disturbance to that essential variable is minimised. For example, in a particular sensorimotor state, where a red ball is held above the ‘closed’ right hand of the robot, a tutor may perturb an essential variable corresponding to the synergy ‘right hand’ (a set of joints), positively or negatively, causing the stability of a relevant set of parameters in the system to change. As a result, the state ‘closed’ of the synergy ‘right hand’ will be more or less likely to be observed in future, depending on the polarity of the perturbation, if and when the sensory situation recurs.

The ARA module provides many configuration settings (see Figure 3) as well as visualisations of the continuously changing variables, allowing users to observe the state of the system in real-time (see Figure 4).
B. Multiple Timescales Recurrent Neural Network

Humans are able acquire many skilled behaviours during their life-times. Learning complex behaviours is achieved through a constant repetition of the same movements over and over while certain components are segmented into reusable elements known as motor primitives. These motor primitives are then flexibly reused and dynamically integrated into novel sequences of actions.

For example, the action of lifting an object can be broken down into a combination of multiple motor primitives. Some motor primitives would be responsible for reaching the object, some for grasping it and some for lifting it. These primitives are represented in a general manner and should therefore be applicable to objects with different properties. This capacity is known as generalisation, which also refers to the ability to acquire motor tasks by different ways.

In addition, one might want to reach for the object and throw it away instead of lifting it up. Therefore, these motor primitives need to be flexible in terms of their order within a particular action sequence. The amount of combinations of motor primitives grows exponentially with their number and the ability to exploit this repertoire of possible combinations of multiple motor primitives is known as compositionality.

The hierarchically organised human motor control system is known to have the motor primitives implemented as low as at the spinal cord level whereas high-level planning and execution of motor actions takes place in the primary motor cortex (area M1). The human brain implements this hierarchy by exploitation of muscle synergies and parallel controllers. These have various degrees of complexity and sophistication that are able to address both the global aspects of the motor tasks as well as fine-tune control necessary for the tool use [32]. Experiments conducted on animals are also consistent with these findings. For example, it has been shown that the electrical stimulation of primary motor and premotor cortex in monkeys triggers coordinated movements such as reaching and grasping [33]. Giszter and colleagues found that a frog’s leg contains a finite number of modules organised as linearly combinable muscle synergies [34].

Models such as MOSAIC [35] or mixture of multiple RNN expert systems [36] implemented functional hierarchies via explicit hierarchical structure where the motor primitives are represented through the local low-level modules whereas the higher-level modules were in charge of recombining these primitives using extra mechanisms such as gate selection systems. These systems, based on predefined hierarchical structures, were appealing because of their potential benefits. For example, learning of one module does not interfere with learning of other modules and it would also seem that by adding extra low-level modules the number of acquirable motor primitives would increase as well. However, it has been demonstrated that the similarities between various sensorimotor sequences result in competition between the modules that represent them. This leads to a conflict between generalisation and segmentation, since generalisation requires the representation of motor primitives through many similar patterns present in the same module whereas different primitives need to be represented in different modules to achieve a good segmentation of sensorimotor patterns. Because of the conflict that arises when there is overlap between different sensorimotor sequences, it is not possible to increase the number of motor primitives by simply adding extra low-level modules [37]. Learning of motor primitives (low-level modules) and sequences of these primitives (hi-level modules) need to be explicitly separated through sub-goals [38][36].

Yamashita’s multiple timescales recurrent neural network model [39] attempts to overcome the generalisation-segmentation problem through the realisation of functional hierarchy that is neither based on the separate modules nor on structural hierarchy, but rather on multiple timescales of neural activities that seem to be responsible for the process of
motor skills acquisition and adaptation as well as perceptual auditory differences between formant transition and syllable level [40][41][42]

Aquila implements the MTRNN system together with the backpropagation through time (BPTT) algorithm for both CPU and GPU architectures. As it is demonstrated in section II-C, GPU significantly improves the speed of the MTRNN activation as well as of the training algorithm.

Multiple timescales recurrent neural network model can be seen as an extension the continuous time recurrent neural network (CTRNN), which was successfully used for producing sensorimotor sequences [43][44][45]. The MTRNN used in the preliminary experiments [46] consists of input-output neurons and context neurons that have different decay rates. The input-output neurons receive sensory inputs from a self-organising map that transforms multidimensional vectors from the iCub robot into topological maps that directly set neural activations on this layer. Context neurons are divided into two categories (fast and slow neurons) where each category represents different timescales characterised by decay rate of neurons. In this study the MTRNN was fully connected except for no presence of direct connection between input-output neurons and ‘slow’ neurons.

In the initial preliminary study the MTRNN system was required to learn 8 different sequences of actions (slide left and right, lift up, left and right, swing, push and pull).

At the end of the training, the learned neural network was tested on the iCub in the same setup as during the tutoring part. The MTRNN system was found to be able to replicate all the eight sequences while successfully manipulating with the object.

C. ‘Modi’ Experiment

Our perception of continuous contact with a rich visual world laid out in front of us is somewhat misleading, in fact our actual sensory input is highly impoverished; visual acuity for example is focused on an area the size of a thumb nail at arms length. Sensorimotor theories suggest that this rich perception is constructed from knowledge of the sensory consequences of performing various actions, thus you can perceive a chair in the periphery of your vision because you can predict that if you look over there you will see the chair. Similarly identifying objects is not so much about processing static images, but rather comes from identifying a profile of manipulations and their consequences in the dynamics of interaction. Such embodiment centric accounts of perception are supported by a large number of psychology experiments exposing various bodily biases in categorisation.

As part of the ITALK project we are modelling these experiments both to develop the perceptual skills of the iCub robot, and to further understand our own categorisation abilities and our own bodily biases in perception.

In a series of experiments conducted by Linda Smith and Larissa Samuelson [47] children between 18 and 24 months of age are repeatedly shown two different objects in turn, one consistently presented on the left, and the other consistently presented on the right. After several presentations of the objects, the child’s attention is drawn to one side or the other and the linguistic label ‘modi’ is presented in the absence of either object. Finally the children are presented with both objects in a new location and asked to find the ‘modi’. Not surprisingly 71% of the children select the spatially correlated object.

In a follow up experiment following the same basic procedure one group of children are presented with only a single object which is labeled while in sight, the other group are repeatedly presented with a consistent spatial relationship until finally an object is labeled while in sight but in the wrong spatial location. In the control group 80% correctly pick the labeled object while in the spatial competition group the majority of 60% select the spatially linked object rather than the object that was actually labeled.

In both experiments changes in posture from sitting to standing eradicate the effect, while other visual or auditory distracters do not. This is strong evidence challenging the hypothesis that names are associated to the thing being attended at the time they are heard.

Our model consists of a number of self-organising maps capturing variations in visual, auditory, and body posture input. These maps are then linked together via the body posture map (acting as a hub) in real time based on the experiences of the robot. With the addition of motion detection in the periphery of the robots vision, causing the robot to look at moving objects or changes in the scene, we are able to replicate the psychology experiments using the robot.

The ‘modi’ module in Aquila gives access to the programs used to replicate these psychology experiments with the iCub robot and provides algorithms for motion detection, eye saccades and object tracking. Speech recognition is provided as is a key-press terminal connection should you wish to use your own speech recognisers. The specific experiment can be performed by following the 6 steps outlined below but the resulting behaviour is not limited to this sequence alone.

1) Object A is presented to the right
2) Object B is presented to the left
3) Steps 1 and 2 are repeated
4) The robots attention is drawn to the right with no

Fig. 5. tutoring the iCub robot while recording the sensorimotor sequences using the Sequence Recorder module of Aquila.

4) The robots attention is drawn to the right with no
IV. Future Work

We have briefly mentioned the motivation for the development of Aquila, described its modules and showed few example applications of their practical use. Despite of many features already present, Aquila is still in early development stages. To date, Aquila has five active developers who use different modules to develop their own artificial cognitive models. This reusability of components allows for rapid development of new modules, however, a lot of work still needs to be done with regards to programming of individual classes that need improvements and more testing. Many new features will be made available as we proceed with our research and optimisation of the existing systems, which will improve the stability of the application.

Current work focuses on porting Aquila to Windows, fixing known bugs, improving interface, extending the MTRNN module with vision and language systems to allow experiments of action and language acquisition. New module, available in the next version, will facilitate investigation of spatial representation with the focus on peripersonal space.

V. Software Repository and User Manual

Aquila can be downloaded directly from SourceForge or iTalk Project software repositories. The project page is on SourceForge (http://sourceforge.net/projects/aquila/) where new developers can join our team.

The user manual that provides detailed module descriptions and installation instructions is available from this link: http://dl.dropbox.com/u/81820/Software/Aquila/Aquila.pdf

REFERENCES


