

User-friendly Robot Environment for Creation of Social Scenarios

Tino Lourens¹ and Emilia Barakova²

¹ TiViPE

Kanaaldijk ZW 11, Helmond, The Netherlands

`tino@tivipe.com`

² Eindhoven University of Technology

P.O. Box 513, Eindhoven, The Netherlands

`e.i.barakova@tue.nl`

Abstract This paper proposes a user-friendly framework for designing robot behaviors by users with minimal understanding of programming. It is a step towards an end-user platform which is meant to be used by domain specialists for creating social scenarios, i.e. scenarios in which not high precision of movement is needed but frequent redesign of the robot behavior is a necessity. We show by a hand shaking experiment how convincing it is to construct robot behavior in this framework.

1 Introduction

A robot equipped with sensors and actuators is much more than a sensing, planning, and acting loop, as initially proposed in the 1980s, but merely a set of real time parallel processes that keeps this natural order of processing. The control software architecture used for these robots, i.e. the way to put these pieces together in a functioning system remains a challenging task. The present day software architectures need to cope with multiple processes running on different machines, often with different operating systems and hardware. Autonomous robots have the technical capacity to perform complex behaviors that can be beneficial to medical therapists, psychologists, designers, and educators. Despite of the potential of robotics in these areas, too much expertise is required today, and hence these robots have not yet been widely used.

Our goal is to overcome this problem by making a programming architecture that facilitates easy construction of complex behaviors and interaction scenarios. We address 3 main issues from both technological and user perspectives. The first main issue is that even the most basic robot competency requires a vast amount of software. Therefore we developed an architecture that allows a high degree of re-usability of behaviors on all levels- from simple movement primitives to behaviors and scenarios. Second, we address the problem of parallel control over the different robot actuators that can be controlled by simple text-like commands. Third, since the major potential users of the robots are people with no or a moderate technical background, the architecture is developed towards an

end-user platform, which implies usage of a graphical programming paradigm. The robot architecture is developed as a part of TiViPE visual programming environment, which is a powerful integration tool.

As a particular application we feature training social skills to autistic children [1,2,3]. In a broader sense, educators, medical therapists, psychologists, scenario and storyboard designers, and software developers are the potential users of the architecture. Hence, we need to take an end-user approach towards robot scenario construction.

Several theories of controlling robots like the subsumption theory of Brooks [4], or a three layered approach of reaction-control, sequencing, and planning, emerged [5,6]. A recent development is the ROS open source operating system for robotics [7]. ROS runtime graph is a peer-to-peer network of processes that are loosely coupled using the ROS communication infrastructure. ROS is a distributed framework of processes that enables executables to be individually designed and loosely coupled at runtime, and aims to support multiple textual programming languages. Yet Another Robot Platform (YARP) also supports building a robot control system as a collection of programs communicating in a peer-to-peer way, its main focus is on long-term software development [8,9]. Another platform is the Open ROBOT Control Software (OROCOS) with focus on thread-safe and real time aspects between tasks [10]. Orca, initially part of the Orocos project, is an open-source framework for developing component-based robotic systems. It provides the means for defining and developing the building-blocks which can be connected to form arbitrarily complex robotic systems, from single vehicles to distributed sensor networks [11]. The authors of Orca make a comparison between seven different robotics software systems [12], but also address a key issue in robotics that *the sheer amount of software necessary for even the most basic competency is quite large*. These factors make software reuse attractive, i.e., *one hopes to integrate existing software modules within a software framework*. TiViPE which is a graphical programming environment [13], emphasizes on the integration of existing software routines from (existing) libraries without additional programming. TiViPE also covers the aspects of multiple processes on multiple (embedded) computers, peer-to-peer communication, graphical programming, massively parallel (GPU) processing, and multiple operating system support.

Because of the mentioned reasons, we build further on TiViPE visual programming environment. From the perspective of the user of such an environment, our focus will be to create a modular structure of command blocks, as already present in some simple solutions (Choreagraph) with the possibility to design new or redesign existing behaviors by a simple textual language used within the command blocks.

To satisfy this purpose, an incremental approach, that is similar to sticking Lego[®] bricks together, is used. It implies that the modular components serve as partial scenarios that can be modified by changing the order of usage, by replacing, or adding of such modules. Using such an approach will enable scenario designers, educators, and therapists to get full control over the robot.

Scenarios need to be easily adapted for user tests and rapid prototyping, and should allow for an incremental approach of scenario development. A graphical language will be used to realize these requirements. We use TiViPE to construct a robot architecture since it naturally can incorporate most of the aspects described in this section. In addition a substantial set of modules for sensory data processing is available within TiViPE.

Section 2 describes a robot framework where a textual robot language is constructed with no more than 4 different control characters and 14 commands. This language provides full control over an advanced humanoid robot. Section 3 provides the robot sensory part. Most advanced robots provide audio and video data together with a set of values providing information about individual sensors. The following section describes how robot commands can be generated using and describe how the concept of using states is implemented using graphical components. Section 4 describes the work from the perspective of a user, gaining easy access to controlling a robot, without the need to have a background in robotics or software development. The paper ends with a summary and future work.

2 Robot software framework

The robot software framework includes the commands that interface with the robot, the language that insures non-conflicting operation of these commands by parallel actions of the robot and by multiple and conflicting action possibilities, and the visual programming environment that allows easy construction and visualization of the action stream.

2.1 Textual language

A textual robot language insures that neither in depth knowledge of a robot hardware is required, nor the behavior designer is confined to a specific robot. A minimalistic language, such that the robot commands can be executed in a serial or parallel manner is the best choice. Therefore we choose for a language structure that resembles adding and multiplying of symbols, where the symbols are equivalent to behavioral primitives:

$$(a + b) * c + d * e \quad (1)$$

Knowing that brackets bind stronger than multiplication, and multiplication has higher priority over addition, we can construct a powerful language that is in essence identical to the following statement from the formal logic:

$$[a \mid b] \& c \mid d \& e \quad (2)$$

In terms of robots commands, $a \mid b$ denotes that a and b are executed in parallel, $d \& e$ denotes that d and e are executed subsequently, first d then e , the square brackets denote the binding order of execution. Commands a , b , c , d , and e are

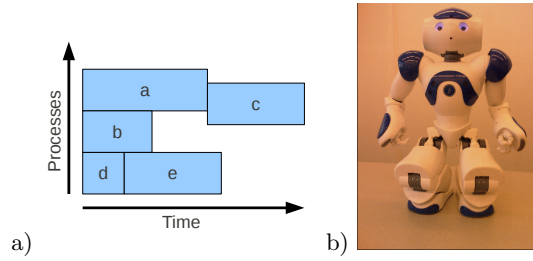


Figure 1. a) Graphical representation of 5 commands as given in (2). b) Application platform - humanoid robot NAO.

individual elementary commands that can be executed on a robot. The graphical representation of (2) is given in Figure 1a.

These individual commands depend on the abilities of a robot, and might seem to contain a substantial number of commands that control the robot. However, even on a multi-functional humanoid robot like NAO, see Figure 1b, only few commands are needed to gain full control over the robot, as it will be explained in Section 2.2.

2.2 Robot commands

Humanoid robot NAO can be controlled fully by as little as 7 different commands. However, for the ease of use, in our implementation we use 14 commands: two generic commands, two to enable or disable the LED's, two audio commands, and eight commands to get full control over all motors, as illustrated in Figure 2.

```
flush ()
wait (duration)ledset (name, value)
ledto (name, value, duration)
say (text[, volume[, language[, personality]]])
play (filename)
move (name, angle, duration[, stiffness])
movem (name, [angle, duration,]+ ...)
stiff ([stiffness, duration, idle,]+ ...)
walk (distance, duration)
walks (distance, duration)
walka (angle, radius, duration)
walkto (x, y, theta)
walkd (x, y, theta, frequency, duration)
```

Figure 2. Commands that provide full control over a robot.

Generic commands. Two generic commands `wait` and `flush` are available to enable the robot to synchronize its actions. The `wait` command lets the

robot await (do nothing) for a given duration period. The `flush` command is used rarely to remove all the residual processes that are scheduled and not in execution, yet. While normal commands are appended, the `flush` command is executed immediately.

Commands for controlling the LEDs. Activating and disabling the LEDs is performed with the `ledset` command, in this case all leds are reached through a unique name, and can be given a value between 0 (`ledoff`) and 1 (`ledon`). For transition from the current color to a new color the `ledto` (also known as `ledfade`) command is used and a duration in milliseconds is provided to control the speed of change.

Audio commands. Two powerful audio commands `say` and `play` can be used to accomplish either text to speech conversion or to play an audio signal from a mp3 player engine.

Motor commands. The stiffness of a motor is usually set at the beginning using the `stiff` command. A typical setting is for example `stiff (1.0, 1000, 0)` that sets all motors to maximum (1.0) stiffness in 1 second (1000ms) time. The movement of a single motor is defined by addressing the motor, and specifying the desired angle of change over a specific time interval. For example `move (HeadYaw, 30.0, 1000)` moves the 'head yaw' motor to an angle of 30 degrees in one second time. In practice multiple moves for multiple motors are performed through a string-like command `[move (HeadYaw, 30.0, 1000) & move (HeadYaw, -30.0, 1000)] | [move (HeadPitch, 20.0, 700) & move (HeadPitch, -20.0, 600)]`. This command can also be written in a more compact manner like `movem (HeadYaw, 30.0, 1000, -30.0, 1000, HeadPitch, 20.0, 700, -20.0, 600)` by using the `move multiple elements` command.

3 Creating behaviors with incremental complexity

The most simple robot behaviors are sequences of motor commands. Using sensory information requires specific design choices that bring another degree of complexity. Meta commands can combine behavioral sequences that are used often in a certain order. It may imply usage of states requires to meet a condition, before another behavioral sequence can be started. Hybrid commands bring yet different practical concerns. The main design choices for each level of complexity is discussed in the upcoming subsections.

3.1 Using sensory information

Using sensory information is crucial for constructing reactive or more complex autonomous behaviors. However, it requires issues as how to construct parallel behaviors that have different cycle (or update) speeds. Moreover, there is a huge difference between the capacity of a robot to perceive (read sensory information) and act behaviors - the behavioral capacity is around two orders of magnitude smaller than the sensory capacity.

Behavior such as letting the robot walk requires already a sensory input, for instance, it needs to be sure that it is standing. In case the robot is lying down it first needs to get up. In order to decide which command to execute we need to get sensory information first.

To do so, a 'RobotSense' module is constructed in a way that it reads out all relevant sensory information from an a-priori known robot at a fixed, cyclic update speed, which is set by the user. Reading sensory information from a robot usually is a relatively trivial task, however, sensory information is obtained at different frequencies. Images in common can be obtained at a maximum rate of 30 frames per second, while internal data can be obtained at higher rates, for instance at 10 or 20 milliseconds rather than the 33 for video. Latter might form a bottleneck on bandwidth which might in turn lead to latency and a lower frame rate. Hence, audio, video and single valued information are decoupled, as illustrated in Figure 3.

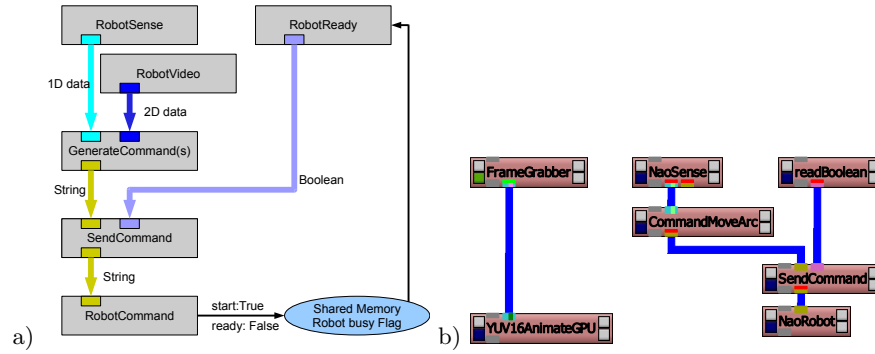


Figure 3. a) Global processes and flows of a command generator and a robot module. b) TiViPE implementation of a) using a NAO robot and automated getting up before walking (in a circle).

A robot needs several seconds to get up, while less is needed for reading out sensory information. To avoid generating commands that the robot is not able to complete in time, a moderator process, called 'SendCommand' (see also Figure 3) is introduced. This process sends a command to the robot if all former commands are accomplished.

This so-called 'RobotCommand' (for NAO it is NaoRobot) processes the given commands. Switches to busy mode until it is ready signalling the 'read-Boolean' module that its ready to receive.

This concept makes continuous sensing, and providing the latest command string on this basis possible without the need of specific knowledge on robot sensing or commanding. Hence, the focus is shifting to scenario's on basis of textual commands that are driven by internal information gathered over time and sensory data, hence one can focus on the 'GenerateCommands' type of modules.

3.2 Hybrid command structure

The textual robot language used provides an architectural advantage that the output of every command module is a string. Most of these modules will have a string as input as well, making it very easy to connect these modules. Since the language gives solely the option to connect new commands in a serial or parallel way string command generation is straight forward. It means that command generation becomes merely a choice of selecting a sequence of commands. The graphical tool allows multiple state sequences, combined with regular commands like 'Say', 'Wait', or its generic command 'GenerateCommand' that converge to a single string using the 'Serial' or 'Parallel' commands.

The graphical tool allows for merging modules to a single one, to get a layered abstraction for commands and command groups. On the other hand it is also possible to open and edit the compound structure of a merged module. The focus in the near future will be on the construction of these robot command modules on different levels of abstraction. Due to the modular concept of this approach an incremental abstraction and refinement of robot behavior will be constructed.

4 Example of social behavior creation - user point of view.

In the previous sections a framework for parallel robot control with text-like commands has been given. Although simple from a technical point of view, the creation of robot behaviors has where a user solely needs to select graphical blocks and connect them such that they provide the right sensory or state information and generate a text string to command the robot. In case such a graphical block does not exist, the user needs to develop a routine call that evaluates relevant sensory information and generates a command string as described in Figure 2, keeping in mind that multiple commands can be placed between square brackets ([]), and are either separated by a | for parallel or a & for serial execution. Users are able to compound within TiViPE a set of these blocks to a single block without additional programming.

The created framework allows a scenario designer to decide what blocks are needed and in collaboration with a developer to construct a set of useful graphical components of more complex robot behaviors or scenarios. When these blocks are available the users solely need to connect some of these modules, eventually these modules might have internal parameters that can be modified.

A student designer was used as a test person to see how the social scenario creation that includes robot implementation will be accomplished. He chose to make a hand shaking behavior between a robot and a child. He first got acquainted with the TiViPE and the robot embodiment. Subsequently, he explored the degrees of movement of the different joints of the robot, and got acquainted with the starting and end angles of joint movements. In parallel he did study on human motion organization - how each movement that is a part of a hand shaking interaction behavior is constructed and organized. For this purpose the

student filmed different test persons performing the handshaking interaction and concluded that most of them would approach the partner, have a brief glance to the meeting points of the hands, and perform the handshaking.

To capture a realistic human movement, he used the following sequence of actions: (1) The usage of markers: markers were used to stick on actors arm; (2); The usage of coordinate: a 3-dimensional coordinate was been put to calculate the directions of each movement; (3) The usage of stick model: the stick model would also be used in this recording process, all human bodies of actors were replaced by stick model; (4) The observation: the observation was taken by both video camera and monitor. The observation was mainly aimed to observe some other small changes during the hand shaking process, such as head movements, eye movements and body movements.

Based on this preparatory work, the handshake behavior was created within an hour. The handshake behavior contains 8 serial movem commands the third and fourth are as follows:

```
movem(RShoulderPitch, 33, 200, 33, 800, 32, 200,
      RElbowRoll,      38, 200, 35, 600, 36, 200, 35, 200,
      HeadPitch,        20, 800,
      HeadYaw,          -25, 800)
&
movem(RShoulderPitch, 33, 200, 33, 600, 30, 400,
      RElbowRoll,      36, 200, 36, 200, 35, 200, 32, 400, 33, 200,
      RHand,           57, 400, 20, 600,
      HeadPitch,        15, 600, 0, 600,
      HeadYaw,          -20, 600, -10, 600)
```

When completed, a user test of handshaking behavior was made in Maria elementary school in Helmond, The Netherlands. The children could naturally understand and perform the handshaking behavior without further guidance. Figure 4 shows some images from the user testing. Figure 4a shows the view from a side camera. Figure 4b and 4c depicts 2 snapshot views of the robot during handshaking interaction.

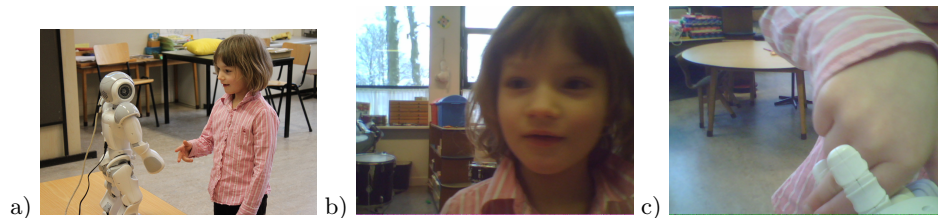


Figure 4. a) Child shaking hand with robot. b-c) View from robot perspective.

5 Summary and Future Work

A key issue in robotics is that the amount of software necessary for even the most basic competency is quite large. Framework TiViPE has been extended with robotics library to substantially reduce the amount of coding necessary to control a robot.

The robot control platform is aimed to develop to an end-user platform. So far, a user-friendly architecture has been developed, which makes possible non-specialists to co-develop useful behaviors together with a person with some knowledge in programming. The future user of the robots is often not a programmer or a person with keen interest in robotics, hence we need to provide easy to use building blocks for these users to provide full control over a robot. This is done by providing ready graphical components for sensing the internal values of a robot as well as video and auditory modules. Further a single component is used to command the robot. No more than a single module for commanding a robot is required since a textual robotics language has been introduced with an a-priori known list of commands. This yields a simple but powerful language to gain full control over a robot. In this framework constructing scenario's has become an interplay between reading out sensory information and based upon this information generating a command string to control a robot. Creating these blocks have become entirely trivial, and in that respect we have shifted the complexity to creating scenario's where numerous blocks are involved.

So far the framework has been tested by design students and professionals that have created behaviors within TiViPE [14,1] and image analysis for robot control [15,16]. Our future focus will be on scenario's and a layered component infrastructure, here the essence is to compound a set of blocks into meaningful partial sub-scenario's. Compounding itself is done without any addition programming with graphical programming environment TiViPE. Hence, once these sub-scenarios are created it will be easy for users to combine several sub-scenarios. For the particular application we are working on, we envision that therapists can do this to create training scenarios for autistic children. The scopes of application of such a generic framework, however, extends from exact control to applications in behavioral robotics and social robotics where issues of uncertainties need to be addressed.

Acknowledgments

We gratefully acknowledge the support of WikiTherapist project by the Innovation-Oriented Research Programme 'Integrated Product Creation and Realization (IOP IPCR)' of the Netherlands Ministry of Economic Affairs.

We also thank Ph.D. students, J. Gillesen and S. Boere, both supported by the WikiTherapist project, for developing novel robot behavior modules within TiViPE.

References

1. E. I. Barakova, T. Lourens, Expressing and interpreting emotional movements in social games with robots, *Personal and Ubiquitous Computing* 14 (5) (2010) 457–467.
2. E. I. Barakova, J. Gillessen, L. Feijs, Social training of autistic children with interactive intelligent agents, *Journal of Integrative Neuroscience* 8 (1) (2009) 23–34.
3. E. I. Barakova, W. Chonnaparamutt, Timing sensory integration for robot simulation of autistic behavior, *IEEE Robotics and Automation Magazine* 16 (3) (2009) 51–58.
4. R. Brooks, A robust layered control system for a mobile robot, *IEEE Journal of Robotics and Automation* 2 (1) (1986) 14–23.
5. R. J. Firby, Adaptive execution in complex dynamic worlds, Ph.D. thesis, Yale University (1989).
6. E. Gat, Three-layer architectures, in: *Artificial intelligence and mobile robots: case studies of successful robot systems*, MIT Press, Cambridge, MA, USA, 1998, pp. 195–210.
7. M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, A. Y. Ng, Ros: an open-source robot operating system, in: *ICRA Workshop on Open Source Software*, 2009.
8. G. Metta, P. Fitzpatrick, L. Natale, Yarp: Yet another robot platform, *International Journal of Advanced Robotic Systems* 3 (1) (2006) 43–48.
9. P. Fitzpatrick, G. Metta, L. Natale, Towards long-lived robot genes, *Robotics and Autonomous Systems* 56 (1) (2007) 29–45.
10. P. Soetens, A software framework for real-time and distributed robot and machine control, Ph.D. thesis, Department of Mechanical Engineering, Katholieke Universiteit Leuven, Belgium (May 2006).
11. A. Brooks, T. Kaupp, A. Makarenko, S. Williams, A. Oreback, *Software Engineering for Experimental Robotics*, Springer Tracts in Advanced Robotics, 2007, Ch. Orca: a component model and repository, pp. 231–251.
12. A. Makarenko, A. Brooks, T. Kaupp, On the benefits of making robotic software frameworks thin, in: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2007)*, 2007.
13. T. Lourens, Tivipe – tino’s visual programming environment, in: *The 28th Annual International Computer Software & Applications Conference, IEEE COMPSAC 2004*, 2004, pp. 10–15.
14. T. Lourens, R. van Berkel, E. I. Barakova, Communicating emotions and mental states to robots in a real time parallel framework using laban movement analysis, *Robotics and Autonomous Systems* doi:doi:10.1016/j.robot.2010.08.006.
15. T. Lourens, E. I. Barakova, My sparring partner is a humanoid robot – a parallel framework for improving social skills by imitation, in: J. R. Álvarez (Ed.), *IWINAC 2009*, no. 5602 in *Lecture Notes in Computer Science*, Springer-verlag, Santiago de Compostella, Spain, 2009, pp. 344–352.
16. T. Lourens, E. I. Barakova, Retrieving emotion from motion analysis: In a real time parallel framework for robots, in: C. S. Leung, M. Lee, J. H. Chan (Eds.), *ICONIP 2009*, no. 5864, Part II in *Lecture Notes in Computer Science*, Springer-verlag, Bangkok, Thailand, 2009, pp. 430–438.