Multi-agent Behavior Composition through Adaptable Software Architectures and Tangible Interfaces

Gabriele Randelli[1], Luca Marchetti[1],
Francesco Antonio Marino[2], and Luca Iocchi[1]

[1] Department of Computer and System Sciences,
Sapienza University of Rome,
Via Ariosto 25, 00185 Rome, Italy
name.lastname@dis.uniroma1.it
[2] Mathematics Department,
Tor Vergata University in Rome,
Via della Ricerca Scientifica snc, 00133 Rome, Italy
marino@mat.uniroma2.it

Abstract. Cooperative behavior realization is an important aspect of Multi-Agent Systems, and it has been widely addressed by the robotics community. However, the interaction among multiple robots and a human operator still requires a non-negligible effort to be effective. The availability of modern input devices can ease the behavior composition process, allowing a user to train Multi-Agent Systems by using alternative methods. In this paper, we introduce a novel approach to integrate Tangible User Interfaces within a reconfigurable software architecture, for cooperative Multi-Robot Systems. To address a real test case, we implemented a strategy training system for a humanoid RoboCup soccer team. Such a system allows a human coach to train several robotic players by using multiple input interfaces, directly onto the application field.

Keywords: tangible user interfaces, multi agent systems, humanoid robots, behavior composition, system architectures.

1 Introduction

Behavior composition is a key task for multi-robot systems in several applications, such as RoboCup soccer or *Urban Search And Rescue*. Because of the intrinsic complexity of this problem, the role of human operators in defining complex behaviors is often fundamental, and it is cognitive demanding. In fact, managing several robots through traditional user interfaces is neither comfortable nor scalable. On the one hand, interfaces are typically overwhelmed by a huge amount of data, thus preventing from an easy assessment. As well as innovative interfaces, another benefit consists of distributing this process among different operators. This requires a novel class of robotic software architectures

J. Ruiz-del-Solar, E. Chown, and P.G. Ploeger (Eds.): RoboCup 2010, LNAI 6556, pp. 278–290, 2011.

for supporting a multi-operator/multi-robot paradigm. The main requirement of such a system is adaptability, in order to manage multiple operators with different input devices, acting in different scenarios with different robotic platforms. Furthermore, flexibility is also crucial, to dynamically tune the system according to run-time changes in the robotic team, in the environment, or in the task to accomplish.

In this paper, we propose a new methodology for the problem of behavior composition in Multi-Agent Systems, through the use of Tangible User Interfaces and designing an adaptable and flexible software architecture to integrate their usage with different robotic platforms and other input devices. Furthermore, we provide a very preliminary evaluation of our system, which will be properly discussed and extended and in some future work.

A *Tangible User Interface* (TUI), is a user interface in which a person interacts with digital information through the physical environment. TUIs provide a high-level interaction paradigm, for example, through natural gestures. Furthermore, they are more comfortable with respect to traditional GUIs, as do not force the operator to switch her attention from the ongoing mission to the interface. Our objective is to develop a methodology to define behaviors for multi-agent systems through a Wiimote device, which allows for a direct interaction with the robots on the field.

We do aim at integrating TUIs with several other input devices, and interacting with heterogeneous platforms in heterogeneous environments (both real and simulated), without reconfiguring the whole system when a change occurs. We designed a flexible architecture that supports classic input methods, as well as innovative ones, and that codifies, through a well defined semantics, every input command into actions for different platforms: real wheeled robots, real humanoid robots, as well as simulated environments, such as Player/Stage[1], USARSim[2] and Webots[3].

The remainder of the paper is structured as follows. In Section 2, we address the problem of behavior composition for multi-agent systems and present some related works about tangible user interfaces. Then, in Section 4 we introduce the Wiimote device and our interaction system, while Section 5 describes our high-level command architecture. Finally, in Section 6 we evaluate the overall system implementing a strategy virtual coaching application for a humanoid RoboCup soccer team, and we conclude with some discussion in Section 7.

2 Related Work

Tangible user interfaces take advantage from human spatial orientation and interaction with tangible objects, two fundamental skills for human interaction in everyday activities. In fact, cognitive studies [5] have assessed how gesturing, as well as speeching, is one of the first learnt communication means (at around 10

[1] `http://playerstage.sourceforge.net`
[2] `http://usarsim.sourceforge.net`
[3] `http://www.cyberbotics.com`

months of age). Concerning a comparison of TUIs with respect to classic input methods, one interesting study is by Guo and Sharlin [3], who evaluated the performance of a Wiimote TUI and a classic keypad through navigation and posture tasks, using a Sony AIBO robot. The results evinced that a tangible device reduces the user thinking time for the correct movement, since they correspond to innate actions performed everyday. However, this does not imply that a TUI always guarantees a better performance with respect to other input devices. Analogous results are presented in the work by Song et al. [8], where they compared four types of devices: a pad-like device, a joystick, a driving device, and a Wiimote equipped with motion sensors. Varcholik et al. [9] address how TUIs provide two novel interaction methods: motion steering-wheel style control or gestural control. The former consists of using the sensors feedback to directly control the locomotion of the robot, while the latter recognizes gestures and associates them to high-level commands through machine learning and pattern recognition techniques. As for this latter, Mlích [6] defined a gesture recognition algorithm based on hidden Markov models that detects basic gestures, such as circles, squares, and so on. However, all these works present a single human-single robot interaction paradigm (SHSR) and, to our knowledge, TUIs have never been adopted with multi-agents systems.

3 Problem Space

A critical component for the success of a Multi-Agent System is the ability to design and implement complex behaviors that involve all the robotic agents.

To achieve an effective result for a team of agents, it is important to identify a global target that should be pursued by the team itself. The global goal should be decomposable in sub-problems, each one carried on by a single agent. The role of the human designer is to identify the intrinsic characteristics of a global task, and then to develop enough sub-tasks to be assigned to the team player.

We assume that such a process of problem analysis is already done, and it is available, to the human operator, as a set of simple behaviors that can be assigned to the agents.

Giving a real example, let us consider the scenario of a robotic soccer team. The global goal of the team is to win the match. If the rules of the match are similar to the real soccer ones, we can identify distinct roles for the players. There would be a *goalie*, some *defenders* and some *attackers*. A robotic coach should, then, dynamically assign a role to each robot, depending on the situation. In autonomous robotic teams the role assignment could be done by the robot itself, directly on board.

How robots interact each other, which areas have to be covered by which robot, are complex tasks that require the interaction with the human operator. In most cases, this is done by trying different configurations of role assignment and covered areas. At the end of several trials, the user can encode many tactics inside some sort of structure (robot-friendly) and the robot-agent will select the most appropriate one.

The problem of defining the interaction between human operator and a team agent is the main focus of this paper. We believe it is possible to relief the burden of the user to interact with complex languages for defining tactics.

Therefore, the assumptions we made on this paper are:

- a number of predefined and simple behaviors for single agent are already available;
- it exists some kind of languages that can be used for composing "atomic" behaviors;
- the system can select multiple behaviors and check if there is any consistency problem (it should verify if there is any incompatibility among selected behaviors);
- the user can execute a single configuration of the composed behaviors and be able to compare the results.

4 Wiimote Interaction System

4.1 Wiimote Technical Characteristics

The TUI we adopted in this paper is a Wiimote [11], reported in Figure 1. It is a commercial-off-the-shelf (COTS) inexpensive remote input device, which provides innovative sensing and feedback functionality, and communicates through the Bluetooth protocol architecture. The Wiimote is equipped with eleven buttons, two types of sensors, and three feedback controllers. As for the sensors, there is a three-axis ADXL330 linear accelerometer that measures the acceleration in a free fall frame of reference, providing the device with motion sensing capabilities. As well as the accelerometer, the device has an IR camera that acquires beacons coming from a Sensor Bar with ten infrared LEDs, five at each end of the bar. The Sensor Bar allows to use the Wiimote as a pointing device, since through triangulation methods it is possible to retrieve the distance between the Sensor Bar and the Wiimote, and to track the position onto the screen where the device is pointing. Concerning the feedback functionality, the Wiimote incorporates four blue LEDs, a rumble feature, and a small low quality speaker which emits short sounds. It is also provided an expansion port at the bottom, to plug in extension controllers, such as the Wii Motion Plus.

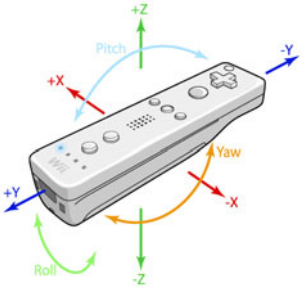

Fig. 1. The Wiimote device with its body-fixed coordinate system

4.2 Wiimote Configuration for Behavior Composition

Our proposed methodology matches the objectives evinced in Section 3 with the Wiimote functionality and limits. We aim at using the Wiimote to create behaviors by directly interacting with robots, without any robot interface in the middle. Such an approach allows the operator to use natural gestures, and, in case of real platforms, to move within the environment where the robots are deployed. Furthermore, through multi-modal feedback, such as rumble or sounds, the operator can look only at the environment, and not at the Wiimote, hence lowering her cognitive fatigue.

The Wiimote offers several types of interaction. For example, thanks to its particular design, it can be used as a pointer to focus on a specific point in the space, or just to select a robot. Moreover, through its motion sensing capabilities, it is possible to associate gestures to high-level commands to control the robotic team. However, the Wiimote embedded sensors give rise to several challenges in localizing precisely the device itself. As an example, accelerometers can sense rough motion, but they are not sufficient to retrieve the full attitude of the device within the space (in particular, with respect to a free fall coordinate system, it is not possible to detect the yaw of the device). A possible solution would be to use the IR camera. However, this must be used in combination with a Sensor Bar, hence forcing the operator to point the device against a specific part of the environment. Sko and Gardner [7] overcome this problem by installing and managing multiple Sensor Bars in a smart environment. However, this approach requires to cable the whole environment, which is something that does not favor the adaptability or the pervasiveness of the system. Our solution extends the Wiimote functionality using the Motion Plus controller, which contains two gyro sensors that report the three angular rates of the device. This allows to track the device attitude over time, without constraining the operator movements, as in the case of the IR camera, and without any change in the environment. This configuration does not use neither accelerometers nor the IR camera, hence its drawback is to track only the Wiimote attitude, but not its position within the environment. Thereby, the operator cannot perform gestures while walking, but she must stand in a fixed spot of the space.

4.3 Wiimote Interaction System

In order to use the Wiimote and Motion Plus extension for behavior composition, we designed an interaction system with the following objectives:

- supporting multiple Wiimote devices;
- integrating the Wiimote to our robotic system and, in particular, to the adaptable command architecture;
- extracting and tracking high-level information from the raw data coming from the Wiimote sensors.

The interaction system component diagram is reported in Figure 2. Our first component is a library to manage the Bluetooth communications with Wiimotes,

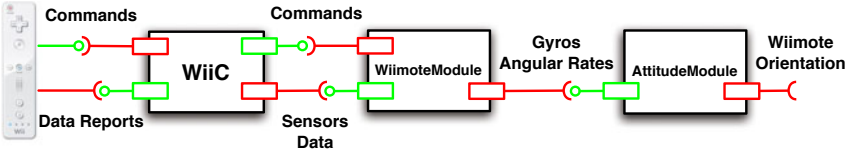

Fig. 2. The Wiimote interaction system UML component diagram

and their input/output messages. We extended Wiiuse, an open-source C library, realizing our own library, *WiiC*[4] . To our knowledge, WiiC is the only C library fully portable on Linux and Mac platforms, which supports Motion Plus and Balance Board extensions. As well as using WiiC for standalone application, it is also supported by the *OpenRDK*[5] framework for robotic applications [1], through the wrapper module *WiimoteModule*. This allows to easily integrate the Wiimote support to our robotic system, which is fully designed via such a framework. Moreover, WiimoteModule exhibits high-level functionality through a user-friendly graphical interface.

Once connected, the Wiimote periodically reports the status of its sensors. More formally, gyros transmit roll, pitch and yaw rates with respect to the Wiimote-fixed coordinate system W, represented at time t with the vector $\omega^W(t) = [\omega_{pitch}(t),\omega_{roll}(t),\omega_{yaw}(t)]^T$. It is worth mentioning that pitch and roll are inverted because, according to the Wiimote reference system, roll is a rotation around the y-axis, while pitch represents a rotation around the x-axis. Our purpose is to track the device orientation with respect to an absolute reference system O, hence we must solve an inverse differential kinematics problem. This is accomplished through the *AttitudeModule* component.

The attitude of the Wiimote is given by the quaternion vector $q(t) = [q_0(t), q_1(t), q_2(t), q_3(t)]^T$. We adopted quaternions to deal with singularities, since they are a more robust representation than Euler angles. Given the quaternion vector, $q(t)$, the following relationship represents the time derivative with respect to the angular velocity given in the body frame $\omega^W(t) = [\omega_{pitch}(t),\omega_{roll}(t),\omega_{yaw}(t)]^T$.

$$\dot{q}(t) = \frac{1}{2}\Omega(t)q(t), \quad \Omega(t) = \begin{bmatrix} 0 & -\omega_{pitch}(t) & -\omega_{roll}(t) & -\omega_{yaw}(t) \\ \omega_{pitch}(t) & 0 & \omega_{yaw}(t) & -\omega_{roll}(t) \\ \omega_{roll}(t) & -\omega_{yaw}(t) & 0 & \omega_{pitch}(t) \\ \omega_{yaw}(t) & \omega_{roll}(t) & -\omega_{pitch}(t) & 0 \end{bmatrix} \quad (1)$$

The integration process has been implemented using the *classical Runge-Kutta method* (RK4). Finally, in order to have an easier interpretation of the attitude we convert back from the quaternion representation to Euler angles.

Let the Euler angle vector $\Theta^O = [\phi, \theta, \psi]^T$ be the attitude of the Wiimote at time t, with respect to the coordinate system O. Then, using the following equations, pitch, roll and yaw angles can be obtained in the absolute frame:

[4] WiiC is available at `http://wiic.sf.net`
[5] `http://openrdk.sourceforge.net`

$$\begin{cases} \phi = \arctan\left(\frac{2(q_3 q_3 + q_1 q_1)}{q_0^2 - q_1^2 - q_2^2 + q_3^2}\right) \\ \theta = \arcsin(-2(q_1 q_3 - q_1 q_2) \\ \psi = \arctan\left(\frac{2(q_2 q_2 + q_1 q_3)}{q_0^2 + q_1^2 - q_2^2 - q_3^2}\right) \end{cases} \tag{2}$$

Since gyros are quite noisy, the *AttitudeModule* filters the incoming angular rates, before integrating them, using a *moving average filter* [10]. However, in case of orientation tracking over long time intervals, such a filtering method is not enough efficient and it is necessary to adopt some control technique, such as a Kalman Filter, as proposed by Luinge and Veltink in [4]. The Wiimote attitude tracking can be useful in several different applications, and we will provide a practical example in Section 6, using it as a pointing device.

5 High-Level Command Architecture

The next step towards a more user-friendly experience in MAS behavior composition requires an architecture to support it. In this Section, and in the following ones, the term "behavior composition" indicates the result of user interaction with a library of simple robotic behavior. The TUI device can select a sequence of simple behaviors, or select a predefined complex one (designed off-line by the user). In Section 6, we defined a case study for behavior composition.

Let us start by analyzing the requirements for building a general-purpose intercommunications architecture. We can identify the following features:

- support for multi communications protocols;
- support for interactive input devices (joypad, keyboard, speech recognition, and so on);
- integration with multiple output devices (robotic platforms, sound synthesis, automatic signaling devices);
- capabilities to adapt at runtime to different devices.

Given these specifications, we designed a complex framework that allows human operators to interact with Multi-Agent Systems. The software framework implemented is built upon *OpenRDK*, an open-source framework for robotic applications development [1]. OpenRDK is a complete framework for realizing effective robotic applications. It focuses on modularity and re-usability, defining some tools for designing software agent. Each *RAgent* is built combining *modules* that express the functionality of the agent itself. The bear of communication methods, properties sharing, as well as low-level integration with robotic platforms are relieved by the developer: the focus of robot programming is on algorithm designing for real problem solving. This framework is successfully used in different contexts, such as robotic platforms for *Urban Search And Rescue* (*USAR*), robotic soccer (*Nao Humanoid*), UAVs.

Within such framework, we designed an adaptable architecture as shown in Figure 3.

The *CmdInterpreter* component is an OpenRDK Module. It declares some basic properties for adapting its behavior to several input and output platforms.

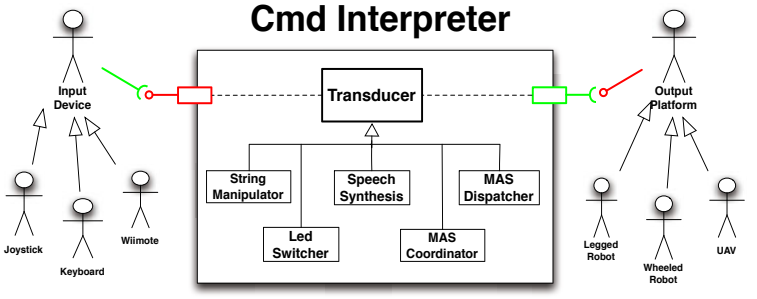

Fig. 3. A UML representation of our adaptable architecture

The *Input* module is the category wherein multiple input devices are declared: it includes, but is not limited to, TUIs (such as Wiimote), joypad and joystick, keyboard interfaces. The *Output* module includes all our supported categories of robotic platforms, real or simulated.

The interesting part is the block named *Transducer*. Inside the module, the *Transducer* component defines a hierarchy of classes, used to interface with input devices. Each class is responsible of creating support properties for handling a specific input device. Moreover, at runtime, it exposes itself within the system architecture.

A typical execution of the system is the following:

1. The *ragent* program loads all the modules defined as main components of the application.
2. The *CmdInterpreter* module loads all the transducers within a prototype database.

   This will be used to instantiate at runtime the appropriate handler related to the application. In fact, inside this module it is possible to select, either before or during the execution of *RAgent*, which transducer to be loaded: the module selects the right class prototype and instantiate the relative transducer.
3. The *CmdInterpreter* module calls the init for a selected transducer.

   At this stage, all input and output variables are created and the *EventHandler* is declared.
4. The full *RAgent* is up and running and it is ready to accept input from the user.

The behavior of *CmdInterpreter* module is dependent on the loaded transducer: the logic of command interpretation is delegated to this minimal component. Currently, we implemented transducer for string manipulation (*StringManipulator*), audio and visual feedback (*LEDSwitch* and *SpeechSynthesis*), MAS interaction (*MASCoordinator*, *MASDispatcher*).

Given to the modularity characteristic of OpenRDK framework, it is possible to combine several *CmdInterpreter* modules, to implement a complex command

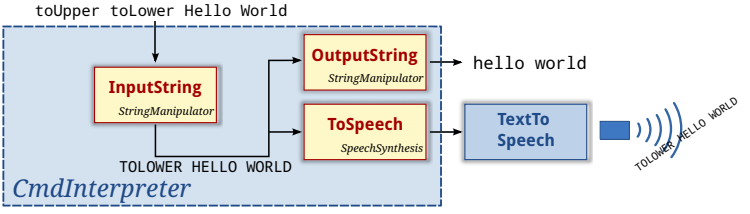

Fig. 4. An example of the polymorphic characteristics of our architecture: two *StringManipulator* and a *SpeechSynthesis*

chain. In Figure 4, it is illustrated an example that combines three different instances of *CmdInterpreter* module.

The *InputString* has an input property that takes a complex sentence as input. The user can insert basic commands for string manipulation (for example, "toLower" and "toUpper"). Let assume she inserts "`toUpper toLower Hello World`". The output string property will be "`TOLOWER HELLO WORLD`".

This is propagated to *OutputString* and to *ToSpeech*. The *OutputString* transducer will decode the input string in the same way *InputString* already did: the resulting output will be "`hello world`". The *ToSpeech* transducer, instead, will propagate the sentence directly to the *TextToSpeech* module that simply plays the whole sentence. The user should hear: "`tolower hello world`".

Despite this simple example shows a very trivial case, it presents some of the key features of the designed software architecture:

1. different transducers can be implemented limiting the programming effort;
2. given their input and output are compatible, they can be used in cascade;
3. they can share some input data each other;
4. multiple inputs and outputs can be defined.

Then, providing a transducer dispatcher (not shown here for simplicity sake), a chain of transducer can handle multiple input devices towards multiple output devices.

6 A Practical Example: Virtual Coaching

To prove the effectiveness of our software architecture, we developed a prototype application for Virtual Coaching. The objective of the system is to compose complex behavior for Multi-Agent Systems, such as game tactics, by using a user-friendly interface. In this scenario, a human operator combines tactics for a soccer team of robotic players. The soccer field is divided into several zones and each one is assigned to a single robot. The operator selects one area for each robot, simulating the placement onto the field. The tactic is, then, built by imposing a distinct role for each player. Since we are using real soccer rules, simple role behaviors are distinguished among: GoalKeeper, Defender, Supporter and Attacker.

6.1 System Overview

The overall system architecture is depicted in Figure 5.

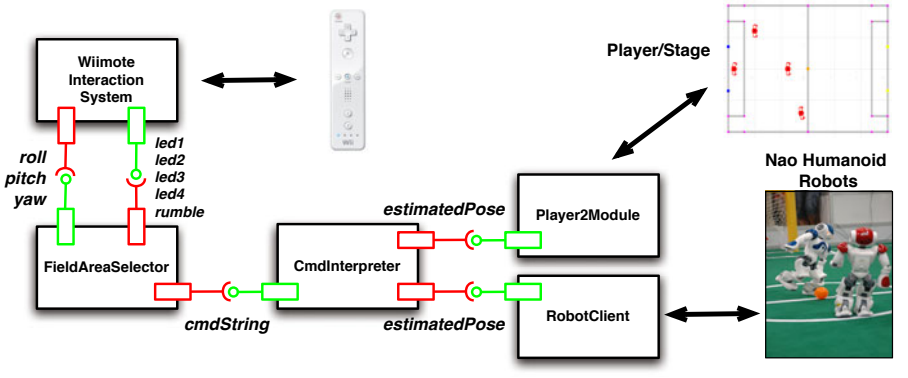

Fig. 5. The proposed architecture for Virtual Coaching

The core of the architecture is the *CmdInterpreter* module that uses a *MASSoccerPlayer* transducer. The *MASSoccerPlayer* component is defined by:

Input. a single string command, where the areas selected by the user are encoded;
Output. a *TargetPose* for each robot player.

A coordination routine is responsible for selecting the appropriate player for a selected area.

As input for the system, we have utilized the Wiimote, configured with the Motion Plus extension, as explained in Section 4.

For the output, we have used the Nao humanoid robot, currently adopted in the RoboCup Standard Platform League (SPL). To speed up the experimental results, we designed a Player/Stage simulated world [2], modeled upon the SPL rules. Commands, decoded by the *CmdInterpreter* module, are propagated both to the real robots and to the simulated one.

The complete data flow for the described application is shown in Figure 6(a).

First of all, the user calibrates the Wiimote. Then, she selects four areas. The positions of the Wiimote are translated into field area selections, and then a message to the *CmdInterpreter* module is sent. At this time, the areas are translated into field coordinates and the coordination procedure is called. It is worth mentioning that the system also validates whether the selected target points correspond to free areas within the field, hence no outer parts of the field nor obstacles can be selected. The last step involves sending the target positions to the correspective robots. A rumble feedback is sent to the Wiimote to warn the user of successfully execution of the command. The complete Finite State Machine of the user interaction is given in Figure 6(b).

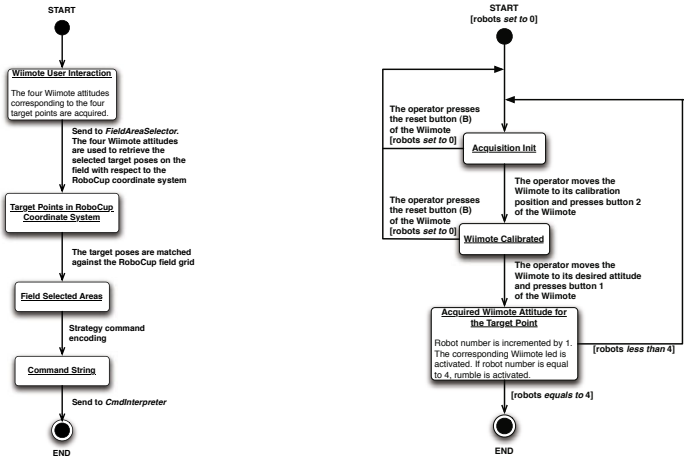

Fig. 6. UML state diagrams for the Virtual Coaching application 6(a) and the Wiimote user interaction 6(b)

6.2 Experimental Results

Experiments have been conducted to establish the correctness of the proposed architecture. They are not intended to be exhaustive, since they lack a strong validation formalism, which will be considered and evaluated in a future work.

Given the limitation of the devices, we defined the following assumptions. In order to correctly determine selected areas, the user should define the approximate height of the Wiimote from the ground. Besides that, we assume that the user selects the points by standing at the center sideline of the field. This information can be changed at runtime, and does not affect the accuracy of the system. Moreover, we discretized the field into nine distinct areas. We have determined that the system has an accuracy of $0.4m$. This means, the architecture is not able to distinguish areas whose size is less than $0.16m^2$. This is due to the noise in the sensor readings, and it determines a lower bound for the accuracy that can be achieved. However, this value is still acceptable to be used in real applications, and we proved that by developing the Virtual Coaching example.

Preliminary experiments have been conducted by using a real RoboCup SPL field. We compared the execution of the Virtual Coaching by using three different environment: text-based, with a visual interface, and using a TUI. In each scenario, we used the adaptable architecture developed so far, but adopting different input interactions. In the *TextOnly* scenario, the user has to insert the command by sending a string using a telnet-like program. In the *GUI* scenario, the operator interacts with the *RConsoleQT* program, provided by the *OpenRDK* distribution. Finally, in the *TUI* scenario, the operator uses a Wiimote connected to the system.

In these experiments, we also evaluated the characteristics of the scenario and the constraints the users have to perform same actions with different input. The Table 1 resumes the expected features emerged by performing such qualitative experiments.

Table 1. Qualitative results for Virtual Coaching experiments

Scenario	User Interaction	Command Structure	Feedback		Access to the Environment
			Visual	Physical	
TextOnly	text	string	×	×	×
GUI	point&click	button/mouse	✓	×	×
TUI	tangible	button/gesture recognition/ sensor motion	✓	✓	✓

7 Conclusions

In this paper, we analyzed the problem of human-robot interaction for Multi-Agent System behavior composition. We focus our attention on how to ease the operator effort to control a team of robotic player. The objective of the operator is to give high-level commands to the robots, by assigning specific tasks to them. We proposed the integration of Tangible User Interfaces, as means for giving such commands. Therefore, we presented a system software architecture for integrating TUIs in a MAS environment. The developed architecture is able to adapt to multiple and different input devices. Moreover, it is able to control several robotic platforms, as well as software simulators.

As practical test case, we presented a Virtual Coaching application, that integrates a Wiimote device and SPL humanoid robots. The conducted experiments validate the effectiveness of our developed architecture, even if they showed some limitations and the evaluation process is still at a preliminary stage. As future improvements, we need to increase the robustness of the system, by relaxing the starting point assumptions. A more robust integration over time of the acquired data can improve the self-localizing abilities of the Wiimote. More in depth experiments on real robots have to be done. Moreover, the system needs a more formal validation of the human-robot interaction, by performing quantitative experiments through user testing and validation.

References

1. Calisi, D., Censi, A., Iocchi, L., Nardi, D.: OpenRDK: a modular framework for robotic software development. In: Proc. of Int. Conf. on Intelligent Robots and Systems (IROS), pp. 1872–1877 (September 2008)
2. Gerkey, B.P., Vaughan, R.T., Howard, A.: The player/stage project: Tools for multi-robot and distributed sensor systems. In: Proc. of the Int. Conf. on Advanced Robotics (ICAR 2003), Coimbra, Portugal, June 30-July 3, pp. 317–323 (2003)

3. Guo, C., Sharlin, E.: Exploring the use of tangible user interfaces for human-robot interaction: a comparative study. In: CHI 2008: Proceeding of the twenty-sixth annual SIGCHI conference on Human factors in computing systems, pp. 121–130. ACM, New York (2008)
4. Luinge, H., Veltink, P.: Measuring orientation of human body segments using miniature gyroscopes and accelerometers. Medical and Biological Engineering and Computing 43(2), 273–282 (2005), http://doc.utwente.nl/61405/
5. Messing, L., Campbell, R.: Gesture, speech, and sign. Oxford University Press, Oxford (1999)
6. Mich, J.: Wiimote gesture recognition. In: Proceedings of the 15th Conference and Competition STUDENT EEICT 2009. Faculty of Electrical Engineering and Communication BUT, vol. 4, pp. 344–349 (2009), http://www.fit.vutbr.cz/research/view_pub.php?id=8933
7. Sko, T., Gardner, H.: The wiimote with multiple sensor bars: creating an affordable, virtual reality controller. In: CHINZ 2009: Proceedings of the 10th International Conference NZ Chapter of the ACM's Special Interest Group on Human-Computer Interaction, pp. 41–44. ACM, New York (2009)
8. Song, T.H., Park, J.H., Chung, S.M., Hong, S.H., Kwon, K.H., Lee, S., Jeon, J.W.: A study on usability of human-robot interaction using a mobile computer and a human interface device. In: MobileHCI 2007: Proceedings of the 9th International Conference on Human Computer Interaction with Mobile Devices and Services, pp. 462–466. ACM, New York (2007)
9. Varcholik, P., Barber, D., Nicholson, D.: Interactions and training with unmanned systems and the nintendo wiimote. In: Proceedings of the 2008 Interservice/Industry Training, Simulation, and Education Conference (I/ITSEC) (2008)
10. Wei, W.W.: Time Series Analysis. Addison-Wesley, Reading (2006)
11. WiiBrew (2009), http://wiibrew.org/wiki/Main_Page