Game Development Framework Based Upon Sensors and Actuators

Ray van Brandenburg, Arie Horst, Bas Burgers, and Nirvana Meratnia

Department of Computer Science, University of Twente, Enschede, The Netherlands
{r.vanbrandenburg,a.p.horst,b.j.burgers}@student.utwente.nl,
n.meratnia@ewi.utwente.nl

Abstract. Urge for comfort and excitement have made gadgets indispensable part of our life. The technology-enabled gadgets not only facilitate and enrich our daily lives but also are interesting tools to challenge human imagination to design and implement new ubiquitous applications. Pervasive gaming, in which human interaction and game/scenario-dependent designs are often common practices, has proved to be one of the areas to successfully combine technology and the human fantasy. By moving away from games being played by humans and by focusing instead on games played by robots and giving humans the leading role of defining game strategies and players' roles, this paper aims at bridging the two fields of robotics and wireless sensor/actuator networks and exploring their potentials in the field of pervasive gaming. A generic game development framework is introduced that accommodates different types of robots and various kinds of sensors and actuators. Being extensible and modular, the proposed framework can be used for a wide range of pervasive applications built upon sensors and actuators. To enable game development, a Wiimote-based robot identification and localization technique is presented. The proposed framework and robot identification, localization, control and communication mechanisms are evaluated by implementing a game example.

1 Introduction

With no doubt technology has changed the way we perceive the world, communicate and interact with others and our environment, live our lives, perform our jobs and entertain ourselves. The great ubiquitous computing vision of Mark Weiser is not far from our reach and researchers and developers have already started challenging the limits of this vision by bringing technologies and human imaginations to the next level. Today more than ever, gadgets "recede into the background of our lives" [1]. The technology-enabled gadgets not only facilitate and enrich our daily lives and increase productivity at work but also are interesting tools to challenge the humans imagination to design and implement new ubiquitous applications. Pervasive gaming has proved to be one of the areas to successfully combine technology and the human fantasy. Schneider et al. define pervasive game as "live-action roleplaying game that is augmented with computing and communication technology in a way that combines the physical and digital space together" [2]. Although this definition does not explicitly specifies players to be humans, the majority of designed pervasive games are centered around human players and have a strong focus on human interaction [3,4,5,6].

In outdoor game scenarios, human identification and localization is often based on large-scale localization using GPS or GSM [7]. Indoor localization on the other hand is generally based on beacon triangulation [8,9,10]. The accuracy of both indoor and outdoor localization techniques significantly varies depending on the technology and the environment where the technology is used and it can be anywhere between few meters to few centimeters.

By moving away from games being played by humans and by focusing instead on games played by robots and giving humans the leading role of defining game strategies and players' roles, this paper aims at bridging the two fields of robotics and wireless sensor/actuator networks and exploring their potentials in the field of pervasive gaming.

2 Identification and Localization

One of the most important aspects of almost every pervasive game is the knowledge of own, and possibly the opponents' location. In addition, it is highly useful to be able to distinguish one player from the other. In other words, the majority of pervasive games, if not all, require player identification and localization. Although identification and localization may sound like two different problems, they may both be solved using the same technique. One should recall that players of the pervasive games we have in mind are robots. Not restricting ourselves to robots that have built-in hardware capable of localization, there is a need to use external sensors for localization and identification. The choice of external sensors depends on many factors such as cost, precision, and flexibility, to name but a few. Possible solutions for both identification and localization include:

- ***Beacon triangulation*****:** By placing either infrared or ultrasound beacons on certain positions on the playing field, it is possible to triangulate the location of an object, for example a robot, using an IR detector placed on the object itself. The main advantage of this approach is its low cost. A disadvantage is that every robot should have its own detector. This technique is more suitable for localization than for identification. Moreover, it requires developing a special communication line for the framework to communicate with the IR detector.
- ***Digital camera*****:** Using a camera and image recognition software it is possible to locate and identify different robots, which can be distinguished by for example, different colored patches. This is a method often used in Robosoccer [11]. The advantages are high accuracy and relatively simple communication mechanism. The main drawback may be the complexity of image recognition software.
- ***Wiimote [12]*****:** Wiimote is a game controller used by the Nintendo Wii game console. It contains accelerometers as well as an infrared camera. Communication with the Wiimote is over Bluetooth. Using a stationary Wiimote, it is possible to detect IR LEDs mounted on robots. The main advantages of this approach are its relative simple and fast implementation and reasonably good identification and localization accuracy. The Wiimote is also a cheap

solution compared to the digital camera. Disadvantages include limited precision at larger distances and also that one Wiimote can only ‘see’ up to a maximum of four LEDs.

2.1 Wiimote

Because of the relatively good identification/localization accuracy and the easy Bluetooth connectivity, we use the Wiimote for robot identification/localization. It can be placed anywhere in the playing field to locate both static and mobile robots equipped with IR LEDs.

In presence of more than four IR sources, Wiimote arbitrarily chooses which LEDs to save in its registry and which ones to discard. The order in which the Wiimote passes the IR source data to the Bluetooth controller is random. In fact the Wiimote does not really track an IR source and rather just passes the positions of up to four IR sources to the recipient. Therefore, constant mapping between incoming IR data and one of the tracked objects is needed.

A ‘must have’ feature of the framework is the ability to distinguish between multiple robots, for example between friend and foe, or between two members of the same team. Using the Wimote, there are multiple possible solutions to this problem. The most obvious solution is to let the LEDs blink at different frequencies. However, there are two drawbacks to this approach. First, internal signal processing inside the Wiimote makes determining the exact frequency with which the LEDs blink hard. To solve this problem, it would be necessary to put the different LEDs very far from each other. This in turn would make tracking the blinking LEDs more difficult. A simpler solution is to determine the identity of a robot by the number of LEDs it has on top. For example, one LED means robot 1 (or team one) and two LEDs means robot 2 (or team two). However, because the Wiimote can only track up to four LEDs simultaneously, this approach limits number of robots the Wiimote can track at the same time.

Communication with the Wiimote is done over Bluetooth using an open source C library called WiiUse [13]. Functions of this library include reading IR and accelerometer data from the Wiimote and changing various internal Wiimote parameters such as IR-sensitivity.

2.2 Localization Technique

An important step in the Wiimote-based localization is transforming raw camera coordinates to locations relative to the Wiimote in the 2D playing field. In order to do so, one constraint has to be introduced. This constraint is that all the LEDs need to be at the same height to convert the 3D problem to a 2D problem. Fig 1, in which various variables used in the transformation are shown, presents a side view of a Wiimote and an IR LED source.

The transformation from raw camera coordinates to positions relative to the Wiimote is carried out using the equations (1) and (2), in which φ_{FOV} is the field of view of the camera, X_{res} and Y_{res} are the resolution of the camera in x and y, x_{raw} and y_{raw} are the raw camera coordinates and h_{diff} is the difference in height of the LED and the Wiimote. Equations (1) and (2) are a function of, among other things, the pitch of the Wiimote. While it is of course possible to determine this pitch by hand, it is more

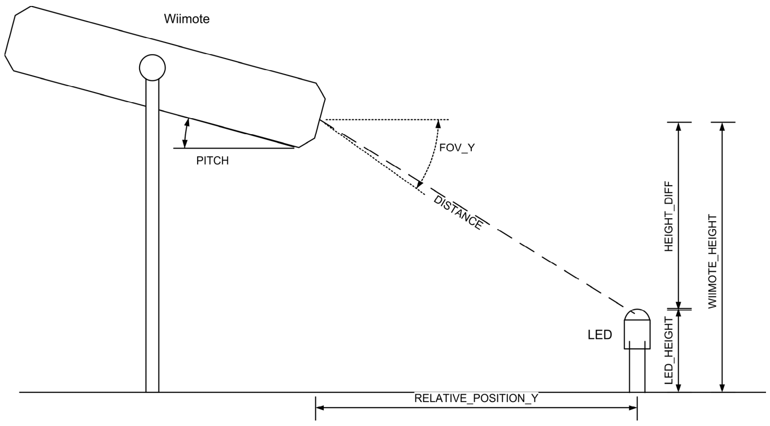


Fig. 1. The side view of the Wiimote

accurate and more dynamic to let the Wiimote itself determine this pitch using the built-in accelerometers. This is done by measuring the direction in which Earth's gravity points relatively to the three-axis-accelerometer. This way when the Wiimote changes pitch, for example because it is mounted on an actuator, it can re-determine its pitch and recalculate all locations automatically.

$$y_{relative} = h_{diff} \cdot \tan\left(\frac{y_{raw}}{Y_{res} / \varphi_{FOV,Y}} + \left(90 - \theta_{pitch} - 0.5 \cdot \varphi_{FOV,Y}\right) \right) \tag{1}$$

$$x_{relative} = \sqrt{\left(y_{relative}\right)^2 + h_{diff}^{\ 2}} \cdot \tan\left(\left(0.5 \cdot \varphi_{FOV,X}\right) - \frac{x_{raw}}{X_{RES} / \varphi_{FOV,X}} \right) \tag{2}$$

$$\begin{bmatrix} x_{world} \\ y_{world} \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x_{relative} \\ y_{relative} \end{bmatrix} + \begin{bmatrix} x \\ y \end{bmatrix} \tag{3}$$

Equations (1) and (2) calculate the position of an IR source relative to the position of the Wiimote. In a game scenario it could be useful to mount the Wiimote on some sort of rotating actuator, which requires determining the positions independently from the Wiimotes own position and orientation. This can be done using the equation (3), in which θ is the rotation of the Wiimote in the (x,y) plane.

2.3 Localization Precision

Several measurements have been performed to test the precision of localization using the Wiimote. Data from these measurements is shown in Fig 2. As it can be seen, the precision of the Wiimote-based localization in the x-direction is fairly constant; an average error of about -10mm to 10mm. In the y-direction, however, the error strongly depends on the distance to the object. This error in the y-direction can be split into three distinct areas, i.e., short, medium and large distance.

At very short distances from the Wiimote, error is relatively large (about 40mm). The reason is twofold. First, when the LED is very close to (beneath of) the Wiimote, only just within the Wiimotes vertical field of view, the accuracy of the IR camera is not very good. This effect can also be seen in the figure depicting the error in the x-direction, where the error is the largest at the places where the Wiimote is only just visible (at -500mm and 500mm). This could be due to the lens distortion effect. Secondly, at short distances from the Wiimote, the error resulting from the pitch calculation has a larger effect and leads to a larger error in position determination.

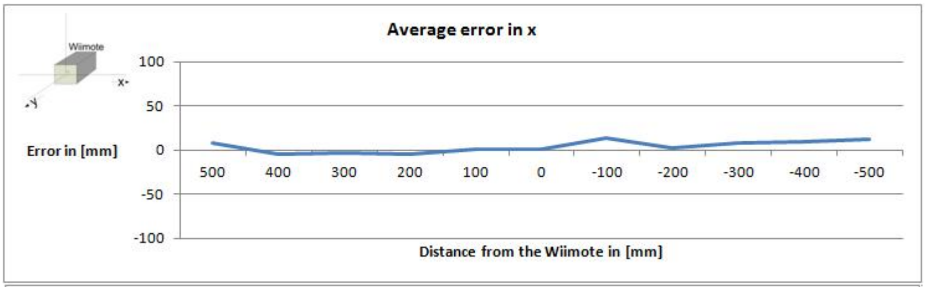
At medium distances, the Wiimote performs quite well, having an average error of about 12mm. This error is comparable to the error found in the x-direction.

At large distances, the limited resolution of the IR camera together with the fact that the height of the Wiimote compared to the distance to the object is small, result in a lower accuracy and an average error of about 35mm.

The maximum range of the Wiimote highly depends on the specific type of LED used. In our experiments performed with a small standard LEDs, the maximum range appeared to be around 2 meters when the LED was pointed directly at the Wiimote. So-called ultra-bright LEDs could dramatically increase this range.

The accuracy of the proposed technique is good for distances up to three meters and is comparable with infrared beacon triangulation methods such as the one described in [8].

One possible way to increase localization accuracy is to use two Wiimotes and to combine the data to form an image of the playing field with better precision. Another



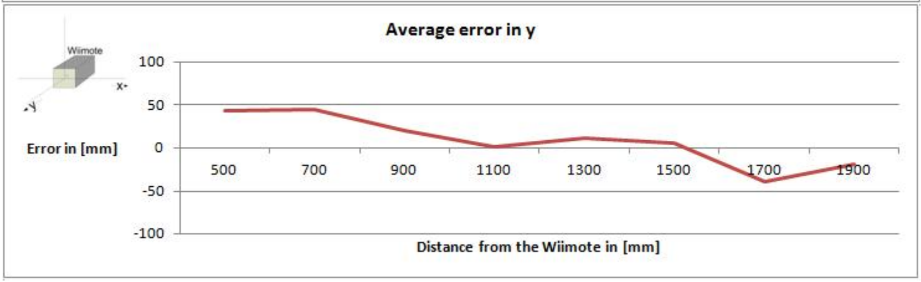


Fig. 2. Wiimote's measurement data (y=1200[mm], height=420[mm]) (top), Wiimote's measurement data (x=0[mm], height=420[mm]) (bottom)

solution would be to hang the Wiimote from the ceiling. However, this will limit the size of the playing field because of limited field of view of the Wiimote.

3 Framework

The framework needs to accommodate various robots, different sensors and actuators and facilitate communication and cooperation between these devices. Not to be restricted to only one or a set of games/scenarios, the framework should provide basic functionality for defining a game with dynamic rules and strategies. Basically, the framework should be:

- Extendable, to be easy to add new robots, sensor, actuator, scenario, etc.
- Flexible, to support creation of different type of games and support dynamic change of roles, game scenarios, and strategies.
- Platform independent.
- Modular, to be able to just use a sub set of sub-systems and certain aspects to make the framework useable for other games and ubiquitous computing applications.

Since not all robots allow one to directly program them, designing the framework in a completely decentralized approach is rather restricting. On the other hand, designing the framework in a completely centralized fashion is limiting its flexibility and extensibility. Therefore, we opt for a combination of centralized approach (for team strategy and game rules) and decentralized approach (for robot control). In order for this combination to work, in addition to “game” and “robot” layers, an extra layer called Role layer is required. Every team strategy defines a list of roles. Each of these roles defines behavior of a certain robot (e.g. defender, attacker). Every robot can be assigned an initial role by the team strategy. It then executes the commands defined in the role and reacts to sensor input as specified in the role. The robot behavior is therefore decentralized. A robot either changes its role by itself based on sensor input or it

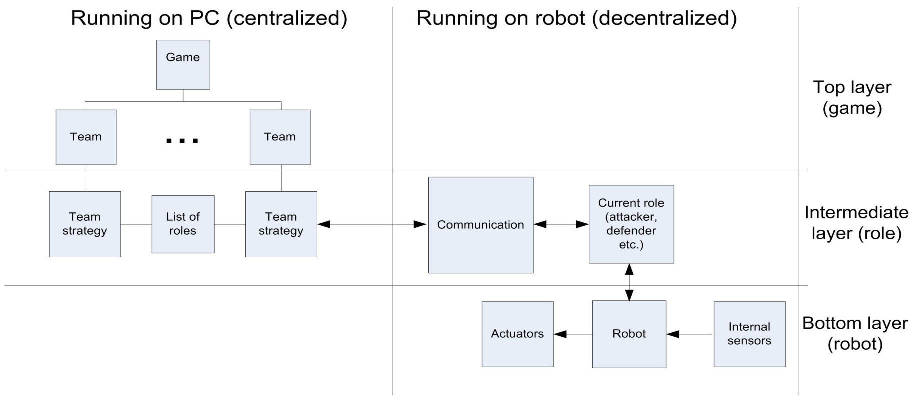


Fig. 3. Diagram presenting different layers of the framework

is given a new role by the team strategy. To ensure the modularity, we use the object-oriented approach to design the framework, which will later be implemented in Java to also assure platform independency. Fig 3 illustrates different layers of the framework.

Robot and Sensor classes assure framework's extensibility, flexibility, dynamicity, and ease of adding new robots, sensors and actuators. These classes contain basic functionality to be used by all types of robots and sensors. This functionality includes communication and identification methods. Robots and sensors are easily implemented by extending these basic classes. The communication protocols for different robots can be placed in separate classes linked to Robot class. This way it will be very easy to use a different form of communication without having to alter the framework. The framework makes a distinction between internal (built-in and embedded in robots) and external sensors (added to the robots, game field, and teams). Internal sensors are integrated in the classes representing the robot, while external sensors are classes extending the basic Sensor class. The reason for doing so is that internal sensors are often read-out by the same protocol that is used to control the robot and therefore it is logical to include them in the Robot class that also governs the protocol for communicating with a particular robot. Moreover, a sensor added to a game serves as a global sensor, which can be accessed by all teams and robots. In the same way a sensor can be added to a team, so that it can be used by all robots in that particular team, or to a robot so that it can only be accessed by that robot. The framework also allows robots to be added to a game instead of a team, so that the robot can function for example as a referee. Fig 4 illustrates the basic class diagram. One should note that actual implementations of game rules, team strategies and roles are placed in classes extending the basic Game, Team and Role classes.

As previously mentioned, using Wiimote requires some form of IR tracking. The framework accommodates this using an object detector, which creates a new instance of a WiiObject class for every source of IR light. As long as the object stays within the field of view (FOV) of the camera, this detector ensures that the WiiObject will always point to the IR source it was created for. The WiiObject instance maintains information concerning the corresponding IR source, such as its position history, number of LEDs an object is represented with, and its last known orientation. Having

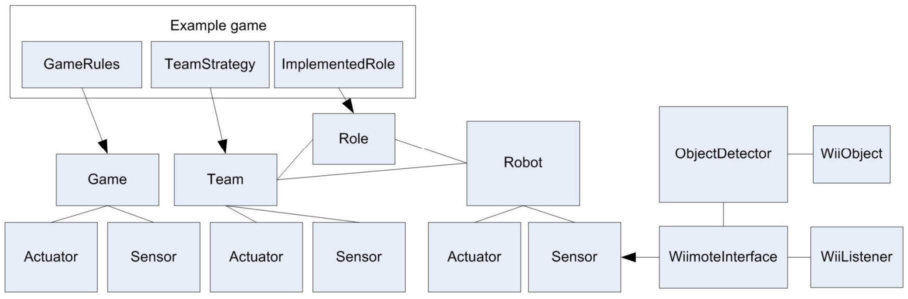


Fig. 4. Simple class diagram

The table below presents the main classes of the framework:

Game	Class that contains list of all teams, robots, sensors and actuators. A class that extends Game contains the rules of a particular game and has the responsibility of creating all sensors and robots.
Team	Class that maintains a list of robots, sensors and actuators belong to a team. Classes that extend Team contain team strategies for a particular game.
Role	Class that represents roles. Every Role is mapped to one robot. Classes that extend Role contain a role designed for a particular robot.
RoleListener	Interface that can be added to an implementation of Team so that it receives events created by roles.
TeamListener	Interface that can be added to an implementation of Game so that it receives events created by teams.
Robot	Abstract class that contains common robot functionality.
Sensor	Class used for external sensors and implements SensorInterface.
Position	Class that represents position and orientation on the playing field.
WiimoteInterface	Class that functions as a gateway between Wiimote and the rest of the framework. It also contains localization techniques. For every Wiimote one WiimoteInterface is needed. It also extends Sensor class.
WiiListener	Class that handles communication with the WiiUseJ library.
ObjectDetector	Class which handles the mapping from raw infrared data to objects on the playing field and creates an instance of WiiObject for every new IR source. It then tracks this source until it leaves the FOV of the Wiimote. It also implements the identification system.
WiiObject	Class which represents an object tracked by ObjectDetector. Maintains information concerning the IR source such as current position, position history and type of object.
EventObject	Every event thrown by a robot or sensor is encapsulated in an implementation of the abstract class EventObject. There are two major types of event objects, TeamEvents and RoleEvent. TeamEvents are handled by TeamListeners (in most cases the implementation of Game), while RoleEvents are handled by RoleListeners (in most cases the implementations of Team).
CommandObject	CommandObject is a class representing a command sent to one of the robots or one of the teams. Examples are StartCommand and GoToStartPositionCommand.

all this information in one place makes sharing information among other entities such as teams or robots easy.

When the used identification mechanism does not provide unique identification properties, the following mechanism can be used. When the tracked object goes out of the Wiimotes' FOV, the WiiObject is noted as not being visible. Subsequently when a new object comes within the FOV, this new object is checked against all objects which have previously been visible in order to find a mapping between objects. Parameters such as last seen position and displacement vector are taken into account for this mapping. This way it is possible to track an object while it is being obscured, for example by another robot or object, without creating new WiiObjects every time a robot re-appears or disappears.

4 Game Development

First step to develop a game using the framework is the choice of robots. Figure below illustrates the two robots we have decided to use: Nabaztag (left) and iRobot Roomba (right).

Nabaztag is a simple non-movable gadget whose basic functionality includes reading emails loud, making comments about the weather and talking with other Nabaztags across the Internet. It has rotatable ears (which can only be positioned accurately in steps of 20 degrees in a range of 0° to 180°), several multicolored LEDs and a speaker used for text-to-speech functionality. It also has two sensors, i.e., microphone and RFID reader. Due to limited sensor/actuator capability of Nabaztag, external sensors/actuators need to be used.

Roomba is a circular formed flat vacuum cleaner robot. It contains the following actuators: two motors enabling differential drive, vacuum cleaning brush, speaker, and multiple-color LED. Roomba has 19 sensors, among others, bump detector, touch sensor, dust detector, infrared sensor, angle, and distance sensors.

4.1 Example Game Scenario

Let us consider the following example game scenario. There are two teams, an attacking team and a defending team. Each team consists of a Nabaztag and a Roomba. Each Nabaztag has a Wiimote mounted on top. The goal for the attacking team is to touch the defending Nabaztag with its Roomba. The goal for the defending team is to prevent the attacking team from succeeding. Defending can be done by touching the attacking Roomba. Both teams have their own Wiimotes, which they use for locating their own as well as the enemy Roomba. When attacking Roomba touches defending Nabaztag or defending Roomba touches attacking Roomba, teams switch and the game starts all over again.

Due to the fact that the presented framework provides a solid base and modules, implementing specific games require very limited additions. For instance, for our example game scenario, we only need to create an extension to the Game class to accommodate the rules for this specific game and to create two different team strategies, each containing several roles and set up the two different robots.

4.2 Technical Problems and Solutions

To attach the Wiimote to the Nabaztag, a Lego construction is attached in place of Nabaztag's ears. The ear controller controls a gearbox (to increase the accuracy of the rotatable ears) and a turntable running on bearings on top of the construct. This construction results in a Nabaztag which is able to turn its ‘head’ (and also the Wiimote) accurately with a 22.5° resolution.

To control the Nabaztag, a communication class called NabaztagComm is created that, among other things, sets the position of the ear between 0° and 180°, sets LEDs with any given color, sends the Nabaztag to sleep and wakes it up. Moreover, the WiiNabaztag class is created, which represents a special case of the robot that is equipped with the Wiimote. This class rotates the ears and keeps track of orientation of the Wiimote attached to it. It also supports setting the Wiimote direction in 16 different directions between 0° and 360°.

Although Roomba’s angle and distance sensors are useful in determining robot’s position and orientation, their values seem to be incorrect. After conducting various experiments, it became apparent that there is neither constant factor between the actual and theoretical number nor is the error systematic. Another problem of Roomba is that it is not possible to program it directly as it has no internal memory that can be accessed.

When the Roomba is supposed to drive a certain distance, the library sends a drive command to the Roomba. At this moment a timer is started. After a given time, which is determined by the specified distance and speed, the library sends a stop command. This mechanism is not very accurate because of the exact timing involved. Using the library in a heavily multi-threaded environment, this becomes a serious problem. In order to determine the magnitude of this problem a couple of tests have been performed.

Table. 1 presents results of various tests in which the Roomba was driven certain distances. A number of observations can be made from these experiments. First, the average errors compared to the driven distances are very small. Secondly, the relatively large gap between the average deviation and the maximum deviation shows that timing is indeed important. Thirdly, Roomba necessitates good calibration. Experiments show that speed 110 [mm/s], which is not as well calibrated as the other two speeds, results in significantly larger errors.

Table 1. Roomba distance measurements (left), Roomba Angle measurements (right)

Speed [mm/s]	**Desired distance [mm]**	**Average deviation [mm]**	**Max. deviation [mm]**	**Average error/meter [mm/m]**
110	1000	9.3	12	9.3
	2000	18.3	29	9.2
	3000	27.0	41	9.0
194	1000	8	16	8
	2000	10.6	20	5.3
	3000	13.3	20	6.7
305	1000	5.7	7	5.7
	2000	5.0	10	2.5
	3000	2.0	3	0.7

Desired rotation [°]	**Average deviation [°]**
360	1.4
-360	0.4
720	1.6
-720	1.6

Because a straight line is not the only path a Roomba is required to traverse, some extra tests were performed in which the Roomba was required to turn specific angels. The tests were performed in both clock-wise and anti clock-wise directions to see if there was difference between the motors driving different wheels. As the results presented in Table 1 show, the error made when rotating the robot is very small. Control of the Roomba's movement after the calibration and under aforementioned circumstances is quite precise.

To implement the example game scenario, we use a simple identification technique in which each robots of one of the teams was equipped with a single LED and each robot of the other team with two LEDs. In addition to this, the framework used input from robot's angel and distance sensors when they were out of view of the Wiimote.

5 Conclusion

The generic framework presented in this paper is able to accommodate various robots, sensors and actuators. Its flexibility, extensibility and modularity enable developing a broad range of pervasive games and game scenarios based upon robots, sensors, and actuators. To demonstrate the framework an example game using two commonly available robots has been developed. Furthermore, the cheap and easily accessible Wiimote-based localization and identification technique presented can be used in variety of ubiquitous systems including pervasive games. The accuracy of the proposed technique is good for distances up to three meters and is comparable with existing infrared beacon triangulation methods. At longer distances, however, accuracy decreases.

To enable controlling the robots more accurately, the future work includes integrating the framework with a full feedback controller using the localization system as input. Another possible extension to the framework is support for more complicated movement patterns such as spline, which will allow more advanced strategies.

References

1. Weiser, M.: The Computer for the Twenty-First Century. Scientific American, 94–110 (1991)
2. Schneider, J., Kortuem, G.: How to Host a Pervasive Game: Supporting Face-to-Face Interactions in Live-Action Roleplaying. In: Proc. Designing Ubiquitous Computing Games (2001)
3. Magerkuth, C., Stenzel, R., Streitz, N., Neuhold, E.: A Multimodal Interaction Framework from Pervasive Game Applications. In: Workshop for Artificial Intelligence in Mobile Systems, USA (2003)
4. Anastasi, R., Tandavanitj, N., Flintham, M., Crabtree, A., Adams, M., Row-Farr, J., Iddon, J., Benford, S., Hemmings, T., Izadi, S., Taylor, I.: Can You See Me Now? A Citywide Mixed-Reality Gaming Experience, Equator Technical Report, University of Nottingham (2002)
5. Björk, S., Falk, J., Hansson, R., Ljungstrand, P.: Pirates! –Using the Physical World as a Game Board. In: Proc. Interact 2001, IFIP TC.13 Conference on Human-Computer Interaction, Tokyo, Japan (2001)

6. Linner, D., Kirsch, F., Radusch, I., Steglich, S.: Context-aware Multimedia Provisioning for Pervasive Games. In: Proc. of the 7th IEEE International Symposium on Multimedia (2005)
7. Benford, S., Magerkuth, C., Ljungstrand, P.: Bridging the physical and digital in pervasive gaming. Communications of the ACM, 54–75 (March 2005)
8. Brassart, E., Pegard, C., Mouadibb, M.: Localization using infrared beacons. Robotica 18 (2000)
9. Mottaghi, R., Vaughan, R.: An integrated particle filter and potential field method applied to cooperative multi-robot target tracking. Autonomous Robots, 19–35 (July 2007)
10. Eom, D., Jang, J., Kim, T., Han, J.: A VR Game Platform Built Upon Wireless Sensor Network. In: Advances in Visual Computing. Springer, Berlin (2006)
11. Weiss, N., Hildebrand, L.: An Examplary Robot Soccer Vision System. In: Workshop on Robots in Entertainment, Leisure and Hobby, Austria (2004)
12. http://en.wikipedia.org/wiki/Wii_Remote
13. Laforest, M.: Wiiuse, http://www.wiiuse.net