

# An interactive HRI for walking robots in RoboCup

Renato Samperio

*Department of Computing and Electronic Systems  
University of Essex, Wivenhoe Park, CO4 3SQ, U.K.  
Email: rsampe@essex.ac.uk*

Huosheng Hu

*Department of Computing and Electronic Systems  
University of Essex, Wivenhoe Park, CO4 3SQ, U.K.  
Email: hhu@essex.ac.uk*

**Abstract**—This paper presents an interactive human-robot interface (HRI) capable of evaluating robot localisation performance and maintaining full control of robot behaviours in the RoboCup domain. The system consists of legged robots, behaviour modules, an overhead visual tracking system and a Graphic User Interface (GUI). A human-robot communication framework is designed for executing cooperative and competitive processing tasks between users and robots by using object oriented and modularised software structure. Some experimental results are presented to show the feasibility and performance of the proposed system.

**Index Terms**—Human-Robot Interface, GUI, Walking robots

## I. INTRODUCTION

Software development platforms play an important role for control design of robots, especially before a prototype system is available. A good software platform can provide simulation functions that speed up the development of different algorithms, including complex programming and huge data collection. There are mainly two types of software development platforms for robotics research. One is the pure simulation platform, in which only models of real robotic systems are presented. It is based on an assumption that the developed algorithms can be implemented in future. Another type is hybrid, which partially relies on the real robotic system. The parameters of these systems are fully collected from real robots and the real environment, which is also based on real experiments. If the algorithm works well under such a hybrid platform, it can also work well on real robots.

Up to now, some human-robot interfaces have been already developed, including mobile robot control [5], behaviour design [2], vision-based applications [4] and multi-activity platforms [1]. These systems manage robot behaviours within independent functionality, and offer an instructional execution between users and robots [11]. The robots are able to react environment stimulus where possible actions are designed by user criteria [10]. Moreover, the robot can support an auto-designing behaviour which assimilates user instructions referred to simple initial trajectory [7]. A control platform also adapts to processing resources and communication schemes for real-time or remote robot control.

Our research work is based on Sony AIBO walking robots. There are some difficulties in the development of control

software and sensing algorithms for these robots. Firstly, programming AIBO robots takes a long time and may have a risk to damage the robot. It would be safe for such development cycle done on a hybrid experimental platform in which most of the work is done on a PC efficiently. Secondly, the processor on the Sony AIBO robot is not powerful enough. Complex algorithms cannot be implemented on real robots in real time. It becomes necessary to develop complex algorithms on a PC during the development phase, then optimized in size for the real robots. Thirdly, the current interface between the Sony robot and human operators is not friendly. Each time when the experiment completed, all the results need to be downloaded from the memory stick manually, which is very time consuming. With the proposed hybrid platform, the progress of experiments is displayed on the screen directly and any problem with the experiment can be seen immediately, i.e. a convenient way to develop control and vision algorithms [8] and [9].

The rest of the paper is organized as follows. Section II describes the system design and a client-server scheme. Section III outlines the robot architecture from an embedding software perspective. Simulated and tracking interfaces for localisation support is detailed in Section IV, and experimental results are given in Section V to show the feasibility and performance of the proposed system. Finally, a brief conclusion and future work are described in Section VI.

## II. SYSTEM DESIGN

### A. System configuration

Figure 1 presents the configuration of the proposed hybrid HRI. The user-robot interface manages robot localisation information and user commands from a GUI. The overhead VICON tracking system is used for evaluating robot positions.

The robot localisation is implemented by visual perception, motion and behaviour planning modules which continuously sends robot positioning information. In this particular case, localisation is executed independently from robot behaviour and shared same processing resources. At last, users can manage and control experimental design with the GUI tool.

## User-Robot Interface

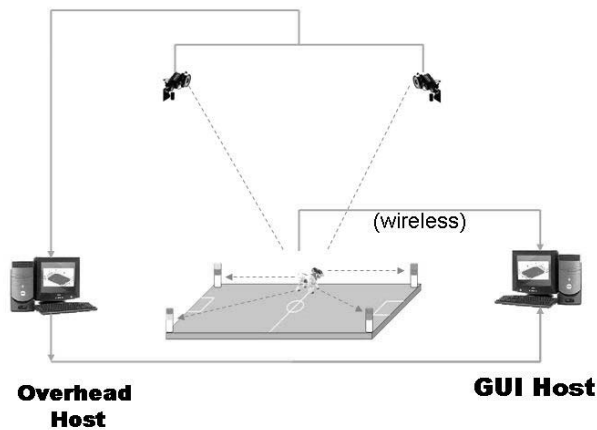


Fig. 1. The configuration of the proposed HRI System

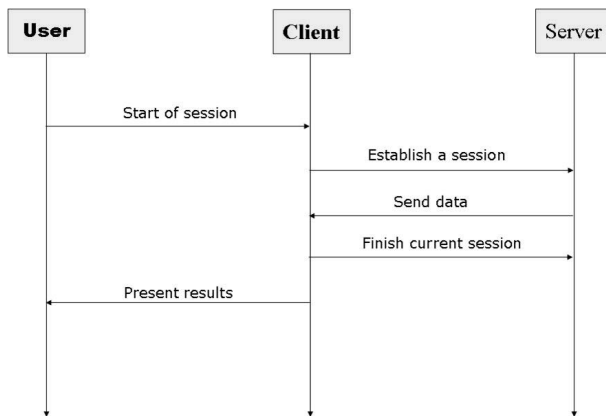


Fig. 2. Client-Server task communication process

### B. Client-Server Module

This section describes a client-server scheme which works as a request and response service for human-robot communication. The design is implemented using a message exchange sequence where server is an information provider and client is a service solicitor. A wireless OpenR IPv4 stack is used for a TCP/IP robot communication process which is conformed by server service, GUI client interface and an instructions message. The service is used as a stand alone communication process among robot, overhead tracking and user interaction. The system has real-time transmission based in petition and response transactions as is shown in Figure 2.

Initially, client starts a service session through a server petition. As soon as the client establishes connection with the server, it deals with communication requirements (time of session, rate of transmission and priority), and then the transmission. After all information is transmitted communication is finished. Then, an TCP/IP message object is used

for GUI client and server transmission interface.

### C. GUI - Client

The client side is contained in a GUI which interacts with user activities such as robot operational commands, image interpretation, landmark mapping and behaviour design. The GUI is implemented into a thin client using a Java front view with a multi-platform and adaptable code reusing. Moreover, native Java packages were used for multi-threading, image treatment operations, distributed processing and communication interfaces into a client module. The additional Java modules implemented are:

- Image treatment: This module uses Java Advanced Imaging package for image manageability in a flexible, extensible and distributed image processing.
- Networking: Java Net package is used for all messages sent among robot commands, behaviour design and any other sensing in a wireless transmission form.
- User Interface: It is created with Java Swing package using a look and feel window screen navigation for calibrating visual tuning, behaviour designing, robot teleoperation and mapping.

The GUI client coordinates experimental execution and integrates synthetic or real-time data of robot positioning with the data from the overhead tracking system. Furthermore, all modules work asynchronously using multi-tasking services for robot control and analysis.

### D. Server

In this implementation we use a server for receiving data from the robot and overhead tracking system via an asynchronous port for each service and transmitting positioning results to GUI for visualising different robot positions in a global frame. On the other hand, robot TCP/IP implementation follows an Apertos object-oriented schema in the OpenR embedded operative system. This service acts a passive robot daemon with some reserved memory space for required operations and variables [5]. In such case, the server side provides networking routines for dealing with transmission delay and data losing.

## III. ROBOT ARCHITECTURE

### A. Layered Design

A layered architecture is developed for structuring information so that each layer controls different robot skills based on sensed information, as shown in Figure 3. It is a modularized architecture and priority-based.

**Actuators, sensors and input/output:** This layer is in charge of controlling hardware devices. Each AIBO robot has a MIPS R7000 64bit RISC processor with 576 MHz of clock speed, 512MB SDRAM and 32MB flash memory. It handles 18 degrees of freedom (mouth, head, legs, ears and tail). It has many sensors such as temperature, infrared, pressure

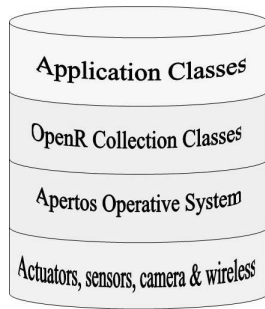


Fig. 3. The layered design of robot software

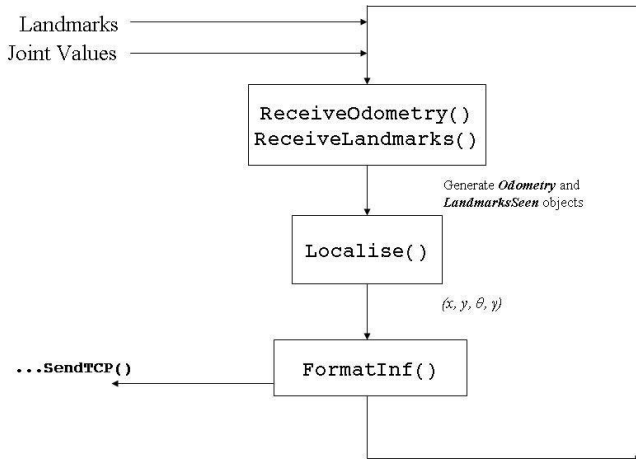


Fig. 4. A flow chart for the localisation module

and vibration, a CMOS image camera (350000 pixels). Some input/output devices are LEDs on face, ear, head and back; wireless IEEE 802.11b; memory stick slot; and miniature microphones and speaker.

**Apertos Operative System:** Apertos layer includes a distributed object-oriented operating system which uses a reflective architecture. A reflector offers meta-operations provided by meta-objects constituting a meta-space, which is defined by a class hierarchy or meta-hierarchy. This sort of reflective programming reuses existing reflectors meanwhile meta-hierarchy checks object compatibility. Objects can also be changed by meta-spaces composition to modify their semantics. Hence, new object behaviour can replace old ones using components of the operative system.

**OpenR Collection Classes:** This layer contains Apertos Operating System with programming operations with three objects such as *OVirtualRobotComm*, *OVirtualRobotAudio* and *ANT* TCP/IP. Object *OVirtualRobotComm* offers service for Effectors, Sensor and *OFbkImageSensor* objects. *OVirtualRobotAudio* deals with audio outputs. The *ANT* object is in charge of the endpoint buffers for input and output data.

**Application Classes:** This layer is used for robot behaviours and localisation module, including 3 main modules:

*AiboEssexControl* - control robot actuators and input/output devices; *AiboEssexObserver* - translates any sensed data to positioning such as robot pose; and *AiboEssexComm* - in charge of TCP/IP communication protocol, as well as of information formatting for transmission. Note that the robot walking styles and head tracking, image analysis and feature detection, sound generation as well as behavioural control are included as subtasks of *AiboEssexControl* object.

### B. Localisation module

The localisation module transforms sensed data from image perception to robot positioning, which is independently processed as a stand alone process. The complete procedure is illustrated in Figure 4 and can be described as follows:

- 1) Initially, Localisation landmarks and robot odometry are generated independently and used respectively in *ReceiveOdometry* and *ReceiveLandmarks* methods. The *ReceiveOdometry* creates suitable odometry values for localisation, which are encapsulated in an *Odometry* object. The *ReceiveLandmarks* method organises observed candidate landmarks into an *LandmarksSeen* object.
- 2) Then, the robot generates a position using a localisation method from *Localise* object and following an update-observe-predict Bayesian structure. At this stage a positioning vector is obtained i.e.  $(x, y, \theta, \gamma)$  - robot space coordinates, orientation angle and a level of confidence correspondingly.
- 3) Lastly, information is formatted and transmitted for further local reuse by *FormatInf*. Also at this stage, information can be sent to client TCP/IP service using *sendTCP()* if a petition is received.

### C. Behavioural control module

We implemented a behaviour module for relating robot execution with its environment. A behaviour is a sequence of states which guides robot activities using perceived environment as a stimulus for a robot activity [2]. The behaviour module has a state machine for assigned actions in which each state executes a robot task or set of tasks.

A behaviour requires transitional state rules for executed states such as initial, ready, play and finish steps. The **Initial** step initialises robot objects and devices; **Ready** step sets up robot devices and uploads object behaviour configuration; **Play** step executes current state and **Finish** step stops any service, destroy memory objects and power off robot devices if necessary. Behaviour execution is iterated until it reached an explicit behaviour end or it receives an external stop command. The behaviour module execution can be described as follows:

- *InitBehaviour()* initialises each behaviour of any required robot devices, and scans robot internal and external status.

- *ExecuteState()* implements robot tasks which are executed as processes and share computational resources for completing a state behaviour.
- *ChooseBehaviour()* chooses next behaviour depending on transition conditions and previous behaviour state. Additionally, it is assigned a confidence value for evaluating the quality of state transition.
- Finally, behaviour can be finished after implied devices are released and robot is sent to a *Stand By* status in which the robot will be waiting for new commands.

Additionally, transitional states are evaluated based on accountability of behaviour performance, frequency of execution and accomplishment of conditional rules for state transition. Therefore, a transition with multiple choices presents a slightly recall to successful states than to ones with lower evaluation value. So, a useful state transition can be traced whenever behaviour states are not being realised as expected.

#### IV. ENVIRONMENTAL INTERFACE

##### A. Localisation Simulator

The first stage for localisation analysis is based on a simulated environment where a moving robot plays with a ball. The robot is simulated as a 2D holonomic movement including position and pose, ball movements and landmark presence as shown in Figure 5. Robot positioning is represented as an absolute value for a football pitch where robot velocities are obtained from the difference between last and current positions. Moreover, robot kinematics values are obtained independently from wheels speeds ( $Speed_{Right}$  and  $Speed_{Left}$ ).

As in previous simulated applications [3], we implement a three-layered software, namely object definition, localisation algorithm and user interface. In this case, each layer has a specific purpose in architectural design, as shown in Figure 6 and described as:

- **Object Layer** - this layer represents an object definition and a graphic representation for robots, ball or landmark, as well as its attributes such as dimensions, kinematics and collisions.
- **Localisation Algorithm** - this layer is in charge of encapsulating localisation algorithms in a *Localisation* object which can be modelled for being used as robot *Localisation* objects with similar input and output. The simulated odometry information is based on a two-wheel robot model with variable noisy speeds.
- **User Interface** - this graphic user interface uses similar front end characteristics of localisation monitor from GUI architecture with options for robot behaviour, pace speed and noise control values.

The simulator is a developing platform to test and evaluate localisation algorithms.

Simulated robot movement

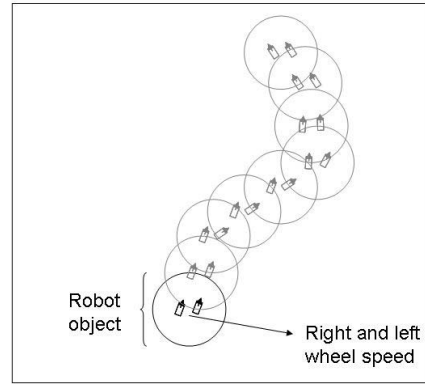


Fig. 5. Simulated holonomic movement in a mobile robot

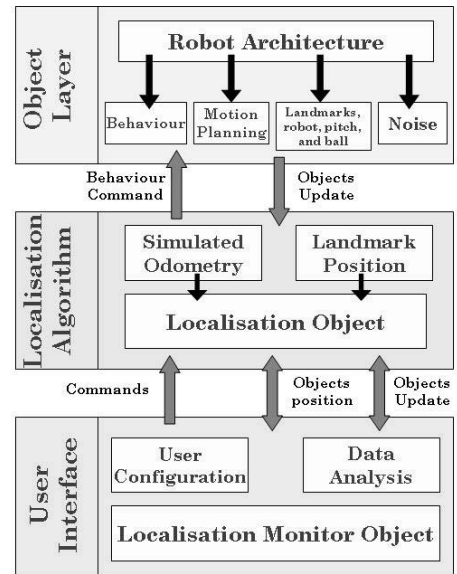


Fig. 6. Three layered architecture for the localisation process

##### B. Tracking system

In order to obtain a grand truth we used an overhead tracking system for recording robot positioning data. The tracking system is a 3D optical VICON system which has less than 10 ms of latency, a positional accuracy of 0.1 mm and an angular accuracy of 0.15 degrees. It is composed of 8 CMOS cameras that are positioned around the robot space and at a height of 4.80 m. Also, it can track up to 150 markers that are made of light reflexive material. The tracking system environment is shown in Figure 7.

An object generated by the tracking system is transmitted, which contains robot heading, position, confidence of perception, time stamp and landmarks position. Five markers are placed on observable joints of a AIBO robot in order to obtain the robot pose  $(x, y, \theta, \varphi)$  through tracking a mass skeleton [6].

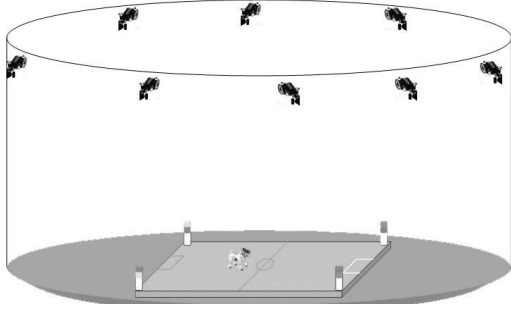


Fig. 7. Overhead VICON tracking system to provide the grand truth

## V. EXPERIMENTS

In this section, EKF, FM and FM-EKF positioning methods are compared. In each experimental set, we analysed simulated and real time experiments which evaluate localisation performance in three types of experiments: (i) simple movements - This stage is realised by a squared trajectory limited by landmark positions along football pitch; (ii) combined behaviours - This stage is composed by a playing session using a single robot which is constantly looking to score; and (iii) kidnapped robot - This stage is realised in randomly sequences of kidnapping by time and pose. In each kidnap the objective is to obtain information about where the robot is and how fast it can localise again. The evaluation criteria for all experiments is defined by the following parameters and shown in Table I:

**Robot Mobility** is the distance covered by robot for a complete experiment obtained from grand truth movement tracking. The total displacement from all experiments is of 186190.46 mm. **Time of Execution** is the time required for each experiment performance. The total time of execution for all the experiments is of 1170.10 s. **Localisation Cycles** is the complete execution of a correct and update step into localisation module. The total amount for these experiments is of 8394 cycles.

Experiment	Mobility	Time	Cycles
Simulated 1	12528.84	160.44	185
Real-Time 1	78821.24	593.67	4878
Simulated 2	12499.48	68.90	268
Real-Time 2	5770.73	38.68	372
Simulated 3	14514.38	38.06	125
Real-Time 3	62055.79	270.36	2566

TABLE I  
EXPERIMENTAL CONDITIONS

The obtained experimental results are compared among Extended Kalman Filter (EKF), Fuzzy-Markov (FM) and Fuzzy-Markov-Kalman (FM-EKF) methods [9]. The experiments also show real-time and simulated trajectories for robot localisation whenever they were in use of the proposed human-robot interface. As can be seen, Simple Movements

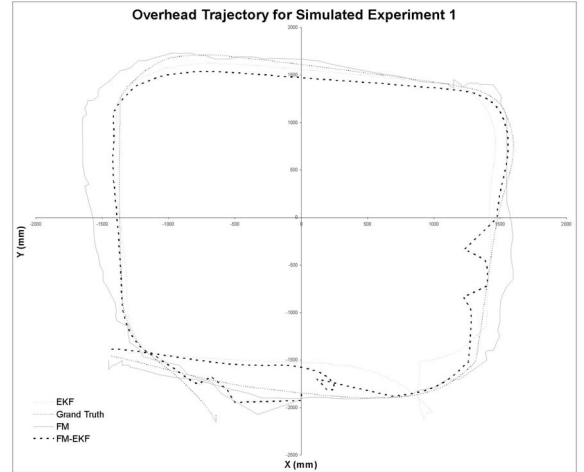


Fig. 8. Simulated experiments during a simple movement

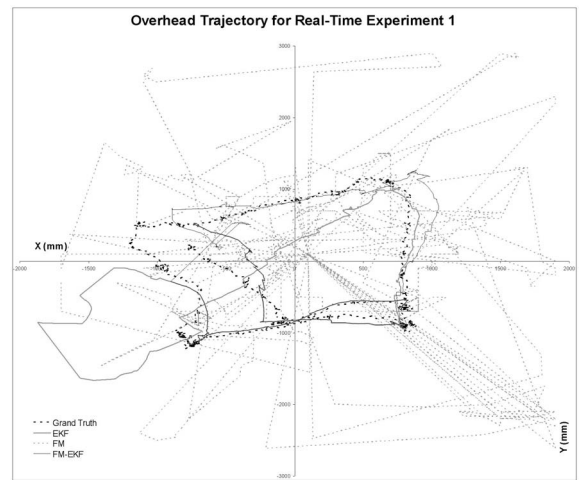


Fig. 9. Real-time experiments during a simple movement.

are shown in Figures 8 and 9, Combined Behaviour in Figures 10 and 11 and Kidnapped Robot in Figures 12 and 13.

## VI. CONCLUSION AND FUTURE WORK

This paper presents an interactive human-robot interface that is a useful tool for the development of robot localisation algorithms in the RoboCup domain. It consists of a walking robot, an overhead visual tracking system and a user interface. Such a system offers real-time control and evaluation facilities for localisation analysis using a GUI as a central support tool. The overhead camera is used to provide grand truth to compare experimental results obtained from real robots with the ground truth. The experimental results demonstrated that the proposed system is feasible and has good performance. Further research will focus on the development of an adaptable behaviour-oriented architecture suitable for any sort of mobile robots with wireless com-

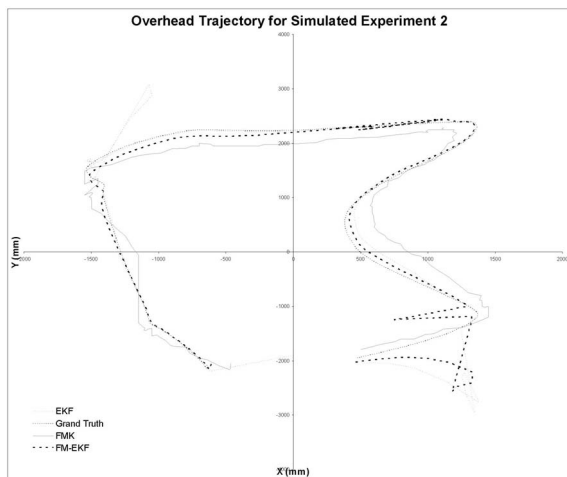


Fig. 10. Simulated experiments during a playing behaviour.

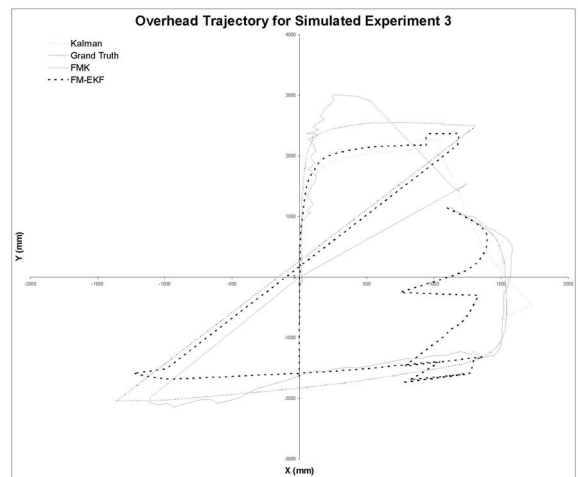


Fig. 12. Simulated experiments during a kidnapping robot trajectory.

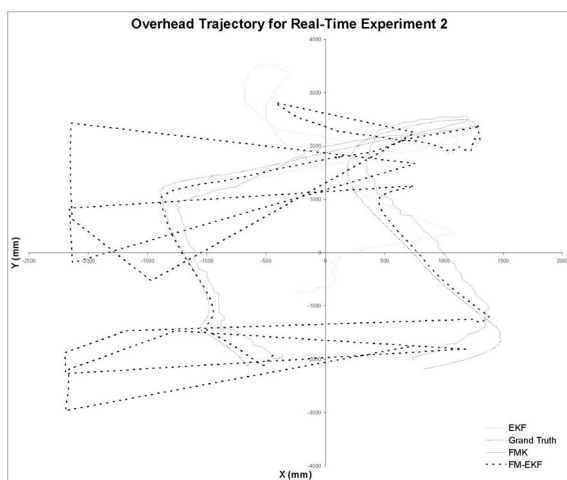


Fig. 11. Real-time experiments during a playing behaviour.

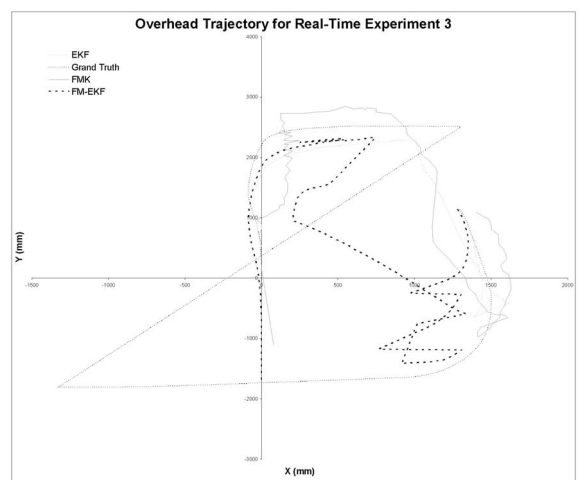


Fig. 13. Real-time experiments during a kidnapping robot trajectory.

munication, including suitable interfaces for user control and behaviour design, robot interaction and online analysis.

**Acknowledgements:** This research was supported by the Mexican government (CONACyT) with scholarship number 178622. Also, many thanks to TeamChaos for the software contribution and technical support of Essex University staff.

#### REFERENCES

- [1] D. Golubovic, B. Li, and H. Hu. A hybrid software platform for sony aibo robots. In *RoboCup 2003: Robot Soccer World Cup VII*, volume 3020/2004, pages 478–486. Springer Berlin / Heidelberg, August 2004.
- [2] V. Hugel, G. Amouroux, T. Costis, P. Bonnin, and P. Blazevic. Specifications and design of graphical interface for hierarchical finite state machines. In *Lecture Notes in Computer Science. RoboCup 2005: Robot Soccer WorldCup IX.*, volume 4020/2006, pages 648–655. Springer Berlin / Heidelberg, 2005.
- [3] J. Liu and H. Hu. 3d simulation of autonomous robotic fishes. *International Journal of Automation and Computing*, 1:42–50, 2004.
- [4] N. Lovell. Real-time embedded vision system development using aibo vision workshop 2. In *Proceedings of the Fifth Mexican International Conference in Computer Science*, pages 268 – 274, 2004.
- [5] F. Martin, R. Gonzalez-Careaga, J. Canas, and V. Matellan. Programming model based on concurrent objects for the aibo robo. In *Proceedings of the XII Jornadas de Concurrencia y Sistemas Distribuidos*, page 367379, June 2004.
- [6] V. MX and V. series Systems. *Polygon 3.1 Visualization and reporting tool. System Tutorial Revision 1.2*. Vicon Motion Systems Limited, 2004-2005.
- [7] H. Ogata and T. Takahashi. Robotic assembly operation teaching in a virtual environment. *IEEE Transactions on Robotics and Automation*, 10(3):391 – 399, June 1994.
- [8] R. Samperio and H. Hu. Kalman filter based localisation for aibo walking robots. In *IEEE International Symposium on Robotics and Automation*, San Miguel Regla Hotel, Hgo., August 25-28 2006.
- [9] R. Samperio, H. Hu, F. Martin, and V. Mantellan. A hybrid approach to fast and accurate localisation for legged robots. *Robotica*, to appear 2007.
- [10] A. Sangpetch. *Visualizing Robot Behavior with Self-Generated Storyboards*. PhD thesis, Carnegie Mellon University, May 10 2005.
- [11] C. Tzafestas, N. Palaiologou, and M. Alifragis. Virtual and remote robotic laboratory: comparative experimental evaluation. *IEEE Transactions on Education*, 49:360– 369, August 2006.