Proceedings of the 2006 IEEE International Conference on Robotics and Automation
Orlando, Florida - May 2006

Augmented Reality Visualisation for Player

T. H. J. Collett and B. A. MacDonald

Department of Electrical and Computer Engineering, The University of Auckland, New Zealand
{*t.collett,b.macdonald*} at *auckland.ac.nz*

***Abstract*— One of the greatest challenges when debugging a robot application is understanding what is going wrong. Robots are embodied in a complex, changing and unpredictable real world, using sensors and actuators that are different from humans'. As a result humans may find the development of robotic software to be difficult and time consuming. We present an augmented reality visualisation tool for the popular open source Player system, that enhances the developers understanding of the robots world view and thus improves the robot development process.**

I. Introduction

Researchers typically program robot applications using an ad hoc combination of tools and languages selected from both traditional application development tools and proprietary robotic tools. However, robotic systems have a number of unique challenges that these tools are not designed to target, particularly:

The nature of the robot environment.

- Environments are dynamic and operate in realtime.
- Unexpected variations cause non-repeatable behaviour and unexpected conditions.

The nature of robot being debugged.

- Robots are mobile (i.e. the target hardware moves away from the programmer).
- A large number of devices are used for input, output and storage, which far exceed human programmers' familiar senses and effectors, compared to the few devices in PCs.
- There are wide variations in hardware and interfaces, as opposed to the highly commoditized desktop.

The nature of mobile robot tasks.

- There is an emphasis on geometry and 3D space
- Complex data types must be represented
- Robot tasks are potentially uninterruptible
- There is simultaneous and unrelated activity on many inputs and outputs

Some of these challenges are also exhibited by some traditional, non-robotic applications, but have generally not been targeted by mainstream development tools. In addition, it is worth noting that concurrency is still not handled well in many debugging environments, although it is often present in traditional applications. This weakness carries over to the robot development domain; robot development tools do not handle concurrency well.

As a result each robotics lab develops its own tools for debugging robot programs, often with different tools for each robot in the lab. Some tools, in particular Player/Stage [1] are emerging as de facto standards since they support a large variety of hardware and have a growing user community. We believe that standardisation in this form is a good thing and aim to promote it by developing Player based tools that support developers in the task of programming robots.

A common thread of these challenges is that the *robot's interaction with the environment* that makes robot programming different and challenging. In other words it is the *programmer's lack of understanding of the robot's world view* that makes robot programs difficult to code and debug.

Augmented reality (AR) provides an ideal presentation of the robot's world view; it displays robot data layered within the real environment. By viewing the data in context with the real world the developer is able to compare the robot's world view against the ground truth of the read world image without needing to know the exact state of the world at all times. This makes clear not only the robot's world view but also the discrepancies between the robot view and reality. The main challenge in AR is to accurately track the human user so that the overlaid data is accurately positioned when the human viewer changes position and orientation. However this is generally a simpler problem than the alternative, which is to track the entirety of a potentially large and dynamic real world environment and explicitly calculate a comparison between the robot view and the actual world view.

The aim of this work is to enhance the developer's interface to the robot by providing AR visualisations of Player's interfaces. The work has been designed to be easily ported to other robot programming tools. Section II gives an overview of Player. Section III presents requirements for effective interaction. Section IV summarises related work in visualisation and AR. Sections V–X present our AR toolkit and tests.

II. Player Overview

Player is a network oriented device server that provides hardware abstraction and distributed access for a wide range of robot hardware [1]–[3]. In Player a *device* represents the binding of a hardware driver, such as the Pioneer driver, to a specific Player interface, such as the sonar interface. Each driver can provide multiple devices, for example the Pioneer robot driver provides both sonar and position devices. Clients initiate communications by connecting to a server and then requesting a subscription to a specific device.

The Player network protocol is modeled on the UNIX file model, each device supports three basic client operations, read

0-7803-9505-0/06/$20.00 ©2006 IEEE

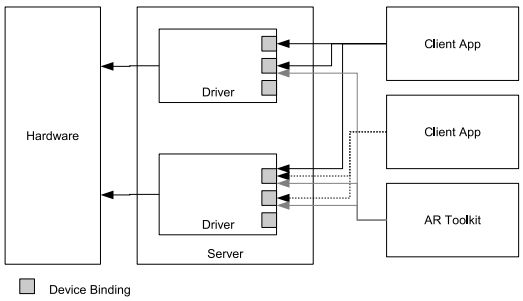


Fig. 1. Overview of Player Structure

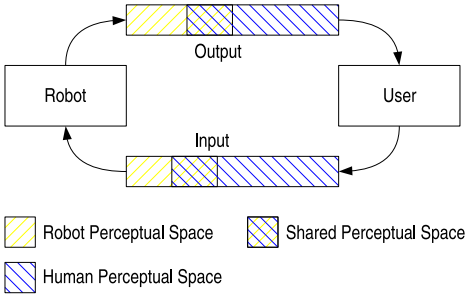


Fig. 2. Shared Human-Robot Perceptual Space.

(Data), write (Commands) and control. The actual communications layer is hidden from the client applications through a client proxy library which processes the low level Player messages. Control messages are implemented as a request and a server response, these are used for a range of tasks such as fetching sensor geometry and configuring sensor parameters.

Fig. 1 gives an overview of a Player system structure. The figure shows two client applications connecting to the Player server as well as the AR tool. The Player server then communicates directly with the hardware. The ability to connect in parallel with the target client application means that the client need not be modified in order to use the visualisation system, and additionally that the visualisation system can be run permanently even when clients are not active.

III. Requirements for Effective Interaction

Breazeal [4] identifies three key requirements for effective human–robot interaction; overlapping perceptual space, appropriate interaction distance, and safety. Also, if the interaction is to be natural, it must use as many interface media as possible.

Both human user and robot have a perceptual space where communication is meaningful, and it is the overlap of these two spaces that is the interaction space of the human-robot coupling (Fig. 2). The interaction space for input may be different from the output space. For example a robot that knows how to represent an emotion using facial expressions, but cannot recognise the facial expressions of others, will have differing input and output spaces. Any mismatch between the perceptual input and output space may put additional cognitive load on the user, who will have to perform additional translations before communicating with the robot.

This paper focuses on providing a shared perceptual space for the robot-developer interaction, which is where the human

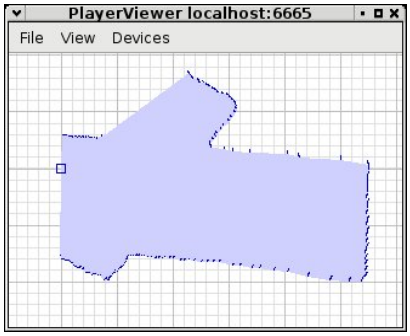


Fig. 3. playerv laser visualisation

and the robot have overlapping perceptual abilities. By allowing the developer to understand the robots world view we are effectively enhancing the shared space as we are allowing the developer to view the world in the same way as the robot.

IV. Related Work

Player [1] has a built in tool, playerv, for visualising its interfaces. However this tool provides the visualisation in isolation. By providing an AR interface for the visualisation we show the data in context with the real objects, making understanding the data more intuitive. Fig. 3 shows an example visualisation of laser data from the existing playerv tool.

Shikata *et al* [5] present an algorithm evaluation system that uses virtual robots in a virtual world to test the performance of avoidance algorithms when human users are involved. The main advantage is that a real human is used rather than a scripted one, thus giving more accurate behaviour while at the same time the virtual robots mean there are no safety issues. In particular the system is used to test avoidance algorithms with no danger to the human user from collisions occurring when the algorithm fails. Milgram *et al* used AR in creating a virtual measuring tape and a virtual tether to assist inserting a peg in a hole [6], [7]. Freund *et al* [8] use AR to more simply display complex 3D state information about autonomous systems. Raghavan *et al* [9] describe an interactive tool for augmenting the real scene during mechanical assembly. Pettersen *et al* [10] present a method to improve the teaching of way points to a painting robot, using AR to show the simulated painting process implied by the taught way points. Brujic-Okretic *et al* [11] describe an AR system that integrates graphical and sensory information for remotely controlling a vehicle.

Daily *et al* [12] use AR to present information from a swarm robotics network for search and rescue. KUKA has began to investigate the use of AR to visualise data during training [13].

Amstutz and Fagg [14] describe an implementation for representing the combined data of a large number of sensors on multiple mobile platforms (potentially robots), using AR and VR to display the information.

V. An AR Tookit for Player Interface Visualisation

Given the rapidly changing nature of available AR hardware and software techniques, to remain relevant any systems

making use of the technology must be flexible and independent of any specific AR implementation.

To be practically effective an AR toolkit must have the following characteristics, which are implemented in our toolkit:

- Modular and flexible in order to make use of core technology advances
- Support a core set of Player interfaces
- Robust, able to be run as a permanent installation in a robots lab
- Detect Player server connections and disconnections
- Underlying AR system must have sufficient accuracy for developer interaction

The software architecture has been designed in a highly modular fashion allowing for individual components to be replaced. While the implemented toolkit is Player based, the modular nature of the AR toolkit allows for the easy addition of support for any other device interface system. Details of the implemented modules are provide in Section VI.

The rendering process for the toolkit is broken into four basic stages: capture, preprocessing, rendering and postprocessing. The rendering stage is further broken down into three rendering layers, each step is described below.

1) Capture: the background frame, orientation and position of the camera are captured.
2) Pre-processing: such as blob tracking for robot registration.
3) Render - Transformation: the position of the render object is extracted from its associated list of position objects, and appropriate view transforms are applied
4) Render - Base: invisible models of any known 3d objects are rendered into the depth buffer. This allows for tracked objects such as the robot to obstruct the view of the virtual data behind them. The colour buffers are not touched as the visual representation of the objects was captured by the camera.
5) Render - Solid: the solid virtual elements are drawn.
6) Render - Transparent: transparent render objects are now drawn while writing to the depth buffer is disabled.
7) Ray Trace: to aid in stereo convergence calculation, the distance to the virtual element in the centre of the view is estimated using ray tracing. This is of particular relevance to stereo AR systems with control of convergence, and optical see through stereo systems.
8) Post Process: once the frame is rendered any post processing or secondary output modules are called. This allows the completed frame to be read out of the frame buffer and, for example, encoded to a movie stream.

The toolkit architecture is centred around an output device. Each output device contains a capture device for grabbing the real world frame (this could be a null object for optical see through AR, or for a purely virtual environment) and a camera device for returning the camera parameters, including pose. Also, the output device maintains three component lists: secondary outputs (which monitor the interface, i.e. movie capture); preprocessing objects (used for image based position tracking); and finally a list of render item pairs. Each *Render Pair* consists of a render object that performs the actual rendering of a virtual element and a list of positions object that define where it should be rendered.

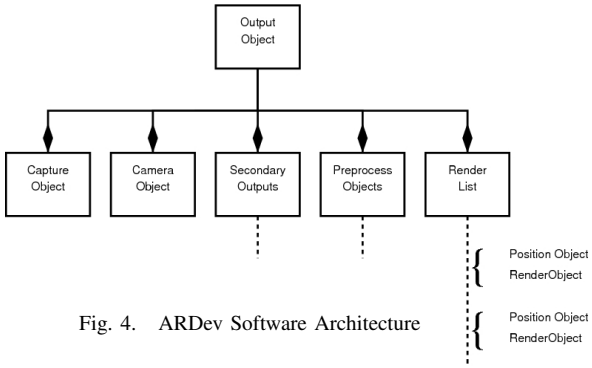


Fig. 4. ARDev Software Architecture

Fig. 4 shows the software structure and each of these components is summarised in Table I. The first four items (Capture, Camera, Secondary output and Preprocessing) are all unique to the output object. The render objects and position objects can be used in multiple combinations, potentially with different output objects. For example a Stereo head mounted display (HMD) needs the same laser data (render object) on both displays (output objects), while for a single output the same origin (position object) could be used to render both laser and sonar data.

To create a permanent lab setup it was important to detect the presence of robots. This presence is in two forms, first the system must cope with network or software presence, if a robot is switched on or off the system must detect this and if required start (or stop) displaying appropriate data. Secondly, if a robot is physically removed from the system and unable to be tracked then the data for that robot must not be displayed, this is particularly relevant for optically tracked robots. In order to achieve these goals a presence variable was added to the PositionObject indicating whether it is successfully being tracked, and the Player render objects now check their current connection state during the update cycle and attempt to connect to the Player server if currently disconnected. In order for the Player connections to function correctly a small patch is needed for the Player 1.x series to add timeouts to some socket methods. Player 2.x will support this directly.

VI. Implemented Modules

This section discusses a selection of the implemented objects in more detail. Table II lists the currently implemented objects included with the toolkit. Specifically visualisations have been implemented for the following Player interfaces: Position2D, Sonar, Laser, IR, Bumper, PTZ, Map, Localize.

The Player rendering modules all have a similar format. The modules connected to a single server share a PlayerClient object which provides the connection. The PlayerClient object connects to the server on initialisation. If at any stage the server disconnects then the object changes to the uninitialised state causing any future request for Player data to trigger a new connection attempt. Using this method the system can

ObjectName	Description
OutputObject	Handles the actual rendering of the information and its display
CaptureObject	Supplies the real world input frames, for example from a Video for Linux device, or a blank frame for optical see through AR
CameraObject	Provides the geometry and optical properties of the camera
SecondaryOutputObject	Allows a secondary output stream to exist, such as output to video, or capturing stills
PreProcessObject	Handles processing of a frame before rendering (allows for tracking of markers using image processing)
PositionObject	Returns the position of a tracked object, relative to an arbitrary base
RenderObject	Handles the rendering of a single augmented entity

TABLE I
CORE OBJECTS

Name	Description
General Objects	
CaptureV4l	Image capture from Video 4 Linux API
OutputX11	Output to an X11 Display
OutputMovie	Output to a movie file using the ffmpeg library
RenderModel	Rendering of 3ds model file
RenderB21rBase	Rendering of B21r invisible base
Player [1] based objects	
CameraFile	Access to static pre-calibrated camera data
CameraPlayer	Access to the pan tilt interface
CameraStage	Camera based on stage simulation
CameraPlayerPosition3D	Camera using position 3D interface
CaptureStage	Simulated camera based on stage simulation
PositionPlayer	Access to position 2D interface
PositionPlayer3D	Access to position 3D interface
RenderPlayerLaser	Rendering of Player laser data
RenderPlayerSonar	Rendering of Player sonar data
RenderPlayerIr	Rendering of Player IR data
RenderPlayerPath	Rendering of past Player position information
RenderPlayerBumper	Rendering of Player bumper data
RenderPlayerPTZ	Rendering of Player PTZ data
RenderPlayerMap	Rendering of Player map data
RenderPlayerLocalise	Rendering of Player localisation data
Blob Tracking Objects	
OpenCVBlobTrackPreProcess	Used to process incoming frames
OpenCVBlobTrackPosition	Uses data from processed frames to give the 3D position of markers

TABLE II
KEY OBJECTS IMPLEMENTED IN THE PROTOTYPE SYSTEM

detect robot connections and disconnections and is thus able to run continuously.

The CameraFile module provides camera pose and image parameters from a pre-calculated calibration file. This data is then used to place the openGL camera in the scene before rendering. A calibration utility is provided with the toolkit which uses a tsai calibration [15] to extract the camera parameters from a set of 27 matched 3d and image points. The utility is capable of extracting the points automatically using simple grayscale segmentation if an appropriate calibration object is used, or alternatively the points can be manually extracted and provided in a file.

```
class RenderPlayerLaser : public RenderObject
{
  public:
    /// Constructor & Destructor
    ...

    int Initialise(bool Active = true);
    void Terminate();

    void Render(); ///< Render the laser object
    void RenderTransparent(); ///< Render the laser object

    /// calculate ray trace distance
    virtual double TraceDistance(const Ray & Offset, const Ray & R, const ARPoint &
            Rotation);
};
```

Fig. 5. Header file for laser render object

The OpenCVBlobTrackPreProcess module provides position estimation from pairs of coloured blobs in the captured image. These are extracted using hue based segmentation, with the following stages:

- Separation to HSV colour planes
- Smoothing the hue and saturation images
- Masking low saturation and value areas of the hue image
- Thresholding to isolate hue values
- Extraction of blob contours
- Geometric tests to remove invalid blobs
- Selection of the best remaining segment

The implementation uses the opencv [16] image processing library. The OpenCVBlobTrackPosition module projects these image points into 3d space on a predefined plane and the resulting 3d positions are passed to the rendering process.

The RenderB21rBase object is an example of a depth only model. The object implements the RenderBase method, which allows it to render the depth information for the existing B21r image data captured by the camera. This data is used to obscure virtual elements that are 'behind' the robot.

VII. ARIDE User Interface

In order to make the Player visualisations simpler to configure and use, a graphical utility, ARIDE, is provided with the toolkit. This utility allows easy configuration of AR details for a robot. This configuration can then be stored in an XML configuration file. The utility also allows control of the visualisation, the user can run, pause and terminate it.

VIII. Example Render Object

Fig. 5 shows an outline header of a sample RenderObject. The key methods to implement are: Initialise and Terminate, these are called for each object as the AR system initialises and terminates respectively; Render and RenderTransparent to perform the actual rendering; and TraceDistance if you wish to support the convergence calculation. The actual data update is the responsibility of the individual driver, which can either spawn its own thread for the data updates, or perform them in the server thread when Render is called. This simple API allows for new render objects to be very simply implemented.

IX. System Tests

The toolkit has been tested with two core AR configurations. A magnetically tracked video see-through HMD (Trivisio

ARVision3D) provides an immersive, stereo environment for a single user. A fixed overhead camera and a large wall mounted AR display nearby the robots, provides a communal environment where humans and robots may interact together.

The toolkit has been tested with three different Player compatible robots, the B21r robot from iRobot, the Pioneer 3DX from ActivMedia and our own in house Shuriken robot, representing a rich set of capabilities, and a range of scales.

Sample output of these systems is shown in Fig. 6.

The visualisation has been run continuously in the lab for 7 days, at which point it was stopped for system upgrades.

X. Discussion & System Performance

As seen in the screen shots the AR system provides an effective means of visualising geometric data sets, for example in Fig. 6(e) it is immediately obvious that the sonar sensor (blue) is unable to correctly measure the distance to the box, whereas the IR sensor (purple) correctly measures this value.

Users of the system found it useful in aiding understanding of robot behaviour. Users unfamiliar with basic robot sensors such as ultrasonic found the system useful for gaining an understanding of the inherent limitations of the sensors (i.e. the effect of the angle of incidence, and minimum range on ultrasonic sensor readings, in Figs. 6(d) and 6(e) respectively).

The map visualisation is useful for visually validating the accuracy of the map data. By overlaying the map on the real world, in Fig. 6(i), any discrepancies are immediately obvious.

The HMD setup has definite advantages in the contextual relevance of data and immersion. However, users experimenting with the system found it difficult to position themselves to obtain the desired view of the robot data, partly due to the cabling restrictions and partly due to the proximity of the robot. In addition, HMD based systems are expensive and technically difficult to implement.

User convenience is also an issue as the user must actively choose when to put on the HMD and when to remove it, returning to the desktop based programming environment. This means that the HMD system will only be used when there is a serious problem the user needs to spend significant time on.

These drawbacks will largely be solved by developments in tracking and registration within the AR community, the development of cheaper, lighter and more capable displays, and advancements in wearable computing and wireless technology. User convenience will still be an important issue as will the user's work flow between the desktop and AR systems.

The overhead setup can be permanently installed in a robotic development facility, creating an intelligent space for robot development where multiple developers can work with many robots. The developer can glance at the display to obtain additional information about the robot's state and intentions.

The overhead AR system utilises lower cost components than the head mounted system, as such it can be used in a wider variety of applications. The setup used for this research will remain available in the robotics lab for use in undergraduate research projects and courses allowing a rich set of experiments that will help improve the system. If greater coverage is needed with the overhead AR then additional cameras and displays can be added.

A combined setup provides a good balance of the advantages of both the low cost overhead camera system for general purpose debugging, and an HMD for difficult problems requiring greater immersion.

The speed and accuracy of the toolkit are heavily dependant on the particular hardware setup and algorithms used. In the case of the overhead camera, registration accuracy is dominated by the camera calibration accuracy and the accuracy of the robot model. System tests showed an error of less than 1 pixel, or around 3mm at a distance of 3m. The overhead system currently runs at 6 frames per second; this is limited mainly by the capture rate of the networked based implementation, and the processing time to track the robot. The performance was acceptable for the target application.

System robustness has been tested in an installation in our lab using an overhead camera and large screen display. The system has been run for more than 7 days continuously without issues before being taken down for upgrades. With a patched version of Player (too add timeouts on connects and reads) the system is also robust to robot connections and disconnections.

XI. Conclusions

Robot programmers are faced with the challenging problem of understanding the robot's view of its world, when both creating and debugging robot software. Requirements for effective interaction under these conditions include the need to maximise the overlapping perceptual space of robots and humans. An AR system can increase this overlap by overlaying information about the robot's sensory data on a real view of the robot and its environment. Input, output, state and algorithmic information may be visualised.

Our AR toolkit provides intuitive representations of geometric data from Player interfaces for robot developers. The use of AR for visualisations gives the developer an immediate understanding of the robot data in context with the real world baseline, allowing limitations in the robot's world view to be understood, and giving faster solutions to related software issues. Our initial implementation of the concepts has shown useful and promising results for robot software development.

The full source for the library is available for download from `http://robotics.ece.auckland.ac.nz`.

Acknowledgement

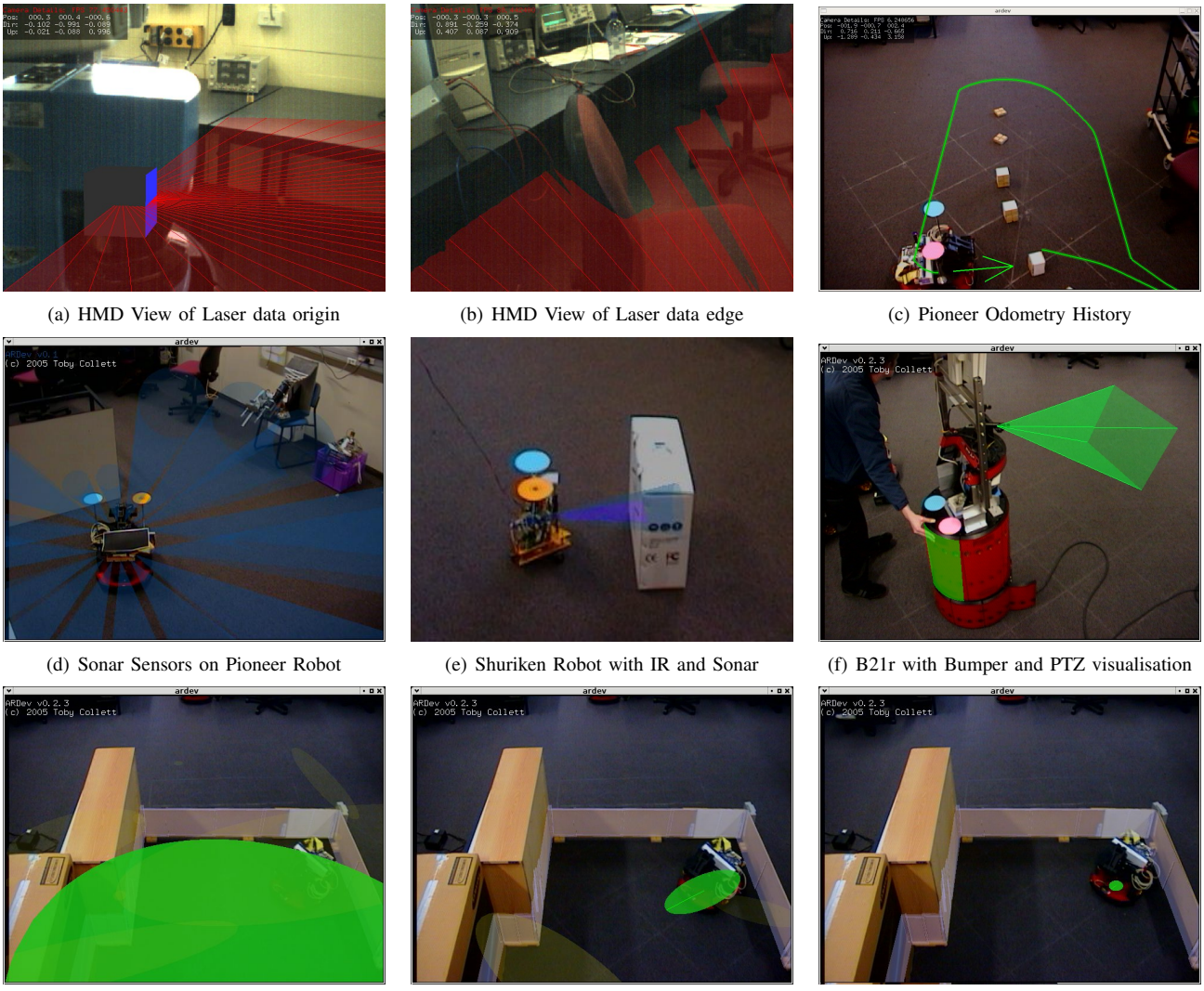
Toby Collett is funded by a top achiever doctoral scholarship from the New Zealand Tertiary Education Commission.

References

[1] Player/Stage. (2005, January) The player/stage project. http://playerstage.sourceforge.net/.

[2] B. P. Gerkey, R. T. Vaughan, K. Støy, A. Howard, G. S. Sukhtame, and M. J. Matarić, "Most Valuable Player: A Robot Device Server for Distributed Control," in *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, Wailea, Hawaii, Oct. 2001, pp. 1226–1231.

[3] B. P. Gerkey, R. T. Vaughan, and A. Howard, "The Player/Stage Project: Tools for Multi-Robot and Distributed Sensor Systems," in *Proc. of the Intl. Conf. on Advanced Robotics (ICAR)*, Coimbra, Portugal, July 2003, pp. 317–323.



(a) HMD View of Laser data origin

(b) HMD View of Laser data edge

(c) Pioneer Odometry History

(d) Sonar Sensors on Pioneer Robot

(e) Shuriken Robot with IR and Sonar

(f) B21r with Bumper and PTZ visualisation

(g) AMCL Initial Distribution, High Covariance

(h) AMCL Improved Estimate

(i) AMCL Converged Result

Fig. 6. AR System Output

[4] C. Breazeal, A. Edsinger, P. Fitzpatrick, and B. Scassellati, "Active vision for sociable robots," *IEEE Trans. Syst., Man, Cybern. A*, vol. 31, no. 5, pp. 443–453, 2001.

[5] R. Shikata, T. Goto, H. Noborio, and H. Ishiguro, "Wearable-based evaluation of human-robot interactions in robot path-planning," in *Proc. IEEE International Conference on Robotics and Automation (ICRA 03)*, vol. 2, 2003, pp. 1946–1953.

[6] P. Milgram, S. Zhai, D. Drascic, and J. J. Grodski, "Applications of augmented reality for human-robot communication," in *Proc. IEEE/RSJ International Conference on Intelligent Robots and System (IROS 93)*, vol. 3, 1993, pp. 1467–1472.

[7] P. Milgram, A. Rastogi, and J. Grodski, "Telerobotic control using augmented reality," in *Proceedings., 4th IEEE International Workshop on Robot and Human Communication. RO-MAN'95*, Tokyo, 5–7 July 1995, pp. 21–9.

[8] E. Freund, M. Schluse, and J. Rossmann, "State oriented modeling as enabling technology for projective virtual reality," in *Proc. IEEE/RSJ International Conference on Intelligent Robots and System (IROS 01)*, vol. 4, 2001, pp. 1842–1847.

[9] V. Raghavan, J. Molineros, and R. Sharma, "Interactive evaluation of assembly sequences using augmented reality," *Robotics and Automation, IEEE Transactions on*, vol. 15, no. 3, pp. 435–449, 1999.

[10] T. Pettersen, J. Pretlove, C. Skourup, T. Engedal, and T. Lokstad, "Augmented reality for programming industrial robots," in *Proceedings of the Second IEEE and ACM International Symposium on Mixed and Augmented Reality*, 7–10 Oct 2003, pp. 319–20.

[11] V. Brujic-Okretic, J.-Y. Guillemaut, L. Hitchin, M. Michielen, and G. Parker, "Remote vehicle manoeuvring using augmented reality," in *International Conference on Visual Information Engineering. VIE 2003.*, 7–9 July 2003, pp. 186–9.

[12] M. Daily, Y. Cho, K. Martin, and D. Payton, "World embedded interfaces for human-robot interaction," in *Proc. 36th Annual Hawaii International Conference on System Sciences*, 2003, pp. 125–130.

[13] R. Bischoff and A. Kazi, "Perspectives on augmented reality based human-robot interaction with industrial robots," in *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 04)*, 2004, pp. 3226–3231.

[14] P. Amstutz and A. Fagg, "Real time visualization of robot state with mobile virtual reality," in *Proc. IEEE International Conference on Robotics and Automation (ICRA 02)*, vol. 1, 2002, pp. 241–247.

[15] R. Y. Tsai, "A versatile camera calibration technique for high-accuracy 3d machine vision metrology using off-the-shelf tv cameras and lenses," *IEEE Journal of Robotics and Automation*, vol. RA-3, no. 4, pp. 323–344, August 1987.

[16] Open Source Computer Vision Library. (2005, September) http://www.intel.com/technology/computing/opencv/index.htm.