

# Developer Oriented Visualisation of a Robot Program

## An Augmented Reality Approach

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

Department of Electrical and Computer Engineering, The University of Auckland, New Zealand

t.collett at auckland.ac.nz, b.macdonald at auckland.ac.nz

### ABSTRACT

Robot programmers are faced with the challenging problem of understanding the robot's view of its world, both when creating and when debugging robot software. As a result tools are created as needed in different laboratories for different robots and different applications. We discuss the requirements for effective interaction under these conditions, and propose an augmented reality approach to visualising robot input, output and state information, including geometric data such as laser range scans, temporal data such as the past robot path, conditional data such as possible future robot paths, and statistical data such as localisation distributions. The visualisation techniques must scale appropriately as robot data and complexity increases. Our current progress in developing a robot visualisation toolkit is presented.

### Categories and Subject Descriptors

I.2.9 [Artificial Intelligence]: Robotics—*operator interfaces*

### General Terms

Algorithms, Human Factors

## 1. INTRODUCTION

Currently when researchers wish to program robot applications they tend to use 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
  - Dynamic, realtime
  - Unexpected variations cause non-repeatable behaviour and unexpected conditions

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

HRI'06, March 2–4, 2006, Salt Lake City, Utah, USA.  
Copyright 2006 ACM 1-59593-294-1/06/0003 ...\$5.00.

- The nature of the robot being debugged
  - Mobility (i.e. the debugging target hardware moves away from the programmer)
  - A large number of devices for input, output and storage, which far exceed human programmers' familiar senses and effectors, compared to the few devices in a desktop or server.
  - Wide variations in hardware and interfaces, as opposed to the highly commoditized desktop.
- The nature of mobile robot tasks
  - Emphasis on geometry and 3D space
  - Complex data types and environments
  - Potentially uninterruptible
  - Simultaneous and unrelated activity on many inputs and outputs

Some of these challenges are also exhibited by some traditional, non-robotic applications, but they 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.

The result is that each robotics lab develops its own tools to aid in debugging the complexities of robot programming, often with different tools for each robot in the lab, and while some of these tools (such as Player/Stage [15]) are becoming more widely used, most of the tools need to be reinvented for each new robot system or driver.

A common thread of these challenges is that the *environment* and the robot's *interaction with the environment* are what 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 real 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 with the real world. The main challenge in AR is to accurately track the human user so that the overlaid data is accurately positioned as the human viewer

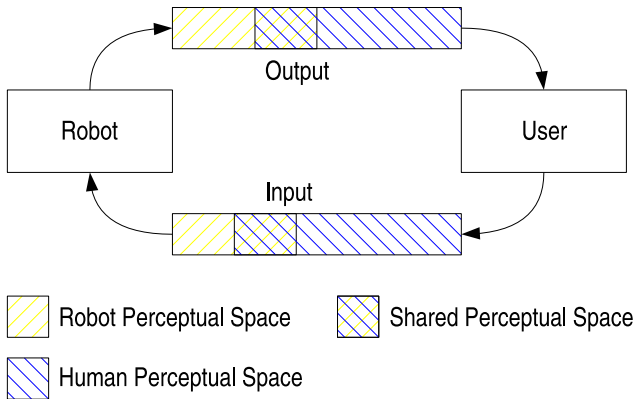


Figure 1: Shared Human-Robot Perceptual Space

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 system in order to promote a better understanding of the robot’s world view. We begin by outlining the nature of robot programs and giving an analysis of what the developer needs to see visualised. Following this we introduce a possible solution by graphically displaying robot sensor data in an AR system.

## 2. REQUIREMENTS FOR EFFECTIVE INTERACTION

Breazeal [4] identifies three key requirements for effective human–robot interaction; overlapping perceptual space, appropriate interaction distance, and safety.

Both the human user and the 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 (Figure 1). 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 its interaction partner 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.

By allowing the developer to understand the robot’s 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.

## 3. RELATED WORK

A number of existing robot applications have visualisations built in to aid in using the systems. Player/Stage [15] provides distributed hardware abstraction and a number of other robot algorithms. Player/Stage includes visualisations in the form of the *playerv* tool, an example laser visualisation is shown in Figure 2. CARMEN [6] provides navigation services with a range of visualisations for the robot’s environment map (Figure 3). Lego Mindstorms Vision Command image processing interface shows processing results [11], and

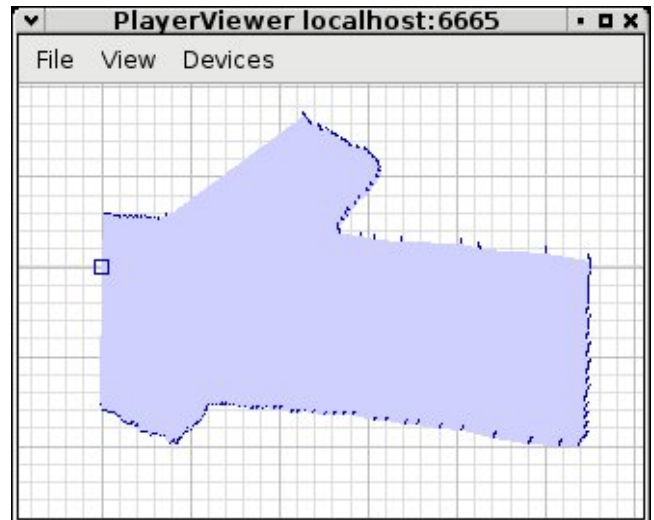


Figure 2: playerv laser visualisation

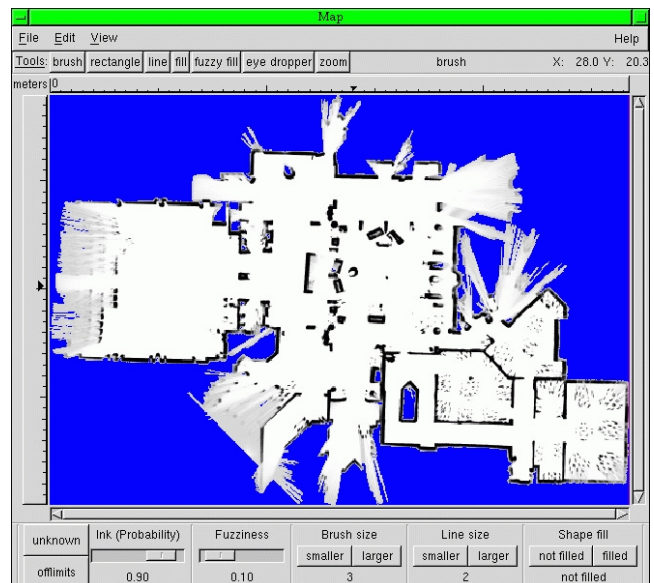


Figure 3: CARMEN Map Editor with map visualisation (from [6])

the GSV tool provides general visualisation within a real-time 3d game engine [18]. UsarSIM also provides a robot simulation with a 3d game engine [19]. Individual sensor manufacturers also provide visualisation system with their hardware, for example Hokuyo Automatic provides an Open GL visualisation of the URG laser range scan including historical data (Figure 4) [2]. These systems either present the data in isolation, with no mapping to a world view, or rely on a static pre-built map of the world.

In addition to these basic robot visualisations, AR and virtual reality (VR) have been used in a number of different robotic applications.

Shikata *et al* [17] 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 be-

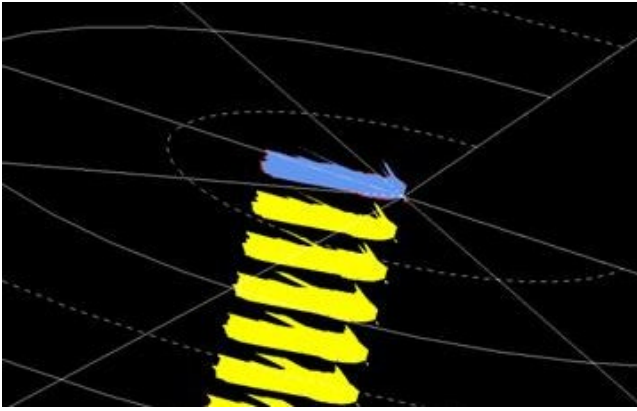


Figure 4: Hokuyo URG laser visualisation

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 [12,13]. Freund *et al* [10] use AR to more simply display complex 3D state information about autonomous systems. Raghavan *et al* [16] describe an interactive tool for augmenting the real scene during mechanical assembly. Pettersen *et al* [14] 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* [5] describe an AR system that integrates graphical and sensory information for remotely controlling a vehicle.

Daily *et al* [7] use AR to present information from a swarm robotics network designed for search and rescue. Recently KUKA have also begun to investigate AR to help visualise data during training [3].

Amstutz and Fagg [1] 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.

This paper focuses on a toolkit for developer oriented AR, initially for sensor visualisation, with an emphasis on understanding the requirements and principles involved.

#### 4. VISUALISING A ROBOT PROGRAM

The visualisation of a robot program can either be approached as an automatic process, where the developer simply writes their code and the system infers how to visualise it, or as a manual process, where the developer is provided with a toolkit, but is expected to generate the visualisation themselves. The pragmatic approach is to attempt to provide automatic visualisations for as much of the system as possible, while still providing a developer with the ability to easily generate their own custom visualisations.

Automatic visualisation for robotics is equivalent to the abilities of traditional debuggers to meaningfully display basic data types such as strings and simple structures. Figure 5 shows a screenshot of the Eclipse debugger [9], presenting the structure of the player laser proxy data. Figure 6 shows a screenshot of the DDD debugger's graphical visualisation for a list structure [8]. Most current developer Integrated Development Environments (IDEs) take the ap-

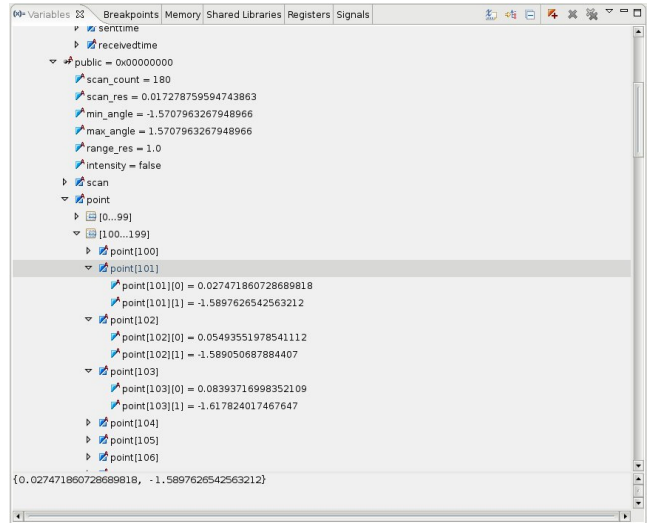


Figure 5: Eclipse IDE debug window

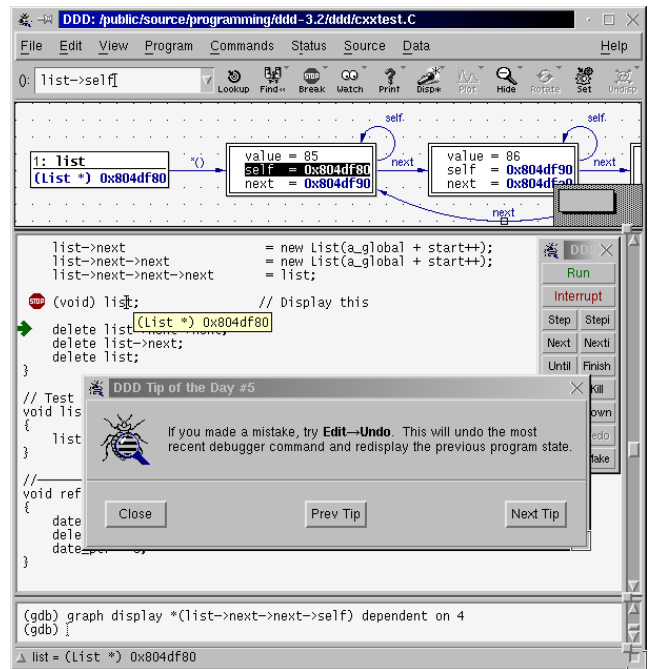


Figure 6: DDD visualisation of a list structure (from [8])

proach of displaying “text plus.” The fundamental data is displayed as text, but it is augmented by simple tree structures and colour, like those shown in Figures 5 and 6.

The ability to generate a custom visualisation is akin to manual debugging output created by each programmer inserting print statements; here the developer can concisely summarise printed information about execution in a particular piece of code as they know what it means. Custom visualisations can be powerful if included in base class designs; providing visualisations for many software objects with a minimum of coding.

In order for automatic visualisation to be useful the program being visualised must have some form of known structure or data types. Conceptually the program can be viewed as having four elements: input, output, state and kernel. If

we require the input and output of the program to conform to standard interfaces then it becomes possible to implement standard visualisations of these, and thus provide automatic visualisations. The state involves the internal data structures a robot program uses, and the kernel the main algorithms.

The state of the program is more difficult to visualise automatically as it does not generally conform to a predefined structure. However at a finer level of granularity individual components of the internal state can have standard visualisations written, such as for a predefined set of geometric datatypes. The difficulty of this approach is ensuring its scalability, the best visualisation for a complex system may not be the visualisation of all its components combined together. For example in an imaging system that tracks colour blobs it may not be useful to display the location of every possible match within an image. Instead a selection of the most likely matches could be more useful. However the criteria would depend on the application; it could involve the single most likely match, the top 10, all the possible matches above a certain size, or some other criteria. It may be difficult to provide a useful automatic visualisation. Given the problem of scalability, for our approach to be useful we need to provide meaningful groupings of the known data types.

A method of easing the problem of visualising internal state is to create robot programs as standalone fragments that communicate through standard interfaces. Here we force the system to adhere to a standard but amorphous structure providing ‘checkpoints’ where the data is in a known form, reducing the amount of freeform program that may require manual visualisations to be written.

Apart from the environmental nature of robot data, such as geometry based representations, the kernels of robot programs are very similar to standard applications, utilising many of the same algorithmic techniques. The visualisation of these algorithmic elements has already received significant attention in areas such as graphical programming and we will not examine these issues here.

It is important to emphasise at this point that we wish to visualise the robot’s view of the world, via its sensor data and historical state. A display of the actual world state is however useful as a baseline reference, if the actual state is known. For example if a robot program is receiving erroneous sonar data, the developer must see this data just as it is, even if the correct value is known to the visualisation system. An AR system inherently shows the correct value alongside the incorrect sensed value, to allow the developer to isolate the differences between the robot’s world view and the actual world state quickly, and program the robot to act accordingly.

The remainder of this paper considers the automatic visualisation of robot data, and the relationships between the data. The issues of manual visualisation and kernel visualisation will be discussed elsewhere.

## 4.1 Characterising Robot Data

Robot data that is input, output or internal state can be characterised in a number of different ways. Here we focus on distinctions that alter the way we desire to view the data. Important distinctions include whether the data is:

- geometric or abstract in nature

- temporal or static
- conditional
- statistical

For example the current laser scan of the robot is geometric data, which may also have a statistical aspect if the measurement error is known. A security robot could use temporal data to represent the time since the last scan of particular areas of its patrol route. Temporal data also applies to scheduled events, for example when a vacuuming robot operates only during certain hours of the day to avoid disturbing residents. Conditional data generally is used to represent an action that might happen, for example options in a planned path, such as backing off if a person approaches too closely.

In general we can display data as textual information, graphical information or a combination of the two. Other display techniques are also available, such as aural or textual display, but these are less generally applicable for information display (for example aural display is often suited to warnings or exclamations rather than data display).

Given the environmental nature of our robotic data we will explore graphically dominated methods of display. This does not mean that there can not be effective textual representations of the data, but we expect that more progress will initially be gained by exploring graphical visualisations that have a direction relationship with the real scene.

The following paragraphs give an outline of the key elements of each type of data, however the ideal method of visualising each will of course be application dependant. Because of this variety of potential visualisations the AR toolkit presented in Section 5 has been designed to allow the developer to create the best visualisation for their application and platform.

### 4.1.1 Geometric Data

To display geometric data we need to display the physical structure of the geometry the data represents, preferably in context on a global view where this is relevant to the data.

For example sonar data could be displayed either as a line or cone that originates from the sensor origin up to the length of the sonar reading. There are an unlimited number of variations in exactly how the visualisation will appear, and the choice of a specific one will largely depend on the specific environment and the purpose of the environment. For example if a sonar reading is displayed as a cone this will give an indication of exactly what the robot can and cannot see with that particular sensor, however displaying the sonar reading as a single ray will often reflect the robots internal model of the sensor more accurately. Figure 2 shows the display of laser scans as geometric data and Figure 9(a) gives an example of an AR representation.

Depending on the complexity of the geometric data set, a method of grouping the individual data points allows the developer to more easily grasp the complete data set.

### 4.1.2 Temporal Data

Temporal data has two important aspects that we need to be able to represent, its age and its dynamic nature. Generally with a temporal data set the elements that are further away from the present time (either in the future or in the past) are less relevant and so emphasis needs to be placed on

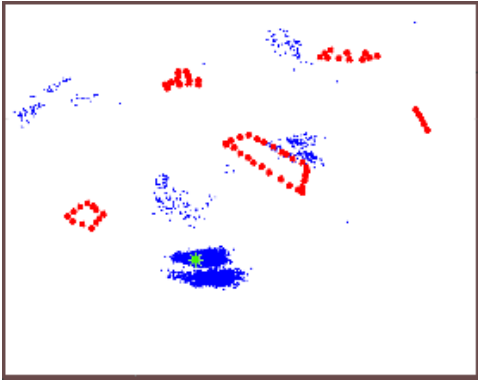


Figure 7: SLAM data representation; the robot position as a green star, the probable robot positions in fine blue dots, laser scan data in coarse red dots. [from David Yuen with permission].

samples with close temporal proximity. Data elements with a dynamic nature, such as moving objects, need a simple way of displaying this.

Temporal data in current robot programs is dominated by historic data, for example the path the robot has travelled and historic sensor readings. The important aspect of visualising this data is to render its temporal value in a non invasive way. One technique for is to graphically age data, for example slowly fading out areas of a map that have not been visited for some length of time. Dynamic data could be shown as vectors or as predicted and historical paths for the moving object. Figures 4 and 10(b) respectively show historic data for a laser scanner and odometry readings

Some aspects of historical data can be treated as statistical data since the accuracy becomes more uncertain as the data becomes older. This could be the case for a security robot that estimates a risk level for its patrol zones based partly on how long ago they were last visited.

#### 4.1.3 Conditional Data

The key aspects of conditional data are that there is a set of potential outcomes (a minimum of 2), and that there is a condition or set of conditions that will affect the outcome.

Conditional data can use a graphical method, such as transparency, to indicate that the data is not yet certain. In addition to this the links between cause and effect could be used to good effect, showing what a particular piece of data depends on to become current.

#### 4.1.4 Statistical Data

Visualisation of statistical data must show the distribution of the data, or the certainty that it is a particular value. The minimum, maximum and expected values can be shown graphically. Intensity or transparency can be varied according to the statistical distribution (eg a Gaussian distribution of intensity can be depicted around the mean value).

Many simultaneous localisation and mapping (SLAM) systems use an ellipse or sampled distribution to represent the probable location of the robot (see for example [20]). The distribution peaks broaden as the location becomes less certain. Figure 7 shows a sampled distribution of possible robot positions (in fine blue dots), along with range data (in coarse red dots) and the actual robot position (the green star); our goal would be to overlay this on the real view of the world.

## 4.2 More than just data

A key concern we have already mentioned is that of scalability in the visualisation of robot programs. Just because the readings of a sonar sensor can be effectively displayed in an intuitive way does not mean that an entire robot system can be displayed so easily. The scaling problem is worse for a multirobot system. As more information must be displayed, it becomes more difficult to add additional, clear, meaningful visualisations.

We can deal with this problem in two main ways, by making the visualisations themselves more scalable and by creating a user interface for the visualisation system that gives the developer sufficient control over how and what is displayed, so that they can still carry out meaningful work with a complex robot system.

To make the visualisations themselves more scalable we must look at the relationships between the different elements to be visualised. This might involve grouping similar elements, such as rendering the outline of a laser scan rather than each individual value, or colouring related items in the same colour. We can also look at the links between otherwise unrelated datasets such as conditional outputs that rely on certain inputs. Finally we can provide alternate visualisations for groupings, rather than just displaying each visualisation from the group together.

The user interface for the developer should ideally be integrated into the existing developer environment, either as part of the robot interface or as part of an IDE. The user interface needs to give the user the ability to control the visualisation, in particular the ability to select which elements to visualise, where to visualise them, the layering of components and control of the grouping of components. To a large extent the features of the visualisation user interface parallel those of a modern window manager in a graphical PC environment, dealing with visualisation elements instead of windows. Just as a programmer opens, closes, resizes, moves and manipulates windows to carry out programming and debugging in graphical user interface, we expect a robot developer to manipulate visualisations in a robot AR environment.

## 5. AN AUGMENTED REALITY TOOLKIT FOR SENSOR VISUALISATION

The remainder of this paper describes our prototype AR developer-robot interaction system. The initial stage of implementation has focused on the development of a sensor visualisation system to aid developer understanding of the robot's world view. Only visualisation of geometric data has been considered at this stage.

Given the rapidly changing nature of available AR hardware and software techniques any systems making use of the technology must be flexible and independent of any specific AR implementation.

The software architecture used for this research has been designed in a highly modular fashion allowing for individual components to be replaced as required. The prototype also provides reference implementations of each type of object, particularly using the Player/Stage project [15], and Video for Linux capture devices. The full source for the library is available for download from <http://robotics.ece.auckland.ac.nz/>.

The rendering process for the toolkit is broken into four basic stages: capture, preprocessing, rendering and post-processing. The rendering stage is further broken down into three rendering layers and a ray trace step is optionally performed to enhance stereo representations. 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-processing: 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 chain of position objects that define where it should be rendered.

Figure 8 shows the software structure. 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 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.

Currently we are utilising two basic setups for our AR work, the first is a overhead camera with a wall mounted

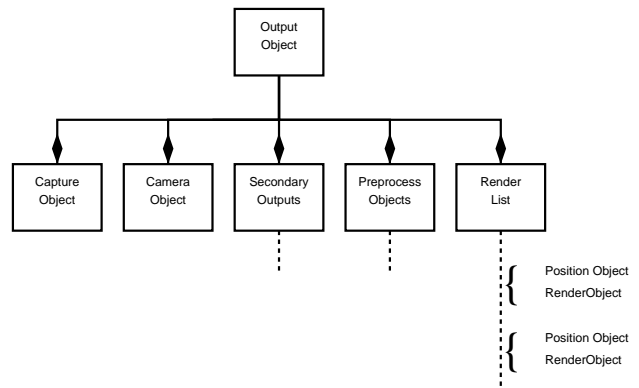
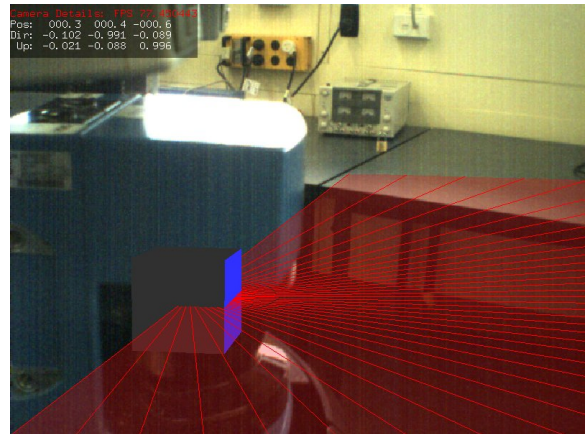
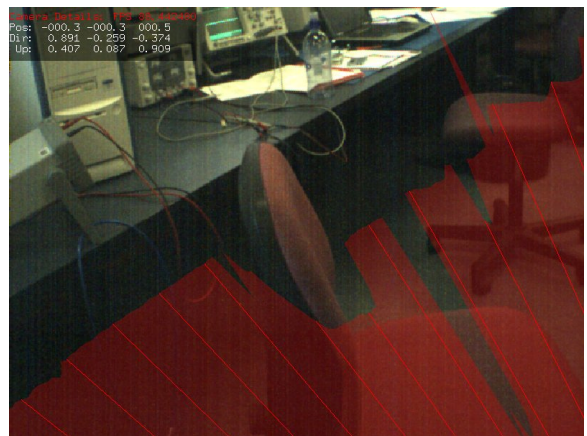


Figure 8: ARDev Software Architecture



(a) Laser data origin



(b) Laser data edge

Figure 9: Magnetically tracked HMD Output

display and the second is a stereo video see through head mounted display. The results of using these systems are shown in Figure 9 and Figure 10.

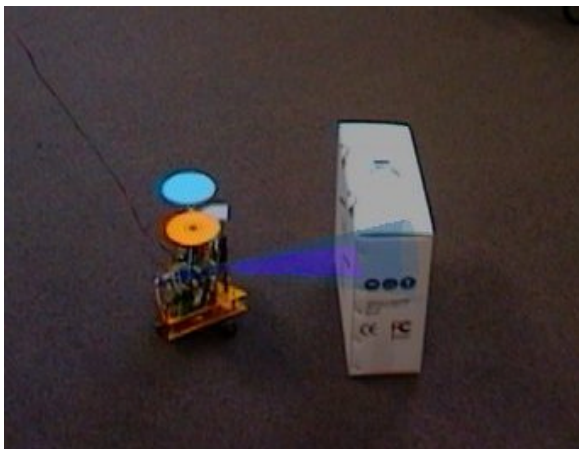
As can be seen in the screen shots the AR system provides an effective means of visualising the geometric data



(a) Sonar Sensors on Pioneer Robot



(b) Pioneer Odometry History



(c) Shuriken Robot with IR and Sonar

Figure 10: Overhead camera AR Output

sets, for example in Figure 10(c) 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.

## 6. DISCUSSION

Given the current system is still in an early prototype stage, it is not yet ready for qualitative user testing. Early informal results of people interacting with the system have been positive. The system has been shown to be good at picking up sensor hardware faults, for example when cables are blocking sensor scans.

The system also demonstrates the fundamental limitations of some of the sensors well which is advantageous to people who are new to robotics, as an example students looking at the system were able to see the effect of the angle of incidence for an ultrasonic sensor. The understanding of the limitations of the sensors is very important for understanding why a robot application is not functioning as expected.

While this paper has considered the applications of AR to developer-robot interaction, we see a wide range of applications for AR technology in HRI for many different user types. AR has the ability to bridge the gap in understanding between the robot and human user, allowing the user to understand the intentions and limitations of the robot in an effective and intuitive way. The toolkit we have developed has intentionally been built as a flexible system that could be applied to these broad areas.

## 7. CONCLUSIONS

Robot programmers are faced with the challenging problem of understanding the robot's view of its world, both when creating and when debugging robot software. Requirements for effective interaction under these conditions include the need to maximize 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 sets for the robot developer. The use of AR for the visualisations gives the developer an immediate understanding of the robot data in context with the real world baseline. This allows any limitations in the robot's world view to be understood, leading to much faster solutions to related software issues. Our initial implementation of these concepts has shown useful and promising results for robot software development.

## Acknowledgement

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

## 8. REFERENCES

- [1] 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)*, volume 1, pages 241–247, 2002.
- [2] H. Automatic. <http://www.hokuyo-aut.jp/>, August 2005.
- [3] 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)*, pages 3226–3231, 2004.

- [4] C. Breazeal, A. Edsinger, P. Fitzpatrick, and B. Scassellati. Active vision for sociable robots. *IEEE Trans. Syst., Man, Cybern. A*, 31(5):443–453, 2001.
- [5] 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.*, pages 186–9, 7–9 July 2003.
- [6] CARMEN. Carmen: Carnegie mellon robot navigation toolkit. <http://www.cs.cmu.edu/~carmen/>, June 2003.
- [7] 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*, pages 125–130, 2003.
- [8] DDD – Data Display Debugger. <http://www.gnu.org/software/ddd>, August 2005.
- [9] Eclipse. <http://www.eclipse.org>, August 2005.
- [10] 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)*, volume 4, pages 1842–1847, 2001.
- [11] LEGO Mindstorms. Lego mindstorms. <http://mindstorms.lego.com/>, January 2005.
- [12] 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*, pages 21–9, Tokyo, 5–7 July 1995.
- [13] 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)*, volume 3, pages 1467–1472, 1993.
- [14] 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*, pages 319–20, 7–10 Oct 2003.
- [15] Player/Stage. The player/stage project. <http://playerstage.sourceforge.net/>, January 2005.
- [16] V. Raghavan, J. Molineros, and R. Sharma. Interactive evaluation of assembly sequences using augmented reality. *Robotics and Automation, IEEE Transactions on*, 15(3):435–449, 1999.
- [17] 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)*, volume 2, pages 1946–1953, 2003.
- [18] F.-E. Trépanier and B. A. MacDonald. Graphical simulation and visualisation tool for a distributed robot programming environment. In *Proceedings of the Australasian Conference on Robotics and Automation*, CSIRO, Brisbane, Australia, December 1–3 2003.
- [19] J. Wang, M. Lewis, S. Hughes, and M. Koes. Validating usarsim for use in hri research. In *Human Factors and Ergonomics Society 49th Annual Meeting, Proceedings of the*, pages 457–461, 2005.
- [20] D. C. Yuen and B. A. MacDonald. An evaluation of sequential monte carlo technique for simultaneous localisation and map-building. In *Proc. IEEE Int. Conf. on Robotics and Automation (ICRA'03)*, pages 1564–9, Taipei, September 2003.