An Augmented Reality Debugging System for Mobile Robot Software Engineers

T. H. J. Collett B. A. MacDonald
Department of Electrical and Computer Engineering, The University of Auckland, New Zealand

Abstract—Robotics presents a unique set of challenges, which change the way that we must approach the debugging of robotic software. Augmented reality (AR) provides many opportunities for enhancing debugging, allowing the developer to see the real world as the robot does, superimposed *in situ* on the real world view of the human, intuitively displaying the limitations and discontinuities in the robot's real world view. This paper contributes a systematic analysis of the challenges faced by robotic software engineers, and identifies recurring concepts for AR based visualisation of robotic data. This in turn leads to a conceptual design for an AR enhanced intelligent debugging space. Both an open source reference implementation of the conceptual design and an initial evaluation of the implementation's efficacy are described. The AR system provides an opportunity to understand the types of errors that are encountered during debugging. The AR system analysis and design provide a reusable conceptual framework for future designers of robotic debugging systems, and guidelines for designing visualisations. In concert with common, standard robotics interfaces provided by Player/Stage, the AR system design supplies a set of common visualisations, so that many data visualisations can be provided to developers with little additional effort.

Index Terms—Augmented reality, visualisation, software development.

1 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 present software engineers with unique challenges for which traditional development tools are not designed. The challenges arise from the *robot environment*, the *inherent nature of mobile robots*, and the *nature of mobile robot tasks*.

Mobile robot environments are uncontrolled real world environments where dynamic and real time behaviour are dominant. Unexpected variations and conditions occur leading to behaviour that is rarely repeatable.

Robots are mobile, so the "programming target" autonomously changes position, orientation and sensory relationship with its environment and the programmer; the programmer does not directly control these relationships. Mobile robots contain a large number of devices for input, output and storage, which have little correspondence with human programmers' familiar senses and effectors. In contrast with the limited devices in desktop PCs mobile robot systems include wide variations in hardware and interfaces.

Mobile robot tasks emphasize geometry and 3D space. Complex data types must be represented, such as high dimensional spaces and paths. Robot tasks are potentially uninterruptible, and there is simultaneous and unrelated activity on many inputs and outputs.

These challenges occur in other applications, such as complex embedded systems, but mainstream development tools do not address this complexity well. For example concurrency is often present in robotic and non–robotic applications, but it is not handled well in many debugging environments.

In the past researchers have reacted to these challenges by developing different robotic software engineering tools in each laboratory, sometimes even different tools for different robots. There have been attempts at standardisation; some tools, such as Player/Stage [1] have emerged as *de facto* standards since they support a large variety of hardware for a growing user community. We believe that standardisation in this form is desirable. Better tools emerge when common concepts are identified, enabling engineers to provide more reusable conceptual designs with reference implementations.

We hypothesise that debugging is a primary programming

Regular paper – *Manuscript received June 1st, 2009; revised September 1st and December 17th, 2009; online since January 20th, 2010.*

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

process that is impeded by the lack of suitable tools for mobile robot programming. As a solution we propose and evaluate a debugging tool based on Augmented Reality (AR). AR does not use the simulated world often used by programmers of Player or Microsoft Robotics Studio (MSRS) [2]. Instead it uses the real view of the robot's world, and augments it with additional, virtual, objects that express the robot's view of the world.

A common thread of the challenges we have mentioned is that the *robot's interaction with the environment* makes robot programming, and in particular debugging, different and challenging. As a result of the complexity of this interaction, the programmer is challenged to understand what data the robot is receiving and how the robot is interpreting that data, in other words it is the *programmer's lack of understanding of the robot's world view* that makes robotic software difficult to debug.

A simulation is not always accurate in expressing interaction with the real world. AR has the potential to provide an ideal presentation of the robot's world view; it can display robot data in a view of the real environment. By viewing the data in context with the real world the developer is able to compare the robot's world view with the ground truth of the real world image. The developer need not know the exact state of the world at all times, because the comparison is directly observable. This makes clear not only the robot's world view but also the discrepancies between the robot view and reality.

This paper:

- Expresses key design questions for future developers of robotic AR debugging systems,
- Identifies recurring concepts for AR visualisation of robotic data, and
- Presents a conceptual design for AR debugging systems which has a general purpose structure that may be applied to other robotic software engineering libraries.

The AR debugging system allows the developer to view a robot's data and meta data in context with the real world operating environment, providing a natural qualitative comparison to ground truth. A reference implementation is presented as a plugin for Player, and also is capable of deployment in other systems.

The work reuses Player's definition of robot data types, providing a standard interface to robotic data, along with default visualisations of a full set of robotic data types. We expect these data models to be easily transferable to other mobile robotic programming systems since they capture the natural data types required in any mobile robotic system (and many other robotic systems).

Future work should explore visualisation of other aspects of robotic software. AR visualisation also fills an important role as a bridge between simulation and real world testing.

Section 2 discusses related work. Section 3 examines the requirements of an AR system for robot developers. Section 4 presents our reusable, conceptual design for an *intelligent debugging space* (IDS). Section 5 summarizes a reference implementation of the IDS. Section 6 discusses our evaluation and user study. Section 7 discusses the IDS and results.

2 Related Work

There is a range of potential programming tools for robot developers. Most of these tools are focused on providing the user with a view of the current robot data. For example, most of the robot frameworks such as ORCA [3], CARMEN [4], ROS [5] and RT middleware [6] provide their own data viewers. MSRS [2] provides a tool for robotic developers, including an integrated development environment and visualisations of robot data in a simulated view. None of these frameworks include any systematic process for designing their debugging tools, or any analysis of their performance. Kooijmans *et al* [7] present a multimodal interaction debugger, for assisting robot developers in human–robot interaction (HRI). Kramer and Scheutz survey robot programming environments [8].

Player is a network oriented device server that provides developers with hardware abstraction and distributed access for a wide range of robot hardware [9], [10], [1]. 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.

Player enables our visualisations to connect in parallel with the target client application. Clients need not be modified when we use the visualisations, and additionally the visualisation system can be run even when clients are not active.

Existing visualisation tools present data in isolation from the environment, for example Player's [1] built in playerv tool visualises laser scan data, as shown in Fig. 1. Isolated visualisations are easy to implement but have key limitations; it can be difficult to understand the relationship between the isolated dataset and the real world values, qualitative errors such as inverted axes or offsets in the data are easily missed and errors identified in the data can be difficult to match to an environmental cause.

A slightly more advanced approach is to add a static model of the world to the visualisation, for example a map of the building, to provide an environmental reference. This approach works well for some types of errors but it is generally not feasible to model every object in the environment, particularly where there are dynamic objects, such as human users. This approach can also have a confounding effect if the static map is in error. If the map errors cause the developer to make erroneous assumptions, the resulting programming mistakes may not be visible in the visualisation. Additionally, if the same flawed assumptions are made for both the static model and the application that uses the model as a reference for visualisation, the flaws may be disguised. AR has the potential

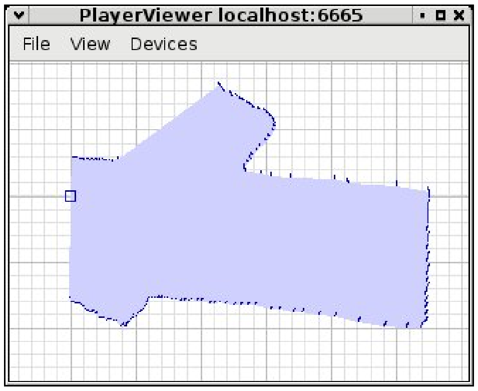


Fig. 1. The playerv laser visualisation

to avoid these difficulties since it can show real robot data in context with the real world. Systems that use a pre built map include the GSV tool [11], the AMCL debug window from player [1], Playerv3D from the eXperimental robotics framework [12] and CARMEN's navigation GUI [4].

AR has also been used for a range of end user interfaces to robotic systems. Several of these systems use AR purely to help present a complex set of data more intuitively to the user. Freund *et al* [13] use AR to more simply display complex 3D state information about autonomous systems. Daily *et al* [14] use AR to present information from a swarm robotics network for search and rescue. KUKA has begun to investigate the use of AR to visualise data during training [15]. Amstutz and Fagg [16] 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.

Other systems use AR to provide feedback to the user of the operations the robot is undertaking or going to undertake. Milgram *et al* used AR in creating a virtual measuring tape and a virtual tether to assist inserting a peg in a hole [17], [18]. Raghavan *et al* [19] describe an interactive tool for augmenting the real scene during mechanical assembly. Pettersen *et al* [20] 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* [21] describe an AR system that integrates graphical and sensory information for remotely controlling a vehicle.

While the use of AR is informative for this work, both as a intuitive display of data for users in a monitoring situation, and for feedback in a control situation, none of the above systems consider the use of AR as a debugging tool for systems under development. More recent work by Stilman *et al* [22], Chestnutt *et al* [23] and Kobayashi *et al* [24] used an advanced motion capture system to provide debugging for the development of humanoid robotic applications with the HRP2. This work with HRP2 lacks a systematic analysis of the needs for a development system and the recurring concepts for software design of AR systems, and is highly tailored to the specific development environment. It provides evidence of the need for more advanced development tools, while also reinforcing the authors' belief in the importance of designing a reusable debugging tool for the general problem of mobile robot development.

3 A Conceptual Augmented Reality (AR) System for Robot Developers

Debugging is essentially a process of elimination and by understanding the type of error occurring developers can quickly eliminate large sets of potential bug sources, substantially reducing debugging time. If a developer makes an incorrect assumption about the source of the bug, then a disproportionate effort is needed before all the potential assumed sources of the bug can be eliminated, and the developer can finally realise the initial assumption is incorrect.

A number of questions must be considered when capturing recurring concepts that may be useful in designing an AR visualisations for developers across different frameworks. What, how and where should data be displayed? How should data be represented?

3.1 What data should be displayed?

The most important program information to display is about the variables that are used for decisions or as control inputs. The data in these variables is often very compact as any filtering and fusion between various inputs has already been carried out before control decisions are made. Visualisations of these data are also more concise. Once a failure is detected an understanding of the decision data lets the developer determine the nature of the error; bad input data, bad decision logic or a platform failure.

Robot development is, or should be, undertaken in two stages, library development and application development, and different visualisations are required in each case. Some applications use existing libraries (so the first stage has been provided by other developers), and unfortunately too often large applications are built without separating out modular libraries for reuse.

The library developer must focus on the low level details of the library function that is under development. The details of the internal working of one or more code fragments need to be displayed, while details for 'out of focus' components should be limited. The focus may change rapidly so the visualisation must be flexible enough to change the amount of detail displayed about each component. In particular it must be straightforward both to display varying levels of detail about connected components and also to change focus to a related or connected component.

The application programmer need only see the interface to the library components; the main focus is on the connections between library components and the high level program logic.

The developer focuses on the standard interfaces of components and does not often need to see the visualisations of the internal component elements.

Robot data, especially at the component interface level, can be characterised in a number of different ways. It is important to focus on distinctions that alter the way the data is viewed, identifying common, underlying data types in order to promote generalisation and therefore reuse of data visualisation techniques.

Player represents a significant user community in robotics research with a significant breadth of devices and components, and so is a good initial source of established common data types. Table 1 contains a list of the data types used in Player version 2.1 (extracted from an inspection of the source code). Each has three properties, a data type, whether the data is spatial and whether it is inherently a sequence. The data type is (S)calar, (V)ector, (G)eometric or (A)bstract.

Geometric types and scalar measurements from transducers have an intrinsic visual representation. All geometric types are able to be rendered directly into the environment. Spatial scalar and vector data can be rendered relative to a known reference frame. For acceleration and velocity this is the robot frame of reference. Scalar quantities such as a measured range can be rendered relative to the transducer origin and along the axis of measurement, for example a range measurement from an ultrasonic sensor.

Non spatial data must be modelled spatially before it can be rendered. Sometimes this is straightforward, for example using coloured bars to indicate battery level. More abstract data such as bit fields or binary data may need to be rendered at a higher level of abstraction. For example the bumpers on a robot may be represented as a bit field in code but are more intuitively understood if they are rendered as a 3D model with the active bits in the field defining some property of the bumper object such as the model's colour. Another alternative for non spatial data is to use a textual representation.

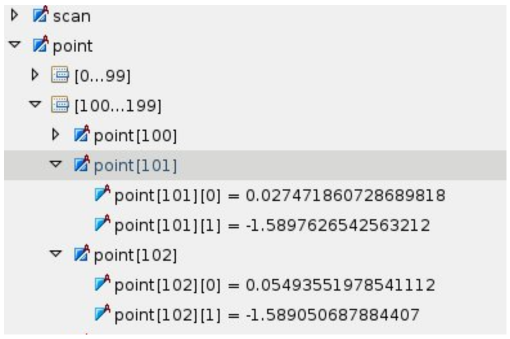
The rendering of spatial and temporal sequences offers some potential for simplification of visualisation, for example rendering the outline of a sequence of range readings from a laser scanner. However in many cases outlines are not useful; for example, an outline may not be a useful representation of a temporal sequence of range values.

In general stochastic properties such as uncertainty or probability add extra dimensions to the data. In some cases, such as a point location, uncertainty can be easily rendered by expanding the object to an extra dimension and representing it as an ellipse or polygon. Where objects are already three dimensional these data must be represented using another property such as colour or transparency.

To summarise, data can be either *spatial* or *abstract* and can be *sequential* or *stochastic*.

Name	**Type**	**Spatial**	**Sequence**
Position	G	★	
Orientation	V	★	
Pose	V	★	
Axis of Motion	V	★	
Length	S	★	
Area	S	★	
Centroid	G	★	
Bounding Box	G	★	
Range	S	★	
Radius of Curvature	S		
Field of View	S		
Point	G	★	
Polygon	G	★	
Line	G	★	
Coordinate System Transforms	A		
Approach Vector	V	★	
Latitude/Longitude	G	★	
Altitude	S	★	
Grid Map	S	★	★
Vector Map	G	★	★
Waypoints	G	★	★
Point Cloud	G	★	★
Range Array	S	★	★
Pan/Tilt	S		
Velocity/Speed	V	★	
Acceleration	V	★	
Current	S		
Voltage	S		
Waveform	S		★
Duration	S		
Amplitude	S		
Frequency	S		
Period	S		
Duty Cycle	S		
Charge (Joules)	S		
Power (Watts)	S		
Uncertainty in Measurement	S		
Time	S		
Magnetic Field Orientation	V		
Temperature	S		
Light Level	S		
State	A		
Tone Sequence	S		★
Encoded Waveform	A		★
Volume	S		
Colour	A		
Identifier	A		
Brightness	S		
Contrast	S		
Bit Fields	A		★
Capacity	S		
Percentage Measures	A		
Memory	S		
Button Data	A		
Intensity	S		
Resolution	S		
Resource Locator	A		
Text	A		★
Network Information	A		
Binary Data	A		★
Image Bit Depth	S		
Image Width/Height	S		
Image Data	S		★
Covariance	S		
Weighting	S		
Variance	S		
Metadata			
Type	A		
Capabilities	A		
Default Values	A		
Indices	A		
Arbitrary Properties	A		

TABLE 1
Data types used in Player interfaces (from an examination of the source code for Player version 2.1). Each has three properties, a data type, whether the data is spatial and whether it is inherently a sequence. The data type is (S)calar, (V)ector, (G)eometric or (A)bstract.



```
▷ scan
▽ point
  ▷ [0...99]
  ▽ [100...199]
    ▷ point[100]
    ▽ point[101]
        point[101][0] = 0.027471860728689818
        point[101][1] = -1.5897626542563212
    ▽ point[102]
        point[102][0] = 0.05493551978541112
        point[102][1] = -1.589050687884407
```

Fig. 2. A section of the Eclipse IDE debug window showing laser data

3.2 How should the data be represented?

The visualisation of robot programs can be seen as a combination of standardised visualisations that are included as part of the visualisation tool, or as plug-in modules, and custom visualisations created by the developer.

Unfortunately standardised visualisations cannot be provided to developers for all possible data sets. Applications will have custom data structures and interfaces and may have unique combinations of interfaces that can be effectively visualised as one. Standardised visualisations are similar to the standard views available in modern graphical debuggers such as DDD [25] and Eclipse [26]. They are based on the semantics of the standard data types provided in many languages and libraries (such as lists). Fig. 2 shows a screenshot of the Eclipse debugger [26], presenting the structure of the player laser proxy data. Fig. 3 shows a screenshot of the DDD debugger's graphical visualisation for laser data [25]. Most current Integrated Development Environments (IDEs) take the approach 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 Fig. 2 and 3.

Custom visualisations are like the manual debugging output created when a programmer instruments code with log statements; the developer can display concisely summarised information about execution in a particular piece of code, based on the semantic interpretation they are giving the data. They know which items of data are most salient. Custom visualisations are most powerful if supporting classes and methods, or other language constructs, are available to aid the programmer in creating the renderings, including building blocks for rendering.

In order for standardised visualisations to be useful the program being visualised must have a known structure and/or known data types. 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. This requirement for standard interfaces is important in software engineering and so is not difficult to meet when projects such as Player provide potential interface standards, including interfaces to devices with standardized device interfaces.

The internal data of the program is more difficult to visualise as it does not generally conform to a predefined structure. However at a finer level of granularity individual components of the internal data, such as geometric datatypes, can be viewed by standard visualisations. However this method may not scale up to complex systems. The best visualisation for complex internal data may not be the combined visualisations of all its components. 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 that determine what is visualised will depend on the application. Perhaps the single most likely match is all that is needed, or the top 10 matches, or all the possible matches above a certain size, or there may be other criteria.

3.3 Where should the data be displayed?

Where the data is displayed in the developer's view has an important influence on the how easy the data is to understand. Particularly when multiple data sets, potentially from multiple robots, are displayed together. The best location for rendering data depends on a combination of the type and source of data that is to be rendered, the needs of the user and the control interface that is available.

It is proposed that in general if a geometric rendering is available then data should be placed at its matching real world location. The user should be given some control over this process with the ability to alter the rendering location in order to provide a clearer view of the full scene. For example, a range measurement may be represented as a line segment from the source to the measured point.

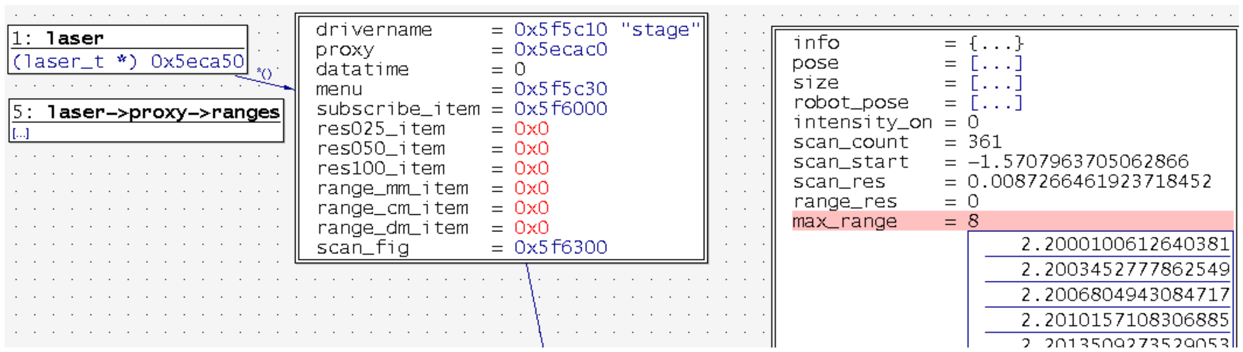
Data without a direct spatial metaphor should be rendered at a location that is related to the robot while taking into consideration the level of clutter in that screen area, again the user should have the ability to change this selection. For example, the battery level may be shown on a part of the robot.

The location of data rendering should be handled much like the position of windows in modern window managers. The initial location is negotiated between the application and the window manager, and this can later be modified by user interactions such as dragging or hiding.

3.4 How should the data be viewed?

There are a number of hardware possibilities:

- The data may be viewed by an augmented, head mounted video display, by an optical see through head mounted display, by a projector or by a large display screen in the robot debugging space;
- The display may be a single view, or a stereo view that provides 3D information, and there may be multiple camera views;



```
1: laser
(laser_t *) 0x5eca50

5: laser->proxy->ranges
[...]
```

```
drivername        = 0x5f5c10 "stage"
proxy             = 0x5ecac0
datatime          = 0
menu              = 0x5f5c30
subscribe_item    = 0x5f6000
res025_item       = 0x0
res050_item       = 0x0
res100_item       = 0x0
range_mm_item     = 0x0
range_cm_item     = 0x0
range_dm_item     = 0x0
scan_fig          = 0x5f6300
```

```
info          = {...}
pose          = [...]
size          = [...]
robot_pose    = [...]
intensity_on  = 0
scan_count    = 361
scan_start    = -1.5707963705062866
scan_res      = 0.0087266461923718452
range_res     = 0
max_range     = 8
                  2.2000100612640381
                  2.2003452777862549
                  2.2006804943084717
                  2.2010157108306885
```

Fig. 3. A section of the DDD visualisation of a list structure for laser data

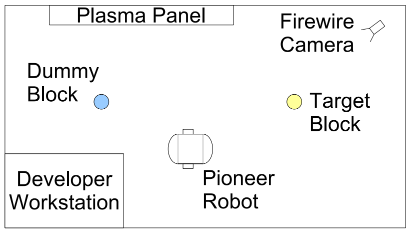


Fig. 4. Intelligent Debugging Space (set up for the Block Finder Task). The space is 5m $\times$ 3m.

- The developer's view may correspond to a static third person view, or a moving first person camera.

These methods have not been evaluated for robot developers, and it is not obvious which will be the most useful. Relevant factors include: the nature of the robot application being developed, whether the installation is permanent or temporary, whether more than one developer is involved, and the cost of the installation. Therefore AR software must be flexible enough to support a wide range of AR configurations.

4 Intelligent Debugging Space (IDS)

A reference implementation of the design concepts has been created as our IDS. It is a permanent installation of a fixed viewpoint video see-through configuration that provides an augmented view of the robot development space, shown in Fig. 4. One end of our laboratory is monitored by a Prosilica High Resolution (1360x1024) firewire camera which looks down on the robot space from the corner of the ceiling. A large 50" Samsung plasma screen displays the camera image (a third person view), augmented by virtual objects that are visualisations of robot data added to the image. The developer works at the station in one corner of the space, and turns to the large screen as needed, to see the AR view.

The IDS is designed for automatic visualisation of the standard data components of the robot application. To achieve this it must be able to track the presence of robots in the development space and determine their capabilities. Once their capabilities are determined a suitable set of visualisations may be displayed for the user.

Both the physical and operational presence of a robot are important. Physical presence is about whether the robot (or its data) are visible in the AR field of view and whether the robot can be tracked. The operational component determines if data can be extracted from the robot, for example a Player robot is operational if the Player server is reachable on the network.

Summarising the details of our IDS (given in [27], [28], [29], [30]):

- The physical presence of multiple robots is tracked using fiducials from ARToolKitPlus mounted on each robot [31]. Fiducial tracking is often used in AR applications [32];
- Player enables:
  - tracking the operational presence of multiple robots;
  - connecting directly to the robot without an intervening user application and providing a set of standard visualisations;
- A configuration manager and static configuration file enable:
  - determining the functionality and configuration of the tracked robots;
  - coarse grained control over what visualisations are present and some basic configuration of the presentation of the data sets (such as colours);
- The AR view is displayed on the plasma screen by the ARDev library which uses the robot tracking to geometrically locate the visualisations in the camera view and add them to the camera image.

To enable the developer to enhance the AR view with additional custom visualisations the IDS also provides a whiteboard style interface where the developer can add basic 2D and 3D elements. This provides high levels of flexibility to the developer but also requires significant developer effort in specifying the visualisation.

The custom renderings are used to augment the standard set, adding or highlighting key features from the developer's program that are relevant to the behaviour being debugged, in particular key decision or control variables can be displayed.

5 Software Implementation

Our ARDev library has been implemented as a highly modular general purpose AR system (detailed in [27], [28], [29], [30] and downloadable as an open source package from Sourceforge[1]). ARDev can operate as a plugin for Player. These modules represent the recurring concept of an output object, an abstract software object with capture, camera, preprocessing and rendering, which makes them highly reusable. Each output object contains a capture object for grabbing the real world frame and a camera object for returning the intrinsic and extrinsic camera parameters. The capture object could also be null for optical see through AR, or for a purely virtual environment (since no capturing is required). The output object maintains three component lists: postprocessing objects, which are provided with the AR frame after rendering; 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. Fig. 5 shows the software structure.

The output object provides the core of the toolkit, including the main thread for implementing the rendering loop, which consists of 8 stages:

1) Capture: the background image, orientation and position of the camera;
2) Pre–processing: e.g. 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 stereo optical see through 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.

Many of the requirements of the IDS are simplified by using Player to encapsulate the robot platform. Player provides a set of standard interfaces and allows the IDS to access robot data independently from other robot applications. By implementing a library of common visualisations for Player interfaces we are able to present a useful set of visualisations for any developer, vastly reducing the requirement for custom rendering; some examples are shown in Fig. 6. The implemented interfaces contain Player data types from Table 1 including: Position, Orientation, Pose, Axis of Motion, Centroid, Range Array, Field of View, Vector Map, Grid Map, Pan/Tilt, Uncertainty in Measurement, Bit Fields (Bumper) and Covariance. ARDev also provides a whiteboard plugin architecture for custom visualisations.

Also provided is a graphical configuration manager.

1. http://ardev.sourceforge.net/

6 Evaluation

Metrics for evaluation of HRI systems are still an open issue. The difficulties are magnified by large variations in the performance of different programmers [33]. A large user base of robotic software engineers would be needed to perform valid quantitative comparisons. To design a suitable evaluation, we consider techniques for usability and software visualisation evaluation as well as HRI metrics.

Nielsen presents issues of usability [34] and Ziegler [35] gives a concise summary of the techniques available. The usability assessment techniques used in the current work are observation, questionnaires and interviews. The interactive technique is AR, and has the potential to include multi-modal interation by voice and gestures.

Hundhausen [36] presents techniques for studying the effectiveness of software visualisations. Those relevant to the current work include observational studies leading to more controlled experiments, using questionnaires and surveys (as primary or secondary evidence), ethnographic field techniques, and usability tests.

Steinfeld *et al* provide an analysis of the metrics that are suitable for HRI evaluation [37]. The most relevant metrics for the current work are the system performance metrics. Key perfomance indicators for a debugging system include quantitative measures such as effectiveness and efficiency as well as more subjective measures such as situational awareness, workload and the accuracy of mental models.

Where the current work differs from the systems considered by the above authors is in the goal of the interaction. Where most evaluations focus on the robot's mission, the goal of a debugging system is to identify an issue in the system under test itself. For a developer debugging their own software there are issues about devising realistic debugging scenarios to challenge the assumptions made in software design, even though the developer may be unaware of some tacit assumptions. For a developer debugging others' software, there is a need to devise debugging scenarios that explore and discover the implicit assumptions made by the original developer.

In the software engineering domain Ellis *et al* [38] use a set of simple performance metric based user trials to evaluate

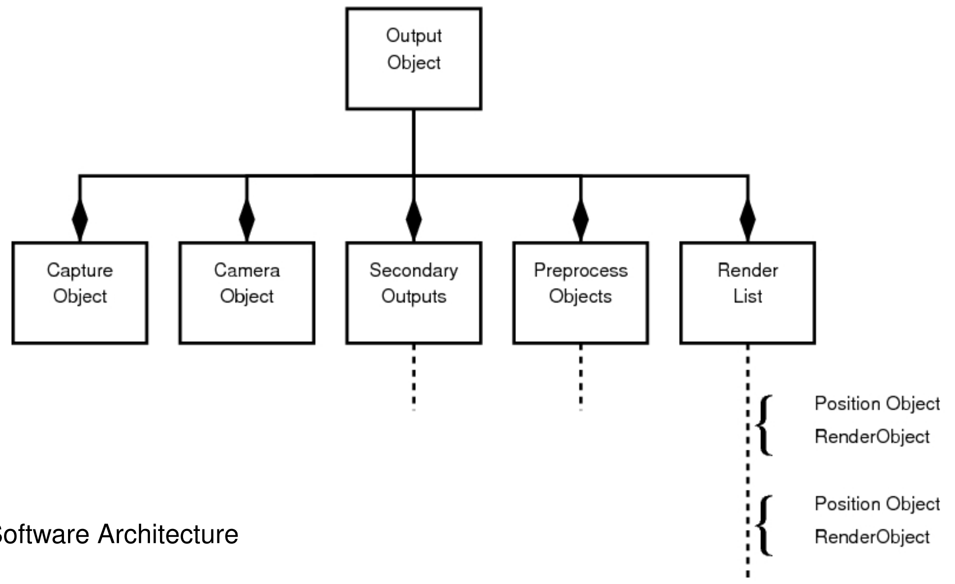
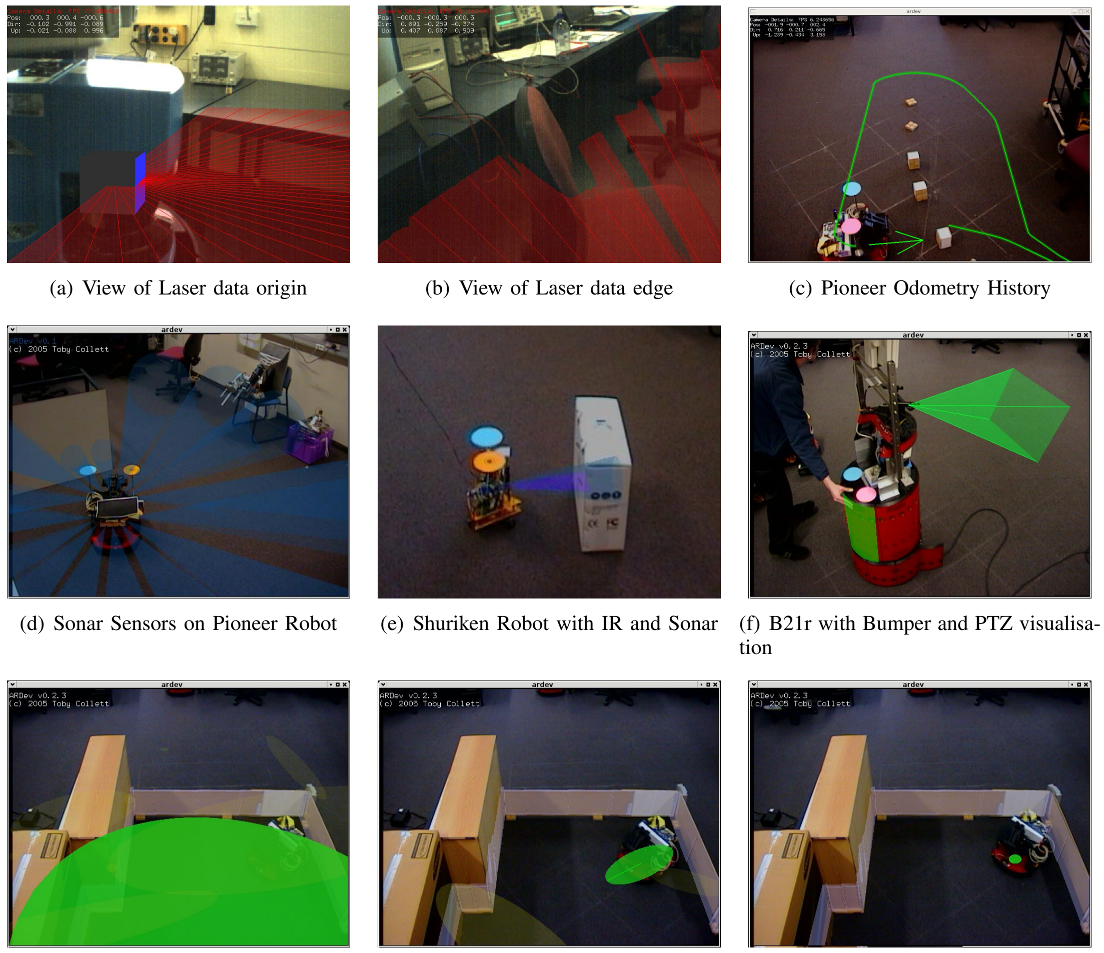


Fig. 5. ARDev Software Architecture



(a) View of Laser data origin

(b) 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 visualisations for laser data, our own Shuriken robot, a Pioneer, and a B21r, including Player's AMCL localisation.

the effectiveness of a specific design pattern. A similar user study could be used to produce quantitative results for the effectiveness of the system described in this paper. However for novel interfaces there are two potential limitations to this approach, the first is that in order to identify specific enough cases to perform a user trial one must carry out some form of initial observational trial. Secondly by focusing too narrowly in a quantitative trial important aspects of a novel interface could be missed.

Therefore our evaluation of the developed system is proposed in three stages.

1) The first is an informal participant study, which we have conducted and is presented below.
2) The intention is to follow this study with a longer observational, ethnographic study as part of a real world robot development team. We expect this to involve something on order of a month long study of the implementation of a larger system, ideally with a robotics company. With these two broader studies we expect to expose: a) specific potential benefits of the AR system for developers, and b) a small set of metrics appropriate for a quantitative study.
3) The results of these two trials can then be used to design a narrower set of targeted user trials with a larger number of participants that can be used to produce a quantitative comparison of ARDev with other potential debugging systems. By narrowing the scope of the evaluation to the most critical issues that arise, and using a large number of participants, we expect to obtain significant quantitative results. At present we expect that this final trial will examine three conditions: debugging without visualisations, debugging with visualisation aids, and debugging with our AR visualisation method. Metrics will be drawn from published metrics listed above, for evaluating software visualisations, software developers' experiences, and HRI.

A full version of the initial evaluation results is available in [30]. The initial study has focused our attention for the ethnographic study, on exploring the five benefits extracted in Section 7.1:

1) Standard visualisations are useful
2) Data validity and prevention of false conclusions
3) Same visualisations for simulated and real robots and environments
4) Immediate confirmation of hardware performance limitations
5) Monitoring role

as well as exploration of how custom visualisations can be made more useful.

6.1 Methodology

The purpose of the initial studies was to examine the use of ARDev in case studies that are standard robot development tasks, in particular to explore whether the AR system was able to aid the developer's understanding of the robot's world view and direct the developer towards the source of bugs in the robot application under test.

We divided the study in to two parts. Initially the ARDev developer (THJC), carried out an exploratory pilot trial of three case studies in order to correct any bugs found in the tool, and to verify the study design. Following this verification, the first two case studies were presented to five independent participants, who were selected from the small pool of robot programmers that were available. All had a small amount of experience using the Player robotics framework, and ranged in general programming experience from university level programming only to 3 years of programming experience in a commercial environment. While all of the participants were aware of the AR system before the trial none had previously used it for robot programming tasks.

The first part is a pilot version of a participant–observer ethnographic study, while the second is like a pure observer ethnographic study. By exploring both we have further insight in to how best to design the next, larger, ethnographic study as discussed above.

6.2 Case Study Design

Three tasks were chosen to span a range of the standard tasks and styles of program that a robot developer is likely to use: follower, block finder and block picker, which include sensing, object detection, navigation and manipulation.

6.3 Test Setup

All three tasks were developed with the popular Pioneer 3DX from MobileRobots [39]. Each robot is equipped with:

- Via EPIA-TC 600MHz ITX computer, running Player on Ubuntu Linux, with wireless networking;
- 5DOF Arm, plus gripper, from MobileRobots;
- URG Laser Scanner from Hokuyo [40];
- Logitech 640x480 webcam.

The developer workstation was a Pentium D computer with dual 19" LCD displays.

6.4 Case Study Tasks

6.4.1 *Follower*

The follower task represents a very simple reactive control loop. The robot uses the laser scanner to detect and approach the closest obstacle, stopping a safe distance from the object.

6.4.2 *Block finder*

The block finder is representative of search tasks, and is a task primarily concerned with a 2D representation of the world. In this task the robot searches through the environment for a specific object. The laser was used to identify targets.

This task can be implemented as a state machine with a simple loop over a switch statement in C/C++. To better imitate the normal robot development cycle the application was initially simulated using Stage. Two cylinders of differing diameters were used as the targets. In the pilot study THJC also explored a variation of this task using the camera to identify objects.

6.4.3 Pick Up the Block

This additional task involves manipulation of objects using the 5 DOF arm on the robot. While the design of the task is a simple linear sequence of actions, the 3D nature and the non-intuitive geometry of the arm have the potential to create difficulties for the developer.

A block is manually placed within reach of the arm in front of the robot. The laser scanner identifies the location of the block. The arm then picks up the block under open loop control and drops it at a predefined location on the robot.

6.5 Initial Pilot Results

The pilot results have limitations since THJC is the ARDev developer. However, there are some interesting errors made by the developer that illustrate ARDev and some reflections that are worth reporting.

6.5.1 Follower

For this task the standard Player laser visualisation was displayed along with a custom visualisation presenting the calculated direction of the closest object as an overlaid white beam. During development, the robot initially turned on the spot continually. An examination of the AR output immediately showed that the robot was miscalculating the orientation of the closest object. This fault was tracked down to a bug in the Player client library where the minimum and maximum angles of the laser scan were inverted. The developer was able to see that the values were correct in magnitude but it was more difficult to identify in passing that they were in the wrong orientation or quadrant.

6.5.2 Block Finder

The identification of potential targets in the laser scan was achieved through segmentation of the laser scan at points of discontinuity. The *graphics2d* interface was used by the developer to visualise the potential targets in simulation, this is shown in Figure 7. The alternating red and green lines show the segments and the cyan diamonds represent the discontinuities that are possible targets.

The visualisation in the simulator highlighted a key limitation of the Stage laser model. The laser model ray traces a subset of the scans in the environment and then interpolates these to get the full requested resolution. This has the effect of dividing any large discontinuities in the laser scan into a sequence of smaller discontinuities producing a number of false targets as shown in Figure 7. This is difficult to correct for, but there are two temporary solutions, changing the interpolation model or increasing the actual scan resolution. The first approach was used to enable the successful completion of the simulation. The visualisation created with the *graphics2d* interface was very effective in identifying this limitation of the Stage model and hence reducing the time needed to fix the issue.

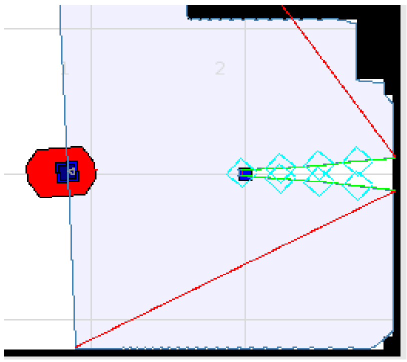


Fig. 7. Errors in the simulated block finding visualisation.

Once the application was functioning in simulation it was tested on the real Pioneer. During testing the standard laser visualisation was used alongside the custom graphics2d visualisation that was developed during simulation.

The subject's reflections include:

- The AR system loses tracking when the robot arm obscures the fiducial on the back of the robot;
- On one occasion the developer noted that the robot appeared deadlocked while orienting its direction to the marker. No obvious reason was apparent, it was correctly pointing in the direction of the marker, possibly it had a very small turn rate and was waiting forever until it reached an angle close enough to the target direction. The error was likely to not be repeatable so the developer increased the gain of the turn rate controller to avoid further such incidents;
- The robot failed to identify the block correctly, possibly it was misaligned with the block. The developer modified the robot's behaviour to continue to align itself with the block as it approached, as opposed to the straight line approach used initially;
- The previous change to the approach mode failed, the developer needed to reduce the gain for turning during approach.

Three subsequent trials completed successfully. In general the results of this task are positive but are limited in scope because of the success of the simulator in catching bugs initially. There are still some important points to make.

The visualisations used for this trial were either standard visualisations which were part of the toolkit, or unmodified custom visualisations used with the simulator. So there was no overhead for the developer in using the AR system.

Visualisations helped to rule out possible causes of bugs.

One limitation was the tracking of the robot's position by a single camera. It was easy for the AR system to loose track of the marker when either it was obscured by the Pioneer arm, or when the robot was too far away for the marker to be decoded. This could be solved a number of ways, for example using alternative trackers or multiple cameras.

During the trial, the AR display exposed some bugs in ARdev system, which were also corrected.

6.5.3 Pick Up the Block

AR visualisations helped develop software for the task, however the arm was not accurate enough to carry out the task reliably.

Prior to undertaking the trial the standard visualisations for the arm were written. This consisted of a visualisation for the Player end effector *(limb)* and joint control *(actarray)* interfaces. While writing the actarray visualisation the developer found that the geometry being reported by the links in the array was in some cases incorrect. The axis of rotation was reported with the wrong sign in about half the cases. This is another case where qualitatively the data appeared correct, but had the wrong sign.

Another issue found during the creation of the visualisation was that the developer was incorrectly assuming the limb coordinates were in the arm coordinate space when in fact they were being reported in robot coordinate space. This was immediately obvious from the AR visualisation, but would have required manual measurement to determine from the raw data. The Player documentation was amended to be more explicit about the coordinate system origin.

6.6 Participant Case Study Results

The participants all undertook the follower and block finder tasks as described above. While no formal performance metrics were recorded all the participants completed the tasks in two to three hours, although a number of interruptions and some difficulties with the wireless connection on the robots made more accurate timing information meaningless.

The participants were instructed to take notes of bugs and critical events they found while carrying out these tasks in a similar fashion to the initial case study. All participants needed to be reminded regularly to continue taking these notes, and so it is suspected that many issues will not have been commented on. Additionally a short interview was held with each participant at the conclusions of their tasks.

The participants were initially given some template code that set up the connections to the robots and gave an example of rendering a cross at a fixed location in front of the robot. They were also given a make file that would build their example tasks. This support structure allowed the participants to focus purely on implementing the designated tasks rather than spending time on setting up the build environment.

6.6.1 General Participant Notes

This section presents a summary of the notes from the tasks and interview for the participants, collected together based on common themes in the results.

Usefulness, understandability and interpretation of data

Positive comments:

- Graphical rendering allowed faster comprehension of the AR data in realtime (rather than examining execution logs at the completion of a test run).
- AR renderings would be useful when transitioning from a simulation environment.
- AR helps the user to understand the base robot platform and the abilities of the laser scanner on different surfaces.
- AR helped confirm that objects were detected by sensors.
- Stopping the robot made debugging of sensor processing code simpler (this requires further study).
- The graphical of data is an improvement on reading text numbers from a stream of data on screen, which is error prone.
- The standard renderings provided by the IDS were valuable, and if no custom visualisations were used, the overhead of using ARDev was close to zero.
- The AR system allows users to interact while viewing the graphical output which allowed for more of the system to be tested with a stationary robot.

Lack of quantitative information:

- Several participants commented that AR was unable to tell them quantitative information such as the scale and size of objects, orientation of axes, direction of rotation.
- A protractor and ruler tool in the AR environment would be useful

Coordinate frames

- Most participants had a small amount of confusion with about the different coordinate frames in use, for example the laser scanner coordinate frame relationship to the robot frame. To use the available frames correctly, a 3D approach is needed, because of a vertical offset.
- Participants tended to use the 2D graphics for rendering data, rather than 3D which was available; this may change when the current 3D simulator in Player becomes more popular.

Custom *versus* default visualisations

Positive comments:

- One participant had difficulty with the second task until they remembered to use the custom rendering functions, which immediately highlighted some errors.

Negative comments:

- Some comments reflected the extra effort needed to create custom visualisations, and questioned whether the effort was worth it. In particular the graphics interfaces require subscription to the IDS, setting of colours, creating a list of points and finally rendering the points. This could be

simplified by extending the interface to support more operations such as drawing symbols, eg crosses and arrows.
- The lack of text support was considered a problem (that could be fixed reasonably easily).
- Time is needed to debug the visualisation code itself, in a similar way to getting debug output wrong in a print statement, and newly written visualisation code was initially not trusted for accuracy.
- Different robots had different base colours, which is confusing if the test robot is changed.

The results about custom visualisations support the advantage of default visualisations — that the developer does not need to create them — but also support the usefulness of custom visualisations in particular circumstances. This suggests the development of reusable libraries of custom visualisations for robotic data that has more complicated semantics, so that developers can create their own specific visualisations with less effort.

Code development process:

- AR visualisation was considered useful as the software was developed, to enable testing at each step of added functionality.

Performance issues:

Data and timing accuracy:

- The lag in the AR system sometimes made it difficult to see what was happening in fast motions. One participant commented that some spikes in the laser data were not shown in the AR view, probably because of the slower refresh rate of the AR system.
- Another participant commented that it was difficult to distinguish between sensor noise and noise in the AR system. This was specifically related to a rotational jitter that is the result of the fiducial tracking method. This could be improved through a better fiducial extraction method, some sort of position filtering or an alternative tracking technology.

Occlusion: In some locations in the robot's field of operation the arm on the robot would obscure the fiducial on the back of the robot causing the system to loose tracking. Comments indicated that this was not a serious problem.

Physical layout: Several participants had some trouble with the AR view being some distance away from the developer workstation meaning they had to leave the development workstation to examine some of the smaller renderings. However the robot itself was unable to be viewed directly at all from the developer workstation. In future studies more care should be taken in the layout of the robot test environment with respect to the developer workstation.

6.6.2 Errors Located With AR System

The AR view allowed one participant to find an error in a bearing calculation, although it was unable to identify the cause of the error, just indicating that it was wrong. Print statements were used to further track down the error.

Another participant used the AR view to locate errors in the expected coordinate systems and calculations of quadrants for locating the target object. AR was also used to identify an issue in calculating the bearing of the target object (the minimum scan angle was not being included in the calculation)

When running the block finder task the participant identified a bug where the robot would incorrectly identify the wall as an object of interest. This was highlighted by the AR visualisation, and the participant commented this was easier to see visually.

Another participant found an AR rendering of the calculated target useful for debugging the follower task. The AR view made it obvious that the tracked target was changing, i.e. that the following code was correct; instead it was the target identification that contained the bug.

The AR display of the located edges for the participant's block finder showed that the system was not robust to some sensor noise. This was obvious in the AR display but harder to see in text output as many of the fluctuations were transitory and were lost in the correct measurements in text.

6.6.3 Limitations of the Participant Study

No participants used a structured development process nor a traditional debugger such as GDB when debugging their tasks; all chose to use either print statements or the AR visualisations. Additionally no participant chose to use a simulation environment to test their applications before using the real robot. The participants were skilled robotic software engineers, but it seems they were not strongly motivated to use debugging tools and aids.

The fact that participants needed to be reminded to continue to take notes implies that many of their thoughts on their programming and the AR system were probably lost.

A future study could use an ethnographic style approach where an observer takes notes about the activities of a group of programmers working on a medium scale robotics project. This could be augmented with screen and video recording of the environment and interviews with the programmers.

This would allow the use of an AR system to be examined alongside formal programming processes. The longer time period would also account for both the learning curve of the visualisation system and the novelty factor of the AR system. The disadvantage of such a study would be the requirement of significant observer resources and the need for an AR system to be installed in an environment that was undertaking a real robotics project, most likely in a commercial company.

7 Discussion

Although the trials were relatively simple, they represented important subcomponents of larger real robotic applications. While qualitative metrics were not obtained with these trials some important observations were made.

7.1 Benefits of ARDev

7.1.1 Standard visualisations are useful

The standard visualisations for Player minimised development effort expended on creating new visualisations. The custom visualisations used the *graphics2d* and *graphics3d* interfaces to render important pieces of meta-data. Using *graphics2d* further reduced the effort since the same visualisations could be used for testing both in simulation and with the real robot.

The ability for developers to visualise robot data using AR, but without requiring modifications to the client code, is important because it makes the debugging overhead minimal. We feel that often developers avoid using debugging tools because of even small overheads in using the tools.

7.1.2 Data validity and prevention of false conclusions

One evident, important benefit of the AR system was the ability to prevent false conclusions being drawn about the validity of data. Data that is qualitatively correct but containing basic errors, such as being inverted, offset or mirrored, often looks correct during casual inspection. When these data are viewed against a real world backdrop these errors are immediately visible.

These simple errors are common in development and if they are passed over initially they can inflict long debugging episodes as many irrelevant components of the application are checked in detail. During these three trials there were at least three occasions when such errors were found; the swapped minimum and maximum scan angles in trial one, the laser offset in trials two and three, and the base offset for the limb in trial three.

7.1.3 Same visualisations for simulated and real robots and environments

The block finder trial had very few errors when changing from the simulated world to the real one. This was largely because the program was first tested in the simulator, and because the task is suited to simulation. The visualisation created during simulation was important as it highlighted a deficiency in the laser model in Stage.

This advantage also allowed the developer to focus on the real performance issues relating to the task, tuning the speeds of turn and approach, without needing to continually check the basic algorithm performance as this was displayed clearly in the AR view.

7.1.4 Immediate confirmation of hardware performance limitations

Throughout the third trial the visualisation was able to confirm that most of the failures occurring were in fact due to limitations in the underlying capabilities of the arm. The arm would often miss the target pickup point on the block while the visualisation identified that it thought it was at the correct location. The offset between these two locations is due to the low cost nature of the servos on the arm. The conclusion of the third trial was that while the application went through the correct motions of performing the task, due to variation in the arm it was only able to achieve a 5-10% success rate. If increased performance was desired either an improved arm or closed loop control would be needed.

7.1.5 Monitoring role

A potential negative effect on developer performance occurs when developers invest time creating renderings which either do not aid in finding bugs, or when the visualisations are created pre-emptively and no bug is found in the code. The general feeling of the comments from developers was that the visualisations continue to pay off by playing a monitoring role even after the specific feature they were written to debug is functioning. This is often not the case with print statements in code that are generally removed or disabled once the bug they were targeting has been resolved.

7.2 Potential extensions to ARDev

7.2.1 Need for visualisation of abstract data

In all three trials abstract data such as the current state or task progress were rendered to the console using plain text output. The added effort of rendering these data, particularly given the lack of support for text in the AR library, was seen as greater than the benefit of having it embedded in the environment. The rendering of these data may become more useful if the abstract state were more complex or there were more individual elements to be concurrently viewed. If an immersive AR system were used these data would be easier to understand if rendered in the virtual view.

7.2.2 Need for 3D immersion

The need for an immersive 3D view of the data was clear to the developer while performing trial three. Given the true 3D nature of the arm's movements it was sometimes difficult to tell the exact approach vectors and location of 3D target points in the environment. Also to understand some of the 3D elements it would have been useful to be able to shift the camera perspective.

7.2.3 Independent rendering library

In general the independent participant's comments fit well with THJC's. One of the important differences was in the effort required to create renderings. THJC had greater familiarity with the rendering system and so the effort to create the visualisations was not as high as for the other participants. However, several of the participants commented on the amount of code needed to generate the visualisations, although: a) this effort was needed only when explicitly creating a custom rendering, b) the base overhead of using the AR system when not creating custom visualisations was almost zero, and c) any effort creating visualisations for simulation can be reused at no cost in the AR environment.

Still, the effort for creating custom visualisations could be reduced by providing a rendering library of useful primitive graphical objects.

7.2.4 Noise

Two of the participants commented on noise in the AR system, one with respect to rotational jitter and the other with respect to the differing frame rate of the AR data and the laser data. Comments along these lines show there is obviously some room for improvement in the base performance of the system, however in general the participants found the system sufficient for most of their debugging needs.

Our system simply displays the data as it arrives. Although Player does provide timestamps we were surprised how useful the system was without better synchronisation.

8 Future Work

The performance of the IDS should be measured through additional user trials, in particular it would be valuable to undertake an in depth study of a small number of developers using the system for a large scale robotics project. A comparison of different types of AR display is also needed. Another area for further study is the choice of visualisations. Further comparisons should be undertaken by implementing ARDev in other frameworks.

AR also has potential in many other areas of HRI. An augmented view can improve a user's understanding of the robot by extending the shared perceptual space of the interaction. AR also provides an out of band data channel for feedback in normal interaction; a robot system could display its interpretation of the components of a task while the task is being explained. As the AR community develops and improves the hardware and software offerings the cost of AR systems will decrease, opening up many more opportunities.

An important area of future research is to *quantify* how much the AR visualisation aids the developer, including a comparison of configurations, such as the immersive head mounted display versus fixed viewpoint AR. Our experience suggests that the head mounted display has significant draw backs in terms of the effort needed to attach and use the display. In particular the restrictive cabling and low resolution of the available hardware was found to be limiting.

9 Conclusions

Robotics presents a unique set of challenges that change the way that we must approach programming. Debugging in particular must be approached as the task of understanding how the robot is seeing the world and any inconsistencies or deficiencies with this view. AR provides enormous opportunities for enhancing debugging, allowing the developer to see the world as the robot does, intuitively displaying the limitations and discontinuities in the robot's world view. This paper presents the analysis, conceptual design, reference implementation, and evaluation of an AR enhanced intelligent debugging space that allows the developer to view the robot's data and meta data in context with the environment the robot is operating in. The analysis gives guidelines for future designers of AR–based debugging systems, and gives a modular framework for the software design that emphasises standard visualisations in order to minimise the overhead for the developer of robotic software.

The designed AR system provides an opportunity to understand the type of errors that are encountered during debugging. Debugging is essentially a process of elimination and by understanding the type of error developers can quickly eliminate large sets of potential bug sources, substantially reducing debugging time. Without our AR system, when a developer makes an incorrect assumption about the source of the bug, then a disproportionate effort is needed before all the potential sources of the bug can be eliminated, and the developer can finally realise that it is their initial assumption about its source that is incorrect.

AR visualisation also provides an important role as a stepping stone between simulation and real world testing. One of the most valuable attributes of simulators, apart from preventing damage to expensive hardware or people, is the ability to see inside the robot, and to see data measured against the ground truth model of the world. AR allows data to be represented in the same way against the ground truth of the real world. This allows the developer to focus on the key issues, such as isolating any bugs that arise due to over simplified simulation models.

References

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

[2] "Microsoft robotics studio," http://msdn.microsoft.com/en-us/robotics/.

[3] O. Robotics. (2006, July) http://orca-robotics.sourceforge.net/.

[4] CARMEN. (2006, July) http://carmen.sourceforge.net/.

[5] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Ng, "ROS: an open-source robot operating system," in *Workshop on Open Source Software in Robotics. IEEE International Conference on Robotics and Automation (ICRA)*, May 17th 2009.

[6] I. Hara, N. Ando, S. Nakaoka, F. Kanehiro, H. Hirukawa, S. Hirai, K. Takahashi, F. Hara, H. Nakamoto, Y. Takano, and H. Saito, "OpenRT Platform: An open software platform for robotic technologies," in *Workshop on Open Source Software in Robotics. IEEE International Conference on Robotics and Automation (ICRA)*, May 17th 2009.

[7] T. Kooijmans, T. Kanda, C. Bartneck, H. Ishiguro, and N. Hagita, "Accelerating robot development through integral analysis of human-robot interaction," *IEEE Transactions on Robotics*, vol. 23, no. 5, pp. 1001–1012, 2007.

[8] J. Kramer and M. Scheutz, "Development environments for autonomous mobile robots: A survey," *Autonomous Robots*, vol. 22, no. 2, pp. 101–132, 2007.

[9] 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.

[10] 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, Jul. 2003, pp. 317–323.

[11] F.-É. Trépanier and B. A. MacDonald, "Graphical simulation and visualisation tool for a distributed robot programmin g environment," in *Proceedings of the Australasian Conference on Robotics and Automation*, CSIRO, Brisbane, Australia, December 1–3 2003. [Online]. Available: http://www.araa.asn.au/acra/acra2003/papers/28.pdf

[12] "eXperimental Robotics Framework," http://erff.berlios.de.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

[22] M. Stilman, P. Michel, J. Chestnutt, K. Nishiwaki, S. Kagami, and J. Kuffner, "Augmented reality for robot development and experimentation," Robotics Institute, Carnegie Mellon University, Tech. Rep. CMU-RI-TR-05-55, November 2005.

[23] J. Chestnutt, P. Michel, K. Nishiwaki, M. Stilman, S. Kagami, and J. Kuffner, "Using real-time motion capture for humanoid planning and algorithm visualization," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, May 2006, [video].

[24] K. Kobayashi, K. Nishiwaki, S. Uchiyama, H. Yamamoto, and S. Kagami, "Viewing and Reviewing How Humanoids Sensed, Planned and Behaved with Mixed Reality Technology," in *IEEE-RAS 7th International Conference on Humanoid Robots (Huamnoids 2007)*, Pittsburgh, USA, November 29 – December 1 2007. [Online]. Available: http://planning.cs.cmu.edu/humanoids07/p/35.pdf

[25] DDD – Data Display Debugger. (2005, August) http://www.gnu.org/software/ddd.

[26] Eclipse. (2005, August) http://www.eclipse.org.

[27] T. H. J. Collett, B. MacDonald, and B. Gerkey, "Player 2.0: Toward a practical robot programming framework," in *Proc. Australasian Conf. on Robotics and Automation*, December 2005. [Online]. Available: http://www.araa.asn.au/acra/acra2005/papers/collet.pdf

[28] T. H. J. Collett and B. MacDonald, "Developer oriented visualisation of a robot program," in *Proc. Conference on Human-Robot Interaction*, Salt Lake City, Utah, March 2–4 2006, pp. 49–55.

[29] T. H. J. Collett and B. A. MacDonald, "Augmented reality visualisation for player," in *Proc. IEEE Int. Conf. on Robotics and Automation (ICRA'06)*, Orlando, May 2006, pp. 3954–9.

[30] T. H. J. Collett, "Augmented reality visualisation for mobile robot developers," Ph.D. dissertation, University of Auckland, 2006. [Online]. Available: http://hdl.handle.net/2292/1510

[31] Handheld Augmented Reality. (2006, August) http://studierstube.icg.tu-graz.ac.at/handheld_ar/artoolkitplus.php.

[32] H. Kato and M. Billinghurst, "Marker tracking and hmd calibration for a video-based augmented reality conferencing system," in *Proceedings of the 2nd IEEE and ACM International Workshop on Augmented Reality*. Los Alamitos, CA, USA: IEEE Computer Society, 1999, pp. 85–94.

[33] L. Prechelt, "An empirical comparison of C, C++, Java, Perl, Python, Rexx, and Tcl for a search/string-processing program," Fakultät für Informatik, Universität Karlsruhe, Germany, D-76128 Karlsruhe, Germany, Tech. Rep., March 2000. [Online]. Available: http://wwwipd.ira.uka.de/EIR

[34] J. Nielsen, *Usability Engineering*. Academic Press, 1993.

[35] J. Ziegler, "Interactive techniques," *ACM Computing Surveys (CSUR)*, vol. 28, no. 1, pp. 185–187, 1996.

[36] C. Hundhausen, S. Douglas, and J. Stasko, "A meta-study of software visualization effectiveness," *Journal of Visual Languages and Computing*, vol. 13, no. 3, pp. 259–290, 2002.

[37] A. Steinfeld, T. Fong, D. Kaber, M. Lewis, J. Scholtz, A. Schultz, and M. Goodrich, "Common metrics for human-robot interaction," in *Proceedings of the 1st ACM SIGCHI/SIGART conference on Human-robot interaction*. ACM New York, NY, USA, 2006, pp. 33–40.

[38] B. Ellis, J. Stylos, and B. Myers, "The factory pattern in api design: A usability evaluation," in *Proceedings of the 29th international conference on Software Engineering*. IEEE Computer Society Washington, DC, USA, 2007, pp. 302–312.

[39] MobileRobots. (2006, June) Mobilerobots activmedia robotics. http://www.mobilerobots.com/.

[40] Hoyuko Automatic. (2005, August) http://www.hokuyo-aut.jp/.

Toby Collett received a BE (1st class) and Ph.D in the Electrical Engineering department of the University of Auckland, New Zealand. Toby is a founder of Inro Technologies, an Auckland based robotics company that produces autonomous vehicles. He is also a key developer for the Player Project, a popular open source robotics platform. His key research interests are in human-robot interaction, augmented reality and robot programming. Toby was named Young Engineer of the Year at the 2008 New Zealand Engineering Excellence awards.

Bruce MacDonald received a BE (1st class) and Ph.D in the Electrical Engineering department of the University of Canterbury, Christchurch, New Zealand. He spent ten years in the Computer Science department of the University of Calgary in Canada then returned to New Zealand in 1995. There he joined the Department of Electrical and Computer Engineering Department at the University of Auckland, led the initiation of a new Computer Systems programme, and set up the Robotics laboratory. His research interests include robotics and intelligent systems.