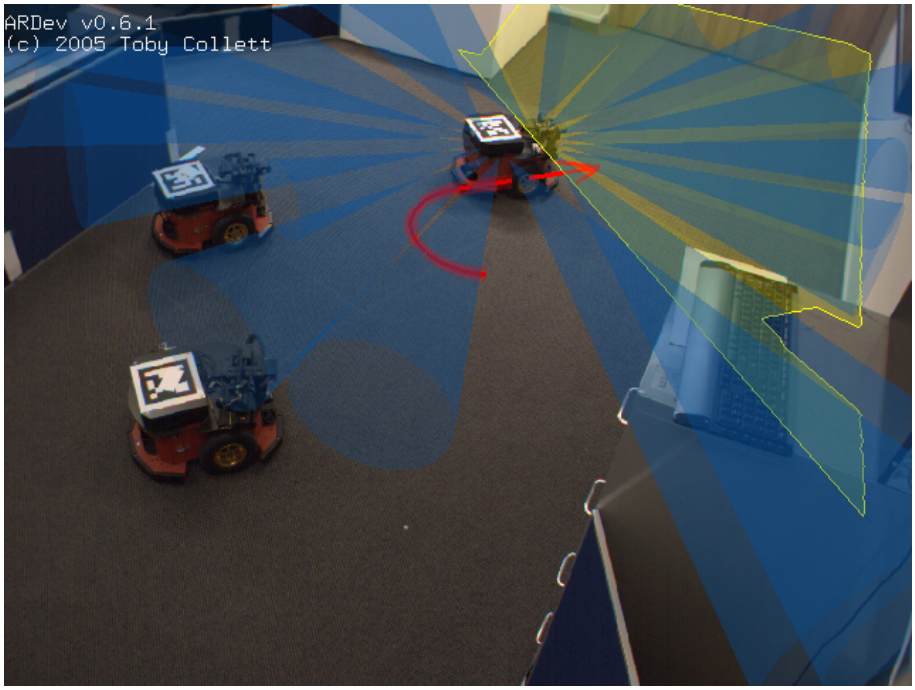
Department of Electrical and Computer Engineering
The University of Auckland
New Zealand



Augmented Reality Visualisation for Mobile Robot Developers

Toby H. J. Collett

August 2007

Supervisor: Dr. Bruce MacDonald

A thesis submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Engineering,
The University of Auckland, 2007

Abstract

Developer interactions with robots during the testing and debugging phases of robot development are more complex than and distinct from general software application development. One of the primary differences is the need to understand the robot's view of the environment and the inconsistencies between this and the actual environment. Augmented reality (AR) provides an ideal way to achieve this, allowing robot program data to be displayed in context with the real world. This allows for easy comparison by the developer, highlighting the cause for any bugs in the robot behaviour. An AR debugging space is created in this work that allows the developer to have this enhanced understanding of the robot's world-view, thus improving developer efficiency.

Over the past decade robots have begun to move out of industrial assembly lines and into environments that must be shared with human users. Many of the tasks that we wish robots to perform in these environments require close interaction and collaboration with human users. The move away from the constrained environment of a production line means that the tasks required of robots are more varied, their operating environment is far more complex and unpredictable, and safety can no longer be achieved through isolation of the robot.

The result of these influences has been to change robot programming from a simple task of instructing the robot to perform a sequence of steps to an open ended challenge of specifying dynamic interactions that robot developers are still coming to terms with. Robot development is more than just design and code entry and a broader approach to improving robot development is needed. One of the founding principles of this thesis is that robot development should be approached as a human-robot interaction issue, this particularly applies to the testing and debugging phases of the development.

The nature of the robot platform, the tasks the robot is required to perform and the environments that robots work within are significantly different from those of the desktop application. Hence robot developers need a tailored tool chain that focuses on this unique combination of issues. Current robot programming research is dominated by robot APIs and frameworks, leaving support tools to be developed in an ad hoc manner by developers as features are required. This leads to disjointed tools that have minimal feature sets;

tools that generally have poor portability when applied to other robot developments. This work examines the needs of the developer in terms of a general purpose robot visualisation tool.

One of the fundamental requirements of a general purpose robot visualisation tool is that a set of stock visualisations must be available for the developer. A prerequisite to providing these is to have a set of standard interfaces to provide the visualisations for. The open source robot framework Player/Stage was used throughout this work to provide standardised access to robot hardware. As part of this research the author has contributed heavily to the Player/Stage project, particularly as one of the key developers of the 2.0 release of Player. This new release simplifies Player development and increases the ease of maintenance of Player drivers and the efficiency of the server core.

To evaluate the benefits of AR visualisation an intelligent debugging space was developed, which runs as a permanent installation in a robotic development lab providing an enhanced view of the robot's behaviour to the developer. The space is capable of automatically detecting the presence of robots and displaying visualisations of the standard interfaces of the robot, such as its sensors and effectors. The debugging space also allows the developer to create custom renderings, leveraging the developer's ability to determine the most salient items of their code and display these.

A set of representative case studies was carried out using the debugging space for testing and debugging. These studies showed that AR 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, focusing on the sections of code that are causing the bug and therefore substantially reducing debugging time.

The implemented system also shows that AR provides an important stepping stone between simulation environments and the real world.

This thesis contributes the novel approach of applying AR to developer interactions with robots. The use of AR has been shown to have significant benefits for the robot developer, enhancing their understanding of the robot's world-view and hence reducing debugging time. As part of the work a flexible AR visualisation tool was developed with close integration to the Player/Stage project. This tool creates an intelligent debugging space where developers can exploit the benefits of the AR visualisation with minimal overhead.

Acknowledgements

First and foremost I would like to acknowledge the support and contributions of my supervisor Dr Bruce MacDonald. His enthusiasm and ideas have helped shape this work from our early discussions on human-robot interaction through to the final stages of proofing.

I would like to thank my wife Katie for her support and encouragement which have helped me through the last four years, and for efforts in proof reading. I would also like to thank my family for their support and help with proof reading.

The Player project has been an indispensable part of this work and I would like to thank Brian Gerkey and the rest of the Player developers for creating and supporting an excellent robotics tool.

I would like to thank Grant Sargent and the rest of the University of Auckland robotics group for their ideas and for making the robotics lab a great place to work.

Finally I would like to acknowledge the financial support of the New Zealand Tertiary Education Commission through the Top Achiever Doctoral scholarship.

Contents

List of Figures

List of Tables

1
Introduction

Over the past decade robots have begun to move away from industrial assembly lines and into environments that must be shared with human users. Many of the tasks that we wish robots to perform in these environments require close interaction and collaboration with human users such as; care of the elderly [125], team construction tasks [75, 150, 178], household assistants such as iRobot's Roomba [71], and "guide dog" robots [83, 152].

The move away from the constrained environment of a production line means that the tasks required of robots are more varied, their operating environment is far more complex and unpredictable, many tasks involve dynamic interactions with human users, and safety can no longer be achieved through isolation of the robot. In addition these tasks require greater capability (and therefore complexity) in robots.

The result of these influences has been to change robot programming from a simple task of instructing the robot to perform a number of steps in sequence to an open ended challenge that robot developers are still coming to terms with. Research has focused on promoting code re-use for large systems by standardising programming frameworks and APIs. While there has been significant progress in this area, enhanced libraries are not the complete solution to enhancing robot development. Robot development is more than just design and code entry and a broader approach to improving robot development is needed. This thesis examines robot programming as a human-robot interaction task, considering the full development cycle and toolchain.

In order to show how the robot developer's toolchain can be enhanced, the robot

development task itself must first be considered. The remainder of this chapter examines the robot development process and outlines the proposed enhancements to the debugging toolset through augmented reality (AR) visualisation.

1.1 Nature of Robot Development

While the *process* of robot programming is similar to general software development at a high level (i.e design, code, test and debug), developers of robotic systems face a number of application specific issues that make the task more complex. The issue of additional complexity in domain specific development is not unique to robotics, for example the telecommunications industry also has many additional challenges for developers [3], however robotics (beyond industrial automation) is still an emerging field and these issues have largely been ignored to date.

Before the specific issues are explored the scope of the term "robot" needs to be set. For the purposes of this work the term robot will be used to refer to a mobile device with some degree of autonomy which is able to perform meaningful tasks by interacting with the environment, including human users. While the scope of this work focuses on mobile robots, it is the authors belief that many of the techniques should be applicable to manipulators as well.

The ways in which robot development differs from general software application development can be divided into three key areas; the robot platform itself, the tasks the robot is required to carry out and the robot's interactions with the environment and human users. The robot platform includes the software infrastructure as well as the hardware system.

The nature of the robot platform

It is often difficult to interact physically with a robot, particularly while it is operational. Many robots are mobile, so the target hardware does not stay in one place. Robots may have moving manipulators that require free space or a safety exclusion zone. Furthermore the act of being in proximity to the robot may alter its performance, for example if the developer is being detected as an obstacle in the robot's laser scanner.

This lack of physical interaction immediately creates a barrier for developers, requiring debugging via a remote connection. The mobility of many platforms also means that perspective taking, the understanding of data from another point of view, is needed in order to understand the robot's view of the world. This increases the cognitive load of the programming tasks and may lead to incorrect assumptions about the cause of flaws in the robot's behaviour.

Robots have a large number of devices for input, output and storage which far exceed human programmers' familiar senses and effectors. The use of unfamiliar devices of which the developer has no intuitive internal model leads to difficulties in understanding the data produced and thus the cause of a robot's behaviour. Human vision, a common sensor between humans and robots, and one of the most advanced human sensors, is vastly different from machine vision. Many of the assumptions humans make about vision cannot be applied in the robotic domain, for example the perception of a set of green items by a robot will be dependant on a number of variables including variations in lighting, assumptions about shadow and physical limitations of the digital image acquisition process such as dynamic range and resolution.

The wide range of devices also leads to wide variation in software interfaces. In comparison, in the highly commodified desktop environment most devices are accessed through highly abstract APIs that work with standard industry components. This variation in interfaces for robot devices will partly be reduced with standardisation over time but the variation between the properties of different types of sensors will still remain. For example both laser and ultrasonic devices measure distances to objects in the environment but have different limitations, specifically in terms of the effect of angle of incidence and the material of the objects.

In addition to having a large range and number of input and output devices these devices can all have simultaneous and unrelated activity, leading to greater difficulties in understanding the current state of the robot.

Robot systems can also have high levels of concurrency and can be highly distributed, further adding to the difficulties of programming them. The concurrency can arise from multiple threads reading from individual pieces of hardware, such as with the Player project, or in reactive systems where code is triggered from external events. Distributed systems are essential where a single computer is unable to provide sufficient processing resources, or where the system is made up of physically distinct units such as a group of exploration robots. Multiple controllers can also provide redundancy.

The nature of the robot environment

Robots operate in the real world which is dynamic and realtime. Many techniques for debugging desktop applications will fail in these conditions, for example, as a result of time delays and sampling effects, stepping through an application that is reading data from a real world sensor may give different results from letting the program run free. Often there are unexpected variations in the environment that cause non-repeatable behaviour and unexpected conditions in robot programs. This variation is important when moving from a test environment to a final robotic product.

The nature of mobile robot tasks

Mobile robot tasks usually have an inherent emphasis on geometry and 3D space; they operate in the real world and must deal with real world geometries. The developer's interpretation of raw 3D data is often error prone particularly when dealing with multiple coordinate systems and 3D rotations. The developer must perform mental perspective changes in order to understand the debugging output of an application. In addition to geometric data, robots need to present other complex data types including stochastic data, sequential data, plans and other abstract data types (such as utility or emotional state).

Robot tasks often cannot be interrupted, for example interrupting robot programs may have safety implications if the interrupted component is a safety critical one. A mobile robot travelling at high speed cannot simply be stopped by a debugger while the developer examines program variables. Also the process of interrupting a robot program to observe it can change the program state and make the observations meaningless. The environment continues to change while the robot application is in an interrupted state, causing similar difficulties.

1.2 Enhancing Developer-Robot Interaction

These unique aspects of robot development lead to unique challenges and opportunities in enhancing the interaction between developers and robots. Interaction between the developer and the robot occurs in the testing and debugging stages of development as the design and coding stages are generally performed offline. Even when modifying the code and design during the testing phases the developer will interact with a development environment rather than directly with the robot system.

During the testing and debugging stages the goal of the developer is to find and isolate any errors and inconsistencies in the robot program and to identify the causes of these. One of the most significant differences in this task from the task of testing and debugging desktop applications is the emphasis on the environment and understanding of the robot's world view.

Humans have relied on visual representation of data for almost as long as they have been able to use tools. Early cave paintings show the use of images to keep track of flock sizes [175] which were later simplified to strokes to increase efficiency. The initial uses of visual representation were mainly for storing information, either numerical quantities such as the flock sizes or historical events and cultural traditions.

In the mid eighteenth century Playfair produced his *commercial and political atlas* [128]; the first work to make broad use of charts and graphs to provide simplified representations of abstract data quantities [170,175]. The importance of this work is that

Playfair's graphs were used to enhance understanding of an abstract dataset and hence it is the origin of the field of data visualisation. Playfair's style of graphics including line and bar charts are now second nature with the production and interpretation of these graphs being taught in primary schools.

It is important to focus on what the user needs to visualise rather than just visualising the components of a robot program directly. For Playfair, the user needed to see the key features and relationships of the dataset, for example the relationship between export quantities and political events. For a service robot, such as the Roomba [71], features such as task status and robot health, e.g. battery level or filter state, are the most important to display. Recent work by Young *et al* [182] presents Roomba state as Bubblegrams, allowing the user to "see" the state of the robot.

In the programming case, the user is the robot developer so it can be difficult to separate the user's needs from a direct representation of the underlying system, as the developer is interested in seeing the details of the system they are developing. From the developer's point of view it is important to highlight differences between what the program state is and what it should be. Additionally developers need control over which parts of the system are displayed at a given time as well as control over the way in which they are displayed, such as the level of detail available for individual components.

This work presents the idea of enhancing developer understanding of robot data and world-view through the use of AR visualisation techniques. Effective visualisations aid the developer in many of the challenges they face. Specifically it is proposed that visualisation can:

- aid in the understanding of 3D geometric data and other complex data sets;
- decrease the time needed to understand robot state, mitigating the non-repeatable and non-interruptable nature of the robot's environment and tasks (if the developer can comprehend robot data in real time, repeated tests may be avoided);
- isolate flaws in the underlying robot platform (saving significant time spent searching for bugs in working application code);
- aid in understanding unfamiliar sensors and actuators on the robot platform;
- increase understanding of how the robot perceives the environment, i.e. the robot's world view, and hence any inconsistencies in this.

Visual information is part of every aspect of our daily lives, whether as part of the advertising we see on the way to work, in the graphics presented on the evening news or the funny photo that was sent around the office. Technology trends are for richer multimedia experiences including cameras as a standard feature in laptops and cellphones,

and internet sensations such as YouTube [183] allowing media to be shared among millions of viewers. Effective visualisation allows the user to approach the data as an entity rather than an abstract data set, enabling them to process the information at a different cognitive level. While it can be difficult to quantify the effectiveness of graphical visualisation, particularly in a relatively new reasearch area such as human robot interaction, by carefully designing our visualisation we can make optimal use of human perceptive abilities [140].

An additional advantage of visualisation is the ability to view the quality of the outcome for a robot task. If there are a range of acceptable outcomes, it is important for the developer to be aware of the distribution of the results within this range. If the results are consistently borderline in a controlled testing environment then it is likely that failures would occur in the final installation.

This work focuses on enhancing developer interaction with robots. It is likely that the techniques presented in this work will be able to be applied to other human robot interaction roles such as robot operators and social interaction partners but this is left to future work in the field.

1.3 Augmented Reality

AR is a technology that allows the user's view to be enhanced by overlaying virtual elements that are registered to items in the real world. This opens up a whole range of new methods for interaction and data display, a detailed review of AR is provided in Section 2.4.

By leveraging AR's natural ability to display environmental data in context we can greatly enhance the developer's ability to understand the robot's perception of the world and any inconsistencies with this. The questions that this work examines are:

- how can AR be applied to robot development?
- what paradigms for interaction are possible?
- how should robot data sets be represented?
- how can the visualisation be integrated into the developer's existing environment?
- what hardware and software infrastructure is needed to support an AR developer system?

1.4 Contributions

- This work introduces the application of AR to the arena of developer robot interaction, providing an analysis of the benefits of AR for developer robot interaction and presenting the paradigms that can be used for providing AR visualisation for robot developers.

- The work provides a discussion of the performance of AR as a visualisation technique for developers. This can be used to inform future study of AR as a developer interface.

- To properly explore the applications of AR for the robot developer an AR toolkit and interaction system were developed by the author. These are available as an open source project which will greatly aid future research in AR for robotics as well as being an aid for robot developers in its own right.

- The principle of creating stock visualisations for developers requires a standard representation for data; The Player project [127] provides one such standard representation. As part of this research the author has made significant contributions to the Player project becoming a key developer for the version 2.0 release [41]. Specifically these contributions include rewriting the player core message queues, simplifying the driver API and implementing a number of hardware and software drivers. Player/Stage is an open source project that is used in a large number of robotics labs world wide, with these researchers benefitting from the improvements in the version 2 release.

1.5 Structure

The remaining chapters of this thesis will present an application of AR to the domain of developer-robot interaction.

Chapter 2 presents related work from the research community; Chapter 3 examines how AR can help robot developers; Chapter 4 presents a conceptual AR system for robot developers. Chapter 5 presents the implementation of the AR toolkit and debugging space with Chapter 6 describing the evaluation of these using a series of case studies. Chapter 7 presents the contributions made to the Player/Stage project during the research. Finally Chapter 8 contains the closing remarks of the work, summarising the contributions and outlining both the potential for future work in this area and the application of the findings in other areas of robotics.

2

Literature Review

This chapter is divided into 4 parts covering each of the major areas of research that contribute to this work: robot programming, human-robot interaction (HRI), visualisation and augmented reality. Section 2.1 presents the current state of the art in robot programming research including work in robot frameworks, APIs, languages and programming tools. Section 2.2 presents the task of programming robots in the context of HRI and examines how current research in HRI can be applied to the programming role. Section 2.3 presents a summary of current work in the visualisation research community with Section 2.4 covering AR specifically.

2.1 Robot Programming

In general the goal of a robot developer is to enhance the capabilities of a robot allowing it to carry out new tasks, or to perform existing tasks better. For the purposes of this work the *process* of robot development will be considered to be similar to traditional computer programming: that is the design, coding, testing and debugging of robot programs. This definition excludes the role of the robot instructor who shares the same goal as a robot developer, the enhancement of capabilities, but achieves this through training the system rather than coding it. This definition also excludes the creation of the base robot platform, which from an interaction viewpoint is an embedded systems task. The robot developer deals with the robot from the point where it is a robot in an abstract sense consisting of

a set of sensors and effectors.

A simple example of a robot programming task would be programming a Pioneer robot [104] to follow a human user using the Player API [127].

The parts of the toolchain that will be examined in this section will be those that relate to the developer's interface. For example while a compiler or interpreter is essential for translating code into an executable program, the developer interface to the compiler is through the programming language and associated tools.

2.1.1 Robot Development Technologies

Developers use a wide variety of technologies when carrying out their work, which can be separated into languages, libraries and support tools. In general the robot developer uses a combination of these tools from both the desktop and robot development domains. The robot specific technologies can again be divided between proprietary vendor technologies, generic robotic technologies and custom developments.

One of the major weaknesses in the robotics community is the disproportionately large number of custom solutions being used, this leads to much duplication of effort between projects and between researchers. Reducing this duplication of effort requires a two pronged approach both increasing standards within the robotics community and the production of tools that are designed for the general task of robot development.

This work is aimed at the creation of one such general robot programming tool and utilises the Player/Stage project as a de facto standard to enhance interoperability.

Robot Programming Languages

In general industrial automation systems have used proprietary languages for programming their systems. These languages generally have similar syntax to popular languages such as BASIC and are specific to the vendor, in some cases even to the particular robot model [11]. These restrictions vastly limit the ability for developers to re-use code to perform a specific task with different robots, again resulting in wasted development effort.

In research robotics there is a preference towards existing general purpose languages such as C/C++, Java and Python [32, 127, 142]. These are mostly extended through the use of libraries which will be described below. A notable exception is the RADAR language which adds robotic extensions directly into the semantics and syntax of the language, in this case through modification of the Python interpreter [13, 14].

Biggs [11, 12] has carried out an extensive survey of programming languages used in robotic systems, and the reader is referred to this for a more in depth treatment. Biggs argues that specialised languages are needed to effectively deal with the challenges of robot programming, particularly real-world data types and support for reactive applications.

Programming languages are essential for producing effective, robust and maintainable programs for any system. However, they are only a part of the full development picture and the availability of supporting libraries and tools is of key importance to the robot developer.

Robot Programming Libraries

The majority of robot programming research has focused on creating functionality in libraries and frameworks for robotic systems. Player/Stage [127] is becoming a de facto standard in the open source robotics world, providing hardware abstraction and network transparency for robot systems. Chapter 7 has more detail about the Player/Stage project including the author's contributions to the Player 2.0 release.

Libraries promote code reuse, reducing the amount of original code that developers need to produce. Libraries encapsulate complicated algorithms and a library with a well designed API makes parts of robot development much simpler and faster. However, an application will always need some development that addresses the issues specific to the task being carried out. This is more so in the case of robotics than with desktop application development as there is far greater variation in both the operating environment and in the interactions between the robot and the environment. The research nature of mobile robotics also leads to a greater need for custom development as there are fewer pre-built components available.

There is much activity already underway in developing robot frameworks and APIs but a good set of support tools is still needed for both the developers of the libraries and to bridge the gap from the library to the specific application problems.

This thesis will make use of the Player/Stage project in order to access a variety of hardware through a standard interface, thus increasing the potential user base of the developed visualisation tool. Other generic robot frameworks include work in real-time frameworks by Kuo [84], ORCA [28,142], MIRO [100], CARMEN [32], CLARAty [38,110, 111] and TDL [163].

Proprietary vendor libraries provide similar functionality as the generic robot projects but they are tailored to the specific abilities and application domains of the vendors' robots. An example is the ARIA library from MobileRobots [104]. ARIA provides access to the underlying robot hardware and can be combined with other packages from MobileRobots including ARNL to provide localisation and navigation functions and ACTS for colour vision.

Other vendors provide interface libraries of varying capabilities for interface to their robot platforms, for example KUKA [82] and Segway [147].

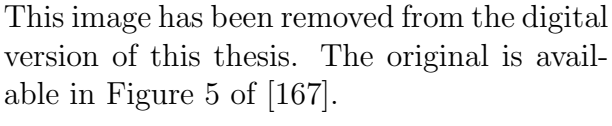


Figure 2.1: Screenshot of the GSV simulation and visualisation tool [167].

Robot Programming Tools

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. Player/Stage [127] provides a number of utilities designed to aid in this task. PlayerPrint and PlayerViewer respectively provide text and graphical representations of the basic Player data for the user. PlayerCam and PlayerNav also present robot data but are specifically tailored to vision and navigation applications. The MIARN project [97] provides a 3D data viewer for Player.

In addition to Player most of the robot frameworks such as ORCA [142] and CARMEN [32] provide their own data viewers. Vendor specific APIs such as ARIA [104] also have data viewers included.

These projects either display data in isolation or referenced against a static prebuilt map. The use of a static map can be extended into 3D, Trépanier *et al*'s GSV tool [167] uses the Torque game engine to create a 3D environment for both simulating and viewing robot data. Figure 2.1 shows a rendering of a simulated B21r and it's sonar beams. The use of a game engine allows for rapid development for both visualisation and simulation as the graphics and physics subsystems are already available. The game engine concept is also useful for simulating basic interactions between users and the robot as game engines are primarily real-time interactive systems. Faust *et al* [49] also use 3D game technology as the base of their simulator, an example screen shot is shown in Figure 2.2. Another 3D simulator based on a game engine, USARsim, is used for simulation of search and rescue operations [176].

While data viewers make up a significant portion of the tools available to robot developers, there have been some attempts to provide other support tools. An integrated robot development environment is described by Gumbley and MacDonald [60], in this work the

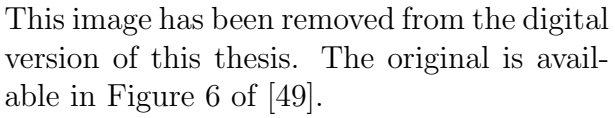


Figure 2.2: Visualisation of robot data in a 3D simulation [49]

Eclipse IDE is extended with a set of plugins designed to integrate with the Player/Stage project making it easier for developers to manage the complicated toolchain that robotics development requires.

Many of these tools have been created to suit the narrow purposes of a specific application or task. While this is suitable in some cases there are many possibilities for enhancements if tools are developed from the users' perspective, aiming to provide support for a broader range of tasks and robot platforms. Projects such as Player make this wide ranging support more achievable by providing a standard interface to robot systems. The developer oriented focus of this work will be expanded in Chapter 3.

2.2 Human-Robot Interaction

Developing a robot system inherently involves extensive interaction with the robot platform, particularly during the testing and debugging stages. As such much of robot development falls into the research arena of HRI. While there have been some ad-hoc tools that examine the specific case of robot developer interaction, these have not been formally developed. An examination of approaches to other HRI challenges is informative when considering how to approach the robot development domain.

This section first presents the role divisions of robot users from the HRI community, following this Sections 2.2.4-2.2.6 examine the interface media that are available for interaction with robots. The motivation of this examination is to determine interfaces that are likely to give good performance in the developer interaction tasks. Section 2.2.7 presents the current methods of evaluation in the HRI community.

2.2.1 The role of the user in Human-Robot Interaction

There have been many attempts to define sets of roles that users play when interacting with robots. The common theme is that in different roles the human-robot partnership has different requirements in terms of interaction capabilities. For example a robot that is able to learn new tasks through demonstration will have a different interface for interactions with its instructor, including more detailed feedback, than when being requested to carry out the learned task by an operator.

This work will extend the model presented by Scholtz [144], she defines six roles that a human user can take during interaction with a robot system; supervisor, operator, teammate, peer, bystander and mechanic. Here supervisor and teammate take the roles that they would in human to human interaction. Operators have direct control of the robot's behaviour. Bystanders do not have any task to perform with the robot but must co-exist in the same physical space. The final role of the mechanic is for physical service operations on the robot.

Other work includes Breazeal's four paradigms of interaction, the robot as a tool, cyborg extension, avatar and sociable partner [27]. Breazeal distinguishes these paradigms based on the mental model the human user has when interacting with the robots. The tool paradigm covers many of the current robot applications where the robot is simply a tool that is needed to perform a task. The cyborg extension refers to robots that are physically merged with the human body, such as an intelligent prosthetic limb. The avatar paradigm refers to projection of the user through a robot, providing a remote presence for the human user. The final paradigm is a robot capable of carrying out social interaction with human users.

Breazeal also explores the concept of using a biological model for enhancing interaction between humans and robots [26]. Here the idea of applying emotional intelligence to a robot allows humans to interact with robots in a similar fashion to their interaction with other humans.

There are a number of more traditional ways of examining the role of a user when interacting with robots. Use case analysis [19, 72] comes from the software community, here rather than identifying generic users of all systems, the focus is on identifying the typical users of a specific system allowing their requirements to be more precisely specified. Task analysis from the human factors community takes a similar approach, breaking down a task into subtasks and identifying all the roles users perform while carrying these out [109, 115].

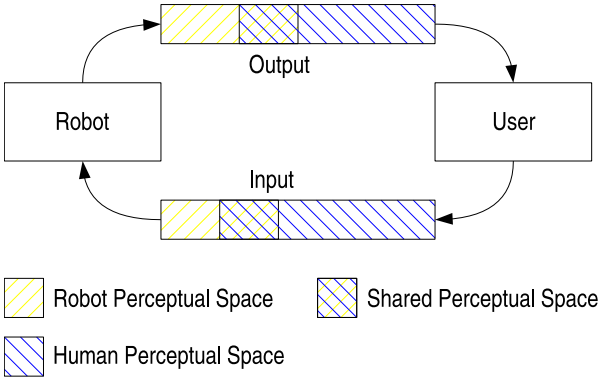


Figure 2.3: Shared Human-Robot Perceptual Space

2.2.2 Requirements for Effective Interaction

Both the human user and the robot have a perceptual space where communication is meaningful [23], and it is the overlap of these two spaces which is the interaction space of the human-robot coupling (Figure 2.3). This interaction space may be different for each direction of information flow (i.e. input and output). 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 space. Any mismatch between the perceptual input and output space may put additional cognitive load on the user as they will have to perform additional translations before communicating with the robot.

It is also necessary that the robot and human are in a situation where interaction is physically possible, normally requiring minimum and maximum physical distances to be maintained. Several approaches have been used to achieve this. Kismet uses facial gestures in order to attract attention of humans too far away or to repel users who become too close [23, 24]. A control algorithm is presented by Morioka *et al* [105] for following humans at an appropriate interaction distance and an alternative algorithm is presented by Prassler *et al* [131].

In addition, for interaction to be physically possible the environment must also be suitable. For example speech may not be a suitable interface in noisy environments and gestures would not work in environments with poor visibility. Many tasks also have further constraints for effective work to be carried out such as the presence of suitable tools and the absence of distractions.

To have natural communication it is important for a robot to include as many human interfaces as possible. Current research platforms for social interaction include an increasing number of interface methods; for example the WE-IV robot uses vision, smell, touch and sound [101] and Kismet displays a range of body language and primitive vo-

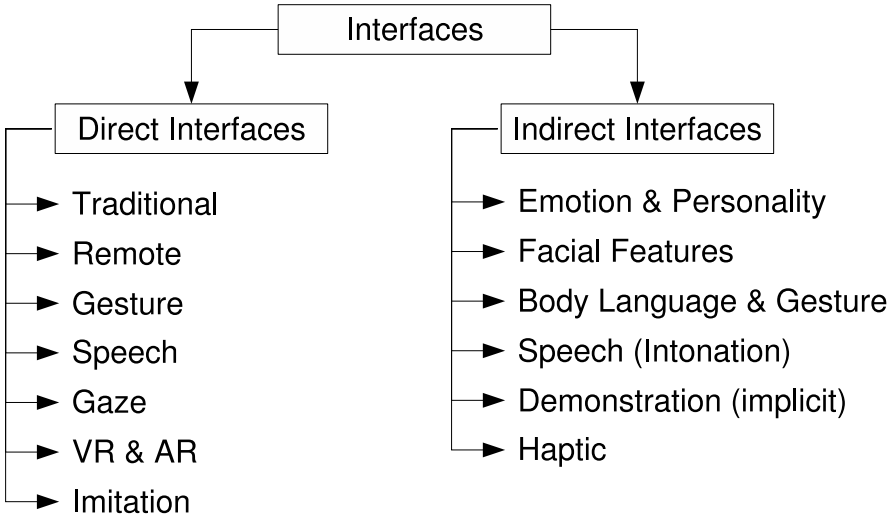


Figure 2.4: Interface Methods

calisation [23, 24].

A final consideration for HRI is that the interaction must be safe, particularly when there is very close interaction with personal assistant robots [130]. Colgate [39] states that with the increasing use of assistive devices on industrial production lines this is an immediate issue that needs careful consideration.

2.2.3 Human-Robot Interface Methods

As stated above, the robot and human user must share a common perceptual ability if they are to interact. This section describes the interfaces that are currently used in robotics to create this shared space, categorising them in a simple hierarchy as shown in Figure 2.4. By analysing the properties of these interfaces the best interfaces for developer interaction are able to be identified.

The interfaces have been divided into two groups, direct interaction, which results in a command or explicit piece of information being transferred, and indirect interaction where the information transferred is not explicit, such as emphasis, mood and context. Indirect interfaces are generally used in conjunction with direct interfaces to make the interaction more natural or convey deeper meaning such as emotion.

2.2.4 Direct Interfaces

Traditional Interfaces

Traditional interfaces predominately consist of keyboards, mice and joysticks for input and a computer monitor for output. There is still active research into how these interfaces can be made more effective, for example the Personal Rover Project uses traditional

GUI wizards for building a set of paths and tasks for a robot [48] and the MATS personal assistant robot uses a PDA based GUI with a joystick [56] or a similar system combining a small display with a data glove [67]. Traditional interfaces can also be combined with other interfaces, for example using gestures to indicate a command in a graphical interface [138].

One of the drawbacks of traditional interfaces is that many robots are mobile and thus require the ability for users to interact when there are no computer terminals available. Wearable computing and PDAs provide possible solutions, however this still restricts who can interact with the robot and so a range of alternative interfaces are being explored that provide mobile, hardware free interaction for example gesture and speech.

Traditional output devices also include simple alarm devices such as audio alarms or warning lights. A shared control robot system was used in experiments to determine the effect of these alarms as feedback methods in a simple manipulation task; the addition of audio, visual and force feedback alarms was found to improve operator performance [58].

Remote Interfaces

Remote interface methods may be used to control teleoperated robots, semi–autonomous task based systems, and to provide feedback or configuration with fully autonomous systems, for example streaming video and map data from an exploration robot.

The two significant areas in remote robot interface research are the computer interface and network performance, particularly response times. There is significant research aimed at improving teleoperated and telerobotic systems using web or network interfaces [37, 62, 88, 92–94, 99, 113, 156]. A method of dealing with network delays by presenting them to the operator via a force feedback device is described by Nohmi [119].

A common application of remote interfaces is remote exploration of museums, libraries [166] and art galleries by avatars [91]. They are also used for search and rescue scenarios.

Noninvasive and keyhole surgery are also application areas for remote operation systems. In order to improve the accuracy of movements some teleoperated interfaces create scaled movements from an input device so that very precise movements can be made during surgery [61]. These techniques can also be applied to micro assembly.

Sano *et al* implemented a general teleoperation system using a 3D display as the master interface [143].

Gesture

Hand posture and gestures can be used as a direct control method where each gesture is mapped to a command. Colombo *et al* [43] describe a system for interacting with an intelligent wall using pointing gestures. A general survey of hand gestures for controlling computers is given by Pavlovic [123], this covers the current range of techniques for

detecting hand posture and gesture, and typical applications. The most common hand pose recognition methods either use a data glove or image processing techniques [53].

The detection of hand posture and movement is also important for programming by demonstration [86] as the ability to learn manipulation tasks depends on the accuracy of the captured hand motion data.

Speech

Speech is a fundamental communication method between humans and so from a human perspective is a powerful interface between humans and robots, particularly for users unfamiliar with the robot system. Current research focuses on both the interpretation and the generation of speech; both the meaning and context of natural language are important. For example, a street map navigation robot [85] accepts speech commands from users such as "take the 3rd left." Another research group is also working on grounding ambiguous human speech by developing a full internal model of the world to relate the commands to. The model also includes a time dimension so that past locations can be referred to [68].

A number of research projects have focused on the use of additional information from other interfaces to improve the accuracy of the speech interface, for example the robot Godot [164] uses bidirectional information transfer to resolve ambiguous voice navigation commands using navigation data, and navigation uncertainty using further voice information. Yoshizaki *et al* [180] use a combination of speech and vision to make recognition more robust while Nakauchi *et al* [108] use a human dialogue model that includes assertion of the command to be undertaken by the robot, allowing for feedback to occur as part of a naturally flowing conversation.

A number of other robot platforms use speech as an interface including; ASKA [118], SIG [107, 120, 121], Robota [16] and RoboX [133, 155]. RoboX uses situational reasoning to determine the appropriate response to an approaching user, for example greeting them on their initial approach [73].

Gaze

There has been recent interest in using eye gaze direction to directly control robots. This is particularly useful for users with limited mobility, such as the elderly or disabled. Yoo *et al* employed the use of infra-red LEDs to estimate gaze direction for robot control by elderly people without accurate arm control [179] and Law *et al* used electrodes to measure gaze direction for the control of an electric wheelchair [87].

Virtual and Augmented Reality

Virtual reality (VR) has been used for a range of teleoperated environments and off-line task programming systems and also for simulation and verification. An algorithm evaluation system is presented by Shikata *et al* [154], which uses robots in a VR world to test the performance of avoidance algorithms when human users are involved. The main advantages are that it uses a real human rather than a scripted one, thus giving more accurate behaviour while at the same time using virtual robots so there are no safety issues.

AR uses virtual objects projected into the user's view of the real-world, generally utilising some form of head mounted display. Currently AR has only been used for a small range of tasks, however as the technologies associated with it mature it will undoubtedly be more fully utilised. AR provides an important link between the traditional computer based interfaces and the necessity for mobility for interaction with many robots. AR is discussed in more detail in section 2.4.

Imitation

Riley *et al* [141] used visual tracking of coloured dots on a human user to imitate their movements with a full body humanoid robot. Inverse kinematics were employed to generate the movements in the robot. Pollard *et al* [129] presented a system for full body imitation that attempts to maintain the individuality of the performance within the limited movement space of the robot, this is particularly relevant for programming performances for entertainment robots.

A system for imitating arm gestures is presented by Cabido-Lopes and Santos-Victor [30]; it uses monocular vision with no special markers to track the arm, and the gestures can be transformed back to the robot's view point, rather than using a direct mirroring approach.

Related work by Tatini [162] demonstrated a technique for reducing the dimensionality of the control signals needed to reproduce full humanoid motion to a level where manual control is practical for basic movements. A neural network approach is used to reduce the dimensionality from 20 DOF to 3 inputs while maintaining reasonable flexibility of motion.

Billard *et al* looked at the problem of deciding what to imitate [15], specifically their system examined repeated manipulations and selected the invariant aspects, which it then reproduced.

A novel imitation system is described by Kosuge *et al* [81]; the result of the research was a wheeled robotic dance partner which taught users how to dance a waltz by moving in the correct steps, thus allowing the human to imitate the robot's movements.

2.2.5 Indirect Interfaces

Emotion and Personality

Emotion and personality play an important role in making a robot socially acceptable, particularly in long term interactions. If a robot's behaviour is predictable then social interaction quickly becomes boring.

The WE-3 and WE-4 projects used a robot head equipped with a range of sensors to investigate the effect of personality on the quality of interaction [101,102]. Kobayashi *et al* have created a robot with a highly realistic human face. The research investigated emotion representation, and reported recognition rates of over 90% for 6 different emotions [77].

The robot SIG used an internal personality representation to direct its attention in multi-user interactions. Both audio and visual data are used for the localisation of the users [120].

The small stylised humanoid robot CERO enhances a simple personal assistant robot and gives it an interface with limited emotion and personality. Even with CERO's 4 DOF control and stylised form, users were able to relate much more naturally to it than to the purely functional robot used in initial trials [151].

Hashimoto presented a fuzzy logic model for combining knowledge, intention and emotion in an assistant robot for elderly and disabled people [65].

Emotion or affective state is also a useful input from the user. Rani *et al* [137] used biofeedback sensors to present direct information about the user's affective state to the robot. This allows the robot to respond appropriately, for example when the user is fatigued or stressed.

Breazeal's robots Kismet and Leonardo [26] both used emotion systems that are based on human emotion models in order to enhance interaction with human users.

Facial Features

Facial features are an important way of communicating internal state to users. There has been related research in other fields such as dramatic acting and psychology on ways to describe facial expression. For example several robotic systems have adapted the facial action coding (FAC) system [54, 78].

Ishibiki *et al* used a stylised human avatar in place of a traditional phone, the facial expression and body language of the avatar was deduced from the electrical properties of the human voice signal. Experiments showed that users would talk to the robot rather than directly to the remote user, and that the visual cues were particularly good for turn taking. Breazeal also studied turn taking behaviour with the robot Kismet [25].

Feelix has a 4 DOF face that allows it to present basic facial emotions [31]. Emotion recognition by users in response to Feelix was reasonable, but not as high as when using

a human actor. The WE-3RIV Robot uses facial colouring to enhance the expression of emotion [103].

Eye movements have significant communicative value in human interaction, and are important for natural communication between humans and robots. Gaze direction indicates attention on the subject and gives context to other communication by glancing at the object of discussion [23, 24].

Body Language & Gesture

To be socially accepted a robot must be able to both interpret and produce body language. The robot's physical form must be acceptable to users, generally recognisable forms are more acceptable than a purely abstract presence.

Kismet gives simple head position cues to communicate that the human is too close or too far for successful image processing. If an object is too close Kismet moves its head back, which both increases the interaction distance and gives the social cue of invaded space, suggesting the interaction partner back off. Similarly, if the object of interaction is too far away Kismet will crane its head forward [23, 24].

A study by Shibuya on human gestures and emotion concluded that the way a gesture was performed varied under different emotion conditions; for example, gestures such as clapping were faster when a pleasant feeling was present [153]. This information could be used to assign priorities to tasks, and also to monitor the user's emotional state such as stress and fatigue.

Emphasis given by gestures is also important as an output method and research by Tsuji *et al* [169] aims to model human arm movements, in order to provide more natural arm movement by humanoid robots, including non-holonomic movement tasks.

Intonation

As well as conveying direct spoken commands speech communicates social and cultural aspects, as well as emotion in intonation. Kismet uses the tone of the voice to convey emotional state [22]. Nishikawa *et al* used a physical model of the human vocal system to produce more natural speech synthesis [117], the project has the dual goals of producing more natural speech and increasing the understanding of the human vocal control system.

Demonstration (Implicit Communication)

Implicit communication is useful when the robot does not have the capacity for direct communication. Nicolescu and Mataric [114] implemented a mobile robot that requests help by demonstrating failed task attempts in the presence of a user.

Haptic

Haptic interfaces are generally used to provide a feedback path to the human operator in teleoperated environments. They are particularly important for tasks that involve precision movement and grasping. Haptic interfaces have huge potential in medical applications, where subtle changes in texture can be very important to the surgeon during diagnosis and surgery, for example when removing tumours [135].

Wosch and Feiten describe the implementation of a tactile interface for controlling an 8 DOF robot arm through natural push and pull commands along with a more extended alphabet of tactile commands [177].

Haptic interfaces are useful for remote activities as they transmit information about the physical environment to the remote operator. Lo *et al* [90] describe a system for capturing the tactile contact of a robot finger and displaying it on a tactile display so that a remote user can sense what the robot is touching. Balijepalli and Kesavadas [10] use a haptic display to assist in the planning of a grinding and polishing task. A 3D display combined with haptic feedback is used by Sano *et al* [143] as an interface to a teleoperated system.

Lin *et al* describes the use of haptic interfaces to enhance the use of computers in "creative processes" [89] by sensing and displaying attributes such as stroke pressure.

2.2.6 Multi Modal Communication

In order to achieve effective natural interaction between humans and robots several of the above interfaces need to be used in a single robot, as they are in human to human communications.

The ETL-Humanoid robot design [34] is based on the principle that the complete range of human interface techniques is needed if natural humanoid interaction is to occur, and additionally, if the full range of human capabilities are integrated then the interaction may be robust and seamless as there are redundant communication pathways. Similarly the robot CoRA [70] uses a wide range of anthropomorphic interfaces including touch, vision, gaze direction and gesture as both input and output interfaces to assist in natural interaction with human users when cooperating on an assembly task. Both Kismet and Leonardo [23, 24, 26] make use of several interface media in order to provide more natural and enjoyable interactions.

Yoshizaki *et al* used multiple interfaces to provide redundancy, where speech was used as a fall back interface if visual identification failed [181]. The ISAC system [148] uses both sound and infrared sensors to enhance tracking of human interaction targets in its vicinity. Ghidary *et al* used gestures and natural language in a system designed to assist an autonomous robot with mapping of an unknown region [55].

Konyo *et al* [79] investigated the combined effect of tactile and visual feedback. Their research showed that while visual information dominated the tactile feedback the tactile information still had an effect on the perceived material. Szemes *et al* [160] used audio, visual and haptic interfaces for a robot guide for the blind which is equipped for guiding the user through crowded urban environments.

A system for developing natural interaction for programming by demonstration is developed in [96]. The system mainly utilises speech and visual data to achieve a natural dialogue between the user and the robot in a small domain of objects in front of the robot.

2.2.7 Survey of Interaction Evaluation

In robotics research the most common method of evaluation is to implement a prototype system and then measure its performance, this is also the case in the more specific area of HRI. The challenge of evaluating HRI systems lies in how to measure the performance of the implemented system.

This is not intended to be an exhaustive survey of usability evaluation techniques, rather it presents those approaches that are currently in use within the HRI community. Nielsen provides excellent coverage of the issues of usability [115] and Ziegler [184] gives a more concise summary of techniques. This thesis concerns itself prodominantly with the understanding of the robot's internal state and world model. This is similar to the idea of situational awareness except with an extra level of indirection as the developer must understand the robot's task as well as their own.

Situational awareness in robotics is evaluated through user studies where the awareness level is usually determined in one of two ways; coding the user's performance for critical events [146] or by questioning the user [149]; often the user is interrupted mid task and asked to describe the current situation. Both of these methods analyse the performance of a small number of users in great detail which has the advantages of only requiring a small user base, and being flexible in the tasks used for the trials. The results obtained are generally subjective and the same person is needed to code events for all trials if there is to be valid comparison, or a number of reference trials are needed to compare different coders. These two approaches are well suited for identifying deficiencies in a system, or attempting to explain externally identified differences between two systems, but are not as well suited for performance comparisons.

All areas of user interaction research have made use of user trials where a carefully defined and controlled task is undertaken by a number of users and performance metrics such as speed of completion and number of errors are recorded. These trials can then be used for comparing the performance of different interaction systems for the given task. There are many difficulties with these trials; a relatively complete system or mock up system is needed, representative tasks are often difficult to create and a large number

of users are needed to get results which aren't biased by variation in individual users. These trials are often combined with analysis of recorded footage in order to gain as much information as possible from the intensive process of user trials.

As an alternative to user studies, heuristic evaluation has been used extensively in HCI [115], particularly in relation to 2D WIMP (Windows, Icons, Menus and Pointing device) interfaces but has also been extended to 3D interaction [20]. Heuristic evaluation applies a set of guidelines defined by usability experts to the application under evaluation. This approach works best in areas where the performance of similar systems can be examined in order to determine the heuristics, given the relative youth of the field of HRI and its applications finding appropriate heuristics can often be impractical.

A survey of the inaugural human-robot interaction conference, HRI2006 [1], shows that user trials are used almost exclusively for evaluation of HRI research, for example [45, 51, 57, 80, 106, 134, 145, 157, 161]. The major challenges that researchers face are; finding appropriate participants, determining metrics, achieving statistical significance and dealing with confounding factors (such as the novelty of interacting with a robot).

One of the challenges of finding appropriate metrics is finding metrics that can be compared between different projects. Steinfeld *et al* [158] have begun the task of documenting a set of metrics that will have some commonality between studies.

Given all of the challenges facing researchers in getting meaningful quantitative data from user studies, often the most valuable results are gained through subjective analysis of the task performances. Case studies with a prototype system will be used for qualitative evaluation in this work, this is discussed in more detail in chapters 4 and 6.

2.3 Visualisation

There is a wide range of texts that give good coverage of the topic of visualisation, interested readers are directed to the books and papers cited below for further detail. There are two areas that this coverage can be broken into; scientific visualisation and information visualisation. These two areas will be further explored below.

Excellent coverage of the history of visualisation as well as basic techniques for ensuring accurate portrayal of data are given in the works of Tufte [170–172] and Wainer [175]. Tufte's emphasis is on the production of elegant graphics that convey the maximum information with the minimum effort on the viewer's part. While Tufte's works are focused almost entirely on the production of data visualisations for the printed page, many of the fundamental principles are equally valid with modern on-screen visualisation.

Information visualisation focuses on presenting data that does not naturally have physical representations, such as network topologies. The emphasis is on extracting structure from these datasets and presenting it to the user. A thorough technical treatment of

information visualisation is collected in *Information Visualization* [33].

Scientific visualisation on the other hand focuses on presenting the wealth of data generated through scientific research and experiment. Here the focus is on highlighting the key features of what are often large datasets, potentially with high orders of dimensionality. *The Visualization Handbook* [64] collects together a comprehensive collection of papers on the subject and readers are encouraged to explore this if they have further interest.

While scientific and information visualisation have some key differences in the source of their data and some of the standard representations, they both have to work within the limitations of the human perceptual system when presenting their output, and hence there is a significant overlap between the two domains. Grinstein *et al* has an important collection of papers on the perceptual issues in visualisation [59] and the reader is again directed to *The Visualization Handbook* for further papers on the topic [64].

Many robot data sets have simple physical representations making it straightforward to create visualisations for them. This shifts the focus from how to visualise a single robot sensor to the more difficult challenge of how to effectively visualise combinations of different robot data sets and how to create context for the data sets aiding in human understanding.

Keim [76] gives a concise summary of the different categories that data can fall into specifically; nominal, ordered and metric. In general a visualisation for a particular data type is more likely to be relevant for another data set of the same type, for example a visualisation for an ordered sequence of path elements is likely to be simple to adapt to an ordered sequence of arm movements, possibly with custom renderings for the individual set elements.

The general conclusions of these works is that while general guidelines can be provided there is no "best" visualisation possible, each application will suit different techniques and the best approach is to have a broad knowledge of techniques that have been applied in the past and to match these to the target dataset. Within the field of robotics, applications are far too varied to apply strict design rules and so a visualisation system needs to support a range of techniques, and in particular support the developer in creating their own visualisations for data.

Several works have been published looking to the future of visualisation, for example [66, 74, 95]. Johnson's work [74] is the most recent citing 15 top research problems for scientific visualisation. His 15 challenges, in particular the last stating the need for a theory of visualisation, emphasise the difficulties of generalising findings from visualisation research. This challenge is increased further in a relatively new field such as AR and in a novel domain such as robotics.

2.4 Augmented Reality

AR involves the realtime rendering of virtual elements at geometrically registered positions within a real scene [8, 9, 20]. Bowman provides a good introduction to AR in *3D User Interfaces* [20] and Azuma provides a detailed survey of the state of the art in AR [9] with recent advances presented by Azuma *et al* [8]. An in depth treatment of AR is given in *Spatial Augmented Reality* [17]. This text takes a practical look at AR focusing on the details of implementing an AR system on a modern PC.

Often AR implies some sort of immersive system where the user wears a head mounted display (HMD) which renders the virtual elements, however AR can also be effective when rendered on a standard desktop display or using projectors to create virtual information in the environment.

AR techniques can thus be divided into immersive and desktop configurations, with the immersive techniques being again divided three ways into video see-through head mounted systems, optical see-through head mounted systems and projection based systems [9]. These different configurations will be discussed in the following paragraphs.

2.4.1 Immersive AR - Video See-Through

AR using a video see-through head mounted display is the "classic" AR configuration. In this setup the user wears a head mounted device that is equipped with both a camera for capturing the real-world frame and a display for presenting the augmented view to the user. More advanced head mounted systems can have dual displays and/or dual cameras to give stereo capabilities however the same fundamental principles apply whether the system is stereo or mono.

The basic AR rendering pipeline for a video see-through system is shown in Figure 2.5 and outlined below:

- Capture the real-world frame;
- Process the frame to extract fiducials that mark positions to render at;
- Render the virtual elements into the frame;
- Display the frame to the user.

Figure 2.6 shows an example AR output from the ARDev system which is described in Chapter 5.

Variations on this process include external tracking of the user, which allows for static or tracked virtual elements to be rendered without the need for fiducials. In outdoor environments often GPS is used to track users [35, 165] while in indoor environments a

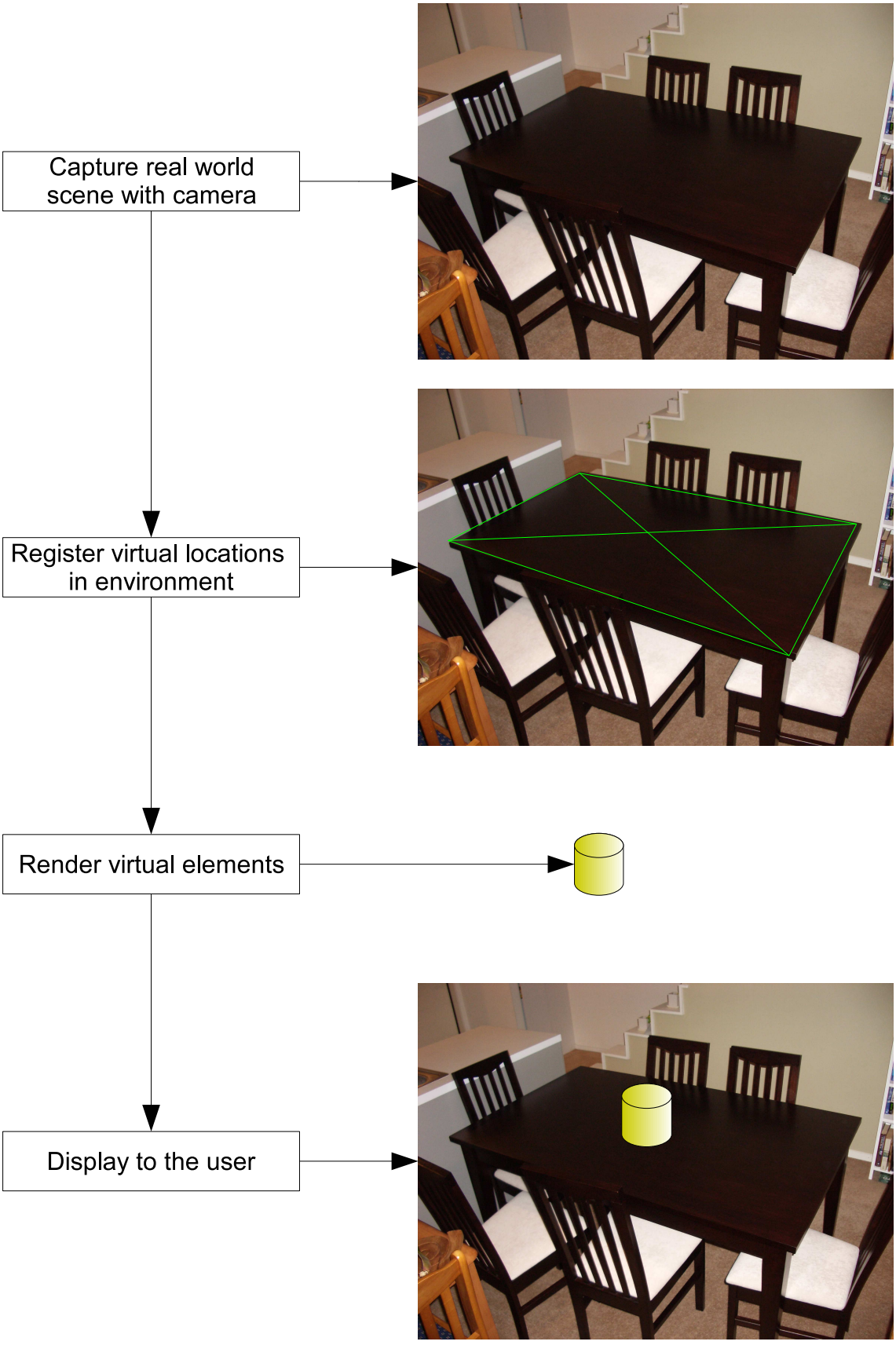


Figure 2.5: Augmented Reality Process

Figure 2.6: Example Augmented Reality Output

number of different tracking technologies are available including; inertial [69], ultrasonic [69], magnetic [6], motion capture [173] and hybrid approaches [69].

The disadvantages of a video see-through HMD are the low resolution of the real world scene combined with a restricted field of view. There is also a certain latency between events in the real-world and the output on the display. This has a range of effects including discomfort for the user, difficulty in performing tasks requiring visual feedback and safety issues. Additionally there is often a physical offset between the cameras and the user's eyes which provides an unrealistic displaced view of the real world scene.

2.4.2 Immersive AR - Optical See-Through

The setup of an optical see-through head mounted display is similar to the video see-through configuration. The key difference is that instead of grabbing the real-world frame with a camera the display is designed in such a way that the user can see through the augmented elements to the real world beyond. While the optical systems do not require a camera for their basic functionality in practice one is often added to perform fiducial tracking tasks.

While optical see-through displays avoid the constraints of real-world resolution and field of view they are generally more restricted in these aspects when considering the virtual elements. In addition to this the latency in rendering of the virtual elements is far more obvious in an optical see-through system as the real world is perceived without latency causing virtual elements to lag behind real-world objects when the user's view is moving.

Optical see-through systems are considered to be safer for the user as in the worst case where power is lost to the system the user retains a view of the world. In video

see-through systems power loss or system failure effectively blinds the user.

2.4.3 Immersive AR - Projected Systems

There are a number of AR systems investigating projection based AR. Projection based AR creates the virtual elements in the real-world by projecting onto appropriate surfaces. This limits the type and location of virtual elements but has some other significant advantages:

- The user needs no hardware to view the AR scene, it is already embedded in the environment;
- Multiple users can interact together;
- By restricting the places that virtual elements can be, in general fiducial tracking is avoided.

Projection based systems are most suited to small constrained objects such as an augmented desktop [139] or wheel chair controller [138].

A significant challenge with projection systems is finding ways to project information where the user interaction does not interfere with the projection. This is not such an issue where back projection is suitable.

2.4.4 Desktop AR

Desktop AR configurations present the augmented view to the user through an external display. Such displays can include a standard desktop PC, wall mounted screens and mobile devices. The main disadvantage of this approach is the lack of immersion, the feeling of the virtual elements merging with the real-world.

There are however some significant benefits of using a desktop AR configuration:

- Lower complexity and cost;
- Reduced cyber sickness effects;
- No cumbersome hardware needed;
- Team interaction is easier;
- Integrates well with desktop computing tasks and tools;
- Can be used for remote viewing;
- Simple interaction techniques are easier to use, such as traditional pointing devices and touchscreens.

2.4.5 Current directions in AR

Current research in AR has a focus on applications in the real world and improving the base technologies to support this. These technology goals include increasing display speed and resolution, reducing weight, extracting fiducials from the environment, improving tracking techniques and outdoor AR technologies. Target application areas include; mechanical servicing in the form of interactive service manuals, tangible interfaces, in particular for communication and collaboration, and ubiquitous information presentation.

Several AR software toolkits have also been gaining popularity recently. These toolkits allow for the rapid creation of simple AR systems. Three of the notable toolkits are ARToolkit [5], ARToolkitPlus [63] and ARTag [4]. All three of these toolkits utilise fiducial markers to register real world locations for 3D rendering. Alternative tracking techniques include the work of Neumann *et al* [112] on natural feature tracking. The use of natural features allows AR to work in environments that have not been prepared in advance making it ideal for use in the field for service tasks.

2.4.6 The Application of AR to Robotics

AR has been used to display robot data in a number of projects. The purpose of the AR visualisation is to either enhance understanding of the task and the environment or for feedback to the user during operation.

An AR system for monitoring robot swarms is presented by Daily *et al* [44]. Daily displays small pieces of information from each robot together, allowing easy interpretation of the large amount of data from the swarm. For example Figure 2.7 shows a group of robots searching for a trapped person. Once the person is located by a robot the message is sent along the chain of robots, the visualisation then shows an arrow indicating the direction of the next robot in the chain creating a vector field directing the user to the trapped person.

Similarly Amstutz and Fagg [2] describe the implementation of a system for representing the combined data of a large number of sensors on multiple mobile platforms (potentially robots) using augmented and virtual reality to display the information. Freund *et al* [52] make use of an AR display in order to more simply display complex state information about autonomous systems such as a processing facility.

Recently AR was also used for debugging purposes while testing a motion planner for a humanoid robot [36,159]. A motion capture system was used to track all the objects in the environment both for the AR rendering and for the motion planning. The visualisation, shown in Figure 2.8, is used for debugging the motion planning algorithm showing the robot's planned foot placements in context with the obstacles that it is avoiding. The AR system in this case is very specific to the programming task being undertaken.

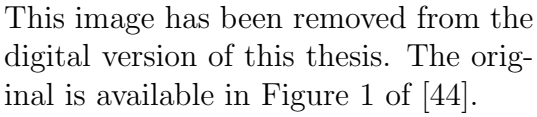


Figure 2.7: AR arrows indicating object of interest with small group of robots [44].

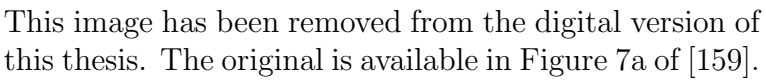


Figure 2.8: AR rendering of planned foot placements for the HRP-2 humanoid robot [159].

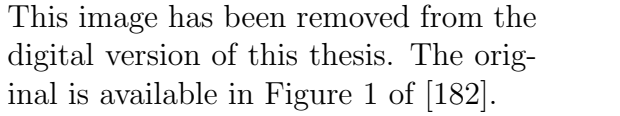


Figure 2.9: AR Bubblegram showing the current state of the Aibo robot [182].

Work on conveying the robot's internal state for users has been carried out by Young *et al* [182]. In this work AR Bubblegrams are placed adjacent to an Aibo robot displaying the current state of the robot and any information items it wishes to convey to the user. An example Bubblegram is shown in Figure 2.9.

Many of the AR applications for robotics have focused on training scenarios. Recently KUKA have begun to investigate the use of AR to help visualise data during training providing immediate feedback to operators when they instruct the robot incorrectly [18]. Figure 2.10 shows the AR visualisation of a robot arm during a training session, the AR elements highlight the orientation of the arm's controllable axes. Raghavan *et al* [136] describe an interactive tool for augmenting the real scene during mechanical assembly. Pettersen *et al* [124] 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* [29] describe an AR system that integrates graphical and sensory information for remotely controlling a vehicle. Milgram *et al* used AR to create a virtual measuring tape and a virtual tether to assist peg-in-hole insertion tasks [98].

From this work it can been seen that AR has great potential as a tool for enhancing robotics and is a suitable interface for a robot development tool, indeed AR has been used in an ad-hoc manner by Stilman *et al* [159] to examine robot data and aid in debugging. This thesis builds upon this idea, creating a formal argument for the use of AR by robot developers.

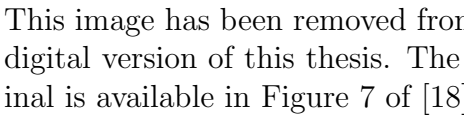


Figure 2.10: AR rendering of the active axes on a KUKA robotic arm [18].

2.5 Summary

The examination of the current state of the art in robot programming shows a lack of targeted tools for the robot developer. Robot programming has a unique combination of challenges which the tools inherited from software engineering fail to meet. While significant progress is being made in the areas of programmer libraries and frameworks other robot tools, such as data viewers, are largely created by developers to meet their minimum needs of the moment. Far from being powerful aids to the developer they are fragmented projects with limited functionality.

This work aims to build on existing research from the HRI community to target the needs of the robot developer while they interact with the robot systems they are creating. The key to this interaction is understanding the robot's world-view; effective visualisation is needed to enhance developer efficiency in the difficult and time consuming task of debugging.

Debugging robot applications involves the comprehension of large amounts of data, to achieve this we need to utilise the human mind's natural ability to process and filter large amounts of visual information. Of the possibilities for visual data display analysed in Section 2.2.7, AR offers the greatest opportunity having the unique capability of rendering data in context with the real-world environment. This work applies the unique benefits of AR to the domain of developer-robot interaction, designing an AR system for robot developers and evaluating it through a prototype implementation.

3

Augmented Reality for Robot Developers

In Section 2.1 the current state of robot programming was presented. Current robot development is carried out mainly with tools from the software engineering domain. While this is appropriate for some aspects of robot development there are a number of unique challenges that are not targeted by these tools. The current approach is somewhat like using a paintbrush to paint a large wall; it will perform the job, and if there are only a few walls to paint it may not be worth the effort of getting a roller, but in the long run a tool tailored for the job is far better. When considering robot development it is now at the start of a long era of development so it is worth getting the tools right.

The examination of the full tool chain needed for robot developers in the previous chapter showed that debugging tools have had the least formal work. There are a number of tools that have been created out of necessity such as the visualisation tools in Player/Stage [127]. However these were designed to meet the need of the moment and are not optimal or complete for the full task of robot programming.

By approaching the design of a debugging tool for robot developers as an HRI issue the needs of the general robot developer are able to be met, as opposed to creating isolated application features as the need arises. This chapter begins with a definition of the problem domain, introducing the role of the robot developer while interacting with the robot. An examination of the developer role is then presented showing the ideal match

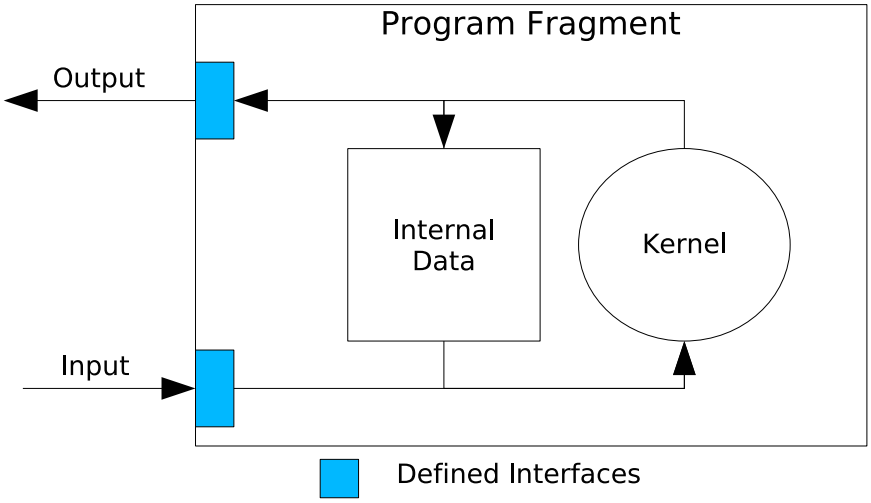


Figure 3.1: General Robot Program Fragment

between the properties of an AR interface and the needs of the robot developer.

3.1 Model of a robot program

In order to examine robot development a model of a robot program is needed. The model, or program fragment, shown in Figure 3.1 consists of three key parts: the algorithmic kernel, the internal program data and the external input and output interfaces. The input and output communications use defined interfaces generally in the form of a library API or network protocol. Throughout the rest of this work the term program will be used interchangeably for both a complete program and a program fragment.

The internal data represents both the current and historic data that has been explicitly stored in the program. This represents both the standard interface data that is available to the world and also the intermediate data of the program that is used to generate the externally visible elements (including stored sensor data, reliabilities and activation levels as well as internal variables, temporary variables and global data). The kernel represents the traditional algorithmic portions of a robot program such as; execution point, concurrency, algorithms, breakpoints, watch points and program traces.

In the case of a robot application designed to follow a human user, for example an assistant carrying a tool for the user, the input would consist of readings from some external sensor such as a laser scanner (the *laser* interface in Player) and the output would consist of commands to the underlying motion actuators (the *position2d* interface in Player). The kernel would be responsible for processing the incoming input data, storing the relevant portions of this in the internal data and then generating appropriate output commands to the actuators.

The model in Figure 3.1 is designed to be as general as possible allowing the explo-

ration of the different possibilities for visualising a robot program. The simple separation between data and kernel allows the kernel component to represent any programming model such as petri nets or finite state machines. One benefit of this general model is that it focuses on the data aspects of the robot program, which is one of the greatest challenges for the robot developer.

3.2 Characterising Robot Data

Robot data can be characterised in a number of different ways. Here the focus is on distinctions that alter the way the data is viewed, identifying commonalities in order to allow generalisation of data visualisation techniques.

In order to characterise robot data it is important to know what data types are in common usage in the research field. The Player Project [127] represents a significant user community in robotics research and so is a good basis point for examining robot data types. Table 3.1 contains a list of the data types that were in use in the Player server interface definitions (as encapsulated in player.h) as at the version 2.1 release.

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 3.1: Data types used in Player interfaces. Type field indicates whether the data is (S)calar, (G)eometric, (V)ector or (A)bstract. The spatial column indicates whether the data represents a value related to 3 dimensional space. The Sequence column indicates whether the data is inherently a sequence of data.

The data in the table has three properties assigned to it, a data type, whether the data is spatial and whether it is inherently a sequence. The data type indicates if the data is a scalar, vector, geometric or abstract type.

In examining this data set from the point of view of visualisation some important groupings are able to be made. The first and most important distinction is whether the data has an intrinsic visual representation. All geometric types are able to be rendered directly into the environment. Similarly spatial scalar and vector data can be rendered relative to some known reference frame. For acceleration and velocity this is the robot frame of reference. For a scalar quantity such as a measured range this would be relative to the transducer origin and along the axis of measurement, for example rendering the measurement from an ultrasonic sensor.

Data that is non spatial needs a spatial model before it can be rendered. This can be achieved easily in some cases, such as using coloured bars to indicate battery level or light intensity. Completely abstract data such as bit fields or binary data generally need to be rendered at a higher level of abstraction to be easily understandable. 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 such as the model's colour. Another alternative for non spatial data is to use a textual representation.

Some data sets are inherently sequential such as a laser scan. Most other data sets such can be meaningfully used as a sequence; either of ordered objects such as a sequence of waypoints, or as a temporal sequence such as the odometry history of a robot. The rendering of 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 it makes things more difficult as the items in the sequence interfere with each other, such would be the case with historic range readings.

Stochastic properties of data are also important to represent in many cases, for example the uncertainty of a measured value. In general properties such as uncertainty or probability add extra dimensions to the data to be represented. In some cases, such as a point location, uncertainty can be easy rendered by expanding the object to an extra dimension and representing it as an ellipse or polygon. In cases where objects are already three dimensional this data needs to be represented using another property such as colour or transparency.

An attribute of data that is not highlighted from this list, but is potentially interesting for visualisation is data that represents a causal link between two other pieces of data, i.e. between a condition and the action that is dependent on it. The term conditional data will be used to describe this. An example would be a visualisation that showed the connection between an active region in front of the robot and the motion vector that would result if an object entered this region.

To summarise, data can be either *spatial* or *abstract* and can also be any of the following in nature; *sequential*, *stochastic* or *conditional*.

3.3 Defining the Robot Developer

Sholtz's division of user roles during interaction includes six roles; supervisor, operator, teammate, peer, bystander and mechanic [144]. The key role that is missing in this division is that of the developer [40].

The developer role shares some attributes with the mechanic, in particular both must be able to determine what is causing flaws in the robot behaviour. The key difference is that the developer needs to understand and find flaws both in existing functionality of the robot and while creating new functionality. This means the developer needs to understand basic faults in the robot platform, coding errors in new and existing code and also any flaws in underlying assumptions being made about the platform, task or environment the robot is operating in.

The design and coding phases of robot development are performed offline, that is to say the interaction during these phases is with a programming environment rather than with the robot system. This means that in terms of interaction with the robot the developer

role can be restricted to the task of testing and debugging a robot application.

The purpose of these two phases is to detect and then identify the cause of any errors in the system. From this purpose we can see that an understanding of what the robot is sensing and doing, and specifically what it is sensing and doing incorrectly, is key to robot development, and this is precisely what the unique properties of AR enable, making it the ideal interface to aid the developer.

3.4 Potential Benefits of Augmented Reality

The major potential benefits of AR are threefold. The first is that by providing a 3D visualisation of robot data sets the developer is able to understand complex data in an intuitive way. These data sets include those that have a simple translation to geometric data, such as a laser scan, and also more abstract data sets that have a physical metaphor, such as representing the robot's emotional state or health as a coloured halo.

The second benefit of AR visualisation is the combination of the dataset with the real world ground truth. This powerful combination allows the developer to quickly identify datasets that are inconsistent with the real world values they are intended to represent. This can be used with both base sensor data, such as an ultrasonic ranger, and also with higher level data sets such as the estimated location of an object being tracked.

By placing robot data in context with the real world the developer is able to identify the type of error that is occurring, specifically whether it is an error with the underlying platform (a faulty sensor), a coding error (the robot turning right when the algorithm should direct it left) or an algorithm error (invalid underlying assumptions with regard to the environment, task or robot platform).

By identifying the type of error, the developer is able to appropriately direct their development effort. For example, a fundamental limitation of a hardware sensor can cause many problems that are often attributed to software bugs. If a developer is able to easily identify the cause of the issue then development effort could be directed to either enhancing or replacing the sensor in question or using a more robust algorithm that is able to cope with the data errors.

The third benefit of AR visualisation is that it provides a stepping stone from simulation to real-world robot development. One of the key benefits of using a simulation environment for robot programming is the ability to examine and visualise internal data that is otherwise not available. In the case of robot simulation, the developer is able to view the robot's sensors in parallel with the simulated world that the value represents. An AR system allows the data available in the simulated environment to be presented when working with robots in the real world.

This benefit should not be underestimated. If a program has been well tested in a

simulated environment then the only bugs that will remain relate to discrepancies between the simulation model and the real world. The best way to highlight these errors is by displaying the real world sensor data in context with the real world; precisely what AR allows.

Of these three potential benefits, only the first can be provided without a real world baseline to compare with. Thus by utilising AR technology we gain benefits that are unable to be provided by any other visualisation system.

AR has enormous potential to aid the robot developer. Programming of any kind is a complicated process and in many ways even more so for robot development. It is difficult to isolate small tasks within the programming domain and test these individually without losing the complex interactions which are the core challenge of the programming task. Given these difficulties the best approach to evaluating the potential of AR for developers is to implement a prototype AR system and examine its use in robot programming tasks.

The following chapter looks at what functionality a theoretical AR system for robot developers should have and how interaction with it could occur. The implementation of this and the performance in case study tasks are reported in chapters 5 and 6, showing that when used in a real-world environment AR does provide significant benefits for developers.

4

A Conceptual Augmented Reality System

This chapter describes a conceptual AR system for robot developers. By considering a hypothetical implementation of an AR system, a set of requirements for the visualisation system can be developed. Chapter 5 describes an implementation of a portion of these requirements which is used for further investigation of AR for robot developers.

Initially the options for the AR system are discussed and following this, two modes of operation are presented along with their requirements, and an examination of the benefits they provide to the robot developer.

4.1 The AR design space

There are a number of questions to consider when designing an AR visualisation system for developers:

- What data should be displayed?
- How should the data be represented?
- Where should the data be displayed?
- How should the data be viewed?

4.1.1 What data should be displayed?

The most important program information to display is the variables that are directly 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. The visualisations of this data are 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 some platform failure.

Robot development, like most software development is undertaken in two stages, library development and application development, and different visualisations are required in each case.

The library developer must focus on the low level details of the library function that is under development. The details of the internal workings of one or more code fragments need to be displayed, while details for "out of focus" components should be limited. There may be rapid changes to which components are relevant 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.

4.1.2 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.

Standardised visualisations promote code reuse and minimise the developer's workload in creating visualisations. However, standardised visualisations cannot be created 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 views available in modern graphical debuggers such as DDD [46] and Eclipse [47], with these debuggers standard visualisations of basic programming datatypes are provided to the user. Figure 4.1 shows a screenshot of the Eclipse debugger [47], presenting the structure of the Player laser proxy data. Figure 4.2 shows a screenshot of the DDD debugger's graphical visualisation for laser data [46]. 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

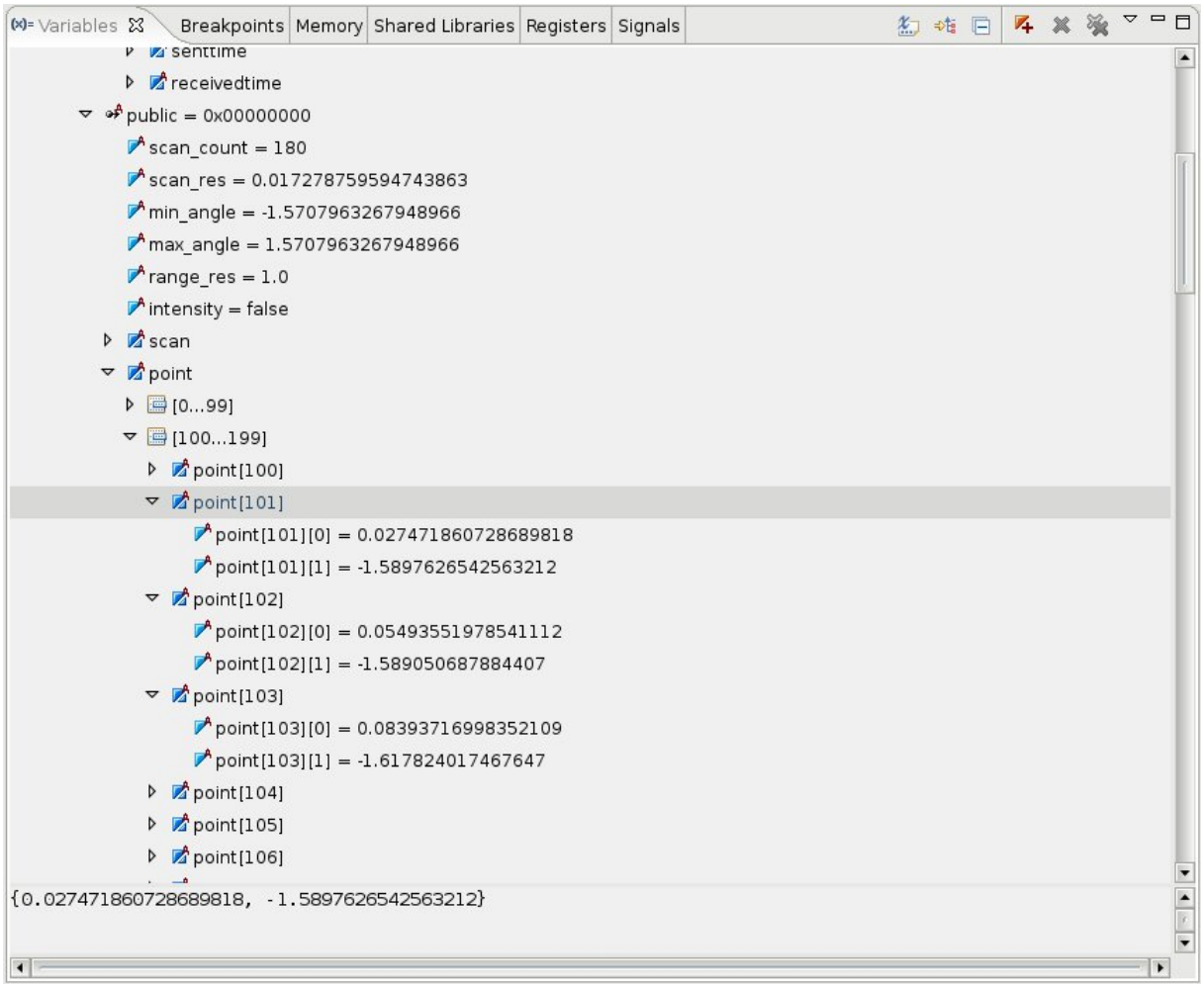


Figure 4.1: Eclipse IDE debug window

tree structures and colour, like those shown in Figures 4.1 and 4.2.

Custom visualisations are like the manual debugging output created when a programmer inserts print statements; the developer can concisely summarise printed information about execution in a particular piece of code. They know which items of data are most salient. Custom visualisations are most powerful if supporting classes and methods, including basic graphical building blocks, are available to aid the programmer in creating the renderings.

In order for standardised visualisation to be useful the program being visualised must have some form of known structure or 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. Many programs already meet this requirement as they make use of standard libraries and frameworks, such as Player/Stage.

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 data types, can have standard vi-

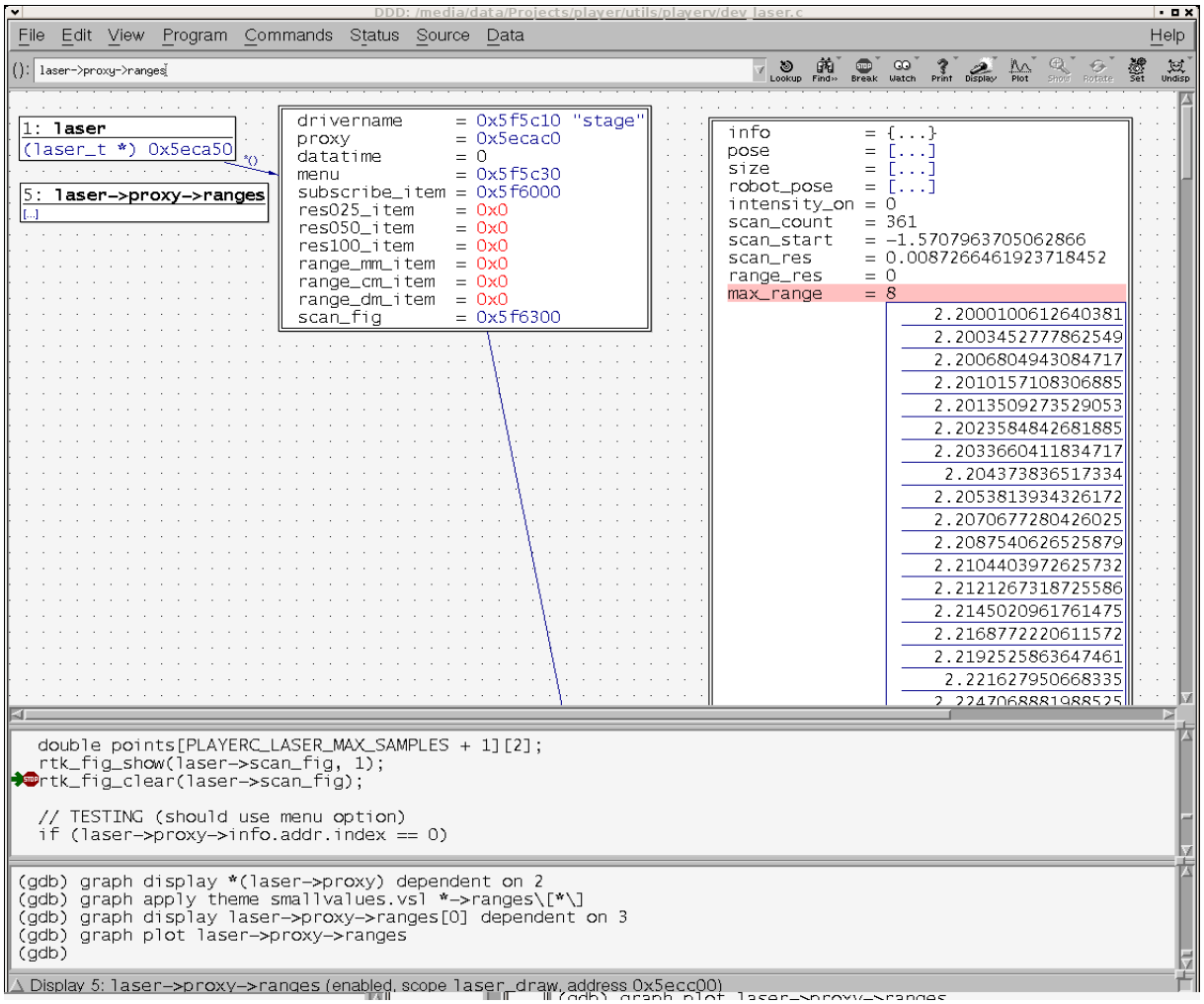


Figure 4.2: DDD visualisation of a list structure

sualisations written. However this method may not scale up to complex systems. The best visualisation for a complex system may not be the visualisation of all its components naively 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. 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 some other criteria. Thus it is difficult to provide scalable standardised visualisations for internal program structure. Meaningful groupings of the known data types need to be provided.

One way of easing the problem of visualising internal data is to make robot programs as stand-alone fragments that communicate through standard interfaces, for which there can be standard visualisations. 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 custom visualisations.

4.1.3 Where should the data be displayed?

Where the data is displayed in the developer's view has an important effect on how easy it is to understand. This is particularly the case when multiple data sets, potentially from multiple robots, are displayed together. The location of the rendering is 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 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.

Data without a direct physical 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 with the user ability to change this selection.

The location of data rendering should be handled much like the position of windows in modern window managers. Here the initial location is negotiated between the application and the window manager, and this can later be modified by user interactions that may move, re-size or hide windows.

4.1.4 How should the data be viewed?

There are a number of hardware possibilities for viewing an AR scene.

- The data may be viewed by an augmented, head mounted video display, by an optical see through head mounted display, by a projection based AR technique or 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;
- The developer's view may correspond to a static third person view, or a moving first person camera.

These methods have not previously been evaluated for robot developers, and it is not obvious which will be the most useful. It is likely that a number of factors will determine the best method, including 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 the AR system must be flexible enough to support as wide a range of AR configurations as possible.

4.1.5 Accessing the AR system and robot data

There are two questions to be answered here, the first is how the AR system accesses the robot program's data and the second is how the robot program accesses the AR system. The AR system can access the developer's data in two basic ways. The first is through direct access to the application's memory, generally by attaching to the process in the style of a debugger. The alternative is to use a *defined API* to access the data, through a network protocol in the case of a distributed system or by explicitly binding with the library.

The first technique is more complex technically but gives the AR system access to all of a program's data. However this access is not useful unless there is some sort of visualisation available for the internal data types. In a robotic system that has a well defined API it can be as effective to use this API or to tap the network stream in the case of a distributed application. The defined API technique requires only a single set of visualisations of the robot data, regardless of which client application is running on the robot. Direct access to the robot works well for systems such as Player/Stage which allow for multiple client applications to connect simultaneously to the robot through standard interfaces.

There are also two options for a robot application to access the AR system, in order to produce custom visualisations. The developer may access the AR system as a library that can be embedded in the application and thus access the visualisation system directly. Alternatively a network access protocol could be provided. The main issue is the potential performance difference in rendering to a remote system or local one. A secondary issue

is whether the AR system can be run on the same computer as the robot application. The network option will generally have lower performance but provides more flexibility in configuration.

4.2 Paradigms of an AR interaction system

In this section four paradigms of interaction between the developer and the visualisation system are defined which cover the range of interactions discussed in section 4.1. The four paradigms; direct library access, an intelligent debugging space, an AR enabled robot IDE and a whiteboard style AR rendering system, are not mutually exclusive and an implementation that supports all four interaction modes would be ideal.

4.2.1 Direct Library Access

For direct access the developer is provided with an AR toolkit that manages the visualisation system. The toolkit should provide a set of stock visualisations as well as the ability to directly instantiate custom visualisation objects.

In this mode of interaction the developer drives the whole system specifying what elements should be rendered and how they should be represented. This mode of access has the advantage of providing the greatest control over the rendering as the developer can be given direct access to the underlying graphics system. This flexibility comes with an added cost in terms of the effort required to create and manage the visualisations. Direct access can also be provided as a plug-in interface to a system such as the intelligent debugging space described below.

4.2.2 Intelligent Debugging Space

The intelligent debugging space (IDS) concept is a permanent installation that is capable of providing an augmented view of the robot development space. The AR view is captured from a fixed camera and then rendered on to a static display or displays in the development space providing a third person AR view.

Several independent users are able to use the space at the same time, since the activities of one user will not generally effect other users in the space. The level of independence largely depends on the display technology, if a single shared display is utilised then there will be some clutter from other users' displayed data.

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.

The presence of a robot is determined by two components; physical and operational. Physical presence is whether the robot (or its data) is visible in the AR field of view and whether the robot is able to be tracked. If a robot is tracked by visual fiducials, the robot can only be tracked when it is in the camera field of view. 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.

The requirements of an IDS are:

- The ability to track the physical presence of multiple robots;
- The ability to track the operational (i.e. network) presence of multiple robots;
- The ability to determine the functionality and configuration of the tracked robots;
- A set of stock visualisations for the robots, and the ability to connect directly to the robot without an intervening user application;
- A control interface that will give coarse grained control over what visualisations are present, and some basic configuration of the presentation of the data sets (such as colours);
- An AR display and the appropriate user tracking for the display. Multiple displays may be needed for a multiple user system.

The major benefit of the IDS is that the developer does not need to write any additional code in order to utilise the AR visualisations. This comes at the cost of reduced flexibility in terms of the choices of how to represent the data and also in the selection of AR hardware. Additionally the IDS is more complex to initially implement than a library approach, although this is a one time cost. Other benefits include increased continuity as the visualisation can run without the developer's application and ease of sharing the AR system with multiple developers.

4.2.3 AR enabled Robot IDE

In the Robot IDE case the user will have greater control over both what and to some extent how robot data is visualised. The IDE would allow the developer to select which interfaces and structures are 'watched' while the program is running. While the debugging space would visualise the robots and the robot data the IDE would allow for debugging of the specific program data.

The IDE could either make use of the IDS for displaying the AR data or have its own contained AR system. Most likely some sharing of resources would be needed in order to limit hardware costs and complexity.

The IDE would not need to track the presence of robots as it would be based around the timeline of the user's application, the IDE based visualisations would appear while the application was being debugged and then disappear when the application was terminated. Some form of logging of sessions could be beneficial allowing the AR data to be replayed at a later date.

An AR enabled IDE could also provide a script interface to the visualisation system allowing the developer to use a language such as Python to render data sets interactively.

An IDE based system is the most difficult to implement, effectively requiring some sort of IDS as a base. The advantage of the IDE system is it would integrate with the developer's existing tools, although it is difficult to quantify this advantage without implementing a prototype system to test with.

4.2.4 AR whiteboard

The AR whiteboard component allows the developer to render custom data into the AR space as a set of primitive 3D objects such as points, lines and polygons. This provides high levels of flexibility to the developer but also requires significant developer effort in specifying the visualisation.

The AR whiteboard could also utilise the IDS for the AR renderings. The custom renderings could be 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 could be displayed.

The key to implementing the whiteboard interface is minimising the developer's effort in creating custom renderings. An API for rendering simple shapes would aid the developer, with more complex renderings being built up from these. For example if objects of interest were being extracted from a laser scan the results of the extraction algorithm could be rendered along side the base laser rendering using the whiteboard programming interface.

The AR whiteboard system allows the developer to program their own renderings of their data while removing any need to control the AR system as a whole. this reduces the workload of the developer when compared to the direct library access system while keeping the flexibility of rendering. It still requires more work to create renderings than the IDS system.

4.3 Evaluation Methodology

How to evaluate HRI systems is still an open issue. In particular while almost all recent HRI research has undertaken user studies for evaluation, there has been great difficulty

in finding good metrics. Many studies rely purely on qualitative analysis. Section 2.2.7 gives more detail on the current state of the art in HRI evaluation.

Heuristic evaluation has been used to achieve good results in traditional HCI systems as described by Nielsen [115,116]. However one of the basic premises of heuristic evaluation is that a set of well known guidelines for the type of interface are available. For 3D user interfaces the guidelines are generally not available [21], this is even more of a problem with respect to AR which is a less mature and more specialised field than 3D interfaces in general.

The difficulty of evaluating developer interaction is magnified by the large variation between the performance of different programmers [132]. Given this variation, even if appropriate metrics are available, a large user base would be needed to perform valid comparisons. The small number of robot developers in any one location make it difficult and prohibitively expensive to find a suitable user base to carry out such trials.

Given these difficulties this work will use a qualitative analysis of an implemented AR system to evaluate the benefits of AR for robot developers. Specifically the implemented AR system was designed to support direct library access, IDS and AR Whiteboard approaches. The details of the implementation are given in Chapter 5.

The implementation is still in early stages of development and so much can be learned about its performance from relatively informal trials. More intensive trials will be suitable when the properties of the system are better understood. A series of case studies were evaluated in order to validate the ability of AR to enhance developer performance when interacting with robot systems, details of the tasks and results are reported in Chapter 6. Future work will perform detailed user studies to confirm the findings of these evaluations.

4.4 Summary

This chapter examined a hypothetical AR system for developers looking at how a developer could interface with the system as a development tool. Four usage paradigms were presented, which could be implemented together for a flexible system to meet the complete set of developer needs.

Given that analytical analysis is close to impossible for novel HRI techniques, evaluation of a subset of the proposed paradigms, specifically the intelligent debugging space, direct library access and AR whiteboard, has been carried out through case study developments with an implemented AR system, the system is described in Chapter 5 and the evaluation in Chapter 6. These three paradigms were chosen to give users of the system a good balance of overhead in using the basic features and renderings combined with flexibility to manually specify renderings when the stock visualisations were insufficient.

5

AR toolkit for Visualisation

This chapter describes the implementation of the AR system for developer visualisation of robot programs. The implementation provides three basic interfaces for the developer: direct library access, an intelligent debugging space (IDS) and an AR whiteboard [42]. These three interfaces provide complementary properties with respect to the amount of work the developer must do to utilise them and the flexibility they offer in rendering. Allowing the developer to select the level at which they wish to interact with the system. Additionally the three interfaces are complimentary in terms of their implementation with the IDS building upon the raw library interface and the whiteboard interface building upon the IDS.

The IDE interaction paradigm discussed in the previous chapter is not implemented as it is a more complicated implementation challenge and was not necessary for the evaluation of the work. It would make an interesting future study as it has many properties that are different from the implemented system.

The implementation is described in two stages. The first half focuses on the basic AR toolkit for direct library interaction. The second part utilises this library to implement the higher level interaction approaches as described in the previous chapter. The chapter concludes with a functional evaluation of the implemented components.

```
/** Example Render Object that renders an openGL teapot of the Given Colour and Size
 */
class RenderTeapot : public RenderObject
{
  public:
    /// Constructor with specified colour components and size
    RenderTeapot(const ARColour & _Colour, float _size)
    {
      Size = _size;
      Colour = _Colour;
    }

    /// Initialise the object
    void Initialise()
    {
      // Nothing to be done here for this model
    }

    /// Clean up the object
    void Terminate()
    {
      // Nothing to be done here for this model
    }

    /// Rend the teapot in GL
    void Render()
    {
      GLfloat mat_specular[] = { Colour.r, Colour.g, Colour.b, Colour.a };
      GLfloat mat_diffuse[] = { Colour.r, Colour.g, Colour.b, Colour.a };
      GLfloat mat_shininess[] = { 50.0 };

      glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
      glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_diffuse);
      glMaterialfv(GL_FRONT, GL_SHININESS, mat_shininess);
      glColor3f(Colour.r, Colour.g, Colour.b);
      glutSolidTeapot(Size);
    }
};
```

Figure 5.1: Source for simple RenderObject extension for ARDev

5.1 ARDev: an AR library for robot developers

The ARDev library is designed as a highly modular AR system built upon OpenGL. The choice of graphics API is relatively simple, only two graphics API's have wide support in rendering hardware: DirectX and OpenGL. Of the two, OpenGL has a far wider range of supported platforms, in particular it is supported on Linux, MacOS and Windows.

The library assumes a perspective camera model is being used for the AR system. The perspective camera model is more complicated than an orthographic or parallel projection but it allows for accurate rendering of geometric data and back projection of points for 3D estimation of tracked fiducials. The greater complexity of calibration and configuration of the camera is a trade off that must be made to gain the essential benefits of accurate geometric rendering.

The remainder of the system design is based around replaceable modules. This means that a user of the system is not constrained by the technologies selected by the system developer. For example if an alternative capture method to those provided by the library is required the end user can simply implement a new capture object.

Figure 5.1 shows the simplicity of adding a new module to the ARDev system. The code shows a *RenderObject* for rendering the example glut teapot model. The render method could easily be replaced with more advanced graphics code to represent any particular dataset that is desired.

5.2 Toolkit API

The toolkit architecture is centred around an output object. Each output object contains a capture object 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 object for returning the camera parameters, including pose. Also, the output object maintains three component lists: postprocessing objects, which are provided with the AR frame after rendering, 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 5.2 shows the software structure. The first four items, Capture, Camera, Postprocessing 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. All of these objects provide a base class and interface which needs to be implemented in order to create the AR system. These objects will be discussed in more detail below.

An example of using the library directly is given in Figure 5.3. The set up of the AR system can be divided into two stages, setting up the AR environment and then configuring the rendered objects. The first half of the code performs the following steps:

- Instantiating the capture and camera objects for a fixed viewpoint firewire camera.
- Connecting to the Player server on the robot.
- Instantiating the X11 display object and then starting the ARDev rendering pipeline.

The remainder of the code then sets of the position and rendering model for the laser data as follows:

- Instantiating the fiducial tracker for getting the robot's base position.
- Setting up the constant offsets between the fiducial and robot origins, and the robot and laser origins.
- Describing the links between the position objects.
- Instantiating the laser rendering model.
- Linking the rendering model to its position and adding them to the active render list.

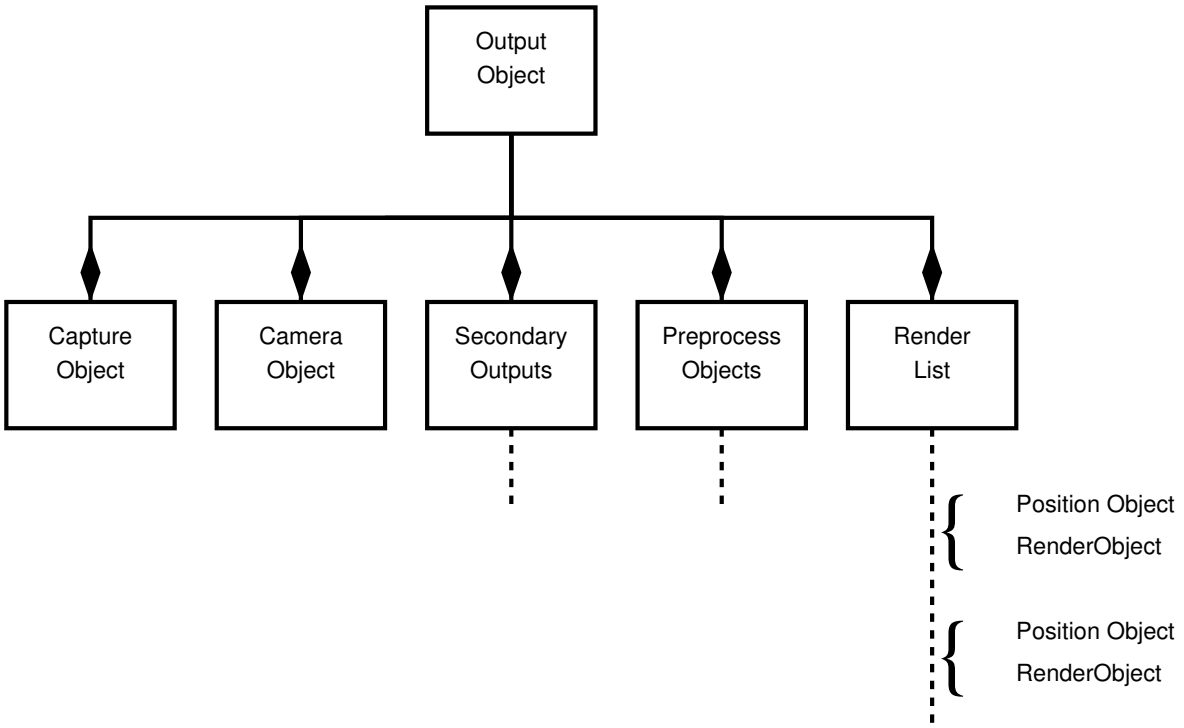


Figure 5.2: ARDev Software Architecture

5.2.1 ARDev Core

The ARDev core object is responsible for managing the AR rendering pipeline. Additionally it looks after the creation, initialisation and flow control of the AR rendering. The ARDev object provides a number of static methods that the developer can call directly from their application allowing for configuration and control of the rendering environment. Specifically the developer can *Start*, *Stop*, *Pause* and *Resume* the AR rendering as well as adding and removing output sets and individual render items.

The ARDev object is the only object that is designed to be stand-alone and not inherited by specific AR modules.

5.2.2 Output Object

The output object provides the core of the toolkit, providing the main thread for actually implementing the rendering loop. The main rendering loop consists of 8 stages as follows:

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;

```
/* Minimal test example for using ARDev as a library
Will render laser data for a pioneer robot */

#include <ardev/ardevconfig.h>
#include <ardev/artoolkitplus.h>
#include <ardev/player.h>
#include <ardev/ardev.h>
#include <ardev/capture.h>
#include <ardev/output_x11.h>
#include <ardev/render_base.h>
#include <ardev/debug.h>

#include <stdlib.h>
#include <stdio.h>
#include <math.h>
using namespace std;

int main(int argc, char * argv[])
{
  // Initialise ARDev
  ARDevInit(argc, argv);
  ARDev::DebugLevel = ARDBG_INFO;

  try
  {
    //set up the capture, camera, output and pre process objects
    // capture from a firewire camera
    CaptureObject * cap = new CaptureDC1394();
    cap->Initialise();

    // create camera and camera position from a calibration file
    ARCamera arcam("/path/to/calibration/file");
    CameraConstant cam(arcam);
    PositionConstant * CamPosition;
    ARPosition CameraOffsetConst(arcam.Origin,arcam.Direction);
    CamPosition = new PositionConstant(CameraOffsetConst);
    CamPosition->Initialise();

    // connect to the robot
    PlayerClientInterface * player = new PlayerClientInterface("RobotHostname",6665);
    player->Initialise();

    // create the X11 output object, with an 800x600 window
    OutputObject * out = new OutputX11((CaptureObject*)cap,&cam,CamPosition,800,600,":0",false);
    // start the AR system rendering
    ARDev::Start(out,"overhead");

    // track our robot with artoolkitplus markers
    // initialise the pre processor
    ARToolKitPlusPreProcess * artkp_pre;
    artkp_pre = new ARToolKitPlusPreProcess(cam);
    artkp_pre->Initialise();
    out->AddPre(artkp_pre);

    // link the pre process output with a position object
    PositionObject * RobotPos = new ARToolKitPlusPosition(*artkp_pre,13,0.31);
    RobotPos->Initialise();

    // set up the geometry of the robot
    // first the distance from the fiducial to the origin of the robot
    ARPosition RobotOffsetConst(ARPoint(0.15,0,-0.31),ARPoint(0,0,0));
    PositionConstant RobotOffset(RobotOffsetConst);

    // also the offset to the laser scanner, height only as the X&Y are provided by player
    ARPosition LaserPosConst(ARPoint(0,0,0.13),ARPoint(0,0,0));
    PositionConstant LaserPos(LaserPosConst);

    // link up the offsets
    RobotOffset.Next = RobotPos;
    LaserPos.Next = &RobotOffset;

    // create the laser render object
    RenderObject * laser = new RenderPlayerLaser(*player, ARColour(1,0,0));
    laser->Initialise();

    // add the laser rendering to the active list
    ARDev::Add(RenderPair(laser,&LaserPos),"overhead");

    // run until the user presses enter in the console
    getchar();

    // shut everything down
    ARDev::Stop("overhead");

    delete laser; delete out; delete cap; delete player;
  }
  catch (...)
  { cout << "Unknown exception caught" << endl; }
  return 0;
}
```

Figure 5.3: Source for simple ARDev session rendering laser data for a pioneer robot

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.

In order to implement an *OutputObject* the developer must implement the *Initialise*, *Terminate* and *ShowFrame* methods. The rest of the methods in the class are provided with the toolkit. In the reference GLX implementation, OpenGL with X windows is used for the output object implementation and the ShowFrame method is simply a call to GLXSwapBuffers.

5.2.3 Capture Object

The *CaptureObject* is responsible for interfacing to the camera driver and providing image frames to the AR system. The *CaptureObject* must implement two methods; *GetFrame* which returns a reference to the current frame, and *Fresh* which allows the rendering pipeline to check whether it needs to update the backdrop texture. The *Fresh* method allows the data rendering to be updated at a greater rate than the AR backdrop which is useful on systems with slow but relatively static video capture.

For efficiency reasons the frame is stored within the class and returned to the rendering system as a reference. As a multi-threaded system the renderer maintains a lock on the *CaptureObject* from before its call to *GetFrame* until after it has finished copying the texture to the GL subsystem and pre-processing the frame.

5.2.4 Camera Object

The *CameraObject* is responsible for maintaining the intrinsic parameters of the camera model. The model is used for setting up the OpenGL camera and thus is relatively simple

and consists of the field of view and aspect ratio of the camera.

The camera object is paired with a position object that specifies the extrinsic parameters of the camera, specifically the origin and orientation of the camera.

In addition to these items specific camera implementations may use more advanced models if vision based tracking is implemented and back projection is needed. The camera implementation provided is discussed in section 5.4.2.

The camera class provides additional support for stereo pairs. The base camera class includes convergence and separation variables that can be shared between a stereo pair. This is of most interest in systems where convergence can be dynamically controlled, such as an advanced video see-through system or optical see-through AR.

For systems where stereo convergence is dynamic but cannot be measured, an estimate is made of the distance to the object of attention during the rendering loop. Ray tracing is used to measure the distance to each virtual object and the closest object in the centre of the image is assumed to be the object of interest. Once this distance is determined the convergence angle can be calculated from the specified camera separation. If the user is not focusing on a virtual object this method will fail. However the effects will not be catastrophic because the convergence calculations only effect the appearance of virtual items.

5.2.5 FrameProcess Object

The *FrameProcess* base class is very simple with only a single method *ProcessFrame* that needs to be implemented. This method will be called for each new frame the system captures. A partner class will need to be implemented (for example a position object) to actually provide the output of the pre-processing to the system. The *FrameProcess* class is also used for post-processing where the complete AR frame can be rendered to a secondary stream such as a video recording.

5.2.6 Position Object

The *PositionObject* provides the position and orientation at which to render a virtual element. A *PositionObject* must implement two methods. The first *GetPosition* simply returns the position to render at. The second *Present* returns an indication of whether this position is valid. For example a vision based tracking system would return false if the marker being tracked had been lost.

A *PositionObject* also contains a next pointer that creates a chain of position transformations allowing a single tracked position object to be used as the base position for several rendered elements. Figure 5.4 shows an example position tree where a robot is optically tracked and static offsets to laser and sonar sensors are provided.

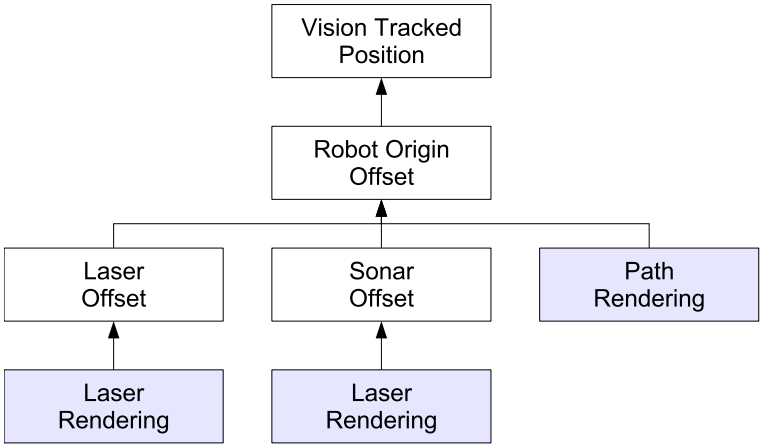


Figure 5.4: Render Tree

5.2.7 Render Object

The final base object is the *RenderObject* which is responsible for rendering a single virtual element. The rendering is carried out in three stages. First *RenderBase* is called which should render into the depth buffer any available models of the real world components of an object. The base rendering stage only affects the depth buffer and so will not be visible; the goal is to recreate known sections of the real depth map so that occlusion of virtual objects by real objects can occur.

The second stage of rendering is the solid components of the virtual element which is achieved with a call to the *Render* method. This renders to both the depth and colour buffers.

The final rendering stage is the transparent components of a virtual element. These are rendered in the *RenderTransparent* method. This renders virtual components into the colour buffers but not the depth buffer. While this does not give perfect transparency it gives an effective approximation if the levels of transparency are not too complicated. The only alternative is to render the virtual elements in order, this requires a complete scene graph to be maintained which is both difficult and restrictive in an AR system.

5.3 System Performance

System performance has been evaluated using two setups, one with a stereo HMD and the other with a fixed camera and a large wall-mounted display. The details of the two configurations follow. Both of these setups used video see-through AR.

Video See-through Head Mounted Setup

- Display: Trivisio ARVision3D HMD;

- Capture: Stereo USB cameras from VRMagic, built into HMD (640x480 resolution);
- Positioning: Ascension Technologies Flock of Birds magnetic tracker (Tracks HMD relative to Robot).

The video see-through system uses non vision tracking for a number of reasons, specifically; the tracking has a higher sample rate, less computer resources are consumed, the system will work when examining the edges of data sets when the robot is not in the frame and it gives the flexibility to change to an optical see through display at a future time.

Wall Mounted Setup

- Display: 50" Plasma TV;
- Capture: Prosilica EC 1350C Firewire camera (1360x1024 resolution);
- Positioning: Vision based fiducial tracking.

The accuracy of both systems is dependant on the camera calibration parameters and tracking accuracy. For the HMD the error was dominated by the camera calibration as the magnetic tracking gives high accuracy results in its operating range. For the wall mounted setup tracking error was also largely dependant on the camera calibration. Calibration errors using the Tsai calibration method were in the order of one pixel for the 2D error and 3mm for the 3D error at a 3m range.

Performance in terms of frame rate is largely dependant on the hardware used. On a Pentium4 3.05 Ghz processing 1360x1024 pixel frames the frame rate is around 6 frames per second (FPS). With non vision based tracking performance is limited by the cameras, around 25 FPS for the Prosilica. Higher frame rates can be achieved for the virtual data by repeating the previous frame if new rendering data has arrived before a new backdrop frame is available.

A photo of the video see-through system is shown in 5.5 and a diagram of the wall mounted setup is shown in 5.6. Example outputs from the system can be seen in Figures 5.8 through 5.14.

5.4 Reference Object Implementations

A number of implementations of each of these base objects are provided with the toolkit and some of these will be described here. The remaining implemented objects are related to the Player/Stage project and will be discussed in section 5.5. As mentioned previously the toolkit API has been designed specifically with ease of extension in mind and so additional capabilities can easily be added to the system.

A summary of all the implemented modules is presented in table 5.1

Figure 5.5: Set up of the video see-through head mounted display with the flock of birds magnetic tracker

Name	Description
General Objects	
CaptureDC1394	Image capture from dc1394 Linux API
CaptureV4l	Image capture from Video 4 Linux API
OutputX11	Output to an X11 Display
OutputMovie	Output to a movie file using the ffmpeg library
RenderModel	Rendering of 3DS model file
RenderB21rBase	Rendering of B21r invisible base
Player [127] based objects	
CameraFile	Access to static pre-calibrated camera data
CameraPlayer	Access to the pan tilt interface
CameraStage	Camera based on stage simulation
CameraPlayerPosition3D	Camera using position 3D interface
CaptureStage	Simulated camera based on stage simulation
PositionPlayer	Access to position 2D interface
PositionPlayer3D	Access to position 3D interface
RenderPlayerLaser	Rendering of Player laser data
RenderPlayerSonar	Rendering of Player sonar data
RenderPlayerIr	Rendering of Player IR data
RenderPlayerPath	Rendering of past Player position information
RenderPlayerBumper	Rendering of Player bumper data
RenderPlayerPTZ	Rendering of Player PTZ data
RenderPlayerMap	Rendering of Player map data
RenderPlayerLocalise	Rendering of Player localisation data
RenderPlayerActArray	Rendering of Player actuator array data
RenderPlayerLimb	Rendering of Player limb data
Blob Tracking Objects	
OpenCVBlobTrackPreProcess	Used to process incoming frames
OpenCVBlobTrackPosition	Uses data from processed frames to give the 3D position of markers
ARToolkitPreProcess	Used to process incoming frames
ARToolkitPosition	Uses data from processed frames to give the 3D position of markers
ARToolkitPlusPreProcess	Used to process incoming frames
ARToolkitPlusPosition	Uses data from processed frames to give the 3D position of markers

Table 5.1: Key Objects Implemented in the Prototype System

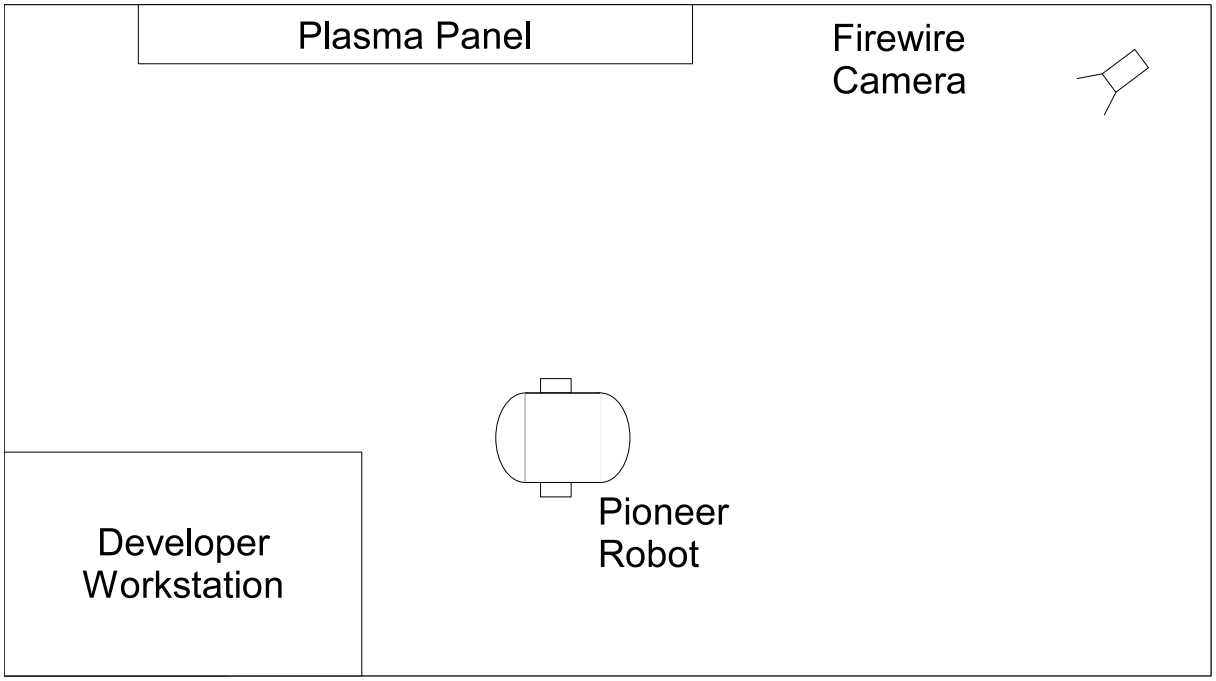


Figure 5.6: Wall mounted hardware configuration

5.4.1 Capture Objects

The toolkit comes with implementations capable of image capture from the video4Linux API, Linux firewire cameras and from static pre-captured frame sets. The latter is intended primarily for debugging purposes. These modules are simple wrappers for the underlying video APIs.

5.4.2 Camera Objects

Currently the only source of camera parameters is a static camera model. Included with the toolkit is a calibration utility that can be used to generate this model. The calibration is based on the Tsai method [168].

The calibration procedure will automatically detect calibration points and perform the calibration if a suitable object is used, as shown in Figure 5.7. If a suitable object for automatic calibration is not available the user can provide an alternative set of matched 2D and 3D calibration points, either by manually identifying these points with an external image viewer or through some alternative automatic extraction routine. Once the matched 2D and 3D points are obtained then these are processed by the calibration application to produce the camera model. The calibration procedure produces a camera model that is suitable both for 3D rendering and the back projection of image points for tracking purposes.

The automatic detection algorithm is a simple gray-level segmentation processing ap-

proach and consists of the following steps:

- Conversion of the image to gray scale;
- Segmentation based on gray level;
- Segments are then ranked based on their aspect ratio, concavity, number of holes and size;
- The best three segments are assumed to be the cube faces and are ordered based on centre of mass;
- Each face is then segmented again to extract the holes;
- The nine largest holes of each face are extracted and ordered;
- The ordered points are then paired with their 3D counterpart ready for calibration.

A variation of this calibration procedure was used for the video see-through system. Here the tracked position of the HMD was subtracted from the calibrated pose to give the calibrated pose of the cameras relative to the tracking system. The offset could then be used at run time to calculate the tracked camera positions.

5.4.3 Output Objects

The system comes with a reference implementation for a GLX output object for use with Linux X11 server.

5.4.4 Process Objects

Over the course of the work three *ProcessObject*'s have been implemented for pre-processing, all of which track the position of an item in the world through vision processing. The first method is based on the ARToolkit project [5] using the ARToolkit code to extract the markers and then back projecting them based on the ARDev camera model.

The ARToolkit was designed for arm length AR interaction and so for large spaces is often not effective for tracking and so a second vision tracking method was implemented. This method uses the opencv [122] library to implement a colour blob tracker. The tracking algorithm matches colour segments in hue space, taking the largest blob that passes basic sanity checks as being the tracked item. Two coloured blobs at a predefined height above the ground plane provide a 3D estimate of pose and orientation.

A third tracking method was implemented for the case studies, this utilises the ARToolkitPlus [63, 174] tracking code to get the objects position. This is very similar to the ARToolkit approach with the exception that different markers and underlying tracking

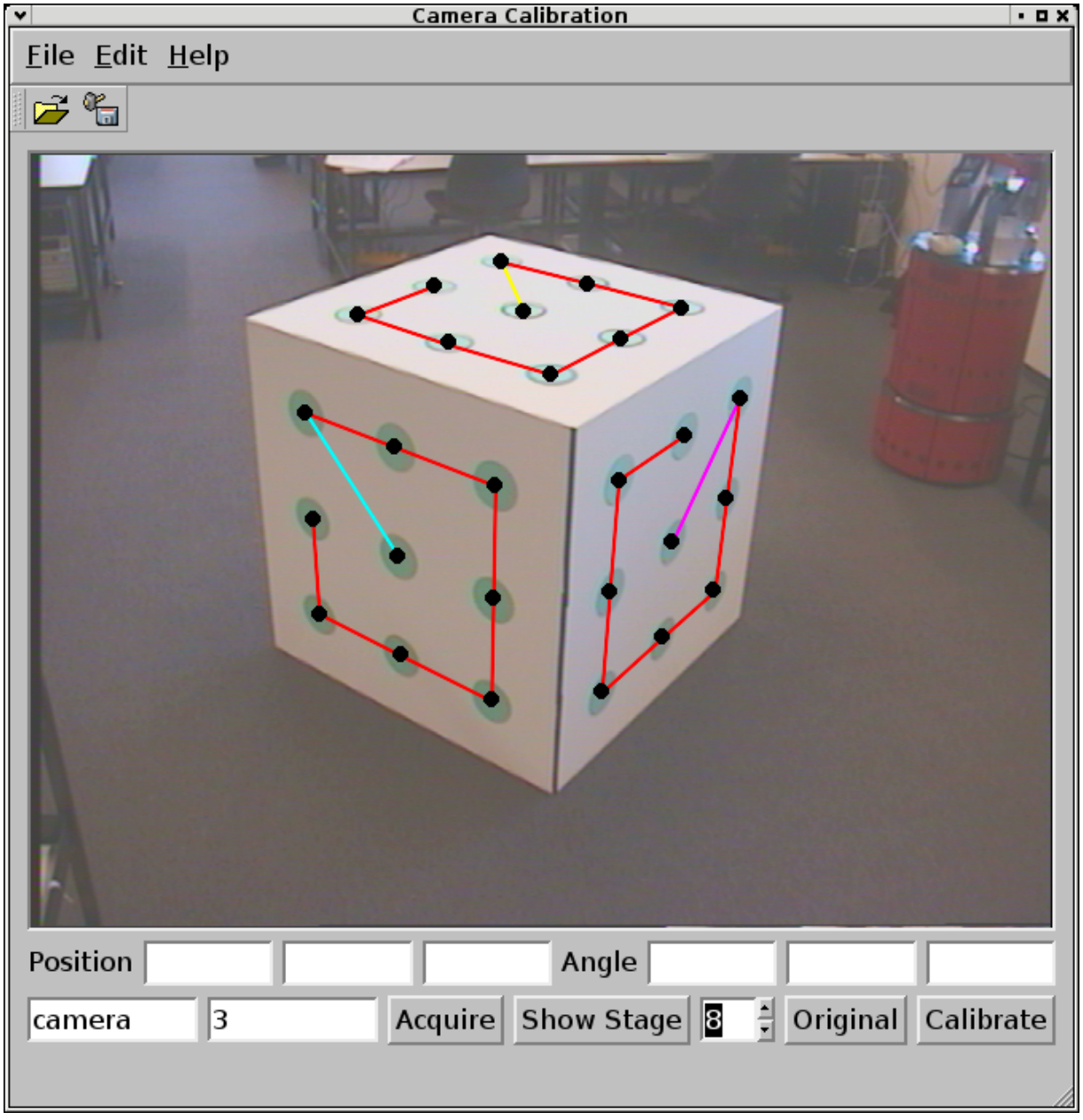


Figure 5.7: Calibration application

code is used. Up to 4096 unique markers can be tracked up to a range of 3 or 4 metres with a high resolution camera and suitably sized markers. In comparison to the ARToolkit marker code ARToolkitPlus did not require templates to be created for the markers and had better performance in varying light levels which was an issue in the development space.

The ability to save the video stream as a video has also been implemented as a Post-Processing Object. This is based on the ffmpeg library [50] and can output in a range of video codecs. Given the CPU intensive nature of both fiducial extraction and video encoding, performance can be adversely affected when using both on the same PC.

5.4.5 Position Objects

Three *PositionObject*s are provided to interface to the fiducial extraction methods described above. These simply take the 3D position from the vision tracker and return it to the system. Access to other hardware such as the Ascension Technologies Flock of Birds is provided through Player/Stage as described in Section 5.5.

5.4.6 Render Objects

The toolkit has built-in support for rendering basic debug objects (such as the rendering axes and the OpenGL Teapot) as well as support for rendering 3D Studio Max (3DS) model files. All of the robot related renderings are accessed via Player/Stage and will be described in Section 5.5.

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

5.5 Implemented Player Modules

This section discusses the modules that were implemented to support visualisation of the Player/ Stage project. Specifically, visualisations have been implemented for the following Player interfaces: *Position2D*, *Sonar*, *Laser*, *IR*, *Bumper*, *PTZ*, *Map*, *Localize*, *ActArray* and *Limb*.

The Player rendering modules all have a similar format. The modules connected to the same server share a *PlayerClient* object which provides the connection. The *PlayerClient* object connects to the server on initialisation. If at any stage the server disconnects then the object changes to the uninitialised state causing any future request for Player data

to trigger a new connection attempt. Using this method the system can detect robot connections and disconnections and is thus able to run as a permanent installation.

Sample output of these modules is shown in Figure 5.8 through Figure 5.14.

5.6 Implementation of an Augmented Reality Debugging Space

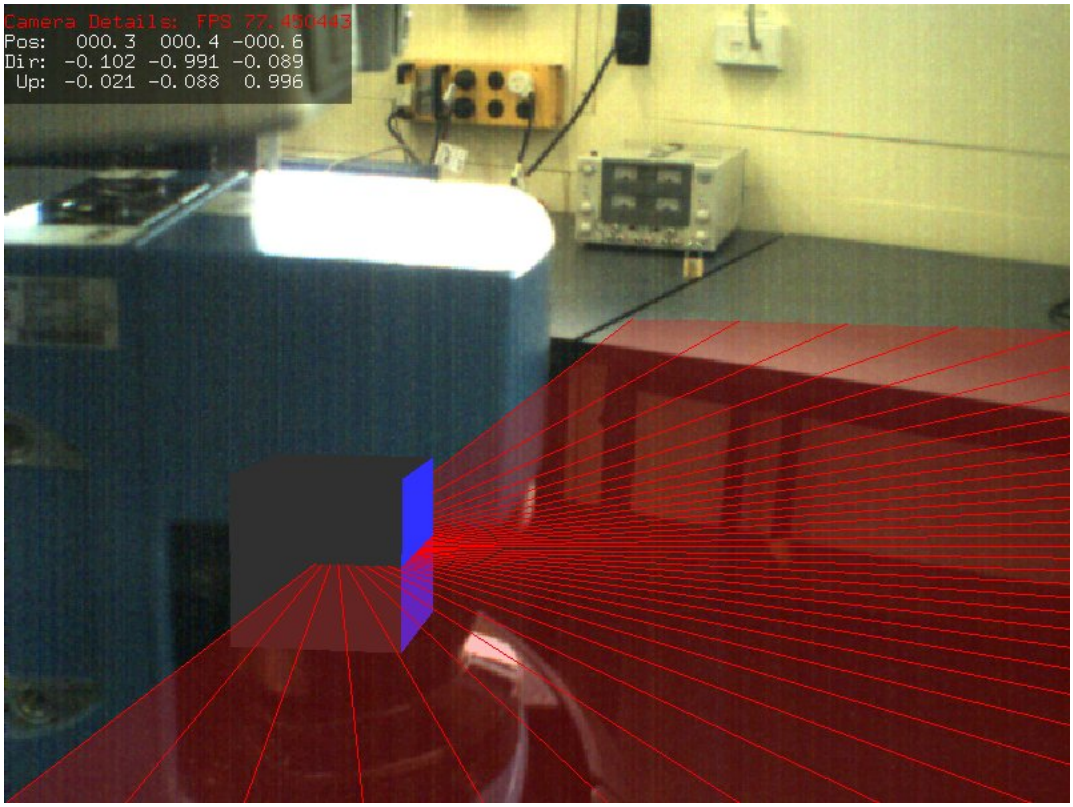
This section describes the implementation of an intelligent debugging space (IDS) for robot developers using the AR toolkit described in the previous sections. The IDS provides a set of stock visualisations for standard robot data with a particular emphasis on visualisations for the Player/Stage project. The debugging space also provides a whiteboard style interface for custom visualisations. The list of the provided stock visualisations for player is given in Table 5.1.

The implemented IDS can support any AR configuration that the underlying toolkit supports such as a video see-through HMD or a fixed viewpoint AR configuration. For the purposes of the case study evaluation a fixed viewpoint AR setup was used, this is discussed more fully in Chapter 6.

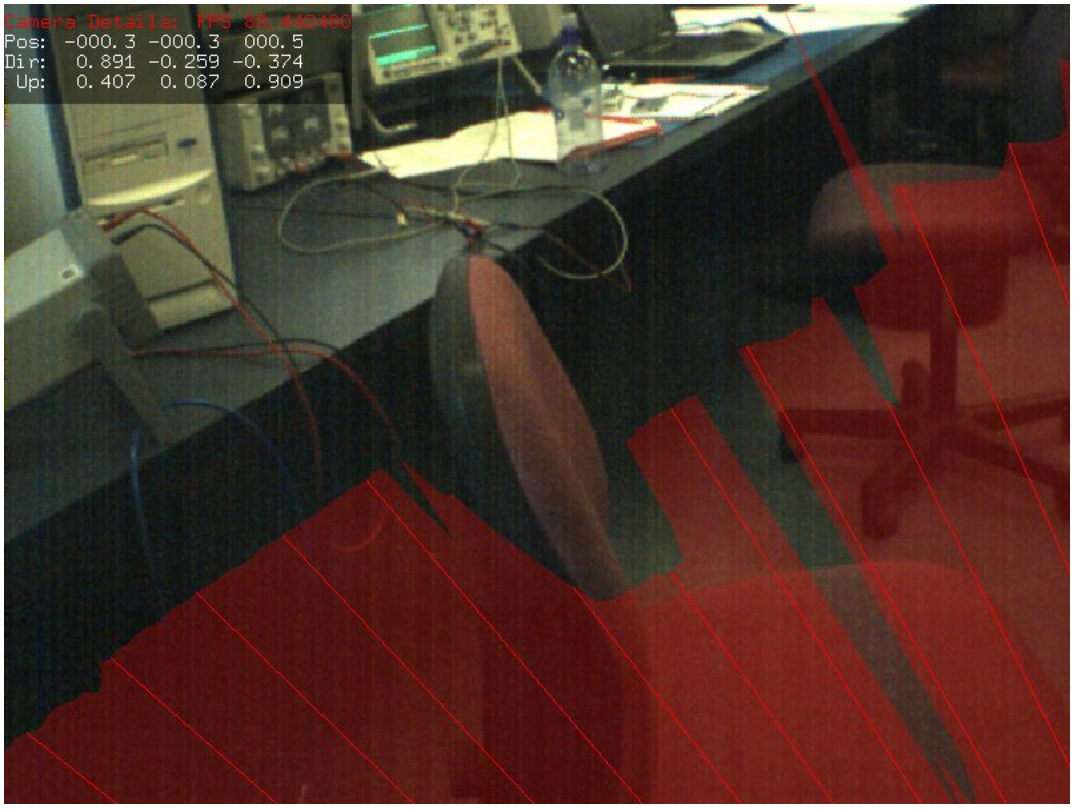
As described in Section 4.2 the requirements of the debugging space are:

- The ability to track the physical presence of multiple robots;
- The ability to track the operational (i.e. network) presence of multiple robots;
- The ability to determine the functionality and configuration of the tracked robots;
- The ability to connect directly to the robot without an intervening user application;
- A set of stock visualisations for the robots;
- A control interface that will give coarse grained control over what visualisations are present and some basic configuration of the presentation of the data sets (e.g. colour);
- An AR display including appropriate user tracking for the display. Multiple displays may be needed for multi-user systems.

By utilising the Player/Stage project to encapsulate the robot platform many of these tasks are made simpler. Player/Stage provides a set of standard interfaces and the ability for multiple clients to connect to the same robot platform. In this way the IDS can access robot data independently from the robot application that the developer is writing. By implementing stock visualisations for the interfaces that Player provides the system is



(a) Laser data origin



(b) Laser data edge

Figure 5.8: HMD view of laser data

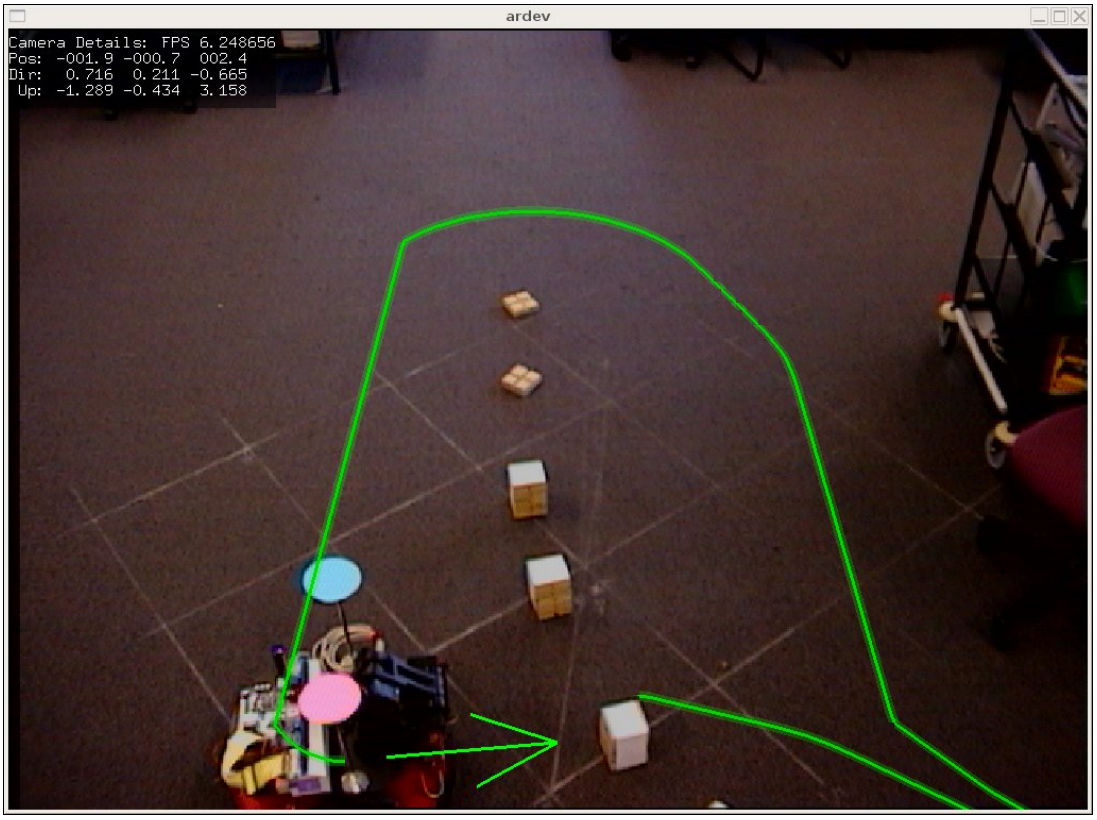


Figure 5.9: Pioneer odometry history

able to present a useful set of visualisations for the developer vastly reducing the amount of custom rendering that needs to be written.

The whiteboard approach requires a method of generating custom visualisations within the developer's code. The Player/Stage *graphics2d* and *graphics3d* interfaces provide a simple method of drawing 2D and 3D graphics over a network connection. The ARDev implementation was built to leverage this capability. To support the graphics interfaces the AR system needs to be run as a Player plug-in driver.

The final requirement for the debugging space, knowledge of the robot capabilities and configuration is essential for working out what we can render. This information can however be provided by a static configuration file for a lab with a small number of robots. Alternatively if some service discovery method is available this could be used. The static configuration approach is used for this work. A graphical tool was developed for creating and testing the AR configuration files.

The remainder of this chapter will describe the details of:

- The static configuration manager;
- The configuration user interface;
- The Player plug-in driver.

Figure 5.10: Sonar Sensors on Pioneer Robot

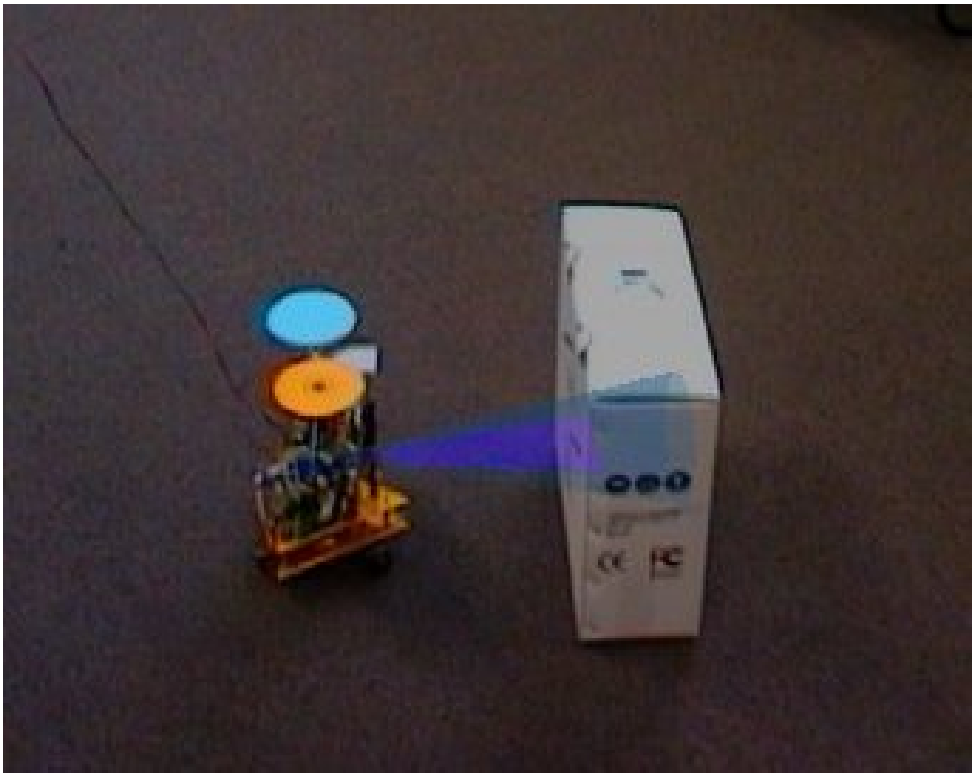


Figure 5.11: Shuriken Robot with IR and Sonar

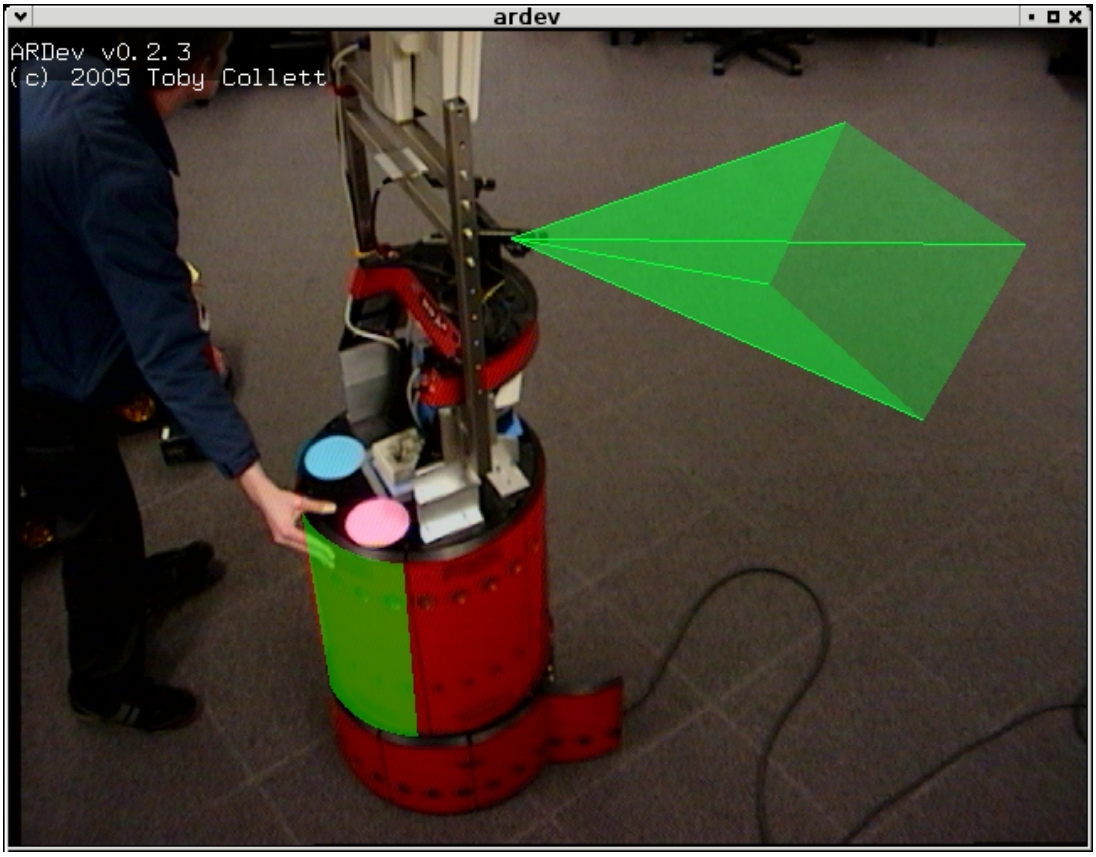
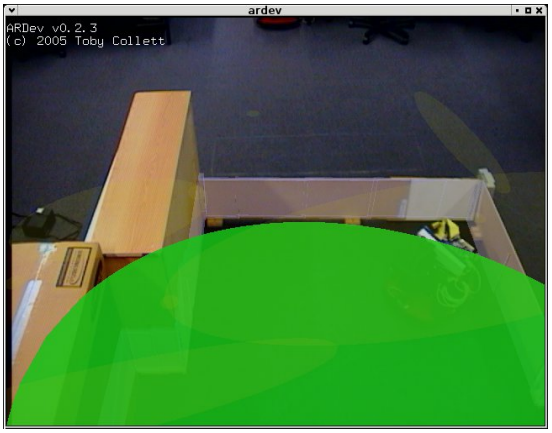
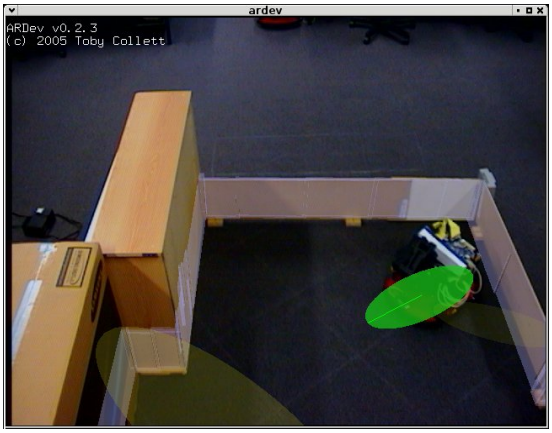


Figure 5.12: B21r with Bumper and PTZ visualisation

Figure 5.13: Actuator Array and Limb interface visualisation. The yellow lines represent the skeleton of the actuator array and the red sphere the end effector of the limb.



(a) AMCL Initial Distribution, High Covariance



(b) AMCL Improved Estimate

(c) AMCL Converged Result

Figure 5.14: Augmented system localisation visualisation, using the Adaptive Monte-Carlo Localisation (AMCL) [126] driver from Player. The ellipses represent the uncertainty in the localisation estimate with the most likely estimate highlighted in green.

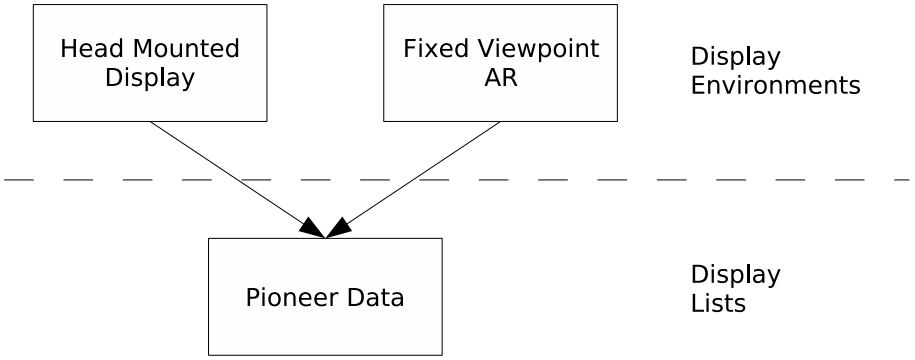


Figure 5.15: Configuration Manager Separation

5.6.1 The AR Configuration Manager

The configuration manager is responsible for loading the static configuration file and instantiating the appropriate set of ARDev objects. The configuration file specifies the configuration of a set of AR display environments, which describe the hardware configurations in use including; camera parameters, capture method, preprocessing objects, and other high level configuration. In addition the configuration specifies a number of display lists which describe the AR scene specifying what elements should be rendered, and where they should be rendered at. There can be multiple environments that use the same display list, this many to one relationship is shown in figure 5.15.

The configuration manager has a set of handler objects that are responsible for; reading in the configuration information, representing this as a Qt widget if needed, converting the configuration to text for saving and instantiating the object when the visualisation is run. Each unique ARDev object needs its own handler to represent it.

The configuration file is specified in XML which is parsed using the Qt XML DOM parser. Once the configuration is loaded the manager walks the DOM tree and creates a handler for each object that is present. If a handler is not available then the manager reports an error. An example configuration file is presented in Appendix B. This config represents the configuration for five robots used in the IDS with the overhead camera setup. For example it contains details such as the mapping between the marker ID on the top of the robot and its IP address and sensor configuration. Most of the other configuration details, such as laser scanner resolution and range are queried at run time from the Player server. The configuration file also contains some details in terms of the display of the data, such as per robot colour specification.

A diagram of the configuration manager and its related classes is shown in figure 5.16. The key functionality is loading a configuration, saving a configuration, initialising the AR system, running the AR system and cleaning up upon termination. The remainder of the methods are utility methods either to aid in these tasks or to allow external objects access to the ARDev objects.

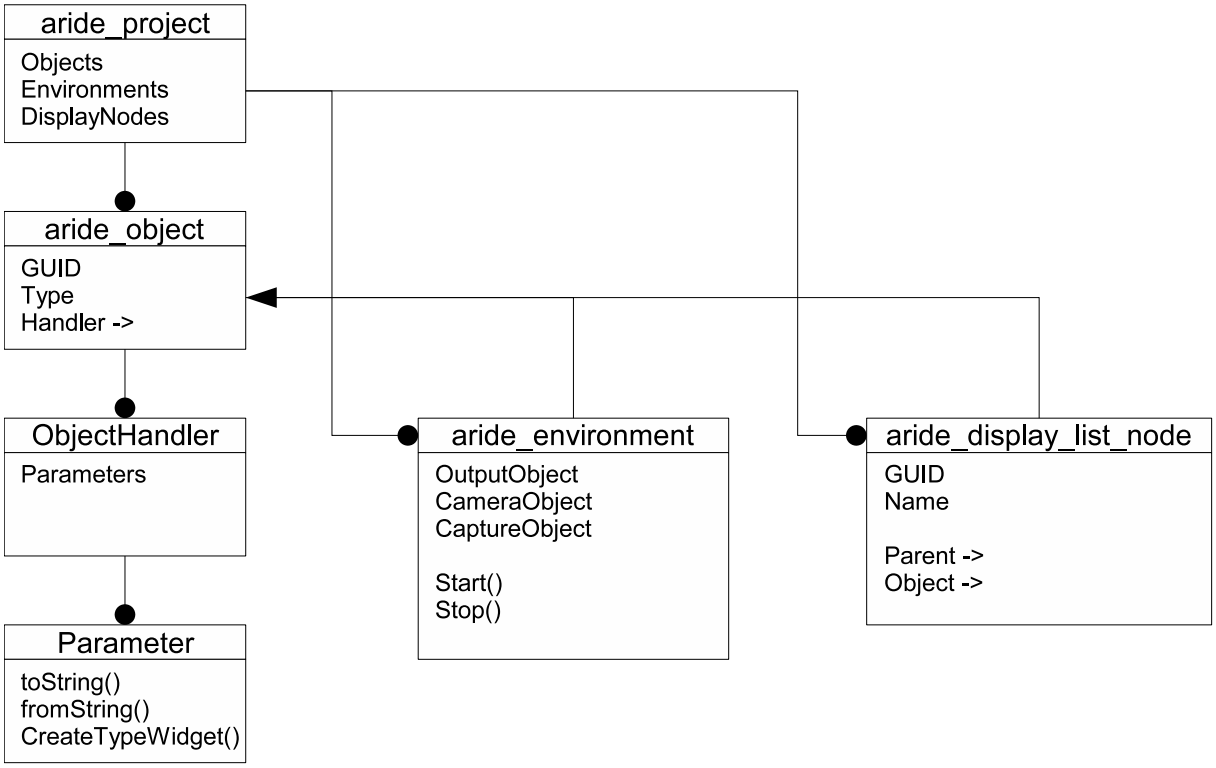


Figure 5.16: Configuration Manager Classes

5.6.2 The AR Configuration UI

The Configuration UI is designed to aid in the configuration of the AR system. The UI is implemented with the Qt GUI toolkit. The UI generally reflects the design of the configuration manager by dividing configuration into environments and display lists.

The UI application is a fairly light-weight framework for representing the handlers in the configuration manager. Each handler is responsible for drawing its own widget and updating its portion of the configuration.

The other functionality of the configuration tool is to provide basic testing of the configuration, allowing the user to start and stop the AR system as well as toggle individual render items as enabled or not.

Figures 5.17, 5.18 and 5.19 show example configuration dialogs from the user interface.

5.6.3 The Plug-in Driver

One of the important features of the AR visualisation system is the ability for developers to render custom data. This can be achieved by creating a new *RenderObject* as a plug-in to the ARDev library. However, for one-off visualisations created by the robot developer this can be inconvenient. The ARDev Player plug-in allows for whiteboard style custom renderings to be created through use of the graphics interfaces and associated Player

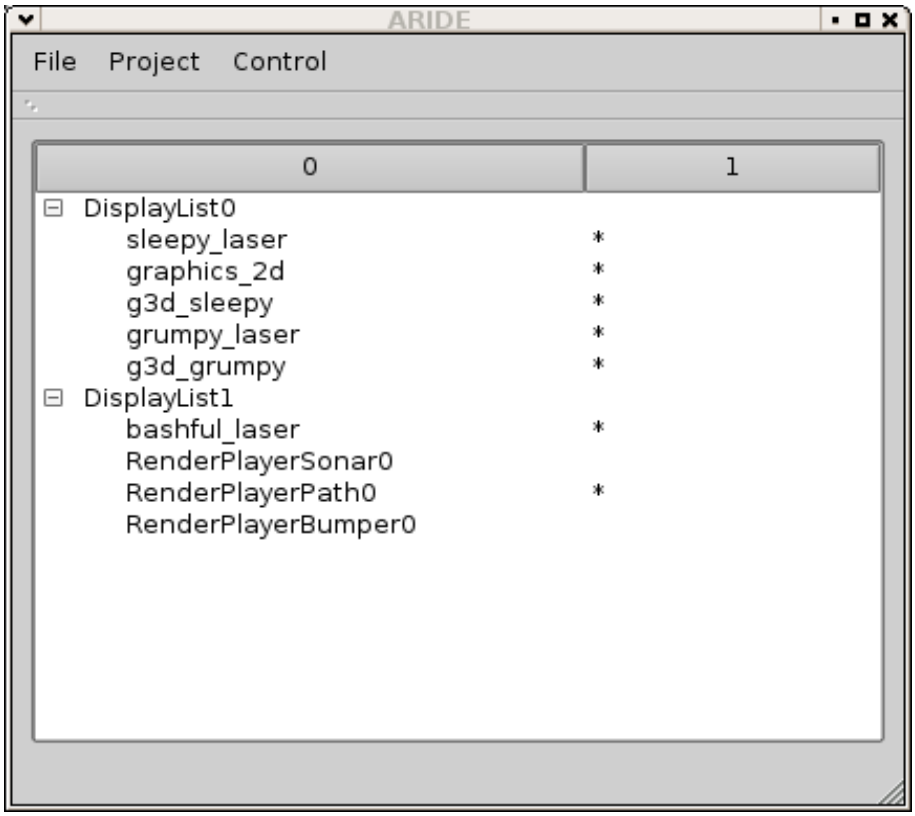


Figure 5.17: Configuration UI - Main Window

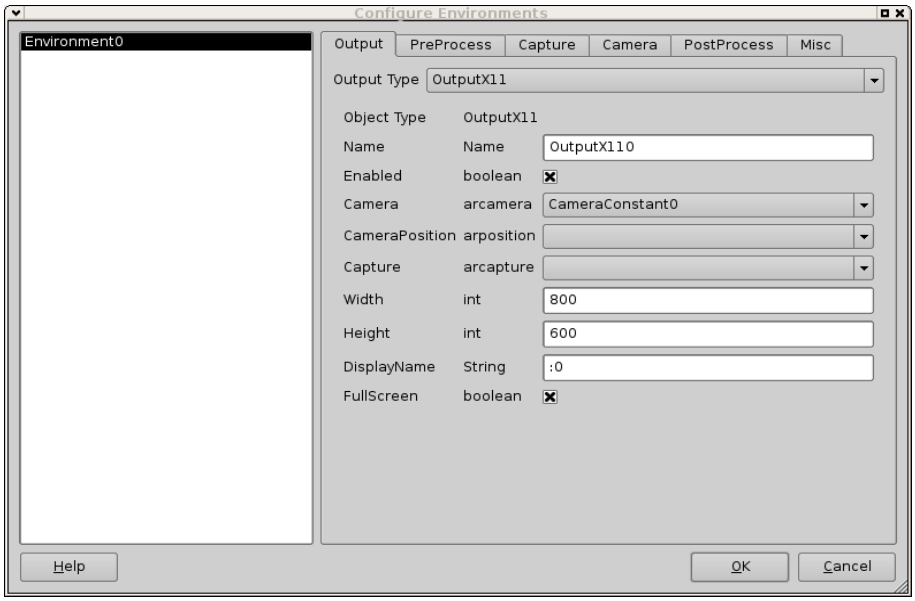


Figure 5.18: Configuration UI - Environment Configuration

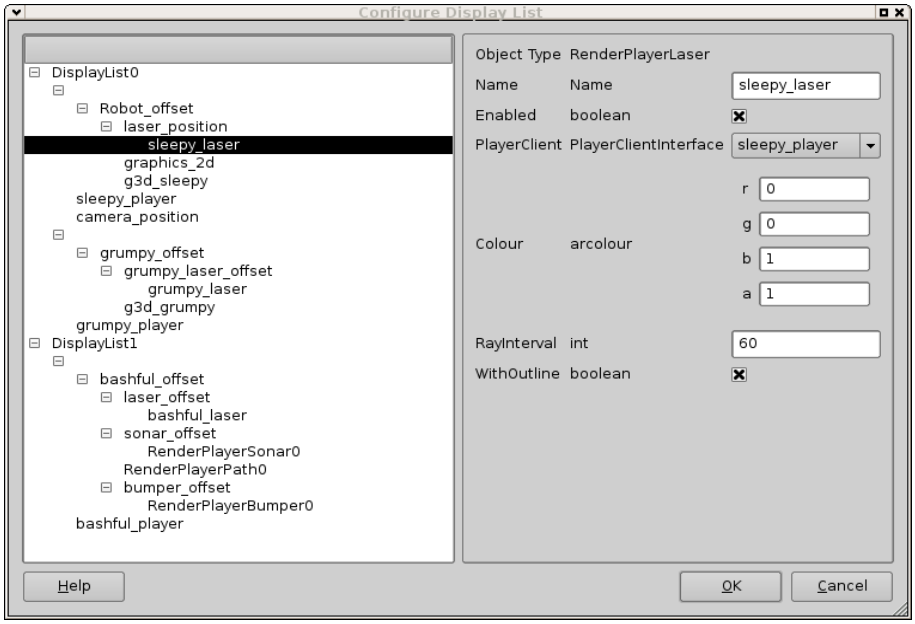


Figure 5.19: Configuration UI - Display list Configuration

proxies. This has a number of advantages, in particular the visualisations can be created in any language with a Player client library (currently C, C++, Python and Java) and the same visualisations can be used with Stage the Player simulation suite.

The plug-in consists of two parts, a plug-in for Player which processes the messages to the graphics interfaces, and an ARDev Render object that performs the actual rendering. The driver creates the AR environment using the configuration manager described above and then connects to the created render object.

5.7 Visualisation Design

As discussed in Section 2.3 the general conclusion of the visualisation community is that while guidelines can be provided about a number of aspects of visualisation of data sets generally the visualisation style will depend on the specific application. Throughout the implementation and case study work with the toolkit a number of visualisations have been implemented and this section discusses the general recommendations of the author with respect to visualisation design for robot developers.

Most of the visualisations that were implemented reflect an underlying sensor that has a simple physical representation. It was found in general that renderings that were close to the physical properties of the sensor worked well, for example rendering sonar sensors as cones of data and laser scanners as a plane of "light". The use of physical details of a data set in order to provide clarity in a broad view is discussed by Tufte where he states as a design strategy "to clarify, add detail" [171]. The key to this strategy is adding the

detail in such a way that it blends into an even texture when viewed on a broad scale.

The laser visualisation was one of the most heavily used in the development of the system and it went through a number of iterations. One simple rendering was to have a ray rendered for each line in the laser scan, however this made the rendering too busy. One of the key advantages of AR is the ability to compare data with the real world and heavy graphics prevent this from happening. An alternative rendering was to use a partially transparent polygon of data bounded by the points the scan measured.

When examining the polygon version of the laser rendering it was difficult to keep track of the origin of the data when the robot was not in the field of view. Two alternative solutions to this were implemented, one option is to render a solid ray approximately every 10 degrees in the laser view. An alternative is to alter the transparency of the polygon every 10 degrees, this gives a fan effect allowing the underlying ray quality of the data to be conveyed. The addition of a solid outline to the polygon was found to aid in distinguishing the edge of the scan, allowing better comparison with objects. Several of the alternate renderings are shown in Figures 5.8, 5.23, 5.20 and 5.21.

Initially little attention was given to colour selection for the visualisations, with selection generally following that of existing tools such as the Player Viewer, or based on arbitrary choices that allowed distinction between the different interfaces that needed visualised together. When the system was initially used with more than one robot it became apparent that the ability to distinguish between the different data sets was important, as shown in Figure 5.20. This was achieved by allowing the visualisations to be parametrised in the configuration for the robot as shown in Figure 5.21. One possibility to support a large set of robots is to allow the renderings to be themed with a base seed specified for each robot, for example a colour that the rest of the settings are based on.

One of the key points with creating the visualisations is what information to display. In terms of aiding debugging coding errors the data that is represented should be the same as the data that is available to the program. For example with any measuring device there is a limit to the accuracy of the measurement, however if the interface to the device does not report this quality information it could be misleading as the developer may be able to see more than the program. On the other hand if the purpose of the visualisation is to aid the developer in understanding the robot platform and how its sensors interact with the world then this additional information is potentially valuable.

The visualisations used in development generally represented the data as seen by the program, a few concessions were made in terms of the physical properties of devices, such as using cones to represent the sonar scans. In the case of the sonar visualisation this selection was made from an aesthetic point of view, no data has been collected which indicates whether simple rays or cones are better for debugging, although the cones seemed to aid understanding for users unfamiliar with the sensors.

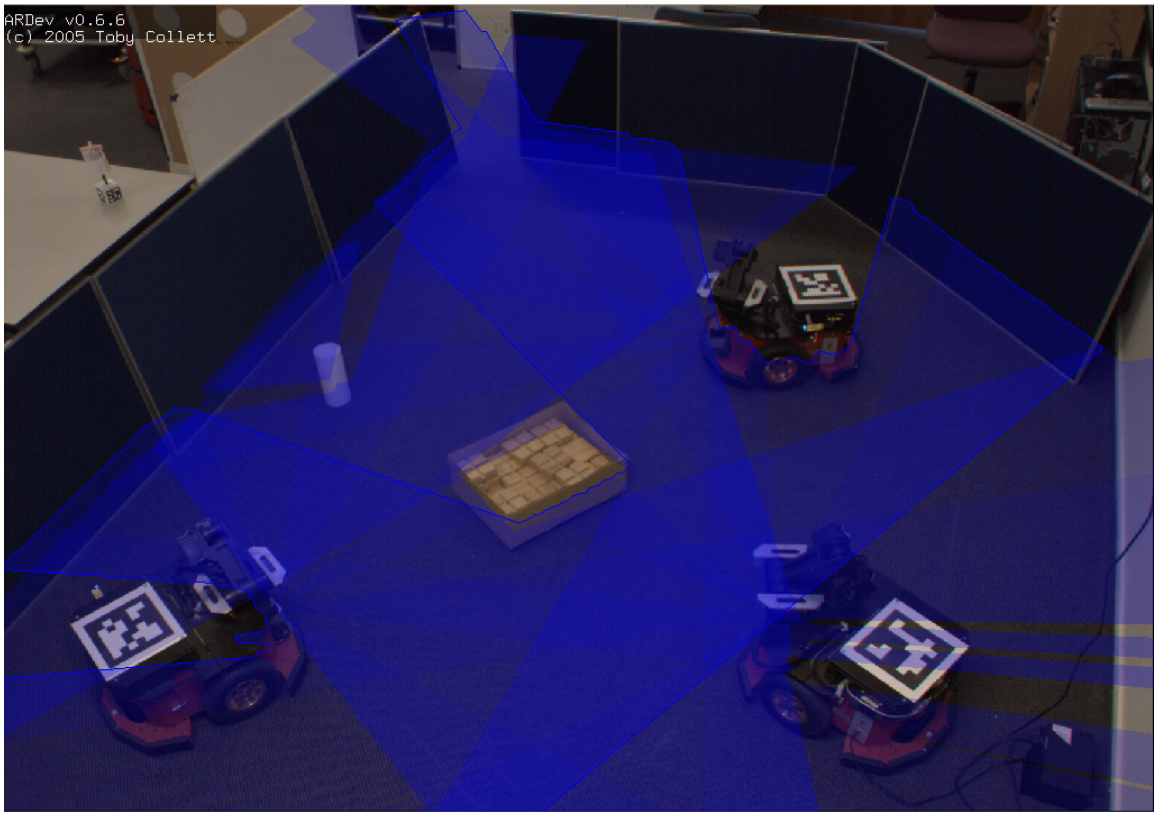


Figure 5.20: Rendering of three lasers in a single colour

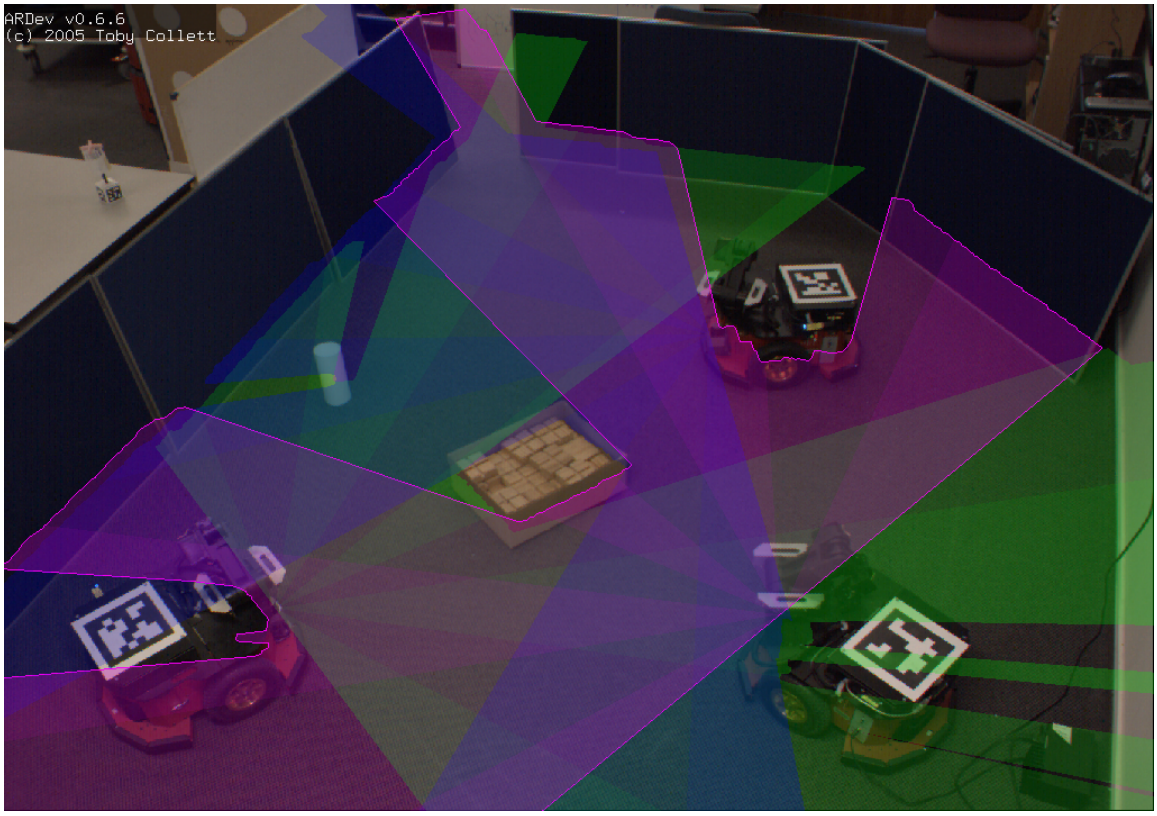


Figure 5.21: Rendering of three lasers with a unique colour for each robot

The visualisation for the localisation interface is an example of an interface that does provide statistical information. The interface provides a list of possible robot locations with the covariance and a weighting specified. These represent the precision and quality of the estimate. The visualisation represents the interface information in three ways; the most likely estimate is highlighted through the use of colour, the covariance is represented with an ellipse with its dimensions determined by the eigenvectors and eigenvalues for the estimate, and the weighting is represented by the transparency of the ellipse.

The need for flexibility specifying renderings was one of the driving factors of the IDS. Figure 5.1 showed the minimum code needed for specifying a new visualisation, and support for the whiteboard paradigm allows existing visualisations to be augmented by the user. The configuration UI also allows components to be enabled and disabled at run time, allowing the user to switch between alternate visualisations either to compare them, or to access some other piece on information.

In summary the guidelines that come out of this discussion are:

- Represent the data of a sensor in a way that is consistent with the physics of its operation.
- Choose representations that minimise interference with the background view, for example transparent regions.
- Allow the rendering to be parametrised so views of multiple robots will not interfere.
- Provide simple stock visualisations and allow developers to augment them with renderings of program decision variables.
- Limit the renderings to the data available in the data interface. If the developer needs more information than this it is probably a sign that the program needs the information as well. The exception here is in providing the developer with an understanding of the properties of an unfamiliar sensor.

There is a wealth of material available with general guidelines for creating renderings, several references were provided as a starting point in the literature review (Section 2.3).

5.8 Functional Evaluation

The basic system performance was discussed in Section 5.3. This section provides a functional evaluation showing that the implemented system meets the requirements as defined in the previous chapter. The requirements of an IDS were defined as:

- The ability to track the physical presence of multiple robots;

- The ability to track the operational (i.e. network) presence of multiple robots;
- The ability to determine the functionality and configuration of the tracked robots;
- A set of stock visualisations for the robots, and the ability to connect directly to the robot without an intervening user application;
- A control interface that will give coarse-grained control over what visualisations are present and some basic configuration of the presentation of the data sets (e.g. colour);
- An AR display and the appropriate user tracking for the display. Multiple displays may be needed for a multiple users system.

To support the additional AR whiteboard interface the system also needs to be capable of rendering custom developer visualisations.

The three fiducial extraction modules provide the ability to detect the robot's physical presence and track its 3D location, fulfilling the first requirement. The use of Player for interfacing to the robot gives the ability to track the operational presence of the robots and allows for the AR system to connect independently to the robot platform.

Player also provides a set of standard interfaces allowing for stock visualisations to be included in the AR system. The implemented toolkit includes visualisations for ten common Player interfaces providing a solid base for developers to work from. The implementation of these visualisations used the plug-in API of the base AR system proving its ability to support extension.

The configuration of the robots is defined using the GUI described in Section 5.6. This is suitable for a moderate number of robots but some sort of automated discovery would be desirable when dealing with large numbers of heterogeneous robots. The user interface also gives control over which elements are active for a given robot and the ability to modify some aspects of their rendering such as the base colour for the visualisation.

Custom visualisations are provided through a Player plug-in driver that implements the *graphics2d* and *graphics3d* interfaces from the Player project. These allow for simple points, lines and polygons to be rendered by the developer from their robot application. These custom renderings are valuable for providing visualisations of key meta data to augment the stock Player visualisations.

With this combined functionality the implemented system was able to run as a permanent installation in the University of Auckland robotics lab. The system is able to robustly detect the presence of robots and begin displaying relevant sensor data when active robots are within the debugging space.

The implementations of the stock Player visualisations also show that the system is powerful enough and flexible enough to render data in a variety of forms, for example

the sequential data in the path visualisation and the stochastic distribution in the AMCL localisation. Given that direct access to the OpenGL library is available the framework is able to render anything that OpenGL can.

One of the key benefits of the implemented system was expected to be the ability to highlight inconsistencies between the robot's world-view and the real world state, and this is indeed the case. A comparison of a pure virtual rendering (Figure 5.22) of the robot laser scan and an augmented view (Figure 5.23) of the same data shows this. It is clear from both figures that only one of the two boxes is visible in the laser scan. However, in the augmented view it is far easier to determine which box is not seen and the cause of this.

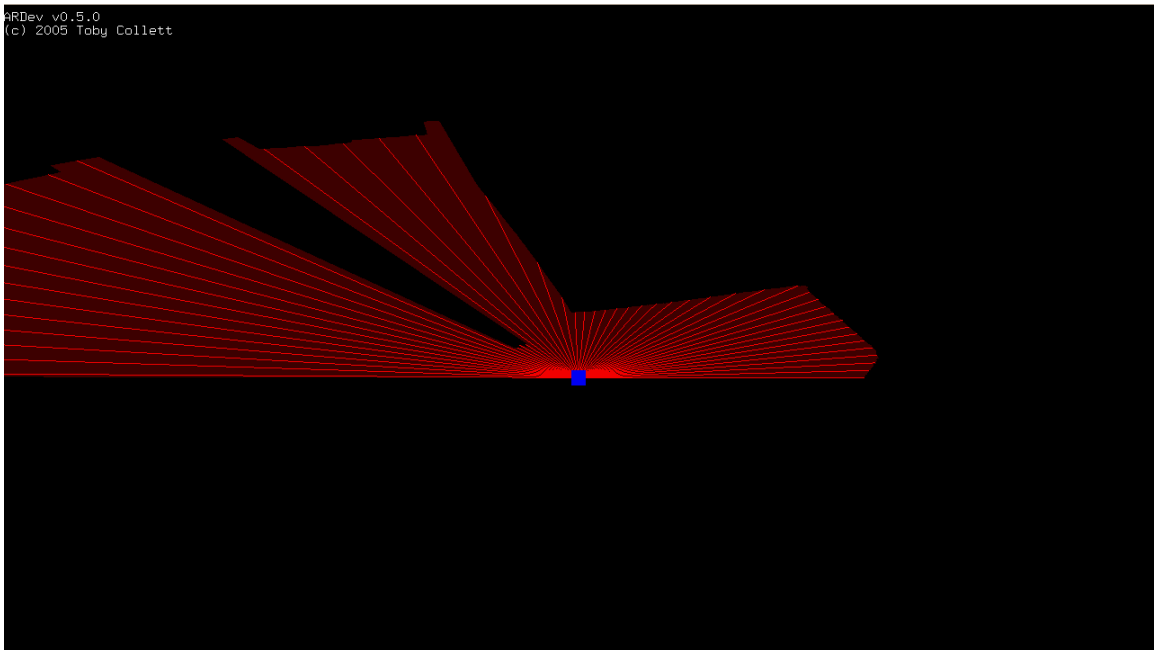
Further analysis of the performance of the system in aiding robot developers is presented in the next chapter.

5.9 Summary

This chapter has described the implementation of an IDS for robot developers. The implementation was in two stages, initially a base AR library for robot developers was implemented along with rendering modules for the Player/Stage project. This was then built upon to create a permanent debugging space with higher level tools for configuration of the system and customisation of the visualisations. The system was integrated as a Player driver to provide robot developers with a simple interface to add custom renderings to the debugging space at the runtime of their robot applications. This ability to render into the real-world using the Player graphics interfaces allows for powerful combinations of the time saving of the stock visualisations and the ability of the developer to pick out key data items to render.

A functional evaluation of the system shows it fulfils the requirements of a debugging space as defined in the previous chapter. Initial work with the system also shows that it has sufficient performance to function as a robot developer aid, an analysis that is continued in the next chapter.

(a) Real-world Scene



(b) Pure Virtual Rendering

Figure 5.22: Comparison of Virtual and AR visualisation of laser scan - Virtual Data

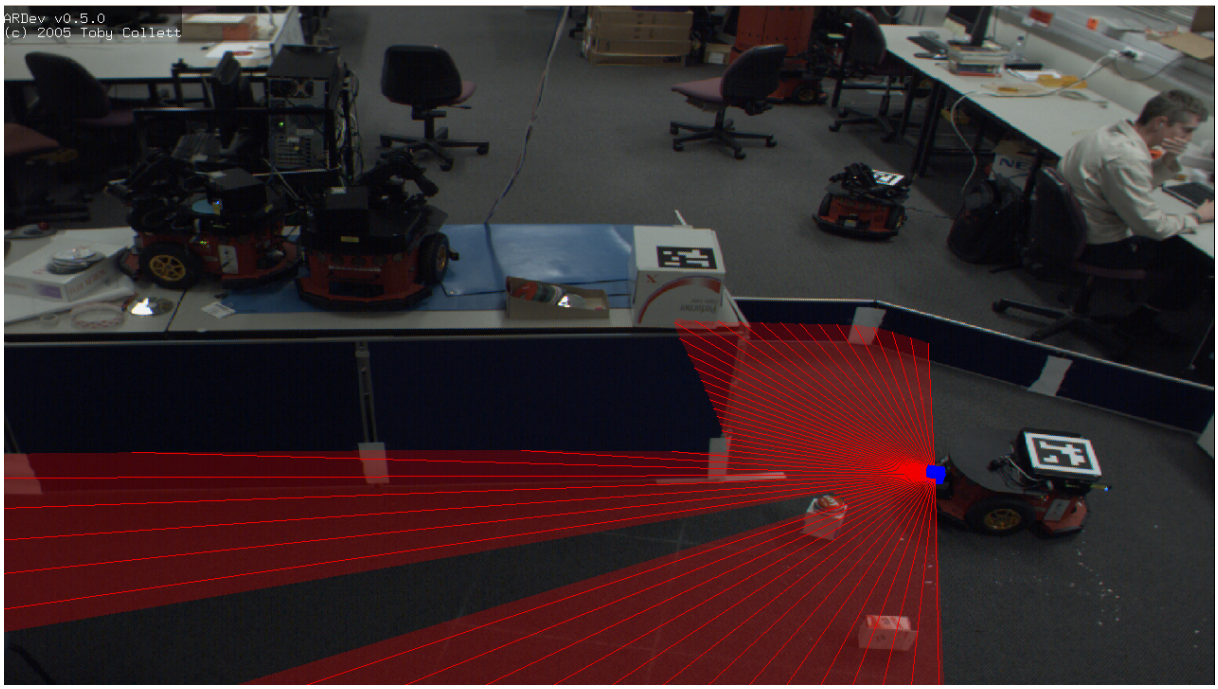


Figure 5.23: Comparison of Virtual and AR visualisation of laser scan - Augmented Data

6

Case Study Evaluation

As part of the evaluation methodology outlined in Section 4.3 a set of case studies was carried out using the implemented toolkit. The purpose of the studies was to examine the use of the AR system in standard robot development tasks, in particular whether the AR system was able to aid developer understanding of the robot's world view and direct the developer towards the source of bugs in the robot application under test.

The initial case study tasks were carried out by the author. While this may cause some elements of bias to creep into the analysis, the author is one of the target user group, robot developers, and so a valid subject for the case studies. The two key advantages of using the author are the ability to access internal thought processes while carrying out the tasks and the possibility of performing longer trials than would otherwise have been possible with the resources available.

In addition to the tasks carried out by the author a subset of the case study tasks were also undertaken by 5 independent participants. These participants 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.

6.1 Case Study Design

In general development is carried out through top down design and bottom up implementation. What this means in practice is that most implementation tasks focus on small sub tasks of the overall project. This makes the emulation of the development process simpler as testing small tasks can be considered representative of a large proportion of the implementation work.

For these case studies three tasks were chosen to test the AR system. These span a range of the standard tasks and styles of program that a robot developer is likely to use. The first of the tasks is the follower. This is an example of a simple reactive control loop where the robot has a tightly coupled loop between its sensors and actuators. The second task is a search task. This requires more complex control structures and could be implemented as a finite state machine. The final task, the block picker, is a manipulation task making use of a robotic arm to manipulate elements in the environment. The block picker was only used for the initial study carried out by the author. The combination of the three types of task, reaction to the environment, search and manipulation, is representative of many robot applications.

6.2 Test Setup

All three of the tasks were carried out with the same hardware setup. Figure 6.1 shows the Pioneer 3DX from MobileRobots [104], which is a good representative platform for the robotics community. It has the capabilities needed for the three tasks and is one of the most popular research robots.

The robot is equipped with the following additional accessories:

- Via EPIA-TC 600MHz ITX computer, running Player on Ubuntu Linux. This onboard computer also provides wireless access to the robot;
- 5DOF Arm, plus gripper, from MobileRobots [104];
- URG Laser Scanner from Hokuyo [7];
- Logitech 640x480 webcam.

The AR setup utilises a “desktop” (fixed viewpoint video see-through) AR configuration. Figure 6.2 shows a plan view of the test area. The AR system has the following components:

- Prosilica High Resolution (1360x1024) Firewire camera;
- Samsung 50” plasma display;

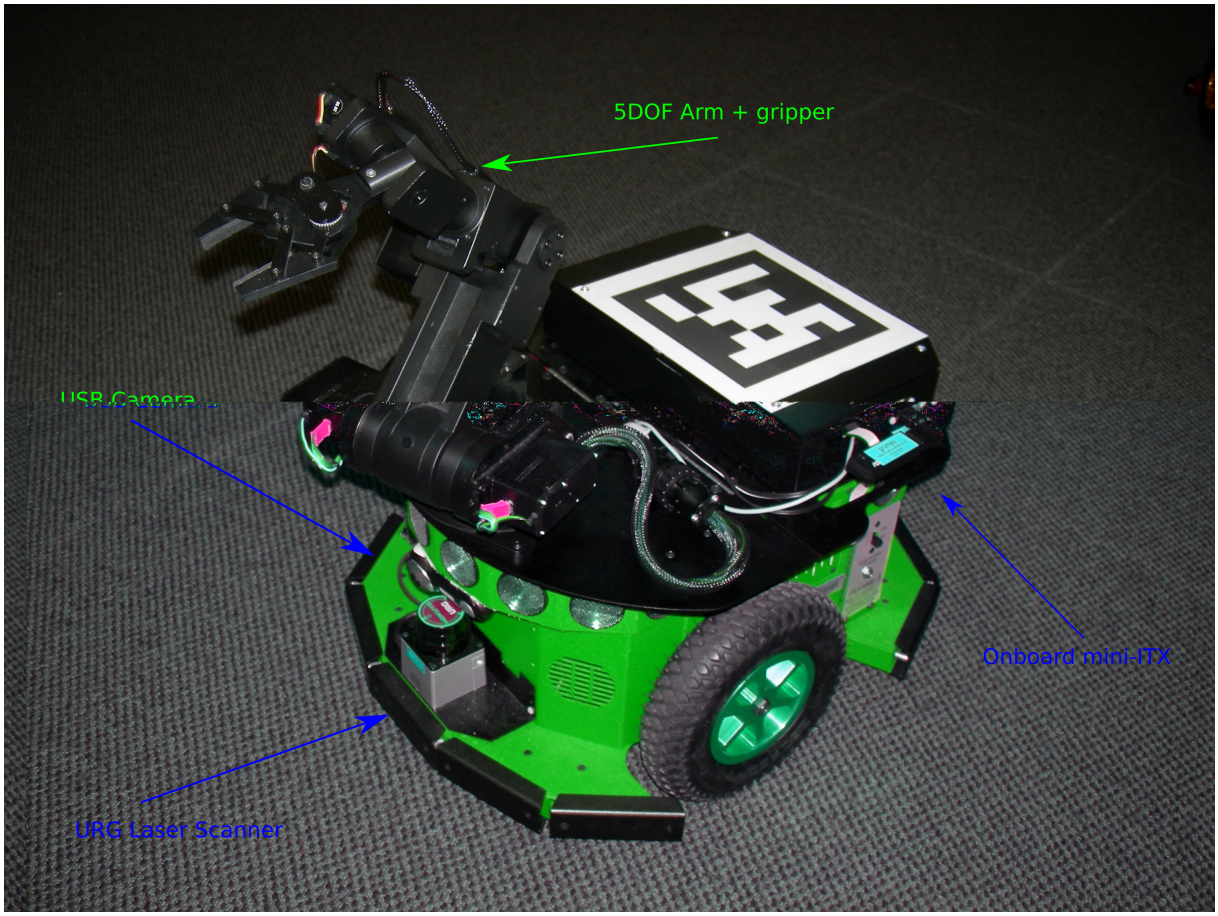


Figure 6.1: Pioneer 3DX Robot

- Pentium D PC for Image processing and Rendering.

The developer workstation also consists of a Pentium D computer with dual 19" LCD displays.

6.3 Case Study Tasks

The tasks selected for the case studies are chosen to cover a range of robotics tasks including sensing, object detection, navigation and manipulation. The tasks are designed to represent small components of larger robot developments. The design of each task will be described here with the results of the trials in the following sections.

6.3.1 Follower

The follower task represents a simple reactive control loop. The task is almost trivial from a design point of view. However this does not mean that the implementation should be issue free. The simplicity of the task also allows confirmation of the basic functionality and fault finding ability of the AR visualisation.

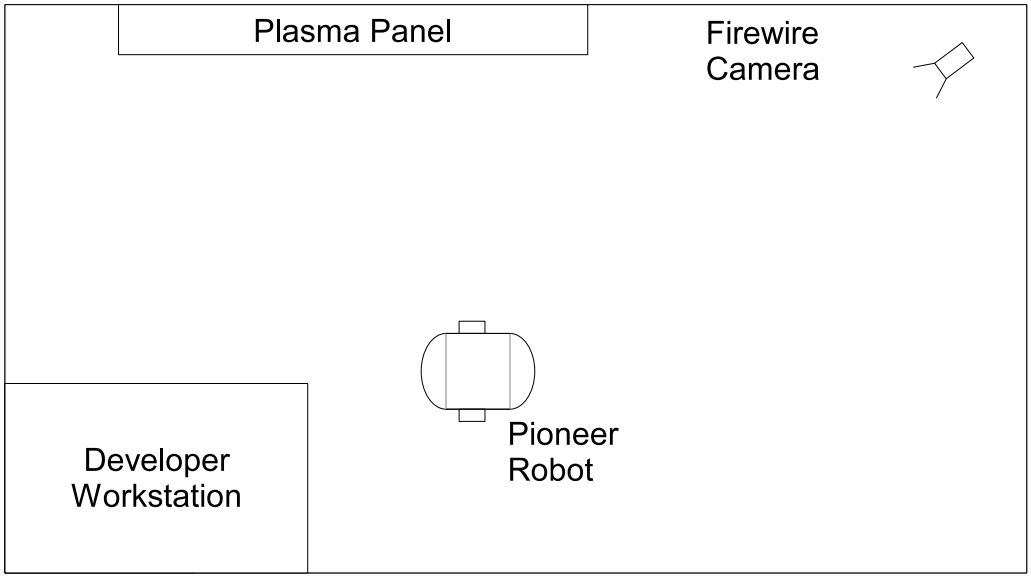


Figure 6.2: Intelligent Debugging Space

For this task the robot uses the laser scanner to detect and approach the closest obstacle, stopping a safe distance from the object. This creates the desired ability to follow the closest object. Given the simple nature of the task it was carried out directly on the robot without first simulating with Player's simulator, Stage.

6.3.2 Blockfinder

The blockfinder is representative of search tasks, it is also a task primarily concerned with a 2D representation of the world. In this task the robot searches through the environment for a coded block as shown in figure 6.3. Earlier experimentation with the cameras showed they suffered from motion blur in low lighting conditions such as those of the test area. So the robot needs to be stationary to reliably identify a marker. The laser was used for the selection of potential targets and the camera was used as a secondary sensor for identifying the markers.

The task can be broken into these stages:

- Random walk while avoiding obstacles, scanning for potential objects. This continues until a potential target has been found and a minimum time has passed. The minimum time ensures the robot has moved on from the last potential target;
- Orientate towards the target;
- Approach to within a fixed distance of the target, the fixed distance is set to something appropriate for the camera to identify the marker, and depends primarily on the marker size;

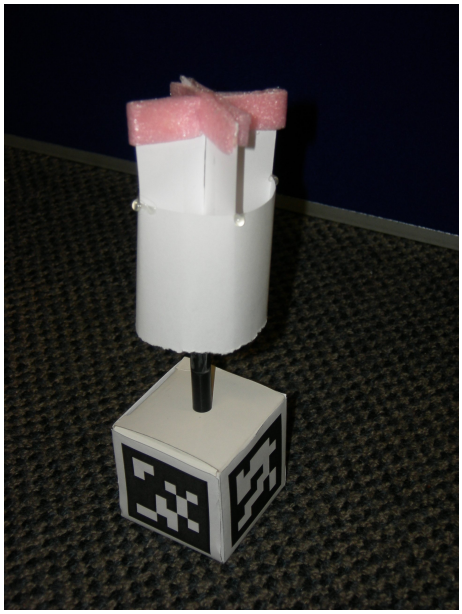


Figure 6.3: Block used in block finder and block picker tasks

- Pause to allow the camera to adjust to lighting conditions;

- Check the blobfinder Player interface to see if the potential target has identifiable markers and can be confirmed as an actual target. If it is the target, end here, otherwise return to the random walk stage.

This task fits the finite state machine model and so can be implemented with a simple loop over a switch statement in C/C++. To better imitate the normal robot development cycle the application was initially developed with a simulated robot using the Stage simulator. Once the task worked in the simulator it was tested on the real robot using the AR system for debugging any further issues. The independent participants all developed on the robot directly without first using the simulator.

The arena was setup with two objects, the target block, and another object with a similar laser profile. The robot was started from the middle of the area, facing the false object. This setup is shown in Figure 6.4.

For the second round of the case study using the independent participants the block finder task was modified slightly. The camera had been removed from the robots and so two cylinders of differing diameters were used as the targets. This changed the task in two ways, first the objects had to be identified by size and secondly the objects could be identified from a distance whereas the coded block needed to be approached before it could be identified. The core of the task remained the same.

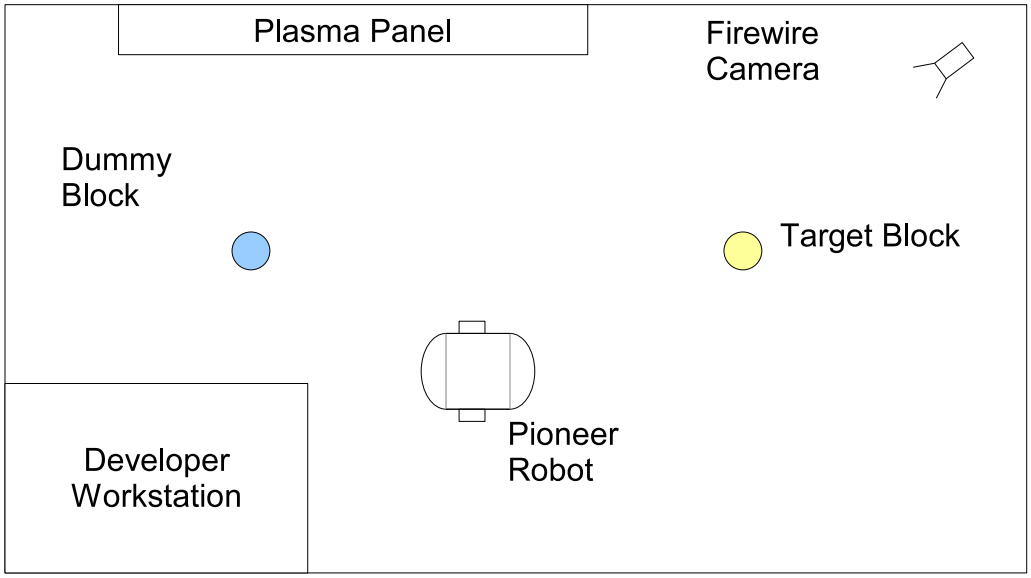


Figure 6.4: Block Finder Task Setup

6.3.3 Pick Up the Block

The final task is concerned with the manipulation of objects in the environment. The task is a 3D task using the 5 DOF arm on the pioneer robot. While the design of the task is simple, consisting of a linear sequence of actions, the 3D nature combined with the non-intuitive geometry of the arm has the potential to provide difficulties for the developer during the implementation of the task.

In this task the robot picks up a block that is placed in a suitable position in front of the robot, within its arm reach. The block is then "stored" at a predefined position relative to the robot, such as the robot's back or off to one side. This task could be combined with the block finder to find and retrieve a block from the environment, as a foraging task.

As the cameras are uncalibrated, the laser scanner identifies the location of the block in front of the robot. Open loop control of the arm is used to pick up the block and drop it at the defined location. This task was not simulated as Stage is a 2D simulator and does not model the Pioneer arm.

6.4 Initial Case Study Results

The results of carrying out each of the case study tasks with the author are presented below, the source code at the conclusion of each trial is presented in Appendix A. The results of the tasks with the other participants are presented in the next section.

6.4.1 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, see Figure 6.5. The custom visualisation was rendered using the *graphics3d* interface.

During development, the robot initially turned on the spot continually, an examination of of the AR output by the developer immediately showed that the robot was miscalculating the orientation of the closest object, as shown in Figure 6.5 (a-f). This fault was quickly tracked down to a bug in the Player client library where the minimum and maximum angles of the laser scan were inverted. Once this was corrected the follower application functioned correctly as shown in Figure 6.5 (g-i).

The inverted laser angle was difficult to detect in the numerical value of the calculation as qualitatively the value seemed correct. Specifically 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. This could easily lead to the error being passed over initially and wasted debugging effort being spent examining good portions of code.

6.4.2 Block Finder

The core component of the block finder is the identification of potential targets in the laser scan. This 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 6.6. In this visualisation 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 6.7. 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.

Once the application was functioning in simulation it was tested on the real pioneer. During testing the stock laser visualisation was used alongside the custom graphics2d

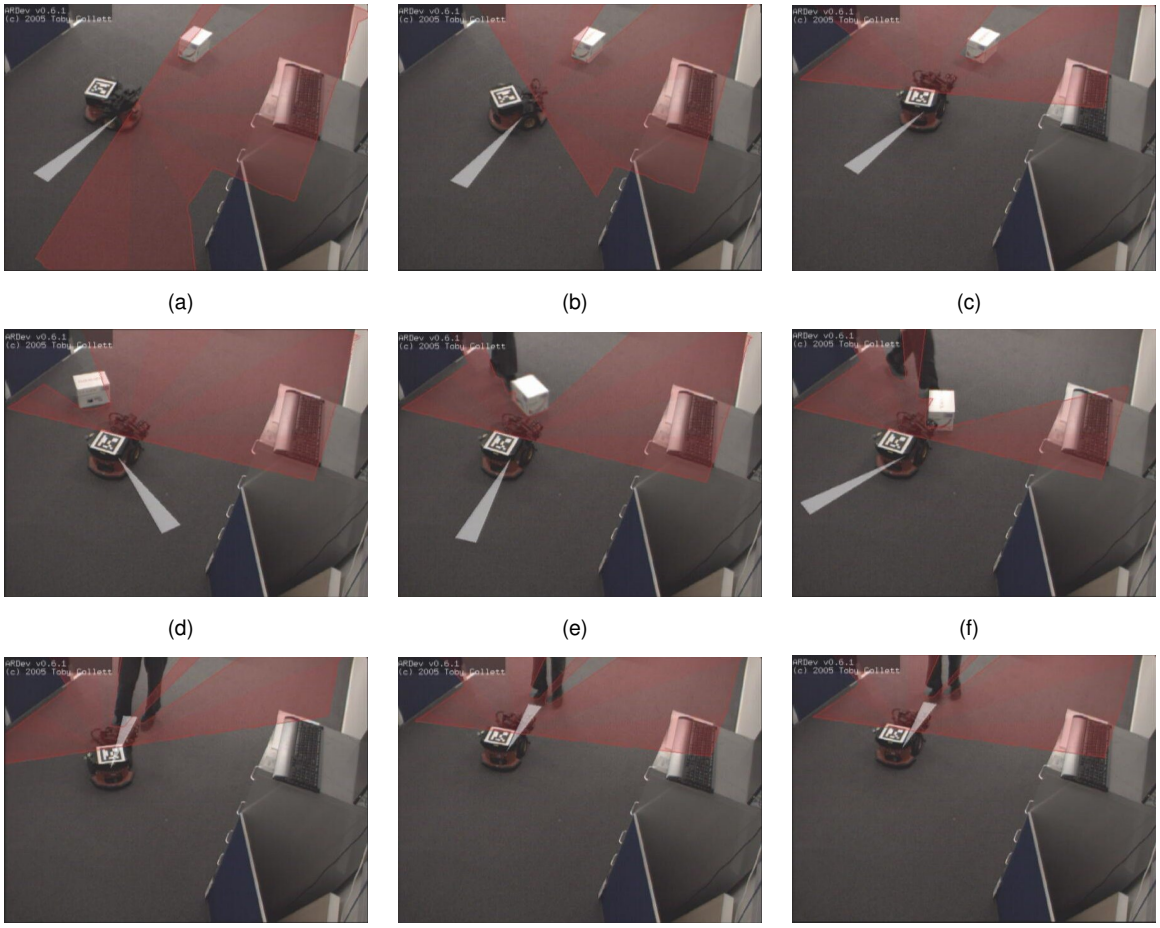


Figure 6.5: Screen captures from the follower trial. Figures (a-c) show a sequence of the initial follower. Figures (d-f) show a sequence when movement was disabled. The final three images (g-i) show the follower after the bug in the Player library was fixed.

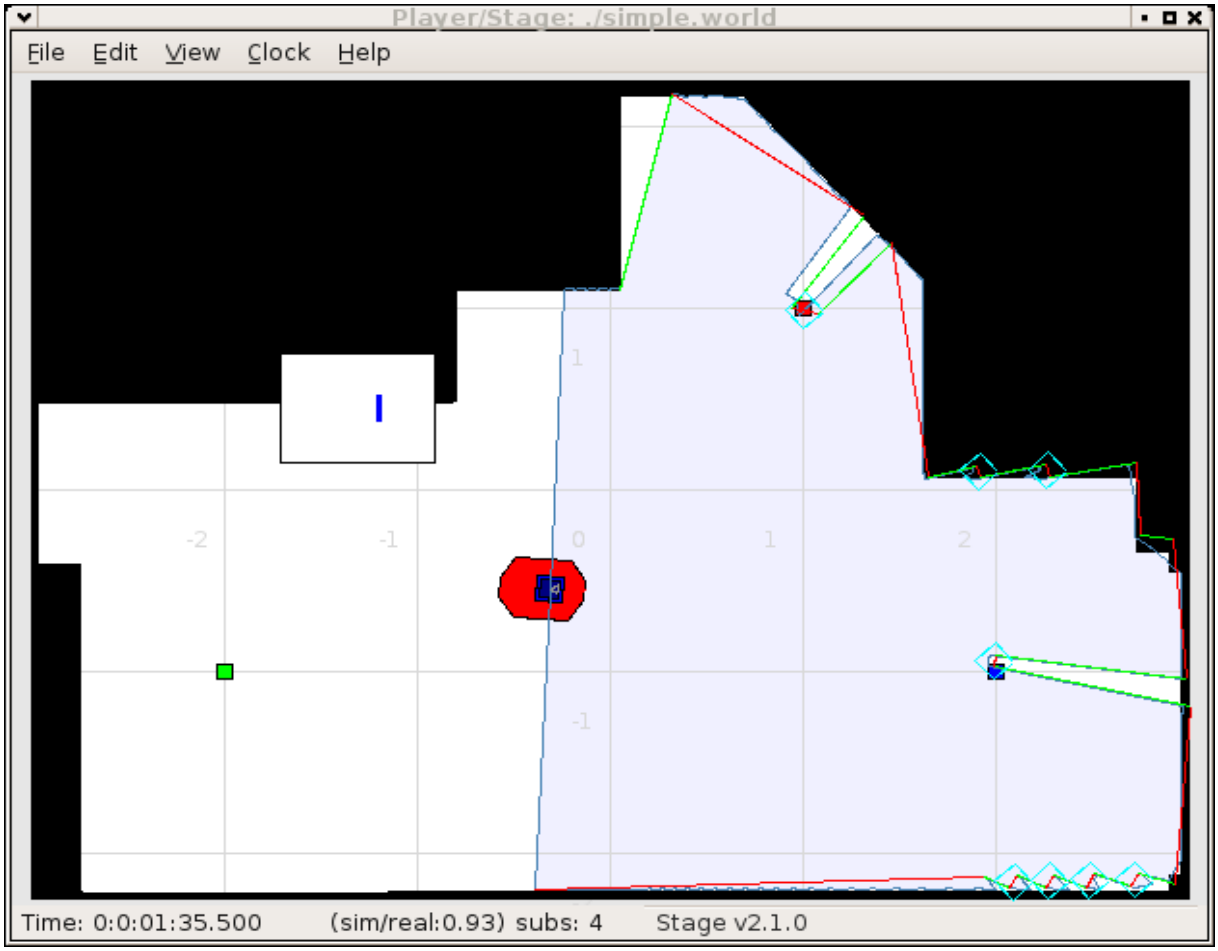


Figure 6.6: Simulated block finding task with visualisation. The alternating green and red lines show the segmented laser scan with potential blocks highlighted with a cyan diamond.

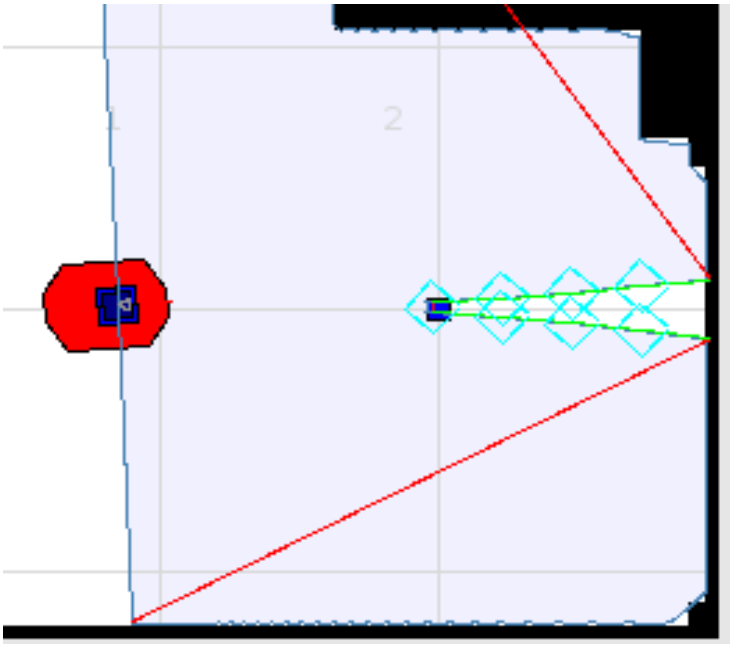
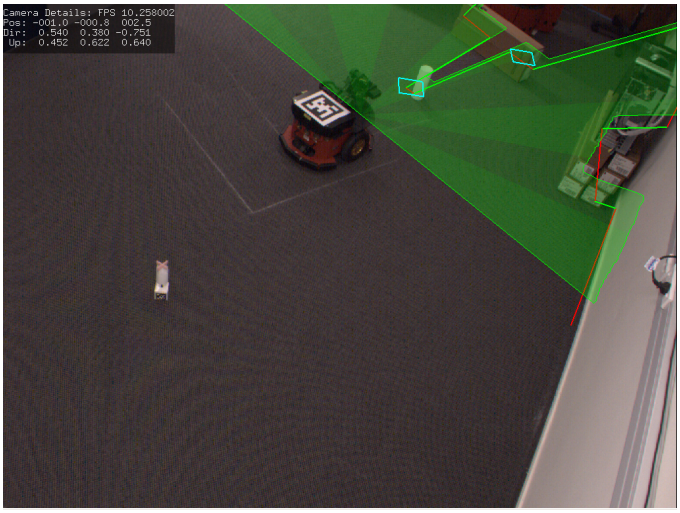


Figure 6.7: Errors in simulation caused by interpolation of laser scans, additional discontinuities (marked with cyan diamonds) were identified.

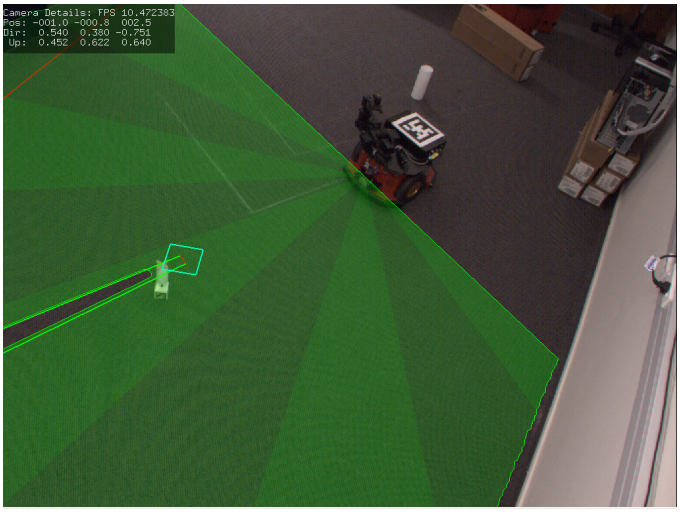
visualisation that was developed during simulation. Figure 6.8 shows the AR system output while the trial was running.

The reflections of the developer are presented below in chronological order.

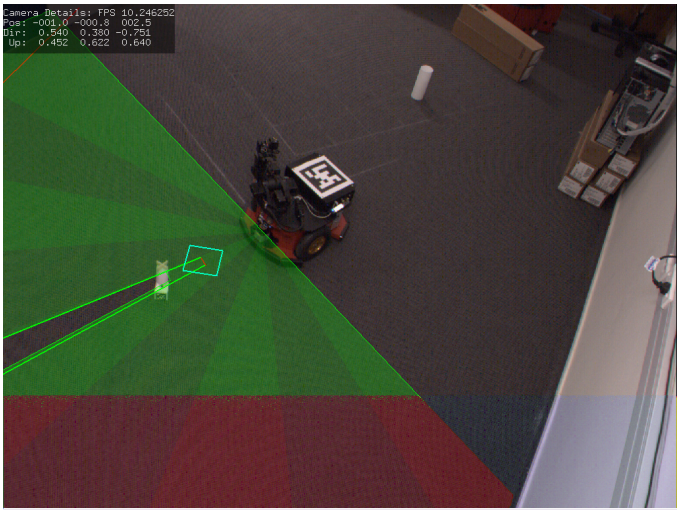
- In the first test the Robot did not move, the motors had to be explicitly enabled on the real pioneer;
- The AR system loses tracking when the robot arm obscures the fiducial on the back of the robot;
- The Robot gets stuck in corners, this is a limitation of the obstacle avoidance behaviour. This could be resolved by building in randomness, however this was omitted for this trial and subsequent test runs were restarted if the robot became stuck;
- A bug was fixed in the visualisation library code where commands were being over written causing only half the visualisation to be rendered at times;
- On one occasion the developer noted that the robot appeared deadlocked in align mode. No obvious reason was apparent, it was correctly pointing in the direction of 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;



(a) Potential targets



(b) align with target



(c) at goal

Figure 6.8: AR visualisation used with the block finding task. The red and green lines show the segmented laser scan and the cyan diamonds the potential targets.

- 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 and the test was concluded. In general the results of this task are positive but are limited in scope because of the success of the simulator in catching bugs at an early stage of the development. There are still some important points to make.

The visualisations used for this trial were either stock visualisations which were part of the toolkit, or unmodified custom visualisations used with the simulator. This means that for the developer there was no overhead involved in using the AR visualisation system.

It was also found that the visualisations helped to rule out possible causes of bugs. In the case where the robot deadlocked in align mode, and also where it failed to identify the marker, the visualisation showed that the robot was facing the potential marker and the laser scan was still successfully segmented. This allowed for an informed hypothesis to be generated about the cause of the fault, which is particularly important as the events were not likely to be easily repeatable.

One of the limitations found during this trial was the tracking of the robot's position. As a single camera is used 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.

Finally, during the trial an offset in the segmented laser data was noted. The geometry calculations were not taking into account the laser base offset from the robot origin when generating the segmented data points. During the trial this was treated as unimportant since the offset was minimal in relation to the task that was being performed. It was realised by the developer some time after the trial that this offset would need to be removed before moving on to the third task where manipulation of objects in the environment was needed.

6.4.3 Pick Up the Block

The pick up task is a simple task in concept, its implementation requires locating the block using the laser scanner, and then picking up the block with open loop control of the robot arm. The steps for the implementation are as follows:

- Calculate the block location and determine a pre-pick position a few centimetres away from the block along an approach vector;

- Move to the approach location;
- Move to the pick up location;
- Close the gripper;
- Move to the drop location;
- Release the block;
- Return to home position.

Nothing in the implementation itself is a particularly difficult programming task, however the 5 DOF arm is not something that is intuitively understood.

Prior to undertaking the trial the stock 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, as shown in Figure 6.9 (a-b). Once this was corrected the visualisation showed that the match between the arm and the reported data was reasonable but not perfect, mainly due to the accuracy of the servo joints (Figure 6.9 c-d).

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 documentation was amended to be more explicit about the coordinate system origin.

Despite the developer noting the importance of including the laser base offset in the previous trial the error was repeated in this trial, but was quickly identified when displaying the calculated position of the block.

The initial plan for picking up the block was to approach from a horizontal direction. However this generally led to a target pose that was not reachable by the arm. The developer amended the approach vector to start from above the block. This required a small modification to the block (the fins on the top were thickened) to facilitate gripping from above.

After the developer changed to the vertical approach the actual pick-up was still attempted from the side, so this was corrected. The arm then attempted the correct manoeuvre but was aiming too high for the block. The developer noted that a stereo

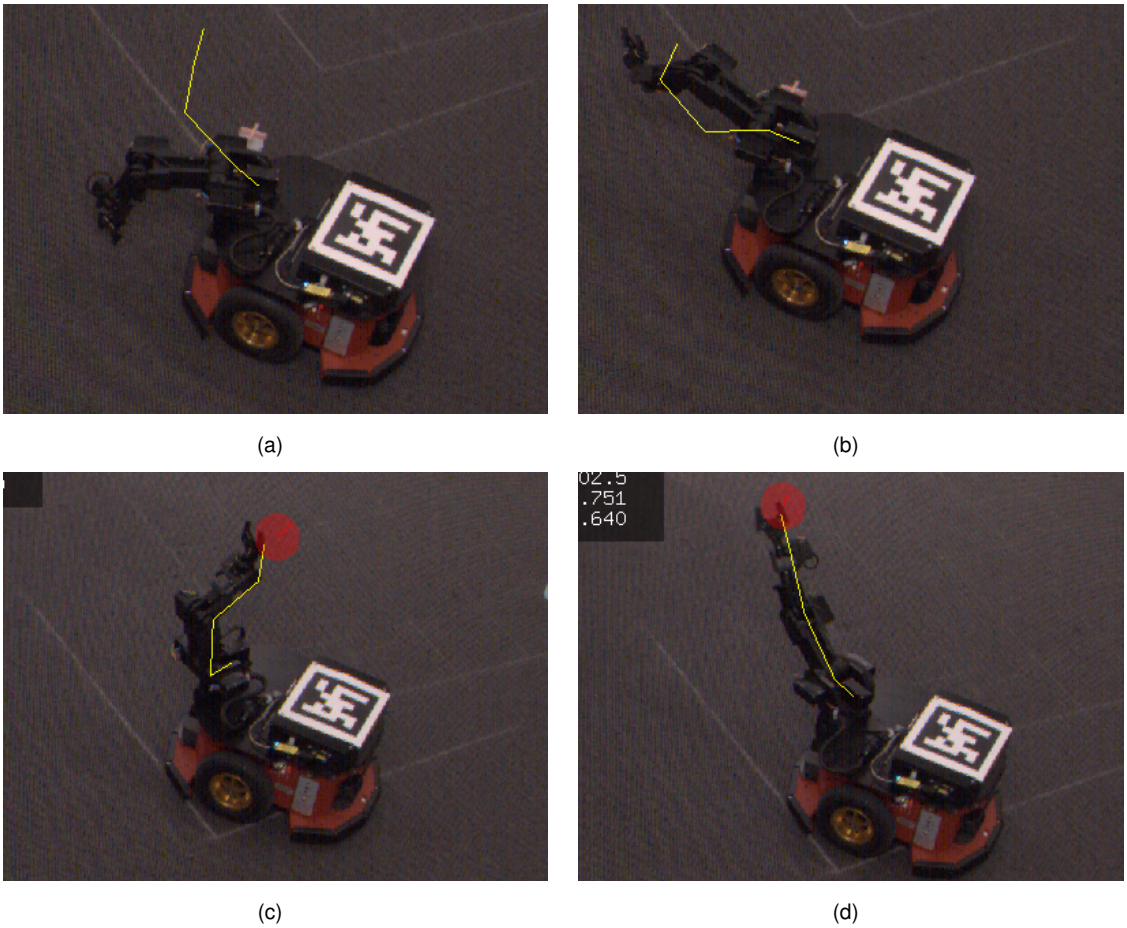
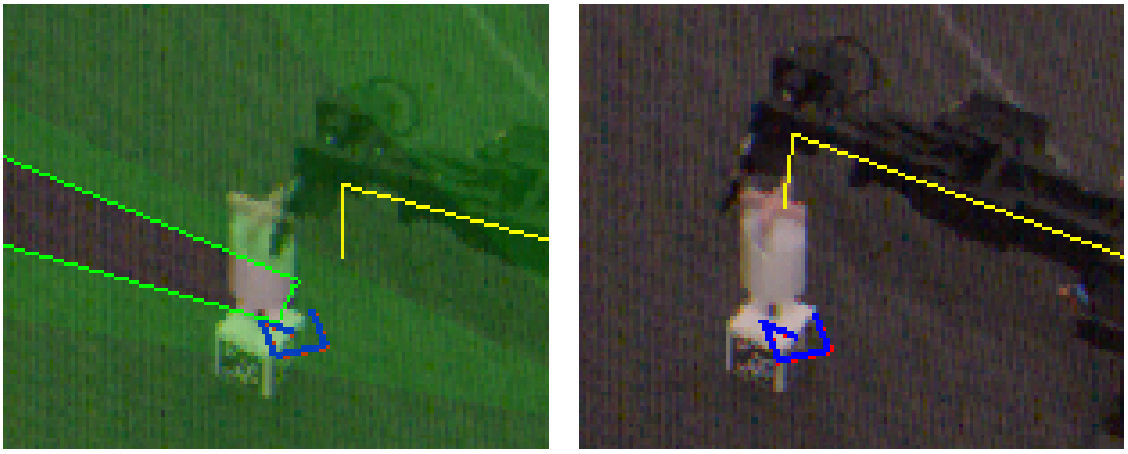


Figure 6.9: AR visualisation of the actuator array. Figures (a) and (b) show the visualisation of the incorrect axes values with Figures (c) and (d) showing the values after the configuration was corrected.



(a) Misalignment of arm (b) Shown without laser

Figure 6.10: Error between actual and reported orientation of end effector causes the end effector pose to be incorrect for pickup.

or first person view of the data would have helped at this stage to determine the exact position the arm was targeting.

The calculated position of the block was based on the closest point in the laser silhouette, however with the above grasping approach this should be the centre of the block. The block was modified slightly, adding a paper cylinder around the fins, to give a more regular silhouette and the developer modified the algorithm to calculate the centre position of the block.

The arm was still having trouble picking up the block, however the visualisation indicated this was due to mis-alignment between the arm's actual and reported position. This seems to be a limitation in the accuracy of the arm, possibly augmented by slight errors in the measured lengths and positions of the arm segments, see Figure 6.10.

Following trial attempts show around a 5–10% success rate with the pick up operation, however the failures appear to be due to limitations in the arm. Again a stereo or first person view would help show arm errors. The trial ended at this point as greater accuracy was not able to be achieved with open loop pickup operations with the pioneer arm.

6.5 Participant Case Study Results

The participants all undertook the follower and modified block finder tasks as described above. While no formal performance metrics were recorded all the participants completed the tasks between two and three hours, although a number of interruptions and some difficulties with the wireless 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.

6.5.1 General Participant Notes

This section presents a summary of the notes from the tasks and interview for the participants. The comments will be collected together based on common themes in the results. Each participant will be identified as P1, P2 etc.

P1 commented that the custom renderings were more work than desireable to create. 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, such as crosses and arrows, directly. P2 and P4 also recommended some sort of library support for symbols.

Several participants (P2, P4 and P5) also commented on the lack of text support, something that could be resolved relatively easily with an extended rendering interface.

P1 commented that it was useful to have the AR visualisation to see what the program was doing at each step, giving earlier feedback on some parts of the code which otherwise would have only been tested properly once the application was mostly finished. P5 also commented that the AR system allows them to interact with the system while viewing the graphical output which allowed for more of the system to be tested with a stationary robot.

P1 commented on having to spend time debugging the debug visualisation, in a similar way to getting debug output wrong in a print statement. Given the increase in complexity of rendering a graphical output the chance of creating a bug in the graphical debug is also likely to be higher than with print statements. On a related note P4 commented that they did not trust the visualisation initially and would look at the visualisation code before their application code for errors. In the second task P4 began to trust the visualisation to highlight issues in their application code.

Several participants commented that AR was unable to tell them quantitative information such as the size of objects, or orientation of axes. P5 suggested that a protractor and ruler tool in the AR environment would be useful, P4 also had some trouble deter-

mining the direction rotation was measured in. P1 also commented on the need for a ruler tool. This need for absolute measurement was highlighted in the block finder task where the targets were identified by their diameter. The lack of scale, combined with the lack of text support that was already mentioned, meant that the participants had to use print statements to debug their absolute size measurements.

P1 noted the lag in the AR system sometimes made it difficult to see what was happening in fast motions. 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, this was commented on by P2 and subsequent participants were warned of this in advance. P2 was asked specifically if the lag in the system was an issue but did not find this to be the case.

P1 commented that stopping the robot made debugging of sensor processing code much simpler. This was something that was found by the author as well. Whether this is an issue with an isolated data view is not clear and is something that would need further studies to determine.

Most of the participants had a small amount of confusion with regard to the different coordinate frames in use. The custom renderings were based at ground level from the origin of the robot. The laser data was reported in the laser coordinate frame which was slightly forward and above the robot coordinate frame. The offset could be accounted for by reading the geometry of the laser device from player, however the vertical offset was only able to be accommodated by switching to the graphics3d rendering interface. P5 was the only participant to use the 3D interface. The other participants simply interpreted the height offset themselves, P1 specifically commented on the fact they needed to do this. A likely reason that the majority of participants used the 2D interface was that the example rendering code given used the 2D interface.

P2 commented that they used the 2D interface as they were used to the interface from their work with graphics2d renderings in the Stage simulator. As a side note future versions of the simulator will support the graphics3d interface making this a suitable interface to standardise on.

Not accounting for the offset in coordinate frames is a common error with the pioneer robots. The laser reference frame is only a small amount in front of the robots origin (in the order of 20cm) so many applications will run acceptably even without taking account of this offset. However this makes these applications less portable and more difficult to maintain as values such as stopping distances often have arbitrary constants added to them in order to account for the data offset.

P1 commented that some spikes in the laser data were not shown in the AR view. The probable cause of this would be the slower refresh rate of the AR system and the laser data. No other participants had any comments about this. P3 did have a comment that

it was difficult at times 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.

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.

P2 commented on colour selection of the custom renderings when overlaid on the base laser scan. This is an important note to keep in mind as the base laser colour was different for each robot meaning the developers visualisations would not be as effective if a different robot was used in a subsequent trial. This ties into the possibility of supporting rendering themes as discussed in 5.7.

P4 noted that the graphical rendering allowed faster comprehension of the AR data set allowing them to watch the data in realtime rather than examining execution logs at the completion of a test run. P4 also commented that the AR renderings would be of use when transitioning from a simulation environment to the real world as the same renderings could be used. P4 also used the AR view to aid in understanding of the base platform (this was their first experience with the Pioneer robot), in particular the limitations of the laser scanner which has some trouble reading from highly reflective surfaces.

P5 noted that reading text numbers from a stream of data on screen is error prone. One error they experienced was related to mixed reading of the value 256 and 265 being read as the same value. The difference between these values is obvious when they are rendered graphically.

P5 also noted that they were having difficulty with the second task until they remembered to utilise the custom rendering functions of the system which immediately highlighted some of the errors that were occurring.

P3 made no use of the ability to render custom visualisations. When questioned about this in the interview P3 stated that they did not feel they needed the extra feedback and that print statements were faster. P3 did note that they found value in the stock renderings provided by the IDS. The coding overhead of using the AR system in P3's case was close to zero due to the decision to not use the custom renderings.

P3 primarily utilised the AR system to confirm the operation of the base hardware platform and for examining the shape of the sensed data scan. P3 noted that it was useful to see if objects were being detected at particular locations, for example when very close to the robot. It is likely that much of the benefit of the AR visualisation for P3 could have been achieved with isolated data visualisation.

6.5.2 Errors Located With AR System

The AR view allowed P1 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.

P2 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 by P2 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 P2 identified a bug where the robot would incorrectly identify the wall as an object of interest. This was highlighted by the AR visualisation, P2 commented this was easier to see visually.

P4 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 P4'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.

6.5.3 Limitations of the Participant Study

Some interesting comments can be made in regard to the participant study. None of the participants followed any structured development process. This means that many issues that should never have made it into code tested on a live robot had to be debugged with the live system. There are likely several reasons for this including;

- the scale of the tasks, programmers in general tend not to use structured processes for very small applications,
- the environment the task was undertaken in, with no long term use of the code planned issues such as code maintenance were not relevant and so there was no emphasis on code quality,
- the artificial nature of the tasks,
- the relative inexperience of the programmers involved, several had not been involved in any large scale programming projects and had minimal experience with commercial development environments

This was also seen in the use of other tools when debugging the tasks. No participants used 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. This was partially due to the build environment they were presented with and the nature of the study (using the AR system). However participants were instructed they could use any tools they wished to and there was no specific discouragement of the Stage simulator.

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. An alternative approach to collecting data for these studies would have been a think out loud exercise. Given the intensive nature of programming tasks it is likely that this would have been as difficult to extract data with and potentially distracting to the programmer.

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 a commercial company.

6.6 Discussion

Although The trials were relatively simple in nature, they represented aspects of real robotic applications. In a larger application many of the implemented sub-components would likely be of similar size to these trial cases. While qualitative metrics were not obtained with these trials some important observations were made.

First, by making use of the stock visualisations for Player the developer was able to minimise development effort expended on creating new visualisations. The custom visualisations used the *graphics2d* and *graphics3d* interfaces to render important pieces of meta-data. Another key point is that by using *graphics2d* the effort was further reduced since the same visualisations could be used for testing both in simulation and with the real robot.

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 this data is 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.

The blockfinder trial had very few errors when moving from simulation to the real world. 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.

Once the application was moved to the real robot the same visualisation was provided via the AR interface. This 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.

The final trial was carried out without the simulator. Before it had even begun several axis inversion errors were noticed in the arm driver. These errors had again passed casual inspection as the values seemed right as a 30 degree joint angle looks the same as a -30 degree joint angle if the orientation axis is not checked.

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 around a 5-10% success rate. If increased performance was desired either an improved arm or closed loop control would be needed.

In all three trials abstract data such as current state or progress was rendered to the console using plain text output. The added effort of rendering this 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 this 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 this data would be easier to understand if rendered in the virtual view.

A desire for an immersive 3D view of the data was felt by 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.

In general the independent participant's comments fit well with the comments found by the author while performing the tasks. One of the important differences was in the effort required to create the renderings. The author had greater familiarity with the rendering system and so the effort to create the visualisations was not as high as for the other participants. This is something that could be resolved through a more powerful graphics interface allowing the rendering of useful primitive graphical objects.

Also the participants noted the need for absolute measurement in the rendering. This did not get noted by the author due to the slight variation in the block finder trial which identified the blocks based on size in the participant trial.

Finally 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.

One of the key observations of these trials relate to the effort required to use the AR system. While several of the participants commented on the amount of code needed to generate the visualisations, this effort was only needed when explicitly creating a rendering which they wished to see. The base overhead of using the AR system when not creating custom visualisations, for example for P3, was almost zero. Additionally any effort creating visualisations for simulation can be reused at no cost in the AR environment.

Where the system may have a negative effect on developer performance is 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 targetting has been resolved.

6.7 Summary

The developer effort required to use the AR system was directly related to the extent of the features the developer intended to use. In the case of P3 this was simply the stock visualisation and no additional effort was needed. While several of the participants commented on the time spent creating and debugging their visualisations they also commented on the ability of the visualisations to aid their debugging once they were implemented. Additionally some small enhancements to the graphics interface could greatly reduce this implementation overhead.

Thus the use of stock visualisations allows the developer to choose an appropriate trade off between visualisation effort and the amount of custom visual information displayed. This is further enhanced by the ability to reuse the visualisations created for the simulation environment.

In particular in these trials the AR visualisation aided in:

- Finding errors in data sets that appeared correct at first glance (e.g. inversion, offset and mirroring errors);
- Identifying differences between fundamental errors and parameter tuning;
- Easing the transition from simulation to real environment.

These case studies only touch the surface of the potential of AR for developers. To really get the full benefit of this powerful tool future work will need to carry out detailed user studies with multiple participants using the system for an extended period on real robotics applications.

7

The Player 2.0 Distributed Framework

One of the fundamental requirements of the AR visualisation tool presented in this work is that a set of standard interfaces to the robot data is available. Throughout the work this interface has been provided by the Player/Stage project [127]. The AR visualisation system initially used the Player 1 series. A number of limitations in the 1.6.x releases, which are discussed in detail below, led to the author to become heavily involved in the development of Player 2, which was then used as the basis of new versions of the AR system. Supporting Player gives two benefits; integration with the most popular open source robotics system and the ability to provide stock visualisations, reducing developer workload.

In addition to the set of standard interfaces Player also provides the AR system with the ability to connect directly to a robot platform without using the developers code. This has two key benefits; continuity in rendering through restarts of the application under development and the ability to communicate with the robot without requiring modifications to the application under development. Also by utilising Player as the interface to the robot platform a wide range of hardware and users was immediately supported.

The move from the Player 1 series to Player 2 occurred over a 12 month period with almost every aspect of the Player project being re-factored, rewritten and improved. This work involved a small group of key developers led by Brian Gerkey. The author's contributions will be described in this chapter along with a brief description of related changes.

An ideal developer framework would fit into the developer's existing toolbox, maximising developer efficiency while minimising the overhead of the tool itself. This ideal is seldom able to be achieved but the energy that a developer needs to expend to use a tool must be minimised wherever possible.

Given the rapid pace of development and change in the robotics community any framework that is to remain relevant for a significant period must be flexible. Given the rapid change of the pool of researchers, existing work must also be easy to maintain. So from a community oriented viewpoint the two key features that need to be aimed for are flexibility and ease of maintenance, the latter through simplicity and transparency of the developer APIs.

Player 2.0 represents a significant rework of the internal structure of Player. These changes work towards the principles of the distributed robot framework described above, with a focus on the development community's needs. In particular, the changes have focused on allowing more flexible usage than the simple client-server model of the original Player, and a rework of the internal structure of Player to provide a simpler message processing system.

The rest of this chapter presents the work carried out by the author on Player 2.0 in the context of the full 2.0 development effort. First the Player 1.6 architecture is described, followed by an analysis of its weaknesses and the motivation for the changes that make up Player 2.0. The actual changes made for 2.0 are then described and the chapter concludes with with an example of creating a plug-in driver for the new server API.

7.1 Original Player Architecture

Player 1.x is a client-server based architecture providing hardware abstraction and network transparency for robot applications. Each type of robot component is represented by a standard interface, for example laser, sonar or position, which client applications are able to subscribe to without knowing anything about the underlying hardware. The server itself contains a set of drivers which can each support a number of interfaces. Each binding between a driver and an interface is known as a device.

Devices in Player 1.x follow the UNIX "device as a file" model. Reads from a device return the current data, writes send a desired command and configuration requests can be made in a similar way to the UNIX ioctl functions. While this model is very simple to implement it has some restrictions in terms of the range of commands and data that can be provided and also it is fundamentally a polling model.

The client end of the Player framework is provided by a set of proxies. The proxies are responsible for interpreting the raw Player messages and providing a simple API to

client applications. Proxies are available in C, C++, Python and also Java through a third party library. Player is also designed to allow many clients to connect to a server simultaneously, this is a significant advantage when writing stand alone developer tools as they are able to non-intrusively connect to the robot.

7.2 Motivation for Changes

The functions that Player must fulfil for the robot community have grown since its conception, as has the range of hardware that is supported. In particular there are now many "virtual" drivers available for Player, such as navigation algorithms, that were not the core purpose of the original Player architecture.

In general the motivation for the changes in Player 2.0 stems from the needs of a diverse and changing developer community: flexibility of the system and simplicity and transparency of the APIs. Flexibility can be enhanced by moving toward a more general robot framework, and significant reworking of the driver APIs and internal structure of Player will enhance the simplicity of the development process.

Specifically there are a number of issues that the development community have found with the existing system. Some of these have been temporarily solved with "work arounds" built into Player 1.x, while several other issues remain unsolved. The following list contains specific key issues that have motivated the design of Player 2.0:

Client-Server model too restrictive: With the growing number of virtual drivers, and with a move toward large distributed systems of robots and sensors, there is a need for more general arrangements of devices on the network.

Wire data transformations not robust or flexible: In Player 1.x, the driver writer had to deal directly with network layer data marshalling issues, which led to code that is difficult to debug and maintain. In addition to this the restriction to integer formats in the Player wire structures also necessitated the use of inconvenient and inconsistent units.

Complicated driver API: The original API, while it allows for flexible lightweight drivers to be written, was difficult for people new to Player to understand and was another source of bugs. The new driver API provides a minimal set of methods that a driver needs to implement in order to process Player messages.

Single data and command types: A single data and command structure for each interface was found to be too restrictive, and many interfaces soon exhibited "work arounds", such as including all possible data (even when some was not needed), or using a discriminated union with a byte describing which data type was actually being

transmitted. The new message namespace allows for interfaces to support multiple data types, with the subtype being specified in the Player message header. This was one of the key limitations in terms of the AR system, with dynamic updates needing to be provided for the geometries of some sensors to support correct rendering.

Desire for alternative transport protocols: While TCP/IP is suitable for a large number of applications it is not ideal in every case. Other transport layers, such as JINI and CORBA, have been discussed for some time and the new Player structure allows for these to be used in place of TCP. Additionally a monolithic Player is now possible using only internal message passing (no network layer).

7.3 Player 2.0 Requirements

Kuo and MacDonald [84] list the properties of a distributed robot framework as; platform independence, enhanced scalability, development process simplification, real-time performance, integration with existing infrastructure, promotion of software reuse, programming language independence and transport independence. The requirements for Player 2.0 come from a combination of these ideal framework properties and the need to address the issues in the previous section. Specifically the requirements for Player 2.0 were:

1. Increase the flexibility of the interface specification allowing for more data types;

2. Reduce barrier to new developers;

3. Increase maintainability of code, both in the core and drivers;

4. Allow for pluggable transport layers;

5. Enhance inter-server communication;

6. Promote consistency throughout the interface specification and client libraries.

The key to these requirements was to make Player focus on its core purpose; defining a set of standard interfaces and providing a set of software components for accessing hardware and algorithms. The rest of the Player libraries need to be kept as light weight and transparent as possible. The implementation of these requirements is discussed in the following sections.

7.4 Player 2.0 library division

Player is now divided into two halves, the core and the transport layer. The Player core is divided into the core library, `libplayercore`, and the built-in driver library, `libplayerdrivers`. Currently a TCP/IP transport layer has been implemented, and this is provided by the combination of two more libraries: `libplayertcp` and `libplayerxdr`. These libraries will be discussed in more detail below.

The separation of the Player core from the transport layer is one of the key changes in Player 2.0, as it allows for more flexible use of the Player system. For example, Player can now be tightly integrated with a JINI or CORBA transport layer, or alternatively can be run as a standalone monolithic system.

In terms of the driver API the split of the transport layer also offers an enhancement as data marshaling is now the responsibility of the transport layer, not the individual drivers. Data marshaling was consistently a source of bugs in Player 1.x and particularly was a barrier for developers unfamiliar with Player.

7.4.1 Player core

The Player core library provides the core API and functionality for the Player system. This includes the device and driver classes, the dynamic library loading code, configuration file parsing and the driver registry. In Player 2.0 the core system is a queue-based message passing system. Each driver has a single incoming message queue and can publish messages to the incoming queue of other drivers, and to specific clients in response to requests. A driver can also broadcast data to all subscribed client queues. The core library is responsible for coordinating the passing of these messages and defining message syntax. The Player interface specification defines the message semantics.

The change to a queue-based message passing model vastly simplifies the driver API which aids both the requirement to reduce the entry barrier and increase maintainability. There is now a single method a driver must implement in order to communicate with the server. This is much simpler than the multiple heterogeneous queues in Player 1.x.

The Player message structure has also been altered slightly. The addressing system has been expanded to allow for inter-server subscriptions and now consists of four values: host, robot, interface, index. This 4-part address is intended to be useful in a variety of different transports. The message namespace has been expanded to two layers, with a type and subtype, which formalises a common workaround in Player 1.x of adding a subtype field to the message body. Given the 2-layer message namespace, a device can consume multiple types of commands and can produce multiple types of data. For example, configuration changes, such as a change in sensor pose relative to the robot can be pushed out by the device for consumption by any interested parties. The new message structure achieves

the goal of greater flexibility and aids in enhancing inter-server communication.

The built-in Player drivers have been separated into `libplayerdrivers`. The separation of this library is largely from the point of view of sanity; if you only wish to use external plug-in drivers you do not need the size overhead of the built-in drivers and from a distribution point of view updates to the drivers can be distributed separately from the Player core.

7.4.2 Transport layer

The work on the transport layer was carried out primarily by Brian Gerkey. The key components of this work included; the ability to plug in alternate transport layers (previously only TCP and UDP were possible), the delegation of data marshalling to the transport layer (this had previously been a major source of bugs in drivers) and the support of floating point in the basic Player structure allowing for unit standardisation (another major source of bugs).

7.4.3 New usage paradigms

Whereas originally Player was, at its core, a TCP-based device server, this usage is now just one of the options open to developers. An example of the new usage is an application that interacts with the devices directly through their message queues, without any network layer. This application might even be written in a language other than C++, such as Python or Java; bindings in these languages for `libplayercore` and `libplayerdrivers` are currently under development. More complex configurations are also possible, and will become more common as new transport layers are developed.

The most common use of Player 2.0 will likely remain a TCP-based client/server system, which is why functionality along these lines has also been enhanced. Player now acts as a distributed framework with servers being able to subscribe to each other to meet the requirements of individual interfaces. There are still some restrictions as there cannot be circular dependencies between Player drivers. Figure 7.1 shows an example Player server network, with the arrows indicating device subscriptions.

7.5 Example plug-in driver for the new API

This section describes the process of writing a Player 2.0 plug-in driver. The URG laser scanner from Hokuyo is used as a case study. Figure 7.2 shows the laser mounted on a Pioneer 3 robot.

The `urg_laser` driver is a simple threaded driver that communicates with the laser scanner through a standard USB ACM device (very similar to a standard UART serial

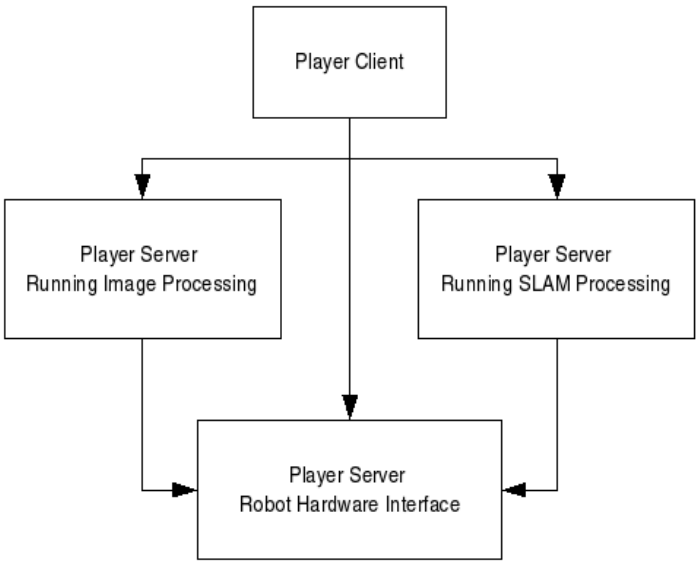


Figure 7.1: Example of potential Player 2.0 server connections

port). The main thread of the driver sits in a loop which processes any waiting messages then performs a blocking read on the device to get a laser scan update.

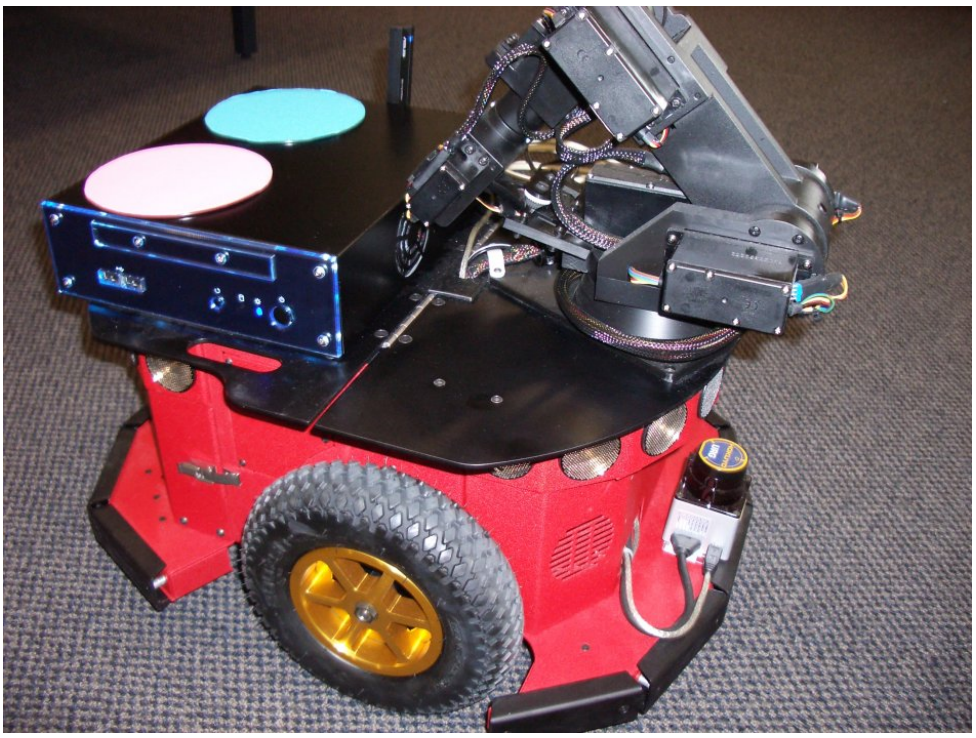
Figure 7.3 shows the class declaration for the driver. The important features are:

- The driver must inherit from the Player *Driver* class.
- The Constructor takes a *ConfigFile* parameter and an integer section parameter.
- The driver must implement the abstract *Setup* and *Shutdown* methods
- The driver re-implements the *ProcessMessage* method to provide support for handling requests and commands
- The driver re-implements Main, which will be called when the driver thread is started.

Figure 7.4 shows the code needed for `urg_laser` to function as a plug-in driver. `URGLaserDriver_Init` is a factory method that the server calls to create a driver instance and `player_driver_init` is called when the module is loaded to register the driver with the Player server core.

Figure 7.5 shows the implementation of the `urg_laser` methods. Particularly note:

- The method of reading config file parameters in the constructor (this has not changed from Player 1.x).
- The calls to StartThread and StopThread in Setup and Shutdown.



(a) URG laser mounted on Pioneer

(b) Laser detail

Figure 7.2: Hokuyo URG laser scanner

```
#include <libplayercore/playercore.h>

class URGLaserDriver : public Driver {
public:

  // Constructor;
  URGLaserDriver(ConfigFile* cf, int section);
  // Destructor
  ~URGLaserDriver();

  // Implementations of virtual functions
  int Setup();
  int Shutdown();

  // This method will be invoked on each incoming message
  virtual int ProcessMessage(MessageQueue* resp_queue,
                             player_msghdr * hdr,
                             void * data);

private:
  // Main function for device thread.
  virtual void Main();

  urg_laser_readings_t * Readings;
  urg_laser Laser;

  player_laser_data_t Data;
  player_laser_geom_t Geom;
  player_laser_config_t Conf;
};
```

Figure 7.3: Header file for urg laser driver

```
// Factory creation function.
// This function is given as an argument when
// the driver is added to the driver table
Driver* URGLaserDriver_Init(ConfigFile* cf, int section)
{
  // Create and return a new instance of this driver
  return((Driver*)(new URGLaserDriver(cf, section)));
}

// Init method called by the module loader
// need the extern to avoid C++ name-mangling
extern "C"
{
  int player_driver_init(DriverTable *dt)
  {
    table->AddDriver("urg_laser", URGLaserDriver_Init);
    return 0;
  }
}
```

Figure 7.4: Additional code for plug-in module

- The ProcessMessage method which is now the single interface point for all driver communications. MatchMessage is used to compare a message definition (type, subtype and address), and Publish is used to post responses.
- The main loop, which processes any pending messages (non blocking), then updates the device data (blocking in this case) and then uses Publish to pass the data onto subscribed devices and clients.

7.6 The Player Client Libraries

The Player 2.0 client libraries are very similar to those available in Player 1.6. This allows for a easy migration path for developers shifting to Player 2.0 as very minimal changes are needed to client source. Most of the client library development for Player 2.0 was

```
URGLaserDriver::URGLaserDriver(ConfigFile* cf, int s)
: Driver(cf, s, false, PLAYER_MSGQUEUE_DEFAULT_MAXLEN, PLAYER_LASER_CODE)
{
  // Initialise data and process config optinos
  memset(&Data, 0, sizeof(Data));
  // ...

  // read options from config file
  Geom.pose.px = (cf->ReadTupleFloat(s,"pose",0,0));
  Geom.pose.py = (cf->ReadTupleFloat(s,"pose",1,0));
  Geom.pose.pa = (cf->ReadTupleFloat(s,"pose",2,0));
}

URGLaserDriver::~URGLaserDriver()
{ // clean up any resources }

// Set up the device.  Return 0 if things go well, and -1 otherwise.
int URGLaserDriver::Setup() {
  // Start the device thread; spawns a new thread and executes Main
  StartThread();
}

// Shutdown the device
int URGLaserDriver::Shutdown() {
  // Stop and join the driver thread
  StopThread();
}

// Process an incoming Message
int URGLaserDriver::ProcessMessage(MessageQueue* resp_queue, player_msghdr * hdr, void * data)
{
  if(Message::MatchMessage(hdr, PLAYER_MSGTYPE_REQ, PLAYER_LASER_REQ_GET_GEOM, this->device_addr))
  {
    assert(hdr->size == sizeof(player_laser_config_t));
    Publish(device_addr, resp_queue, PLAYER_MSGTYPE_RESP_ACK, PLAYER_LASER_REQ_GET_GEOM, &Geom, sizeof(
        Geom))
    return 0;
  }
  return -1;
}

// Main function for device thread
void URGLaserDriver::Main()
{
  // The main loop; interact with the device here
  for(;;) {
    // test if we are supposed to cancel
    pthread_testcancel();

    // Process any pending messages
    ProcessMessages();

    // update device data
    Laser.GetReadings(Readings);
    // fill in the data structure
    Data.min_angle = Conf.min_angle;
    Data.max_angle = Conf.max_angle;
    Data.resolution = DTOR(2.0*270.0/768.0);
    Data.ranges_count = (768/2);
    for (int i = 0; i < 768/2; ++i)
    {
      Data.ranges[i] = Readings->Readings[i*2] < 20 ? (4095) : (Readings->Readings[i*2]);
      Data.ranges[i]/=1000;
    }

    Publish(device_addr, NULL, PLAYER_MSGTYPE_DATA, PLAYER_LASER_DATA_SCAN, &Data, sizeof(Data));

    // you may need to sleep here if this loop consumes
    // too much processor time
  }
}
```

Figure 7.5: Source of urg laser driver

carried out by other members of the Player development community.

7.7 Summary

The standard interface descriptions that Player provides are essential for tools such as the AR visualisation system presented in this thesis. Without standardisation developer tools are restricted to the individual project they are created for and developers are doomed to recreate these tools for eternity.

Player 2 was thoroughly tested throughout its development as part of this visualisation work. The visualisation toolkit was kept in sync with the Player CVS repository throughout development stages and used on many other projects within the University of Auckland robotics research group, including undergraduate courses.

Having an open source robot framework such as Player/Stage is important for the community as it provides a base for collaboration, peer review and standardisation. Player has come to fill this role in the community and now with Player 2.0 it has taken on board many of the features that the community has been desiring.

Player 2.0 is an important step forward for the Player/Stage project. The Player server core has been turned from a "device as a file" model to an event driven message based protocol, making Player 2.0 simpler, more flexible, easier to maintain, easier to learn and more powerful. Additional changes to the library structure have made Player easier to package and extend.

The ease of extension was essential for the AR tool developed in this thesis as the new graphics2d interface was used and the graphics3d interface created. The author also wrote numerous hardware drivers throughout the work saving significant development time and improving code quality by using the enhanced Player 2.0 interfaces.

8

Future Work and Conclusions

8.1 Future Work

This work began with a hypothesis that AR visualisation of robot data would aid robot developers. Through a prototype based evaluation it has shown that AR has significant real benefits for developers and has opened up enormous opportunities for future research. An important area of future research is to quantify how much the AR visualisation aids the developer. This is important to investigate when considering the trade off in complexity and cost of installing and utilising an AR system. Similarly the difference in effectiveness between AR configurations needs to be quantified. For example with immersive head mounted displays there are significant constraints placed on the developer; whether these are justified by the increased understanding provided is yet to be evaluated. A fixed viewpoint AR system has much lower overhead so perhaps in some situations this would be more appropriate.

Informal discussions with users trying out the system and the author's own experience suggest that the use of an HMD 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. Given these indications, much of the evaluation focused on the fixed view point AR which minimised these added complications, at the cost of not allowing an immersive experience. Work is needed to formally compare the user experience and performance with a range of different AR hardware setups including

video and optical see through HMDs.

Long term studies of developers using the system are needed to further validate and improve the system. Programming is a very complicated and variable task with great variations in performance and style being found between different programmers. As such multiple developers working on real world tasks need to be studied in depth. The wider properties of programming also need to be considered; does the AR system produce more concise or more maintainable code? Are bugs found earlier in the development cycle due to the developer being able to "just see" them even when working on other parts of the system? Are more bugs caught before the release of a product?

Another area that needs much more in-depth work is the choice of the visualisations. Is there an ideal way to visualise a laser scanner or is this application dependent? In Section 5.7 some general guidelines were presented, however these only scratch the surface of the possibilities and have not been formally evaluated. There is a wealth of information available with regard to visualisation in both the fields of information and scientific visualisation, these fields need to be plundered to determine which aspects best apply to robotic visualisation.

The interaction toolkit implemented in this thesis needs additional work to extend the ability of users to interact with the system. Currently the system is predominately a display system and interaction techniques need to be integrated to give the developer greater control over the visualisation. Work is planned to use a touch screen to enhance the interaction with the flat screen display in the debugging space allowing users to select and modify elements directly in the visualisation. This could be extended to include 3D gesture interactions in an immersive system.

AR also has potential in many other areas of HRI. An augmented view can provide users with an intuitive understanding of "native" robot data increasing the shared perceptual space of the human-robot pairing and thus enhancing interaction. The guidelines from Section 5.7 could be used a starting point for these interactions. 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. This would allow the user to immediately clarify or correct any errors rather than having to repair the fault further down the line. Again with these AR applications it generally comes down to the tradeoff between the benefits of an AR system and the cost of the additional complexity. As the AR community develops and improves the hardware and software offerings this cost will begin decreasing opening up many more opportunities.

8.2 Conclusions

This work began by outlining the challenges that face the robot developer, showing that robotics has 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;

(a) how the robot is seeing the world,

(b) any inconsistencies or deficiencies with this view.

This approach to robot development lands it firmly in the domain of HRI, leading this work to introduce a new role of user interaction, that of the robot developer. While development occurs behind the scenes of a robot system it still shares the fundamental issue of HRI, the creation of a shared perceptual space where the human and robot can understand each other. Once this understanding is created many of the difficulties of robot programming fall away.

AR provides enormous opportunities for creating one side of the shared perceptual space, allowing the developer to see the world as the robot does, intuitively displaying the limitations and discontinuities in the robot's world view. The AR enhanced intelligent debugging space created in this work allows the developer to view the robot's data and meta data in context with the environment the robot is operating in.

In section 1.3 the following questions were put forward:

- how can AR be applied to robot development?
- what paradigms for interaction are possible?
- how should robot data sets be represented?
- how can the visualisation be integrated into the developer's existing environment?
- what hardware and software infrastructure is needed to support an AR developer system?

The first two questions were explored in chapters 3 and 4. These chapters concluded with the idea of an intelligent debugging space for robot developers, which also provides a solution to the fourth question, that of integration. Chapter 5 presented a prototype software system for robot developers which supports a range of AR hardware. The exact hardware selection depends on both the environment and the development tasks being undertaken. It was found that the fixed viewpoint AR used in the case studies provided a good balance between complexity, encumbrance and providing an AR view for the tasks that were carried out. However, as mentioned in the previous section more work is needed to examine the differences between the possible hardware configurations. The

question of data representation is not able to be definitively answered, however section 5.7 presented some general guidelines motivated by the author's experience in implementing the prototype system.

The designed and implemented ARDev debugging space supports a standard set of visualisations that are available to the developer out of the box. This reduces the developer's visualisation workload to the task of rendering application specific meta data such as the key decision variables that the robot is operating with.

The ARDev system provides an opportunity to understand the types of errors that are being encountered during debugging. Debugging is essentially a process of elimination and by understanding the type of error developers can quickly eliminate large sets of bug sources, substantially reducing debugging time. If a developer makes an incorrect assumption about the source of the bug, a disproportionate effort is needed before they can rule out all the potential sources of the bug and realise that it was their initial assumption about its source that was incorrect.

To have a set of stock visualisations there must be standard interfaces available. For this reason, among others, robot programming frameworks, in particular Player/Stage, will play an increasingly important role in the future of robotics. A standard framework such as Player allows for code reuse at many levels, allowing drivers and algorithm implementations to be shared, clients to be reused with different robot platforms and, most importantly for this thesis, tools such as AR visualisation to be written once and then used by a large community of developers. As part of this research the author made significant contributions to the development of Player/Stage, most importantly as one of the key developers of the 2.0 release.

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.

This thesis has built upon the collection of development tools that have been created in an ad-hoc manner by robot developers. The analysis of the role of the robot developer that formed the starting point of this work provides an opportunity for a unified approach to developer tools targeting the developers specific needs. The novel approach of applying AR to developer interactions with robots provides the ability to meet these needs.

Through case study developments with the ARDev system the use of AR has been shown to have significant benefits for the robot developer, enhancing their understanding of the robot's world-view and hence reducing debugging time. As part of the work a

flexible AR visualisation tool was developed with close integration with the Player/Stage project. This tool creates an intelligent debugging space where developers can utilise the benefits of the AR visualisation with minimal overhead.

Finally the principle of a standard set of tools for robot developers cannot exist without projects that are supported by the wider developer community. Part of this research has included significant contributions to the Player/Stage project, particularly with the 2.0 release, producing a more efficient Player server with a simpler API for developers. The foundation of this work is the creation of effective support tools for the robot developer, a task that needs to be continued in terms of this work, Player and other tools in the community, in order to push robotics beyond the production line and into a multitude of roles in everyday life.

A

Case Study Source

A.1 Study 1: Follower

```
/*
 * follower
 *
 * application to approach the closest object, if this is moving
 * the robot will follow it
 *
 */

#include <libplayerc++/playerc++.h>
#include <iostream>

#include "args.h"

#define RAYS 32

player_point_3d_t pts[1024];

int
main(int argc, char **argv)
{
  parse_args(argc,argv);

  // we throw exceptions on creation if we fail
  try
  {
    using namespace PlayerCc;

    PlayerClient robot(gHostname, gPort);
    PlayerClient vis("gin", gPort);
    Position2dProxy pp(&robot, gIndex);
    LaserProxy lp(&robot, gIndex);
    Graphics3dProxy g3d(&vis, gIndex);

    std::cout << robot << std::endl;

    pp.SetMotorEnable (true);
    lp.RequestConfigure();
```

```
    std::cout << "laser_config:_" << lp.GetMinAngle() << "_" << lp.GetScanRes() << std::endl;

    // go into read-think-act loop
    for(;;)
    {
      double newspeed = 0;
      double newturnrate = 0;

      // index to minimum value
      int min = 0;

      // this blocks until new data comes;
      robot.Read();
      std::cout << "laser_config:_" << lp.GetMinAngle() << "_" << lp.GetScanRes() << std::endl;

      uint count = lp.GetCount();
      for (uint j=0; j < count; ++j)
      {
        if (lp[min] > lp[j])
          min = j;
      }

      double min_angle = (-(double)lp.GetMinAngle()) + ((double) min) * ((double)lp.GetScanRes());

      std::cout << "min:_" << min
                << "_min_angle:_" << RTOD(min_angle) << "_" << min_angle;


      newspeed = lp[min] < 0.4 ? 0 : lp[min]/10;
      newturnrate = min_angle/5;

      std::cout << "_speed:_" << newspeed
                << "_turn:_" << RTOD(newturnrate)
                << "\r";

      g3d.Clear();
      player_color_t colour;
      colour.red = 255;
      colour.green = 255;
      colour.blue = 255;
      colour.alpha = 128;

      g3d.Color(colour);
      pts[0].px = 0.0;
      pts[0].py = 0.0;
      pts[0].pz = 0;
      pts[1].px = cos(min_angle-DTOR(5));
      pts[1].py = sin(min_angle-DTOR(5));
      pts[1].pz = 0;

      pts[2].px = cos(min_angle+DTOR(5));
      pts[2].py = sin(min_angle+DTOR(5));
      pts[2].pz = 0;

      g3d.Draw(PLAYER_DRAW_TRIANGLE_FAN, pts, 3);

      // write commands to robot
      pp.SetSpeed(newspeed, newturnrate);
    }
  }
  catch (PlayerCc::PlayerError e)
  {
    std::cerr << e << std::endl;
    return -1;
  }
}
```

A.2 Study 2: Block Finder

```
/*
 * blockfinder
 *
 * a simple algorithm to find a block which appears as a specific blobfinder ID
 *
 */
```

```
#include <libplayerc++/playerc++.h>
#include <iostream>
#include <vector>
#include <math.h>
#include <libthjc/misc.h>


using namespace PlayerCc;
using namespace std;

typedef enum MoveState
{
  RANDOM,         // Wander randomly for fixed time while avoiding
  RANDOM_SEARCH,     // Wander randomly until a potential block is identified
  ALIGN,          // Align potential block to robot X axis then approach
  APPROACH,       // Approach to fixed distance from block then identify
  IDENTIFY,       // Pause for short time to allow camera to get good shot then check blob tracker
             // If it is valid target stop, otherwise go to random wander
  END           // we are all done
} MoveState;

MoveState CurrentState = RANDOM;

int
main(int argc, char **argv)
{
  char * RobotHostname = "localhost";
  int RobotPort = 6665;
  char * VisHostname = "localhost";
  int VisPort = 6665;
  char * BlobHostname = "localhost";
  int BlobPort = 6665;

  int BlobIDMin = 20;
  int BlobIDMax = 30;
  double BlobSize = 0.1;

  StopWatch Timer;
  Timer.GetElapsedDouble();
  double ElapsedTime = 0;

  for (int i = 1; i < argc; ++i)
  {
    if (i < argc-1 && strcmp("--host",argv[i])==0)
      RobotHostname = argv[++i];
    if (i < argc-1 && strcmp("--vishost",argv[i])==0)
      VisHostname = argv[++i];
    if (i < argc-1 && strcmp("--blobhost",argv[i])==0)
      BlobHostname = argv[++i];
    if (i < argc-1 && strcmp("--port",argv[i])==0)
      RobotPort = atoi(argv[++i]);
    if (i < argc-1 && strcmp("--visport",argv[i])==0)
      VisPort = atoi(argv[++i]);
    if (i < argc-1 && strcmp("--blobport",argv[i])==0)
      BlobPort = atoi(argv[++i]);
  }

  // we throw exceptions on creation if we fail
  try
  {
    PlayerClient Robot(RobotHostname, RobotPort);
    PlayerClient Vis(VisHostname, VisPort);
    PlayerClient Blobtracker(BlobHostname, BlobPort);

    Position2dProxy pp(&Robot, 0);
    LaserProxy     lp(&Robot, 0);
    BlobfinderProxy bf(&Blobtracker, 0);
    Graphics2dProxy g2d(&Vis, 0);

    std::cout << Robot << " " << Vis << " " << Blobtracker << std::endl;

    pp.SetMotorEnable (true);
    lp.RequestConfigure();

    // make sure we have some data from the robot
    for (int ii = 0; ii < 6; ++ii)
      Robot.Read();
```

```
// go into read-think-act loop
for(;;)
{
  ElapsedTime += Timer.GetElapsedSeconds();
  if (Robot.Peek(100))
    Robot.Read();
  Vis.ReadIfWaiting();
  Blobtracker.ReadIfWaiting();

  double newspeed = 0;
  double newturnrate = 0;

  // index to minimum value
  int min = 0;

  // find the closest laser point and segment the laser
  vector<player_point_2d_t> SegmentedScan;
  vector<player_point_2d_t> PotentialBlocks;
  player_point_2d_t Current = lp.GetPoint(0);
  SegmentedScan.push_back(Current);
  uint count = lp.GetCount();
  for (uint j=0; j < count; ++j)
  {
    player_point_2d_t Temp = lp.GetPoint(j);
    if (sqrt(pow(Temp.px-Current.px,2)+pow(Temp.py-Current.py,2)) > BlobSize)
    {
      if (sqrt(pow(SegmentedScan.back().px-Current.px,2)+pow(SegmentedScan.back().py-Current.py,2))
          < 2*BlobSize)
      {
        player_point_2d_t Avg;
        Avg.px = (SegmentedScan.back().px+Current.px)/2;
        Avg.py = (SegmentedScan.back().py+Current.py)/2;
        PotentialBlocks.push_back(Avg);
      }
      SegmentedScan.push_back(Current);
      SegmentedScan.push_back(Temp);
    }
    Current = Temp;
    if (lp[min] > lp[j])
      min = j;
  }

  double min_angle = lp.GetMinAngle() + ((double) min) * ((double)lp.GetScanRes());
  double range = (lp.GetMaxAngle() - lp.GetMinAngle());

  // speed and turn rate to approach the closest object
  newspeed = lp[min] < 0.3 ? 0 : lp[min]/10;
  newturnrate = M_PI/4 * (min_angle < 0 ? 1 : -1) *(range/2 - fabs(min_angle))/(range/2);

  // find the closest potential block
  double MinObjectDistance = 10000;
  double MinObjectAngle = 0;
  for (vector<player_point_2d_t>::const_iterator itr = PotentialBlocks.begin(); itr !=
       PotentialBlocks.end(); ++itr)
  {
    double NewDist = sqrt(itr->px*itr->px + itr->py*itr->py);
    if (NewDist < MinObjectDistance)
    {
      MinObjectDistance = NewDist;
      MinObjectAngle = atan2(itr->py,itr->px);
    }
  }

  cout << CurrentState << "␣" << ElapsedTime << "␣␣" << newturnrate << "␣␣␣␣␣␣␣␣" << min_angle << "
       ␣␣␣␣␣␣␣␣" << range << "\n";

  // clear our previous graphics overlay
  g2d.Clear();

  // render the segmented scan
  player_point_2d_t Points[5];
  Points[0] = SegmentedScan.front();
  int i = 0;
  for (vector<player_point_2d_t>::const_iterator itr = SegmentedScan.begin(); itr != SegmentedScan.
       end(); ++itr, ++i)
  {
    Points[1] = Points[0];
```

```
    Points[0] = *itr;
    if (i%2)
      g2d.Color(255,0,0,0);
    else
      g2d.Color(0,255,0,0);
    g2d.DrawPolyline(Points,2);
  }

  // render the potential blocks
  for (vector<player_point_2d_t>::const_iterator itr = PotentialBlocks.begin(); itr !=
       PotentialBlocks.end(); ++itr)
  {
    g2d.Color(0,255,255,0);
    for (int i = 0; i < 4; ++i)
      Points[i] = *itr;
    Points[0].px -= BlobSize;
    Points[1].py -= BlobSize;
    Points[2].px += BlobSize;
    Points[3].py += BlobSize;
    Points[4] = Points[0];
    g2d.DrawPolyline(Points,5);
  }

  // do our state machine
  switch(CurrentState)
  {
    case RANDOM:
      if (ElapsedTime > 10)
        CurrentState = RANDOM_SEARCH;
      break;

    case RANDOM_SEARCH:
      if (PotentialBlocks.size() > 0)
        CurrentState = ALIGN;
      break;

    case ALIGN:
      if (PotentialBlocks.size() == 0)
      {
        CurrentState = RANDOM_SEARCH;
        break;
      }
      newspeed = 0;
      if (fabs(MinObjectAngle) < ((5.0/180.0)*M_PI))
      {
        newturnrate = 0;
        CurrentState = APPROACH;
      }
      else
        newturnrate = MinObjectAngle;
      break;

    case APPROACH:
      if (PotentialBlocks.size() == 0)
      {
        CurrentState = RANDOM_SEARCH;
        break;
      }
      newturnrate = 0.2*MinObjectAngle;
      if (MinObjectDistance < 0.5)
      {
        ElapsedTime = 0;
        CurrentState = IDENTIFY;
      }
      else if (range < 0.45)
      {
        ElapsedTime = 0;
        CurrentState = RANDOM;
      }
      break;

    case IDENTIFY:
      newturnrate = 0;
      newspeed = 0;

      // wait for 1 second so we dont have motion blur in the image
      if (ElapsedTime > 1)
      {
```

```
            // find the blob closest to the center of the image
            if (bf.GetCount() == 0)
            {
              CurrentState = RANDOM;
              ElapsedTime = 0;
              break;
            }

            double Width = bf.GetWidth();
            double Height = bf.GetHeight();
            double bestx = 0.5; // only match if with center half of image
            int best_id = -1;
            for (unsigned int i = 0; i < bf.GetCount(); ++i)
            {
              // normalise, 0 = center
              double x = (static_cast<double>(bf[i].x) - Width/2)/Width;
              if (fabs(x) < bestx)
              {
                bestx = x;
                best_id = bf[i].id;
              }
            }
            if (best_id >= BlobIDMin && best_id <= BlobIDMax)
              CurrentState = END;
            else
            {
              CurrentState = RANDOM;
              ElapsedTime = 0;
              break;
            }

          }

          break;
        case END:
          newturnrate = 0;
          newspeed = 0;
          break;

        default:
          // we should never get here
          CurrentState = RANDOM_SEARCH;

      }

      // write commands to robot
      pp.SetSpeed(newspeed, newturnrate);

      }
    }
    catch (PlayerCc::PlayerError e)
    {
      std::cerr << e << std::endl;
      return -1;
    }
}
```

A.3 Study 3: Block Picker

```
/*
 * blockpicker
 *
 * a simple sequence to pick up a block
 *
 */

#include <libplayerc++/playerc++.h>
#include <iostream>
#include <vector>
#include <math.h>
#include <libthjc/misc.h>


using namespace PlayerCc;
using namespace std;
```

```
int main(int argc, char **argv)
{
  char * RobotHostname = "localhost";
  int RobotPort = 6665;
  char * VisHostname = "localhost";
  int VisPort = 6665;

  StopWatch Timer;
  Timer.GetElapsedDouble();
  double ElapsedTime = 0;

  for (int i = 1; i < argc; ++i)
  {
    if (i < argc-1 && strcmp("--host",argv[i])==0)
      RobotHostname = argv[++i];
    if (i < argc-1 && strcmp("--vishost",argv[i])==0)
      VisHostname = argv[++i];
    if (i < argc-1 && strcmp("--port",argv[i])==0)
      RobotPort = atoi(argv[++i]);
    if (i < argc-1 && strcmp("--visport",argv[i])==0)
      VisPort = atoi(argv[++i]);
  }

  // we throw exceptions on creation if we fail
  try
  {
    double HandleHeight = 0.17; // height above floor that the arm should grab at
    double BlockRadius = 0.001; // radius of circle about top of block thing

    // start the robots in threaded mode
    PlayerClient Robot(RobotHostname, RobotPort);
    Robot.StartThread();
    PlayerClient Vis(VisHostname, VisPort);
    Vis.StartThread();

    LaserProxy Laser(&Robot, 0);
    LimbProxy Limb(&Robot, 0);
    ActArrayProxy aa (&Robot,0);
    GripperProxy Gripper(&Robot,0);

    // initialise the arm
    aa.SetSpeedConfig (0, 10.000);
    aa.SetSpeedConfig (1, 10.000);
    aa.SetSpeedConfig (2, 10.000);
    aa.SetSpeedConfig (3, 10.000);
    aa.SetSpeedConfig (4, 10.000);
    Limb.MoveHome();

    Graphics2dProxy g2d(&Vis, 0);

    std::cout << Robot << " " << Vis << std::endl;

    Limb.RequestGeometry();
    Laser.RequestConfigure();
    Laser.RequestGeom();

    player_pose_t LaserPose = Laser.GetPose();

    // make sure everything is initialised
    cout << "Wait for a bit of data" << endl;
    sleep(1);

    // find the closest point in the laser scan
    int MinIndex = 0;
    for (uint ii=1; ii < Laser.GetCount(); ++ii)
    {
      if (Laser[ii] < Laser[MinIndex])
        MinIndex = ii;
    }

    player_point_2d_t Closest = Laser.GetPoint(MinIndex);
    Closest.px += LaserPose.px;
    Closest.py += LaserPose.py;

    // clear our custom visualisation
    g2d.Clear();

    // draw a cross where we think the block is
```

```
player_point_2d_t Points[5];
Points[0] = Points[1] = Points[2] = Points[3] = Points[4] = Closest;
Points[1].px += 0.03;
Points[3].px -= 0.03;
Points[2].py += 0.03;
Points[4].py -= 0.03;

g2d.Color(255,0,0,0);
g2d.DrawPolyline(Points,5);

// calculate a point in the middle of the block
double PreRange = Laser[MinIndex] +BlockRadius;
double Bearing = Laser.GetMinAngle() + Laser.GetScanRes() * MinIndex; //Laser.GetBearing(MinIndex);
cout << Bearing << " " << Laser.GetBearing(MinIndex) << endl;

player_point_2d_t CentrePoint;
CentrePoint.px = PreRange * cos(Bearing);
CentrePoint.py = PreRange * sin(Bearing);
CentrePoint.px += LaserPose.px;
CentrePoint.py += LaserPose.py;

// display cross in blue where pre point is
Points[0] = Points[1] = Points[2] = Points[3] = Points[4] = CentrePoint;
Points[1].px += 0.03;
Points[3].px -= 0.03;
Points[2].py += 0.03;
Points[4].py -= 0.03;

g2d.Color(0,0,255,0);
g2d.DrawPolyline(Points,5);

// calculate the approach vector
player_point_2d_t Orientation;
Orientation.px = CentrePoint.px - Closest.px;
Orientation.py = CentrePoint.py - Closest.py;

// Calculate a point about 5 cm above the target point.
// move to pre position
cout << "Move to pre position" << endl;
Limb.SetPose(CentrePoint.px, CentrePoint.py, HandleHeight + 0.05, 0,0,-1, Orientation.px,
     Orientation.py, 0);

// wait for it to get there (this could actually poll the arm to check)
sleep (10);

if (Limb.GetData().state == PLAYER_LIMB_STATE_OOR)
{
  cout << "Target out of reach, terminating" << endl;
  Limb.MoveHome();
  return 0;
}

aa.SetSpeedConfig (0, 3.000);
aa.SetSpeedConfig (1, 3.000);
aa.SetSpeedConfig (2, 3.000);
aa.SetSpeedConfig (3, 3.000);
aa.SetSpeedConfig (4, 3.000);

// move to block
cout << "Move to block" << endl;
Limb.SetPose(CentrePoint.px, CentrePoint.py, HandleHeight, 0,0,-1,Orientation.px, Orientation.py,
     0);

// wait for it to get there
sleep (3);

if (Limb.GetData().state == PLAYER_LIMB_STATE_OOR)
{
  cout << "Target out of reach, terminating" << endl;
  Limb.MoveHome();
  return 0;
}

// now close the gripper
Gripper.Close();
sleep(3);

// move arm to position over back
```

```
    Limb.SetPose(0.119, -0.183, 0.234, -0.031, 0.408, -0.912, 0, -0.91, -0.409);
    sleep(5);

    // release block
    Gripper.Open();
    sleep(3);

    // move to home position
    Limb.MoveHome();
    g2d.Clear();

  }
  catch (PlayerCc::PlayerError e)
  {
    std::cerr << e << std::endl;
    return -1;
  }
}
```

B

Example ARDEV Configuration

```
<aride_project version="0.3" >
 <objects>
  <output Width="800" DisplayName=":0" Camera="GUID::3" Height="600" CameraPosition="GUID::7" Type="
      OutputX11" GUID="GUID::1" Enabled="True" Parent="" Name="OutputX110" FullScreen="True" Capture="
      GUID::2" />
  <capture Type="CaptureDC1394" GUID="GUID::2" Enabled="True" Parent="" Name="capture" />
  <camera Origin="0 0 0" Aspect="1.6" Direction="1 0 0" Type="CameraConstant" GUID="GUID::3" Enabled="
      True" Parent="" Up="0 0 1" Name="CameraConstant0" SensorWidth="0.008" CalibrationFile="/home/
      robot/overhead.calib" f_fov="45" />
  <preprocess CameraObject="GUID::3" Type="ARToolKitPlusPreProcess" GUID="GUID::4" Enabled="True"
      Parent="" Name="ARToolKitPlusPreProcess0" />
  <position Height="0.31" Type="ARToolKitPlusPosition" GUID="GUID::6" Enabled="True" Parent="" MarkerID
      ="11" Name="sleepy" PreProcessor="GUID::4" />
  <misc PlayerPort="6665" PlayerServer="sleepy" Type="PlayerClientInterface" GUID="GUID::9" Enabled="
      True" Parent="" Name="sleepy_player" />
  <position Type="CalibratedPosition" GUID="GUID::7" Enabled="True" Parent="" Name="camera_position"
      CalibrationFile="/home/robot/overhead.calib" />
  <position Type="PositionConstant" GUID="GUID::10" Enabled="True" Parent="" Name="Robot_offset"
      Position="0.15 0 -0.31 0 0 0" />
  <position Type="PositionConstant" GUID="GUID::8" Enabled="True" Parent="" Name="laser_position"
      Position="0 0 0.13 0 0 0" />
  <render WithOutline="True" Type="RenderPlayerLaser" GUID="GUID::11" Enabled="True" Parent="" Colour="
      0 0 1 1" Name="sleepy_laser" PlayerClient="GUID::9" RayInterval="60" />
  <render Type="RenderGraphics2DHandler" GUID="GUID::12" Enabled="True" Parent="" Name="g2d_sleepy" />
  <render Type="RenderGraphics3DHandler" GUID="GUID::13" Enabled="True" Parent="" Name="g3d_sleepy" />
  <position Height="0.31" Type="ARToolKitPlusPosition" GUID="GUID::14" Enabled="True" Parent=""
      MarkerID="12" Name="grumpy" PreProcessor="GUID::4" />
  <position Type="PositionConstant" GUID="GUID::15" Enabled="True" Parent="" Name="grumpy_offset"
      Position="0.15 0 -0.31 0 0 0" />
  <position Type="PositionConstant" GUID="GUID::16" Enabled="True" Parent="" Name="grumpy_laser_offset"
       Position="0 0 0.13 0 0 0" />
  <misc PlayerPort="6665" PlayerServer="grumpy" Type="PlayerClientInterface" GUID="GUID::17" Enabled="
      True" Parent="" Name="grumpy_player" />
  <render WithOutline="True" Type="RenderPlayerLaser" GUID="GUID::18" Enabled="True" Parent="" Colour="
      0 1 0 1" Name="grumpy_laser" PlayerClient="GUID::17" RayInterval="60" />
  <render Type="RenderGraphics3DHandler" GUID="GUID::26" Enabled="True" Parent="" Name="grumpy_g3d" />
  <position Type="PositionConstant" GUID="GUID::19" Enabled="True" Parent="" Name="grumpy_g2d_offset"
      Position="0 0 0.1 0 0 0" />
  <render Type="RenderGraphics2DHandler" GUID="GUID::20" Enabled="True" Parent="" Name="grumpy_g2d" />
  <position Height="0.31" Type="ARToolKitPlusPosition" GUID="GUID::22" Enabled="True" Parent=""
      MarkerID="16" Name="sneezy" PreProcessor="GUID::4" />
```

```
 <position Type="PositionConstant" GUID="GUID::23" Enabled="True" Parent="" Name="sneezy_offset"
      Position="0.15 0 -0.31 0 0 0" />
 <position Type="PositionConstant" GUID="GUID::24" Enabled="True" Parent="" Name="laser_position"
      Position="0 0 0.13 0 0 0" />
 <misc PlayerPort="6665" PlayerServer="sneezy" Type="PlayerClientInterface" GUID="GUID::25" Enabled="
      True" Parent="" Name="sneezy_player" />
 <render WithOutline="False" Type="RenderPlayerLaser" GUID="GUID::27" Enabled="True" Parent="" Colour=
      "1 0 1 1" Name="sneezy_laser" PlayerClient="GUID::25" RayInterval="20" />
 <render Type="RenderGraphics2DHandler" GUID="GUID::28" Enabled="True" Parent="" Name="sneezy_g2d" />
 <render Type="RenderGraphics3DHandler" GUID="GUID::29" Enabled="True" Parent="" Name="sneezy_g3d" />
 <render Radius="0.05" Type="RenderPlayerLimb" GUID="GUID::31" Enabled="False" Parent="" Colour="1 0 0
      1" Name="grumpy_limb" PlayerClient="GUID::17" Index="0" />
 <render Type="RenderPlayerActArray" GUID="GUID::32" Enabled="True" Parent="" Colour="1 1 0 1" Name="
      grumpy_act_array" PlayerClient="GUID::17" Index="0" />
 <position Height="0.31" Type="ARToolKitPlusPosition" GUID="GUID::33" Enabled="True" Parent=""
      MarkerID="13" Name="dopey" PreProcessor="GUID::4" />
 <position Type="PositionConstant" GUID="GUID::34" Enabled="True" Parent="" Name="Robot_offset"
      Position="0.15 0 -0.31 0 0 0" />
 <position Type="PositionConstant" GUID="GUID::35" Enabled="True" Parent="" Name="laser_position"
      Position="0 0 0.13 0 0 0" />
 <misc PlayerPort="6665" PlayerServer="dopey" Type="PlayerClientInterface" GUID="GUID::36" Enabled="
      True" Parent="" Name="dopey_player" />
 <render WithOutline="True" Type="RenderPlayerLaser" GUID="GUID::37" Enabled="True" Parent="" Colour="
      0.58 0 0.24 1" Name="dopey_laser" PlayerClient="GUID::36" RayInterval="20" />
 <render Type="RenderGraphics2DHandler" GUID="GUID::38" Enabled="True" Parent="" Name="dopey_g2d" />
 <render Type="RenderGraphics3DHandler" GUID="GUID::39" Enabled="True" Parent="" Name="dopey_g3d" />
 <position Height="0.31" Type="ARToolKitPlusPosition" GUID="GUID::41" Enabled="True" Parent=""
      MarkerID="14" Name="bashful" PreProcessor="GUID::4" />
 <position Type="PositionConstant" GUID="GUID::42" Enabled="True" Parent="" Name="robot_offset"
      Position="0.15 0 -0.31 0 0 0" />
 <misc PlayerPort="6665" PlayerServer="bashful" Type="PlayerClientInterface" GUID="GUID::43" Enabled="
      True" Parent="" Name="bashful_player" />
 <position Type="PositionConstant" GUID="GUID::44" Enabled="True" Parent="" Name="LaserPosition"
      Position="0 0 0.13 0 0 0" />
 <render WithOutline="False" Type="RenderPlayerLaser" GUID="GUID::45" Enabled="True" Parent="" Colour=
      "1 0.5 0.2 1" Name="bashful_laser" PlayerClient="GUID::43" RayInterval="20" />
</objects>
<environment Camera="GUID::3" GUID="GUID::0" Enabled="True" Name="Environment0" Capture="GUID::2"
     Output="GUID::1" >
 <item GUID="GUID::4" />
</environment>
<display_list GUID="GUID::47" Name="Global" >
 <item GUID="GUID::7" />
</display_list>
<display_list GUID="GUID::5" Name="Sleepy" >
 <item GUID="GUID::6" >
  <item GUID="GUID::10" >
   <item GUID="GUID::8" >
    <item GUID="GUID::11" />
   </item>
   <item GUID="GUID::12" />
   <item GUID="GUID::13" />
  </item>
 </item>
 <item GUID="GUID::9" />
</display_list>
<display_list GUID="GUID::21" Name="Sneezy" >
 <item GUID="GUID::22" >
  <item GUID="GUID::23" >
   <item GUID="GUID::24" >
    <item GUID="GUID::27" />
   </item>
   <item GUID="GUID::28" />
   <item GUID="GUID::29" />
  </item>
 </item>
 <item GUID="GUID::25" />
</display_list>
<display_list GUID="GUID::30" Name="Dopey" >
 <item GUID="GUID::33" >
  <item GUID="GUID::34" >
   <item GUID="GUID::35" >
    <item GUID="GUID::37" />
   </item>
   <item GUID="GUID::38" />
   <item GUID="GUID::39" />
  </item>
 </item>
```

```
  <item GUID="GUID::36" />
 </display_list>
 <display_list GUID="GUID::40" Name="Bashful" >
  <item GUID="GUID::41" >
   <item GUID="GUID::42" >
    <item GUID="GUID::44" >
     <item GUID="GUID::45" />
    </item>
   </item>
  </item>
  <item GUID="GUID::43" />
 </display_list>
 <display_list GUID="GUID::46" Name="Grumpy" >
  <item GUID="GUID::14" >
   <item GUID="GUID::15" >
    <item GUID="GUID::16" >
     <item GUID="GUID::18" />
    </item>
    <item GUID="GUID::26" />
    <item GUID="GUID::19" >
     <item GUID="GUID::20" />
    </item>
    <item GUID="GUID::31" />
    <item GUID="GUID::32" />
   </item>
  </item>
  <item GUID="GUID::17" />
 </display_list>
</aride_project>
```

Bibliography

[1] ACM. *Proceedings of the 2006 ACM Conference on Human-Robot Interaction*. ACM Press, March 2006.

[2] P. Amstutz and A.H. 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.

[3] Joe Armstrong. The development of erlang. In *ICFP '97: Proceedings of the second ACM SIGPLAN international conference on Functional programming*, pages 196–203, New York, NY, USA, 1997. ACM Press.

[4] ARTag. `http://www.artag.net/`, July 2007.

[5] ARToolkit Home Page. `http://www.hitl.washington.edu/artoolkit/`, August 2006.

[6] Ascension Technologies. `http://www.ascension-tech.com/`, July 2006.

[7] Hoyuko Automatic. `http://www.hokuyo-aut.jp/`, August 2005.

[8] R Azuma, Y Baillot, R Behringer, S Feiner, S Julier, and B MacIntyre. Recent advances in augmented reality. *Computer Graphics and Applications, IEEE*, 21(6):34–47, 2001.

[9] Ronald T. Azuma. A survey of augmented reality. *Presence: Teleoperators & Virtual Environments*, 6(4):355–385, August 1997.

[10] A. Balijepalli and T. Kesavadas. An exploratory haptic based robotic path planning and training tool. In *Proc. IEEE International Conference on Robotics and Automation (ICRA 02)*, volume 1, pages 438–443, 2002.

[11] Geoffrey Biggs. The mobile robot programming problem. Technical Report No. 630, Department of Department of Electrical and Computer Engineering, The University of Auckland, May 2006.

[12] Geoffrey Biggs and Bruce MacDonald. A survey of robot programming systems. In *Proceedings of the Australasian Conference on Robotics and Automation*, CSIRO, Brisbane, Australia, December 1–3 2003.

[13] Geoffrey Biggs and Bruce MacDonald. A design for dimensional analysis in robotics. In *Third International Conference on Computational Intelligence, Robotics and Autonomous Systems*, Singapore, December 2005.

[14] G.M. Biggs and B.A. MacDonald. Specifying robot reactivity in procedural languages. In *Proc. IEEE/RSJ Int. Conference on Intelligent Robots and Systems*, Beijing, China, October 2006.

[15] A. Billard, Y. Epars, G. Cheng, and S. Schaal. Discovering imitation strategies through categorization of multi-dimensional data. In *Proc. IEEE/RSJ International Conference on Intelligent Robots and System (IROS 03)*, volume 3, pages 2398–2403, 2003.

[16] Aude Billard. Robota: Clever toy and educational tool. *Robotics and Autonomous Systems*, 42(3-4):259–269, 2003.

[17] Oliver Bimber and Ramesh Raskar. *Spatial augmented reality : merging real and virtual worlds*. A K Peters, Wellesley, Mass., 2005.

[18] Rainer Bischoff and Arif 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.

[19] G. Booch, J. Rumbaugh, and I. Jacobson. *The unified modeling language user guide*. Addison-Wesley, Reading Ma., 1999. Grady Booch, James Rumbaugh, Ivar Jacobson. Series Title: The Addison-Wesley object technology series City: Reading Ma. : Includes index.

[20] D. A. Bowman, E. Kruijff, J. J. LaViola Jr., and I. Poupyrev. *3D User Interfaces: Theory and Practive*. Addison-Wesley, Boston, 2004.

[21] D. A. Bowman, E. Kruijff, J. J. LaViola Jr., and I. Poupyrev. *Evaluation of 3D User Interfaces*, chapter 11, pages 349–384. Addison-Wesley, Boston, 2004.

[22] C. Breazeal. Emotive qualities in robot speech. In *Proc. IEEE/RSJ International Conference on Intelligent Robots and System (IROS 01)*, volume 3, pages 1388–1394, 2001.

[23] 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.

[24] C. Breazeal and B. Scassellati. How to build robots that make friends and influence people. In *Proc. IEEE/RSJ International Conference on Intelligent Robots and System (IROS 99)*, volume 2, pages 858–863, 1999.

[25] Cynthia Breazeal. Toward sociable robots. *Robotics and Autonomous Systems*, 42(3-4):167–175, 2003.

[26] Cynthia Breazeal. Function meets style: insights from emotion theory applied to hri. *Systems, Man and Cybernetics, Part C, IEEE Transactions on*, 34(2):187–194, May 2004.

[27] Cynthia Breazeal. Social interactions in hri: the robot view. *Systems, Man and Cybernetics, Part C, IEEE Transactions on*, 34(2):181–186, May 2004.

[28] Alex Brooks, Tobias Kaupp, Alex Makarenko, Anders Orebk, and Stefan Williams. Towards component-based robotics. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2005.

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

[30] M. Cabido-Lopes and J. Santos-Victor. Visual transformations in gesture imitation: what you see is what you do. In *Proc. IEEE International Conference on Robotics and Automation (ICRA 03)*, volume 2, pages 2375–2381, 2003.

[31] L. Canamero and J. Fredslund. I show you how i like you - can you read it in my face? *IEEE Trans. Syst., Man, Cybern. A*, 31(5):454–459, 2001.

[32] CARMEN. `http://carmen.sourceforge.net/`, July 2006.

[33] Chaomei Chen, editor. *Information Visualization*. Springer, 2nd edition, 2004.

[34] Gordon Cheng, Akihiko Nagakubo, and Yasuo Kuniyoshi. Continuous humanoid interaction:: An integrated perspective – gaining adaptivity, redundancy, flexibility – in one. *Robotics and Autonomous Systems*, 37(2-3):161–183, 2001.

[35] Adrian David Cheok, Kok Hwee Goh, Wei Liu, Farzam Farbiz, Siew Wan Fong, Sze Lee Teo, Yu Li, and Xubo Yang. Human pacman: a mobile, wide-area entertainment system based on physical, social, and ubiquitous computing. *Personal Ubiquitous Comput.*, 8(2):71–81, 2004.

[36] 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].

[37] Wusheng Chou, Tianmiao Wang, Da Liu, and Zengmin Tian. Computer and robot assisted tele-neurosurgery. In *Proc. IEEE/RSJ International Conference on Intelligent Robots and System (IROS 03)*, volume 4, pages 3367–3372, 2003.

[38] CLARAty homepage. `http://claraty.jpl.nasa.gov/main/`, July 2006.

[39] J.E. Colgate, M. Peshkin, and S.H. Klostermeyer. Intelligent assist devices in industrial applications: a review. In *Proc. IEEE/RSJ International Conference on Intelligent Robots and System (IROS 03)*, volume 3, pages 2516–2521, 2003.

[40] Toby H. J. Collett and Bruce MacDonald. Developer oriented visualisation of a robot program. In *Proc. Conference on Human-Robot Interaction*, pages 49–55, Salt Lake City, Utah, March 2–4 2006.

[41] Toby H J Collett, Bruce MacDonald, and Brian Gerkey. Player 2.0: Toward a practical robot programming framework. In *Proc. Australasian Conf. on Robotics and Automation*, December 2005.

[42] Toby H. J. Collett and Bruce A. MacDonald. Augmented reality visualisation for player. In *Proc. IEEE Int. Conf. on Robotics and Automation (ICRA'06)*, pages 3954–9, Orlando, May 2006.

[43] C. Colombo, A. Del Bimbo, and A. Valli. Visual capture and understanding of hand pointing actions in a 3-d environment. *IEEE Trans. Syst., Man, Cybern. B*, 33(4):677–686, August 2003.

[44] M. Daily, Youngkwan 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.

[45] K. Dautenhahn, M. Walters, S. Woods, K. L. Koay, C. L. Nehaniv, E. A. Sisbot, R. Alami, and T.Siméon. How may i serve you? a robot companion approaching a seated person in a helping context. In *Proceedings of the 2006 ACM Conference on Human-Robot Interaction*, pages 172–179. ACM, ACM Press, March 2006.

[46] DDD – Data Display Debugger. `http://www.gnu.org/software/ddd`, August 2005.

[47] Eclipse. `http://www.eclipse.org`, August 2005.

[48] Emily Falcone, Rachel Gockley, Eric Porter, and Illah Nourbakhsh. The personal rover project:: The comprehensive design of a domestic personal robot. *Robotics and Autonomous Systems*, 42(3-4):245–258, 2003.

[49] Josh Faust, Cheryl Simon, and William D. Smart. A video game-based mobile robot simulation environment. In *Proceedings of the 2006 IEEE/RSJ International Conference on Intellegent Robots and Systems (IROS06)*, pages 3749–3754, Beijing, China, October 2006. IEEE/RSJ.

[50] FFmpeg. `http://ffmpeg.mplayerhq.hu/`, August 2006.

[51] Terrence Fong, Clayton Kunz, Laura M. Hiatt, and Magda Bugajska. The human-robot interaction operating system. In *Proceedings of the 2006 ACM Conference on Human-Robot Interaction*, pages 41–48. ACM, ACM Press, March 2006.

[52] 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.

[53] M Frigola, J. Fernandez, and J. Aranda. Visual human machine interface by gestures. In *Proc. IEEE International Conference on Robotics and Automation (ICRA 03)*, volume 1, pages 386–391, September 2003.

[54] T. Fukuda, J. Taguri, F. Arai, M. Nakashima, D. Tachibana, and Y. Hasegawa. Facial expression of robot face for human-robot mutual communication. In *Proc. IEEE International Conference on Robotics and Automation (ICRA 02)*, volume 1, pages 46–51, 2002.

[55] S.S. Ghidary, Y. Nakata, H. Saito, M. Hattori, and T. Takamori. Multi-modal human robot interaction for map generation. In *Proc. IEEE/RSJ International Conference on Intelligent Robots and System (IROS 01)*, volume 4, pages 2246–2251, 2001.

[56] A. Gimenez, C. Balaguer, A.M. Sabatini, and V. Genovese. The mats robotic system to assist disabled people in their home environments. In *Proc. IEEE/RSJ International Conference on Intelligent Robots and System (IROS 03)*, volume 3, pages 2612–2617, 2003.

[57] Rachel Gockley, Jodi Forlizzi, and Reid Simmons. Interactions with a moody robot. In *Proceedings of the 2006 ACM Conference on Human-Robot Interaction*, pages 186–193. ACM, ACM Press, March 2006.

[58] W.B. Griffin, W.R. Provancher, and M.R. Cutkosky. Feedback strategies for shared control in dexterous telemanipulation. In *Proc. IEEE/RSJ International Conference on Intelligent Robots and System (IROS 03)*, volume 3, pages 2791–2796, 2003.

[59] Georges Grinstein and Haim Levkowitz, editors. *Perceptual Issues in Visualization*. Springer, 1995.

[60] Luke Gumbley and Bruce A MacDonald. Development of an integrated robotic programming environment. In *Australasian Conference on Robotics and Automation*, Sydney, 5–7 December 2005.

[61] Shuxiang Guo, K. Sugimoto, and S. Hata. A human scale tele-operating system for microoperation. In *Proc. IEEE International Symposium on Computational Intelligence in Robotics and Automation*, pages 230–235, 2001.

[62] Kuk-Hyun Han, Yong-Jae Kim, Jong-Hwan Kim, and S. Hsia. Internet control of personal robot between kaist and uc davis. In *Proc. IEEE International Conference on Robotics and Automation (ICRA 02)*, volume 2, pages 2184–2189, 2002.

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

[64] Charles D. Hansen and Christopher R. Johnson, editors. *The Visualization Handbook*. Elsevier, 2005.

[65] T. Hashimoto. Emotion model in robot assisted activity. In *Proc. IEEE International Symposium on Computational Intelligence in Robotics and Automation*, pages 184–188, 2001.

[66] B Hibbard. Top ten visualization problems. In *Proc. ACM Siggraph*, volume 33,2, pages 21–22. ACM Press, 1999.

[67] W.A. Hoff and J.C. Lisle. Mobile robot control using a small display. In *Proc. IEEE/RSJ International Conference on Intelligent Robots and System (IROS 03)*, volume 4, pages 3473–3478, 2003.

[68] Kai-yuh Hsiao, N. Mavridis, and D. Roy. Coupling perception and simulation: steps towards conversational robotics. In *Proc. IEEE/RSJ International Conference on Intelligent Robots and System (IROS 03)*, volume 1, pages 928–933, 2003.

[69] Intersense. `http://www.intersense.com/`, July 2006.

[70] I. Iossifidis, C. Theis, C. Grote, C. Faubel, and G. Schoner. Anthropomorphism as a pervasive design concept for a robotic assistant. In *Proc. IEEE/RSJ International*

Conference on Intelligent Robots and System (IROS 03), volume 4, pages 3465–3472, 2003.

[71] iRobot. irobot corporation: Home page. `http://www.iRobot.com/`, June 2006.

[72] I. Jacobson. *Object-oriented software engineering : a use case driven approach.* Addison-Wesley Pub., [New York] ACM Press Wokingham, Eng. ; Reading, Mass., 1992. Ivar Jacobson ... [et al.]. City: [New York] : Wokingham, Eng. ; Reading, Mass. : Includes bibliographical references (p. 513-520) and index.

[73] B. Jensen, R. Philippsen, and R. Siegwart. Narrative assesment for human-robot interaction. In *Proc. IEEE International Conference on Robotics and Automation (ICRA 03)*, volume 1, pages 1503–1508, September 2003.

[74] Chris Johnson. Top scientific visualization research problems. *IEEE Computer Graphics and Applications*, 24(4):13–17, Jul/Aug 2004.

[75] K. Kaneko, F. Kanehiro, S. Kajita, H. Hirukawa, T. Kawasaki, M. Hirata, K. Akachi, and T. Isozumi. Humanoid robot hrp-2. In *Robotics and Automation, 2004. Proceedings. ICRA '04. 2004 IEEE International Conference on*, volume 2, pages 1083–1090, 2004.

[76] Daniel A. Keim, R. Daniel Bergeron, and Ronal M. Pickett. Test data sets for evaluating data visualization techniques. In Georges Grinstein and Haim Levkowitz, editors, *Perceptual Issues in Vizualiszation*, IFIP Series on Computer Graphics, chapter 2, pages 9–21. Springer, 1995.

[77] H. Kobayashi, Y. Ichikawa, M. Senda, and T. Shiba. Realization of realistic and rich facial expressions by face robot. In *Proc. IEEE/RSJ International Conference on Intelligent Robots and System (IROS 03)*, volume 2, pages 1123–1128, 2003.

[78] H. Kobayashi, Y. Ichikawa, T. Tsuji, and K. Kikuchi. Development on face robot for real facial expressions. In *Proc. IEEE/RSJ International Conference on Intelligent Robots and System (IROS 01)*, volume 4, pages 2215–2220, 2001.

[79] M. Konyo, S. Tadokoro, M. Hira, and T. Takamori. Quantitative evaluation of artificial tactile feel display integrated with visual information. In *Proc. IEEE/RSJ International Conference on Intelligent Robots and System (IROS 02)*, volume 3, pages 3060–3065, 2002.

[80] Tijn Koojimans, Takayuki Kanda, Christoph Bartneck, Hiroshi Ishiguro, and Norihiro Hagita. Interaction debugging: an integral approach to analyze human-robot

interaction. In *Proceedings of the 2006 ACM Conference on Human-Robot Interaction*, pages 64–71. ACM, ACM Press, March 2006.

[81] K. Kosuge, T. Hayashi, Y. Hirata, and R. Tobiyama. Dance partner robot -ms dancer. In *Proc. IEEE/RSJ International Conference on Intelligent Robots and System (IROS 03)*, volume 4, pages 3459–3464, 2003.

[82] KUKA Industrial Robots. `http://www.kuka.com/en/`, July 2006.

[83] Vladimir A. Kulyukin and Chaitanya Gharpure. Ergonomics-for-one in a robotic shopping cart for the blind. In *HRI '06: Proceeding of the 1st ACM SIGCHI/SIGART conference on Human-robot interaction*, pages 142–149, New York, NY, USA, 2006. ACM Press.

[84] Yuan-hsin (Oscar) Kuo and Bruce MacDonald. A distributed real-time software framework for robotic applications. In *Proc. IEEE Int. Conf. on Robotics and Automation (ICRA'05)*, pages 1976–81, Barcelona, 18–22 April 2005.

[85] T. Kyriacou, G. Bugmann, and S. Lauria. Vision-based urban navigation procedures for verbally instructed robots. In *Proc. IEEE/RSJ International Conference on Intelligent Robots and System (IROS 02)*, volume 2, pages 1326–1331, 2002.

[86] C. Laschi, M. Gonzalo-Tasis, J.F. Codes, and P. Dario. Recognizing hand posture by vision: applications in humanoid personal robotics. In *Proc. IEEE International Conference on Robotics and Automation (ICRA 02)*, volume 2, pages 1439–1444, 2002.

[87] C.K.H. Law, M.Y.Y. Leung, Y. Xu, and S.K. Tso. A cap as interface for wheelchair control. In *Proc. IEEE/RSJ International Conference on Intelligent Robots and System (IROS 02)*, volume 2, pages 1439–1444, 2002.

[88] A. Leleve, P. Fraisse, and P. Dauchez. Telerobotics over ip networks: Towards a low-level real-time architecture. In *Proc. IEEE/RSJ International Conference on Intelligent Robots and System (IROS 01)*, volume 2, pages 643–648, 2001.

[89] M.C. Lin, W. Baxter, M. Foskey, M.A. Otaduy, and V. Scheib. Haptic interaction for creative processes with simulated media. In *Proc. IEEE International Conference on Robotics and Automation (ICRA 02)*, volume 1, pages 598–604, 2002.

[90] Wang-Tai Lo, Yantao Shen, and Yun-Hui Liu. An integrated tactile feedback system for multifingered robot hands. In *Proc. IEEE/RSJ International Conference on Intelligent Robots and System (IROS 01)*, volume 2, pages 680–685, 2001.

[91] S. Maeyama, S. Yuta, and A. Harada. Remote viewing on the web using multiple mobile robotic avatars. In *Proc. IEEE/RSJ International Conference on Intelligent Robots and System (IROS 01)*, volume 2, pages 637–642, 2001.

[92] R. Marin, P. J. Sanz, and A. P. del Pobil. A predictive interface based on virtual and augmented reality for task specification in a web telerobotic system. In *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 02)*, volume 3, pages 3005–3010, 2002.

[93] R. Marin, P.J. Sanz, and J.S. Sanchez. A very high level interface to teleoperate a robot via web including augmented reality. In *Proc. IEEE International Conference on Robotics and Automation (ICRA 02)*, volume 3, pages 2725–2730, 2002.

[94] R. Marin, P. Vila, P.J. Sanz, and A. Marzal. Automatic speech recognition to teleoperate a robot via web. In *Proc. IEEE/RSJ International Conference on Intelligent Robots and System (IROS 02)*, volume 2, pages 1278–1283, 2002.

[95] B.H McCormick, T.A. DeFanti, and M.D. Brown, editors. *Vizualization in Scientific Computing*, chapter A, pages 15–21. ACM Press, 1987.

[96] P. McGuire, J. Fritsch, J.J. Steil, F. Rothling, G.A. Fink, S. Wachsmuth, G. Sagerer, and H. Ritter. Multi-modal human-machine communication for instructing robot grasping tasks. In *Proc. IEEE/RSJ International Conference on Intelligent Robots and System (IROS 02)*, volume 2, pages 1082–1088, 2002.

[97] MIARN: Modules for Intelligent Autonomous Robot Navigation. `http://miarn.sourceforge.net/`, July 2006.

[98] 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.

[99] T. Mirfakhrai and S. Payandeh. A model for time-delays for teleoperation over the internet. In *Proc. IEEE International Symposium on Computational Intelligence in Robotics and Automation*, pages 236–241, 2001.

[100] Miro. `http://www.ics.uci.edu/~seguti/miro/Miro\_index.html`, July 2006.

[101] H. Miwa, T. Okuchi, H. Takanobu, and A. Takanishi. Development of a new human-like head robot we-4. In *Proc. IEEE/RSJ International Conference on Intelligent Robots and System (IROS 02)*, volume 3, pages 2443–2448, 2002.

[102] H. Miwa, A. Takanishi, and H. Takanobu. Experimental study on robot personality for humanoid head robot. In *Proc. IEEE/RSJ International Conference on Intelligent Robots and System (IROS 01)*, volume 2, pages 1183–1188, 2001.

[103] H. Miwa, T. Umetsu, A. Takanishi, and H. Takanohu. Human-like robot head that has olfactory sensation and facial color expression. In *Proc. IEEE International Conference on Robotics and Automation (ICRA 01)*, volume 1, pages 459–464, 2001.

[104] MobileRobots. Mobilerobots activmedia robotics. `http://www.mobilerobots.com/`, June 2006.

[105] K. Morioka, Joo-Ho Lee, and H. Hashimoto. Physical agent for human following in intelligent sensor network. In *Proc. IEEE/RSJ International Conference on Intelligent Robots and System (IROS 02)*, volume 2, pages 1234–1239, 2002.

[106] Lilia Moshkina, Yoichiro Endo, and Ronald C. Arkin. Usability evaluation of an automated mission repair mechanism for mobile robot mission specification. In *Proceedings of the 2006 ACM Conference on Human-Robot Interaction*, pages 57–63. ACM, ACM Press, March 2006.

[107] K. Nakadai, H. G. Okuno, and H. Kitano. Robot recognizes three simultaneous speech by active audition. In *Proc. IEEE International Conference on Robotics and Automation (ICRA 03)*, volume 1, pages 398–405, September 2003.

[108] Y. Nakauchi, P. Naphattalung, T. Takahashi, T. Matsubara, and E. Kashiwagi. Proposal and evaluation of natural language human-robot interface system based on conversation theory. In *Proc. IEEE International Conference on Robotics and Automation (ICRA 03)*, volume 1, pages 380–385, September 2003.

[109] Chrisopher P. Nemeth. *Human Factors Methods for Design: Making Systems Human-Centered*. CRC Press, 2004.

[110] I.A.D. Nesnas, R. Volpe, T. Estlin, H. Das, R. Petras, and D. Mutz. Toward developing reusable software components for robotic applications. In *Intelligent Robots and Systems, 2001. Proceedings. 2001 IEEE/RSJ International Conference on*, volume 4, pages 2375–2383, Nov 2001.

[111] Issa A. D. Nesnas, Anne Wright, Max Bajracharya, Reid Simmons, and Tara Estlin. Claraty and challenges of developing interoperable robotic software. In *Proceedings of the 2003 IEEE/RSJ Intl. Conference on Intelligent Robots and Systems (IROS03)*, volume 3, pages 2428–2435, Las Vegas, Nevada, October 2003.

[112] U Neumann and S You. Natural feature tracking for augmented reality. *Multimedia, IEEE Transactions on*, 1(1):53–64, 1999.

[113] Lung Ngai, W.S. Newman, and V. Liberatore. An experiment in internet-based, human-assisted robotics. In *Proc. IEEE International Conference on Robotics and Automation (ICRA 02)*, volume 2, pages 2190–2195, 2002.

[114] M.N. Nicolescu and M.J. Mataric. Learning and interacting in human-robot domains. *IEEE Trans. Syst., Man, Cybern. A*, 31(5):419–430, 2001.

[115] Jakob Nielsen. *Usability Engineering*. Academic Press, 1993.

[116] Jakob Nielsen. *Heuristic Evaluation*, chapter 2, pages 25–62. John Wiley & Sons, New York, 1994.

[117] K. Nishikawa, H. Takanobu, T. Mochida, M. Honda, and A. Takanishi. Development of a new human-like talking robot having advanced vocal tract mechanisms. In *Proc. IEEE/RSJ International Conference on Intelligent Robots and System (IROS 03)*, volume 2, pages 1907–1913, 2003.

[118] R. Nisimura, T. Uchida, A. Lee, H. Saruwatari, K. Shikano, and Y. Matsumoto. Aska: receptionist robot with speech dialogue system. In *Proc. IEEE/RSJ International Conference on Intelligent Robots and System (IROS 02)*, volume 2, pages 1314–1319, 2002.

[119] M. Nohmi. Space teleoperation using force reflection of communication time delay. In *Proc. IEEE/RSJ International Conference on Intelligent Robots and System (IROS 03)*, volume 3, pages 2809–2814, 2003.

[120] H. G. Okuno, K. Nakadai, and H. Kitano. Realizing personality in audio-visually trggered non-verbal behaviours. In *Proc. IEEE International Conference on Robotics and Automation (ICRA 03)*, volume 1, pages 392–397, September 2003.

[121] H.G. Okuno, K. Nakadai, K.I. Hidai, H. Mizoguchi, and H. Kitano. Human-robot interaction through real-time auditory and visual multiple-talker tracking. In *Proc. IEEE/RSJ International Conference on Intelligent Robots and System (IROS 01)*, volume 3, pages 1402–1409, 2001.

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

[123] V.I.S. Pavlovic R. and T.S. Huang. Visual interpretation of hand gestures for human-computer interaction: a review. *IEEE Trans. Pattern Anal. Machine Intell.*, 19(7TY - JOUR):677–695, 1997.

[124] 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.

[125] Joelle Pineau, Michael Montemerlo, Martha Pollack, Nicholas Roy, and Sebastian Thrun. Towards robotic assistants in nursing homes: Challenges and results. *Robotics and Autonomous Systems*, 42(3-4):271–281, 2003.

[126] Player Adaptive Monte-Carlo Driver. `http://playerstage.sourceforge.net/doc/Player-2.0.0/player/group__driver__amcl.html`, July 2007.

[127] Player/Stage. The player/stage project. `http://playerstage.sourceforge.net/`, January 2005.

[128] William Playfair. *The commercial and political atlas.* printed for J. Debrett; G. G. and J. Robinson; J. Sewell; the engraver, S. J. Neele; W. Creech and C. Elliot, Edinburgh; and L. White, Dublin, 1786.

[129] N.S. Pollard, J.K. Hodgins, M.J. Riley, and C.G. Atkeson. Adapting human motion for the control of a humanoid robot. In *Proc. IEEE International Conference on Robotics and Automation (ICRA 02)*, volume 2, pages 1390–1397, 2002.

[130] C. Pradakier, J. Hermosillo, C. Koike, C. Braillon, P. Bessiere, and C. Laugier. Safe and autonomous navigation for a car-like robot amoung pedestrian. In M. Armada and P. Gonzales de Santos, editors, *Proc. IARP 3rd International Workshop on Service, Assistive and Personal Robots*, pages 49–56. IARP, 2003.

[131] E. Prassler, D. Bank, and B. Kluge. Motion coordination between a human and a mobile robot. In *Proc. IEEE/RSJ International Conference on Intelligent Robots and System (IROS 02)*, volume 2, pages 1228–1233, 2002.

[132] Lutz Prechelt. An empirical comparison of C, C++, Java, Perl, Python, Rexx, and Tcl for a search/string-processing program. Technical report, Fakultät für Informatik, Universität Karlsruhe, Germany, D-76128 Karlsruhe, Germany, March 2000.

[133] P.J. Prodanov, A. Drygajlo, G. Ramel, M. Meisser, and R. Siegwart. Voice enabled interface for interactive tour-guide robots. In *Proc. IEEE/RSJ International Conference on Intelligent Robots and System (IROS 02)*, volume 2, pages 1332–1337, 2002.

[134] Erin Pudenz, Geb Thomas, Peter Coppin, and Nathalie Cabrol. Searching for a quantitative proxy for rover science effectiveness. In *Proceedings of the 2006 ACM Conference on Human-Robot Interaction*, pages 18–25. ACM, ACM Press, March 2006.

[135] A.E. Quaid and R.A. Abovitz. Haptic information displays for computer-assisted surgery. In *Proc. IEEE International Conference on Robotics and Automation (ICRA 02)*, volume 2, pages 2092–2097, 2002.

[136] 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.

[137] P. Rani, N. Sarkar, and C.A. Smith. Affect-sensitive human-robot cooperation-theory and experiments. In *Proc. IEEE International Conference on Robotics and Automation (ICRA 03)*, volume 2, pages 2382–2387, 2003.

[138] R.S. Rao, K. Conn, S.H. Jung, J. Katupitiya, T. Kientz, V. Kumar, J. Ostrowski, S. Patel, and C.J. Taylor. Human robot interaction: application to smart wheelchairs. In *Proc. IEEE International Conference on Robotics and Automation (ICRA 02)*, volume 4, pages 3583– 3588, 2002.

[139] M. Rauterberg, M. Bichsel, M. Meier, and M. Fjeld. A gesture based interaction technique for a planning tool for construction and design. In *Robot and Human Communication, 1997. RO-MAN '97. Proceedings., 6th IEEE International Workshop on*, pages 212–217, 1997.

[140] Penny Rheingans and Chris Landreth. Perceptual principles for effective visualizations. In Georges Grinstein and Haim Levkowitz, editors, *Perceptual Issues in Vizualiszation*, IFIP Series on Computer Graphics, chapter 6, pages 59–73. Springer, 1995.

[141] M. Riley, A. Ude, K. Wade, and C.G. Atkeson. Enabling real-time full-body imitation: a natural way of transferring human movement to humanoids. In *Proc. IEEE International Conference on Robotics and Automation (ICRA 03)*, volume 2, pages 2368–2374, 2003.

[142] Orca Robotics. `http://orca-robotics.sourceforge.net/`, July 2006.

[143] A. Sano, H. Fujimoto, H. Kajino, G. Isobe, and H. Takeuchi. Intuitive teleoperation by micro dome system. In *Proc. IEEE/RSJ International Conference on Intelligent Robots and System (IROS 01)*, volume 4, pages 1836–1841, 2001.

[144] Jean Scholtz. Theory and evaluation of human-robot interaction. In *Proceedings of Hawaii International Conference on System Science (HICSS) 36*, 6–9 January 2003.

[145] Jean Scholtz, Mary Theofanos, and Brian Antonishek. Development of a test bed for evaluating human-robot performanc for explosive ordnance disposal robots. In *Proceedings of the 2006 ACM Conference on Human-Robot Interaction*, pages 10–17. ACM, ACM Press, March 2006.

[146] Jean Scholtz, Jeff Young, Jill L. Drury, and Holly A. Yanco. Evaluation of human-robot interaction awareness in search and rescue. In *Proceedings of the 2004 IEEE International Conference on Robotics and Automation*, pages 2327–2332, April 2004.

[147] Segway RMP. `http://www.segway.com/products/rmp/`, July 2006.

[148] A.S. Sekmen, M. Wilkes, and K. Kawamura. An application of passive human-robot interaction: human tracking based on attention distraction. *IEEE Trans. Syst., Man, Cybern. A*, 32:248– 259, 2002.

[149] Brennan P. Sellner, Laura M. Hiatt, Reid Simmons, and Sanjiv Singh. Attaining situational awareness for sliding autonomy. In *Proceedings of the 2006 ACM Conference on Human-Robot Interaction*, pages 80–87. ACM, ACM Press, March 2006.

[150] Brennan Peter Sellner, Laura M. Hiatt, Reid Simmons, and Sanjiv Singh. Attaining situational awareness for sliding autonomy. In *Proceedings of HRI 2006*, March 2006.

[151] Kerstin Severinson-Eklundh, Anders Green, and Helge Huttenrauch. Social and collaborative aspects of interaction with a service robot. *Robotics and Autonomous Systems*, 42(3-4):223–234, 2003.

[152] Yafang Shi, S. Kotani, and H. Mori. A pre-journey system of the robotic travel aid. In *Robotics and Automation, 2002. Proceedings. ICRA '02. IEEE International Conference on*, volume 4, pages 4149–4154, 2002.

[153] K. Shibuya, T. Ogawa, and S. Komatsu. Influence of pleasant and unpleasant feelings on human gesture motion. In *Proc. IEEE/RSJ International Conference on Intelligent Robots and System (IROS 02)*, volume 2, pages 1240–1245, 2002.

[154] 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.

[155] R. Siegwart, K. O. Arras, S. Bouabdallah, D. Burnier, G. Froidevaux, X. Greppin, B. Jensen, A. Lorotte, L. Mayor, and M. Meisser. Robox at expo.02: A large-scale installation of personal robots. *Robotics and Autonomous Systems*, 42(3-4):203–222, 2003.

[156] P. Skrzypczynski. Guiding a mobile robot with an internet application. In *Proc. IEEE/RSJ International Conference on Intelligent Robots and System (IROS 01)*, volume 2, pages 649–654, 2001.

[157] Peter Squire, Greg Trafton, and Raja Parasuraman. Human control of multiple unmanned vehicles: Effects of interface type on execution and task switching times. In *Proceedings of the 2006 ACM Conference on Human-Robot Interaction*, pages 26–32. ACM, ACM Press, March 2006.

[158] Aaron Steinfeld, Terrence Fong, David Kaber, Michael Lewis, Jean Scholtz, Alan Schultz, and Michael Goodrich. Common metrics for human-robot interaction. In *Proceedings of the 2006 ACM Conference on Human-Robot Interaction*, pages 33–40. ACM, ACM Press, March 2006.

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

[160] P.T. Szemes, Joo-Ho Lee, and P. Korondi. Guiding assistant for disabled in intelligent urban environment. In *Proc. IEEE/RSJ International Conference on Intelligent Robots and System (IROS 03)*, volume 3, pages 2853–2858, 2003.

[161] Fumihide Tanaka, Javier R. Movellan, Bret Fortenberry, and Kazuki Aisaka. Daily hri evaluation at a classroom environment – reports from dance interaction experiments–. In *Proceedings of the 2006 ACM Conference on Human-Robot Interaction*, pages 3–9. ACM, ACM Press, March 2006.

[162] K. Tatani and Y. Nakamura. Dimensionality reduction and reproduction with hierarchical nlpca neural netowrks - extracting common space of multiple humanoid motion patterns. In *Proc. IEEE International Conference on Robotics and Automation (ICRA 03)*, volume 1, pages 1927–1932, September 2003.

[163] TDL: Task Description Language. `http://www.cs.cmu.edu/~tdl/`, July 2006.

[164] C. Theobalt, J. Bos, T. Chapman, A. Espinosa-Romero, M. Fraser, G. Hayes, E. Klein, T. Oka, and R. Reeve. Talking to godot: dialogue with a mobile robot. In *Proc. IEEE/RSJ International Conference on Intelligent Robots and System (IROS 02)*, volume 2, pages 1338–1343, 2002.

[165] B.C. Thomas B., J. Donoghue, J. Squires, P. De Bondi, M. Morris, and W. Piekarski. Arquake: an outdoor/indoor augmented reality first person application vo -. *Wearable Computers, 2000. The Fourth International Symposium on*, pages 139–146, 2000.

[166] T. Tomizawa, A. Ohya, and S. Yuta. Book browsing system using an autonomous mobile robot teleoperated via the internet. In *Proc. IEEE/RSJ International Conference on Intelligent Robots and System (IROS 02)*, volume 2, pages 1284–1289, 2002.

[167] Félix-Étienne Trépanier and Bruce 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.

[168] Roger Y Tsai. A versatile camera calibration technique for high-accuracy 3d machine vision metrology using off-the-shelf tv cameras and lenses. *IEEE Journal of Robotics and Automation*, RA-3(4):323–344, August 1987.

[169] T. Tsuji, Y. Tanaka, and M. Kaneko. Bio-mimetic trajectory generation based on human arm movements with a nonholonomic constraint. *IEEE Trans. Syst., Man, Cybern. A*, 32(6):773– 779, 2002.

[170] Edward R. Tufte. *The visual display of quantitative information.* Graphics Press, 1983.

[171] Edward R. Tufte. *Envisioning Information.* Graphics Press, 1990.

[172] Edward R. Tufte. *Visual explanations : images and quantities, evidence and narrative.* Graphics Press, 1997.

[173] Vicon. `http://www.vicon.com/`, August 2006.

[174] Daniel Wagner, Thomas Pintaric, Florian Ledermann, and Dieter Schmalstieg. Towards massively multi-user augmented reality on handheld devices. *Lecture Notes in Computer Science*, 3468:208–219, 2005.

[175] Howard Wainer. *Visual revelations : graphical tales of fate and deception from Napoleon Bonaparte to Ross Perot.* Copernicus, 1997.

[176] Jijun Wang, Michael Lewis, Stephen Hughes, and Mary Koes. Validating usarsim for use in hri research. In *Human Factors and Ergonomics Society 49th Annual Meeting, Proceedings of the*, pages 457–461, 2005.

[177] T. Wosch and W. Feiten. Reactive motion control for human-robot tactile interaction. In *Proc. IEEE International Conference on Robotics and Automation (ICRA 02)*, volume 4, pages 3870–3812, 2002.

[178] K. Yokoyama, H. Handa, T. Isozumi, Y. Fukase, K. Kaneko, F. Kanehiro, Y. Kawai, F. Tomita, and H. Hirukawa. Cooperative works by a human and a humanoid robot. In *Proc. IEEE/RSJ International Conference on Intelligent Robots and System (IROS 03)*, volume 3, pages 2985–2991, 2003.

[179] Dong Hyun Yoo, Jae Heon Kim, Do Hyung Kim, and Myung Jin Chung. A human-robot interface using vision-based eye gaze estimation system. In *Proc. IEEE/RSJ International Conference on Intelligent Robots and System (IROS 02)*, volume 2, pages 1196–1201, 2002.

[180] M. Yoshizaki, A. Nakamura, and Y. Kuno. Mutual assistance between speech and vision for human-robot interface. In *Proc. IEEE/RSJ International Conference on Intelligent Robots and System (IROS 02)*, volume 2, pages 1308–1313, 2002.

[181] M. Yoshizaki, A. Nakamura, and Y. Kuno. Vision-speech system adapting to the user and environment for service robots. In *Proc. IEEE/RSJ International Conference on Intelligent Robots and System (IROS 03)*, volume 2, pages 1290–1295, 2003.

[182] James E. Young, Ehud Sharlin, and Jeffrey E. Boyd. Implementing bubblegrams: The use of haar-like features for human-robot interaction. In *Proceedings of the 2006 IEEE Conference on Automation Science and Engineering*, pages 308–313. IEEE, October 2006.

[183] YouTube. `http://www.youtube.com/`, November 2006.

[184] Jrgen Ziegler. Interactive techniques. *ACM Computing Surveys (CSUR)*, 28(1):185–187, 1996.