# Aneka: A control software framework for autonomous robotic systems

Arun Raj Vasudev

December 2005

# Acknowledgements

My deepest gratitude is due to Dr. Prahlad Vadakkepat, ECE department, NUS, for his kind and patient guidance throughout the course of my Masters study and preparation of this thesis. Thank you, Sir.

# Contents

**Abstract**

Control software in intelligent autonomous systems has a role that is vastly different from that of merely implementing numerical algorithms. The classical reference signal is replaced by a higher-level goal (or goals) that the controller is to accomplish, and several layers - varying from lower level to higher level automation - have to be used to implement the controller. Compounding the difficulties is the fact that software traditionally is seen as an information processing tool, and concepts such as stability and system response are not deemed relevant in a software context. There is a dearth of system-theoretic tools and concepts for effectively using intelligent software in feedback loops of physical systems.

This thesis discusses Aneka (meaning "several" in Sanskrit), an architectural framework for control software used in autonomous robotic systems. The thesis proposes modeling the most common software components in autonomous robots based on classical concepts such as systems and signals, making such concepts relevant in a software context. A reference implementation on a multi-robot soccer system is provided as an example of how the ideas could work in practice, though the approach taken itself can be translated to several robotic domains.

The framework along with its reference implementation demonstrates how perception, planning and action modules can be combined with several ancillary components such as simulators and interpreted programming environments in autonomous systems. Besides providing default implementations of the various modules, the framework also provides a solid foundation for future research work in machine vision and multi-robot control. The thesis also discusses how issues such as system response and stability can be relevant in autonomous robots.

# Publications

## International Conference Papers

1. Prahlad Vadakkepat, Liu Xin, Xiao Peng, Arun Raj Vasudev, Tong Heng Lee, "Behavior Based and Evolutionary Techniques in Robotics: Some Instances", p136-140; Proceedings of the Third International Symposium on Human and Artificial Intelligence Systems, Fukui, Japan, Dec. 6-7, 2002

2. Tan Shih Jiuh, Prahlad Vadakkepat, Arun Raj Vasudev, "Biomorphic Architecture: Implications and Possibilities in Robotic Engineering", CIRAS Singapore, 2003.

3. Quah Choo Ee, Prahlad Vadakkepat and Arun Raj Vasudev, "Co-evolution of Multiple Mobile Robots", CIRAS Singapore, 2003.

## Book chapters

1. PRAHLAD, V, Arun Raj Vasudev, Xin Liu, Peng Xiao and T H Lee, "Behaviour based and evolutionary techniques in robotics: Some instances." In 'Dynamic Systems Approach for Embodiment and Sociality : From Ecological Psychology to Robotics', edited by Kazuyuki Murasse and Toshiyuki Asakura. International Series on Advanced Intelligence, Volume 6, edited by R.J. Howlett and L.C. Jain, Volume 6 ed., pp.137-150. Adelaide: Advacned Knowledge International Pty Ltd, 2003.

## Symposia

1. Arun Raj Vasudev and Prahlad Vadakkepat, "Fuzzy Logic as a Medium of Activity in Robotic Research", Singapore Robotic Games 2003 Symposium, The National University of Singapore, May 2003.

2. Arun Raj Vasudev and Prahlad Vadakkepat, "Cooperative robotics and robot-soccer systems", Singapore Robotic Games 2004 Symposium, The National University of Singapore, June 2004.

# Chapter 1

# Introduction

Advances in computational and communications technologies have made the implementation of highly autonomous and robust controllers a more achievable goal than ever. The field of intelligent and autonomous robotic systems especially has seen the application of a wide array of new control strategies like soft computing in the past decade. Much of the attention paid to autonomous systems, however, go into specialised areas of the overall autonomous system design problem such as evolutionary tuning of controllers, machine vision or path planning. The issue of how various components of an autonomous system, each of which may be implemented using techniques different from the others, could be brought together into a coherent architecture seldom forms an area of investigation in its own right. This thesis attempts to address such a need by discussing a control software framework for autonomous systems that, while incorporating ideas from classical systems and control theory, simultaneously allows for the application of a large variety of novel techniques and algorithms in a simple, streamlined architecture.

## 1.1 System theoretic perspectives in autonomous robotic systems

Automatic control has played a vital role in the evolution of modern technology. In its broadest scope, control theory deals with the behaviour of dynamical systems over time. A system is most commonly defined as a combination of components that act together and

perform a common objective. Systems need not be limited to physical ones: the concept can be applied to abstract, dynamic phenomena such as those encountered in economics or environmental ecosystems. More pertinently, a system theoretic view is equally valid when applied to highly autonomous entities such as robots that interact with their environment on a real-time basis, adapting their behaviour as required for the achievement of a particular goal or set of goals.

Despite this generality, traditionally Control Theory and Artificial Intelligence have more or less flourished as distinct disciplines, with the former more concerned with physical systems whose dynamics can be modeled and analysed mathematically, and the latter with areas that are more computational and symbolic in nature. Superficially at least, the differences between the fields go even further. Control theory has a tradition of mathematical rigour and concrete specification of controller design procedures. Artificial intelligence, on other hand, has approaches that are more heuristical and ad-hoc. Again, control theory conventionally has dealt with systems in which physics is reasonably well-defined and behaviours modified at a low level in real-time, like attitude control of missiles and control of substance levels in chemical containers. Artificial intelligence problems, such as game playing, proof or refutation of theorems and natural language recognition, are usually posed symbolically, and solutions are not expected to be delivered real-time. Indeed, it could be said that the domain of control theory has been fairly simple and linear systems, while artificial intelligence dealt with problems where highly complex and discontinuous behaviours - sometimes approaching the levels exhibited by humans - are the norm.

An area that has over the past decade or so forced a unification of methodologies and perspectives from both disciplines has been that of autonomous robotic systems. Autonomous robots exist in a physical environment and react to external events physically on a real-time basis. But physical existence is just one characteristic that robots exhibit: they also take in a wide array of input signals from the environment, analyse (or appear

to analyse) them, and produce, through appropriate actuators, behaviours that are complex enough to be termed "intelligent". Hence, the simultaneous relevance of system and machine intelligence concepts in autonomous robotics should not be surprising since ultimately, robots are intelligent systems that exist in a feedback loop with the environment.

Designing autonomous robots in any meaningful degree has become possible only with the recent surge in computational, communications and sensing technologies. Consequently, the problem of devising a systematic controller design methodology for autonomous systems that is similar to the rigorous and well-proven techniques of classical control has received not a little attention from researchers. Passino [13], for instance, discusses a closed loop expert planner architecture (Figure 1.2) that incorporates ideas from expert systems into a feedback loop. Another intuitive architecture presented [13] is that of an intelligent autonomous controller with interfaces for sensing, actuation and human supervision (Figure 1.3).

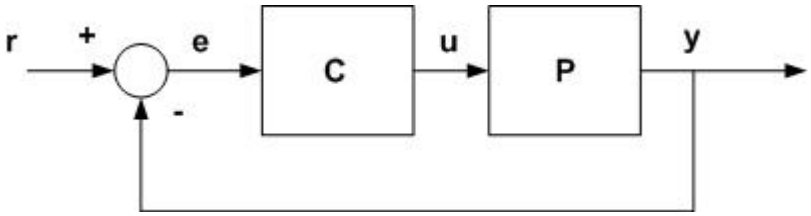


Figure 1.1: A simple continuous feedback control loop.

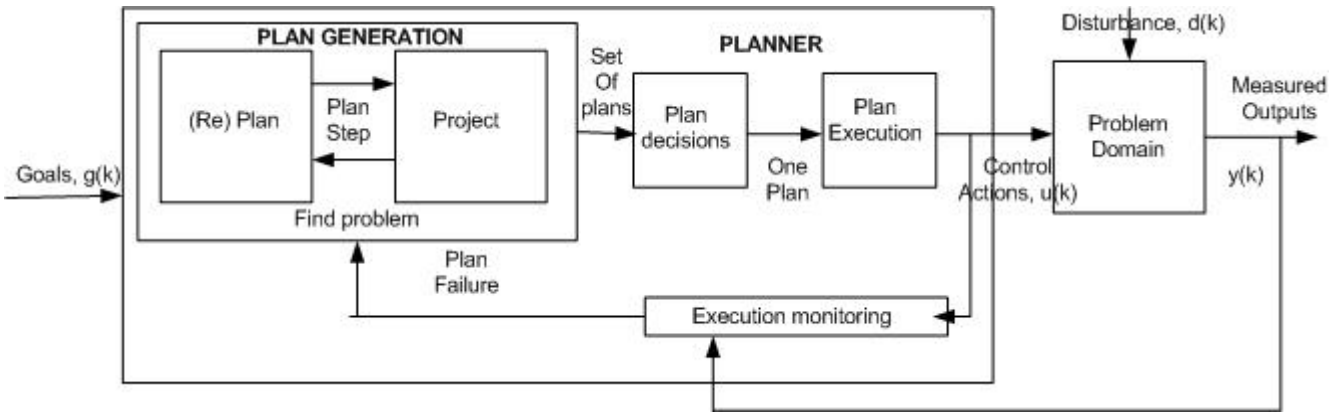


Figure 1.2: An expert planning based controller scheme.

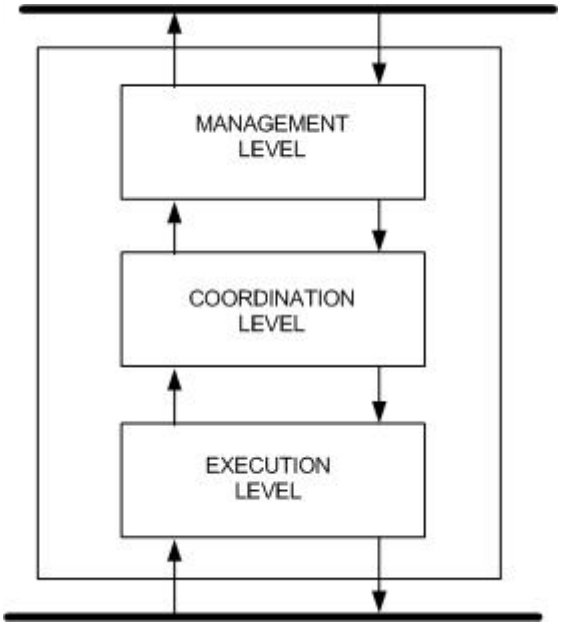


Figure 1.3: A hierarchical scheme for intelligent, autonomous controllers.

Brooks [14] introduces an approach called behaviour based or subsumption architecture that is different from a traditional perceive-think-act scheme. A behaviour based intelligent autonomous controller does not have distinct modules executing perception, planning and action. Instead, the control action is generated by behaviours that can be simultaneously driven by sensory inputs. The overall behaviour of the robot results from how each of the activated behaviours interact with each other, with some behaviours subsuming others lower-level behaviours. Figure 1.4 [14] shows an example where behaviours interact with each other to give a coherent controller.

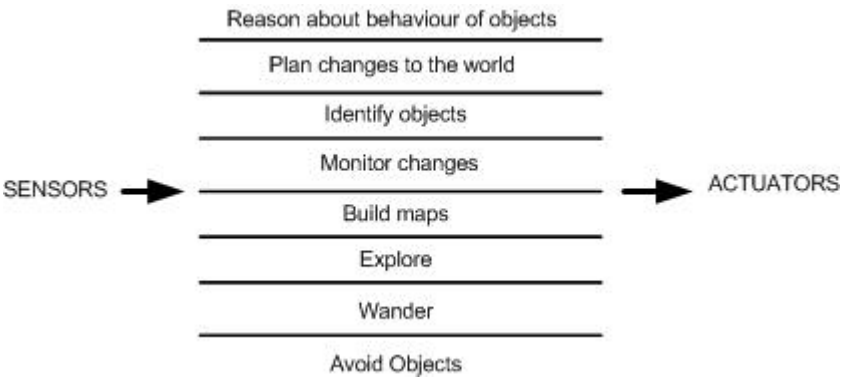


Figure 1.4: Behaviour-based architecture of a mobile robot decomposed into component behaviours.

Subsumption based architectures are often contrasted with the so-termed SMPA (Sense, Model, Plan and Action) based robotic architectures that have been traditionally the norm in artificial intelligence research. An SMPA based architecture essentially consists of discrete and cleanly defined modules, each concerning itself with one of four principal roles: sensing the environment, modeling it into an internal representation, planning a course of action based on the representation, and carrying out the action. As intuitive as this scheme may seem, it has been criticised widely in literature especially over the last decade for imparting "artificiality" to machine intelligence [31], since cleanly defined and fully independent perception and action modules are rarely found in natural living beings.

In answer to the shortcomings of SMPA based approaches to robotics, modern techniques tend to stress more on embodiment of intelligence in autonomous systems [31] [34]. Embodiment requires that the intelligent system be encased in a physical body. There are two motivations for advancing this perspective: first, only an embodied agent is fully validated as one that can deal with the world. Second, having a physical presence is necessary to end "the regression in symbols" the regression ends where the intelligent system meets the world. Brooks [31] introduced the term "physical grounding hypothesis" to highlight the significance of embodiment, as opposed to the classical "physical symbol systems hypothesis" [32] that has been considered the bedrock of Artificial Intelligence practice.

Architectures for distributed robotic systems involve, in addition to implementation of intelligent controllers for each individual robot, concerns related to coordination of robot activities with each other in pursuit of a common goal. Consider Figure 1.5, which depicts a "robot colony" consisting entirely of partially autonomous robots, each having a set of capabilities common to all robots in the system and additional specialised functionalities specifically suited to the role the robot plays in the colony. The aim of the colony is to accomplish a "spread and perform" task, where extensive parallelism and simultaneous coverage over a large area are major requirements for satisfactorily tackling the problem.

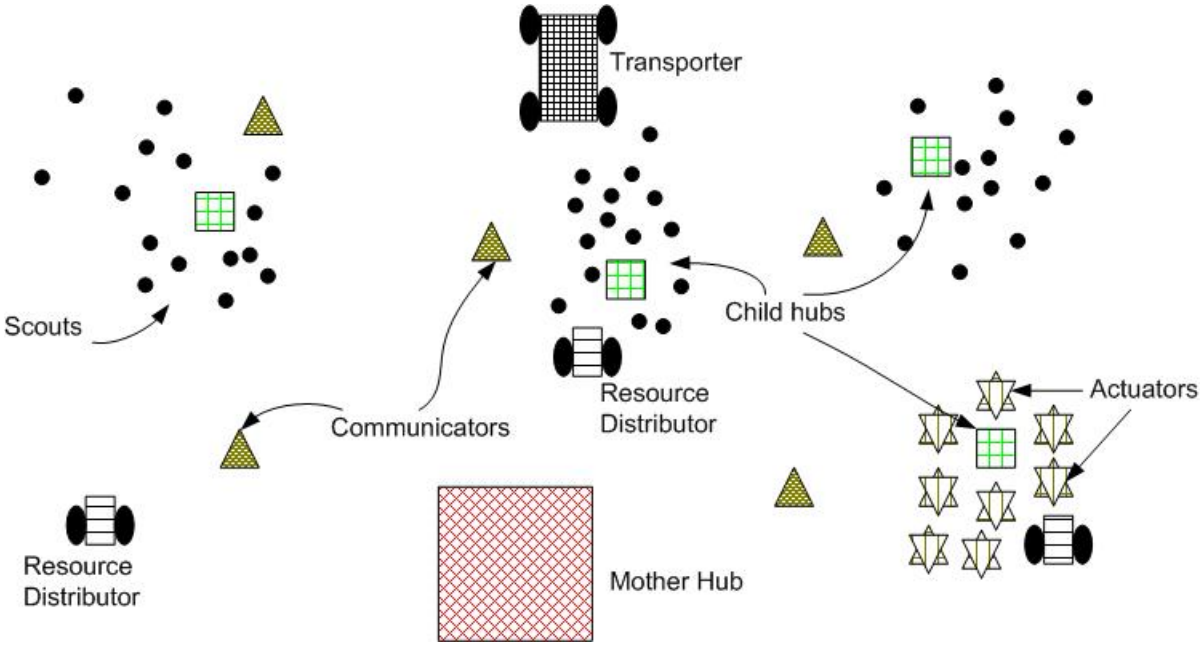


Figure 1.5: An ecosystem consisting of specialised, partially autonomous robots that are overseen and coordinated by a distributed controller, itself implemented through autonomous robots.

Examples of such tasks include search and rescue operations in regions struck by natural disasters, exploration and mining, distributed surveillance and information gathering, putting out forest fires, and neutralisation of hazardous chemical, biological or nuclear leakages. Design of robots based on their roles in the team could be a natural way of partitioning the overall controller.

For example, scout robots in Figure 1.5 could be assigned the role to forage a designated area and search for given features (e.g., chemical signatures, human presence etc.) and report their finding to the child hubs that control them. They might be built with basic capabilities like path planning and obstacle avoidance, along with specialised modules for sensing the environment. The scouts thus are perception units that achieve a form of distributed sensing for the colony as a whole. The mother hub, on the other hand, is a large autonomous vehicle that controls child hubs, which, in turn, locally control robots having still minor roles. Similarly, communicators have specialised modules that allows them to spread out and establish a communication network in an optimal manner, thus facilitating communication between the various robots.

## 1.2 Intelligent control of autonomous robots.

Intelligent control techniques and the recent emphasis on novel techniques in autonomous systems form an interesting combination because intelligent control aims to achieve automation through the emulation of biological and human intelligence [13]. Intelligent control techniques often deal with highly complex systems where control and perception tasks are achieved through techniques from fuzzy logic [8] [9], neural networks [11] [10], symbolic machine learning [6], expert and planning systems [7] and genetic algorithms. Collectively also known as "soft computing", these approaches differ from hard-computing in that they are more tolerant to imprecise information - both in the signals input to the system and in the model assumed of the environment - and lead to controllers that are less "brittle", i.e., that are less prone to breaking down completely if any of the assumptions made in designing the controllers do not hold. They also have the ability to exhibit truly wide operational ranges and discontinuous control actions that would be difficult, if not impossible, to achieve using conventional controllers.

## 1.3 Control software frameworks for autonomous robots

Intelligent control techniques and autonomous robotic systems extensively utilise software for realisation of controllers. Software continues to be the most practical medium for implementing intelligent autonomous controllers primarily because the complexity of behaviour expected from an autonomous system is difficult to engineer using hardwired controllers having narrow operational ranges. Implementing autonomous systems in software at once gives the control engineer or roboticist the ability to express powerful control schemes and algorithms easily. Not only does this facilitate experimentation with control architectures that can lead to more effective control techniques, but it also frees the practitioner to concentrate on the control or perception algorithm itself rather than ancillary and incidental issues related to their implementation.

While high-level schematic descriptions of intelligent control methodologies abound

in literature, the important problem of their implementation in software - and the concomitant design issues it raises - has not received an equal amount of attention from the research community. This is a handicap when implementing controllers for autonomous robots because very often, a significant amount of effort has to be expended on building the supporting software constructs of the autonomous system before any real work on the controller itself can begin. A more important issue, however, is not that of wasted effort, but that in treating software implementation as an adjunct issue, the efficacy of the autonomous system itself could be diminished with unexpected behaviours manifesting when the controller is employed in the real world. A few examples very typical in robot soccer systems are shown in Figure 1.6. As the examples demonstrate, system and control theoretic issues must be built into the software right from the start, rather than added on as an afterthought.

The major thrust of this project is to investigate how such an effective control software framework can be built for autonomous robots. A software framework is essentially a design scheme that can be reused to solve several instances of problems that share common characteristics. The framework designed and investigated as part of this project is named "Aneka", which comes from the word for "several" in Sanskrit. The name was inspired by both the intended universality of the design ("several domains"), as well as by the fact that a system consisting of several robots was used as a reference implementation of the framework. Aneka's major aim is to take a first step in standardising development of custom software-based intelligent controllers by capturing a common denominator of requirements for a control-software framework, and providing an interface through which the framework can be extended into various domains. The reference implementation's generality also allows it to be used as a research platform for specialised areas in robotics, such as perception, state prediction and control.

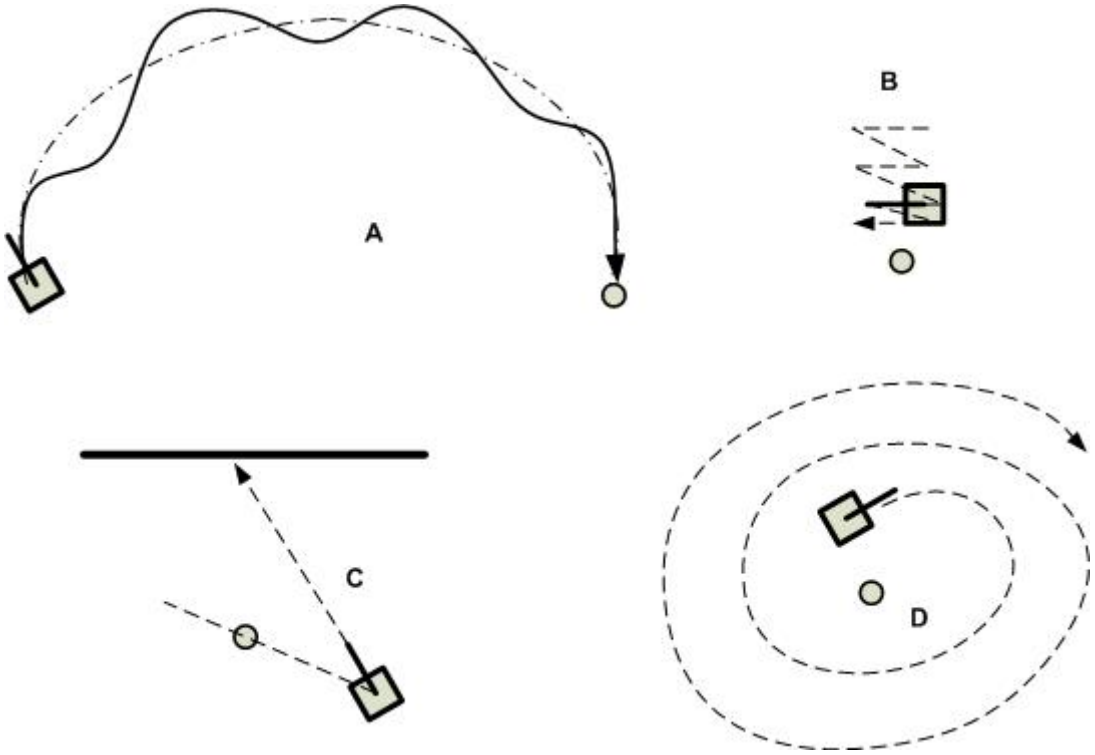


Figure 1.6: Intelligent controllers implemented in software can fail in unexpected ways if system concepts are not taken into account while writing the software. (A) The robot continuously overshoots while reaching the prescribed intermediate points of a given trajectory, resulting in a wobbly path towards the ball. (B) The robot zigzags infinitely even when the ball is but a few centimeters away. (C) Steady state error in achieving the correct orientation causes the intelligent robot to collide energetically against the wall, missing the ball altogether. (D) Control inputs driving a robot to instability. In a robot soccer environment, instabilities result in highly amusing "mad" robots that can disqualify the soccer team, while in real-world applications, results can be downright catastrophic.

## 1.4 Robot soccer systems

Multi-robot soccer systems (Figure 1.7) have emerged in the recent years as an important benchmark platform for issues related to multi-robot control [5]. Specific areas of interest that can be investigated using robot soccer include multi-robot cooperation and decision-making, environment modeling, learning of strategies, machine vision algorithms and robust communication techniques. Robot soccer systems are typically guided by visual inputs, though conceivably other forms of environment sensing, such as SONAR, could be used. The distribution of intelligence in the system can follow broadly three schemes:

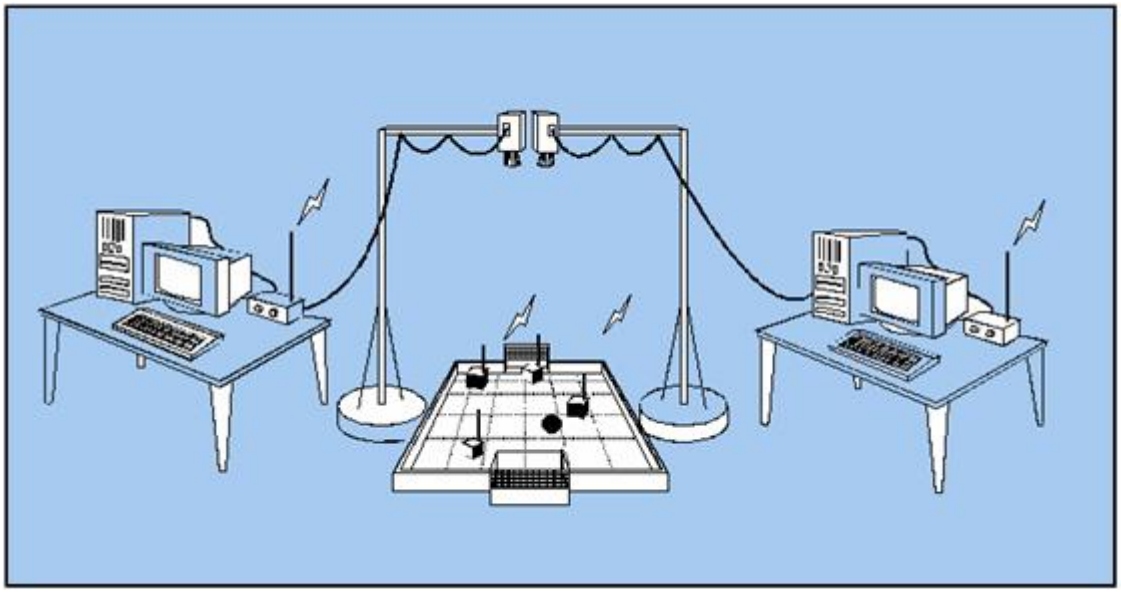

Figure 1.7: A schematic diagram of a typical micro-robot soccer system.

1. Controllers fully centralised on a central computer.

2. Controllers implemented partially on the robots and partially on a central controller.

3. Fully autonomous robots with no central controller at all.

This project, along with proposing a control software framework for autonomous systems, also implements several modules related to a micro-robot soccer system within the framework. Micro-robot soccer system was chosen as a reference implementation platform because, in addition to their availability, they also have a few additional advantages that make them especially suited for testing something as generic as a control software framework:

1. They are an inherently distributed system, making the control problem more open to new algorithms and approaches.

2. All parts of classical robot systems - such as machine perception, robot planning and control, and communication - are adequately represented.

The thesis does not seek to provide an in-depth discussion of object oriented architecture or source-level implementation details. This is due to two reasons. Firstly, such

information is fully contained in the reference implementation's source-code and the accompanying documentation created by a documentation generator (Doxygen) from the source-code and its embedded comments.

Secondly, the project's aim was not to produce a specification of an application programming interface (API) for control software used in autonomous robotic systems. Instead, it was to investigate architectural approaches that would make system-theoretic concepts relevant in software used to control a wide variety of autonomous robotic systems. For example, the Control Executive (Chapter 2, Architecture ) is realised in the reference C++ implementation by the class `CRunner`. In a different domain, the functionality of the Control Executive ( i.e., that of running Linked Systems and transferring Signals generated by them to one another) could be implemented by a differently designed software component or even a combination of hardware and software. `CRunner` is thus mentioned not as a generic interface that can be extended straightforwardly in other domains, but as an example of how the Control Executive could be realised in practice.

The chapters of this thesis are organised as follows:

1. Chapter 1 discusses the relevance of control software frameworks in autonomous robotic systems.
2. Chapter 2 provides an overview of the Aneka framework, a discussion of the layered interface architecture and the rational behind it.
3. Chapter 3 presents the reference implementation of a machine vision algorithm as an example of how machine perception modules could be implemented in Aneka.
4. Chapter 4 discusses implementation of various robotic control related modules, such as a PID controller for path following, a virtual machine based interpreted programming environment for Aneka, and a simulator that can function as a replacement for the external environment.

5. Chapter 5 concludes the thesis, discussing various future directions the current work can take.

# Chapter 2

# Architecture

Aneka provides a framework of real-time supporting processes and interfaces that standardises the architecture of autonomous intelligent controllers for a wide variety of applications. By providing a systematic architectural framework that is independent of the problem domain, the framework helps to avoid redundant mistakes with the implementation of new intelligent controllers. This chapter demonstrates how classical control and systems concepts are extended to a software context in Aneka, setting the background for a detailed description of the Aneka framework in the chapters that follow.

The problem of designing a control software framework is challenging and vaguely defined since intelligent systems can have vastly varying forms, but the problem's complexity can be tamed by concentrating on designing a framework for a very popular subset of intelligent controllers, viz., controllers based on a Sense-Model-Plan-Act (SMPA) scheme (Figure 2.1).

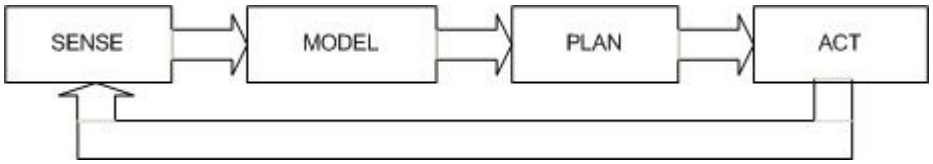


Figure 2.1: Block diagram depicting a Sense-Model-Plan-Act based autonomous control scheme.

SMPA based (or Perceive-Think-Act) controllers have generally been the norm in Ar-

itificial Intelligence and Autonomous Robotics research [30], though it has been criticised by several researchers for its unsuitability in building truly robust and intelligent systems ([30] [33]). Despite the criticisms, SMPA based architectures have the advantages of being widely studied and consequently well-understood, and of being easy to implement for solving pratical problems effectively.

An SMPA based scheme consists of a few logically distinct and well-defined modules that work with each other to achieve the control task. Commonest of these modules fall into the following major categories:

1. **Sensing modules**, such as cameras and other transducers, take input from the environment based on which the intelligent system take actions.
2. **Modelling modules** process information received from the sensing modules into representations of the environment that can be processed by the plan and action modules.
3. **Plan modules** (i.e., controllers) decide on a strategy of action, and specify the steps required to achieve the goals to the action modules
4. **Action modules** contain actuators and/or other output devices that cause a particular action to be taken on the environment.

There exists a well developed mathematical systems and control theory for the study of physical systems represented through mathematical models, especially those that use oridinary linear differential equations. Though controllers in an intelligent robotic system are typically implemented using software that uses discontinuous symbolic rules and instructions to determine how the system should behave, the software itself acts in a real-time environment and must interact with the system's actuators and sensors in a real-time manner within a feedback loop. Hence a useful abstraction would be to model the various modules as systems themselves, and the communication between the models as signals. This is the approach that Aneka takes.

## 2.1 Interface layering in Aneka

The modules in Aneka are modelled on classical concepts systems and signals, and assumes that the controller and supporting systems will be implemented in accordance with a SMPA model. This was primarily done to reduce the problem scope to a manageable level of complexity, and yet keep it wide enough to be of use in designing modern robotic systems.

Autonomous robotic systems - which are presumed to be consistent with an SMPA model - within the Aneka software framework are implemented in three levels (Figure 2.2):

1. Generic System Level Interfaces and the Control Executive
2. Domain Specific Interfaces
3. Concrete implementations of Domain Specific Interfaces

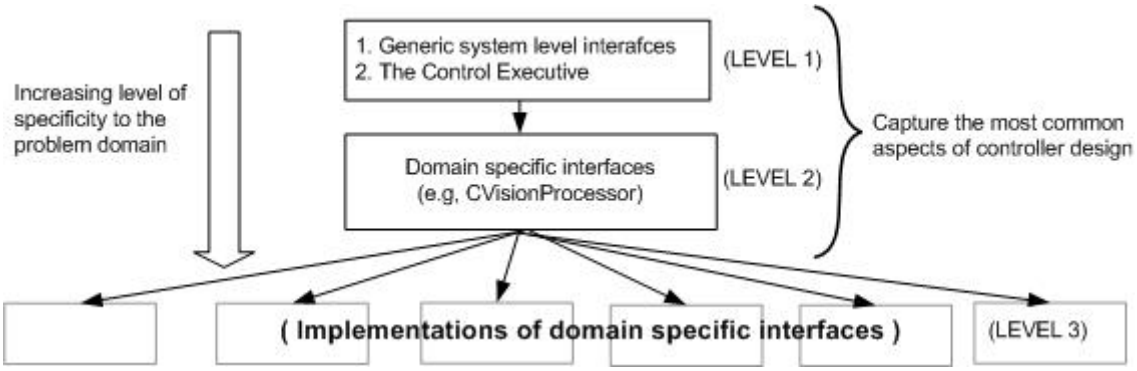


Figure 2.2: Aneka framework specifies module interfaces at three levels. The ultimate system implementation is realised by extending the Domain Specific Interfaces.

The framework itself is implementation agnostic - i.e., very little assumption is made about the programming languages or operating system that will be used. For the purposes of this project, a reference implementation of the framework in C++ spanning some 25,000 lines of code was created to investigate how the ideas work in practice, but the framework would be just as valid with Ada or Java, or even in scenarios where the linked

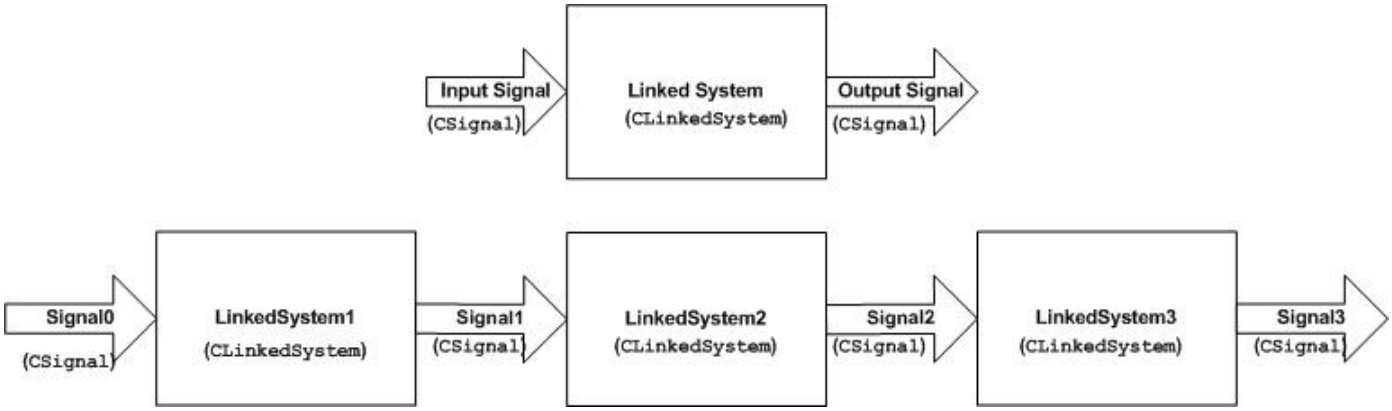


Figure 2.3: All entities in Aneka are linked systems. Linked systems use signals to communicate with other linked systems.

systems could be implemented without using software at all. While it is possible to discuss the architecture of Aneka in an implementation-independent way, it is fruitful to discuss specifics of the reference implementation in relation to their respective higher-level concepts.

## 2.2 Generic System Level Interfaces

Dynamical systems and signals are represented in Aneka through the interfaces **Linked System** and **Signal** (classes `CLinkedSystem` and `CSignal` respectively in the reference implementation). This architecture helps introduce control theoretic concepts into a software context, and conceptually bridges the gap between the physical world and the intelligent system itself. We can readily and intelligibly ask questions about a system's stability, performance and transient response (Chapter 5).

The Linked System is the most fundamental construct in Aneka, and primarily serves to virtually embody abstract entities and algorithms within the controller as distinct objects just like the physical systems the controller deals with. All modules within the Aneka framework - such as controllers, vision modules, frame grabbers and environment state predictors - must implement the abstract methods provided by the generic interface Linked System ( `CLinkedSystem` in the reference implementation). The linked systems

accept inputs and produce outputs in the form of signal objects (Figure 2.3). Linked systems can be connected to ther linked systems, and communicate with them through signal objects which may be different for different linked systems. Thus, a predictor module within an intelligent controller could be a linked system accepting current playground state signals as input signals, and give future playground states as output signals.

Linked systems mandatorily go through a set of states (Figure 2.4) that indicate the nature of the activity they're currently indulging in. The system states in an linked system are defined as:

1. SYS_DEAD
2. SYS_INITIALISED
3. SYS_RUNNING
4. SYS_WAITING
5. SYS_PAUSE_REQUESTED
6. SYS_PAUSED
7. SYS_STOP_REQUESTED

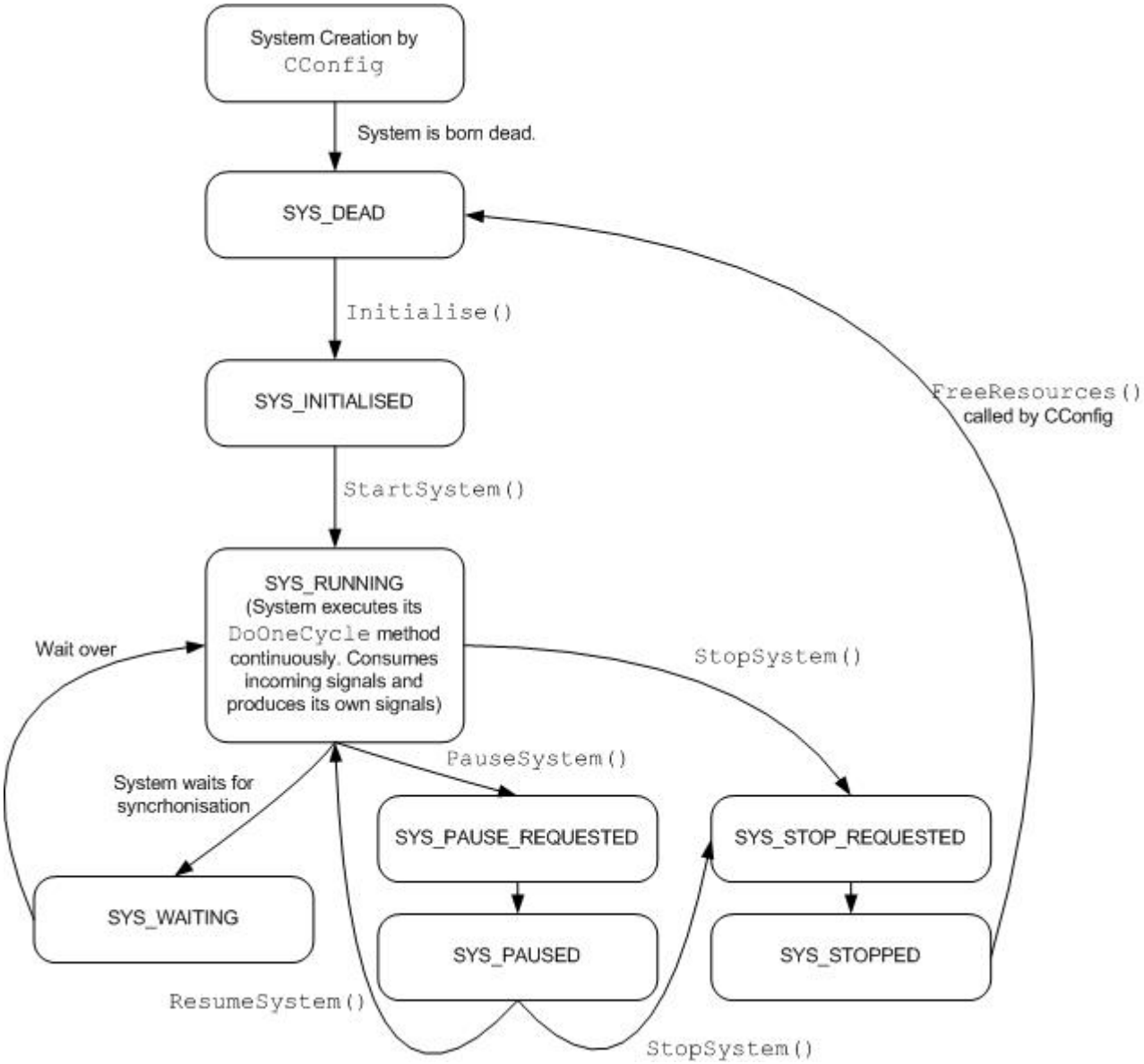


Figure 2.4: State transition diagram depicting the life-cycle of a linked system within the Aneka control software framework.

Implementation of the linked systems (and even specification of abstract functions in the interface) in software is dependent on the specific language being used. For example, the basic class `CLinkedSystem` in the reference implementation contains several methods that can be grouped into the following categories:

1. System State transition functions, including functions for initialisation, starting, pausing, resuming, and freeing resources related to the linked system.

2. System state query functions.

3. Functions for registering and deregistering clients of linked systems (i.e., entities the linked system communicate with ).

4. Functions for getting the current signal and cycle count associated with the system.

In the reference implementation, majority of the system specific processing (such as the control algorithm and the vision algorithm) are implemented in the abstract `DoOneCyle()` method of `CLinkedSystem`. To run a specific system (say, the vision processor, `CVisionProcessor`), its `DoOneCycle()` method is called repeatedly by the Aneka control executive (the `CRunner`).

The transition of linked systems from one state to another is brought about by function calls to the system from external or internal parties. The transitions could be either internally generated or triggered by a user interface event (such as the user requesting that the system be paused).

## 2.3 The Control Executive

Unlike their physical counterparts in the real world, linked systems within a software controller have to be deliberately executed and the signals they produce and consume deliberately transferred from point to point. This function of coordinating and "running" linked systems is carried out in Aneka by a central coordinating authority called the **Control Executive**. The software entity that represents a control executive within the reference implementation is called a "runner", and the class that implements the runner is named `CRunner`. A close analogy of the Aneka control executive would be the kernel of a modern operating system within whose context and supervision various user applications are run.

The framework doesn't insist on how linked systems must be organised physically. This allows linked systems to be run within the same computer process, over multiple processes in the same computing node, or over multiple computing nodes (Figure 2.5) communicating over a network. The interfaces of linked systems and signals easily facilitate several running schemes, and at the same time insulates individual linked systems from the details of communication. The communication details are handled by each computing node's control executive, which ensures that the signals produced by the linked systems within its control are correctly transferred to their consumers in other computing nodes.

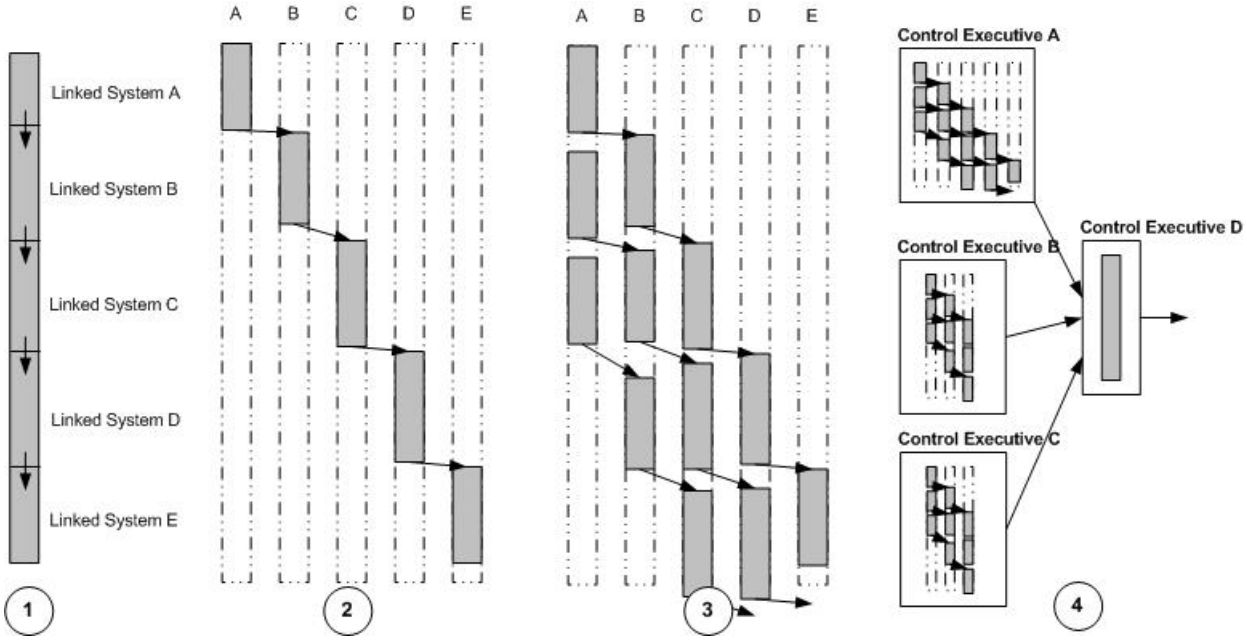


Figure 2.5: (1) Strictly serial running of linked systems within a single process. (2) Strictly serial running over multiple processes. (3) Fully parallel running over multiple processes. (4) Systems running on physically distinct computing nodes, with each node having its own control executive.

Since multi-threading strategies and techniques are dependent on the particular computer platform used, the control executives must be implemented separately for each operating system Aneka platform runs on. A default instance for the Windows operating system, the `CMSRunner` is provided with the reference implementation.

A control software framework must not only specify system-level interfaces, but in practice, must also provide several supporting functions that the linked systems and control executives can use to perform their roles. The Aneka framework provides a number of ancillary classes to help with these functions. These strictly do not form a part of the interface, but they are important implementational considerations that can vary from platform to platform. A few examples would be:

1. `CConfig` for handling system wide configurations.
2. `CGraphicsObject` for primitive graphics operations used by the linked systems to display information about themselves to the user.
3. `CEvent, CSemaphore` etc. for common multi-thread coordination requirements.
4. `CRenderable` for visually "rendering" out an entity for the user to examine its state.

A linked system's execution is started usually in response to a request by the user through the system interface. In the reference implementation, the following series of events take place once the request is received:

1. The linked system is initialised.
2. It's `StartSystem()` method is called.
3. In the `StartSystem()` method, a `CRunner` object for the current platform is acquired from the global configuration object.
4. The linked system runs itself through the `CRunner` class's `Run()` method.

## 2.4 Domain specific interfaces

Linked systems, signals and control executives together form a set of abstractions that can be readily applied to a wide variety of domains. However, these interfaces and functionalities by themselves cannot achieve any useful task, and must be extended to from interfaces that are specific to the domain we are interested in.

| Generic Interfaces | Domain Specific Linked Systems | Domain Specific Signals |
|---|---|---|
| CLinkedSystem | CFrameGrabber | CGrabbedImage |
| Csignal | CVisionProcessor | Cplayground |
| CRunner | CController | CControlAction |
| | CCommunication | |

Figure 2.6: Core classes as implemented in the reference C++ implementation of Aneka.

The intelligent control application investigated in the reference implementation was the multi-robot soccer system. Multi-robot soccer systems provide a challenging problem area where several aspects of intelligent autonomous systems can be investigated. At the domain specific level, the Aneka framework provides abstract classes that encapsulate the functionalities of systems and signals occurring within a typical multi-robot soccer system. These classes also serve as examples on how the core abstract system and signal classes can be extended to autonomous intelligent systems other than robot soccer systems. A sketch of domain specific interfaces for other robotic domains is provided in Chapter 5.

The linked system interfaces specific to multi-robot systems provided in the reference implementation are as follows:

1. Frame Grabber (`CFrameGrabber`)
2. Vision Processor (`CVisionProcessor`)
3. Controller (`CController`)
4. Communication (`CCommunication`)
5. Serial System (`CSerialSystem`)

The signal classes provided in the reference implementation are.

1. The image signal (`CGrabbedImage`)
2. Playground state (`CPlayGround`)
3. Control action (`CControlAction`)

The linked systems and the signals they produce and consume in a multi-robot system designed using the Aneka framework for a classical sense-model-plan-act cycle as shown in Figure 2.7.

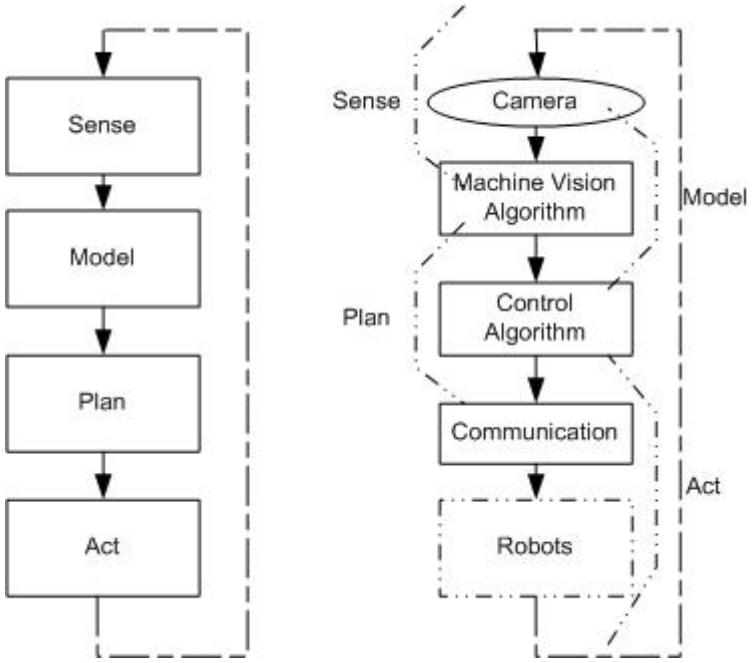


Figure 2.7: Modules of the the robot soccer system broken down by their roles in an SMPA scheme.

The linked systems and signals implemented in software form a closed loop with the external physical environment as shown in Figure 2.8. As the figure demonstrates, using a methodology explicitly based on linked systems and signals helps us visualise the software components as systems that ultimately exist seamlessly with the external systems of the physical world.

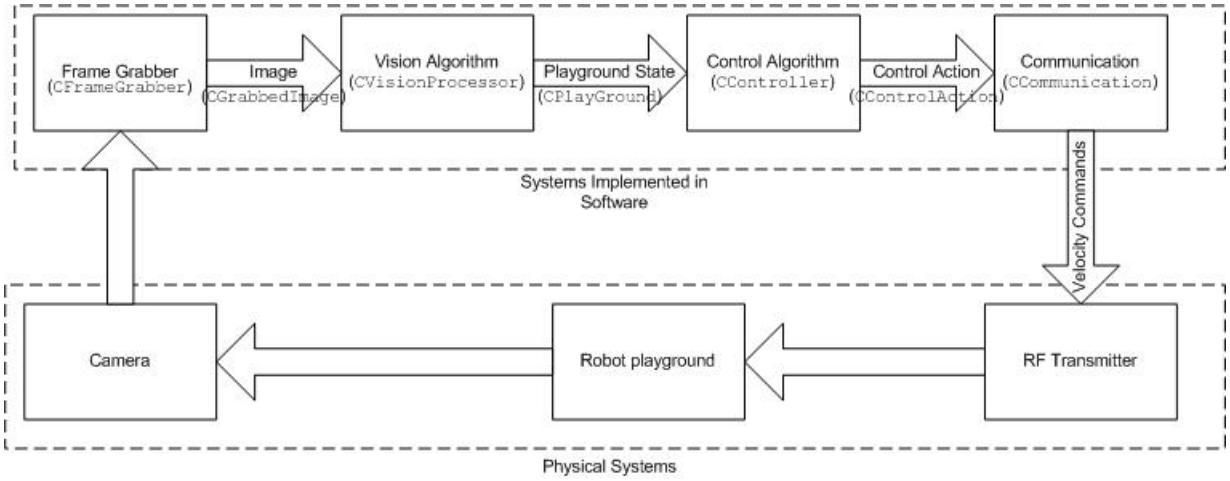


Figure 2.8: A simplified version of the final feedback loop formed in a robot soccer system implemented under the Aneka framework. Each of the linked systems themselves could be composed of linked systems that communicate with each other through signals.

## 2.5 Implementations of domain specific interfaces

The ultimate implementation of the intelligent system is done by concrete instances of the domain specific interfaces. The reference implementation provides concrete instances of frame grabbers, vision algorithm and control and communication algorithms for a robot soccer system, the detailed decription of which are provided in the chapters that follow. All implementations based on the Aneka interfaces form a natural tree-structure as depicted in Figure 2.9.

At the top-most level, the autonomous systems share generic system interfaces and the control executives. Within any domain area, they share domain specific interfaces. An important advantage of such a layered scheme is the coherence it gives to the entire design process because irrespective of their details, ultimately, anything in the controller is a linked system or a signal that is operated by the control executive (Figure 2.8). Furthermore, since implementations within a domain area are required to conform to the domain specific interfaces, it is easy to replace a specific implementation of a module interface (such as the system's vision algorithm) with another. The linked systems inter-

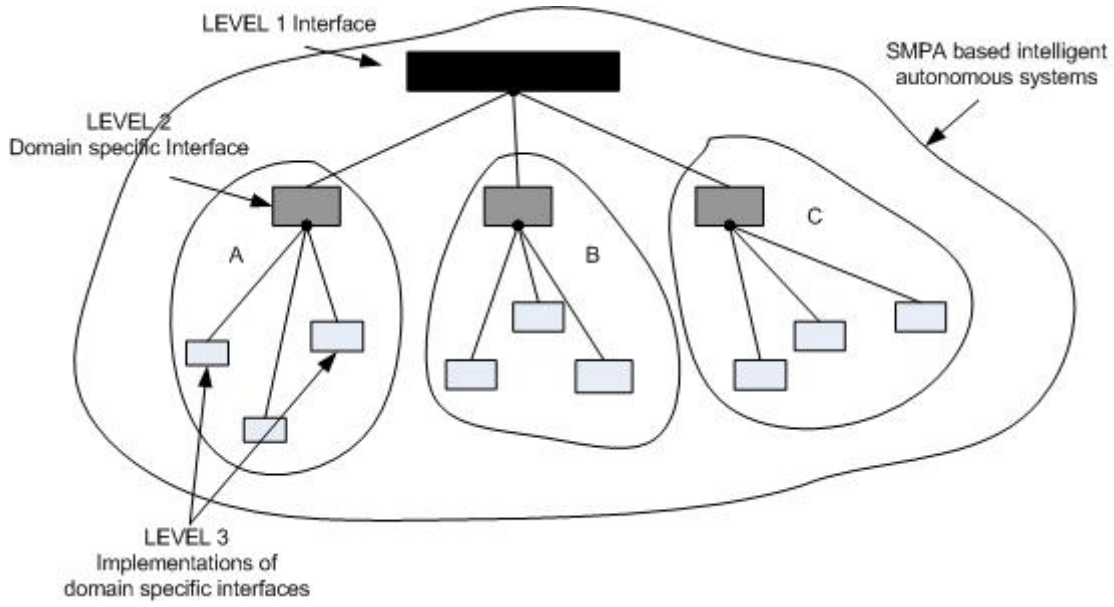


Figure 2.9: All SMPA based controllers share generic system level interfaces and the control executive. Separate domains, such as multi-robot soccer systems or a distributed robotic search and rescue system, share their own domain-specific interfaces.

acting with each other depend only on the interface definitions and not on the particular idiosyncracies of the implementations themselves (Figure 2.10).

The concrete issues that need to be tackled when investigating a topic such as generic control software frameworks are best studied by implementing a qualitatively complete autonomous system under the framework. The robot soccer system used for such a study in this thesis has several qualities that make it a particularly attractive problem area:

1. Robot soccer is a robotic system straightforwardly modelled using a SMPA architecture.

2. Several approaches, ranging from the most primitive to the more esoteric and novel can be used to tackle issues such as machine vision, path planning, prediction and inter-robot cooperation.

3. The control problem can be solved in a parallel and distributed computing environment, providing an opportunity to see how well the control software framework holds in such situations.

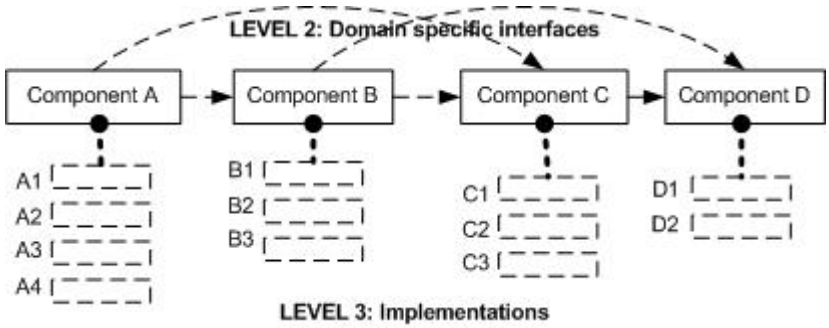


Figure 2.10: Particular implementations of domain specific interfaces are insulated from each other's details by communicating at the domain specific - rather than implementation specific - level. Thus, when component of type A (A1, A2, A3 or A4) sends a signal, it could be transmitted trasparently to any implementation of B, such as B1, B2 or B3. The various implementations can be changed transparently without affecting the overall system design.

The ensuing chapters discuss aspects of implementation of the various major modules in a typical robot system under the Aneka framework. Some of them, such as the implementation of an evolvable multi-agent simulation platform and that if a robust, parallelisable machine vision algorithm are interesting topics in themselves, but a prime focus in their discussion will be to see how well similar independent modules can be integrated into Aneka.

# Chapter 3

# MachineVision

This chapter demonstrates the implementation of sensory and perception modules in intelligent systems under the Aneka framework through a discussion of the machine vision module of the reference robot soccer system.

Machine vision in a robot soccer system when designed using Aneka framework can be decomposed into two distinct modules: the frame grabber, which is responsible for acquiring frames from the real world, and the vision processor, which manipulates the acquired images and forms a model of the environment from it. Frame grabbers and vision processors are straightforwardly specified as Level 2 (domain specific) interfaces that, in turn, extend the linked system interface specified in Level 1 (Figure 2.3). The final implementation occurs at Level 3, when Level 2 interfaces are further refined to embody specific algorithms.

Implementation of the machine vision module is illustrative of the layered architectural scheme outlined in Chapter 2.

1. Linked Systems and Signals form the highest level abstractions of the module. These are represented in the reference implementation by the C++ classes, CLinkedSystem and CSignal.

2. `CFrameGrabber` and `CVisionProcessor` are linked systems that are specific to ma-

chine vision modules in the domain of robot soccer systems. `CFrameGrabber` produces signals of the form `CGrabbedImage`. The `CGrabbedImage` signals are consumed by the linked system `CVisionProcessor`, which in turn produces signals of the form `CPlayground`.

3. Specific realisation of robot soccer controllers must provide concrete instantiations of `CFrameGrabber` and `CVisionProcessor`. For example, the reference implementation provides `CM2FrameGrabber` (representing frame grabbed through Matrox frame grabbers ) and `CFileFrameGrabber` (for grabbing frames from video sequences saved into files) as concrete implementation of the domain level `CFrameGrabber` interface.

Aneka framework thus only guides the decomposition of the machine vision module into three layers in the manner described above. The specific class methods themselves are not stipulated, and is the responsibility of the software designer.

## 3.1 Frame Grabbing

Frame grabbers are digital devices that convert light impinging on the camera into pixel arrays processed by the machine vision module. The frame grabber in a robot-soccer setup is equivalent to any number of transducers conventional robotic systems may have.

Apart from accommodating for the low-level aspects of varying underlying frame-grabber architectures, the frame grabber module must allow for scenarios where frames may not even come from a traditional nearby camera setup, but may be, for instance, transmitted over a long distance from a remote location or generated on the fly by an auxiliary simulator. The interface created by the frame grabber module, ideally, must be transparent to the underlying modules, and its general system characteristics must be reasonably analyzable independent of the rest of the system.

The frame grabbing system takes in images from the camera, and outputs signals as image buffers to the underlying modules. The implementation of the grabber itself is not

consequential as long as the output conforms to a standard form, for example, an array of pixels (Figure 3.1).

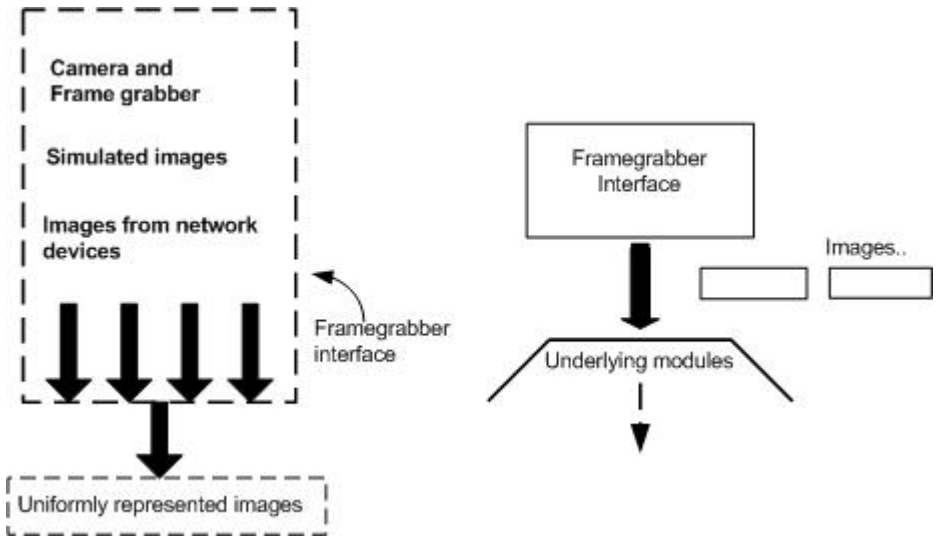


Figure 3.1: Enforcing a standard interface allows flexibility in choosing where the images come from without having to modify underlying modules.

The Aneka framework provides for the modeling of digital transducers such as frame grabbers through the basic system and signal interfaces, as shown in figure 3.2. In the reference implementation, the domain-level interface that represent all frame grabbers possible in the system is `CFrameGrabber`, which accepts input signals from the external world and outputs signals in the form of `CGrabbedImage`. The grabbed images are simply an array of RGB pixels that also contain meta-data such as image dimensions and bits per pixel. The frame grabber is relatively straightforward to model in software, though there could be occasions when images need to be artificially generated, such as during simulations or when reading from images that were pre-recorded. However, as long as the object generating the artificial images conform to the frame grabber interface `CFrameGrabber`, the system architecture itself need not be modified.

## 3.2 Machine Vision

Visual perception forms a very core portion of our cognitive ability. Consequently, attempts to develop artificial perception systems have given due importance to vision, and

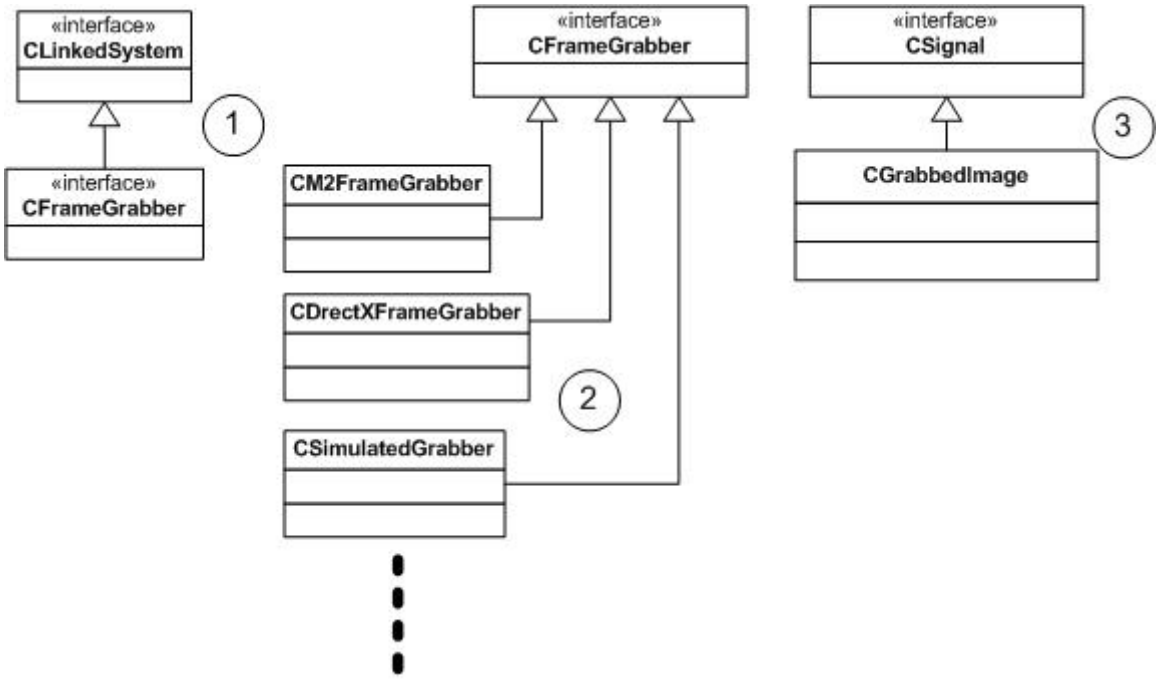


Figure 3.2: Frame grabbers implement the frame grabber interface, which in turn extends the system interface. Images are similarly mapped to the signal interface.

machine vision, especially in the context of the surge in computational power and image acquiring/processing technologies, continues to be a vibrant research area with practical applications in areas such as autonomous vehicle navigation, human face detection and coding, industrial inspection, medical imaging, surveillance and transport.

The machine vision module in a robot soccer system analyses images coming in from the frame grabber, extracts features of interest from it to form a symbolic representation of the environment, and passes the resulting representation on to higher modules for further processing. An intermediate signal processing stage such as a machine vision module has traditionally been a commonality in most robotic systems capable of perception.

Aneka assumes that the primary role of machine vision is one of information processing, i.e., converting incoming signals to symbolic models that other modules can understand. The information-processing approach to machine perception is a classic one that has nevertheless been contested from several quarters. Radically different approaches - such as

visual servoing [24] - based on using the incoming signals to directly drive the system's actuators have recently made their appearances, though scaling them to more complex scenarios have been invariably a challenge. Requiring the intelligent system to interact with the external world without using explicit internal world models implies that the sensory inputs of the system directly drive the motors of the system. This is often called the principle of sensory motor coordination [25], and is considered to be a fundamental characteristic of reactive systems (i.e. systems that "react" to the environmental stimuli [26] [27]). The implication of sensory-motor coordination is that perception and action become highly interdependent. The type of intelligence exhibited by such reactive systems is often referred to as "reactive intelligence", as opposed to the classical "deliberative intelligence" that involve explicit model formation and planning [28] [29].

The Aneka framework itself assumes the system is based on sense-model-plan-act cycles. The machine vision module, its inputs and outputs are readily modelled using Aneka's system and signal interfaces as shown in Figure 3.4, irrespective of the machine vision algorithm actually used to process images coming in from the frame grabber. The interface in the reference implementation that models vision processors is `CVisionProcessor`, which accepts signals of type `CGrabbedImage` and outputs signals of type `CPlayground`. Figure 3.3 depicts how the model of the robot soccer playground can be produced by several Aneka systems implementing the vision processor interface, while at the same time hiding details of implementation effectively from the other modules.

Subsection 3.2.1 describes in detail the implementation of a prototype vision algorithm in the Aneka framework.

### 3.2.1 A windel-based approach to fast segmentation

Several approaches to filtering information from images in robotic applications have been proposed in the literature [23] [22] [21] [19]. The complexity of vision algorithms necessarily have to be limited due to the near real-time nature of the application.

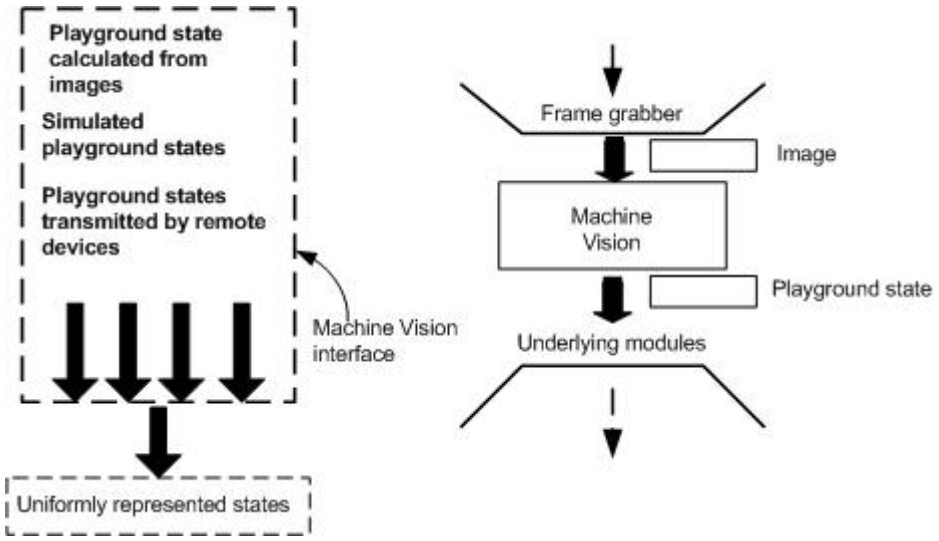


Figure 3.3: Encapsulation of vision processor module by the vision processor interface.

An image captured by the camera in general may contain several objects and, in turn, each object may contain several regions corresponding to parts of the object. Variations in characteristics of the same object over different areas of the image are relatively "controllable"' in robot soccer since rules allow suitable colour codings to be used. The problem still poses some difficulty since variations do occur in luminous intensities throughout the playground. The basic technique used in deciphering the playground state from images from the frame grabber consists of two stages:

1. Identifying the regions corresponding to the ball and markers on the robots.
2. Calculating information such as ball position, ball velocity and robot orientations from the geometric characteristics of the identified regions.

Image segmentation is a typical major bottleneck in vision algorithms that arises both due to the necessity of the task as well as having to traverse the entire image for identifying logically connected regions. The reference implementation of Aneka uses constructs called windels (i.e., "window of pixels") to simultaneously achieve the conflicting goals of accuracy in and speed during image segmentation stage of machine vision.

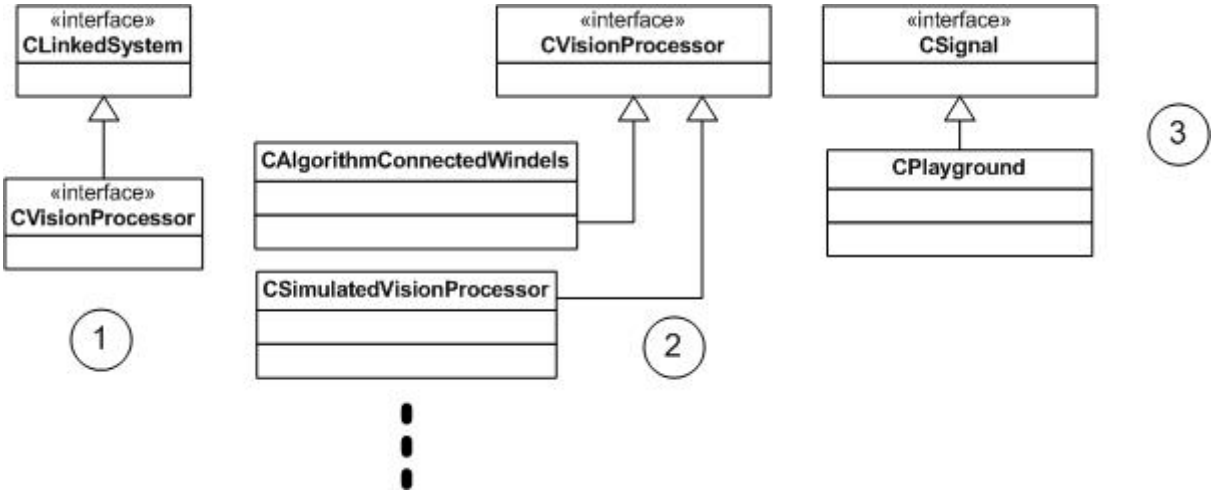


Figure 3.4: The machine vision interface extends the linked system interface. All vision processors in the system must implement the machine vision interface, and output model of the environment as playground objects. Playground itself implements the signal interface provided by Aneka.

Most approaches in image segmentation fall under techniques that use colour pixel-based segmentation, boundary estimation using edge detection, or a combination of both [17] [18] [16] [15]. The windel based approach discussed here is a straightforward generalisation of a region-based connected components algorithm. Intuitively, windel based detection of connected components sacrifices some accuracy in determining intra-regional connectivity in images to gain speed by avoiding a lot of expensive non-linear logic operations by lumping pixels together rather than treating each pixel individually. The expensive per-pixel operations are replaced by a reduced number of extremely low-cost operations whose execution can be done simultaneously for different parts of the image.

A windel, $w$, is defined as a (usually small) rectangular array of $n \times n$ pixels. Each pixel $p$ is assigned a label $l$ by a labelling function $L$, that operates on a "characteristic value", $\chi(p)$, of the pixel $p$. $\chi(p)$ should be carefully chosen so as to satisfy the following conditions:

1. $\chi(p)$ should be fast to compute.

2. If two pixels, $p_a$ and $p_b$ belong to different logical regions having labels $l_a$ and $l_b$ respectively, $\chi(p_a)$ and $\chi(p_b)$ should be adequately different as well. The label of a

pixel $p$ is given by,

$$l_p = L(\chi(p)) \tag{3.1}$$

A windel's label, $l = L(w)$, is defined as simply the label that is common to a maximum number of pixels in the windel. That is, if the number of pixels having label $l$ in a windel $w$ is given by $N(l, w)$, then,

$$L(w) = l_j, \text{where}, N(l_j, w) >= N(l_i, w), i \neq j \tag{3.2}$$

The function $\chi$ chosen for the current implementation gave the YUV values of the pixel as a vector $\boldsymbol{m}, \boldsymbol{m} \in \boldsymbol{R}^3$. Labelling function $L$ of pixels was learned via a k-means clustering algorithm run on sample images. It is not considered necessary for pixels in a windel to be connected in a strict sense. In Figure 3.5, for example, all the shown windels have the same label irrespective of the connectedness of pixels within the windel.
Lumping pixels into windels gives us the following immediate advantages:

1. Segmentation procedure becomes more tolerant to errors, since one-off pixels in the windel do not influence the labelling of the entire windel.

2. Computation can be speeded up tremendously because connected components check is not done for each pixel in the image.

3. The segmentation procedure is parallisable in a very straightforward manner.

4. The labelled pixel count of the windel forms an important characteristic that can be used for discarding spurious windels.

A major disadvantage of lumping pixels into windels is that the determination of connected components ignores fine cracks in the image that run through windels. Though in an environment such as robot soccer where pixels are naturally marred by noise and liaising the significance of such cracks is relatively trivial, there are applications such as medical imaging where the information may be significant. Figure 3.5 shows four windels that have the same label even though there are disconnected regions within some of them. If these four windels occur adjacent to one another, they become part of a larger region enclosing them.

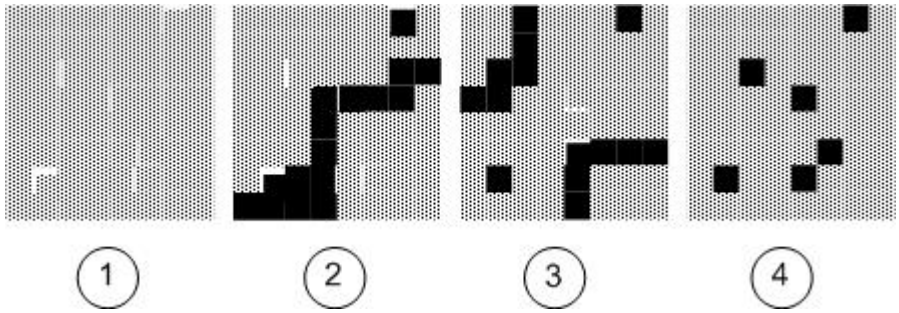


Figure 3.5: Windels have the label of the maximum number of pixels within them. Windels 1, 2, 3 and 4, for example, have the same label though there are disconnected regions within windels 2 and 3.

The windel-connected components algorithm itself is described in listing Algorithm 1. The algorithm consists of four stages:

1. Culling of background pixels.
2. Learning region-labels.
3. Determination of connected components in sub-images.
4. Coalescing regions in sub-images.

1. **Culling of background pixels :** Determine the histogram of the seed images. Remove YUV values with the largest pixel-count from consideration in determining clusters.

2. **Learning labels :** $k$-means clustering algorithm Initialize $\boldsymbol{\mu_1, \mu_2, \mu_3, ..., \mu_k}$, corresponding to the labels, $l_1, l_2, ...l_k$.

3. **Do:**

4. **For each** pixel $p_i$ in the learning sample, do:

5. Classify $p_i$ as $l_j$, where $\boldsymbol{\mu_j}$ is the closest mean to $\chi(p_i)$.

6. **For each** cluster $j$, $1 <= j <= k$, do:

7. Recalculate the mean $\boldsymbol{\mu_j}$ based on the pixels in cluster $j$.

8. **Until** no change in $\boldsymbol{\mu_j}$.

9. **Determination of connected regions in sub-images :** Divide the image into $n$ sub-images, $\boldsymbol{I_1, I_2, I_3, ...I_n}$.

10. **For each** sub-image $\boldsymbol{I_j}$:

11. **For each** windel $w_m$ in $\boldsymbol{I_j}$:

12. Calculate $L(w_m)$.

13. Identify the connected components in $\boldsymbol{I_j}$ based on the labels of its windels, $L(w_1), L(w_2)...$ The identified regions are $\boldsymbol{R_1 ... R_p}$.

14. **Coalescing regions in sub-images : For each** $I_j$:

15. Coalesce regions that touch one another and are of the same type in into larger regions.

16. **Output** the coalesced regions.

17. **End**.

Algorithm 1: Parallel windel connected components algorithm

**Background pixel culling**

In this stage, we exploit the fact that the largest chunk of pixels in a typical robot soccer playground image actually belongs to the background itself, and can be straightaway ignored from consideration when calculating ball and robot positions. The range of the background pixels in the YUV space, however, must be automatically determined. This is done by calculating histograms of a few seed images, and finding the YUV ranges that have the maximum pixel count. When creating a histogram, getting the pixel count for each possible YUV vector $[y, u, v]$ is computationally inefficient: in a typical 8-bit image, we may have to deal with 3 dimensional array of size $255 \times 255 \times 255$. Instead, we discretise the Y space into 5 units, and the U and V spaces into 16 units each to get an array of size $5 \times 16 \times 16$, which allows for greater flexibility with vision algorithms at the loss of some precision. The loss of precision is somewhat compensated by increasing the granularity of U and V values that are normally most indicative of a logical change in regions.

The result of histogram computation is a function $H : \boldsymbol{R}^3 \rightarrow \boldsymbol{R}$, mapping YUV vectors to their pixel counts. As shown in Figure 3.6, the ranges with the maximum pixel counts (i.e., pixel counts exceeding a pre-decided threshold, $N_t$) are ignored in the k-means clustering stage.

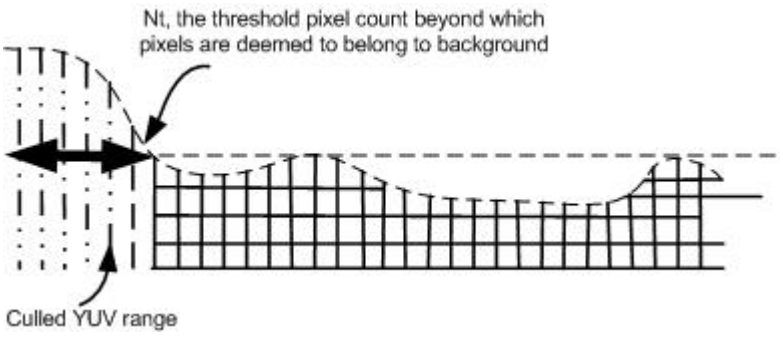


Figure 3.6: Learning of the background pixels' YUV ranges through histogram culling. YUV values with counts greater than a given threshold are assumed to belong to the background.

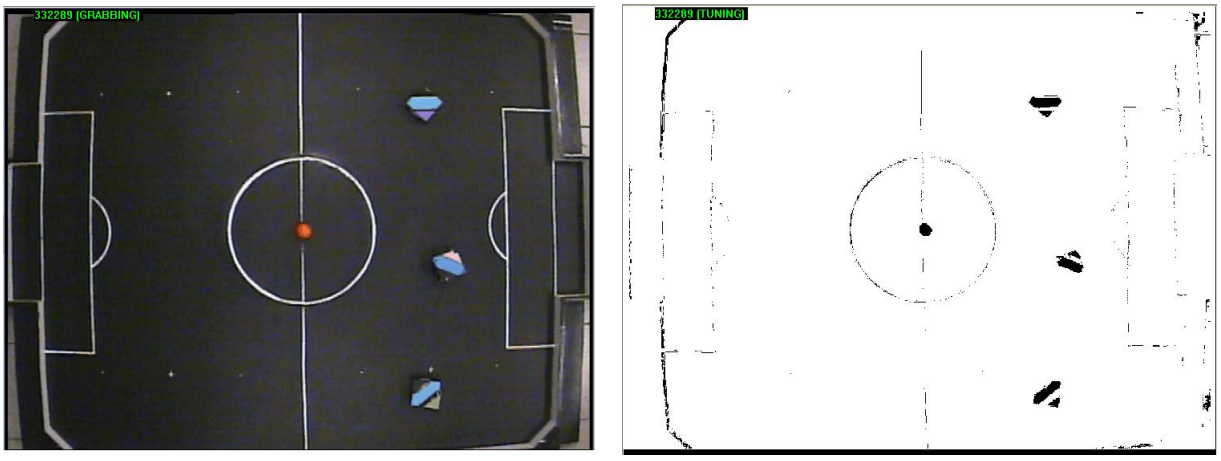


Figure 3.7: (A) A typical input image to the vision algorithm. (B) Image after background culling. Notice that due to the characteristics of images in robot soccer systems, the non-background regions are robustly identified.

**k-means clustering**

k-Means clustering is next used for unsupervised learning of clusters of pixels present in the seed images. It is assumed that the number of label types, $k$, is known beforehand. It is possible to learn this number as well to achieve optimal clustering at the cost of longer computational time.

The principle objective is to cluster a sample of pixels, $S$, into $k$ partitions to minimise the intra cluster variance,

$$V = \sum_{i=1}^{k} \sum_{j \in S_i} |\boldsymbol{x_j} - \boldsymbol{\mu_i}|^2 \tag{3.3}$$

,

where there are $k$ clusters $S_i, i = 1, 2, 3, ..k$, and $\boldsymbol{\mu_i}$ is the centroid or mean of points $\boldsymbol{x_j}$ in $S_i$.

The output of the clustering stage are characteristic values, $\boldsymbol{\mu_1, \mu_2, \mu_3, ..., \mu_k}$, corresponding to labels, $l_1, l_2, ..., l_k$. The meanings given to the labels depend on identification of the clusters by an observer. Thus, the algorithm may correctly cluster all pixel colours

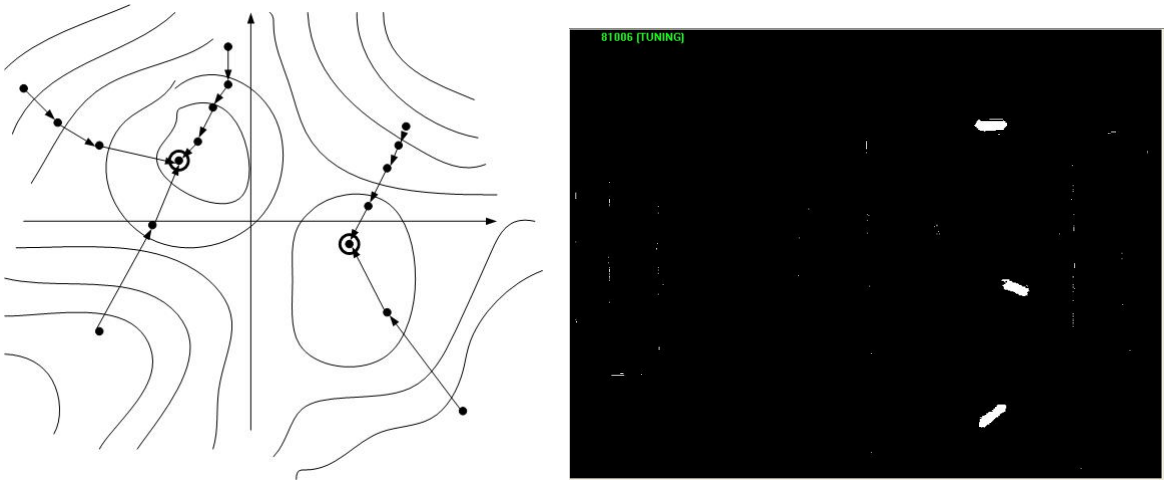


Figure 3.8: (A) k-Means clustering is used as an inexpensive unsupervised clustering algorithm. The figure shows movements of two means (reduced to two dimensions for depiction) to stable values over a few iterations. (B) Home team colour thresholded using the values learned from k-Means clustering.

belonging to the soccer ball into a single category, but it cannot determine that the colours actually denote the ball and not some other object on the playground. A major advantage of incorporating an automatic parameter learning stage is that the vision algorithm reacts robustly to minor changes in luminosity as the game progresses by running the algorithm whenever the objects on the ground cannot be determined accurately.
Once the characteristic values of regions are learned, they are retained until inaccuracies surface in playground parameter estimation.

#### Parallel connected components algorithm

To determine connected regions in intermediate images, the image is first divided into smaller sub-images that can be parallely processed. The number of divisions made depends on various factors such as the number of processors available and the communication overhead involved in distributing and collating data. Each sub-image is sent out for calculation of connected regions *within* the sub-image to the processors available for vision processing. Once the results get back to the processor issuing the request, the neighbouring connected regions of the same type are stitched together to get the full regions.

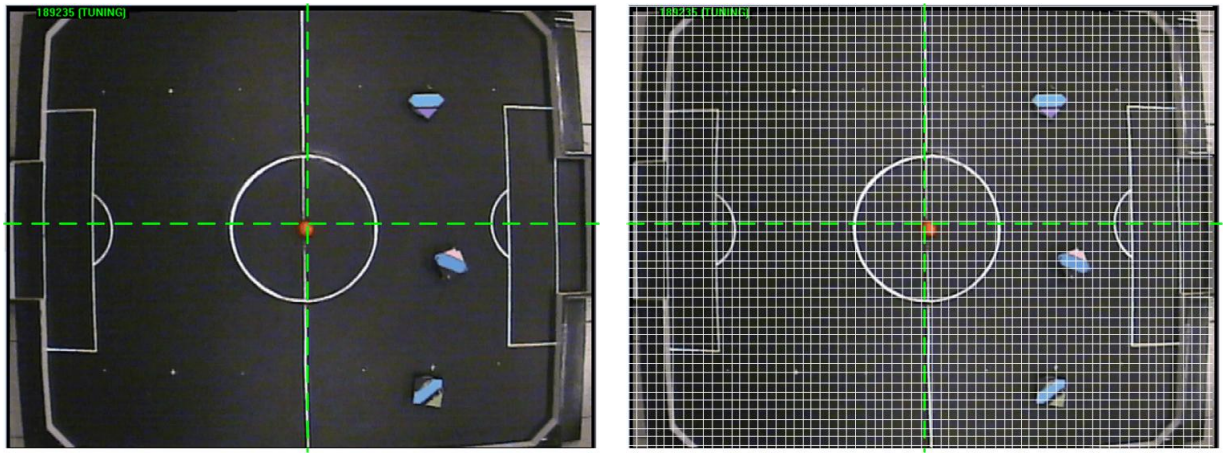

Figure 3.9: (A) The input image split into four sub-images. (B) Windels of the image.

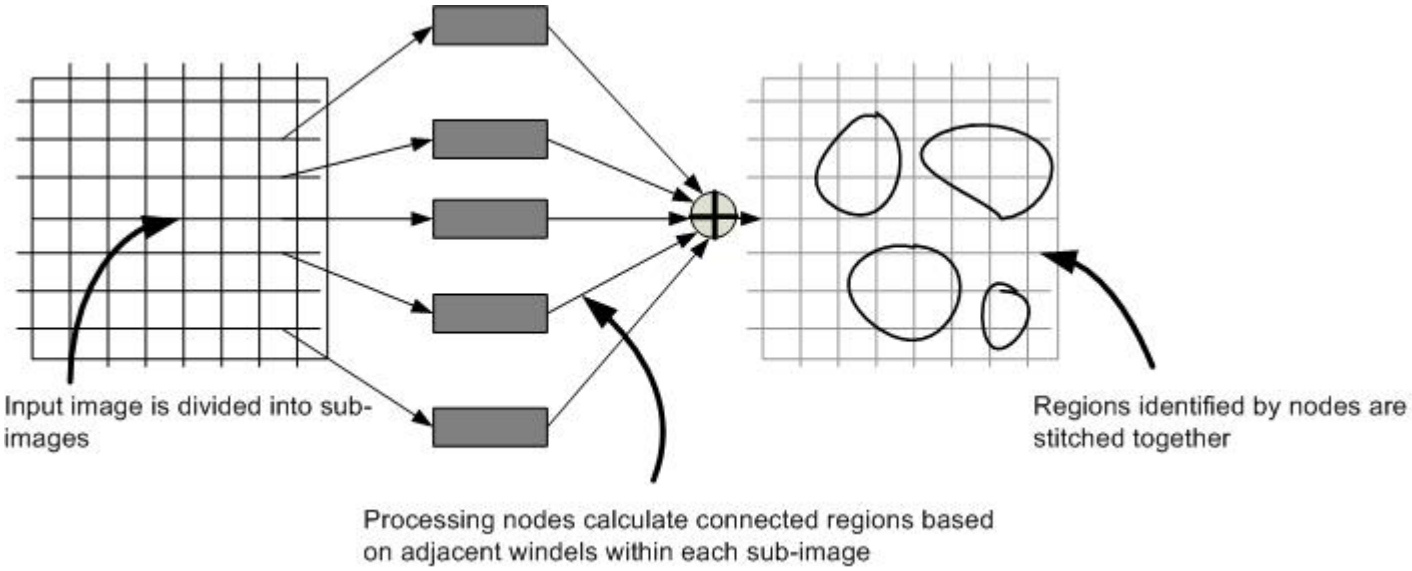


Figure 3.10: Overall flow of data in the parallel connected components implementation.

Within a sub-image, connectivity of a windel A is determined as described in Algorithm 2.

1. Assign a new region $R$ to $A$.
2. **Let** list $l = (A1, A2, A3, A4)$, the neighbouring windels of A (Figure 3.11).
3. **For each** windel $w_i$ in list $l$:
4. **If** $L(w_i) = L(A)$, (i.e., if windels both are of the same type):
5. **If** $A$'s current region is $R1$ and $w_i$'s region is $R2$, and $R2 \neq R1$, merge $R1$ and $R2$.
6. **Output** The final region $R$ of $A$. $R$ would contain all windels thus far discovered that are connected to $A$.
7. **End**.

Algorithm 2: Determining connectivity of windels within sub-images

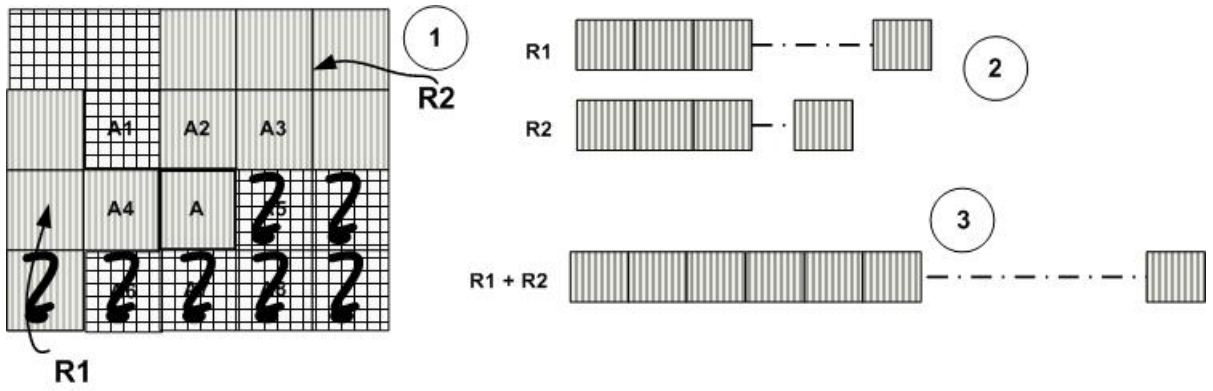


Figure 3.11: Till reaching windel A, R1 and R2 exist as separate regions. Windel A establishes connectivity, and the region lists for R1 and R2 are merged to get the new region R1 + R2. The windels shown with question marks are the ones not discovered yet.

#### Coalescing of connected regions from sub-images

Coalescing connected region information from individual sub-images to give the region information for the entire image is speeded up tremendously by working at the windel, rather than pixel, levels. Algorithm 3 describes the steps involved for a typical sub-image $C$, that is surrounded by sub-images $A, B$ and $D$ on four sides.

1. **Let** $S$ be the list of all of $C$'s windels bordering the sub-images $A, B$ and $D$.

2. **For each** windel $s_i \in S$:

3. **For each** windel $w_i \in A \cup B \cup C$ adjacent to $s_i$:

4. **If** $L(w_i) = L(s_i)$, (i.e., if windels both are of the same type):

5. $s_i$'s current region is $R1$ and $w_i$'s region is $R2$, and $R2 \neq R1$, merge $R1$ and $R2$.

6. **End**.

Algorithm 3: Coalescing connectivity information from sub-images

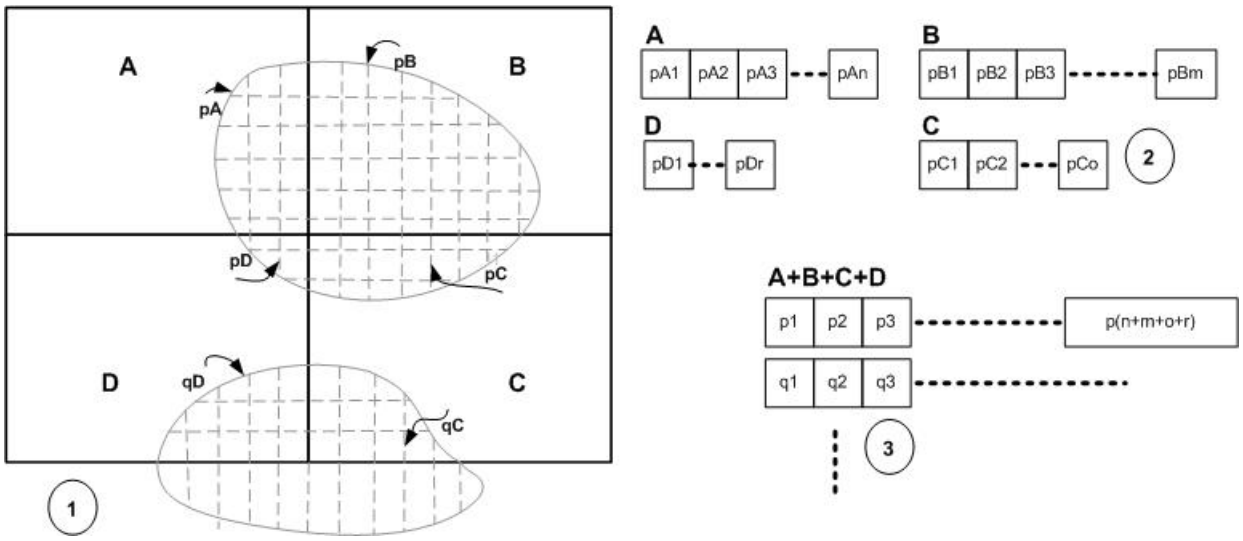


Figure 3.12: In general, when coalescing connectivity information from a sub-image $C$, portions of a region in $C$ may be present in any of its neighbouring sub-images. The coalescing algorithm stitches together individual regions in the sub-images to give the final regions of types $p$ and $q$.

### 3.2.2 Improving the robustness of machine vision

Typical outputs of the windel connected components algorithm are shown in Figure 3.13. A common problem in the machine vision vision stage is that regions that do not belong to objects of interest could nevertheless share their colour characteristics. For example, in Figure 3.13 B, the white line markings on the ground are wrongly identified as being of the same region type as that of the ball. This could arise due to several reasons, two

major ones being noise and aliasing in the input image, and wrongly learned colour characteristics of regions.

Filtering the incoming images could be done to remove typical noise pixels, but this is usually an expensive operation unless dedicated hardware is available. A few inexpensive heuristic measures can be readily applied to remove spurious windels from consideration while calculating object positions and orientations:

1. If the number of pixels having the label $l = L(w)$ within the windel $w$ is below a minimum threshold, the windel is discarded.

2. The connected region's total pixel count (i.e., area) should lie between acceptable thresholds specified by an observer during the clustering stage. Regions above or below this range are discarded.

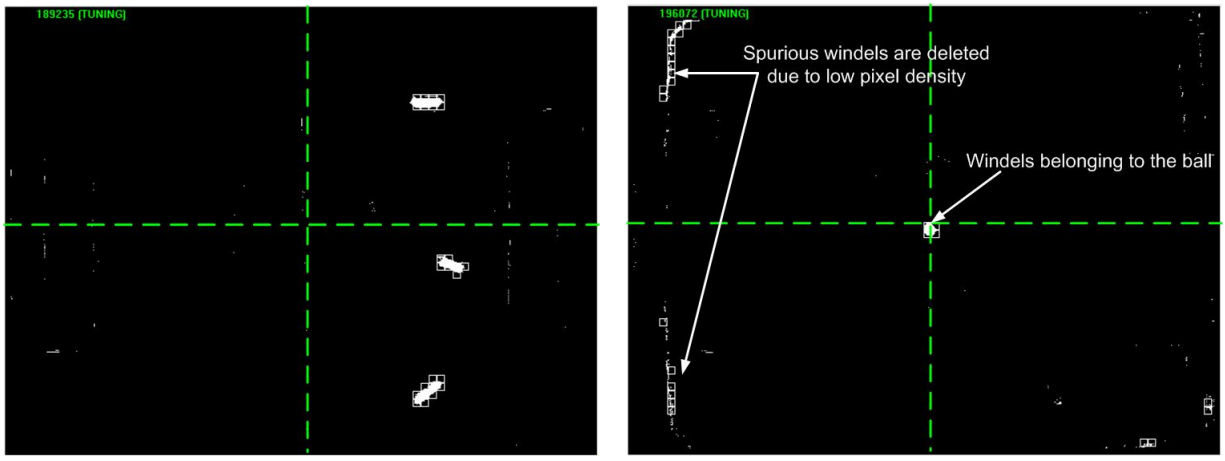


Figure 3.13: (A) Connected windels beloging to the home colour region. (B) Regions identified as belonging to the ball. Spurious regions can be identified by their low pixel density, thereby increasing the robustness of detection.

### 3.2.3 Calculation of object positions and orientations

Once the regions are identified, calculation of object positions and velocities is relatively straightforward. The position of the ball is simply the centroid of the region identified as belonging to the ball. Determination of the positions and orientations of robots depend

on the specific colour scheme being used. In the reference implementation, robots were coloured coded using a major team colour, and a minor player ID colour as shown in Figure 3.14. The robot positions are given by the centroids of the major team-colour region, and orientations by the line formed between the centroids of major and minor colours. Typical final outputs of the vision algorithm are shown in 3.15.

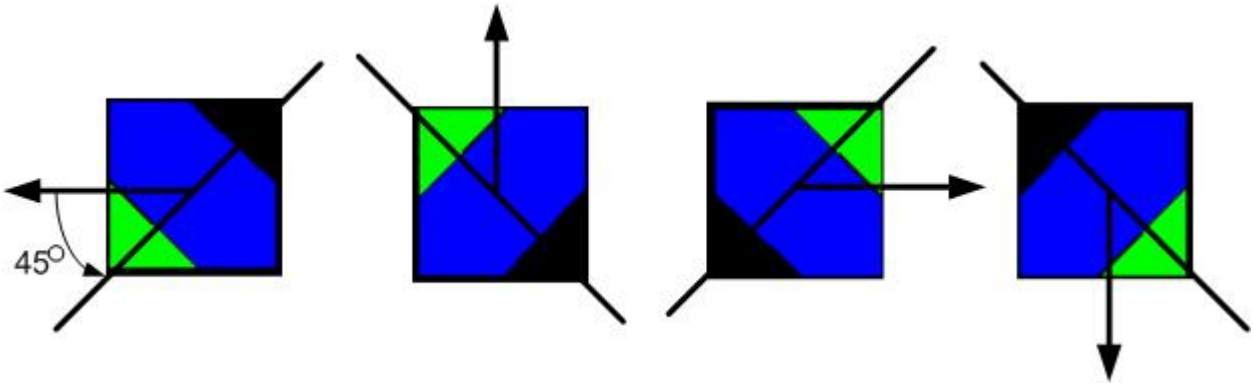


Figure 3.14: Measurement of robot orientation from identified image regions. Orientation calculations typically depend on an assymetry present in robot colour coding. Colouring scheme used in the reference implementation allowed large areas for denoting regions on a robot.

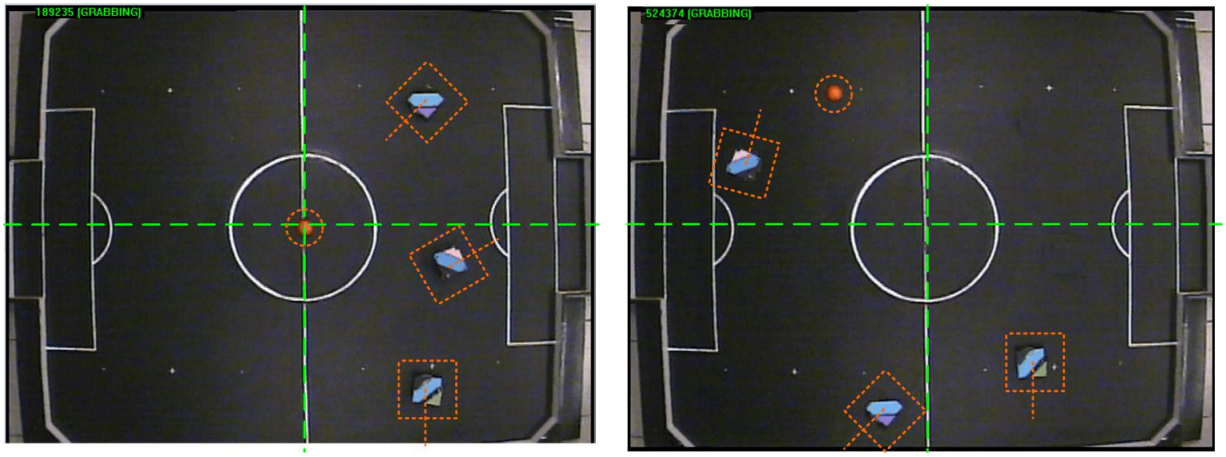


Figure 3.15: The final processed image with the identified objects.

# Chapter 4

# Controller and Communication

This chapter discusses implementations of the controller and communication modules of a robot soccer system within the Aneka framework. While the roles played by these modules are vastly different from those of image acquisition and processing modules, the interfaces at the highest levels remain the same.

## 4.1 Controller

Aneka models the controller as a module distinct from the perception and action modules, as is common in a conventional robotic system with separate perceive-plan-act cycles. The role of the controller is to accept the environment state (in the case of a robotic soccer system, the state of the playground represented by positions, orientations and velocities of the robots and the ball), and process it to produce control actions which are then applied on the system's actuators.

Controllers are represented in software as linked systems (class `CLinkedSystem` in the reference implementation) that accept and output signals (class `CSignal`). Aneka allows for the inclusion of a wide variety of controllers by defining strict interfaces through which the controllers must interact with the rest of the system, but not imposing any further conditions on the implementations of the controllers themselves. The domain level interfaces present in the reference implementation are `CController`, which encapsulates a

generic controller of a robot soccer system, and `CControlAction`, which encapsulates the signal produced by the controller (Figure 4.2). The inputs to the controller are playground state signals (`CPlayground`) generated by the vision processor, and the outputs are control actions that are applied to the system's communication module.

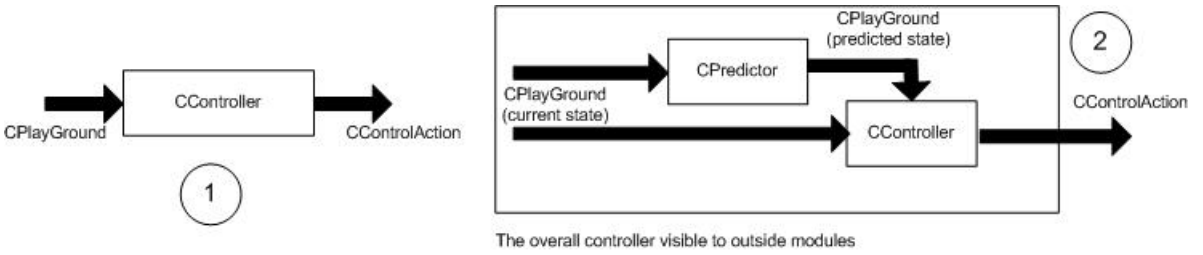


Figure 4.1: Controllers are modelled as linked systems that accept playground state as input and output control actions. Controllers themselves may be composed of sub-systems, such as a predictor of playground states.

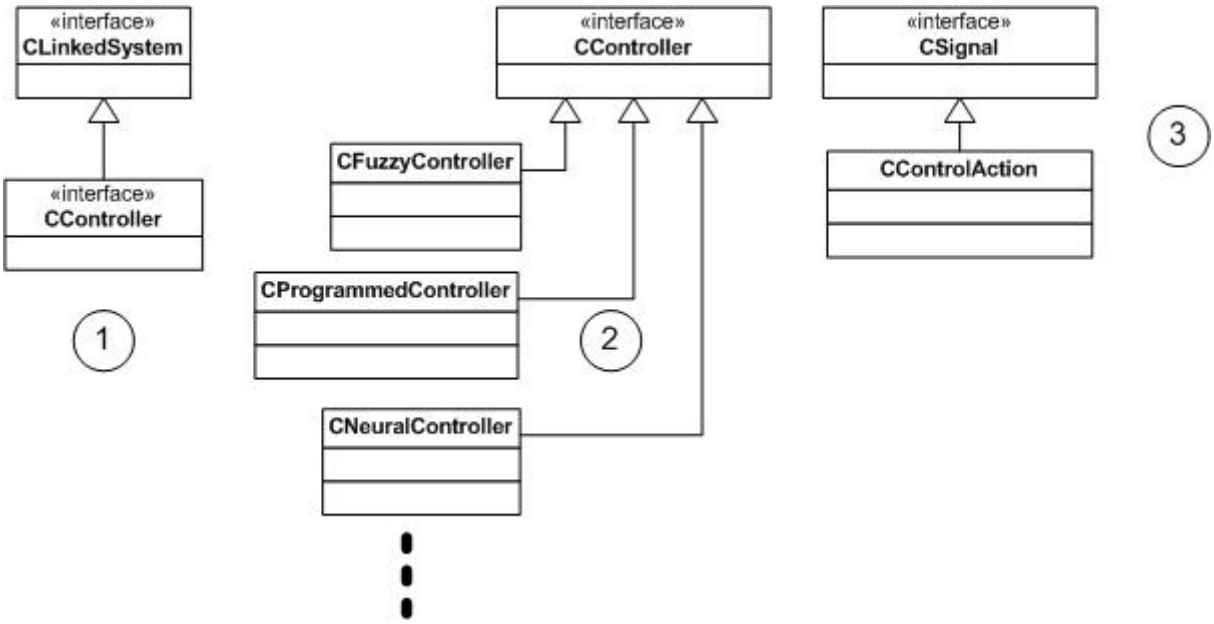


Figure 4.2: Different control strategies can be accommodated as long as they conform to the predefined domain specific interface, CController, and output actuating commands as CControlAction.

Figure 4.2 shows how various fuzzy logic, neural network, or programmed controllers can be used to implement controllers by extending the domain level interface `CController`. The implemented controllers themselves need not be monolithic modules, but may utilize


Figure 4.3: Various control strategies can be transparently applied since the controller communicates with the other modules through the Aneka interface.

a further combination of linked systems internally to accomplish the control tasks (Figure 4.1). Perception and action modules interacting with the control system, however, are hidden from any complexities that are internal to the overall controller.

The rest of this chapter investigates implementations of a low level PID path planning controller, a programmed controller, and a simulation environment within the Aneka framework. The discussion clarifies how their inclusion is facilitated by the interfaces provided by the framework.

### 4.1.1 Implementation of a PID based motion controller

Controlling the low-level motion of the robots along a specified path is often a requirement irrespective of the higher level decision making strategies used. This section describes the implementation of a simple PID based control scheme within the Aneka framework. Figure 4.4 depicts how the controller fits into the Aneka framework.

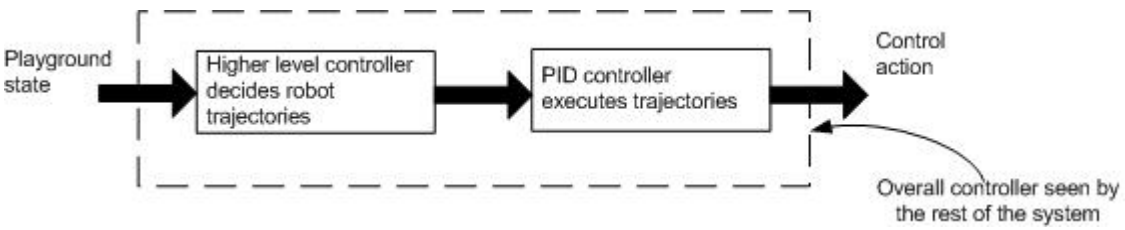


Figure 4.4: Construction of a controller divided into higher and lower level decision making modules. The PID controller controls the motion of the robot through intermediate points specified by the higher level trajectory generator.

PID controller is modelled as a linked system that takes in trajectory specifications

(represented by class `CTrajectory`) as signals and outputs the final control actions, viz., the right and left wheel velocities of the robots. PID controller alone is ineffective as a whole control system since it does not directly accept playground state as input, but it is nonetheless a core portion of many intelligent higher level controller schemes.

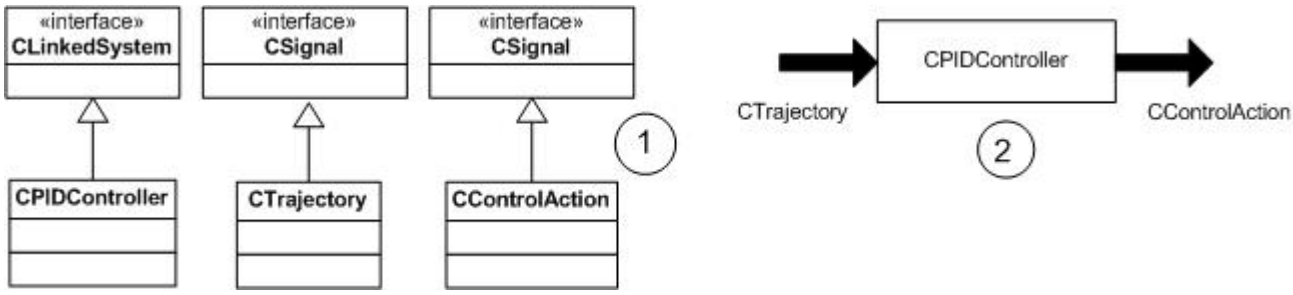


Figure 4.5: The PID controller is modelled as a linked system that takes in signals of the form trajectory, and outputs control actions specified by the left and right wheel velocities of the robots.

The robots used in Aneka are two-wheeled mobile robots (WMR) utilizing differential drives. The posture of the robot, $P$, can be described using the position $(x, y)$ and the orientation angle, $\theta$ . This configuration is shown in 4.6.

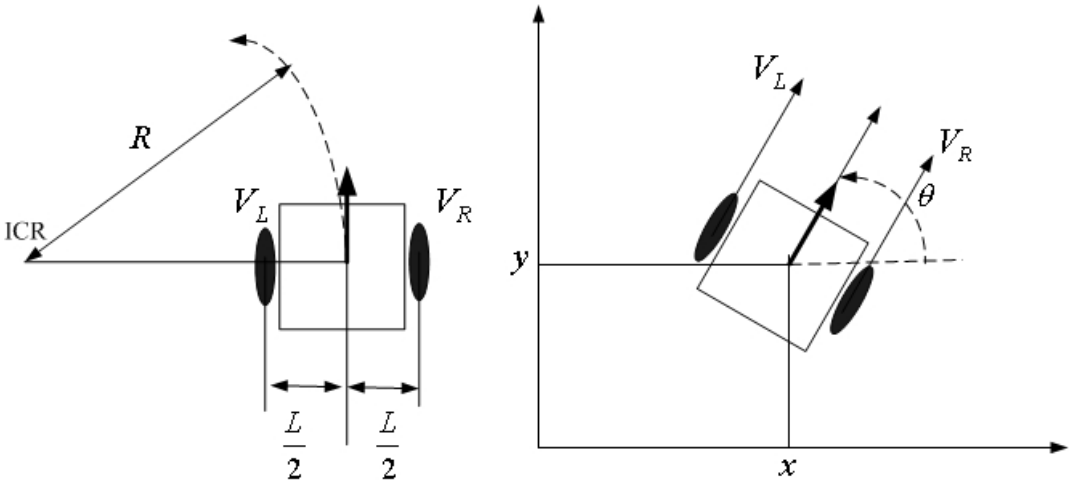


Figure 4.6: Posture of robots in a robot soccer system.

In the system, the right and left wheel velocities ($v_R$ and $v_L$) constitute the control input to the robots. The linear velocity, $v$, and angular velocity, $\omega$, of the robot are related to the control inputs as:

$$v = (v_R + v_L)/2 \tag{4.1}$$

$$\omega = (v_R - v_L)/2r_w \tag{4.2}$$

,where $r_w$ is the distance of the wheel to the centre of the robot.

With the linear and angular velocities of the robot, the kinematical equations are obtained as:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} cos(\theta) & 0 \\ sin(\theta) & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ w \end{bmatrix} \tag{4.3}$$

Combining 4.1, 4.2 and 4.3, we get:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} cos(\theta) & 0 \\ sin(\theta) & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} (v_R + v_L)/2 \\ (v_R - v_L/2r_w) \end{bmatrix} \tag{4.4}$$

Thus, by setting $v_R = v_L$, the robot moves in a straight line since the change in orientation $\dot{\theta} = 0$. Setting $v_R = -v_L$ enables the robot to spin in place. The equations govern the kinematic behaviour of the robots and are used extensively to calculate the resulting posture $(x, y, \theta)$ with a given set of control inputs. However, the velocity of the wheels exhibit transient behaviour before reaching the control inputs due to the dynamics of the robot motors.

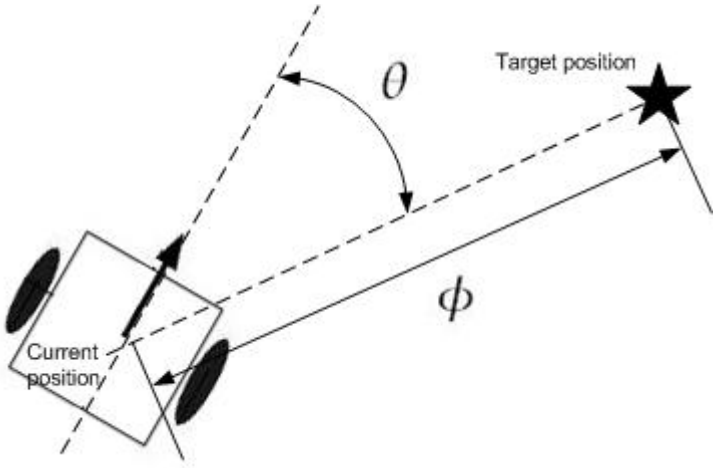


Figure 4.7: The error inputs for a low level PID path-planning controller.

The error inputs to the PID controller are the distance error ($\phi$) and the angle error ($\theta$) as shown in Figure 4.7. For a simple controller that uses only proportional control, the control outputs $V_L$ and $V_R$ are calculated as follows:

$$V_L = k_{dist}\phi - k_{angle}\theta \tag{4.5}$$

$$V_R = k_{dist}\phi + k_{angle}\theta \tag{4.6}$$

, where $k_{dist}$ and $k_{angle}$ are the proportional controller coefficients for the distance error $\phi$ and the angle error $\theta$ respectively. Thus, the distance error determines the base velocity of the robot, and the angle error determines the velocity difference between the wheels. The goal is to arrive at values for the gain that will cause the robots to reach the point quickly and with least overshoot.

Integral and derivative controllers too could be included in the above scheme to achieve better controller performance. Taking a base velocity $V_C$ to be constant, the control laws for the left and right wheel velocities are calculates as in a PID controlled scheme is given by:

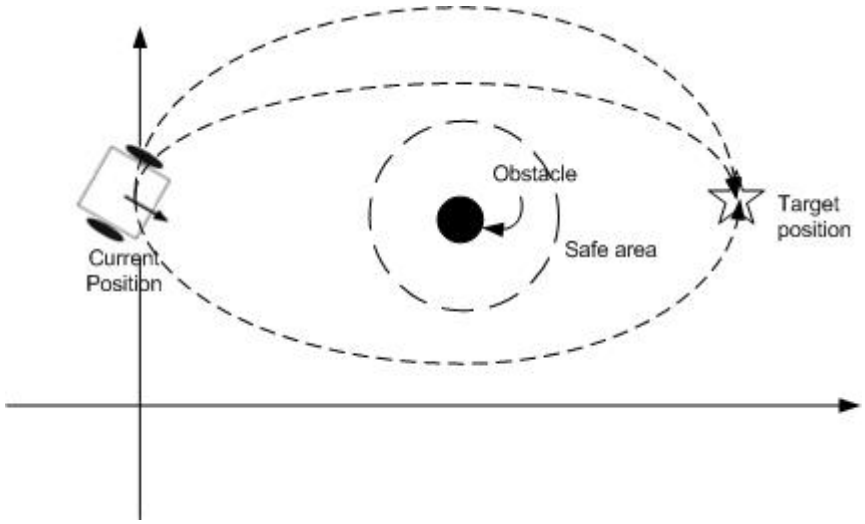


Figure 4.8: A generic depiction of an obstacle avoidance problem in robot soccer. The robot's goal is to follow a smooth trajectory that avoids the safe area and reach the target.

$$V_L = V_C - (k_P\theta + k_D\frac{d}{dt}\theta + k_I\int_0^t \theta dt) \tag{4.7}$$

$$V_R = V_C + (k_P\theta + k_D\frac{d}{dt}\theta + k_I\int_0^t \theta dt) \tag{4.8}$$

The input to the PID controllers is a trajectory generated by higher level modules. Trajectory planning in robotic systems is a reasonably broad area in itself, but for the purpose of demonstration, a polynomial based scheme is used in the sample PID controller.

Consider, for instance, a common sub-problem of obstacle avoidance that has to be solved several times in the course of a normal soccer game. A static depiction of the problem is given in Figure 4.8. The controller's aim is to cause the robot to maneuver around an obstacle to reach a target location - which need not be stationary - that lies across the obstacle. Figure 4.9 depicts the result of a PID controller used to achieve trajectory-tracking in simple situations.

The simplest way to generate a smooth trajectory that avoids the given obstacle would be to generate the intermediate points to fall on a second order polynomial curve given by,

$$y = a_2x^2 + a_1x + a_0 \tag{4.9}$$

,with the starting and the ending points as shown in Figure 4.8. The three parameters, $a_0$, $a_1$ and $a_2$ are fully determined if a third point is arbitrarily chosen to fall outside the safe area of the obstacle. The safe area would be large if the obstacle has been moving while the trajectory is being planned. Figure 4.9 show execution of the PID controller for a sample set of target points.

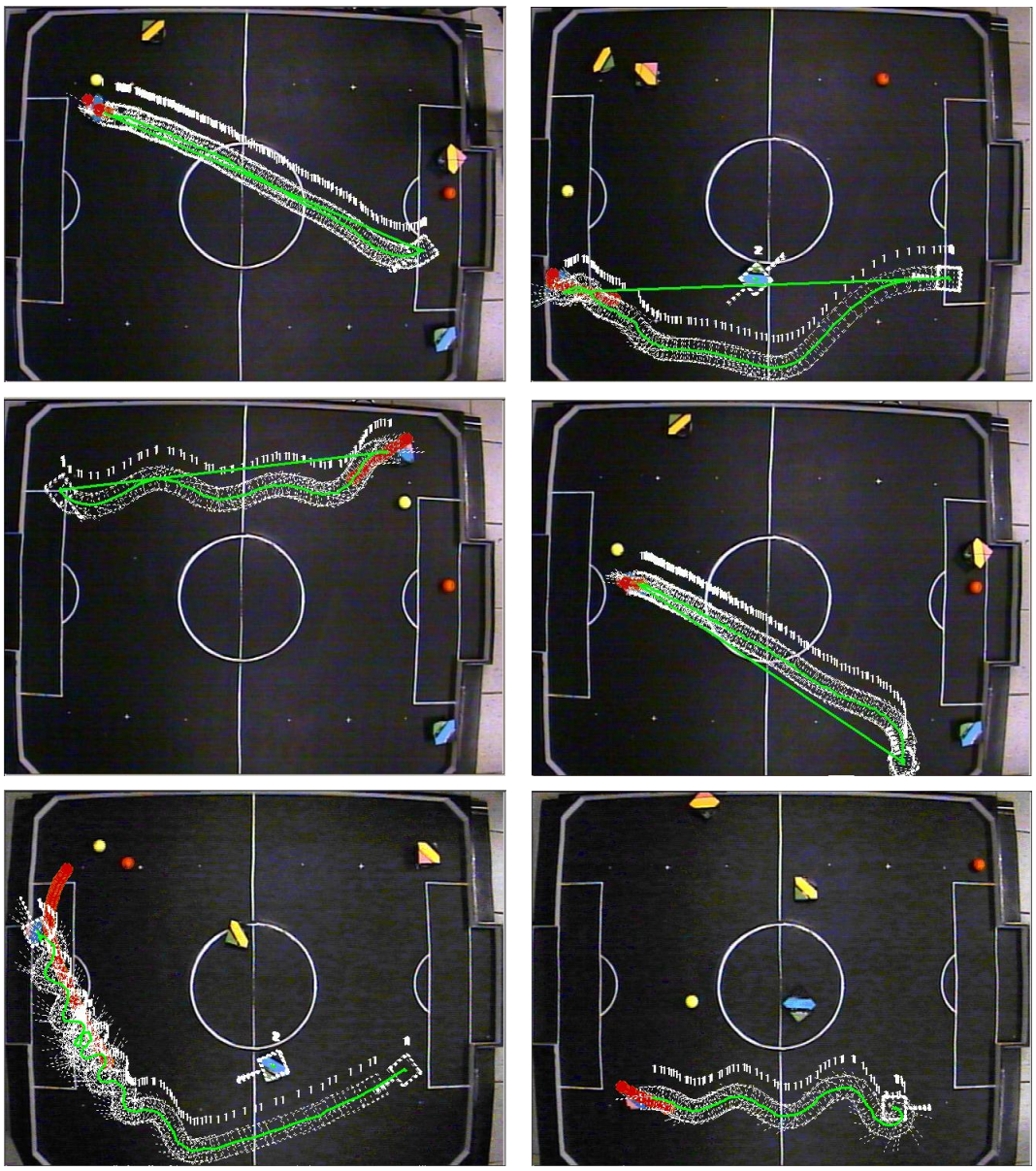

Figure 4.9: Low-level control of robot motion through PID controllers.

### 4.1.2 Interpreted Programming Environment

Higher-level decision making can be implemented in several ways. One of the more straightforward ways to generate complex control actions in a robotic system is to model a human operator's heuristics through a convenient programming language. While the control strategy can be hard coded with the entire system, it is often desirable to be able to modify heuristics rapidly on the fly until effective strategies are found. This makes the option of an interpreted programming environment for robot systems attractive.

This section discusses the implementation of an interpreted programming environment within the Aneka framework. The programming language used to code the semantics of robot behavior, called the Aneka Programming Language, loosely resembles the C programming language in syntax with robot soccer specific constructs included. The language allows the control-strategy designer to specify several trigger conditions and actions that should be taken when the conditions are met. The programmed controller, represented by `CProgrammedController`, is implemented as a linked system (Figure 4.2) that accepts playground states as inputs, and generates low level trajectory (`CTrajectory`) as outputs. The trajectory is then passed to the low level controller for translating them into wheel velocities that are followed by the robots. In the reference implementation, trajectory information generated by the programmed controller does not contain target velocity information for intermediate points in a trajectory. This is arguably a major handicap in real world where the robots must maintain a smooth motion while simultaneously following a prescribed path. The collection of low level control actions available to the high level layer are also extremely limited, and conceivably, more interesting behaviours could be added. However, the default implementation does serve its purpose in demonstrating inclusion of sophisticated controllers within the Aneka framework.

The programming environment consists of an editor through which the statements are entered by a human operator, an interpreter that compiles the high-level statements into byte-codes, and a virtual machine that runs the byte codes by linking them to built-in

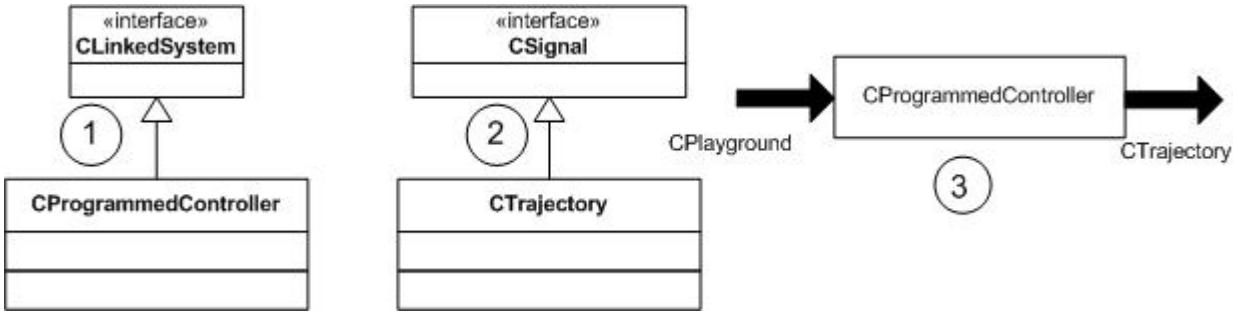


Figure 4.10: The programmed controller is modelled as a linked system that accepts playground states as input signals, and generates trajectory information as output signals.

lower level control functions 4.11. The virtual machine provides a fund of lower level actions that can be invoked using specific keywords, such as `guard`, `within` and `pass`. Programmed controllers can be very effective in generating sophisticated behaviours by stitching together simpler behaviours, but can break down easily if conditions not taken care of in the program are met with in the real world.

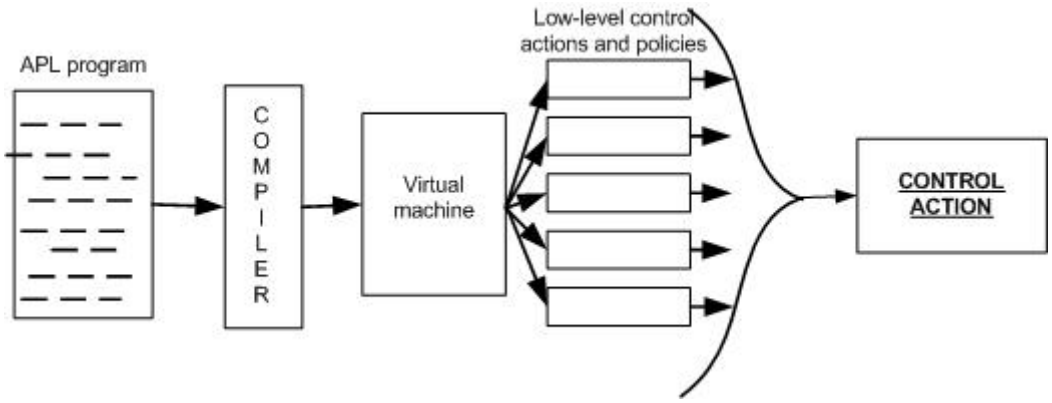


Figure 4.11: Implementation of the interpreted environment.

### 4.1.3 Simulator

A simulator is usually included in robotic systems to function as a surrogate for the real-world, allowing the controller to test out the efficacy of control strategies without having to actually take action. This section details the implementation of a simple simulator within the Aneka framework.

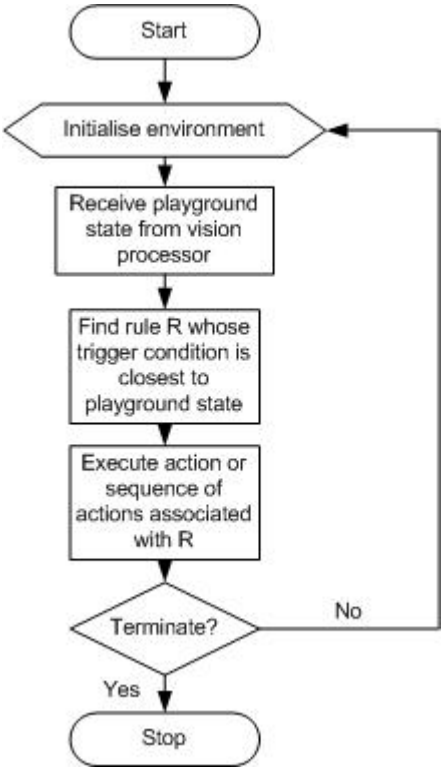


Figure 4.12: Execution of the virtual machine. The actions whose pre-conditions are closest to the existing playground state are executed in each cycle.

```
initbot(0,37.5,45);     /* initialize bot 0 position*/
initbot(1,37.5,65);     /* initialize bot 1 position*/
initbot(2,37.5,85);     /* initialize bot 2 position*/
gamespeed 50;           /* initialize game speed */

begin rules
{
      bot1 as goalkeeper;     /* assign bot 1 as goal keeper */

      if(ballx < 75) /* bally also available */
      {
            bot0 guard 40,45 range 60;    /* defend position */
            bot2 guard 40,85 range 60;
      }

      if(bot0 within 10 ball && ballx >75)
      {
            bot2 guard 110,65 range 10; /* attack position */
            bot0 pass bot2;             /* passing position */
      }

      /* more rules.. */
      if(ball within 20 opp_post)
      {
            begin sequence
            {
                  move bot(X)
            }
      }
}
```

```
LOADR 2 1
LOAD 1 37.5000
LOADR 3 1
LOAD 1 85
INIT 2 3 1
LOAD 1 10
GS 1
DISP
GK 1
BALLX
LOADR 2 1
LOAD 1 75
LOADR 3 1
LT 1 2 3
JZ: 1
LOAD 1 40
LOADR 2 1
LOAD 1 45
LOADR 3 1
LOAD 1 60
GUARD 0 2 3 1
LOAD 1 40
LOADR 2 1
LOAD 1 85
LOADR 3 1
LOAD 1 60
GUARD 2 2 3 1
JTO: 1
```

Figure 4.13: (A) A sample APL program fed to the runtime. (B) Section of the corresponding virtual machine code generated.

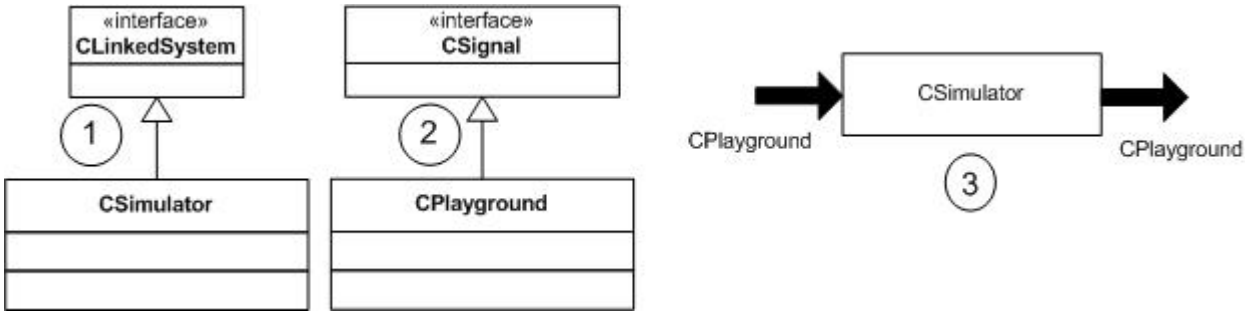


Figure 4.14: The simulator is implemented as a linked system that accepts and outputs playground states. In addition to the input signals, several parameters such as coefficients of friction and elasticity of collistions between different surfaces may also be used to configure the simulator.

The simulator in the reference implementation is modelled as a linked system that accepts and outputs signals in the form of playground states. When run in a stand-alone mode, the simulator simply iterates through the states starting from an initial input state, showing the state-trajectory of the environment visually through a GUI display. The simulation engine `CSimulator` can also be linked to the Aneka system to act as either a predictor of states in machine vision and control modules, or as a replacement for the real environment.

#### Model of the robot soccer environment

The robot soccer environment as modelled by the Aneka simulator consists of two fundamental units - the ball and the robot. A realistic simulation of the environment requires that the kinematics and dynamics of these units be modelled as accurately as possible, while constraints of computational resources and mathematical complexity of models significantly limit the fidelity of the simulator.

The robot motor dynamics can be modelled as a second order system, having the standard form,

$$\frac{1}{\omega_n^2}\ddot{x} + \frac{2\zeta}{\omega_n}\dot{x} + x = f(t) \tag{4.10}$$

, where $f(t)$ is the input function, $\omega_n$ is the undamped natural frequency of the system, and $\zeta$ is the damping coefficient.

A further simplification can be made if a first order model is used to simulate the robot dynamics, using the transfer function,

$$\frac{Y}{U} = \frac{1}{1 + s\tau} \tag{4.11}$$

, where $Y$ is the output, $U$ is the input, and $\tau$ is the time constant of the system. The time constant used in the simulator was obtained by observing the actual behaviour of the robots on the playground.

An onboard PID controller is used to control the motors according to the reference velocity inputs. Combining the PID parameters, the model of the robot dynamics becomes,

$$\frac{u}{e} = K_P + \frac{T_D s}{1 + (T_D s)/N} + \frac{1}{T_I s} \tag{4.12}$$

, where $u$ is the control input,and $e$ the error input to the controller.

PID tuning is done through Ziegler-Nichol's Open Loop tuning method. The values obtained for the PID gains, and the corresponding system response is shown in Figure 4.15.

The ball used in the real environment is an orange golf ball of 42.7mm in diameter and 46g in weight. The golf ball used in the competition has dents that are well distributed throughout the ball, but these details are ignored in simulation. This simplifies the ball model greatly although the simulation effect would have some deviations from the actual results.

The ball has two modes of free motion: pure rolling without skidding and rolling with skidding. Under normal circumstances, the ball only rolls. During rolling, the ball follows a straight line in the direction, $\theta$. It slows down because of a rolling friction coefficient $\mu_R$ between the ball and the soccer field. The linear motion can thus be described by the

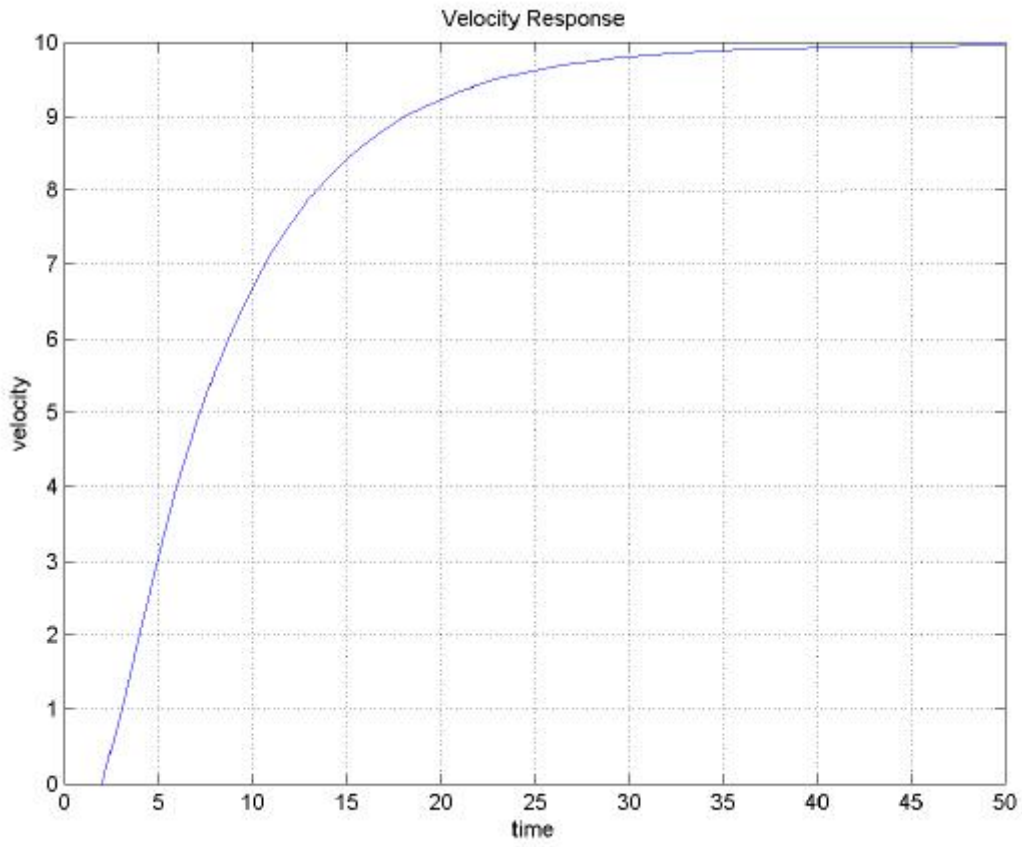


Figure 4.15: Response of the simulated onboard motor controller after PID tuning.

equations,

$$a = -\frac{f}{m} = -\mu_R g \tag{4.13}$$

$$v(t+1) = v(t) + a\Delta t = v(t) - \mu_R g \Delta t \tag{4.14}$$

, where $\Delta t$ is the time elapsed between consecutive simulation iterations, $v(t)$ is the linear velocity of the ball at time $t$, and $g$ is the acceleration due to gravity. Since negative values of velocity make little sense in the above equations, they are clipped to zero. Models that are more complex can be used where the fidelity of simulation is of critical importance [35].

The simulator must also take into account interactions among particles, and consequently, the issue of collision detection naturally arises. Collision detection as an independent area in its own right has been widely investigated in literature. Examples include the Lin-Canny closest features algorithm [36], I-Collide [37] and OBB-tree [38] [35] [39]. The Lin-Canny algorithm maintains the pair of closest features (vertices, edges, or faces) between two convex polyhedral moving through space. By exploiting the fact that the current closest features are probably near the previous closest features, an almost constant query time is achieved in practice. The distance between two polyhedral is easily found once the closest features are known. A collision is declared when this distance falls below some limit.

The simulator uses an approach based on OBB trees, primarily because it is relatively simple to implement and gives an acceptable level of accuracy. An OBB-Tree is a hierarchical representation using oriented bounding boxes. An oriented bounding box, OBB, is a rectangular bounding box at an arbitrary orientation in 3-space. In an ideal case, the OBB would be oriented such that it encloses an object as tightly as possible. Thus, the bounding box is the smallest possible bounding box of arbitrary orientation that can enclose the geometry in question. When compared with Axis-Aligned Bounding Boxes (AABB), OBB generally allow geometries to be bounded more tightly with fewer boxes.

In order to speed up the process, precise detection is not done for all particles. The detection procedure is divided into two phases. The first phase tests if a collision is possible between two particles using a simple algorithm. If this test is successful, a more precise calculation is done to see if collision has indeed occurred. Thus, the more costly algorithm in the second phase is only used if there is a possibility of a collision. If the particles are extremely far apart, they will fail the first phase test.

In the first phase of collision detection, the Axis Aligned Bounding Boxes (AABB) method is used. This involves using a bounding box around the particle, which encom-

passes the entire particle without consideration of its orientation. In the case of the robot, the bounding box has the diameter equivalent to the length of the diagonal of the robot. For the ball, the bounding box has the diameter equivalent to the diameter of the ball.

In the second phase of detection, oriented bounding boxes (OBB) are used. To employ this technique, the vertices of the robots have to be tracked. For collisions between robots, if the first phase collision test is successful, the four vertices of the robots involved are tested to see if the point is contained within the boundaries of the other robot using the Jordan Curve Theorem (i.e., a point is inside a polygon if, for any ray from this point, there is an odd number of crossings of the ray with the polygon's edges.)

The OBB of the robot is more complex since it is a polygon. However, for collisions between the spherical ball and a robot, it is adequate to only check the distance from the ball to the edges of the robot. By employing this 2-phase detection method, the computational cost of collision detection can be reduced while still maintaining an acceptable level of accuracy.

A collision may cause velocities of the involved particles to change instantaneously. From conservation of momentum and energy considerations, we have

$$\begin{aligned} m_1 \times v_{i_1} + m_2 \times v_{i_2} &= m_1 \times v_{f_1} + m_2 \times v_{f_2} \\ \tfrac{1}{2} m_1 \times v_{i_1}^2 + \tfrac{1}{2} m_2 \times v_{i_2}^2 &= \tfrac{1}{2} m_1 \times v_{f_1}^2 + \tfrac{1}{2} m_2 \times v_{f_2}^2 \end{aligned} \tag{4.15}$$

, where $m$ denotes the mass of colliding particles, and $v_i$ and $v_f$ are the initial and final velocities of the particles. By defining the elasticity of collision, $e$,

$$e = \frac{v_{2_f} - v_{1_f}}{v_{2_i} - v_{1_i}} \tag{4.16}$$

,

, the new velocities of the particles can be calculated as,

$$
\begin{aligned}
v_{x_1} &= \frac{((e+1)\times m_2 \times v_{i_2} \times cos\theta_2 + (m_1 - e \times m_2 \times v_{i_1} \times cos\theta_1))}{m_1+m_2} \\
v_{y_1} &= \frac{((e+1)\times m_2 \times v_{i_2} \times sin\theta_2 + (m_1 - e \times m_2 \times v_{i_1} \times sin\theta_1))}{m_1+m_2} \\
v_{f_1} &= \sqrt{v_{x1}^2 + v_{y1}^2}
\end{aligned} \tag{4.17}
$$

$$
\begin{aligned}
v_{x_2} &= \frac{((e+1)\times m_1 \times v_{i_1} \times cos\theta_1 - (m_1 - e \times m_2 \times v_{i_2} \times cos\theta_2))}{m_1+m_2} \\
v_{x_2} &= \frac{((e+1)\times m_1 \times v_{i_1} \times sin\theta_1 - (m_1 - e \times m_2 \times v_{i_2} \times sin\theta_2))}{m_1+m_2} \\
v_{f_2} &= \sqrt{v_{x_2}^2 + v_{y_2}^2}
\end{aligned} \tag{4.18}
$$

, where $e$ is the elasticity of the collision, $\theta$ is the orientation of the particle, $v_x$ and $v_y$ are the $x$ and $y$ components of the velocity. The elasticity of collisions is available to the user as an adjustible parameter.

Modeling the ball collision appropriately would result in better simulation results, especially in prediction of ball location. Ball collision modelling requires greater care than that of the robot, since any change from preset values in robot's velocity imposed by the collision will be resisted significantly by the robots' onboard controllers, while the ball moves largely due to the momentum imparted to it by the robots and the walls of the playground.

The model for ball collision is adopted from Huang [40]. In Figure 4.16,

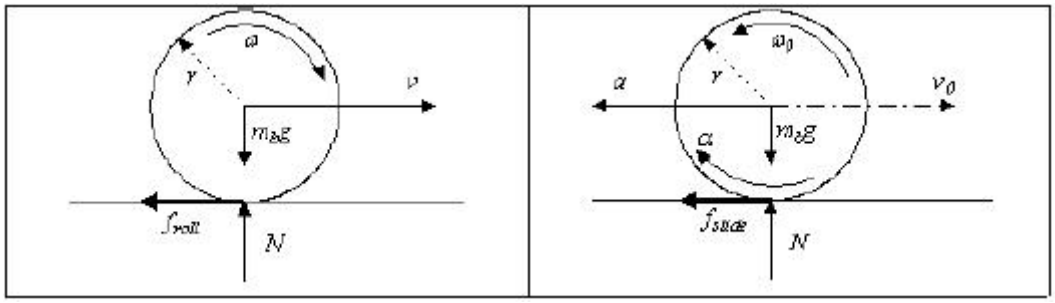


Figure 4.16: Modelling of pure rolling, and rolling with sliding.

, $\omega$ and $\omega_0$ are the angular velocities before and after collision respectively, $v$ is the linear velocity, $v_0$ is the incident velocity on the wall, $\alpha$ is the angular acceleration, $a$ is the linear acceleration, $f_{roll}$ is the rolling frictional force, $f_{slide}$ the sliding frictional force, $m_b$

is the mass of the ball, $g$ is the acceleration due to gravity, and $N$ is the normal reaction force with magnitude $mg$. The motion of the ball just after collision is governed by,

$$v(t) = v_0 - \mu_s g \Delta t \tag{4.19}$$

$$\alpha = \frac{f}{I_b} = \frac{5\mu_s m_b g r}{2m_b r^2} = \frac{5\mu_s g}{2r} \tag{4.20}$$

$$\omega(t) = \omega_0 + \alpha t = \omega_0 + \frac{5\mu_s g}{2r} t \tag{4.21}$$

, where $I_b = \frac{2m_b r^2}{5}$ is the moment of inertia of the spherical ball with respect to the rotation axis, and $\mu_s$ is the sliding friction coefficient. Equation 4.21 is valid until $v(t) = \omega(t)r$.

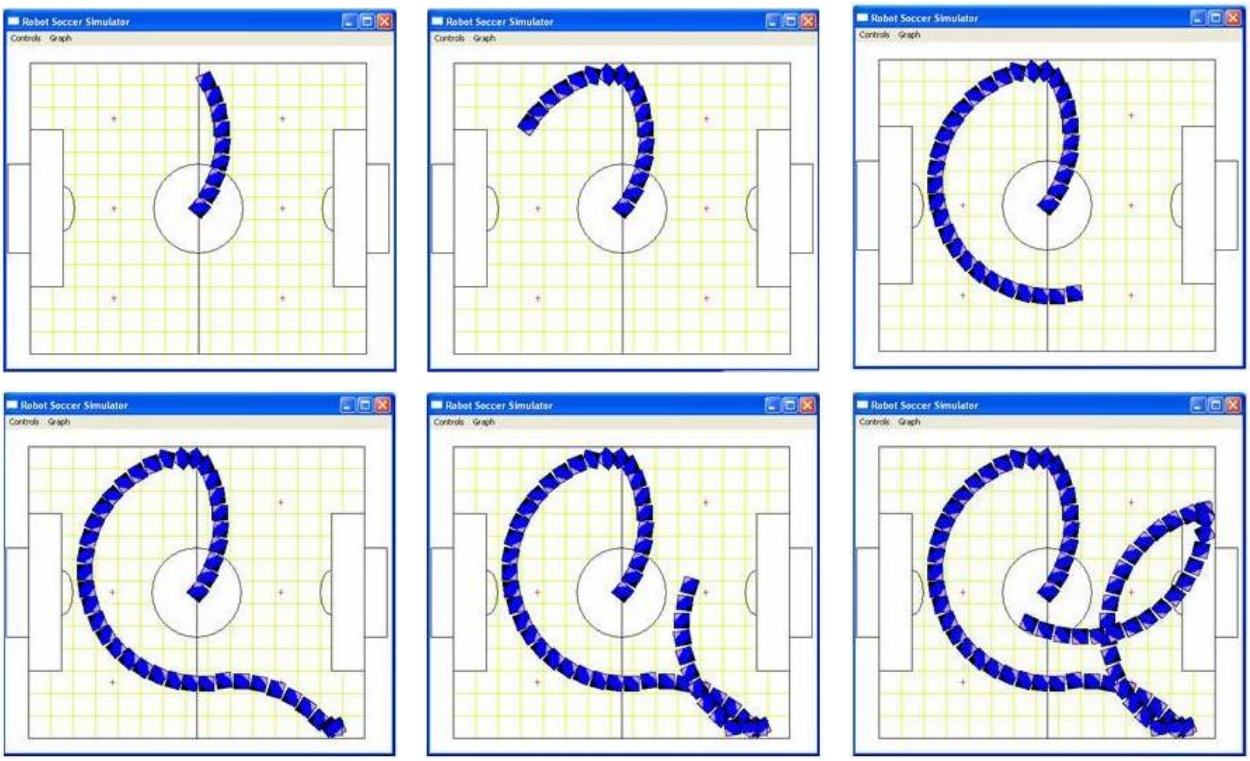


Figure 4.17: In this simulation, the future robot positions after possible collisions with walls are predicted from current tracked positions.

**Simulators in autonomous robotic systems**

The simulator primarily acts as a predictor of states of the playground. In a robot soccer system, the computer has absolute control over the wheel velocities of the home robots so

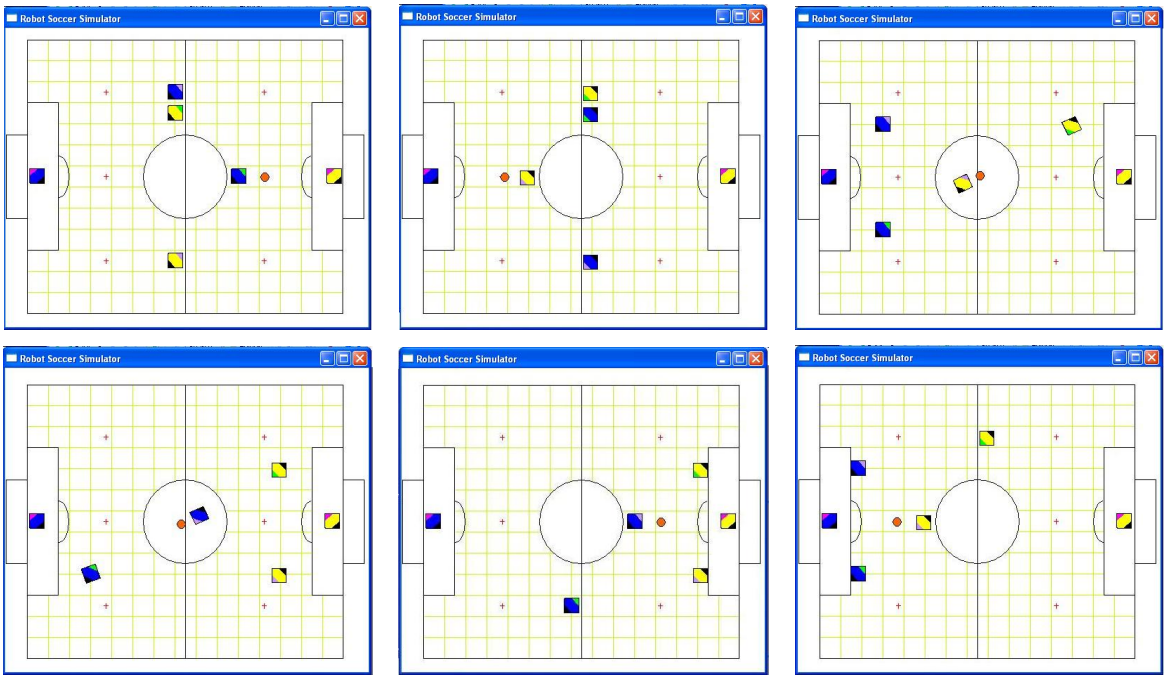


Figure 4.18: The simulator as a replacement for the real world during development of control algorithms offline. (From top to bottom, clockwise) Simulation of penalty kicks (A, B), kick offs (C, D) and free kicks (E, F).

it is possible to know where the robots are moving. However, the movement of the ball is erratic and is subject to collisions, occasional slipping and rolling. Thus, the ability to anticipate the ball's position and orientation after $n$ seconds for a given $n$ is beneficial for the development of control algorithms. Several algorithms in the area of robotic control proposed employ the idea of prediction [41] [42] [43] [44]. The Aneka framework allows the predictor to be included as a linked system without adding further complexities to the autonomous system.

The simulator can also function as a replacement for the physical world when developing control algorithms. When a satisfactory algorithm has been built and tested virtually, it can be transferred to the robot soccer system.

Yet another use for a simulator in an autonomous control software framework is as a platform for evolving control strategies. Evolutionary algorithms (EA) have long been used to evolve control algorithms (e.g., [45]). [46] discusses an approach for planning

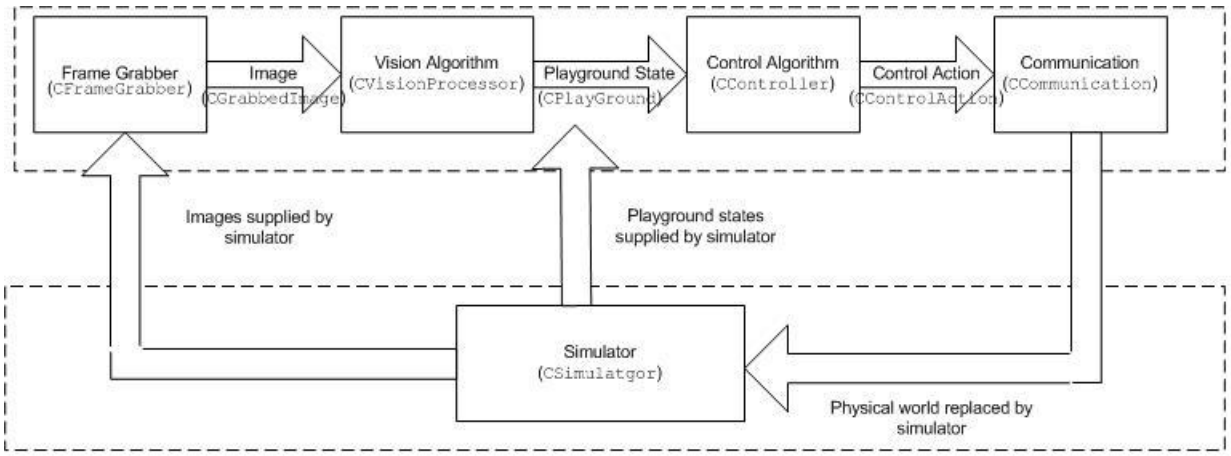


Figure 4.19: A simulator when implemented as a linked system in Aneka can be used to artificially create images to be processed by the autonomous controller or directly generate the playground states input to the system, facilitating offline development of vision and control algorithms.

collision-free paths for two robots sharing the same workspace based on a new kind of GA known as Cooperative Genetic Algorithms (CGA). Each robot is associated to one population of the CGA and each string of a population represents a complete path for its robots. The main idea is for both the populations of the CGA to interact in order to find a collision-free path.

Fuzzy logic is employed in [47] and [48]. Bonarini [47] proposes the Evolutionary Learning of Fuzzy Rules (ELF). This is a system that is able to evolve a population of fuzzy rules to obtain a Fuzzy Logic Controller. In ELF, the pool of fuzzy rules is divided into several sub-populations, whose members share the same values for the past variables. Within the sub-population, a rule that matches the current state is randomly selected. All the sub populations then cooperate to produce the control action.

In view of the many different methods available, it is important to have standards for comparing the efficiencies of the different strategies. In some of the articles, frameworks have been proposed to allow newly developed algorithms to be compared [49] [50].

In addition, [48], [51] and [52] describe self-organization as an important feature of

evolutionary learning, allowing an autonomous agent to be "genuinely intelligent" by having it interact with the environment without supervised instructions. However, the main problem of EA [53] is that the robots must play thousands of soccer games to obtain an optimum algorithm. The robots' batteries probably would not last more than a few hours, thus time-consuming recharging periods are needed. Human interference is also unavoidable during the course of each game and the robots may need repairs often. As a result, a control algorithm evolved in the real environment probably need months before an acceptable product is obtained. This is a major reason why evolving a control algorithm in the virtual environment is an attractive solution.

Recently, Zagal [54] proposed a new approach to evolutionary robotics which combines learning from reality and simulators into a single framework. The robot learns by alternating virtual and real experiences, while simultaneously, the simulator learns from the robot real behavior execution. While the robot interacts with the real world, the simulator converges to reality allowing for fast and representative evolutionary learning. The theory was carried out in experiments [55] and optimistic results were obtained. Evolution in simulation was also carried out successfully in [57] [58] [59].

Another major advantage of using simulators in designing robotic systems that is that in applications where actual construction of robots is prohibitively expensive due to the sheer scale of the problem, simulators can help to design major aspects of the control scheme inexpensively. Such a simulation environment, in which controllers for Braitenberg vehicles can be evolved to exhibit some interesting behaviours, was implemented as part of the author's thesis work. The simulation environment was implemented as a standalone research platform in C++ that can also be run from an Aneka-based context.

In the book "Vehicles: Experiments in Synthetic Psychology" [60], Valentino Braitenberg described thought experiments in which "vehicles" having very simple internal structure exhibit complex intelligent behaviours. These vehicles, put together, have self-

organizing abilities and evolutionary behaviors responding to their experiences and environment.

Braitenberg vehicles are interesting because they are essentially reactive systems (as opposed to symbolic deliberative systems), i.e., they simply react to their external stimuli in certain pre-determined ways, and yet exhibit interesting levels of intelligence. Very simple organisms are reactive systems, and since we have evolved from them, there is a reason to suspect that our intelligence could be reactive in nature.

The Braitenberg vehicle environment simulations show how primitive behaviours can be learned and can be potentially stitched together to form more sophisticated control algorithms. The simulator presents to the vehicles two types of general objects: obstacles and resources. The vehicles use sensors attached on their bodies to sense a potential field generated by each object that falls exponentially with increasing distance from the object. The simulator allows the objects to have two grid schemes (Figure 4.22): a full-body grid scheme, in which the entire area of the obstacle is significant to the generation of potential field, and a perimeter grid scheme, in which only the perimeter is significant. The potential field generated by the objects are sensed, amplified and added using weights attached to the input channels, the output of which are then input to the vehicle's wheels as actuating signals. Thus, intelligence in a simulated Braitenberg vehicle resides in the weights connecting the sensors to wheels.

Figures 4.23, 4.24 and 4.25 show interesting behaviours that autonomous teams in the simulation environment were designed to execute. In a future development, the simulator can be used to automatically evolve the weighted networks.

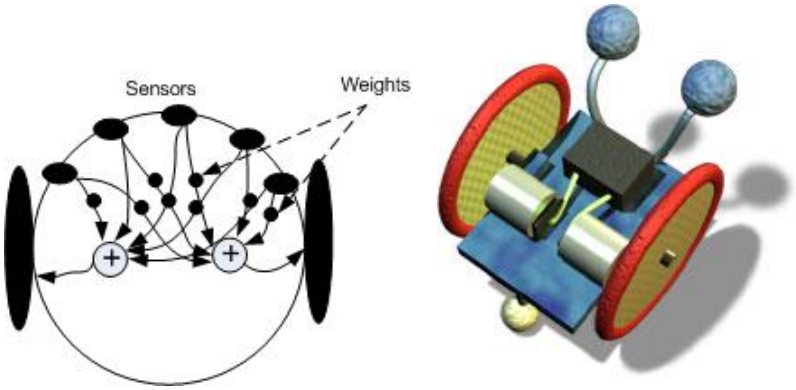


Figure 4.20: Braitenberg vehicles with internally connected neural networks.

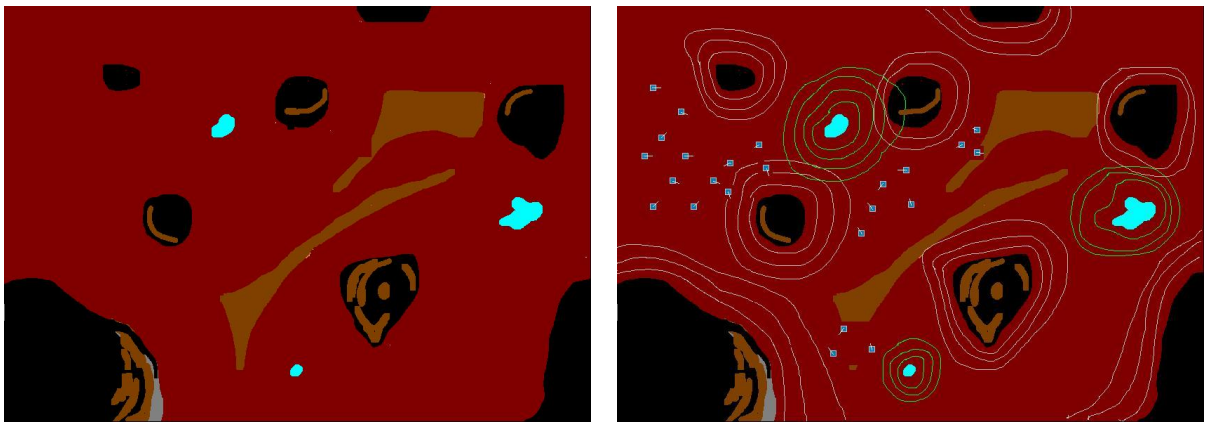

Figure 4.21: Empty terrain, and terrain with potential fields drawn.

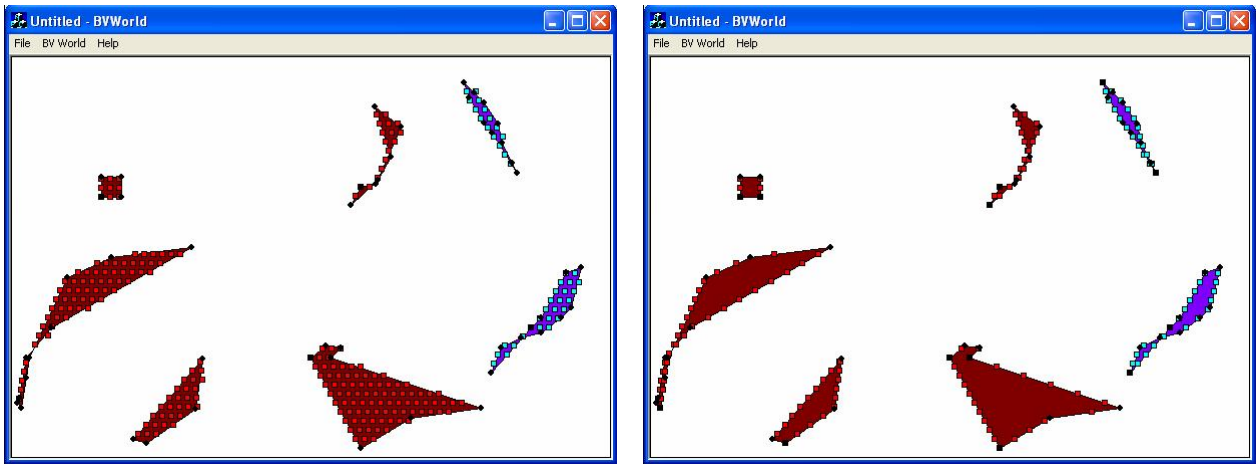


Figure 4.22: Full-body grid and perimeter grid schemes for obstacles.

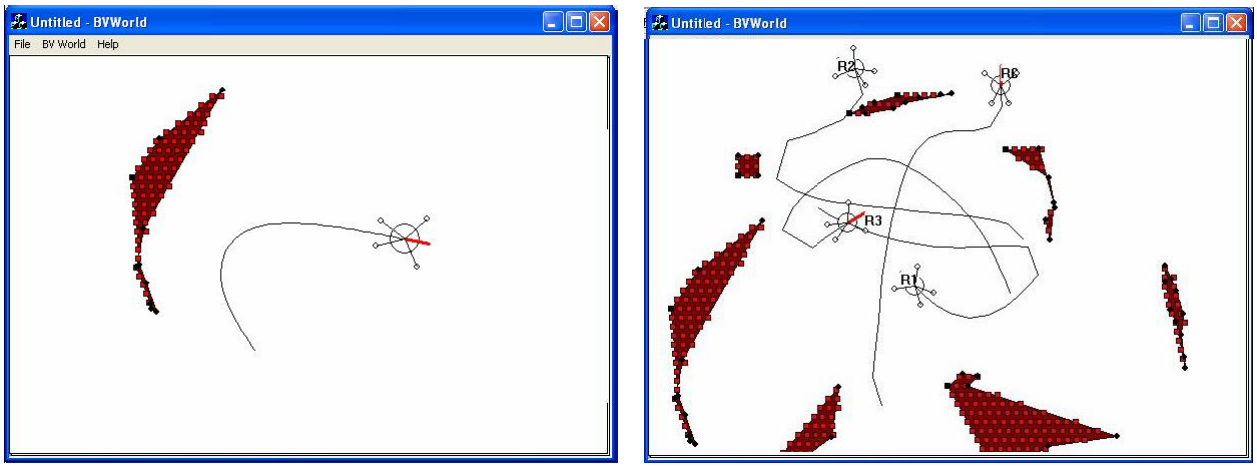


Figure 4.23: Braitenberg vehicle with obstacle avoidance behaviour, alone and in groups.

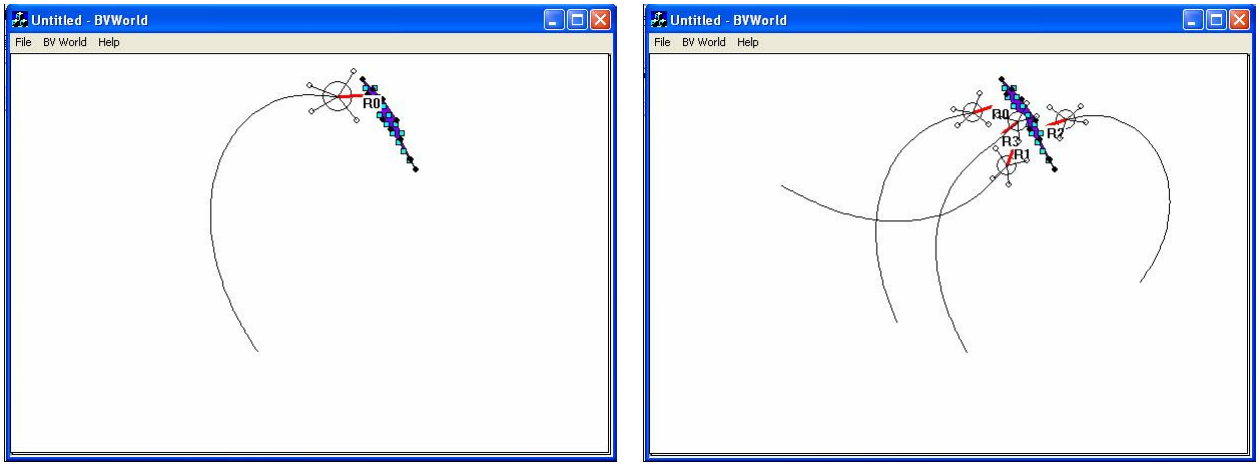


Figure 4.24: Braitenberg vehicle resource fetching, alone and in groups.

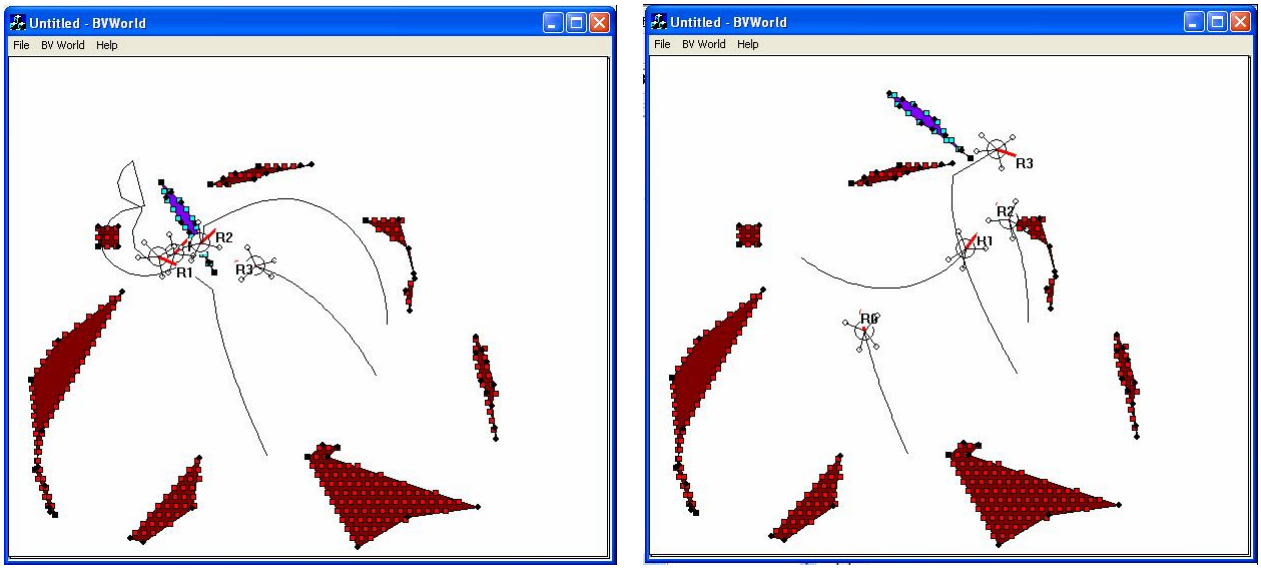


Figure 4.25: Combined behaviours of vehicles.

## 4.2 Communication

The last major linked system in the domain specific inteface layer for a robot soccer systems is the communication module.

The role of the communication module in a robot soccer system is to transmit the control actions generated by the controller to the robots on the field. In the reference implementation, the low level aspects of transmitting signals were handled by a pre-designed external RF transmitter communicating with the PC through an RS-232 (serial port) interface. The reference implementation thus provides only a minimum implementation of the communication module as the time constraints did not permit creation of a more elaborate communication scheme, and this section is present primarily for the sake of completeness.

Implementation of the communication module within the Aneka control software framework is shown in figure 4.26. The module (class `CCommunication` in the reference implementation) is defined as a linked system that accepts a control action signal as input, and transmits the control actions to the real-world robots.

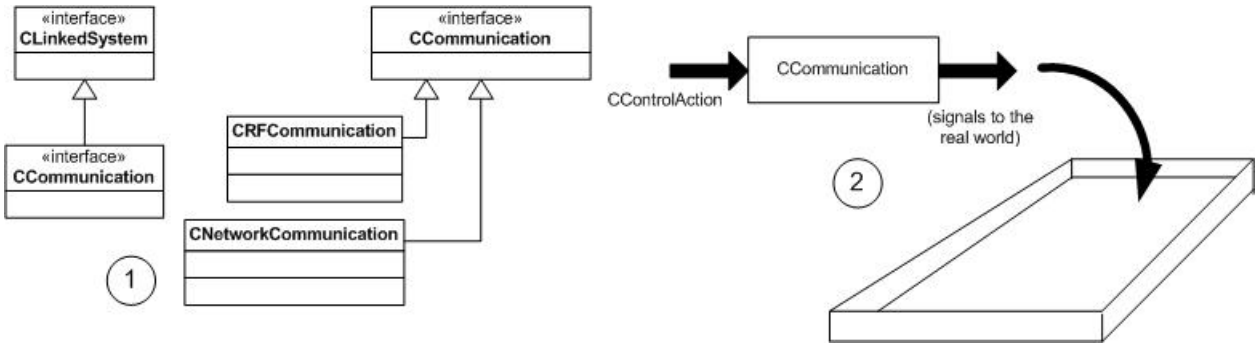


Figure 4.26: Implementation of communication module within the Aneka framework.

The interface presented to controller by the communication module hides any specific details that deal with the method of delivery of control actions to the robots. For example, the communication module could relay the velocity information over a network,

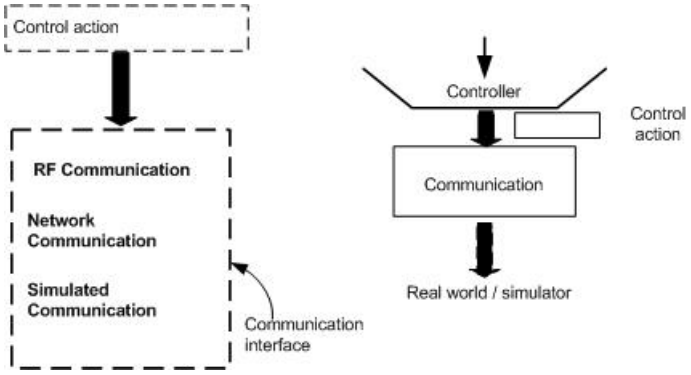


Figure 4.27: Communication module presents a standard interface to the controller through a domain specific interface of type Communication.

remotely-controlling the robots. If a simulator is used instead of the real world, the communication module would be relaying information to the simulator itself. The reference implementation provides on the RF communication and simulated communication (which is a part of the Aneka simulator), and more mechanisms could be added as needed.

# Chapter 5

# Conclusion

Implementation of autonomous robots require a variety of approaches dealing with parts of the overall system like machine perception, control and communication. While each of these individual areas, such as machine vision, multi-sensor fusion and path planning in mobile robots are given due consideration by the research community, the issue of how the various subsystems can be brought together in a streamlined and cohesive software architecture very rarely forms an independent subject of investigation. Yet, in an environment where software is the defacto medium for implementing intelligent controllers, the importance of the subject should not be underestimated.

This thesis discussed Aneka, an extensible control software framework for autonomous robotic systems that can be used to implement controllers for a wide variety of robotic domains using a clean architecture. A reference implementation of typical modules for a multi-robot soccer system is provided to demonstrate how the framework can be used in practice. Along with serving as a proof of concept of the proposed software framework, the reference implementation in itself forms a solid platform for future research in fields such as machine vision and multi-robot coordination.

The question of whether any single control software framework can be best suited for implementing autonomous systems may be rightfully asked. While creating a software

framework, we often need to consider what characteristics of the end product we are ultimately interested in. Some of these characteristics include,

1. Safety and robustness requirements of the domains in which the software will be used.
2. Speed with which the framework can be implemented in a problem area.
3. Ease of use of the framework by the implementors of the software.
4. How well and elegantly the framework models the domain of interest.

The prime goal of Aneka was to bring in "system thinking" right into the initial stages of creating software-based autonomous controllers for robots. While in the minimality of the concepts used for this purpose Aneka can be said to be optimal, it is nevertheless a valid question whether it may be possible to design frameworks where other criteria - such as correctness, thoroughness, speed of development, etc. - relevant to software based systems can be optimised.

## 5.1 Aneka in other domains

Presentation of a generic control software framework such as Aneka cannot be complete unless a straightforward implementation can be demonstrated not just the chosen robotic domain, but in vastly different domains as well. This section discusses two instances where fundamental interfaces and layering are used to implement SMPA based autonomous robotic systems.

Figure 5.1 shows the schematic of a hexapod robot that senses its environment through two antennae, a low resolution light sensor array, and force sensors attached to the legs of the robot. The values from these sensors are regularly scanned by the control executive, packaged into signal objects, and relayed to an internal modelling module. The modelling module fuses the miscellaneous sensory data, and produces an environment model (`CEnvModel`) that contains information on the state of the robot's environment. The

control executive then passes the environment model to the robot's high level controller, which decides on a plan of action based on them.

All modules of the robots can be extended from the fundamental interface of the linked system (`CLinkedSystem`), and they communicate with each other through specialised objects that implement the signal interface (`CSignal`). At the domain specific level of the hexapod shown in Figure 5.1, application of the Aneka framework might give the robot the following linked systems:

1. `CTouchSensor`
2. `CLightSensorArray`
3. `CForceSensor`
4. `CModeller`
5. `CHLController` (High level path planning)
6. `CLLController` (Low level leg movements, control and coordination)

The signals interfaces involved in the controller are:

1. `CBooleanSignal`
2. `CBinaryArray`
3. `CForceSignal`
4. `CEnvModel`
5. `CLegMovement`

The linked systems produce and consume signals in the context of a real-time control executive, `CControlExecutive`. Each of the domain-specific interfaces themselves could have specific implementations that ultimately realise their algorithms. While sensing and action modules in a hexapod may not normally call for much complications, modelling modules and high level planning modules could be implemented using an extremely wide

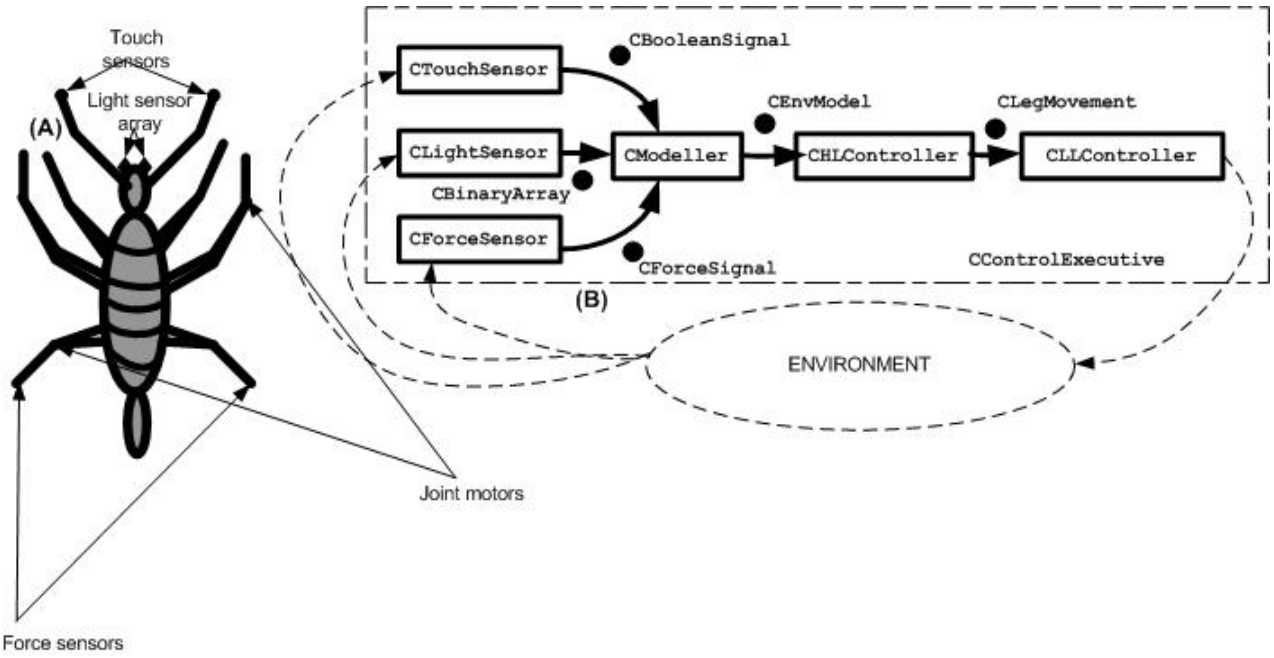


Figure 5.1: Controller for a hexapod implemented using constructs provided in Aneka. All modules and communications are modelled using linked systems and signals that are coordinated by a control executive, and implement the fundamental interfaces of linked system and signal. The domain specific interfaces, such as high level and low level controllers, specify the functionality that must be realised by the ultimate implementations.

variety of algorithms. Since the overall controller itself is implemented under a common, layered framework, the individual modules - such as the control or machine vision algorithm - can be transparently replaced without affecting other subsystems.

The hexapod presented above could be made as complex as the technology available at hand allows. At times, however, robots may need to work in extremely resource constrained environments. Figure 5.2 shows the schematic of such a simple robot which, despite its simple architecture, is nevertheless capable of avoiding obstacles in its way and reaching a predetermined destination. The disc-shaped robot consists of bump sensors that are arranged symmetrically along its lateral surface. The bump sensors are connected to an internal controller that drives inputs to two wheel motors. While it is possible to have an analog circuit directly hardwire the sensor inputs to the wheel motors, in practice the robot might need to exhibit relatively complex behaviours, such as not attempting

to move in a direction where it had "sensed" an obstacle before, effectively creating a map-like structure of its environment.

The simplicity of the robot's architecture may imply that the software interacts at a very close level with sensors and actuators. The implementation of the autonomous system would be simple in this case since effectively, we need to design only two linked systems (the bump sensor and the controller), and two signals (the bump and the wheel velocities). The linked systems, following the naming convention of the reference implementation, are,

1. `CBumpSensor`
2. `CVelController`

, and the signals objects in the robot are

1. `CBump`
2. `CVelocity`

Even though more complex SMPA robots might call for heavy object oriented modelling in software, in the case of the mobile robot, the signals could be extremely low bandwidth objects like scalar real or boolean values. By separating and studying the behaviour of the robot's velocity controller as a linked system and defining clearly how it interfaces with the other modules, we can investigate the effects of a large number of control strategies while still maintaining a coherence of design in the controller. Linked systems, signals and control executives also provide a strong conceptual foundation based on which the system can be designed.

## 5.2 Future directions

There are several aspects of control software frameworks in autonomous robotic systems that could be studied with Aneka and its reference implementation as starting points.

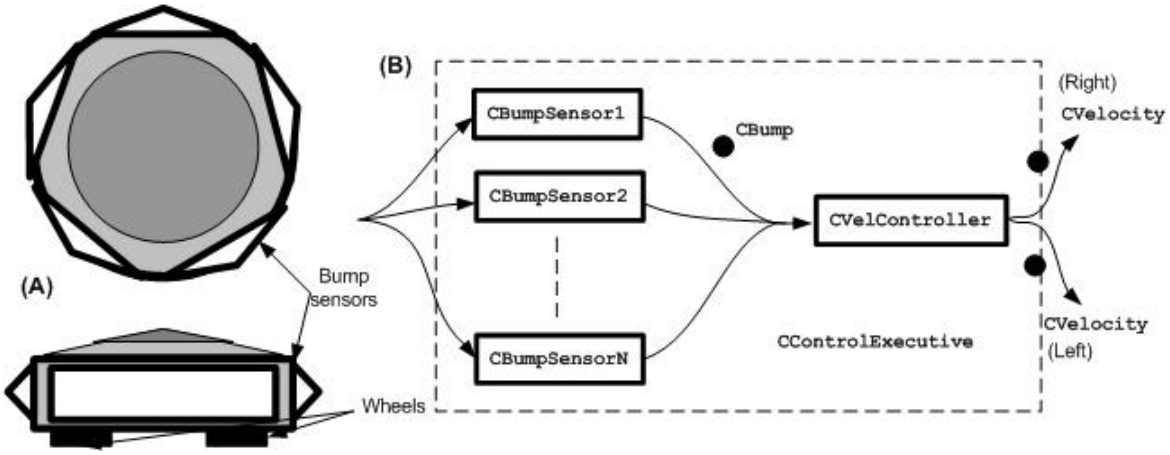


Figure 5.2: A simple mobile robot controller could be implemented with a homogeneous array of sensors directly connected to the controller. Explicit modelling or communication modules are not necessary, allowing the controller to be implemented using just two linked systems and two signals.

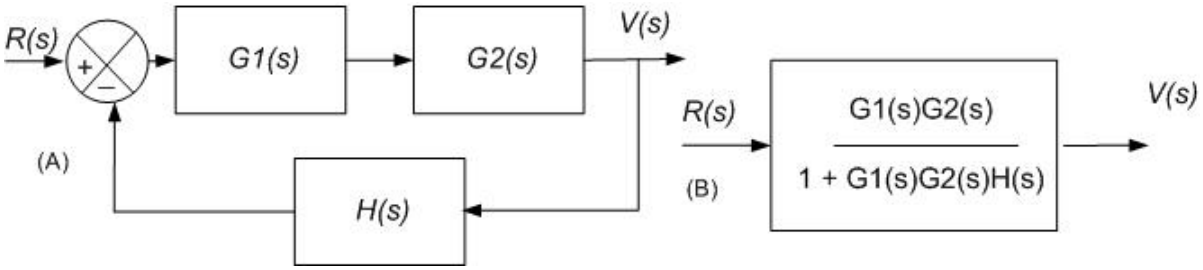


Figure 5.3: Deriving the overall system transfer function from the transfer functions of the composing sub-systems in classical control.

A major area of concern in composing sub-systems with varying characteristics into a composite system is how the overall dynamics is affected by the dynamics of the individual systems. In classical control, this problem is fairly easily dealt with in the case when the sub-systems are linear and time invariant (Figure 5.3).

Despite the importance of being able to determine overall system behaviour from that of individual components, however, the resolution is not very straightforward in the case of autonomous systems such as robots. A major difficulty is in determining what exactly do we mean by "intelligence" in a machine, and how intelligence of a composite system changes when several intelligent subsystems are stringed together.

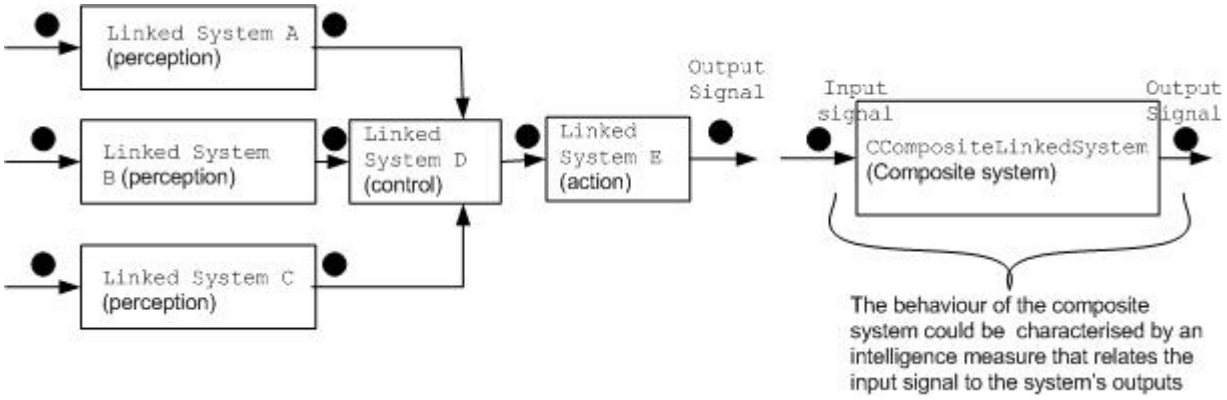


Figure 5.4: Characterising the overall behaviour of composite linked systems in Aneka. Intelligence characteristics of sub-systems influence the overall intelligence of the autonomous robot.

The notion of "intelligence" in artificial systems and the requirements for such intelligent systems are discussed in detail in [1]. In a control system, at a minimum level, "intelligence" implies the ability to make decisions based on a given set of inputs in order to attain a specified set of goals. One way to characterise the dynamics of such systems might be to formulate metrics based on which an "intelligence quotient" of the system could be derived [2] [3] [4] and measure the quality of the system's solution along with the time taken to reach it. This could be considered a close symbolic analog of response of linear systems to unit impulse inputs (Figure 5.4).

A measurement scheme for machine intelligence might proceed just like intelligence tests in humans, and have the following steps:

1. Create an problem scenario that the system must solve.

2. Specify characteristics of the correct solution.

3. Measure the quality of the solution against the time taken to reach it.

There are also other system and control theoretic concepts that are relevant in the domain of autonomous robotic systems. For example, Passino et. al. [12] propose analogies of classical concepts such as controllability, observability and stability in the context of artificial intelligent systems, that can be extended to autonomous robots functioning in

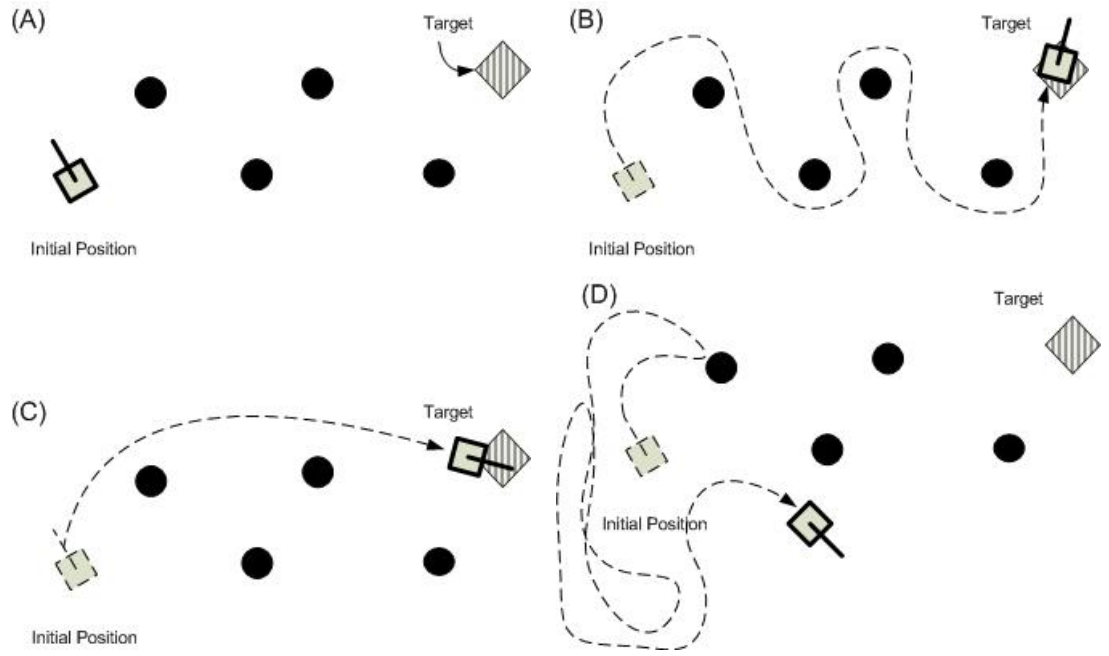


Figure 5.5: In a scheme for measuring intelligence of autonomous systems, the system could be asked to solve a number of problems, and the solutions evaluated for accuracy and speed. (A) The robot is required to find a path from a starting point to a target, (B) The robot produces an overly complex solution. (C) The problem is solved optimally. (D) The robot fails at the task due to malfunctioning communication or vision algorithm. Though the path planning algorithm itself may be sound, overall intelligence suffers when systems are composed together.

the physical world.

Controllability refers to the ability of a system's inputs to change the state of the system. In classical and modern control theory, a system is said to be completely controllable at a time $i$ if there exists a finite time $j > i$ such that for any state $x_i$ and any state $x$, there exists an input sequence $u_i, ..., u_j$ that will transfer the state $x_i$ to the state $x$ at time $j$ [12]. A system is weakly controllable at time $i$ if there exists a finite time $j > i$ such that for any states $x_i$ and $x$, both in the set of the system's controllable states, there exists an input sequence $u_i, ..., u_j$ that will transfer the system from state $x_i$ to $x$. Extending the concept of controllability to autonomous systems, we might say that a problem is controllable if, given an initial state of the plant and a final target state, the autonomous controller can produce a series of actions that will solve the problem.

Observability in control theory refers to ability to determine the state of the system from the inputs, outputs, and model of the system. A system is said to be completely observable at time $i$ is there exists a finite time $j > i$, such that for any state $x_i$ of the problem representation, the sequence of inputs, and the corresponding sequence of outputs over the time interval $[i, j]$ uniquely determine the state $x_j$. In intelligent, autonomous systems, observability could imply our ability to design "situation assessors" that can determine the state of the system in question from a series of past inputs and the system's model.

The notion of stability of dynamical systems can be similarly extended to the domain of autonomous robots. In control, a system is said to be internally stable when with no inputs, if the system begins in some particular set of states and the state is perturbed, it will always return to its initial set of states. An intelligent system can be said to be input-output stable if for all good input sequences to the system, the outputs are also good, where the goodness of an output is determined by whether the state reached by the system is desirable or not.

Also, the reference implementation of Aneka for the multi-robot soccer system has default implementations of several modules that, in themselves, deal with broad areas such as machine vision and robot path planning. Their inclusion in a unified but flexible framework could further facilitate the reference implementation being used as a standalone platform for research in each of these specialised fields.

# Bibliography


[1] **J. S. Albus.** "Outline for Theory of Intelligence". IEEE Trans. System, man and Cybernetics, Vol. 21, No.3, pp473-509, 1991.

[2] **J.M. Evans, E.R. Messina.** "Performance Metrics for Intelligent Systems". Proc. of Performance Metrics for Intelligent Systems Workshop. 2000 PerMIS Workshop, 2000.

[3] **L.A. Zadeh.** "The Search for Metrics of Intelligence - A Critical View". Proc. of Performance Metrics for Intelligent Systems Workshop. 2000 PerMIS Workshop, 2000.

[4] **Robert Finkelstein.** "A method for evaluating the "IQ" of Intelligent Systems". Proc. of Performance Metrics for Intelligent Systems Workshop. 2000 PerMIS Workshop.

[5] **Jong-Hwan Kim, Hyun-Sik Shim, Myung-Jin Jung, Heung-Soo Kim, and Prahlad Vadakkepat.** "Cooperative multi-agent robotic systems: From the robot-soccer perspective". Invited paper, Proc. of the Second Micro-Robot World Cup Soccer Tournament (MiroSot'97), KAIST, Taejon, Korea, pp. 3-14, June 1997.

[6] **J.R. Quinlan.** "Induction of Decision Trees". Machine Learning, vol. 1, pp 81-106, 1986.

[7] **Donald A. Waterman.** The guide to expert systems. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 1985.

[8] **Zadeh, L.A.** "Fuzzy Sets". Information and Control, 8, 338-353. 1965.

[9] **Dubois and H. Prade.** "An Introduction to Possibilistic and Fuzzy Logics", in G. Shafer and J. Pearl (eds.), Readings in Uncertain Reasoning, Morgan Kaufmann Publishers, 1990.

[10] **E. Franco Scarselli, Ah Chung Tsoi.** "Universal Approximation Using Feedforward Neural Networks: A Survey of Some Existing Methods, and Some New Results". Neural Networks 11(1): 15-37, 1998.

[11] **S. Haykin.** Nueral networks, a comprehensive foundation. Macmillan, 1994.

[12] **Kevin M. Passino and Panos J. Anstalkis.** "A system and control theoretic perspective on artificial intelligence planning systems". Applied Artificial Intelligence. Vol. 3, No. 1, pp. 1 - 32, 1989.

[13] **Kevin M. Passino.** "Intelligent Control: An Overview of Techniques", in T. Samad, Ed., Perspectives in Control Engineering: Technologies, Applications, and New Directions, pp. 104-133, IEEE Press, NY, 2001.

[14] **Rodney. A. Brooks.** "A Robust Layered Control System for a Mobile Robot", IEEE Journal of Robotics and Automation, Vol. 2, No. 1, pp. 14-23, March 1986.

[15] **Thomas Deschamps and Laurent D. Cohen.** "Grouping connected components using minimal path techniques", Computer Vision and Pattern Recognition, 2001 (CVPR 2001). Proceedings of the 2001 IEEE Computer Society Conference, 2001.

[16] **K.S. Fu and J.K. Mui.** "A survey on image segmentation", Pattern Recognition, vol. 13, pp. 3-16, 1981.

[17] **N. R. Pal and S. K. Pal.** "A review on image segmentation techniques", Pattern Recognition, Vol. 26, No. 9, pp. 1277 – 1294, 1993.

[18] **Guy. B. Coleman and Harry C. Andrews.** "Image segmentation by clustering", Proceedings of the IEEE, 67(5):773–785, May 1979.

[19] **C.S Hong, S.M Chun, J.S Lee, K.S Hong.** "A Vision-Guided Object Tracking And Prediction Algorithm for Soccer Robots", Proc. IEEE Int. Conf. on Robotics and Automation, Alberquerque, New Mexico, pp 346-351, Apr 1997.

[20] **Junichi Akita.** "Real-time Color Detection System using Custom LSI for High Speed Machine Vision", Kanazawa University, Japan, 1998.

[21] **Thorsten Schmitt, Robert Hanek, Michael Beetz, Sebastian Buck, Bern Radig.** "Cooperative Probabilistic State Estimation for Vision-based Autonomous Mobile Robots", in IEEE Transaction on Robotics And Automation 18(5): 670-684, October 2002.

[22] **Bo Li, Edward smith, Huosheng Hu, Libor Spacek.** "A Real Time Visual Tracking System in the Robot Soccer Domain", in Proceedings of EUREL Robotics-2000, Salford, England. Apr 2000

[23] **James Brusey and Lin Padgham.** "Techniques for Obtaining Robust, Real-time, Colour-based Vision for Robotics", Royal Melbourne Institute of Technology, CSIRO Mathematical and Information Sciences. 1998.

[24] **Hutchinson, S., Hager, G., Corke, P.** "A tutorial on visual servo control", IEEE Trans. Robotics and Automation, 12(5). pp. 651-670., 1996.

[25] **Pfeifer, R., Scheier, C.** Understanding Intelligence. The MIT press. Cambridge, USA., 1999.

[26] **Agre, P.E. Chapman, D.** "Pengi: An implementation of a theory of Activity", In 'Proceedings, AAAI-87'. Seattle. USA. pp. 268-272., 1987.

[27] **Connel, J.H.** Minimalist Mobile Robotics: A Colony Architecture for an Artificial Creature, Academic Press., 1991.

[28] **Giralt, G., Chatila, R. Vaisset, M.** Minimalist Mobile Robotics: A Colony Architecture for an Artificial Creature, Academic Press., 1991.

[29] **Laird, J.E. Rosenbloom, P.S.** "Integrating, Execution, Planning, and Learning in Soar for External Environments". In 'Proceedings, AAAI-90'. pp. 1022-1029., 1990.

[30] **Rodney A. Brooks.** "An integrated navigation and motion control system for autonomous multisensory robots". In Brady, M., Paul, R. (Eds.). 'First International Symposium on Robotics Research'. MIT Press. Cambridge, USA., 1983.

[31] **Rodney A. Brooks.** "Intelligence without representation". Artificial Intelligence 47. pp. 139-159., 1991

[32] **Newell, A., and Simon, H.A.** "Computer Science as empirical enquiry: Symbols and Search". Communications of the Association for Computing Machinery, 19(3). pp. 113-126., 1976.

[33] **Rodney A. Brooks.** In Maes, P. (Ed).'Designing Autonomous Agents: Theory and Practice from Biology to Engineering and Back'. MIT Press., 1990

[34] **R. Pfeifer, F. Iida and J. C. Bongard.** "New Robotics: Design Principles for Intelligent Systems". Artificial Life. Vol. 11, Issues 1-2, pp. 99 - 120 - Winter-Spring, 2005.

[35] **M. Veloso, P. Stone, K. Han, and S. Achim.** "The CMUnited-97 Small Robot Team", In RoboCup-97: Robot Soccer World Cup I, pages 243-256, 1998.

[36] **D. Eberly.** "Dynamic Collision Detection Using Oriented Bounding Boxes", Magic Software, Available from http://www.magic-software.com.

[37] **D. Eberly.** "Intersection of Objects with Linear and Angular Velocities using Oriented Bounding Boxes", Magic Software, Available from http://www.magic-software.com.

[38] **B. K. Quek.** "Towards an Intelligent Robot Soccer System", Bachelor Thesis, National University of Singapore, 2003.

[39] **P. R. Kedrowski.** "Development and Implementation of a Self-Building Global Map for Autonomous Navigation", Master Thesis, Virginia Polytechnic Institute and State University, 2001.

[40] **H. P. Huang, C. C. Liang and C. W. Lin.** "Construction and Soccer Dynamics Analysis for an Integrated Multi-agent Soccer Robot System", Proceedings of the National Science Council Vol. 25, No.2, pp. 84-93, 2001.

[41] **A. F. Foka and P. E. Trahanias.** "Predictive Control of Robot Velocity to Avoid Obstacles in Dynamic Environments", Proceeding of IEEE/RSJ International Conference on Intelligent Robots and Systems, 2003.

[42] **A. F. Foka and P. E. Trahanias.** "Predictive Autonomous Robot Navigation", Proceeding of IEEE/RSJ International Conference on Intelligent Robots and Systems, 2002.

[43] **S. Behnke, A. Egorova, A. Gloye, R. Rojas, M. Simon.** "Predicting away Robot Control Latency", Proceedings of 7th RoboCup Internatioal Symposium, 2003.

[44] **B. Browning, G. Wyeth and A. Tews.** "A Navigation System for Robot Soccer", Proceedings of the Australian Conference on Robotics and Automation, pp. 96 - 101, 1999.

[45] **S. Nolfi, D. Floreano, O. Miglino and F. Mondada.** "How to Evolve Autonomous Robots: Different Approaches in Evolutionary Robotics", Proceedings of the Fourth International Workshop on the Synthesis and Simulation of Living Systems, (Artificial Life IV), pp. 190 - 197, 1994.

[46] **V. de la Cueva and F. Ramos.** "Cooperative Genetic Algorithms: A New Approach to Solve the Path Planning Problem for Cooperative Robotic Manipulators sharing the same Work Space", Proceedings of the 1998 IEEE/RSJ International Conference on Intelligent Robots and Systems, 1998.

[47] **A. Bonarini.** "Evolutionary Learning of Fuzzy Rules", Competition and Cooperation, Fuzzy Modelling: Paradigms and Practice, pp. 265 - 283, 1996.

[48] **T. Pal, N. R. Pal.** "SOGARG: A Self-Organized Genetic Algorithm-Based Rule Generation Scheme for Fuzzy Controllers", IEEE Transactions on Evolutionary Computation Vol. 7 No. 4, 2003.

[49] **D. Floreano and J. Urzelai.** "Evolutionary Robots with On-line Self-organization and Behavioral Fitness", Neural Networks Vol. 13, pp. 431 - 443, 2000.

[50] **L. P. Kaelbling.** "Foundations of Learning in Autonomous Agents", Robotics and Autonomous Systems Vol. 8, pp. 131 - 144, 1991.

[51] **S. Nolfi.** "Evolutionary Robotics: Exploiting the Full Power of Self-organization", Connection Science Vol. 10 No. 3 and 4, pp. 167 - 184, 1998.

[52] **P. F.M.J. Verschure, B. J.A. Krse and R. Pfeifer.** "Distributed Adaptive Control: The Self-organization of Structured Behavior", Robotics and Autonomous Systems Vol. 9, pp. 181 - 196, 1992.

[53] **M. Walker.** "Evolution of a Robotic Soccer Player", Research letters in the Information and Mathematical Sciences Vol. 3, 2002.

[54] **J. C. Zagal and J. Ruiz-del-Solar.** "Back to Reality: Crossing the Reality Gap in Evolutionary Robotics. Part I: Theory", 5th International Federation of Automatic Control (IFAC/EURON) Symposium on Intelligent Autonomous Vehicles , 2004.

[55] **J. C. Zagal, J. Ruiz-Del-Solar and P. Vallejos.** "Back To Reality: Crossing the reality Gap in Evolutionary Robotics. Part II: Experiments", 5th International Federation of Automatic Control (IFAC/EURON) Symposium on Intelligent Autonomous Vehicles, 2004.

[56] **O. Miglino, H. H. Lund and S. Nolfi.** "Evolving Mobile Robots in Simulated and Real Environments", Artificial Life Vol. 2 Issue 4, 1996.

[57] **N. Jakobi, P. Husbands and I. Harvey.** "Noise and the Reality Gap: The Use of Simulation in Evolutionary Robotics", Proceedings of the 3rd European Conference on Artificial Life (ECAL95), pp 704 - 720, 1995.

[58] **L. Meeden.** "Bridging the Gap Between Robot Simulators and Reality", Proceedings of the Third Annual Genetic Programming Conference, pp. 824 - 831, 1998.

[59] **D. F. Hougen.** Learning with Holes: Pitfalls with Simulation in Learning Robot Control, Machine Learning Technologies for Autonomous Space Applications Workshop of the International Conference on Machine Learning, 2003.

[60] **Braitenberg, B.** Vehicles: Experiments in synthetic psychology. Weidenfield and Nicolson. London. 1984.