
University of Crete
Department of Computer Science

FO.R.T.H.
Institute of Computer Science

# Development of a robotic platform: hardware and software for control, perception and movement

(MSc. Thesis)

Christos Papachristou

Heraklion
February 2005

Department of Computer Science
University of Crete

# Development of a robotic platform: hardware and software for control, perception and movement

Submitted to the
Department of Computer Science
in partial fulfillment of the requirements for the degree of
Master of Science

28th February, 2005



Author

Christos Papachristou
*Department of Computer Science*

**Committee**

Supervisor

Panos Trahanias
*Professor*

Member

Apostolos Traganitis
*Professor*

Member

Evangelos Markatos
*Associate Professor*

**Accepted by**

Chairman of the Graduate Studies Committee

Dimitris Plexousakis
*Associate Professor*

Heraklion, February 2005

# Acknowledgements

This thesis...
"...though it cannot hope to be useful or informative on all matters, it does at least make the reassuring claim, that where it is inaccurate it is at least definitely inaccurate."
THE HITCHHIKER'S GUIDE TO THE GALAXY

And it would not have been possible without the help and support of many people. Firstly, I would like to thank my supervisor Prof. Panos Trahanias for his support and the confidence he has shown in me throughout my years in the institute. Also, the members of the committee, Prof. Apostolos Traganitis and Prof. Evangelos Markatos for their patience and comments. And in particular Prof. Stelios Orphanoudakis is greatly acknowledged for his support and for providing the idea for the implementation of a robotic soccer player.

Both the University of Crete and the Foundation for Research and Technology–Hellas (FORTH), where this thesis was conducted, have provided a wealth of academic stimuli as well as a pleasant and creative environment for which I'm heartily thankful. FORTH also provided financial support and the appropriate equipment which is gratefully acknowledged as well.

I would also like to thank my colleges at the Computational Vision and Robotics Lab for the friendly atmosphere and the useful discussions that have greatly influenced this work. Very special thanks go to Mr. Pantelis Georgiadis, whose profound knowledge of electronics has prevented many explosions, and who has helped me in the high–power electronics designs and proofread and provided comments for all others, but more importantly has been an extremely helpful and pleasant co-worker tolerating my untidiness. Many thanks also go to Dr Antonis Argyros for the many discussions, ideas and inputs in the field of machine vision.

Last but more important than any academic and financial help or support, this thesis was achieved thanks to my parents for reasons too numerous to state, to Zacharoula for being funny, to Maria for putting up, to Ioanna for encouraging and to Antonis and Maria for true friendship, comradeship and food. It is dedicated to all of them.

# Abstract

In this thesis we present the complete development and construction of a robotic platform. The primary goal was to provide a complete hardware and software bundle for researchers in the field of machine vision and autonomous mobile robots, taking into account their needs and good or bad experiences with other robotic platforms. Apart from that, one of the prerequisites was that the robot should meet the specifications of the middle sized league of the RoboCup robotic soccer competition, so as to serve as a testbed for cooperative algorithms and behaviours.

The RoboCup competition provides a well defined framework for the testing of multi-robot cooperation, agent-based behaviour algorithms, and perception through vision. Having in mind as the ultimate goal to construct a robotic platform that would eventually be able to serve as a prototype for a team of robots allowed to participate in the competition, the preliminary architectural guidelines for it were dictated by the RoboCup middle sized league player specifications. The only restrictions posed by these were robustness, dimensions and the ability to handle a ball. Apart from them, the robotic platform was designed to be usable as a general research tool.

The rest of the architectural guidelines were laid down after discussions with researchers in the field of machine vision and robotics and according to their recommendations and requirements as application writers. These guidelines were followed in all stages of the robot construction, which involved theoretic and practical work in three fields: mechanical, electrical/electronic and software engineering. These three were required to interact very closely and were developed together, taking into account and trying many alternative implementations, in order to achieve an optimum balance between ease of use, stability, efficiency and cost. Almost all functional modules and parts of the system were designed from scratch following as close as possible the rules of good engineering, and what is more important, they were also implemented. The result is a prototype robot that really works and has already been used by students and researchers to develop robotic applications, thus testing and proving both the design and the implementation decisions.

# Περίληψη

Στον τομέα της ρομποτικής υπάρχουν πειράματα, τα οποία δεν μπορούν να διεξαχθούν χωρίς πραγματικά ρομπότ. Παρόλο που υπάρχουν πολλά περιβάλλοντα προσομοίωσης, τα οποία επιτρέπουν γρήγορη και εύκολη αποτίμηση των προτεινόμενων αλγορίθμων και συμπεριφορών, υπάρχουν πάντα λιγότερο ή περισσότερο σημαντικές παράμετροι του πραγματικού κόσμου που δεν μπορούν ή δεν πρέπει να προσομοιωθούν. Ιδιαίτερα για τις συμπεριφορές που βασίζονται σε σύστημα πρακτόρων (agents), οι καθιερωμένες θεωρίες αναφέρουν ότι η νοημοσύνη μπορεί να αποκτηθεί μόνο μέσω ενεργητικής αλληλεπίδρασης με το περιβάλλον. Επίσης, όλοι οι τομείς της ρομποτικής απαιτούν πραγματικά ρομπότ ως το τελευταίο στάδιο δοκιμής τεχνικών και αλγορίθμων, κάτι που κάνει την ανάπτυξη ενσώματων ρομποτικών συστημάτων σημαντικό κομμάτι της έρευνας.

Σε αυτή την εργασία παρουσιάζουμε την πλήρη ανάπτυξη και κατασκευή μιας ρομποτικής πλατφόρμας. Ο κύριος στόχος ήταν να δημιουργηθεί ένα πακέτο υλικού και λογισμικού για χρήση από ερευνητές στους τομείς της μηχανικής όρασης και της ανάπτυξης αυτόνομων ρομπότ, λαμβάνοντας υπόψη τις ανάγκες τους και την καλή ή κακή εμπειρία τους με άλλες ρομποτικές πλατφόρμες. Εκτός αυτού, μια ακόμα προϋπόθεση ήταν να καλυφθούν οι προδιαγραφές του διεθνούς διαγωνισμού ρομποτικου ποδοσφαίρου RoboCup , ώστε μέσω αυτού η πλατφόρμα να χρησιμοποιηθεί ως εργαλείο για την έρευνα πάνω σε αλγόριθμους και συμπεριφορές συνεργαζόμενων αυτόνομων ρομπότ. Τα σχεδιαστικά κριτήρια τέθηκαν μετά από συζητήσεις με ερευνητές και φοιτητές στους τομείς της μηχανικής όρασης και της ρομποτικής και σύμφωνα με τις προτάσεις τους και τις απαιτήσεις τους για ένα εργαλείο ανάπτυξης. Αυτά τα κριτήρια ακολουθήθηκαν σε όλα τα στάδια της ανάπτυξης, η οποία περιελάμβανε θεωρητική και πρακτική εργασία σε τρεις τομείς: μηχανολογικό, ηλεκτρολογικό/ηλεκτρονικό και λογισμικού. Και οι τρεις απαιτείτο να αλληλεπιδρούν πολύ στενά και αναπτύχθηκαν μαζί, λαμβάνοντας υπόψη και δοκιμάζοντας διάφορες εναλλακτικές σχεδιαστικές επιλογές για να επιτευχθεί ένας όσο το δυνατόν καλύτερος συνδυασμός μεταξύ της ευκολίας χρήσης, της σταθερότητας, της αποτελεσματικότητας και του κόστους.

Η παρούσα εργασία περιγράφει την εξ αρχής ανάπτυξη και κατασκευή μιας ερευ-

νητικής ρομποτικής πλατφόρμας γενικής χρήσης. Η εμπειρία ερευνητών του εργαστηρίου Υπολογιστικής Όρασης και Ρομποτικής του ΙΤΕ από τη χρήση και τον προγραμματισμό ενσώματων ρομπότ δημιούργησε τις γενικές προδιαγραφές που θα ήταν επιθυμητές σε μια ρομποτική πλατφόρμα για ερευνητικούς σκοπούς. Με βάση αυτές τις προδιαγραφές και με τους λιγότερους δυνατούς περιορισμούς λήφθηκαν όλες οι αποφάσεις που αφορούσαν στην αρχιτεκτονική του όλου συστήματος και το σχεδιασμό και την κατασκευή των επιμέρους τμημάτων του. Για κάθε σχεδιαστική απόφαση εξετάστηκε η διαθεσιμότητα και το κόστος μιας έτοιμης ολοκληρωμένης λύσης. Στις περιπτώσεις που δεν βρέθηκαν ικανοποιητικές επιλογές, αναπτύχθηκαν και κατασκευάστηκαν εξ αρχής τα απαιτούμενα περιφερειακά.

Η βασικότερη προδιαγραφή υπαγόρευε ότι η ρομποτική πλατφόρμα θα έπρεπε να μπορεί να στρίβει επί τόπου. Για λόγους μηχανικής απλότητας και αξιοπιστίας επιλέχθηκε η λύση της διαφορικής κίνησης με δύο ελεύθερους τροχούς, μεταφέροντας έτσι την πολυπλοκότητα στους ελεγκτές των τροχών. Το φέρον πλαίσιο είναι μια απλή και εύκολη στη συναρμολόγηση κατασκευή από αλουμίνιο και αποτελείται από 4 κυκλικά πατώματα που συνδέονται μεταξύ τους με ράβδους και ορίζουν 3 διαμερίσματα διαφορετικού μεγέθους.

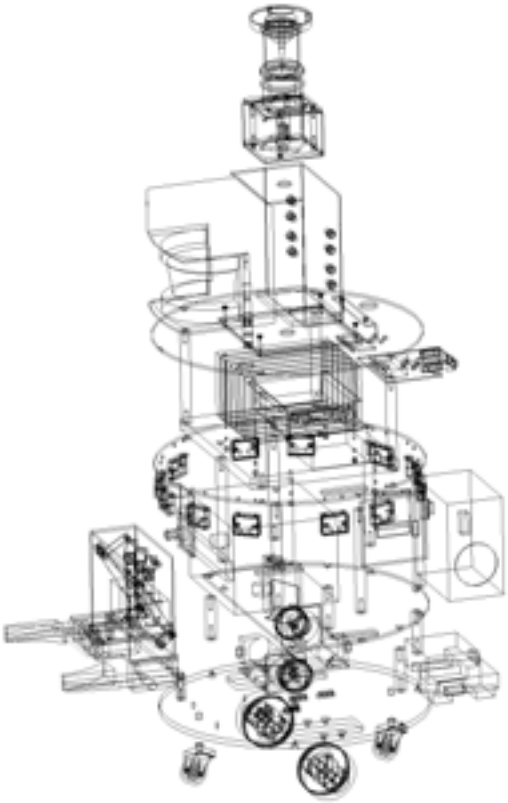
Η βασική σχεδιαστική φιλοσοφία που ακολουθήθηκε από την αρχή ήταν ότι η ρομποτική πλατφόρμα θα έπρεπε να αποτελείται από τελείως ανεξάρτητα modules , τα οποία θα επικοινωνούσαν ισότιμα μεταξύ τους μέσω του ίδιου διάυλου. Η δεύτερη επιλογή ήταν να ομαδοποιηθούν τα modules και όσα συνεργάζονταν να επικοινωνούν με αφιερωμένους διαύλους, το οποίο θα ήταν ίσως πιο αποδοτικό, αλλά θα δημιουργούσε δυσκολία στην ανάπτυξη, στην αποσφαλμάτωση και στην επεκτασιμότητα του συστήματος. Τελικά αποφασίστηκε ότι μόνο σε μία περίπτωση (ελεγκτές κίνησης) υπήρχε η επιλογή να χρησιμοποιηθεί ο ίδιος δίαυλος, αλλά το κέρδος από τη χρήση διαφορετικού αιτιολογούσε την επιπλέον πολυπλοκότητα. Επίσης το σύστημα όρασης ήταν αναπόφευκτο να έχει το δικό του αποκλειστικό μέσο επικοινωνίας με τον υπολογιστή, όπως και το σύστημα μέτρησης αποστάσεων με laser.

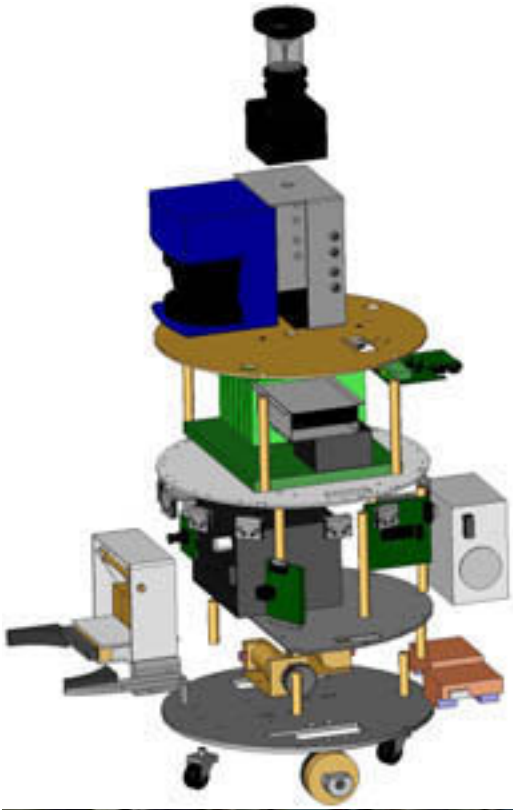
Ως δίαυλος επικοινωνίας μεταξύ των modules επιλέχθηκε το CAN bus, το de facto πρότυπο σε αυτοκινητιστικές και εργοστασιακές εφαρμογές. Ικανοποιεί όλες τις προδιαγραφές που επέβαλε η σχεδιαστική φιλοσοφία: ανοχή στο θόρυβο με ελάχιστους αγωγούς (2 ή 3 για ακόμα μεγαλύτερη αξιοπιστία), ομότιμους κόμβους, αξιόλογες ταχύτητες και μεγάλη διαθεσιμότητα ολοκληρωμένων κυκλωμάτων. Στους ελεγκτές κίνησης, εκτός από το CAN bus κρίθηκε αποδοτικότερο να χρησιμοποιηθεί και ο δίαυλος $I^2C$ . Οι τελείως διαφορετικές απαιτήσεις σε ρυθμό μετάδοσης δεδομένων για τις κάμερες οδήγησαν στην επιλογή του πρωτοκόλλου FireWire , με το οποίο μπορεί να υπάρχει εύκολη επεκτασιμότητα στον αριθμό των καμερών , κοινή διεπαφή προς

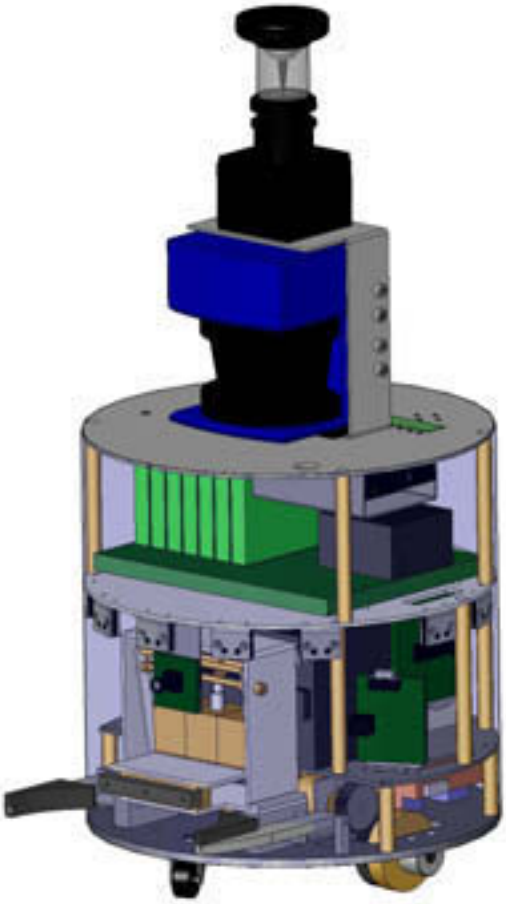
αυτές και ομοιογενής τρόπος χειρισμού τους. Το αποστασιόμετρο laser χρησιμοποιεί αναπόφευκτα επίσης έναν αφιερωμένο δίαυλο, τον παραδοσιακό σειριακό RS-232 ή RS-422,τη βιομηχανική παραλλαγή του πρώτου.

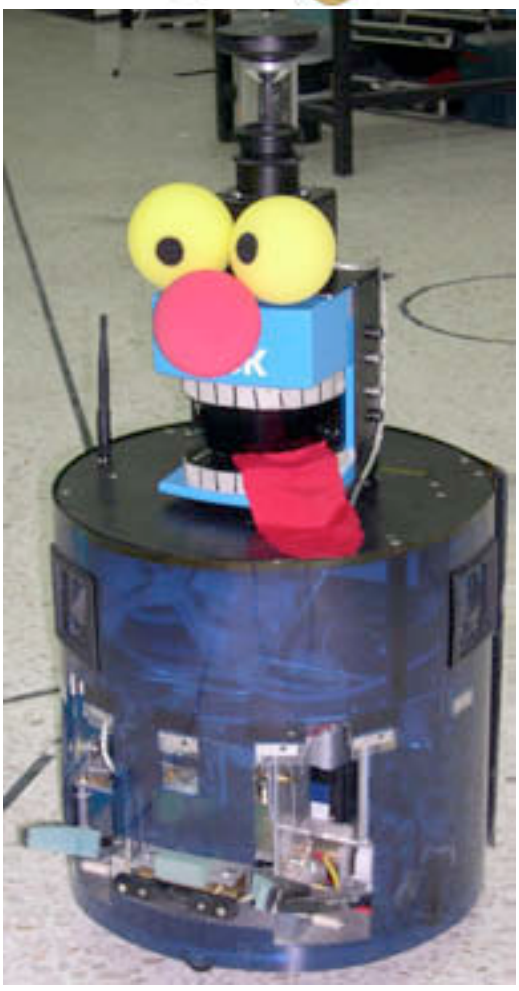
Οι βασικές λειτουργίες της ρομποτικής πλατφόρμας μοιράστηκαν σε 7 modules. Το laser αποστασιόμετρο, τους ελεγκτές κίνησης, τον ελεγκτή των υπέρυθρων αισθητήρων, τον ελεγκτή του μηχανισμού κλωτσήματος, τον ελεγκτή της κονσόλας ελέγχου και τηλεκατεύθυνσης, το σύστημα όρασης και τον κεντρικό υπολογιστή. Οι 2 πανομοιότυποι ελεγκτές κίνησης για κάθε τροχό, επικοινωνούν ανεξάρτητα από τα υπόλοιπα modules μεταξύ τους και εμφανίζονται σε αφηρημένο επίπεδο σαν ένας, όπως και το σύστημα όρασης, το οποίο μπορεί να αποτελείται από μια αλυσίδα μίας εώς έξι καμερών. Οι ελεγκτές βρίσκονται όλοι πάνω στον δίαυλο CAN , η αλυσίδα καμερών στον δίαυλο FireWire , το laser στο σειριακό δίαυλο, ενώ ο υπολογιστής είναι συνδεδεμένος και με τους τρεις. Ο υπολογιστής, οι κάμερες και το αποστασιόμετρο laser είναι τα modules για τα οποία χρησιμοποιήθηκαν έτοιμες ολοκληρωμένες λύσεις του εμπορίου. Οι ελεγκτές σχεδιάστηκαν και υλοποιήθηκαν από την αρχή για τη συγκεκριμένη εφαρμογή και καταβλήθηκε προσπάθεια να γίνουν όσο το δυνατόν πιο αυτόνομοι και ανθεκτικοί σε σφάλματα και να παρουσιάζουν ομοιογένεια στις διεπαφές τους και τα πρωτόκολλα επικοινωνίας τους. Στον υπολογιστή σχεδιάστηκαν και προγραμματίστηκαν τρεις διεργασίες/διακομιστές που αναλαμβάνουν να συντονίζουν τις λειτουργίες των modules του CAN bus, του συστήματος όρασης και του laser αντίστοιχα. Τέλος, η ανάγκη για πανοραμική κάμερα που να έχει όμως κοινό και συνεπή χειρισμό με τις απλές, οδήγησε στον μερικό σχεδιασμό του κατάλληλου κελύφους για τη συναρμολόγηση μιας αφιερωμένης κάμερας FireWire .

Ο σχεδιασμός και η κατασκευή της ρομποτικής πλατφόρμας απαίτησε το συνδυασμό πολλών επιστημονικών πεδίων και πάρα πολλές ώρες δουλειάς και σκέψης πάνω σε λιγότερο η περισσότερο σημαντικές λεπτομέρειες, που ήταν όμως απολύτως απαραίτητες για να γεφυρωθεί το χάσμα μεταξύ θεωρίας και πράξης · κάτι ευτελές για έναν θεωρητικό και ουσιώδες για έναν μηχανικό, που είναι όμως το απόλυτο σύνορο ανάμεσα σε μια ωραία ανεφάρμοστη ιδέα και σε κάτι που δουλεύει πραγματικά. Το αποτέλεσμα της εργασίας ήταν μια ρομποτική πλατφόρμα που δουλεύει πραγματικά και ως υπέρτατη δοκιμασία το σύστημα δοκιμάστηκε σε πραγματικές ερευνητικές εφαρμογές από τρίτους. Παρόλες τις αδυναμίες του ως πρωτοτύπου, η ρομποτική πλατφόρμα αποδείχτηκε σταθερή ως σχεδίαση και εύχρηστη ως περιβάλλον εφαρμογής. Το υλικό του και τα ψηφιακά και ηλεκτρονικά ισχύος ανταπεξήλθαν σε όλες τις δοκιμασίες. Με περαιτέρω χρήση του ρομπότ θα επέλθει και η ωρίμανση του κώδικα, ώστε να αποτελέσει ένα πολύ εύχρηστο, σταθερό και αποτελεσματικό ερευνητικό εργαλείο.









x

# Contents

# List of Figures

# List of Tables

The Encyclopædia Galactica defines a robot as a mechanical apparatus designed to do the work of a man. The marketing division of the Sirius Cybernetics Corporation defines a robot as " Your Plastic Pal Who's Fun To Be With."

THE HITCHHIKER'S GUIDE TO THE GALAXY

"And now for something completely different"

MONTY PYTHON

# Chapter 1

# Introduction

In the field of robotics there are experiments that cannot be carried out without real robots. Although there are many simulation environments, which permit fast and easy evaluation of proposed algorithms, there are always more or less significant, real-world parameters that cannot or shouldn't be modelled. Especially for agent-based behaviour algorithms it is stated that intelligence must be acquired through active experience with the world. Therefore, an agent can become intelligent only if it is both situated and embodied [1]. Moreover, almost all research fields in robotics require real robots, instead of simulations, as the final stage of evaluation of techniques and algorithms. Hence, the development of robotic systems is an important task in robotics research nowadays.

In this thesis we present the complete development and construction of a robotic platform. The primary goal was to provide a complete hardware and software bundle for researchers in the field of machine vision and autonomous mobile robots, taking into account their needs and good or bad experiences with other robotic platforms. Apart from that, one of the prerequisites was that the robot should meet the specifications of the middle sized league of the RoboCup competition. This imposed only a few restrictions in size and the ability to handle a ball. Apart from that, the robotic platform was expected to have as much features as possible, in order to be used in a broad range of research activities, and not to be a dedicated robotic soccer player.

The RoboCup competition provides a well defined framework for the testing of multi-robot cooperation, agent-based behaviour algorithms, and perception through

vision. Having in mind as the ultimate goal to construct a robotic platform that would eventually be able to serve as a prototype for a team of robots allowed to participate in the competition, the preliminary architectural guidelines for it were dictated by the RoboCup middle sized league player specifications (Appendix B) . The only restrictions posed by these were robustness, dimensions and the ability to handle a ball. Apart from them, the robotic platform was designed to be usable as a general research tool.

At all stages of the design and construction there were many alternative options to be followed. The final decisions were made based on one or more of the following and as few exceptions to these rules have been made as possible:

- Availability of commercial complete solutions or integrated circuits
- Performance
- Cost
- Expandability
- Ease of use
- Stability
- Fault tolerance

The architectural guidelines were laid down after discussions with researchers and students in the field of machine vision and robotics and according to their recommendations and requirements as application writers. This led to a first draft regarding a very abstract outline of the robot's functional parts and their connections. The construction of these parts would require one or the combinations of three fields: mechanical, electrical/electronic and software engineering. These three were required to interact very closely and on many occurencies decisions made in one field were mirrored in the other, or even caused complete redesign or reconstruction. In the course of development and construction, the details of the specifications and architecture were altered and fine-tuned many times to account for new empirical findings and technical difficulties.

Not every aspect of the design and construction can be covered in the following chapters, since this would require getting into many trivial or non-trivial details, which may not seem interesting, but required great amount of work in many fields and were absolutely necessary in order to account for the theoretical or practical problems encountered. From an engineer's point of view, the distance between theory and praxis is huge, and thousands of hours of work and thought make the difference between something that exists as an idea and something that really works. Sometimes, overcoming

the practical difficulties of the implementation may require much more effort and may be much more complex and intellectually challenging than the theoretical aspect of an idea. From an academic point of view this is often not understood or appreciated, but that's another sad story.

In Part I the general architecture of the system is presented. Chapter 2 states the mechanical considerations that were taken into account to design the robotic platform and explains the choices that were made for the components. In Chapter 3 the architecture and logical structure of the distinct modules that comprise the robot are described. In this part the different approaches are reported, as well as the reasons for rejecting them

Part II describes the robotic platform as it is implemented. In Chapters 4,5,6 we present respectively the mechanical, electrical/electronic and software components of the robot. The first real application that was implemented completely on the platform is discussed in Chapter 7.

The application interface to the control software is presented in Appendix A, while the rules of the RoboCup competition that dictate some basic aspects of the robotic platform design are stated in Appendix B.

"Back to the drawing board"

WILE E. COYOTE

# Part I

# Architecture of the robotic platform

"The major difference between a thing that might go wrong and a thing that cannot possibly go wrong is that when a thing that cannot possibly go wrong goes wrong it usually turns out to be impossible to get at or repair."

THE HITCHHIKER'S GUIDE TO THE GALAXY

# Chapter 2

# Mechanical Considerations

With respect to the mechanical aspects of the robotic platform, the most basic design decisions that have to be made, concern the carrying frame of the platform, which in turn depends on the choice of the movement configuration.

## 2.1 Driving Mechanism

As a robotic soccer player, the platform should be able to move and turn swiftly and it would be a great asset for its ball handling ability if it could rotate in place. As a research robot it should also be able to move with exceptional precision at very low speeds. The two obvious choices were the synchro-drive and the differential drive configuration.

### 2.1.1 Synchro Drive

The synchro-drive system is a two–motor, three/four wheeled drive configuration where one motor rotates all wheels to produce motion and the other motor turns all wheels to change direction. At any given moment all wheels turn an equal amount.

Figure 2.1 illustrates the case for a 60° rotation of the wheels in a 3 wheel synchro-drive configuration. Using two separate motors, one for translation and one for wheel rotation guarantees straight-line translation when the rotation motor is not actuated. This mechanical guarantee of straight-line motion is a big advantage over the differential drive method where two motors must be dynamically controlled to pro-

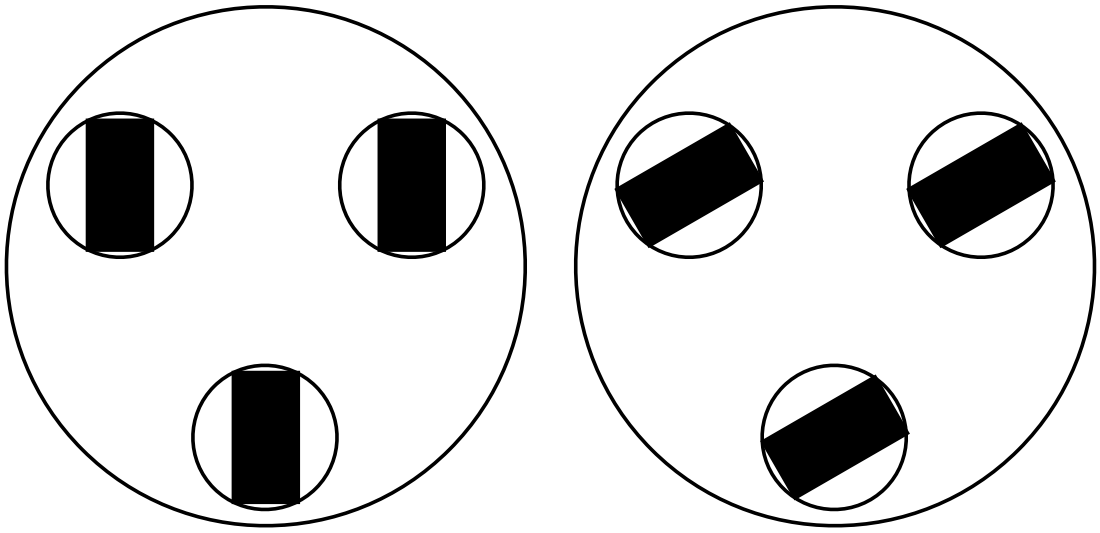


Figure 2.1: Synchro drive configuration

duce straight-line motion. Arbitrary motion paths can of course be done by actuating both motors simultaneously. The separate motors for translation and rotation make control much easier and straight-line motion is guaranteed mechanically — there is no need for high-precision synchronisation of the motors. However, the mechanism which permits all wheels to be driven by one motor and turned by another motor is fairly complex. Wheel alignment is critical in this drive system — if all wheels are not parallel, the robot will not translate in a straight line. This configuration is very often used in research robots and could meet all movement requirements of our platform at the cost of high mechanical complexity.

### 2.1.2 Differential Drive

The differential drive is a two-wheeled drive system with independent actuators for each wheel. The name refers to the fact that the motion vector of the robot is the sum of the independent wheel motions, something that is also true of the mechanical differential (however, this drive system does not use a mechanical differential). The drive wheels are usually placed on each side of the robot and toward the front. Yet, for the robot to be able to rotate in place without moving its footprint, they should be placed along its symmetry axis.

In figure 2.2, the large black rectangles are the drive wheels, each driven by a separate and independent motor. The small grey rectangles inside the circles are non-driven wheels which form a support structure for the body of the robot. Usually, the non-driven wheels are caster wheels, i.e. small swivelled wheels found in office furniture. Unfor-

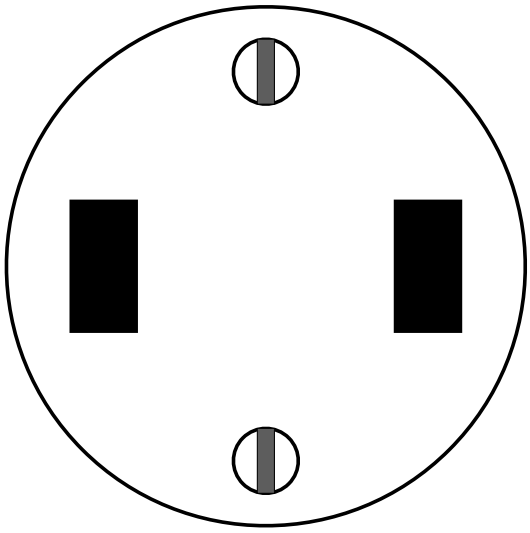


Figure 2.2: Differential drive configuration

tunately, caster wheels can cause problems if the robot reverses its direction. Then the caster wheel must turn 180 degrees and, in the process, the offset swivel can impart an undesired motion vector to the robot. This may result in a translational heading error. Straight-line motion is accomplished by turning the drive wheels at the same rate in the same direction, although that's not as easy as it sounds . In-place (zero turning radius) rotation is achieved by turning the drive wheels at the same rate in the opposite direction. Arbitrary motion paths can be implemented by dynamically modifying the angular velocity and/or direction of the drive wheels.

Mechanically, the differential drive system is very simple, and the drive wheel can be directly connected to the motor with the appropriate reduction gears. However it can be difficult to make a differential drive robot move in a straight line or follow a predefined path as closely as possible. Since the drive wheels are independent, if they are not turning at exactly the same rate the robot will veer to one side. Making the drive motors turn at the same rate is a challenge due to slight differences in the motors, friction differences in the drive-trains, and friction differences in the wheel-ground interface. To ensure that the robot is travelling in a straight line, it is necessary to adjust the motor current very often and it is also very important to have accurate information on wheel position from odometry sensors, which means that fast and very accurate motor controllers are needed. This configuration has the advantage of its mechanical simplicity, although to approximate the accuracy of the Synchro-Drive, more complex electronics and programs are necessary.

The electronic and programming complexity was preferred over the mechanical,

so a differential drive with two caster wheels was chosen as the driving mechanism of the robotic platform. The almost direct torque transmission with very few gears from the motors to the driven wheels would ensure low power loss ratios due to friction, an important feature to a battery powered mobile platform and also an asset to its agility, acceleration and peak speed. The positioning of the drive-wheels along the symmetry axis of the platform footprint provides the ability of in place rotation. Lastly, with the appropriate power and digital electronics and control algorithms, the problematic accuracy of the differential drive configuration could be enhanced to acceptable levels.

## 2.2 Motors

In order to achieve the requirements imposed by the intended use of the robotic platform, high quality motors are needed. They must be able to deliver high torques, which will ensure high acceleration and accuracy. The maximum speed is a secondary issue, because that can be easily adjusted with the appropriate reduction gears. It is much easier to design a very fast motion system than an accurate and responsive one, since most widely available DC motors are aimed at applications that require high rotational speeds. On the other hand, motors with high torques and wide envelopes of operation are not common stock items and must be specifically ordered from the manufacturer and not a reseller. Thus, they should be carefully selected to suit an application.

Another essential issue is that the motors must include quadrature-phase optical encoders in their casing. External solutions have proven to be highly inaccurate and sensitive to damage, so an integrated solution is necessary.

## 2.3 Carrying Frame

At this stage, only the most fundamental decisions about the robotic platform's carrying frame need to be made. As such, we decided for a frame whose footprint at any intersection parallel to the ground is exactly circular, i.e. a cylinder. This is sufficient to ensure that the robot can rotate exactly in place, that is without moving its footprint. It also simplifies the evaluation of sensor information, since there are no protruding parts that should be taken into account when estimating ranges measured or implied by the sensors.

All mechanical, electric and electronic components must be fitted inside a cylindrical body, whose footprint occupies at most a 50 cm $\times$ 50 cm square and whose actuators do not extend beyond the space denoted by 60 cm $\times$ 60 cm square, as imposed by the RoboCup rules. The actuators correspond to the necessary retractable

mechanical construct that would handle the ball. Taking into account the size of a full-sized ATX motherboard (30.5 cm $\times$ 24 cm), which will serve as the main computer of the platform, a cylinder with a diameter of 40 cm should be sufficient to meet all requirements, without getting too crowded with components or too bulky.

## 2.4 Ball Handling Device

The only component that is strictly specific to the RoboCup competition and not a requirement for a research robotic platform, is the ball handling device. We need a mechanical construction whose function is to loosely confine/hold the ball, an ability greatly constrained by the RoboCup rules , and push/kick it.

### 2.4.1 Holding The Ball

Only a few possible configurations are acceptable by the RoboCup rules. Two retractable arms, with the maximum size allowed for them, the right curvature imposed by the circumference of the ball and actuated by servos, should be sufficient. See sections B.1.6,B.3.2,B.1.3 in Appendix B for the constraints in actuator design and the allowed shapes for the robots.

### 2.4.2 Kicking The Ball

The design of the kicking mechanism is not at all constrained by the competition rules and is only subject to mechanical and electrical (power) considerations and difficulties. An appropriate mechanism should be able to effectively store and transfer with a linear motion the maximum possible amount of energy. Two other significant concerns are the repetition rate of the kicker and the ability to regulate the energy transferred to the ball. There are 4 available options to achieve one or more of the aforementioned prerequisites with varying degrees of efficiency.

- Electromagnetic actuators (solenoid magnets)
- Electropneumatic actuators
- Spring loaded actuators (charged by motors)
- Actuators operated solely by electric motors

A mechanism operated solely by motors could not be powerful enough for our application. A very high amount of energy in a very short time-span is required, and the conversion of rotational motion into linear proves to be very problematic, since the

momentary torque of a relatively small DC motor is not sufficient to move the ball effectively, whereas any reduction gears would result in a very slow mechanism. In order for the device not to be prohibitively expensive in power consumption, a way to store energy is required. There are basically two ways that can be used in our application to achieve this, by loading a spring or by charging a capacitor to a higher voltage than that of the power supply . Charging a capacitor alone would not suffice, since this couldn't increase the transferred energy greatly enough without damaging the motor. On the other hand, a motor charging a spring could be powerful enough. However, there is the disadvantage of very low repetition/firing rate, typically some seconds before each recharge, and if we were to incorporate the ability to regulate the firing power, a very complex mechanism would have to be designed.

An electropneumatic actuated kicking device would probably exhibit the best functional characteristics. However, it would require a refillable gas tank. We chose not to adopt this solution, as we wanted all functions of the robotic platform to depend solely on its battery.

Solenoid magnets seem to be the best choice for our application. They don't provide the maximum possible thrust compared to the other configurations. Yet, they satisfy all other prerequisites of a suitable kicking device. Their firing power can be easily regulated almost linearly from zero to maximum by use of pulse width modulation and they have a very fast repetition rate. They also can be operated at more than two times their nominal voltage, thus the kinetic energy they provide can at least be quadrupled just by adjusting their supply power.

"But remember our job isn't really to destroy equipment or frighten the daylights out of our users. That's an added bonus in our selflessly devoted lives as technical support persons."

BASTARD OPERATOR FROM HELL

# Chapter 3

# Electrical, Electronic and Programming Considerations

## 3.1 Logical structure

At the logical level, we need to define the logical and functional blocks of the robotic platform and their connections to each other. Our basic design philosophy states that these blocks or modules should be independent and self-contained, yet should be able to cooperate efficiently and integrate seamlessly into complex subsystems. Taking into account the sensory, action and control equipment that is dictated by the intended use and operation of the robotic platform, we define the following modules:

- Computer
- Movement controller
- Infrared sensors controller
- Main console and display controller
- Kicker controller
- Vision system
- Laser measurement system

These 7 modules can be divided into 3 groups, each with different functional and bandwidth requirements. The computer will serve as the central coordinator and is the common point between the three groups, the first of which includes only the vision system, the second the laser measurement system , while the remaining controllers belong to the third. Although this suggests a hierarchical structure with the computer as the root node, there is also the requirement that the modules of the third group should also be able to operate and communicate independently.

## 3.2 Vision system

The vision system is intended to be the main sensory equipment of the robotic platform. Thus it should be designed to be as efficient and versatile/expandable as possible. By its nature it requires very high bandwidths, so a very fast communication bus to the computer is required. Apart from that, the requirements from the vision system for the deployment of the robotic platform both as a soccer player and as a general research robot indicate the need for more than one camera.

Theoretically, the fastest bus that can be used for video grabbing is the PCI bus at 127 MBytes/sec (32 bit@33 Mhz). Yet, a separate frame grabber card would be needed for each one or two cameras, depending on the type and the capabilities of the card. Besides that, the most common available cameras output their video signal as composite, offering degraded quality over more expensive ones with S-video, and also need an external power supply. Fast frame rates and good picture quality require both expensive frame grabbers and cameras. If any of them is of lesser quality it would be a bottleneck to the vision system, reducing its efficiency to that of the worst component. Lastly, adding more cameras requires more frame grabbers, so the expandability of the vision system in terms of hardware and software is not very straightforward

The configuration that seems to overcome the restrictions of frame-grabbers and satisfies most of our requirements, is a vision system based on FireWire cameras. The FireWire bus provides, at 400 Mbit/sec, a consistent interface for medium to high bandwidth data transfer , such as video grabbing, and is becoming the protocol of choice in computational vision research. One important feature for our application is that relatively cheap and good quality FireWire cameras are widely available. Even more important is the fact that FireWire devices can be connected in a daisy chain fashion, up to a theoretical number of 63 devices per controller, though the camera bandwidth requirements practically limits this to less than 10, depending on picture resolution and frame rate. Thus, the number of installed cameras can be expanded just by adding more to the FireWire chain, with no other modifications in software or hardware, provided

that the coordinating software is designed not to make any assumptions about the number of available cameras.

For the purposes of the RoboCup competition it was decided that the robotic platform would require at least one panoramic and one wide angle camera pointing at the front. This configuration could also serve many other robotic applications, but the basic software and hardware must be able to support at least 2 regular cameras besides the panoramic, since one of the main fields of computational vision and its applications in robotics deals with stereo vision.

## 3.3 Motion, sensors and control

The backbone of the robotic platform comprises of four modules: motion, IR, kicker and main console and display. Each module must be able to operate independently of the others in case any of them fails, since they control the most basic functions of the robot. Any malfunction may in the worst case lead to the damage of the hardware or the surrounding environment, e.g. in the case of an erratic motion controller that causes the robot to cruise at full speed towards a wall. The physical medium must be immune to noise, which may be significant taking into account the amount of electrical equipment in the body of the robotic platform and especially the relatively powerful induction motors. In addition to that, it is even more important to design communication protocols and firmware programs that are as fault tolerant as possible. Non critical errors must not interfere with the operation of the other modules and should be automatically corrected without user intervention when possible. Critical errors should always leave the robot in a safe state.

### 3.3.1 Module Communication

The fact that the presence or absence, as in the case of a malfunction, of a module should not affect the others, imposes a bus topology for the communication medium. The most common used one in heavy electrical noise environments is the CAN (Controller Area Network) bus. Its noise immunity is well tested in millions of installations and although it is a relatively slow medium, it easily exceeds by far the bandwidth requirements of the modules in question. For the physical layer, a twisted pair multi-drop cable is specified with a length ranging up to 40 m at 1 Mbps, although it is typically operated at 500 kbps and below (according to the physical layer specifications as defined in the ISO-11898 standard ([2]).

One of the most important features of the CAN bus for our application is that its MAC (Multiple Access Control) algorithm features a priority based non-destructive ar-

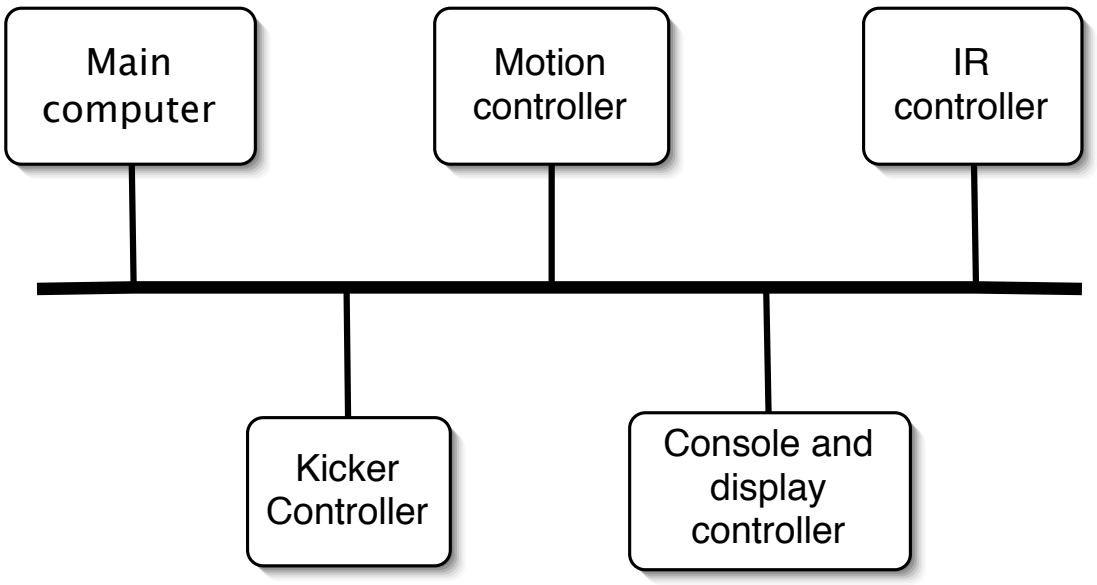



Figure 3.1: Bus topology

bitration, due to the open collector logic of its physical layer. That is, each node is assigned a different priority level and its messages take precedence over the other nodes' massages. If two nodes with different priorities start transmitting at the same time, the one with the lowest priority just backs off without destroying and thus delaying the message of the other node, which for example is the case with ethernet. See reference [3] for a more detailed presentation of the CAN bus protocol.

### 3.3.2 Motion Controller

The most important module is the one that controls the motion of the robotic platform. The following requirements indicate its complexity:

- It must handle efficiently high currents for long periods of time
- It must protect the motors and the power electronics from overloading and overheating
- It must be able to measure the position of each wheel as precisely and fast as possible
- It must be able to follow a given trajectory as closely as possible, without stressing the mechanical and electronic parts in normal conditions, even when that is requested by the trajectory
- It must be able to handle peak loads and stresses in case of emergency
- Its controlling firmware program must be as bug free as possible. Since the latter is almost an illusion in programming, at least at the first stages of development, it must

at least be able to recover from any failure without risking damage to the platform, the electronics, the motors or the environment. This applies to all modules and programs in our application, however only the motion controller can really damage anything, so these guidelines must be followed much more strictly.

- It must off-load the main computer from any CPU-intensive work that has to do with odometry or following a trajectory.

Since there are two motors, all the power electronics and most of the digital ones would have to be used twice. Most failures occur usually in power electronics, so on grounds of easy repairability, the need for two identical but separate motion controllers arises. However this would place considerate computation and synchronisation load on the main computer. Thus, a master-slave configuration has been chosen, although this increases further the complexity of the controller firmware. That is because the two controllers must communicate each other's errors, so that no damage can occur if any of them is left inoperable, or even just recovering from a non critical condition.

The two controllers should appear as one module to the others. Internally they must exchange information and odometry very often in order to be kept synchronised and calculate the robot's position. The estimation of position in a differential drive robot requires some trigonometric calculations and is typically carried out in the main computer, taking into account separate odometry readings from each wheel. It is hopeless to think that a complete position estimation feature can be provided at the motion controller, since odometry readings are by nature highly inaccurate and also there are many complex estimation algorithms used in different applications, so restricting the user by choosing one of them wouldn't be reasonable. Yet most of the times the main computer must make simple computations for each wheel as fast as possible, to approximate the curvature of the recent trajectory. With that in mind, it seems appropriate to provide on demand both separate odometry readings from each motor or the trajectory. The controllers themselves can refresh their readings much more often, thus providing adequate accuracy for some centimetres, and even meters if the application is not very demanding. The main computer can decrease its CPU and bus load and yet achieve the same uncertainty probability as in the case with separate wheel readings, for many different positioning algorithms implemented on it.

In order to provide the position calculation feature, the master controller must query the slave on regular and short intervals. Since they are identical in hardware, this communication can be channelled through the CAN bus. However this will load the bus with information that is not useful to the other modules. Messages in the bus must be initiated either by the main computer, by triggered events or error conditions and

not by other means. We chose to reserve the bandwidth of the CAN bus for the inter-module communication solely and dedicate another medium for the exchange of information between the motion controllers.

Since we will use a complementary bus anyway, it is reasonable to examine whether it can have other uses as well. As a robotic soccer player the platform will have to follow a ball, thus deviating from the commanded trajectory, so its movement will have to be adjusted by the application program very often. It seems possible that the refresh rate of the movement commands can be reduced if we can provide some helpful feature for ball following at the hardware level. The motion module controls the motors, based on the difference between the desired and the actual trajectory. The desired trajectory can be adjusted given information from the kicker controller about the distance to the ball, provided it is in the range of its sensors. Thus the complementary bus can be also used to provide an independent way of communication between the motion and kicker modules, which will form a close cooperating entity.

The medium chosen is the $I^2C$ (Inter Integrated Circuit) bus. $I^2C$ is a synchronous, serial 2-wire protocol used extensively for the communication of ICs in a wide range of applications. It is not very immune to electrical noise, however it is reasonable to assume that the motion controllers can be conveniently placed close enough so that simple noise protection measures would be sufficient. Moreover, since the bus is synchronous, its speed can be adjusted to achieve a desired error probability up to a maximum of 400 kHz, which, taking into account processing delays in the microcontrollers, CRCs and framing bits, corresponds to data rates of more than 200 kbps. That would be more than enough bandwidth for the internal communication requirements of the motion controllers.

The available lead/acid batteries that are big enough to provide the necessary power for the computer alone and for a period of some hours weigh over 10 kg. The laser measuring system discussed below weighs 5 kg. So the motors of robotic platform have already 15 kg to move, and that without taking into account the frame and the rest of the equipment. In order to make some calculations about the motor specifications, it is safe to assume that the final weight of the platform will be over 40 kg. We want to accelerate and decelerate these 40 kg at 1 m/sec$^2$. The friction is difficult to estimate, yet a high coefficient of 0.3 would take into account many possible losses. By using wheels of 80 cm in diameter we will need a torque of

$$T = Fr = (F_{friction} + ma)r = (40 \cdot 9.81 \cdot 0.3 + 40 \cdot 1) \cdot 0.04 \approx 6.3 \text{ Nm}$$

So we will need motors able to provide constantly after reduction about 3.5 Nm of torque each. This applies when the robot is accelerating from a stopped position. We

also want to be able to achieve this responsiveness when the robot is moving at least at one third its peak speed. For a peak speed of 1.5 m/sec this leads to the following power estimation for each motor:

$$P_{output} = T\omega = T2\pi\frac{\frac{v_{max}}{3}}{2\pi r} = 3.5 \cdot \frac{0.5}{0.04} = 43.75 \text{ Watts}$$

The gearbox efficiency is typically 90% per stage of reduction, as stated in most small motors datasheet, and the power loss of a good quality DC motor is about 10%. For 3–4 reduction stages we achieve an efficiency of 60–70% and thus each motion controller must be able to provide constantly constantly at least 80 Watts of power to the motor it controls. This figure should be expected to be even doubled at peak loads, meaning that the power electronics of the controllers must operate safely at 10-15 Amps, since the available DC voltage for the operation is determined by the commercially available lead/acid batteries which output 12 Volts. There is no suitable integrated circuit to drive DC motors capable of supplying that high current readily available. We will need to design the appropriate power electronics (H-bridge and driver) to meet these specifications.

### 3.3.3 IR Sensors Controller

The robotic platform needs a way to detect immediate obstacles. A circular belt of sensors will be used to give rudimentary information about nearby objects. There are three options for the kind of sensors suitable for this.

- Sonars
- IR sensors with distance estimation by triangulation of the reflected beam
- IR sensors without range estimation, triggered by the intensity of the reflected beam

This feature is included primarily for safety reasons and not for localisation, since even the most sophisticated sonars or IR sensors with distance estimation are almost incapable of providing the precise and dense measurements necessary for this task.

Sonars are useful for the estimation of medium range obstacles, typically from 60 cm to 10 m. Their accuracy is limited and cannot correctly detect immediate objects. Although they are very useful in some cases, they are unsuitable for a robotic platform designed to meet the specifications of the RoboCup competition and cooperative robots research. When two or more robots with sonar sensing equipment operate in the

same environment, multiple echoes from the sonars can and will confuse their readings.

IR sensors with range estimation, use a low resolution CCD array to measure the reflection angle of the IR light beam and compute the distance of the reflection surface by triangulation. They are fairly accurate and can detect obstacles in distances that range from 4 cm to 1 m, depending on the their type. However, like in the case of sonars, it is most likely that the sensors of two identical platforms standing in close range would interfere with each other.

We chose to use on/off IR sensors, without exact distance estimation, adjusted to detect obstacles closer than 20 cm, in order to overcome the interference problem. This kind of sensors can be easily used to emit certain patterns so that the real source of the IR beam is recognised, with sufficiently low error probability

The IR controller must be able to provide power to, control and synchronise all IR sensors. It must activate the sensors in such a way as to minimise the interference to each other and also to other identical robots, if present. Since the information about triggered sensors indicates immediate obstacles and possible collisions, it is very critical and must be broadcast on the bus without explicit request from the user or other modules.

### 3.3.4 Kicker Controller

The kicker controller operates the kicking mechanism and evaluates the sensors attached to it that sense the presence of and distance to the ball. It must have the necessary power electronics to supply the solenoids of the kicking device with variable duration pulses of about 10 Amps, in order to control the strength of the kick. It will also have to operate the ball restraining arm.

At least two sensors are needed to assist in the handling of the ball. The first can be positioned at the ball restraining arms to sense the presence of the ball between them. A simple IR emitter on one arm and a receiver at the other would be sufficient and will be operated by the kicker controller, so the proper electronics should be included in its design. The second sensor is needed to measure the distance between the ball and the frame, once it lies between the restraining arms, and will be positioned at the appropriate place on the front part of the robotic platform. Thus, the interface to a range estimating IR sensor must be taken into account in the design of the kicker module.

The kicker module will have to accept commands to kick the ball, lower or raise the handling arms and report the state of its sensors. It will also have to cooperate with the motion controller on demand of the user, by sending to it through the $I^2C$ bus

information about the distance to the ball, so that it can be used to implement the ball following capability.

### 3.3.5 Control and Display Module

An external interface to control some basic functions of the robotic platform and display status information about it would be extremely useful. Although all its functions can be easily implemented in software on the main computer, there are cases where this is not convenient or even possible, hence the need for an easily accessible external interface. The control and display module will integrate various input/output options, apart from the necessary CAN bus: An LCD display, warning LEDs, a button panel, a speaker for audible alarms, a serial port, an RF receiver, connections to the main computer (on, reset and LEDs) and a joystick port. Using its I/O interfaces, the module will have to:

- Indicate battery state
- Indicate errors
- Indicate miscellaneous information about the state of the platform
- Switch the main computer on and off
- Accept movement commands from RF controller
- Accept movement commands from joystick
- Accept movement and basic configuration commands from its button panel

Movement commands will be directly sent to the motion controllers irrespective of their source. All queries about the state of the other modules will also be sent directly to the respective ones, without the intervention of the main computer, so that the C&D module can retain all its functions whether the computer is switched on or not. However, although it is easy to switch the computer on just by pressing a button, it is more difficult to turn it gracefully off. We will need a serial connection between the computer and the module, so that the latter can log in and issue the appropriate shutdown commands.

## 3.4 Laser Measurement System

The laser measurement system provides an extremely accurate estimation of the range of objects around the robot, in a fixed horizontal intersection of the surrounding space. Although not very flexible, since it is mounted at a fixed height, it is accurate enough

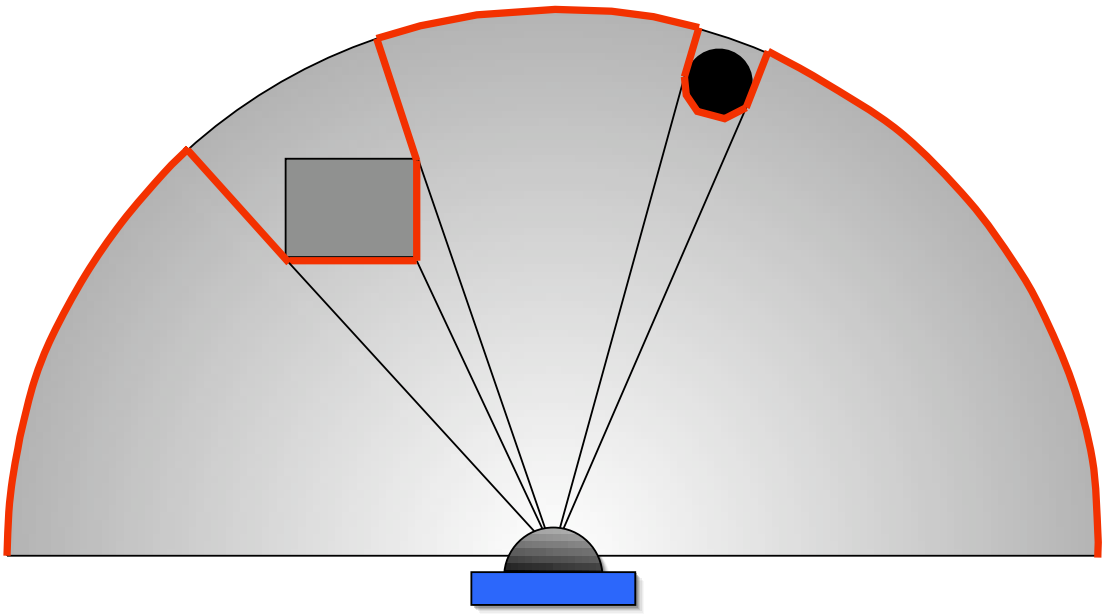


Figure 3.2: Range measurement by laser

to be used by itself for localisation. The device to be used is a well known in robotic applications industrial laser measurement system that has a 180° wide field of view, i.e it scans a half-plane passing through it. There are two issues to be taken into consideration when choosing the location of the laser. Firstly the mounting height, which defines the distance of the scanned half-plane to the ground, must be carefully chosen in order to detect as much of the most often encountered obstacles as possible. Secondly, the frame structure around the laser must be such as to not hinder any laser beam, in order to use the maximum scanning area of the device.

A height of about 30–50 cm from the ground would be optimal to detect most common furniture. The lower end of this range and maybe even lower would make the robotic platform better suited for the RoboCup competition. However, due to the imposed size constrains, positioning the measurement system at this height would either lead to a complex frame structure and/or bigger dimensions, in order to use the full scanning field, or we would have to limit the usable scanning field. The other option is to try and keep the height of the robotic platform as low as possible, and position the laser on top of it, where no surrounding structural elements can limit its field of view. This would mean that it would be positioned somewhere at the upper part of the optimal range, where it would be much less useful for the RoboCup competition. We will choose this option, since the simplicity far outweighs the drawbacks and its combined use with the vision system and IR sensors would certainly meet the required localisation accuracy of the competition. Moreover, by being mounted on the exterior, the measurement system can be easily removed and used elsewhere, a great asset for

such an expensive device.

The communication to the device is carried out through an RS-232 or an RS-422 serial interface, so we are bound to use this bus as well. The RS-232 interface is still available by default on all motherboards, however to be able to use the maximum refresh rate of the laser, we will have to use the industrial RS-422 differential serial interface by means of an adapter or a PCI card.

## 3.5 Computer and Software

The computer and the servers running on it are the layer that separates the application programs from the underlying hardware. It must provide the necessary tools and utilities for the programming of robotic behaviours, while the servers must provide a robust interface for the programs to the modules of the platform.

The tools and the utilities, although a major concern, are more or less standardised and readily available. In this topic we can only try to make the best choice for an operating system. However, the server programs are quite complicated pieces of software, since they must synchronise and keep track of many modules and application programs while being as consistent, robust, stable and efficient as possible. All design decisions must be made taking into account worst case scenarios. Everything must be programmed with the assumption that many and probably not well behaving clients will be accessing the services provided. Received commands must be checked for validity and access rights and permissions must be defined for each application to minimise the risk of damage due to an erratic program. The users will interface to the facilities of the robotic platform by means of simple commands given in programming libraries. These commands will correspond to simple or very complex procedures that must be followed by the servers to accomplish the requested action or to gather and return the desired data.

This robotic platform is a prototype. And things go bad in prototypes more often than they go well. At any point in the code of the server the assumption must be made that something can go wrong and at all critical parts there must be procedures to check for the presence of errors, in case of which the best possible correction should be made. According to the severity and the kind of an error:

- It is corrected without the user noticing it.
- If not possible to correct it, the robotic platform must be brought to a safe state and the user must be notified.
- If the error seems to be on the part of the user, he/she is either notified or disconnected

according to the situation, in order to protect other users. If the action to be taken is to disconnect the user, the robotic platform must be brought to a safe state.

The servers must be programmed without polling and infinite time blocking functions so that nor internal bugs nor buggy clients can make them hang. Only errors in system calls or library functions should make the server to exit with failure.

The three primary busses used imply the use of three separate servers, each of them controlling all the modules connected to a single bus. This is primarily to ensure that each server has the complete knowledge of the state of the bus it controls and to a lesser extent a decision suggested by the logical function of the modules. A different option would be for example that a server would incorporate all perception functions while another all actions. This would however mix CAN and FireWire bus devices, making synchronisation and error handling on modules connected on the same bus, but not controlled by the same server, very difficult. Apart from that, the completely different bandwidth requirements of the busses and the modules connected to them imply completely different approaches to the architecture of each server.

The control of the laser measurement system is the easiest of all. It uses for its communication the simplest interface available, the traditional serial port. Since it occupies a bus by itself and this excludes synchronisation issues, its control could be assigned to a server responsible for other modules as well. However, the laser can operate either as a low bandwidth (38.4 kbps) or a medium bandwidth (500 kbps) device, according to the type of serial interface used. This means that the mode of operation suggests either the vision server, which is designed for high bandwidth data transfers, or the low bandwidth CAN bus server as more appropriate to accommodate the laser measurement services. Yet we cannot choose between the two interfaces. The RS-232 interface may be slow, yet it is available on all computers and most electronic devices, while the RS-422 is reasonably fast but very uncommon. Moreover, the device will be detachable, in order to be also used on other platforms, which suggests a fully separate server. So, the controlling server must be standalone and able to use both interfaces, which is easy in programming since these serial devices are handled in exactly the same way by the operating system, but its architecture must be aimed towards the medium bandwidth operation.

### 3.5.1 Main Computer

The main computer will be the host of the servers and also of the application programs. It has to be connected to both busses so the appropriate interfaces are necessary. FireWire is standard in most motherboards today, but an uncommon CAN interface card must be separately installed along with the usual peripherals. Also a

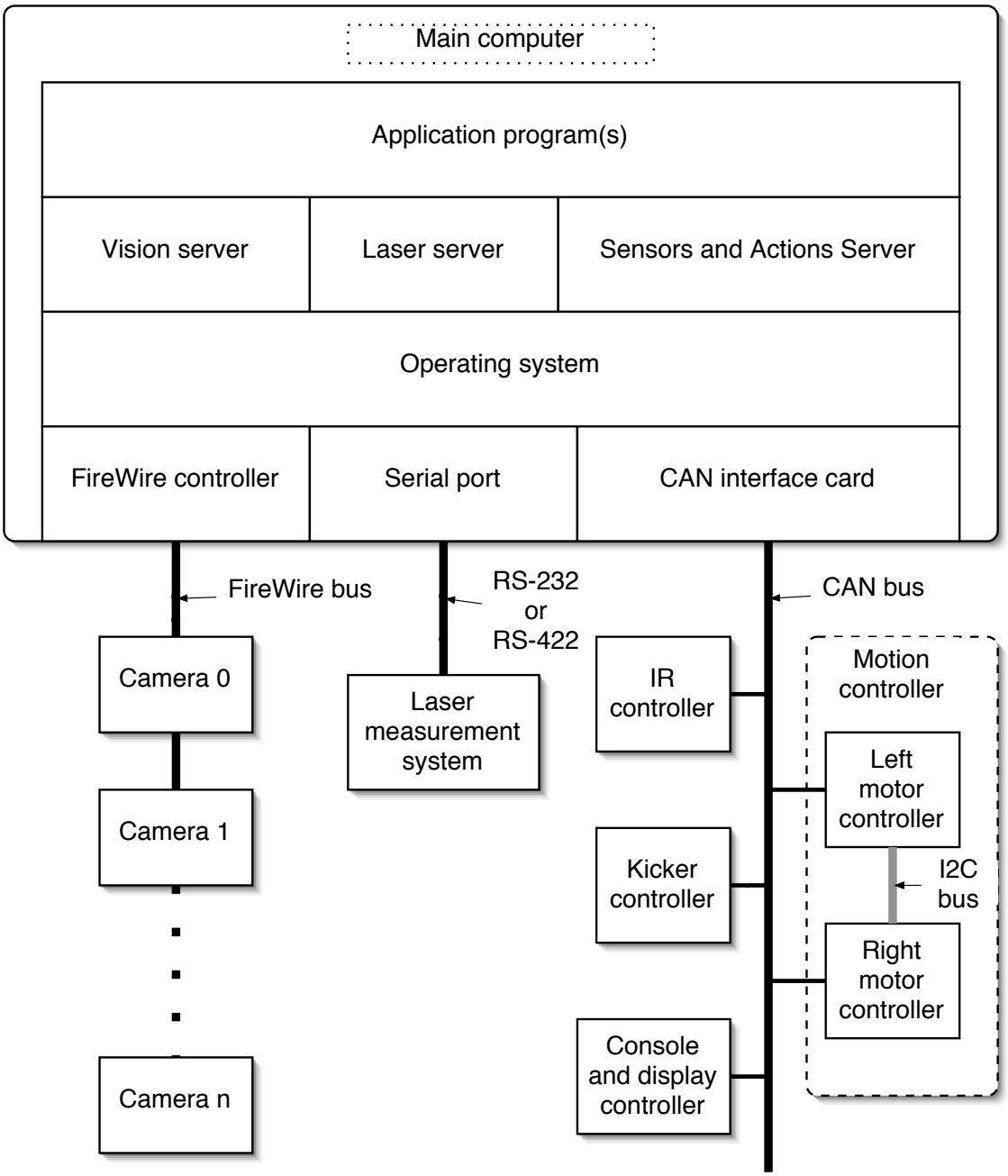



Figure 3.3: System architecture

wireless network (WiFi) card will be needed for the remote control, data exchange and programming of the robotic platform.

As regards the operating system there are primarily only two options, namely Windows or some flavour of UNIX . Windows, apart from being a terrible excuse for an operating system, doesn't provide by default the necessary development tools and applications. Moreover, the architectural philosophy of UNIX fits to the requirements posed to the server programs. They must operate as efficiently and close to the hardware as possible, and UNIX provides a very thin, yet powerful layer to it. On the contrary, Windows intervenes many convenient abstraction layers between the application and the hardware, which in the case of the servers is not useful and adds much overhead. Also, the networking capabilities of UNIX are far more advanced and the application programs running on it seldom make any distinction between local and remote platforms, which are very convenient both for the development and the normal operation of the robot. Finally, the availability of free open source software for UNIX , could be the reason for its choice by itself. The only substantial decision for the choice of the operating system that is left to be made is the flavour of UNIX . There are two viable options here too, Linux and FreeBSD. We have chosen Linux , primarily due to the more wide range of drivers, utilities and libraries available for it.

The consistency of network and local resources in UNIX implies that the servers should be programmed in such a way that they do not make any assumptions whether the controlling program is executed remotely or not. However, due to the nature of network communications and especially wireless ones, much attention must be paid to error handling matters like abrupt termination of connections .

### 3.5.2 Vision Server

The server that will control the operation of the FireWire bus has only the vision system under its supervision, so all its functions and facilities are dedicated to vision. Its first function is to initialise the bus, check for available devices and to configure and synchronise them. No specific assumptions should be made about the number of cameras installed and their capabilities. It is the server's duty to find these out and to provide a consistent interface to the user regardless of them.

Contrary to what was said about the uniformity of local and remote applications, in the case of the vision server this doesn't make much sense. The very high bandwidth required by video makes the development of remote image processing applications highly improbable. A stream of uncompressed $640 \times 480$ RGB images from one camera at 15 frames per second would require a bandwidth of 13.1 Mbytes/sec, which is slightly more than the nominal throughput of the available cabled network interface and an

order of magnitude more than the wireless one. By deciding that the server will not have to serve remote clients, we have at our disposal many powerful facilities of the operating system which are provided to local applications only and are aimed at speed and efficiency, like shared memory.

The main function of the vision server is to provide the clients with images when requested. It operates the cameras on behalf of the client programs without them having to know any details about them. For technical reasons and to achieve the maximum efficiency, the server must sustain a constant flow of images from the cameras and keep them buffered, so it can serve any request without additional delays, apart from the duration of the negotiation with the client. A primary requisite is that at any time there can be an arbitrary number of clients, up to a maximum. The server must be able to synchronise its procedures and reply to all requests as quickly as possible and without letting any problems caused by a client interfere with other ones. This holds for errors as well as for normal operation, meaning that the response time to requests must not be greatly affected by the number of users.

### 3.5.3 Laser Server

The laser server initialises and controls the laser measurement system through the serial port. The device can refresh its range readings up to a maximum of 75 Hz, each of them carrying a payload of 800 bytes, which including frame bytes, synchronisation and CRCs leads to about 500 kbps. This is not an especially large bandwidth, but combined with the fact that the laser sends continuous streams of data forces us to use a similar architecture as the one used in the vision server. However, this amount of data is not as high as to justify the exclusion of remote application in favour of speed, meaning that the server will make use of facilities common to local and remote applications.

In order to achieve the minimum response delay to requests for reading, the laser server will also have to implement an efficient buffering feature. However, although the CPU utilisation at peek refresh rates will not be substantial, it would be also useful to implement a reading on demand feature, adding at most $1/75 \approx 13.3$ msec to the delay of each reply, imposed by the time needed by the scanner to prepare its next reading. This will totally free the CPU when no readings are requested, without adding much to the complexity of the laser server. Anyway, it is not very likely that applications will often have to evaluate range data at the maximum rate, so the gain in available CPU time justifies the small delay when using this feature and the extra complexity when implementing it.

### 3.5.4 Sensors and Actions Server

This server will control and monitor all modules connected to the CAN bus. The bandwidth requirements are very low, taking into account that the most used module, i.e. the motion controller, will be able to follow simple trajectories, as simple as accelerating and decelerating smoothly to reach a specified point, with just one command. This way there is no need for constant corrections to the course, or at least not as many as in a more dumb controller like that used in the B21r scientific robot, thus limiting the bandwidth of the CAN bus used under normal conditions and also the CPU utilisation of the server controlling it.

Although the bandwidth is low and therefore not much effort must be dedicated on matters of efficiency , the server has a rather complex architecture. The reason for this is that it controls the modules likely to exhibit most failures, since they are all prototypes, and among them the one that deals with motion and most likely to cause damage. So the server must be very thorough about detecting and handling errors. All commands sent to the modules must be checked and acknowledged by them and the server must be able to keep track of them. It must recognise acknowledgement timeouts , which means that all its blocking functions must be either timed or controlled by a watchdog thread. On any error, it must be able to bring the robot in a safe state and, if possible, resume its normal operation after or without notifying the user, depending on the severity and/or settings set by the user.

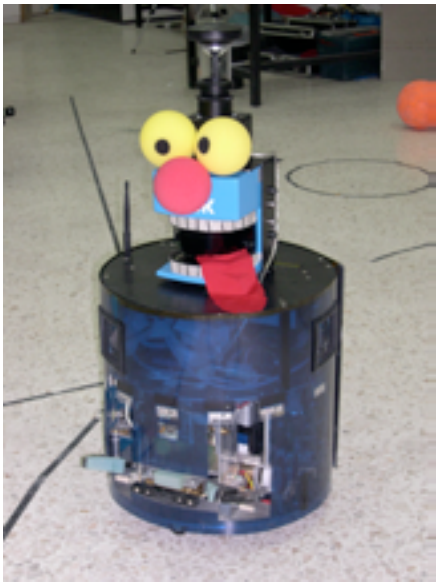
For reasons of both safety and simplicity, two types of connections will be served by the program. Active or primary and passive or secondary ones. Only one primary connection is allowed, and the program which has established it has the complete control of the robotic platform, i.e. it can move it and read all the readings provided by the sensors & actions server. On the other hand, there can be many secondary connections using a subset of the same command set, but these will not be able to change the state of the robot in any way. This means they can have access to all sensors and readings, but cannot move the robot nor change the configuration of any module or the server. The only privileged command available for safety reasons to both connection types will be the kill command that stops the robot and denies the processing of any motion command until specifically cancelled. The secondary connections are intended to be used for monitoring purposes, while the main behaviour application will occupy the primary one.

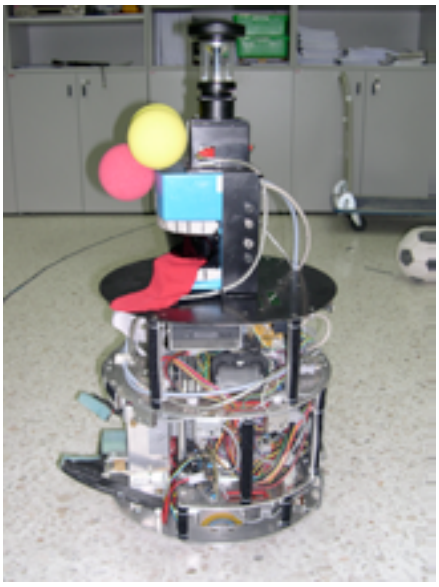
"In a solitary chamber, or rather cell, at the top of the house, and separated from all the other apartments by a gallery and staircase, I kept my workshop of filthy creation"

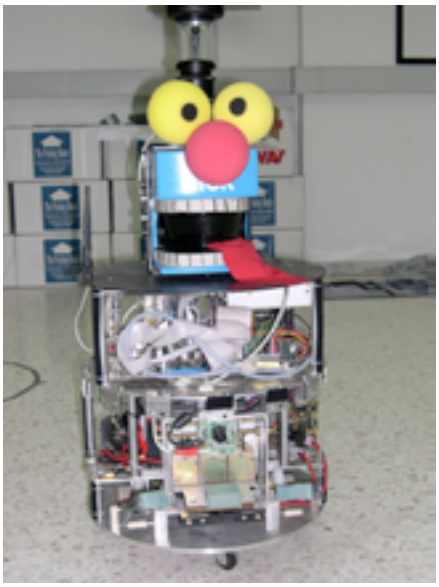
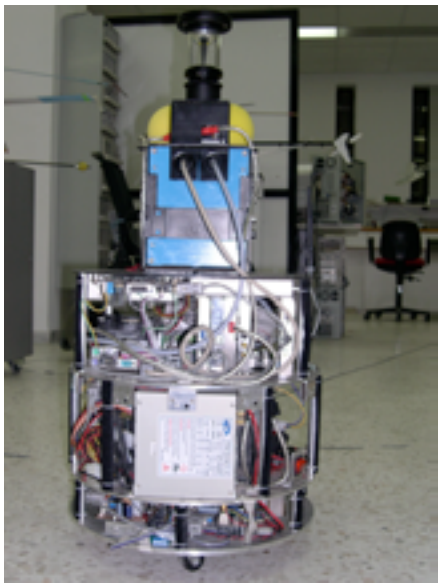
Dr. Victor Frankenstein

# Part II

# Implementation of the robotic platform





> It was a crazy piece of near junk. It looked as if it had been knocked up in somebody's backyard, and this was in fact precisely where it had been knocked up. The astonishing thing about the ship was not that it was one well (it wasn't) but that it was done at all.
>
> THE HITCHHIKER'S GUIDE TO THE GALAXY

# Chapter 4

# Mechanical Engineering

The carrying frame consists of 4 aluminium discs of 40 cm in diameter. The bottom plate has a thickness of 8 mm, the two middle ones 4 mm, and the top cover 3 mm. They are connected to each other by means of brass rods of varying length, thus forming 4 compartments of different sizes. The lower three are closed and well defined in size by their respective top and bottom plates, while the fourth is open, having only the top cover as its bottom plate. Appropriate cutouts for the cables that are used for the miscellaneous electrical connections of the components are drilled on every plate. From bottom to top, we have the following compartments:

- Motor compartment
- Battery compartment
- Computer compartment
- Top outer compartment

During the construction and assembly of the components, the need for many minor changes in their design has arisen. No effort has been spared in altering and even completely redesigning some mechanical parts. Fortunately there were only a few of them. In most cases very slight adjustments were sufficient or even not at all. In the following pages only the final decisions will be presented and only the components that required some effort in design to address certain mechanical considerations or requirements.

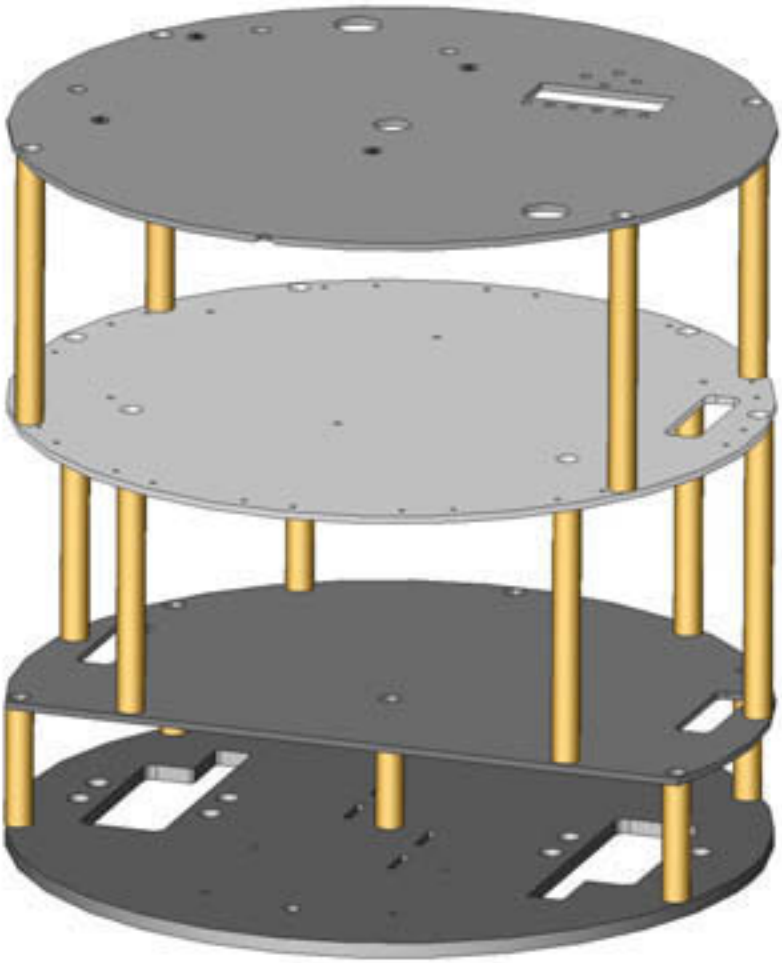


Figure 4.1: Carrying frame

The construction of the robotic platform involved many trivial yet tedious construction and assembly tasks that are not of much interest and will not be discussed here.

## 4.1 Motor Compartment

The motor compartment has a height of 65 mm. It houses the motors and the reduction gears, the wheels, the motor controllers and part of the kicker mechanism. The front 1/3 part of this compartment is not separated from the corresponding part of the battery compartment, so that the kicker mechanism can occupy both of them. The available free footprint on the base plate after placing the motors is quite limited, so the space the kicker mechanism occupies will have to extend in the vertical direction.

### 4.1.1 Wheels, Transmission and Motors

Two 35 mm in diameter caster wheels are positioned under the front and the back part of the base aluminium plate. The drive wheels are 80 mm in diameter and are also mounted on the bottom of the lower plate, but they extend inside the motor compartment through cutouts in the base plate. A 75 tooth gear is attached to each drive wheel, which in turn is coupled to a 45 tooth gear attached to the motor shaft, thus giving a reduction ratio of 1/0.6. Both gears are made of ertalon polyamide plastic to reduce the noise produced when spinning. The motors used are high quality DC ones with

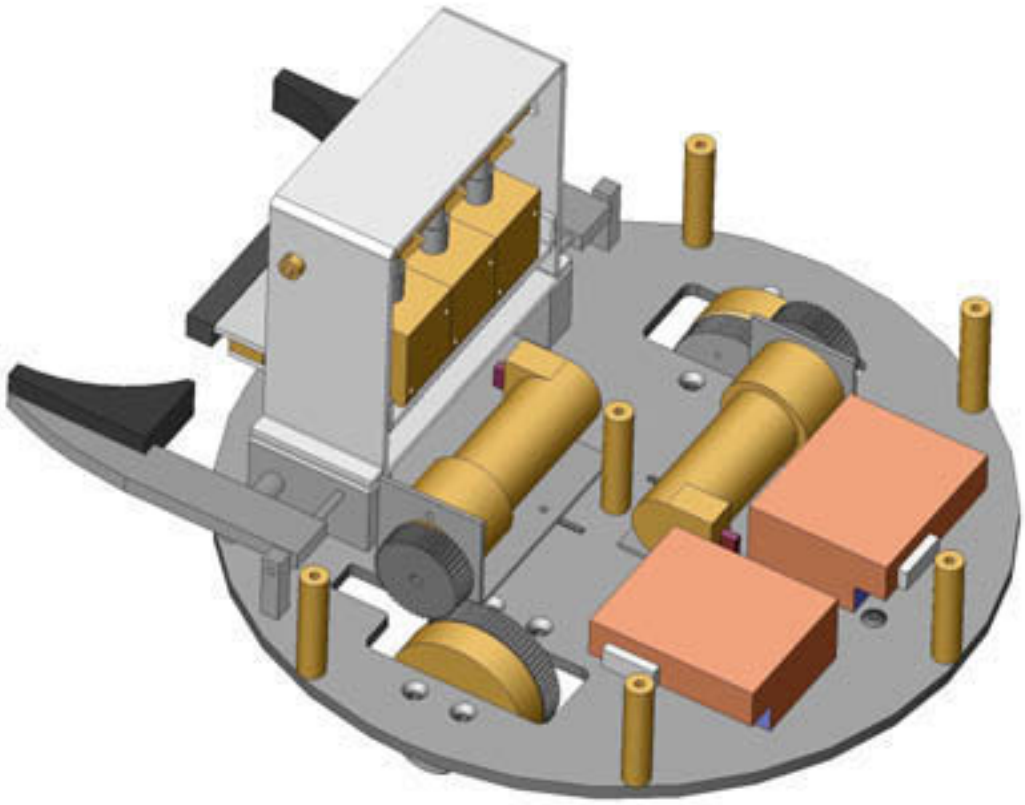


Figure 4.2: Motor compartment

built-in reduction gears and 500 CPR (counts per rotation) optical quadrature phase encoders. The built-in gears give a 5.9/1 reduction ratio, which combined with the external reduction gears achieve a total of 9.83/1. This brings the theoretical maximum resolution of the optical encoders to

$$\frac{2\pi}{500}\frac{1}{9.83} = 0.00127 \text{ rad or } 0.0732^\circ$$

of revolution of the drive wheel. Taking into account the diameter of the wheels we can easily calculate that given zero mechanical tolerances, this configuration will have

a minimum odometry uncertainty of 0.0511 mm due to the quantisation imposed by the optical encoders. Of course the mechanical tolerances will be far from zero and the errors introduced by them will be much greater than the quantisation errors, but clearly this is a matter of construction precision and not of design. With a little care in construction we can confine the odometry uncertainty in the millimetre or even sub-millimetre range.

These calculations are carried out without taking into account wheel slippage and calculation approximations which are unavoidable in computers and microcontrollers. They indicate the accuracy with which the encoder can determine the revolution of the wheel it monitors, that is the one-dimensional odometry uncertainty. The two dimensional odometry uncertainty is determined by all the aforementioned factors plus some more which is not our purpose to discuss here.

## 4.2 Battery Compartment

The battery compartment has a height of 160 mm. It accommodates the following peripherals:

- Battery
- Computer power supply
- IR controller
- IR sensors
- Kicker controller
- Charging socket
- Fuses
- Wide angle camera
- Part of the kicking mechanism

The only peripheral that demanded some effort in mechanical design is the ball handling device. The other peripherals required only to be appropriately mounted on the frame. However, mounting should be as firm as possible, yet the peripherals should be easily removable in case of failures. Their positions have been chosen either to achieve an even weight distribution and/or to allow easy access and convenient/short cable connections to them.

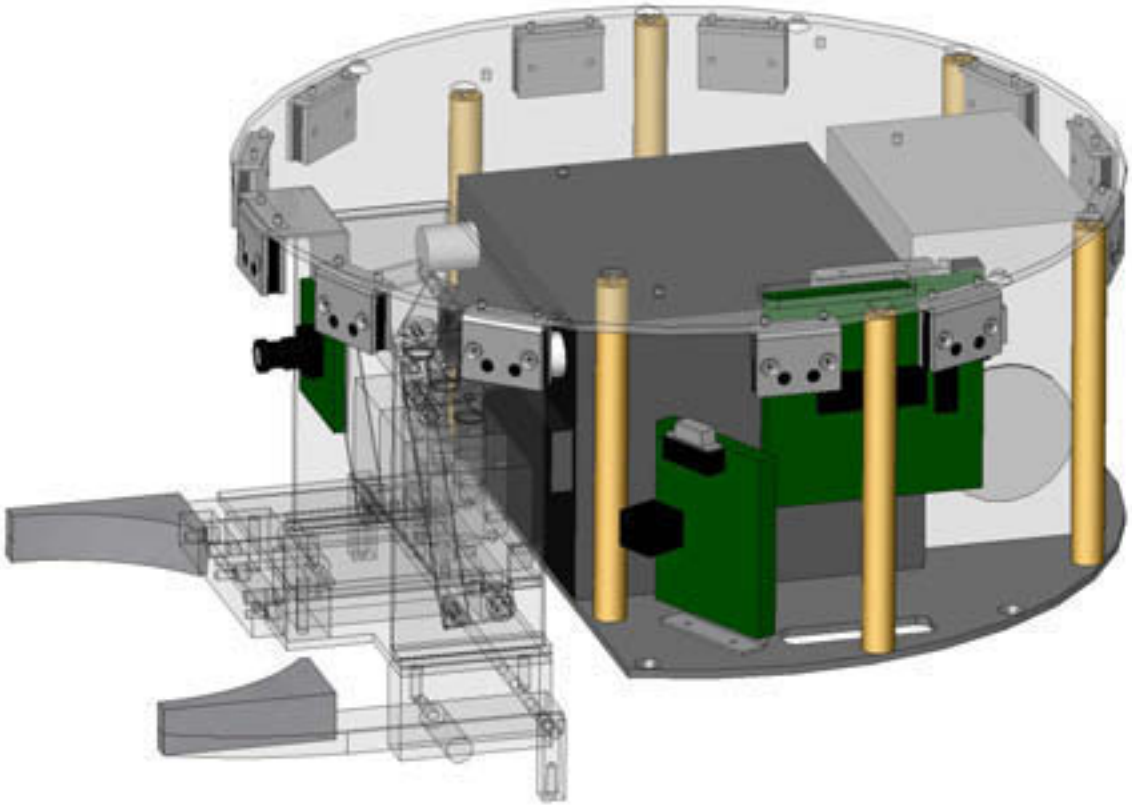


Figure 4.3: Battery compartment

### 4.2.1 Ball Handling Device

The ball handling device provides the robotic platform with the ability to confine and kick the ball. Two servo-actuated retractable arms are used to hold it, while kicking is carried out by a piston driven by 3 solenoid magnets.

The retractable arms are very simple in function and construction. Due to the constrains imposed by the RoboCup competition rules, the arms must have a very limited size, so our primary concern is their shape and the speed of their movement. The curvature of the surfaces that comes in contact with the ball is defined by that of the official FIFA size 5 ball specifications, i.e. an arc of a 22 cm in diameter circle. The distance of the arms to each other is adjusted so that the ball is restrained to an area of $\pm 1$ cm from the centre of the kicking mechanism. Urethane foam is used on the contact points to prevent the ball from bouncing too much when the robot moves or stops it. Both arms are connected to the same axis, which is rotated by a medium size heavy-duty servo through a lever. The servo chosen is able to withstand high torques due to the use of metal gears and is strong enough to fully retract or extend the ball handling arms in less than a second. Although lifting or lowering the arms requires relatively small force under normal conditions, high torques are exerted upon the servo

due to the frequent shocks the platform, and especially the handling mechanism, are subject to. The leverage used is dictated by the space available and the maximum useful rotation range of the servo, and is about 2 : 1, i.e. 90° rotation of the arm axis for 180° rotation of the servo axis.

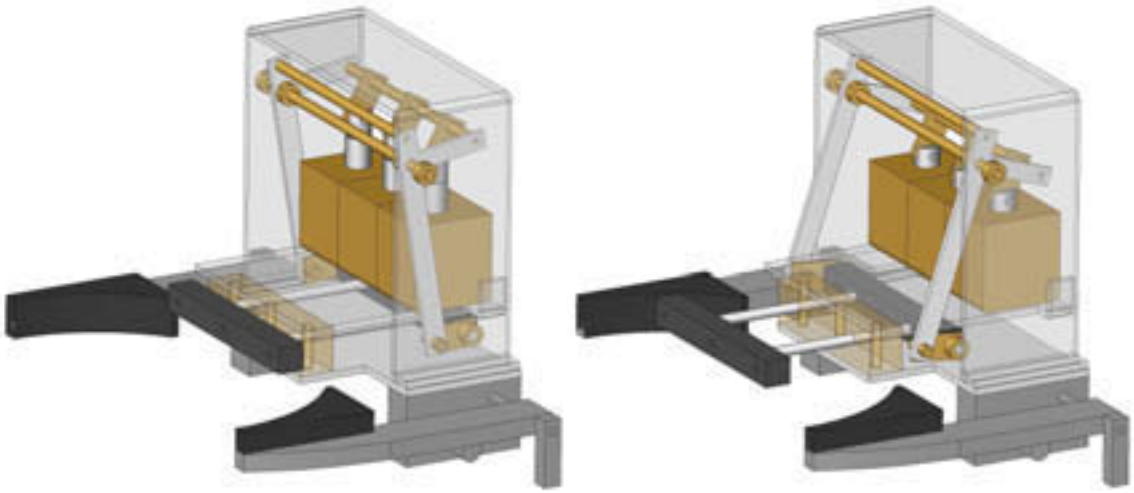


Figure 4.4: Kicking mechanism operation

The design of the kicker mechanism has to overcome many difficulties. The maximum amount of energy available must be transferred to the ball, so much effort has been spent to minimise losses due to friction and gravity. The piston that comes in contact with the ball and the plungers of the solenoids are the heaviest moving parts of the construction. The former is moving parallel to the ground so that the gravity doesn't hinder its motion to either direction. Simpler pendulum-like constructions that have been tested and also used by other teams in the competition suffer more or less from this problem. To minimise the friction applied to the piston, firstly its construction must be precise, but much more importantly the forces exerted upon it must be on the direction of its free motion. However we cannot avoid forces perpendicular to this direction. We have a very limited space for the footprint of the mechanism, so it is not possible to transfer the pulling force of the solenoids without the use of levers. Moreover, some leverage is demanded because of the very small stroke length of the solenoids. We have chosen ones with the longest stroke commonly available, yet this is only about 20 mm. Without leverage, we are obliged to use a heavy piston that will transfer its kinetic energy to the ball by elastic impact. If the ball is even a few millimetres closer or further than the optimum point of impact, which is almost certain in real conditions, this design will not perform well. By extending the piston path length so that apart from hitting it can also push , the ball doesn't need to be so precisely positioned prior to a kick.

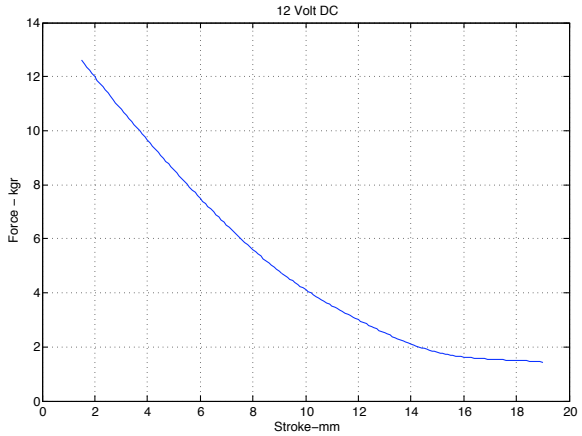



Figure 4.5: Combined force plot of solenoids

We have chosen a leverage ratio of about 1:2.75, which translates to roughly 55 mm piston movement for the 20 mm plunger stroke. Of all the ratios that have been tried out, this seemed to perform quite well. Our next concern is to minimise the friction caused by the forces which are perpendicular to the motion of the piston and are unavoidable due to the use of levers. We have tried to keep the lever configuration simple by using only one. However this means that the two points of application of forces, the ones exerted by the solenoids and the piston, follow circular paths. To achieve the maximum efficiency and minimise the friction on the joints and the piston rods, much effort has been spent to keep the force vectors for the greatest part of the path as close to the tangent of the circles as possible, thus approximating the maximum torque and energy transfer.

Each solenoid has an average pulling force over its stroke range of about 20 N with a peak of 40 N and consumes 48 Watts. They are operated four times beyond their nominal specifications, so they have a maximum duty cycle of 25%, which must be ensured by the kicker controllers. We are using 3 solenoids in parallel, thus achieving a peak force of 120 N and an average power consumption of 150 Watts. Figure 4.5 shows the combined force curve of the three solenoids over their stroke range. The same curve, multiplied at each point by a factor of $0.8 \cdot \frac{1}{2.75} - \frac{1}{2.75} \Rightarrow 0.29 - 0.36$ as indicated by the leverage ratio and the varying efficiency coefficient. The efficiency coefficient is defined by the friction and the deviation of the applied forces from the tangent. The exact modelling is not difficult, yet out of our scope.

The solenoids can only extend the kicker piston. For retraction a low tension rubber band is used. The mechanism moves very easily, so very little tension is needed to retract it, and compared to the the summed force of the solenoids it is almost a negligible loss during the extension of the piston.

### 4.2.2 IR Sensors

Figure 4.6: IR sensor

The robotic platform is equipped with a belt of 12 IR sensors comprised of separate commercial emitters and detectors in custom made enclosures. The belt lies at a height of 26 cm from the ground, slightly above the highest point of the ball (FIFA official size 5), since in the RoboCup competition it would often be mistakenly detected as an obstacle. The sensors are radially mounted on the top plate of the battery compartment and are not evenly distributed along its perimeter . Their formation is denser at the front part of the robot, namely 5 of them, while 2 are placed at either side and 3 facing at the back. Figure 4.7 shows the exact distribution of the sensors.

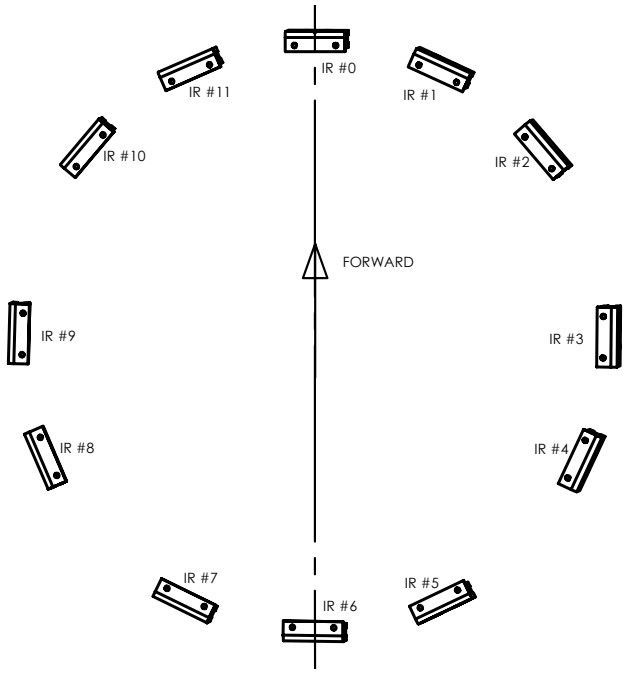



Figure 4.7: IR sensors distribution

The sensors are numbered from 0 to 11, starting from the frontal one and in a clockwise succession as viewed from the top.

## 4.3 Computer Compartment

The computer compartment has a height of 140 mm and houses the DC/DC converter for the laser measuring device, the console & display controller and the computer along with all its peripherals.

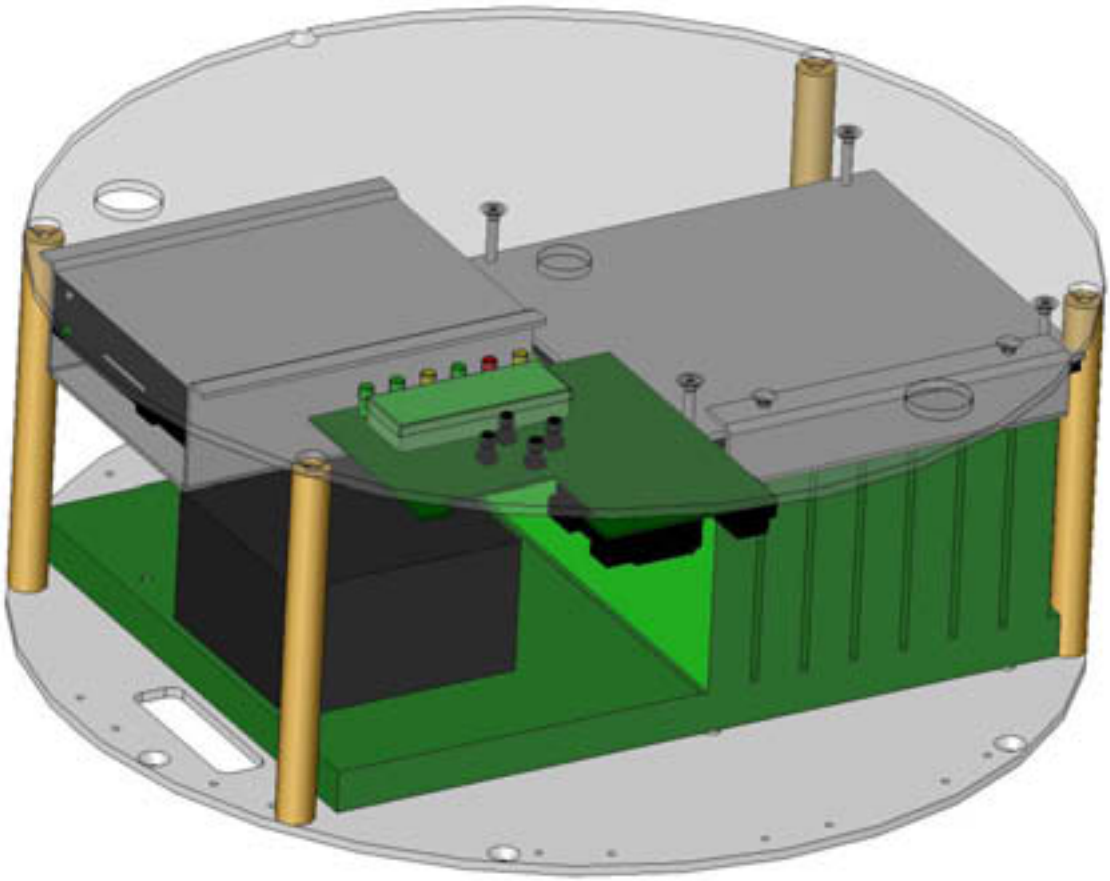


Figure 4.8: Computer compartment

There are hardly any mechanical challenges in this compartment. The only thing to be taken into consideration is to ensure that none of sensitive computer components can be displaced or damaged by the frequent shocks the platform will be subject to. At the same time the various ports of the computer must be easily accessible.

The motherboard used is a common full size ATX one. Any smaller form-factor either lacks processing power or expansion slots. We need both and the gain in space and even the lower power consumption are not significant to justify alternative choices. The computer compartment will be anyway dedicated and not too crowded, and the battery is big enough to withstand 1-2 Amps more consumption in exchange for the higher CPU power. The main problem caused by using a common desktop motherboard for a mobile platform is the risk of displacement of the expansion cards due to

the frequent shocks. Another option would be to use a more solid industrial form–factor like PC-104, yet this would seriously limit the range of available peripherals and increase their cost. We tried to decrease the risk of displacement by carefully securing all PCI cards and by adding a spring loaded press-plate on top of them. The plate provides constant pressure to all cards and seems to work satisfactorily, until of course proven otherwise. Anyway, during all the tests conducted on the robotic platform, displacement of the cards was the only problem that wasn't encountered at all. Finally, the most sensitive peripheral of the computer, namely the hard disk, had also to be protected from vibrations, so it is mounted using rubber shock absorbents.

## 4.4 Top Outer Compartment

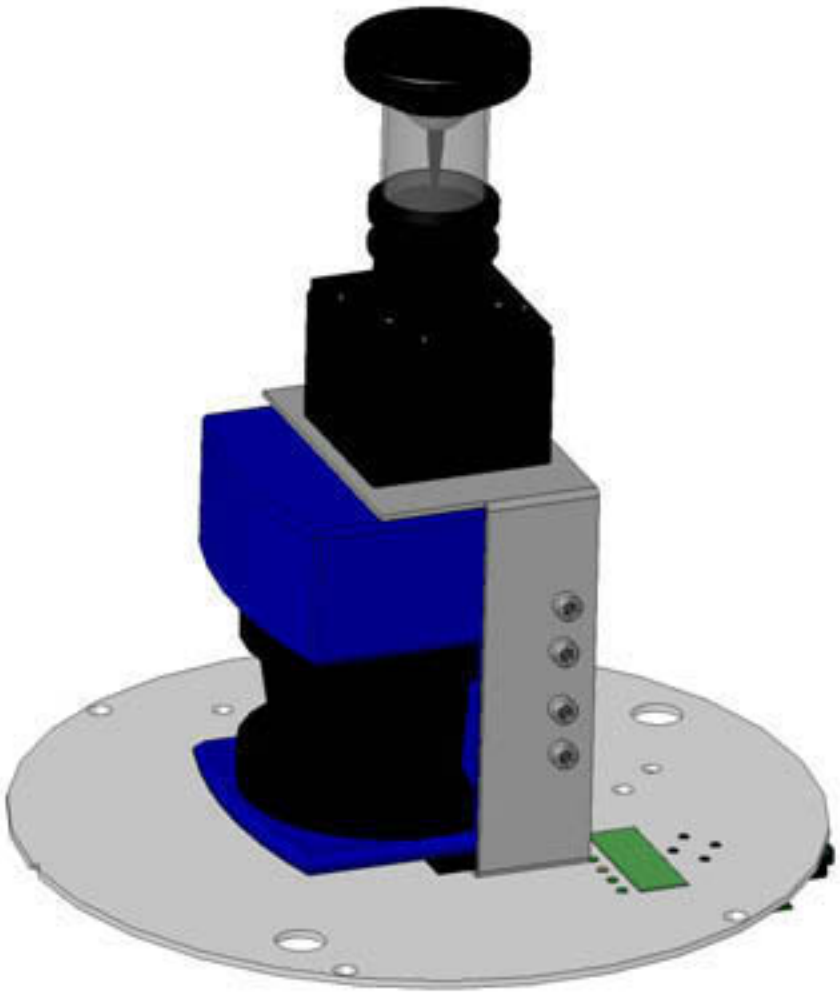


Figure 4.9: Top outer compartment

The top outer compartment is not strictly a compartment since it only has bottom, defined by the top cover of the robotic platform. Yet, the space above the cover is

occupied by some of the most important components of the platform and for reasons of uniformity we use this definition.

The outer compartment is occupied by the laser measurement system, the panoramic camera, the function display and a 4-button console. The laser is housed inside a simple yet firm aluminium superstructure and can be very easily removed. A partially custom-built panoramic camera is placed on top of the structure at the exact centre of the robot. The camera is mounted in such a way that it doesn't block the removal of the laser and can remain in place supported by the superstructure with or without it.

Figure 4.10: Panoramic Camera

FireWire panoramic cameras are still both uncommon and expensive. So we decided to partially construct our own, which consists of two parts. The upper one is taken from a commercially available panoramic camera containing a parabolic mirror and an aperture ring inside a cylinder of protective glass. A non-reflective plastic cone was additionally fitted to the mirror, which effectively decreases unwanted reflections from the protective glass. The lower part is a custom made lightproof shell containing

a FireWire board camera fitted with an appropriate lens, i.e. a narrow angle one (tele) with 8 mm focal length.

On the top cover of the robot and towards the back and left is the multifunction console and its display. The $2 \times 16$ LCD in combination with six LEDs display the most essential information about the state of the robot, like the battery charge and errors . Four buttons, which is the minimum for relatively easy selections, are used to navigate through a few menus and select basic functions, like turning the computer on and off or activating the alternative movement options like joystick and RF remote control.

"There is nothing like the smell of napalm in the morning"

APOCALYPSE NOW

# Chapter 5

# Electrical and Electronic Engineering

Apart from the main computer, the cameras and the laser measurement system every other module and part was built from scratch. This demanded great effort in electrical, digital and power electronic engineering. A great many deal of decisions had to be taken and they were not always the best possible, yet for each one of them this was at least the purpose. The construction of the modules involved the evaluation of many and sometimes contradicting parameters, so the decisions had to be based on different grounds. For the components some building blocks were specifically purchased for this application and were chosen among many as the most suitable for the application, according to one or more[1] of the architectural guidelines stated in Part I . Others were just the only ones available and the design had to be adjusted accordingly, however no effort was spared to try and make a cheap or unsuitable component work reliably and not endanger or limit the capabilities of the platform.

## 5.1 Electrical specifications

The power source of the robotic platform is a 12 V 33 AH Absorbed Glass Mat non-spillable lead/acid battery. This type of battery is very durable and resistant to mishandling. It can also be charged very quickly, and though this limits its capacity and its life expectedness, it is a very useful feature in certain applications and circumstances. The

[1] or less

charging circuit is very simple and we rely heavily on the charger to avoid damage to the battery due to overvoltage. The only means of protection incorporated into the robotic platform is a high current diode to avoid damage caused by reverse polarities and a melting fuse rated at 30 Amps. When the battery is fully discharged and we are forcing a rapid recharge without limitation, the charging current can easily reach 25 Amps. The charging voltage is 14.3–14.5 Volts and not the usual to lead/acid type batteries 13.6 Volts, in order to account for the voltage drop on the safety diode.

### 5.1.1 Power Consumption

The constant power consumption of the platform when not moving and the computer and laser measurement system are not in operation is about 0.5 Amps. This only covers the steady state operation of the modules connected to the CAN bus, excluding the computer. These modules form a subsystem that can operate independently of the computer and the vision and laser modules, e.g. when the robot is remotely controlled by RF. When the robotic platform is moving, only the motion controllers increase their consumption. Each motor can take up to 16 Amps in stall conditions, however the controllers limit the current at 10 Amps. Thus the maximum current drawn by the motion module is 20 Amps. This is reached only on extreme motion changes, like in case of a sudden stop after full speed. Under normal conditions and at low speeds, the motors seldom consume more than 3 Amps each.

The next considerable consumer is the kicker mechanism. Its 3 solenoids when used at full power draw 12 Amps. However they operate at very short bursts and their current peaks are smoothed by the capacitance of the power bus.

On all normal operating conditions of course the computer will be also operational and consuming power. It draws about 5 Amperes minimum and reaches 6+ for very short and seldom periods of time. However this can be reduced if the power saving functions of the software and hardware are exploited, although not very much . The computer gets the right voltages by a self protected DC ATX power supply.

The last almost constant load to the power source is the laser measurement system which draws 2 Amps without any peaks. Evidently that these 2 Amps can be completely eliminated by applications that don't use this sensor. The laser, as most of its industrial type, operates at 24 Volts, so the appropriate DC/DC converter is used to power it.

The vision system takes its power from the FireWire bus, which means that each new camera in the chain adds to the consumption of the computer, yet it is not worth mentioning for calculation purposes. All the other modules add up to a minimum of 5.5 Amps and a highly improbable maximum of 41 Amps. To be on the safe side, all modules are protected by a single 25 Amp melting fuse. The computer power supply

is self protected and the motion controllers sense the current they draw and try to limit it, although it wouldn't be too hard or useless for them to also have additional melting fuses. In any case, the single 25 Amp fuse has proven to be sufficient and enough in all operating conditions tested and provided to the robotic platform the necessary protection during all accidents in its construction.[2]

### 5.1.2 Connections

Quite a few cables run throughout the frame between the components of the robotic platform. They are of 6 kinds

- Power cables
- Ribbon cables
- FireWire cables
- RS-232 serial cables
- CAN bus cables
- $I^2C$ bus cable

The miscellaneous power cables that are present everywhere and the ribbon cables which are used to connect the computer to its peripherals and the console controller are trivial, as long as the current ratings are met and, in the few cases that this is essential like in the UDMA interface of the hard disk, also the shielding ones. The FireWire cables are also standard and there are no special considerations about them since the length of the FireWire chain is well inside their specifications, even after the addition of several cameras. Serial connections are used between the computer on the one side and the laser measuring system. Here also no special provisions need to be made, however just to be on the safe side standard shielded cables are used though not necessary, since the baud rates used are quite low.

Only the $I^2C$ and the CAN bus cables had to be constructed. For both protocols there is no industrial standard for them or their connectors. The $I^2C$ bus is used only between the motor controllers and the kicker controller which are placed side by side and it is very short, however the motor compartment is an environment of great electrical noise and shielding is not an exaggerated precaution which is taken. The CAN bus connections is a little more complex case. The ISO/DIS-11898 physical layer standard, with which all transceiver chips used on the CAN bus modules are compatible, specifies that the cables used should have an impedance of 120 Ω with an allowed margin

[2] but not from the accident of its construction

of $\pm 12\,\Omega$. Yet cables with that specific impedance are not common. We have used CAT 5 ethernet cable, which is sufficiently shielded, uses distinct twisted pairs appropriate for the differential signal of the CAN bus and has an impedance of 100 Ω, which is not far from the suggested minimum of 108 Ω. The ISO/DIS-11898 standard does not give any recommendation about the connectors to be used in the CAN bus chain. We have chosen the common 9-pin D-Sub connectors also used in the RS-232 protocol and the pin layout recommended for it by CiA, which is pretty much the unofficial industrial standard.

## 5.2 Kicker Controller

The kicker controller is controlling the ball restraining arms and the kicker mechanism. Its inputs are two on/off IR sensors mounted on the arms that detect the presence of an object between them and one distance measuring IR sensor pointing at the front, which estimates the distance to the ball, if present. Its outputs are two PWM (Pulse Width Modulated) signals, one to drive the servo actuating them and the other to fire the solenoids connected to the kicker piston. The second output is rated at high currents, so the appropriate power electronics are implemented.

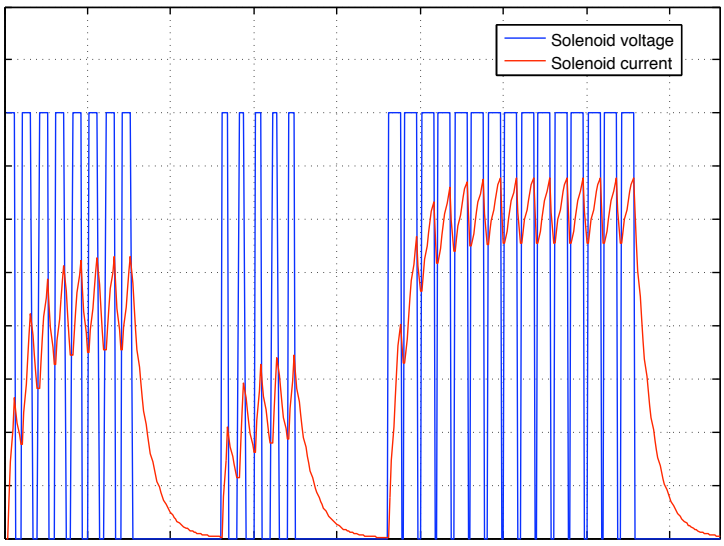



Figure 5.1: Kicker PWM

The ball is kicked with a user specified strength and duration. The solenoids are fed with PWM current of 1-100% dutycycle for 1-1000 ms. The dutycycle sets the momentary power of the kick, and combined with the duration they both approximately set the energy that is transferred to the ball. Figure 5.1 shows the voltage/current relationship for 3 combinations of PWM dutycycle and duration.

### 5.2.1 Hardware

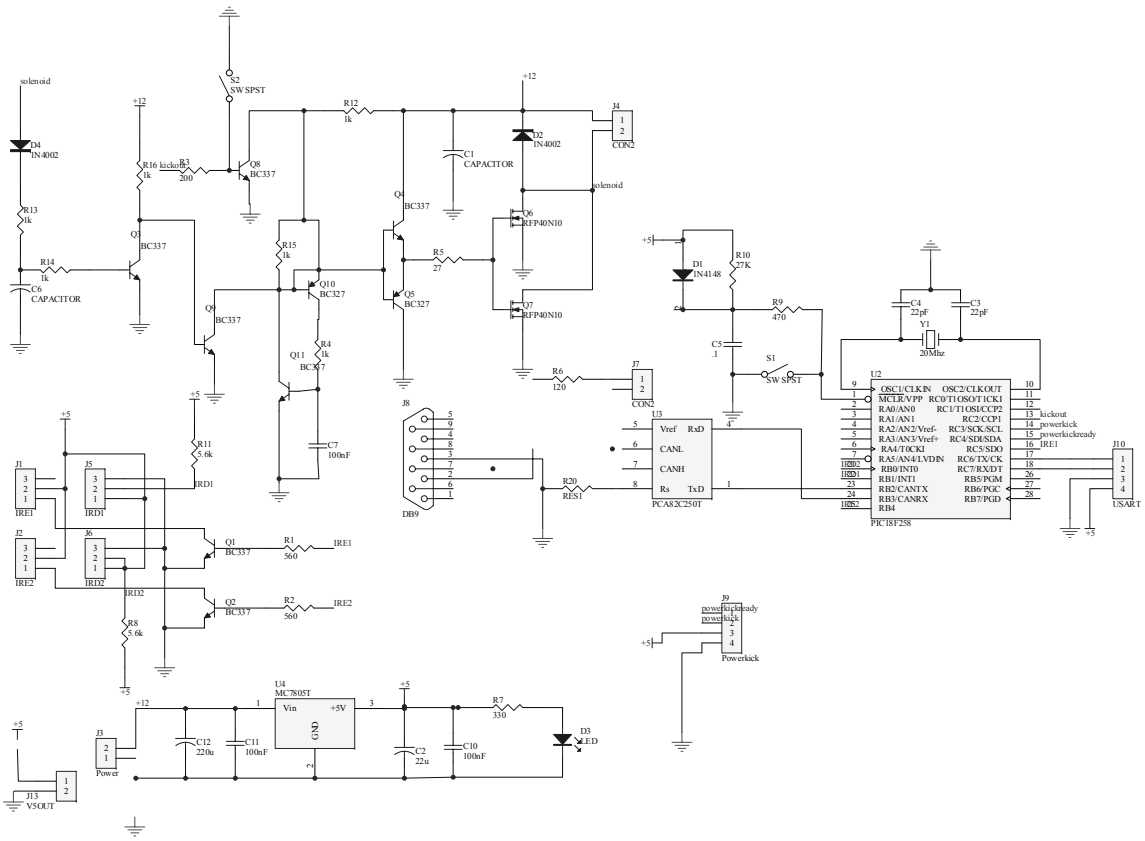



Figure 5.2: Kicker controller schematic

The kicker module is built around a PIC 18F258 microcontroller operating at 40 MHz. The microcontroller is responding to commands from the CAN bus and an asynchronous serial port. This chip has a built-in CAN protocol peripheral and is interfaced to the physical CAN bus through a PCA82C250 chip that provides the appropriate voltage levels and protection to make it ISO/DIS-11898 compatible. The microcontroller and the physical interface chip achieves CAN bus of up to 1 Mbps, however they are operated at 500 kbps. The serial port input is however not RS-232 compliant, since it outputs pulses of 0-5 Volts, so it cannot be directly connected to a computer. It is intended primarily for debugging purposes and an additional daughterboard containing a MAX232 interface chip is attached to the controller to provide the $\pm$12 Volts needed for RS-232 communication. The serial port speed is set to 115200 kbps.

The current to the solenoids is controlled by two RFP40N10 FETs in parallel, which can together withstand up to 20 Amps. Since the solenoids are operated at two times their nominal voltage specifications, it is important to keep their dutycycle below 25%, i.e. 1/4 of the time on and 3/4 off to avoid damage due to heating. This is done in software, however an additional electronic circuit is implemented for protection. This

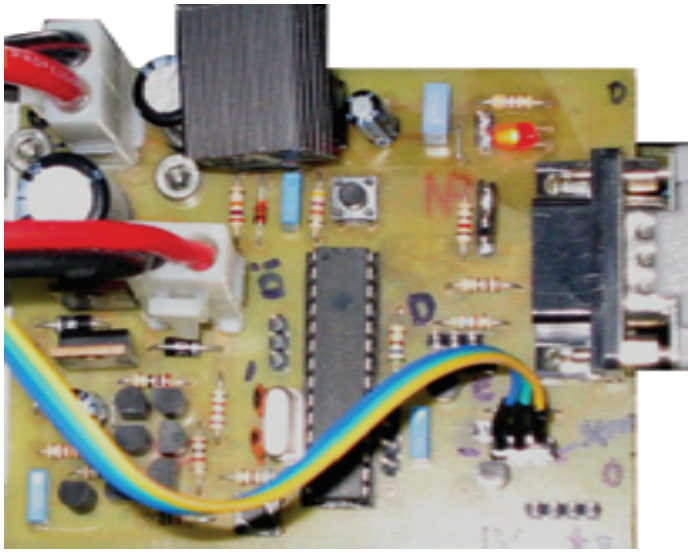


Figure 5.3: Kicker controller

circuit is fully analog and thus not subject to programming errors and it limits the duration of the pulses coming from the microcontroller to 1.5 seconds. This is to ensure that there is no way the solenoids are left on continuously by an error in the firmware or the microcontroller.

The module is powered directly from the battery at 12 Volts. The microcontroller, the interface chips and the IR sensors operate at 5 Volts and are all driven by a 78S05 voltage regulator, while the solenoids and the protection circuit operate unregulated at 12 Volts. The board has high current sockets for the 12 Volt power input and the connection to the solenoids. Low current headers are used for the IR emitters, the IR receivers and the serial port, and a male 9-pin D-Sub connector interfaces the module to the CAN bus.

### 5.2.2 Firmware

The controlling firmware receives and evaluates commands from either the CAN or the $I^2C$ bus. Its scheme of operation is strictly passive. The controller is always idle and acts or responds only on demand. Its functions are to:

- Report the state of the ball sensing IRs
- Report distance to ball if present
- Report error between desired and actual ball distance. This error is multiplied
- Move the arm up/down or at an intermediate position

- Kick the ball

The controller can be optionally connected to the $I^2C$ bus of the motion module. It then reports at 40 ms intervals the error between the desired and actual ball distance multiplied by a user defined coefficient. This quantity is used by the PID algorithm of the controllers to adjust the current speed of the robotic platform so as to keep the ball rippling a few millimetres in front of the robot, between the handling arms. This is done on demand and after the controller has verified that the handling arms are lowered and the IR sensors indicate the presence of the ball. This feature has not been thoroughly tested, however it seems to work satisfactorily at low speeds, both angular and tangential.

## 5.3 Motion Controller

The motion controller is the most complex and crucial component that has been designed and implemented completely from scratch for the robotic platform. It controls the motion and the odometry of the robot, and its complexity is imposed by three factors:

- Both odometry and trajectory following must be extremely precise, which is a prerequisite for a research robot.
- Motion is the only active function, that if it is malfunctioning or misused can physically damage the platform or the operating environment. Thus stability and proper error handling are critical issues.
- The controller will have to safely handle high currents. This places high demands on the design and implementation of the power electronics part.
- The control algorithm and firmware should protect the hardware from high stress, both electronic and mechanical, even when the received commands request it to do so. This is desirable only when handling regular motion commands, in order to protect the platform from user inflicted errors. In emergency cases, such as an immediate stop, this behaviour should be overriden.

Although the motion controller is viewed and handled by the applications as a single module, it consists of a separate controller for each motor. This is for reasons of easy maintenance only and imposes further complexity. Since power electronics are included, hardware failures must be seriously taken into account, usually happening to one motor each time. Both motor controllers are identical in hardware and firmware

and are designed as standalone blocks that when connected together perform different functions, forming a master/slave pair specifically fitted to control a differential drive. The firmware of each one implements all single, master and slave mode functions, thus making a field exchange very easy by having only one type of controller that can substitute the failed one regardless of its type. Although it would be easier to design identical single mode controllers, this would mean that the controlling software, i.e. the motion server, should implement many synchronisation functions needed for the precise operation of a differential drive. In order to off-load the computer from these trivial yet CPU-intensive tasks, the master/slave configuration seems quite appropriate, despite its programming complexity.

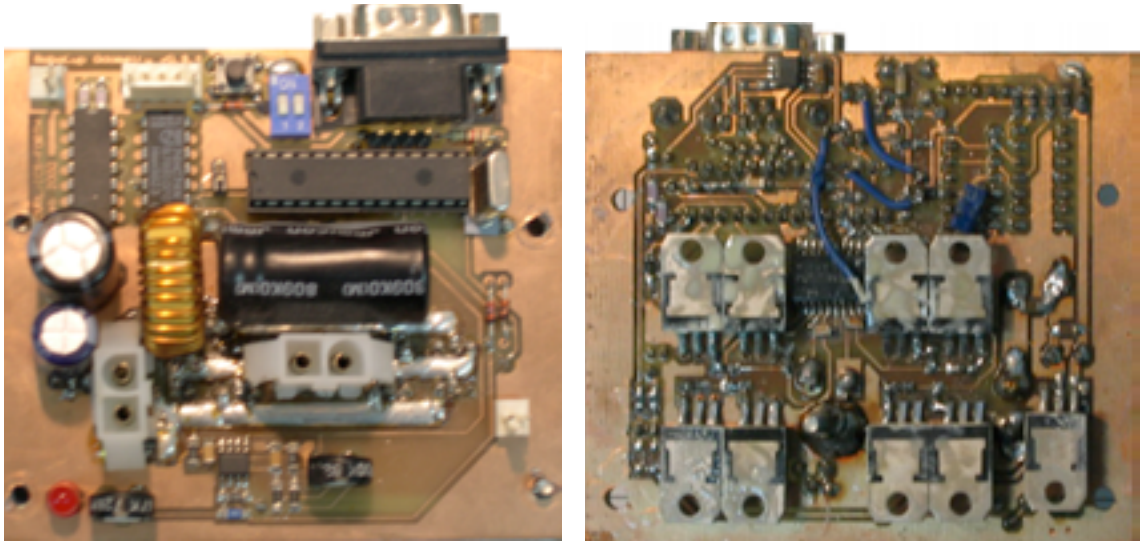


Figure 5.4: Motor controller

### 5.3.1 Power Electronics

A low power loss and able to handle high currents H-bridge circuit is necessary to drive each motor, which can draw up to 16 Amps when stalled. Although the motors should never be allowed to reach the stalling current, we need an H-bridge that can easily provide it in order to keep the delicate power electronics in a safe operating area. In our design we have used pairs of TO-220 packaged n-channel FETs with built-in surge diodes in parallel for every leg of the H-bridge, each of them taking up to 10 Amps@12 Volt continuous load and 60 Amp peaks. Thus, we have a nominal maximum rating of about 20 Amps continuous current, giving us a reasonable safety margin.

By using n-channel FETs for both high and low sides of the bridge, we need a driving circuit capable of providing the appropriate voltage to the high side ones. The TD-340 is an integrated component with built-in bootstrapping and protection circuits

specifically fitted to the demands of medium power H-bridges. It is driven by a PWM signal, or more specifically it buffers and feeds it to the FETs, switching them on and off at the right moments and introducing the appropriate delays to avoid surge currents. Although the FETs have also reverse diodes that provide an outlet for the "free wheeling" current that is induced by the motor when the FETs are switched off, this approach leads to great heat dissipation by the diodes and in turn by the FET packages. The particular driver deviates from the traditional and simple H-bridge operation and synchronises the high-side FETs, which have a very low ohmic resistance, to switch at the appropriate times and substitute the reverse diodes, thus resulting in a much lower heat dissipation.

The H-bridge driver can buffer PWM signals of up to 25 kHz. The values of the peripheral components are chosen so that the driver and bridge circuits must be tuned to a certain frequency, taking into account the following facts:

- Frequencies between 20 Hz and 18 kHz may produce audible screaming from the speed controller and motors, which may be an added attraction to the robot, yet a little unpleasant.
- RF interference emitted by the circuit will be worse, the higher the switching frequency is.
- Each switching on and off of the speed controller FETs results in a little power loss. Therefore the more time spent in switching compared with the static on and off times, the more will be the resulting switching loss in the FETs.
- The higher the switching frequency, the more stable is the current waveform in the motors. This waveform will be a spiky switching waveform at low frequencies, but at high frequencies the inductance of the motor will smooth this out to an average DC current level proportional to the PWM demand.

Figure 5.5 shows the equivalent circuit of the motor, and the current waveform as the PWM signal switches on and off. This shows the worst case, at 50:50 PWM ratio, and the current rise is shown for a stationary or stalled motor, which is also worst case. Regardless of the average power consumption, great rippling in the current also means that the momentary torque of the motor ripples too, which stresses all mechanical parts involved. We calculate the lowest frequency for which the ripple of the current is less than 5 percent, which is an acceptable value.

Taking the falling edge of the current waveform, this is given by the equation

$$i = Ie^{-\frac{tR}{L}}$$

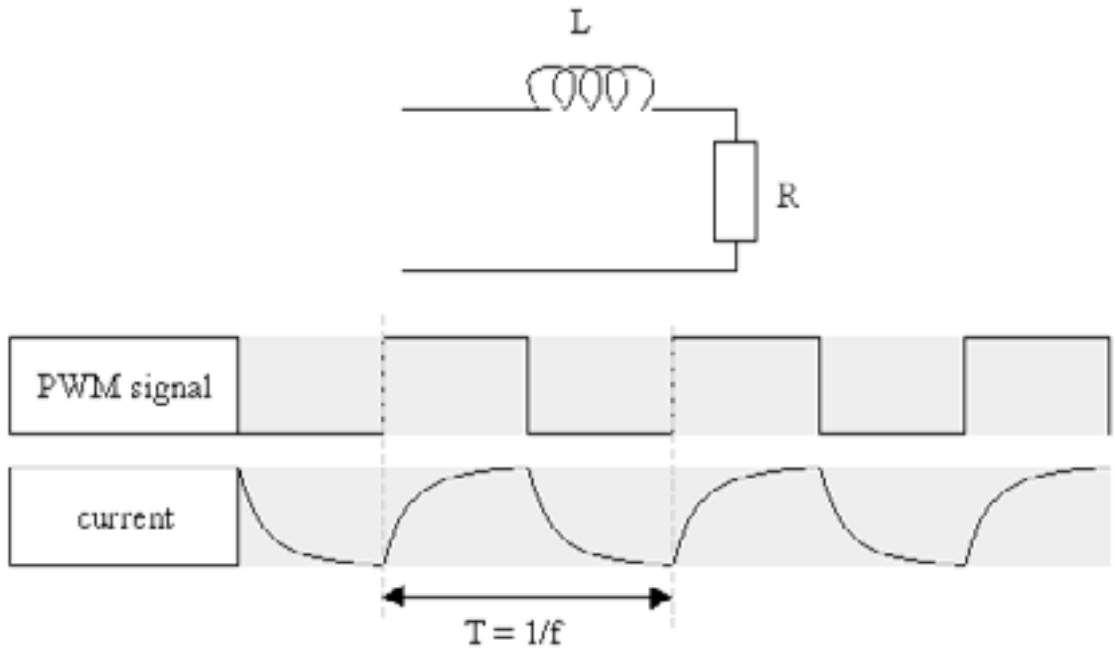



Figure 5.5: PWM driven inductive load

So the current at time t = T/2($i_1$) must be no less than P% lower than at t = 0($i_0$). This means there is a limiting condition:

$$i_1 = (1 - \frac{P}{100})i_0$$

So:

$$Ie^{-\frac{tR}{L}} = (1 - \frac{P}{100})Ie^0 \Leftrightarrow -\frac{TR}{2L} = \ln(1 - \frac{P}{100}) \Leftrightarrow T = -\frac{2L}{R}\ln(1 - \frac{P}{100})$$

,giving a frequency of

$$f = \frac{R}{-2L\ln(1 - \frac{P}{100})}$$

The datasheets of motors we use state the following values for the coil inductance and resistance: L = 0.66 mH and R = 0.71 Ω. Thus, we calculate for 5% ripple a minimum frequency of 10.486 Khz. This frequency is well into the audible area and gets quite on the nerves, so we choose a value of 17 kHz , which is the lowest one possible found after tests that doesn't produce any sound.

The load is led to ground through a shunt resistor, in order to implement a security and monitoring circuit. An op-amp buffers and amplifies the voltage drop on it, and its output is used to provide a reading about the current passing through the load, and is measured by an ADC input of the microcontroller. Another op-amp configured as a comparator set to a fixed threshold value by a trimmer is also fed by this signal, which

provides a means of safely limiting the maximum current allowed to pass through the load and FETs. The output of the op-amp is an inverted signal indicating the over-current, i.e it is high when the current is within limits and low otherwise. This signal is used by the microcontroller to trigger all actions necessary to limit the current and notify the server program. Yet, however small the delays introduced by the firmware program, one cannot rely on it to effectively protect the FETs, which can be damaged even in such a short time span. In order to overcome this risk, the overcurrent signal is also connected through reverse diodes to the input of the H-bridge driver. When the current limit is exceeded, the low signal cancels immediately the driving pulse of the microcontroller, thus cutting off the current with a delay in the order of nanoseconds.

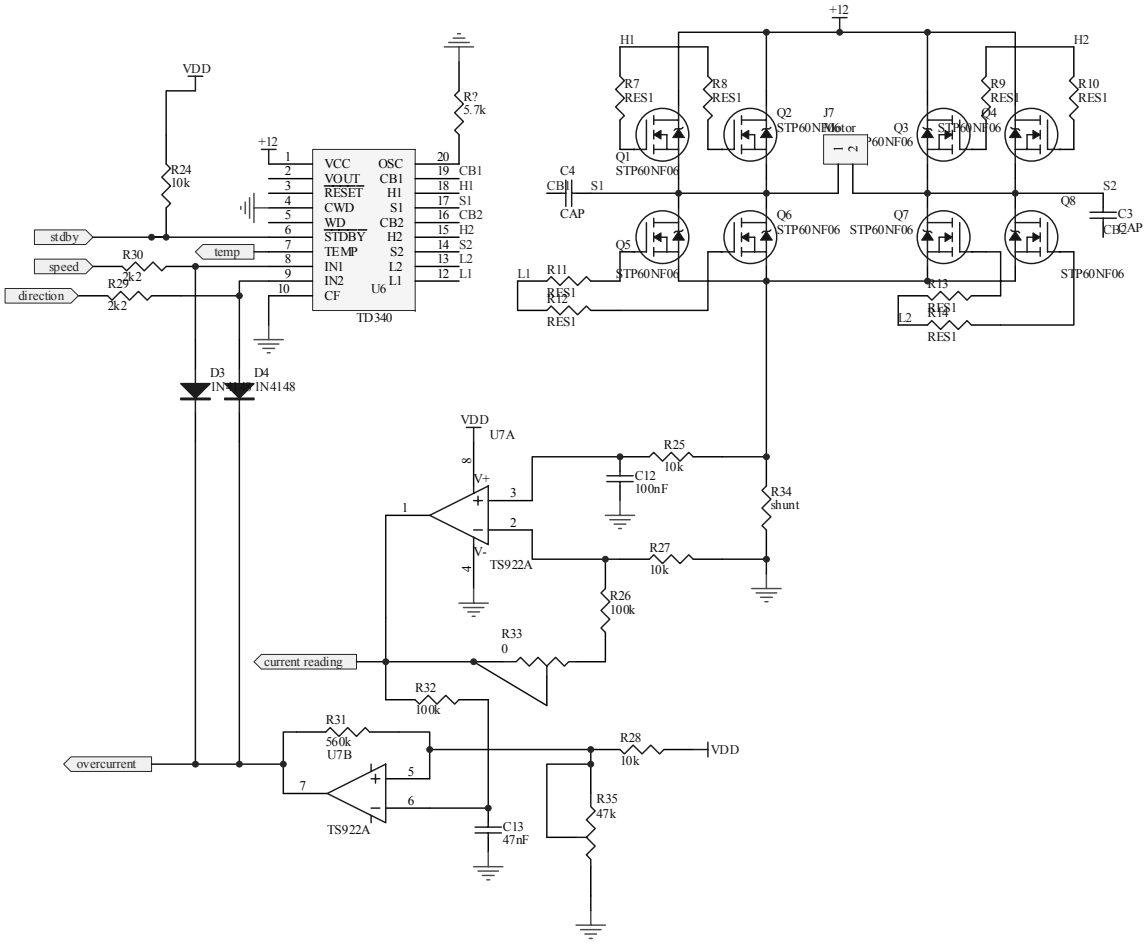



Figure 5.6: Motor controller power electronics schematic

This design has proven to be very efficient. The controllers were subject to extreme mishandling and shortcutting during their development, in order to test the behaviour of the power electronics part under heavy loads[3]. The circuit at least managed on all cases to protect both itself and the motors from permanent damage. The handling

[3] And of course due to some stupid mistakes that would make even a brave electrical engineer weep

of these error cases by the firmware and the server programs proved to be a more problematic issue.

### 5.3.2 Digital Electronics

The main component of the digital electronics part is a PIC 18F258 microcontroller operating at 40 MHz. It processes and evaluates readings or commands from:

- The quadrature phase optical encoder on the motor
- The analog current estimation and digital overcurrent signals from the power electronics part
- The fellow controller through the $I^2C$ bus
- The battery

The pulses from the optical encoder are selectively directed by two flip-flops into one of two inputs of the microcontroller, according to the direction of the motor shaft rotation. This simple circuit offloads the microcontroller from the procedure of determining the direction and provides separate trains of pulses to indicate this, which are directly usable by the firmware. One interrupt enabled input is dedicated to the overcurrent signal to achieve the fastest possible response in such an error case and regardless of the state of the firmware program execution at the moment. Finally, two ADC inputs are used to measure the current through the H-bridge, presented as voltage drop on the shunt resistor and amplified by the op-amp in the power electronics part, and the battery voltage through a simple voltage divider.

The microcontroller receives commands from one of two buses, the CAN and the $I^2C$ , according to its mode of operation. The former typically needs a media conversion circuit, which is accomplished by a PCA82C250 interface IC. The mode is selectable through a dip switch and configures the controller in firmware as master or slave. The master receives and responds to commands through the CAN bus and issues periodic or triggered ones to the slave through $I^2C$ . The slave is controlled under normal circumstances only through it, however the CAN bus functionality is still retained to report errors.

All components in the digital electronics part are operating at 5 Volts through a 78S05 voltage regulator powered by the same source as the power electronics part. According to the specifications of all ICs, the motion controllers can safely drive DC motors with nominal operating voltages of 6-30 Volts without any modifications.

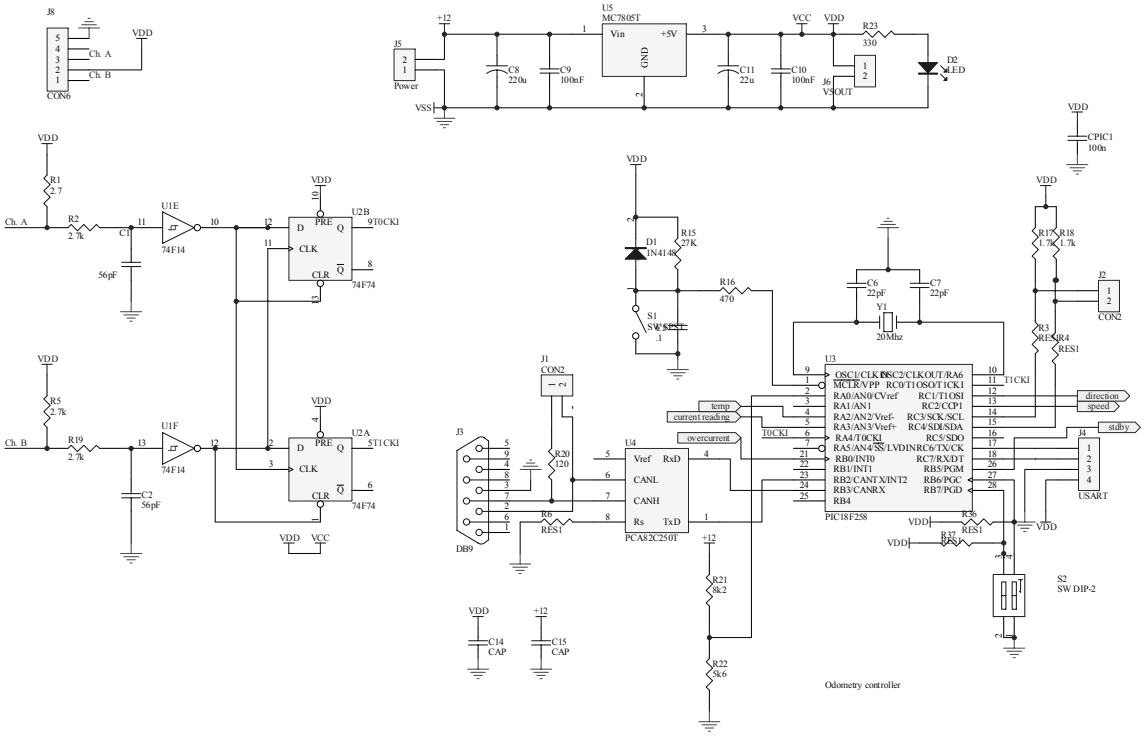



Figure 5.7: Motor controller low power and digital electronics schematic

## 5.3.3 Firmware

The firmware of the motion controllers is quite complex, since it controls very important and sensitive functions and has to be very fussy about error detection and handling. Moreover, it is the same code for both the master and the slave controller, although it has to perform many actions that are different between these two. The mode of operation of each controller is determined upon power-up by the firmware by reading the position of a DIP switch, so a single spare controller can substitute any of them in case of a failure[4].

The operations performed by the firmware are split into three priority levels:

- High interrupt level: The actions in this level are up to a few details common to both the master and the slave. They are interrupt driven and take precedence over everything else and are never masked out for more than a few operation cycles. All computations and data readings that are essential for the control of the motors are performed here. Evaluation of watchdog timers and the overcurrent signal are also carried out in the high interrupt level. All functions here are independent from the course of execution in the lower levels. For the program to be consistent and as stable as possible, all variables used in this level are either carefully buffered and can be activated by periodic events in the same level or are protected through the use of interrupt masks on each access.

[4]Field replacement they call it and it is charged extra

- Low interrupt level: The actions in this level are also interrupt driven, yet a higher level interrupt action can suspend their execution. The two main data busses on which the controllers are connected, the CAN and the $I^2C$ , are handled at this level. Packets and commands coming from them are assembled in this level and they are passed to the lower level for execution. There are significant in the actions performed by the master and the slave in this level. The CAN bus functions are the same, yet the $I^2C$ functions are completely different since the master plays an active role on it, while the slave is strictly passive.

- Normal execution level: This level is used for all not time critical functions. The evaluation and syntax check of the commands, trajectory calculations, ADC readings and error checks are performed at this level. Also, the handling of the auxiliary serial input/output which is used for debugging and as an alternative command path take place in this level. The difference between the actions performed by the master and the slave are again significant, especially as regards the evaluation of commands and errors.

General guidelines for the consistent separation of the three levels and the communication between them are followed as closely as possible but never with disregard to efficiency or safety. All actions are performed asynchronously and are protected without exceptions by watchdog routines running periodically on the high interrupt level, so as to exclude any cases of hangs.

#### 5.3.3.1 Control Algorithm

The main function of the motion controllers is to drive a DC motor with the greatest achievable accuracy. The output of the microcontroller is a variable dutycycle PWM and a direction indication signal that are fed to the H-bridge driver and its input is the reading of the quadrature encoder attached to the motor shaft. A discrete time PID algorithm is used as the mathematical model that links them. The PID (Proportional-Integral-Derivative) algorithm is the most widely used one for servo motor control. It performs exceptionally well when its is tuned correctly, however its parameters must be specifically fitted to the electromechanical characteristics of the control system and cannot be easily adjusted when these are not known. Well known experimental procedures for PID tuning are used, which require measurement equipment and are in most cases problematic, however in our application the robotic platform and the controllers incorporate all necessary facility for such a task. Routines running on the server or as applications could easily measure the characteristics of the system and adjust the motion parameters accordingly and even dynamically.

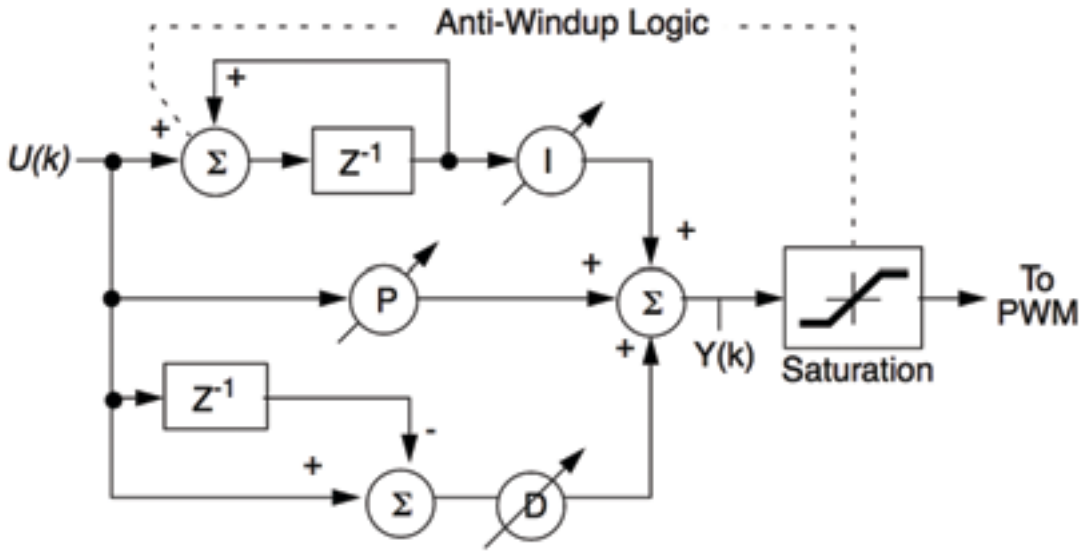



Figure 5.8: Digital PID control algorithm

Figure 5.8 shows the general PID algorithm, which is exactly as implemented in the motor controllers. In our implementation we can easily measure the rotations of the motor shaft, so the PID control receives as its input U(k) a position error, calculated as the difference between the actual and desired rotation. At each step the output is calculated as the digital analogue of the function

$$Y(t) = k_i \int_0^t U(\tau)d\tau + k_d \frac{dU(t)}{dt} + k_p U(t)$$

, whereas $k_i$,$k_d$,$k_p$ are the integral, derivative and proportional coefficients respectively, which must be specifically adjusted to the electromechanical system driven. The output signal Y(k) is a 600 step PWM pulse for both directions, suggested by the maximum resolution of the integrated peripherals of the PIC 18F258 in the specific frequency (17 kHz). The output of the PID controller is refreshed every 902 μsec, which is chosen both due to certain implementation issues and in order to achieve the maximum accuracy in the range of velocities of the specific motors.

The anti-windup logic stated in the figure is a procedure used to avoid the problem of integrator windup, which is a condition that occurs in PID controllers when a large following error is present in the system, for instance when a large step disturbance is encountered. The integrator then continually builds up during this following error condition even though the output is saturated. The integrator then ”unwinds” when the servo system reaches its final destination, causing excessive oscillation. To avoid this, in our implementation we inhibit the action of the integrator during output saturation.

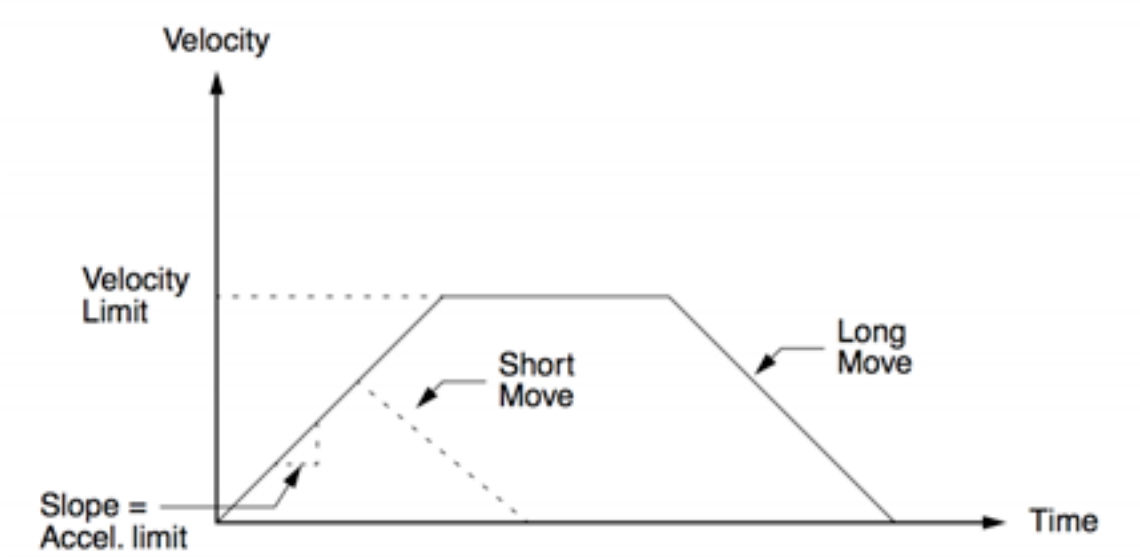



Figure 5.9: Velocity ramp segments

The error U(k) is expressed as the difference between the actual position of the motors and the ideal trajectory. In an ideal trajectory the motor should reach the desired position with a piecewise linear velocity . At each refresh instant, the velocity is incremented by a constant acceleration value until a specified maximum velocity is attained. The maximum velocity is maintained for a required amount of of time and then decremented by the same deceleration value. The velocity trajectory is thus trapezoidal for a long move and triangular for a short one, where maximum velocity is not reached. The motion follows the same constant acceleration scheme even when a new movement command is received before the last one is completed. To achieve this and maintain positioning accuracy, the instants of acceleration and deceleration are checked on each refresh, and not only at the beginning of each new trajectory.

#### 5.3.3.2 Trajectory Planning and Odometry

Each controller is very accurate at following a predefined trapezoidal velocity trajectory. When operating in a master-slave configuration they synchronise their actions and act as a single differential drive motion controller. The master controller receives and interprets the motion commands and passes them to the slave as needed. Each motion command implicitly or explicitly defines the following:

- Tangential (linear) velocity
- Angular velocity
- Acceleration

- Distance

These quantities are expressed with regard to to the footprint centre of robotic platform and correspond to different linear values for each wheel. The master controller calculates the equivalents for both itself and the slave controller and issues them to the latter. The algorithm concentrates at keeping the robot as close to the path as possible, while the desired acceleration is a secondary requisite. If it exceeds hard-coded safety limits it is accordingly adjusted.

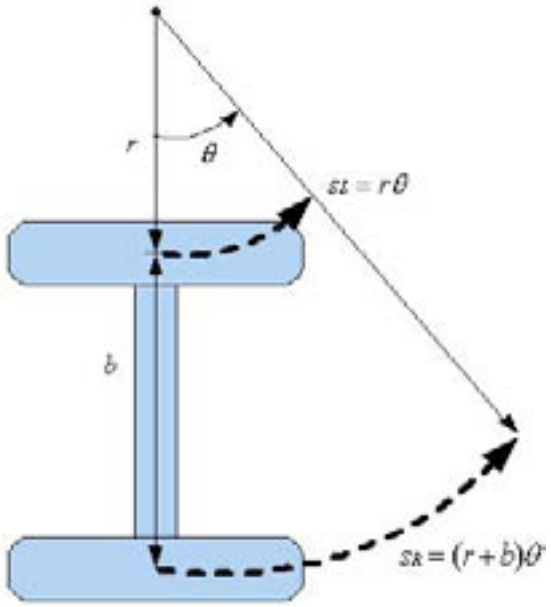



Figure 5.10: Path of wheels through a turn

Figure 5.10 shows the path of the driven wheels of the robot through a turn. One can easy see that the following equations apply:

$$v_L = r\frac{d\theta}{dt}$$

$$v_M = \left(r + \frac{b}{2}\right)\frac{d\theta}{dt}$$

$$v_R = \left(r + b\right)\frac{d\theta}{dt}$$

or simply $v_M = (v_L + v_R)/2$, where $v_L$,$v_M$ and $v_R$ are the velocities of the left wheel, the centre of the robot and the right wheel accordingly, b the wheel base and l the curvature radius. These equations are used to determine the velocity for each controller and are solved only at the start of each motion command.

Each controller is at any moment aware about the exact position of the motor it drives, yet calculating the 2-dimensional odometry is a little more complicated. The

master controller performs the necessary calculations for basic 2D odometry, since it can achieve much higher refresh rates with few resources than a higher level program. The model used assumes that between refreshes the velocities of the wheels remain constant, which is reasonable for time-spans of a few milliseconds. For our calculations we as a reference the centre point of the left wheel, which is the point where an idealized wheel makes contact with the floor. The motion of the right wheel within the frame of reference follows a circular arc with a radius corresponding to the length of the wheel base.

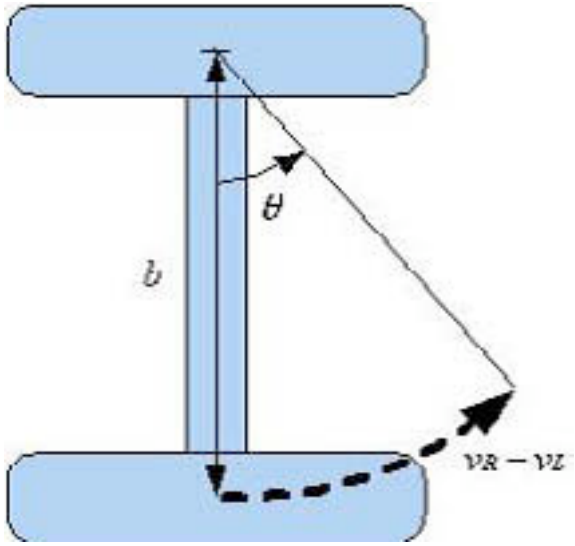



Figure 5.11: Wheels at different velocities

The momentary change in orientation as seen in Figure 5.11 is:

$$\frac{d\theta}{dt} = \frac{v_R - v_L}{b}$$

By integrating and taking the initial orientation of the robot as $\theta(0) = \theta_0$ we calculate

$$\theta(t) = \frac{v_R - v_L}{b} t + \theta_0$$

Taking into account that the the velocity of the middle point is the average of the wheel velocities, we easily come up with the fact that dor the absolute frame of reference the following differential equations apply:

$$\frac{dx}{dt} = \frac{v_R + v_L}{2} \cos\left(\theta(t)\right) \qquad \frac{dy}{dt} = \frac{v_R + v_L}{2} \sin\left(\theta(t)\right)$$

Integrating and applying the initial position of the robot , we have:

$$x(t) = x_0 + \frac{b(v_R + v_L)}{2(v_R - v_L)}\Big( \sin\big((v_R - v_L)t/b + \theta_0\big) - \sin\big(\theta(t)\big)\Big)$$

$$y(t) = y_0 - \frac{b(v_R + v_L)}{2(v_R - v_L)}\Big( \cos\big((v_R - v_L)t/b + \theta_0\big) - \cos\big(\theta(t)\big)\Big)$$

All odometry calculations are carried out by the master, which requests the readings of the slave through the $I^2C$ bus. The equations stated above are converted into integer math and fast trigonometric routines have been implemented in assembly to keep the execution time as small as possible. Another issue is that the refresh rate of the odometry calculations cannot be kept constant, since according to the velocity some numbers may be too small or to big. For instance, at low speeds it may be that between two or even hundreds of trajectory calculations (902 μsec), the encoders move a few clicks or not at all and this information cannot be used in calculations as it would lead to void results due to the integer arithmetic used. Thus, after each motion command, the master controller calculates an appropriate refresh rate for odometry readings in the range of 28 msec to 0.23 sec, in order to minimise the quantisation error of the calculations.

#### 5.3.3.3 Error Handling

As stated many times before, the motor controllers can be dangerous. This means that error handling is a very important issue and must be consistent. Each controller has to recognise and handle error cases concerning:

- Command syntax and values out of bound
- High temperature
- High current
- Failure of motor
- Presence and smooth operation of fellow controller

The most demanding is the last error case. Both the master and the slave must detect the absence or malfunction of their fellow controller, stop immediately and infer from the execution of further commands until the error is resolved, which in the best case is a loose cable. To achieve this, the master, which is in control of the $I^2C$ bus, sends a heartbeat packet containing status information on a regular basis, apart from the semi-regular odometry requests and receives an acknowledge packet also containing

status info. If for any reason this handshake does not succeed or if the status info indicates a critical error, both controllers cut off the current supply to the motors and notify the server. The procedure has proven itself to protect the robotic platform at least from bad cable contacts[5].

## 5.4 IR Sensors Controller

The sole responsibility of the IR module is the control of the peripheral IR sensors, so both its hardware and firmware are quite simple. However, it must perform flawlessly, since it is the last protection between the robotic platform and a misplaced wall. The second important issue is that it should be able to operate reliably even in a IR noisy environment, i.e. when other IR emitting devices like fellow robots are present.

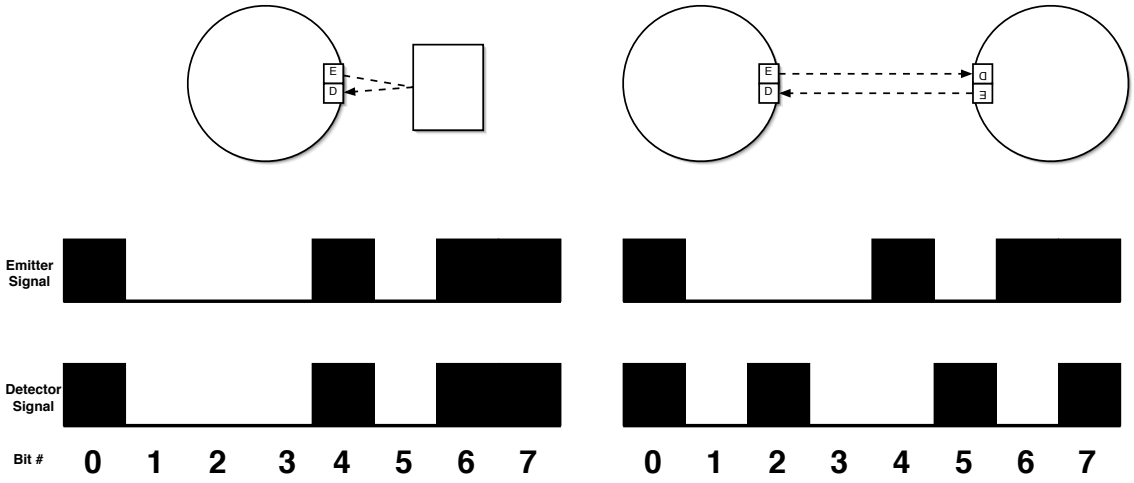



Figure 5.12: IR transmission sequence

The IR sensors used transmit and receive a modulated IR signal and are sensitive to its intensity and not its angle of reflection. The fact that the signal is modulated to a specific frequency means that the receivers are not highly affected by other sources of IR light. Yet, in the RoboCup competition the robotic platform will have to cooperate with identical teammates which will have the same IR sensors at the same height. The particular IR receivers can detect an obstacle from a distance of 10-30 cm depending on its reflectivity, but can also be triggered by a transmitter aiming directly at the from a distance of up to 2 metres. 4 robots times 12 sensors in a $8 \times 12$ field have good chances of rendering all IR sensors useless by themselves, and this is not as good as jamming the opponents. To avoid this problem, the equivalent of a open collector logic circuit is implemented. Each sensor emits a sequence of pseudorandom pulses

[5] but not from its own bugs

and expects to detect the reflection of the same sequence. If the receiver detects an IR signal that doesn't match the transmission, it is assumed that it is not a reflection and it is silently ignored. Each transmitter uses an 8-bit sequence, which on option can be either pseudorandom or fixed. The pseudorandom option has been tested to perform quite well with a very small error probability and doesn't need any configuration, which is necessary for the fixed sequence. The latter however guaranties that no two robots use the same key. In both cases, the first and last bit are always 1 and are used as framing bits, leaving 64 different patterns. Figure 5.12 shows the case of a reflected and a direct signal reception. The direct one would not confuse the detection logic of the controller.

## 5.4.1 Hardware & Firmware

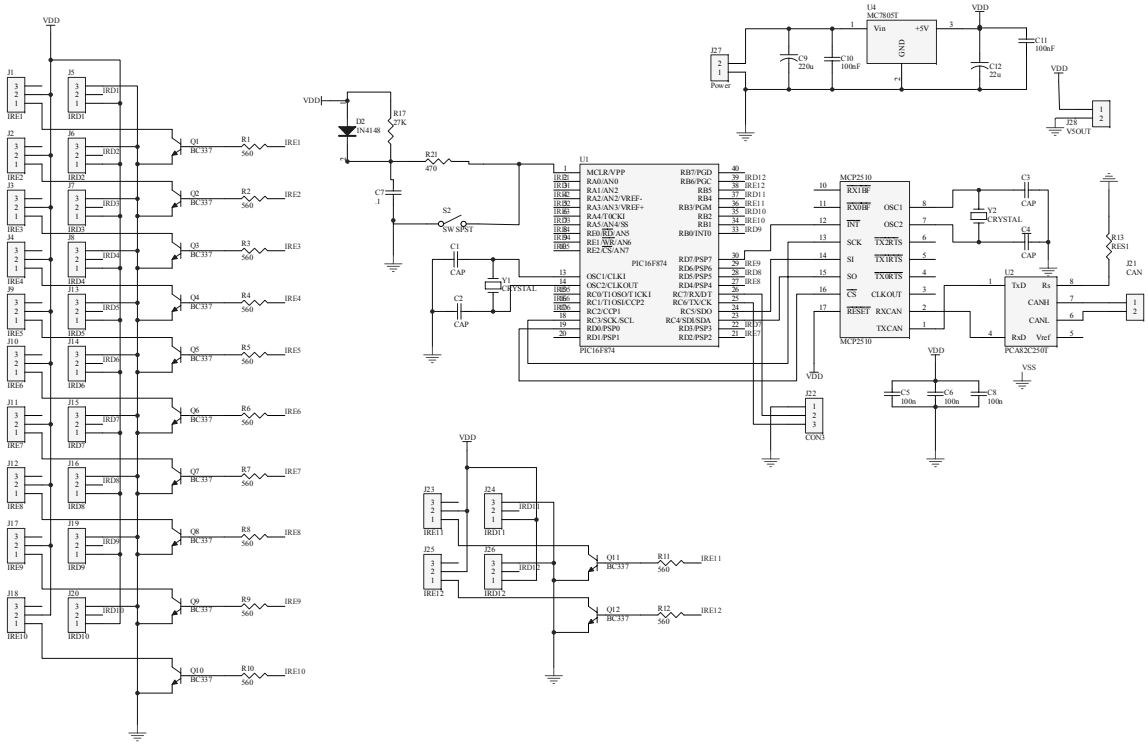


Figure 5.13: IR sensors controller schematic

The main component of the IR module is a PIC 18F452 microcontroller operating at 20 MHz. This device has all the necessary I/O ports (24 just for the sensors), yet lacks a CAN bus interface. It communicates via a dedicated serial connection to an MCP2510 CAN controller which in turn is interfaced to the CAN bus through a PCA82C250 chip. Each of the 12 IR emitters can be individually turned on and off by a npn transistor controlled by the microcontroller, while the IR detectors are directly connected to it. The IR module has a relatively high current consumption when all IR

emitters are turned on and so the power source, which is once more a 78S05 voltage regulator, must be thoroughly cooled. A small aluminium cooler is used just to be on the safe side, since the firmware never allows more than one IR emitter to be switched on in the first place . The control program activates and tests one sensor at a time in a round-robin fashion. The 8-bit pulses are transmitted at a frequency of 1 KHz and the period between two successive activation rounds varies from 13 ms to 104 ms. The refresh rate is not constant because the controller does not always transmit the full 8-bit code for each sensor. The sequence is interrupted as soon as a disparity in the received signal is detected. In the common case that no object or IR light source is present, this means that each sensor will transmit only the first framing bit and terminate its sequence since no reflection will be detected.

Each time a collision is detected by the firmware, a certain type of packet is broadcast to the CAN bus. This collision indication packet contains the ID of the sensor that detected the collision and is assigned the highest priority in the address/priority scheme of the CAN bus. A similar packet with slightly lower priority indicates the end of a collision. Both types of packets are intercepted by the server and the motion controller, the latter of which can be configured to brake automatically when a collision is detected, without the aid of the server. Apart from these triggered messages, the controller can also send information about its state upon request. It follows the same conventions as the rest of the controllers, i.e. it signals its presence on the CAN bus by replying to echo requests issued by the server and acknowledges each received command with an ACK packet carrying the modulo 256 sum of it.

## 5.5 Console & Display Controller

The console and display controller is a module which has some distinct features compared to any of the other ones. None of its functions are necessary for the operation of the robotic platform when controlled through the main computer. It is connected both to the CAN bus and the serial port of the computer and does not comply to the general hierarchical structure of the modules, according to which the computer is the root. The main function of the console module is to take over the synchronisation of the rest of the modules when the main computer is not in operation, and as such it is an extra feature to make the robotic platform more "user friendly"[6], mainly as regards its motion. It incorporates a wide range of peripherals that are used as motion command inputs, making the robot an expensive remote controlled toy, which is however a very important and useful facility for research platforms. Secondly, the console module

[6] just like Windows?

provides indications about the status of the platform, such as battery level and errors. This facility operates in parallel with the server, which also has access to the same information but needs at least a terminal or a network connection to be queried about. It is also used to control the switching on and off of the computer, the switching on being anyway very easy but the poweroff demanding a means of communication to the operating system. Of course, the latter can be also accomplished by configuring the ACPI features of the computer to switch to runlevel 6 when the power button is pressed, however it is useful to have a full access terminal connection to the computer[7]. Lastly, one very simple function that it performs and is essential even when the server program is in command of the robot is the kill command. This function is extremely simple and could be assigned to any of the rest controllers. Anyway, one of them must be used to check the kill request buttons and send the appropriate packet to the CAN bus. Because of its placement on top of the robotic platform this is performed by the console controller .

### 5.5.1 Hardware

All functions in the console module are controlled by a PIC 18F458 operating at 40 MHz. Its inputs and outputs are carefully chosen to achieve the maximum possible number of operational options while sparing computational resources. Not a single pin is left free and a couple of them are even charged with different functions. Commands to the controller can be issued in 4 different ways:

- Through 4 buttons that when the controller is installed to platform are accessible on the top cover. These buttons are used to navigate through the menus of the display and select options. Each one of them is interfaced to the microcontroller by a debouncing circuit using Schmitt-triggers.

- Through RF signals. The module provides 4 inputs compatible with commercial radio control receivers. They provide the necessary voltage levels to the receiver through appropriate headers. The outputs of the receiver, which are usually 50 Hz PWM signals, are connected to interrupt enabled ports on the microcontroller, in order to have their dutycycle estimated using as little resources as possible and no polling functions.

- Through analog joystick. A standard 15 pin D-Sub female jack is connected to the microcontroller and provides an interface to old-fashioned commercial analog joysticks. The x and y potentiometers are interfaced to the ADC inputs of the microcontroller, while the buttons are buffered by Schmitt-triger debouncing circuits.

[7] and on the other hand, why make things easy?

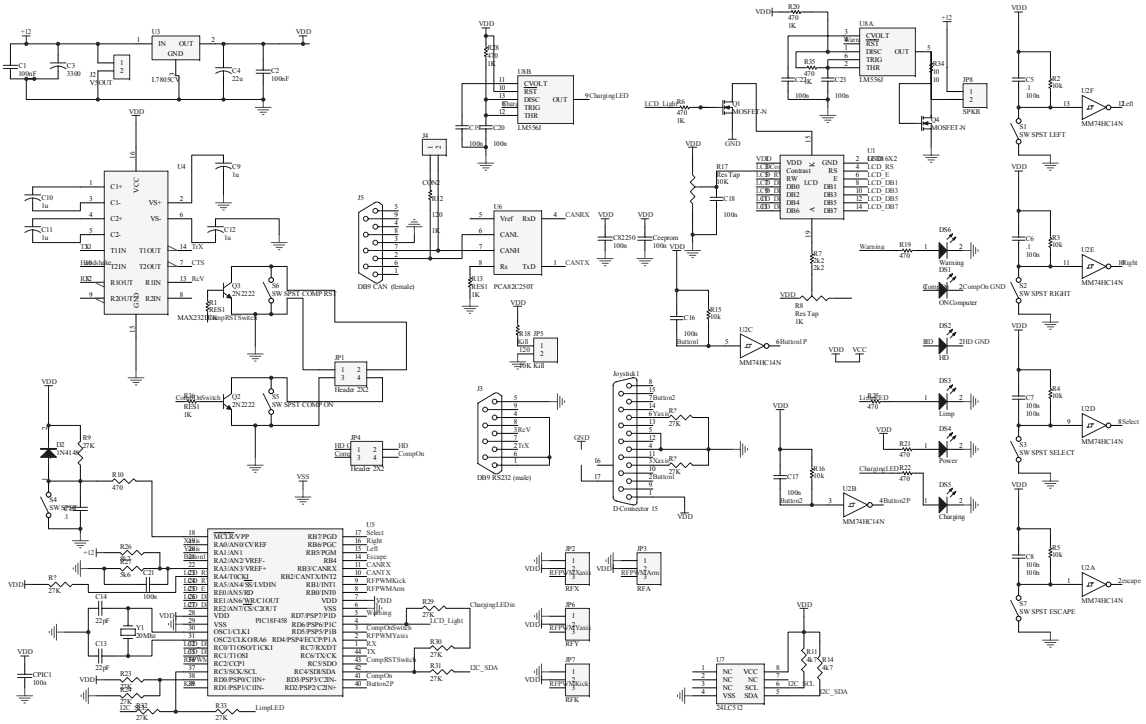


Figure 5.14: Console controller schematic

- Through a serial terminal. The serial input of the microcontroller is coupled by a MAX232 chip making it compatible to the RS-232 standard. Any DTE can be connected to it through a 9-pin D-Sub female connector. The pinouts are configured as a DCE, which enables us to use the exceptionally common standard modem cable for the connection with a PC instead of the rare-when-needed null modem cable.

  As means of communication with the outer world, the module utilises the following peripherals:

- A $2 \times 16$ LCD. It is a HD44780 compatible device, which is pretty much the one and only standard for character displays. Apart from the data connections to control it, the microcontroller also drives a FET that enables its backlight.

- 6 indicative LEDs. 4 are driven by the microcontroller while the rest are connected to the computer and the power source and are thus independent.

- A buzzer. Since no pin on the PIC was free to provide an audible frequency pulse, a 555 timer circuit is tuned at 1 kHz and drives a little speaker through a FET

- A CAN bus interface. As usual, a PCA82C250 chip is used to make the appropriate signal conversions. Although the firmware has the complete functionality to send and receive commands through the CAN bus, at present the reception is not used. There

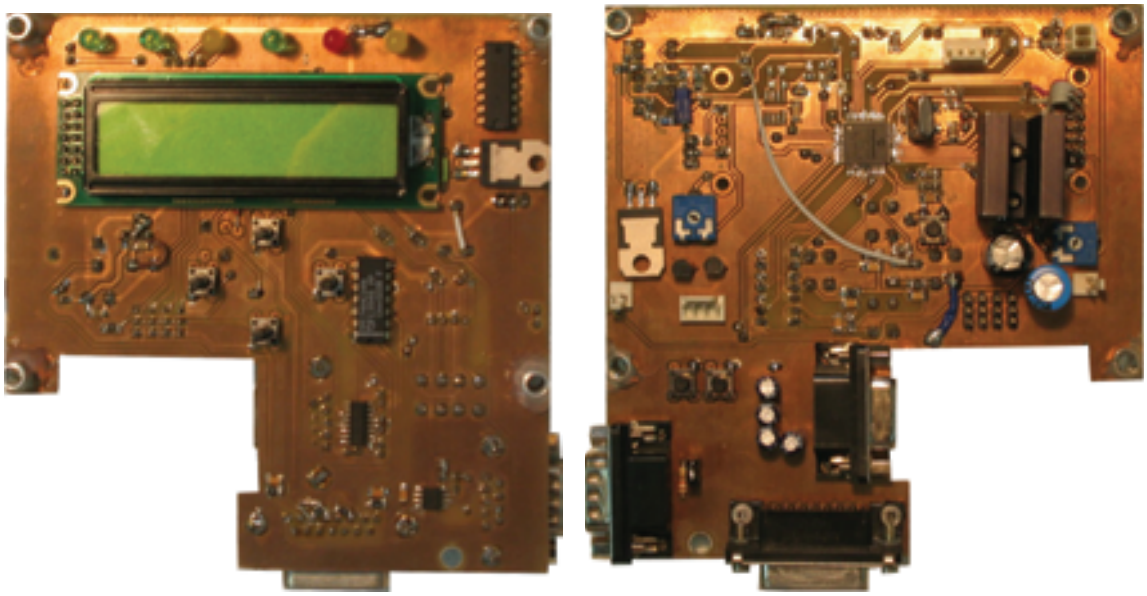


Figure 5.15: LCD controller

is no functionality useful to the other modules that would be requested by a CAN bus command, apart from presenting some information on the LCD. Although this is useful, the serial port is much more suitable for this. The only command that is processed is the echo request that is issued by the server to check the presence of the controllers on the bus.

- The serial port. It is clear that it can be used both for receiving and dispatching commands. At the current state of development and during normal operation the serial is used to request a poweroff from the computer. To do this, the module has only to log-in as a certain user to the serial console. With very few configuration changes to the operating system this is as easy as sending a single string. There are many open source projects that can be used and it is even easier to develop a small server that would do more complex things such as displaying information on the LCD and accepting more commands. Yet, this is probably more of a game than something really useful.

The reset and power-on buttons of the computer are also controlled by the module, but bypass switches are included too so that they can operated by hand as a last resort. Finally, the kill button is connected to an interrupt enabled input of the microcontroller.

Despite of being quite complex in hardware as many different peripherals are implemented, the firmware of the microcontroller is not very complex but merely large. There aren't any critical functions apart from the kill request, and the implementation of this function is trivial. The RF functionality was a very small issue, since we have to measure the duration of 4 independent pulses with a reasonable precision. On the

other hand, the many different interfaces do not operate simultaneously, so the decent processing power of the microcontroller can be dedicated to each one of them without worrying about guaranteed delays and resource allocation. So for the most part a simple and not time critical code evaluates the inputs of the peripherals and transmits the appropriate commands, mostly motion ones, to the CAN bus. However, the code can and will be large due to the human interface, which are the menus presented on the LCD. At this state of development, the display controller is a module with a really wide range of capabilities but it is primarily there for ease of use and only a few are used. With the proper feedback from application programmers and researchers it will be made clear which are useful and whether more should be implemented or not.

"Any sufficiently advanced bug is indistinguishable from a feature"

RICH KULAWIEC

# Chapter 6

# Software

The hardware of the platform is useless as a robot without the proper software. Firstly, pieces of software are necessary to synchronise and interface to each module and since most of them are custom made there are no readily-made solutions. The fact that they are custom made gives great flexibility in the design, since at many stages of the development the hardware was altered to fit the software and vice-versa. On the other hand, bugs were flying through the air as with every prototype that has self-respect, so especially the controlling software had to be designed and implemented as stable and fault-proof as possible.

Two were the basic aims in the design or choice of the software for the robotic platform. Firstly the computer should be loaded with the appropriate programs and and peripherals and configured in such a way as to be a self contained and easily accessible and usable platform. The second and most difficult to achieve issue, is that the application programmers should be able to interface to the facilities of the platform completely transparently and without knowing anything about the underlying hardware.

## 6.1 Operating System

The operating system of the computer is Linux , kernel version 2.6.4-52. The distribution chosen is SuSe 9.1, however the reason for this is solely ease of maintenance and there are no features used that are specific to this distribution.

As with any Linux distribution, a wide variety of very efficient and free program-

ming, development, networking and visualisation tools are provided out of the box. We have used them for the programming and testing of the servers and test application programs and they are also available to the application writers. The servers are built using the lowest level facilities possible, in order for them to be fast and not CPU-intensive. The programming libraries used are well known , efficient, debugged and independent from the underlying hardware. Only the motion server relies on a hardware specific beta version module, namely the driver of the CAN interface, however its architecture is not heavily based on it.

### 6.1.1 Communication with the Computer

The computer has the usual keyboard, mouse and monitor sockets, however these are seldom a convenient I/O option for a mobile robotic platform. The network interfaces are the primary means of communication, through TCP/IP sockets for consoles or X-Windows. A wireless and a cabled ethernet interface, `/dev/eth0` and `/dev/eth1` respectively, are available on the robot. The cabled interface is designated as the default route and has a valid IP, however it is only used for high bandwidth communication, such as file transfer and X-Windows/visualisation. The wireless interface is the one that is used on normal conditions, i.e. when the robot is not stationary. It creates a freely joinable 802.11b network, with a class C private IP address . This configuration is suitable for controlling the platform through WiFi equipped standalone computers. When a structured 802.11 network is available it can be easily changed to be better fitted to it.

A serial port is used as a back-up login option. It is configured as a 115200 or 9600 bps VT100 terminal input and is normally connected to the console controller. Under normal conditions it only receives the shutdown command from the console, however it has full access to the operating system and can be used any time with a terminal[1] (more likely a terminal emulator) for maintenance.

### 6.1.2 Booting

The computer is switched on through the console on top of the robotic platform. It boots directly into Linux and establishes the network connections and launches the servers. Both wireless and cabled ethernet interfaces are enabled on boot by default.

The SuSe distribution of Linux uses the System V booting procedure. A compatible shell script is located in `/etc/init.d` for each one of the server programs, invoked through the standard boot scripts or manually with the `start/restart/stop` com-

[1] how about a real DEC VT100?

mands. They all direct the standard output and standard error of their corresponding programs to logfiles in `/var/log/`, cleaned on each boot. So we have 3 scripts with the following file names:

- `roboserver` for the motion server, logging to `roboserver.log`
- `visionserver` for the vision server, logging to `visionserver.log`
- `laserserver` for the laser server, logging to `laserserver.log`

## 6.2 Vision Server

The vision server handles and synchronises the cameras of the robotic platform. It provides a fast and efficient way for multiple application programs to access the vision system simultaneously. Due to the high bandwidth of video, the control and data channels are separate, namely sockets for the former and shared memory for the latter. Apart from achieving large data transfer rates, this configuration ensures that the average delay to requests is faintly affected by the number of clients and by using DMA and ring buffers we also try to keep this delay as small as possible.

Upon launching, the server program initialises the FireWire bus/es and searches for available cameras. It then queries the capabilities of the cameras and initialises them too, according to a separate configuration file for each camera. The naming scheme and format of the configuration files are given in Appendix A.1.1 . The appropriate initialisation of the FireWire controller and cameras is a relatively long procedure, due to the functional characteristics of this bus and the DMA facility used. When a device on the bus has not cleanly quitted, the reuse of its resources imposes very thorough resetting procedures, in which the allocated DMA buffers are released and the data stream stopped. The same considerations hold also for the consistent deallocation of the FireWire devices and controllers, i.e. upon releasing a camera or quitting the server program.

After all initialisations are complete, the server spawns a listener thread that accepts TCP socket connections on port 3973. Clients not running on the local machine are rejected, since they cannot have access to shared memory, which is used for data transfer. When a local client is accepted, a separate thread is spawned for each of them that is passed control over the connection from the listener thread, which continues to wait for incoming connections. The connection handler is responsible for receiving the control commands from the clients, interpreting them and acting accordingly. The commands and the replies are short printable ASCII strings for debugging and programming clarity purposes, The use of the more efficient binary strings is not justified

by the gain in size and speed, since the bandwidth of the traffic through the socket is very small. The connection handler thread constantly monitors its connection and the syntax of the commands for errors, in which case it disconnects them and deallocates their resources. Handshake points exist in the procedures of all commands that require synchronisation between the client and the server. A misbehaving client could hang the thread assigned to it when it doesn't comply with the handshaking procedures, so the listener thread also acts as a watchdog by periodically checking all active threads for hangs. In case of one it cancels them and deallocates all resources used by it and the corresponding client.

Any camera device can be requested by any client. Unused ones can be configured as regards the size, the colourspace, the format and the frame rate of the capture. Cameras already in use are also available, yet attempts for changes in the parameters set by the first client are denied to the consecutive ones. The format of the video stream can only be changed after all clients have released a particular camera. However, all clients using a device have access to the settings of the capture, such as exposure and brightness. This way one can change, for example, the contrast of the capture while an independent vision algorithm program is running.

The server program is not bound to a particular type of cameras. Theoretically all routines are programmed in such a way as to either not rely on the capabilities of the camera at all or to query them explicitly and not make any assumptions. The clients can configure the video stream to any particular type, however if the requested options are not available the server will try to find the closest match if any. If the requested colourspace is supported by the camera, the closest frame size and lastly frame rate are selected, otherwise the configuration is aborted. On either case the effective settings or the failure to find an appropriate supported format is reported back to the client program as part of the communication procedure for the configuration. The installed cameras support the video modes shown in Table 6.1.

| Mode | Size | Frames/sec |
|---|---|---|
| RGB-24 | 640×480 | 15 , 7.5 , 3.75 |
| YUV-444 | 160×120 | 30 , 15 , 7.5 |
| YUV-422 | 640×480 | 15 , 7.5 , 3.75 |
| | 320×240 | 30 , 15 , 7.5 , 3.75 |
| YUV-411 | 640×480 | 30 , 15 , 7.5 , 3.75 |
| 8-bit grayscale | 640×480 | 30 , 15 , 7.5 , 3.75 |

Table 6.1: Fire-i camera video modes

A feature that is provided by the server and is independent of the type of cameras is the rectification of panoramic images. The settings for this are also configured

along with the video stream format by the first client. The rectification discards the unused black areas of a picture frame taken through a parabolic mirror and transforms the circular image to a panoramic view. This transform is often used by the application programs so it is provided as a feature since the processing time needed to do a memory copy along with the transformation computations is comparable to that of a mere memory copy, which is anyway unavoidable in our architecture of the server. The computations are carried out with the aid of trigonometric look-up tables to minimise the overhead as possible. The parameters for the rectification are specific to the optical characteristics of the panoramic camera and are defined in the configuration file for the device (Appendix A.1.1). Figures 6.1 and 6.2 show frames taken through our robotic platform's panoramic camera with and without rectification.

After a client has successfully configured the video stream of a camera and the rectification options for the frames, it receives information about the effective settings and about the shared memory address it will have to use to get the frames. Then the client program can send the request to start the acquisition.

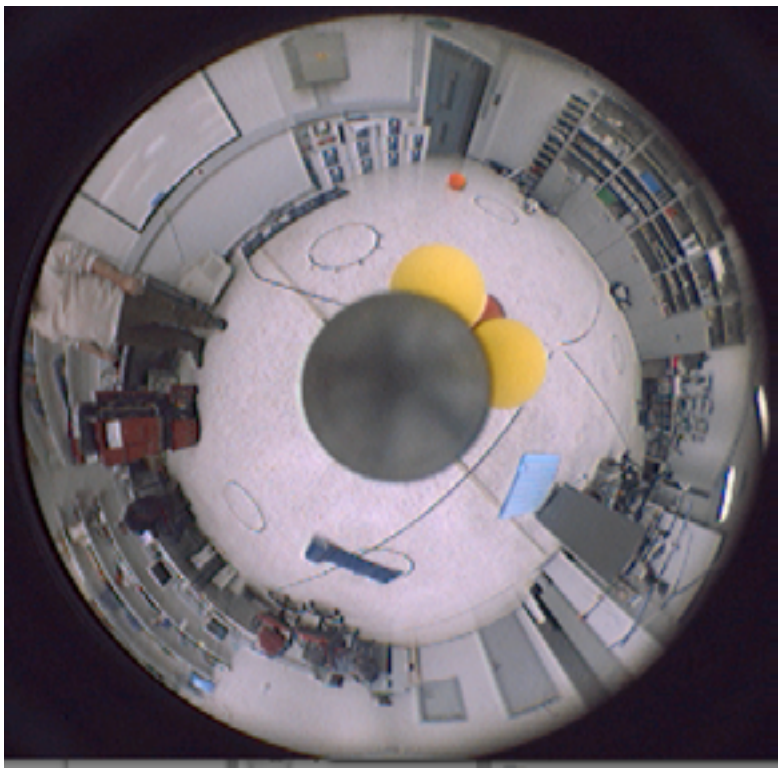


Figure 6.1: Raw panoramic image

Figure 6.2: Rectified panoramic image

### 6.2.1 Vision Data Handling and Policy

When the start of the acquisition is requested, the server launches a thread to handle the video stream of the camera it is responsible for and remains active until the device is released by all clients using it. The thread allocates a ring buffer designated as shared memory to store the incoming frames and, if rectification is also desired, it fills up the necessary trigonometric look-up tables. It then instructs the camera to begin the transmission of the configured video stream through the bus and the FireWire controller to start filling a non shared ring buffer with frames by DMA (direct memory access), that is without the intervention of the central processor. After initialising, the acquisition thread constantly transfers frames from the non shared to the shared ring memory buffer, either raw or by simultaneously rectifying them, and also adds a timestamp to each one. The use of two ring buffers is imposed by the fact that the server must be able to handle many clients at the same time.

When a client requests an image, its serving thread replies to it with the index to the address in the shared memory that contains the most recent frame available and the buffer is marked as used, however many clients can use the same buffer simultaneously if no newer frame has been already captured. The clients can keep the buffer reserved as long as each of them needs to process the image and it becomes free only after all of them have released it. While the buffer is occupied, the acquisition thread omits its address space in the ring buffer sequence. The client programs must either explicitly release the buffer or request another image, in which case it is automatically freed and the one holding the most recent frame reserved and its index passed to the application, in order not to let any misbehaving program occupy more than one slot in the ring.

### 6.2.2 Programming Issues

The server is programmed to be as fast and stable as possible. There is thorough error checking on the commands processed as well as on the procedures carried out. The program tries to protect itself from misbehaving clients and to recover completely from all errors caused by them. No assumptions are made concerning the type of the cameras, apart from that they are FireWire ones. Their capabilities are determined by

querying, so changing the camera brand and type should not be an issue.

All functions are blocking and DMA is used in the video stream processing procedure in order to unload the CPU. The buffering scheme and the separate control and data channels ensures that the average delay to a frame request remains small and is almost not affected by the number of users. On the other hand , the command scheme is not optimised, yet it has been already stated that this is not substantial and serves primarily debugging and programming clarity purposes.

The vision server has been tested with up to 3 cameras at 15 frames/sec and 640×480 resolution and up to 10 clients using different or the same cameras. Although this program handles large amounts of data, the CPU occupation percentage was almost not measurable.

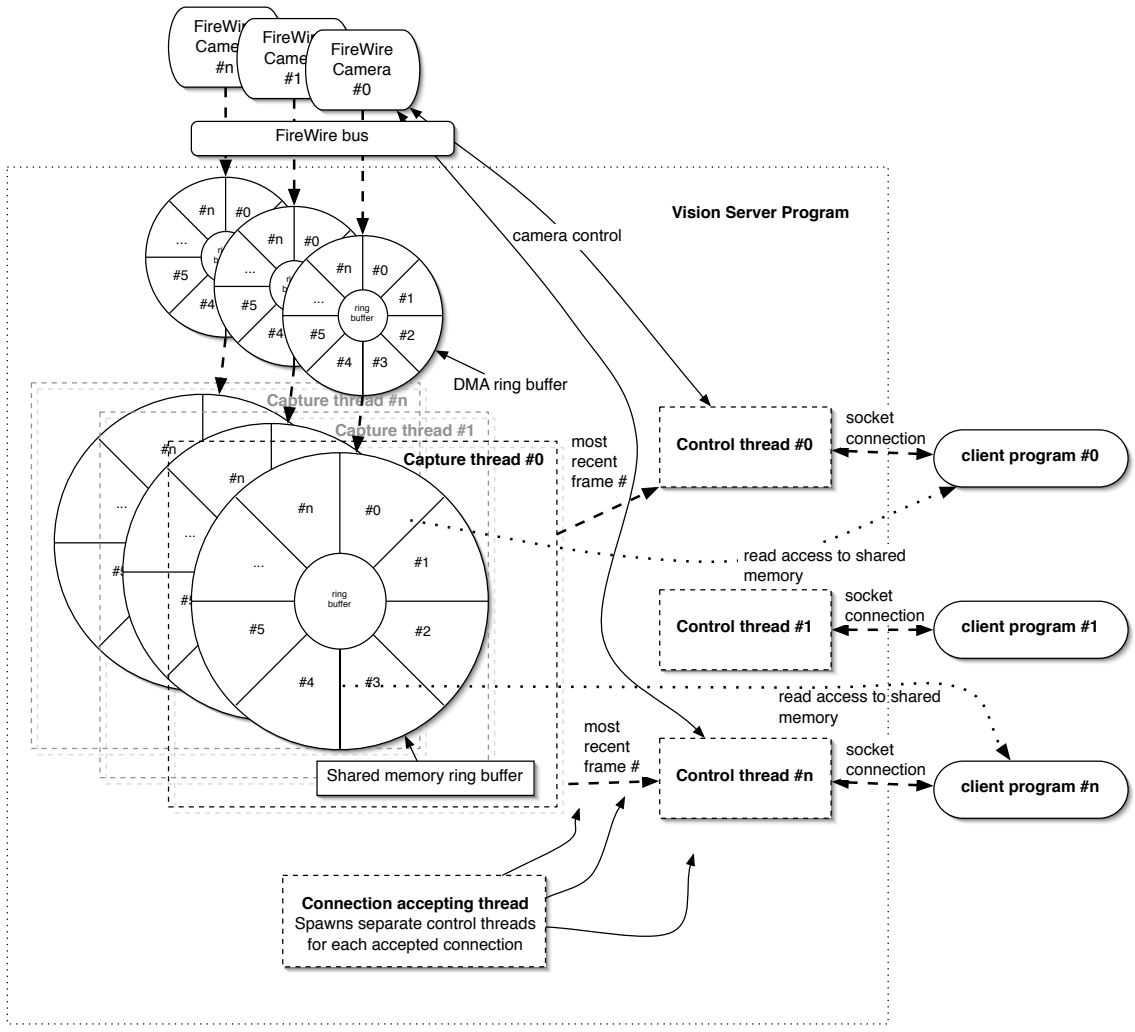



Figure 6.3: Vision server architecture

## 6.3 Motion Server

The motion server controls and synchronises all modules connected to the CAN bus. Stability and proper error handling are the main requisites for this program, since it is responsible for motion functions that are an accident waiting to happen if misused. Moreover, all devices on the CAN bus are prototypes, so errors and bugs do and will happen during their operation. The server must be able to detect them and in co-operation with the recovering facilities of the controllers always bring the robotic platform to a safe state.

Two types of connections are handled by the program, one primary and multiple secondary ones. Primary connections are served through the TCP port 3970, while secondary ones through TCP port 3971. No distinctions are made between local and remote clients, since socket connections are transparent. The syntax of the commands and the connection/communication procedures are the same for both primary and secondary clients, the only difference being the access rights to some of them. For safety and logical consistency purposes, only one client, the one connected through the primary port, is granted access to the motion and active commands, such as changing controller parameters and functions. At the same time, the server allows many secondary clients to issue passive commands, such as querying the odometry or the IR sensors' state. If a secondary client issues one of the privileged active commands, it is ignored and the access rights error is notified to the offending program. There is only one exception to this scheme. All clients have access to one privileged command, namely the `kill` command, as a safety precaution. Its function is to stop immediately the robotic platform and suppress all subsequent movement requests until the robotic platform is re-enabled by the `enable` command. The format of the commands is mnemonic and printable ASCII. So the the robot can be moved or monitored and killed by any RFC-1184 (linemode option) compliant telnet client, which luckily or unluckily excludes the native Windows implementation.

### 6.3.1 Programming Issues

After the motion server is launched, it firstly configures the CAN bus controller card and then checks for the presence of all modules connected to it by sending a broadcast echo request packet. All controllers respond to this request by sending an echo packet and their version information. If all modules are on bus, the server program resets them, in order to get them in a known state. The handling of the responses for each controller is assigned to separate threads. The CAN bus modules generate as a default option a lot of debugging and status messages and each thread is responsible for a

particular controller only. Thus a common framework is present, but each thread suits the specific characteristics and demands of each module.

In Figure 6.4 a (c)rude depiction of the basic function blocks of the server program is shown. Though there are separate threads for each motor controller, the one dedicated to the slave is included for consistency. The motor controllers behave as a single module through the master, while the slave is transparent and channels its error messages through the former. However , commands can still be sent to the slave and replies received. This is only in few cases useful, since it might confuse the master or force it to reset, in order to recover. This extra functionality is supported by the architecture of the server but not used as such. It is also useful and consistent in programming and structure in extreme error cases, such as when the the master controller malfunctions or is disconnected. Great effort has been given to program the server in such away that if possible all error conditions that can be a hazard for the robot are sufficiently handled. This means at least to bring the robot to a safe state and notify the user either directly or by log files. Non critical errors are just passed on to the clients and generally do not invoke other actions on behalf of the server.

Primary and secondary connections are handled by two threads, one for each type. These threads take over both the handshake and acceptance of a connection and the processing of commands. The primary connection thread serves a single client only,

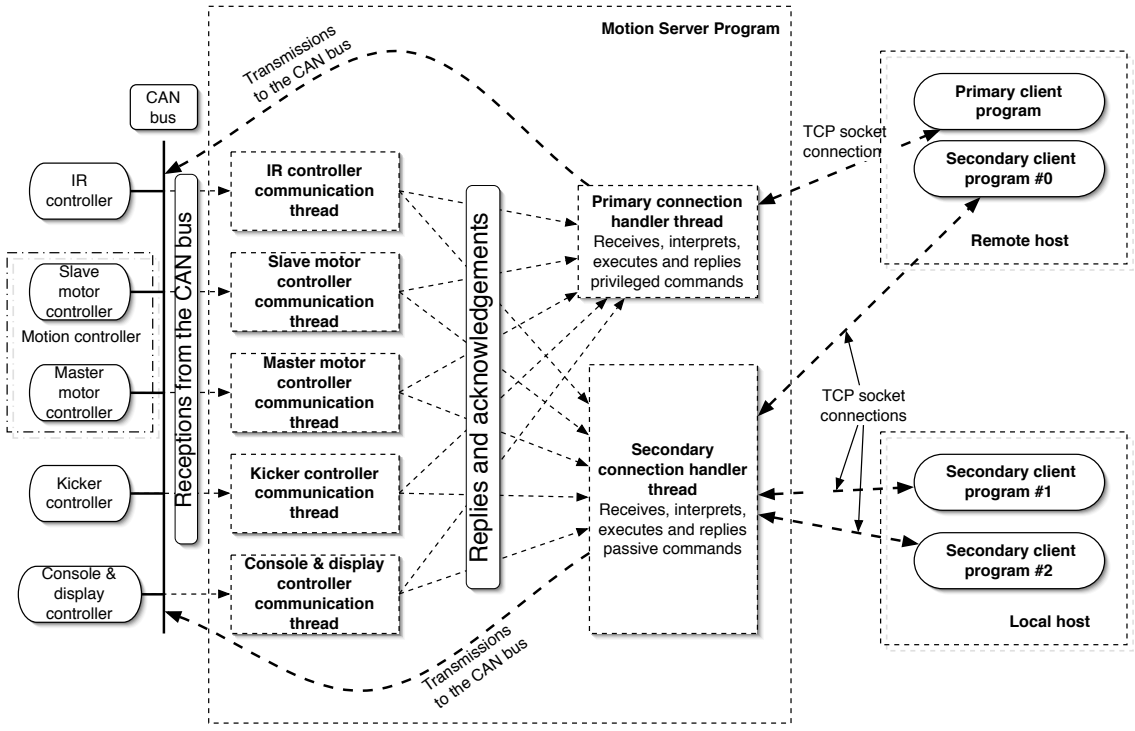



Figure 6.4: Motion server architecture

| Error Description | Severity | Action |
|---|---|---|
| Missing module | Critical | Terminate server program |
| Motor not responding | Critical | Warning |
| Motor controllers intercommunication error | Severe | Warning/Reset controllers |
| Overcurrent protection activated | Light | Warning |
| High current consumption | Trivial | Warning |
| Low battery voltage | Light | Warning |
| Very Low battery voltage | Severe | Warning/Shutdown |

Table 6.2: Error conditions

while the secondary handles many in a pseudo-parallel fashion. This way the number of secondary clients negatively affects the response delay they experience, yet it is an easy way to ensure that the primary connection always holds a higher priority and has a bounded delay. The threads use a common module to check the syntax and interpret the commands from the clients and communicate through it to the modules involved. The responses and acknowledgements are received by the thread responsible for the module in question and piped to the primary or secondary connection threads, which in turn may or may not notify the caller and all other clients according to the particular circumstances and commands.

The CAN bus specifications impose a packet size of up to 8 bytes. In order to keep each packet self contained, all commands and responses are also up to 8 bytes long. For programming clarity and debugging purposes, the commands are more or less mnemonic, using printable ASCII characters. However, in the interest of efficiency, the numbers that are passed as options are binary. As a consequence, most commands perform relatively simple actions, which in turn means that some very often used procedures require multiple packets. Yet this "reduced instruction set" is more flexible and less error-prone. The packet format for the commands and the responses of each of the modules varies between them. There are some common packets and a general common syntax but it is not always followed if it suits the needs of the particular controller better. Regardless of the format of the packets, a checksum packet acknowledges the reception of every command by the controllers. The checksum is very simple and is included only for error detection and not error correction. It is trivially and very quickly calculated by a modulo 256 addition of the data bytes in the received packet. The server checks for time-outs after each command sent to the modules and notifies the client in case of one or quits if more appropriate.

The server and the modules are all prototypes and built and coded from scratch and thus form a great bug with many complex interrelationships. At all stages of the development, this set was the most buggy one, either in serious or trivial ways. However,

the general architectural guidelines followed on each piece of software and hardware seem to make up for this satisfactorily. A crude but efficient way we used most of the times is to include functions to reset the devices on all critical or semi-critical cases, which is in most cases a safe and well-known state. Test applications that were tried out at the most premature stages of development behaved when seen from the outside as expected, while the server and the controllers were going through many error conditions, from which they successfully recovered many times. Most these errors were due to unappropriate delay values for some internal actions on the modules, yet they provided a good simulation for many extreme working and error conditions and both the server and the controllers performed well in it.

## 6.4 Laser Server

The laser server combines features, and also pieces of code, from both other servers, since the demands and requirements lie also in between. It has to handle data, whose bandwidth ranges from a few up to 480 kbps, which is neither considerate nor trivial. Also, the laser measuring device outputs a continuous stream of measurements, in sets of about 1 kbyte, which must be accessible to many clients. These two facts suggest the following:

- In order to serve many clients with a minimum request processing delay, we will have to use buffering. More specifically, by using a ring buffer we can be certain about the consistent operation of the server in a worst case scenario, when the maximum number of clients is connected and all of them are misbehaving as regards the allocation of resources
- The bandwidth of the measurements' stream is marginally low enough as to make remote clients practical. So both data and commands should be channelled through sockets, which make no distinction between local and remote applications. However, the use of a ring buffer and the necessary access protection scheme means that we already have the framework to also provide the much faster and usable only by local applications shared memory mechanism.

Taking into account these facts, an architecture similar to the one of the vision server is used. Most notable difference is the support of remote clients, although internally this is achieved with just minor modifications and additions.

The access and data handling policy remain identical to the the one followed by the vision server. TCP port 3974 is used for the incoming connections, which are accepted by a dedicated thread that in turn spawns a separate one to handle each of them. At

connection time, the client requests either full socket communication or sockets for commands and shared memory for data. The server can handle an arbitrary number of mixed local and remote clients, up to a compile time set maximum. As in the case of the vision server, no configuration changes are accepted while the device is used by another client, meaning that only the first one is allowed to do so or that everyone else must release it beforehand. As stated before, the major feature that differentiates the laser from the vision server as seen from the outside is easily fitted in the existing framework. When a remote client asks for a set of measurements, its request is handled in the same way as are the local ones and as described in 6.2.1. Yet, instead of getting an index to the appropriate shared memory segment it gets the whole data through the socket.

The functional characteristics of the laser measuring device and the communication bus impose further modifications in the vision server architecture, but luckily they are in a direction towards making it simpler. The first one is that the server is not designed to handle more than one device or one with different characteristics. Doing otherwise would not make any sense due to its cost, its availability and the kind of sensory data it

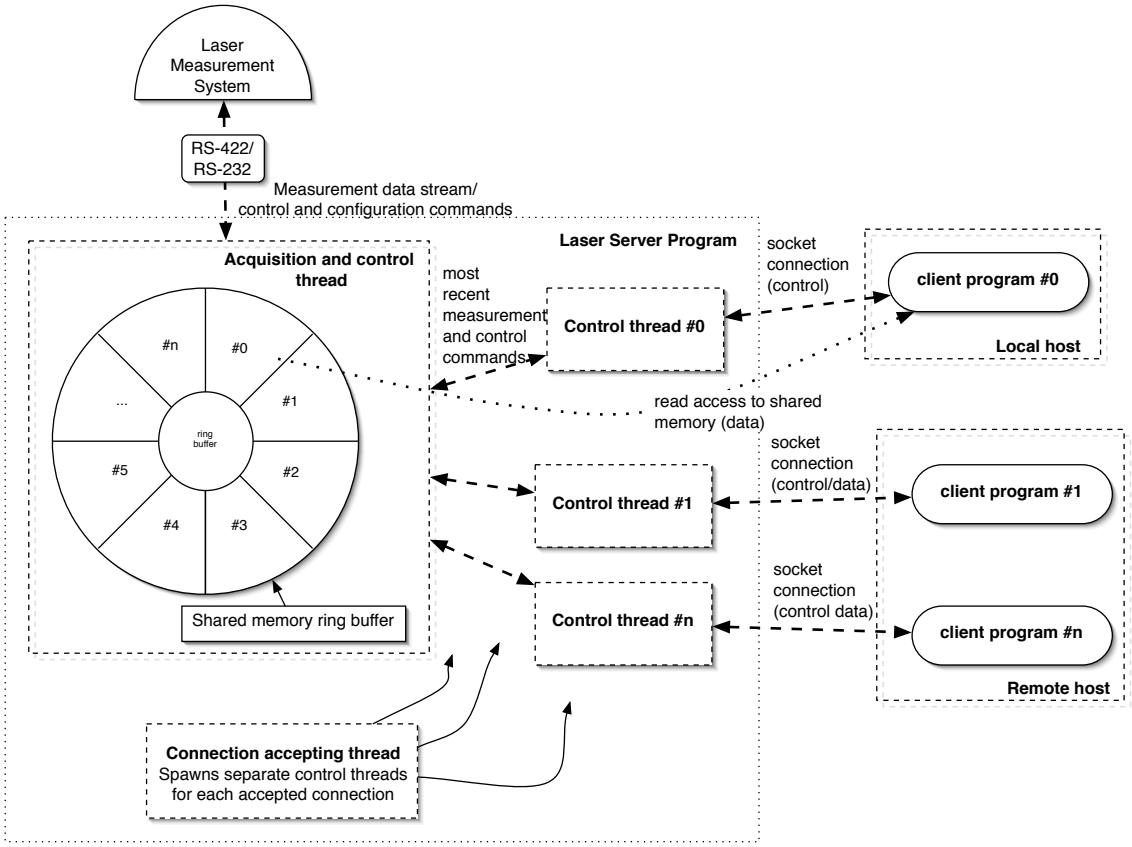



Figure 6.5: Laser server architecture

provides[2]. This fact makes the software completely dedicated to one and well-known device and many complex procedures necessary for a more general purpose server are simplified. Secondly, the laser scanner does not accept any commands or have any options that can be changed while it is streaming measurement data. All available configuration settings, of which only a few are useful for our application including the rate of transmission, the angle, the granularity and the precision of the samples, affect the size and type of the stream and are only valid at the instant of the initial configuration or after the device is stopped. Also, the type of the bus, namely RS-232 /RS-422 suggests a centralised control and exclusive use of it by a single thread. This point significantly differentiates the laser from the vision server, in which each client-serving thread needs to send commands to the cameras, since they take effect during the acquisition and the bus and driver libraries allow this. Lastly, the initialisation of the bus and the device as also as the error handling are much simpler and less bug-prone.

[2] one multi-thousand euro costing laser is already too much

> On the day of the Great On-Turning two soberly dressed programmers with brief cases arrived and were shown discreetly into the office. They were aware that this day they would represent their entire race in its greatest moment (...)
>
> THE HITCHHIKER'S GUIDE TO THE GALAXY

# Chapter 7

# Launching the Robot

Towards the end of its construction, the robotic platform had the opportunity on several occasions to be tested as the application environment it was designed to be. Some small demo programs were written at different stages of development and code maturity to verify each new installed feature. This was part of the design process and aimed at realising the deficiencies or merits of the implementation details and the needs of a robotics researcher when writing programs to test behaviours on an embodied agent. However, there were major differences compared to a real application.

Firstly, the test programs focused heavily on one particular facility each time. When not, they were simple behaviours exhibiting marginally any intelligence and thus did not bring forth the complex interrelationships between software entities that would stress hardware, firmware and servers to their normal operation limits and beyond that. Always the worst case was in mind, and certain presumably improbable sequences of commands that were tested on purpose revealed implementation flaws that would later prove to happen very often in real use, yet a complex application would reveal even more.

Secondly, the test programs were hand in hand with the server programs development, and they were changed sometimes together as a whole when a problem was encountered. In their case, the application writer had a complete overview of the functions, facilities and peculiarities of both the hardware and the software. This would however under no circumstances be the case of robotics researcher that would like to program the platform easily and efficiently. So the first use of the robotic platform by

someone that didn't know and care about its innards would be the ultimate test for it.

## 7.1 The First Application

The first opportunity to test the platform as a complete application environment was given at the final stages of development and provided very useful information about the last implementation details and the correctness of the design architecture. The platform was programmed by students as part of a project for a course of computational perception and action. The project stated that the robot should be able to accomplish the following tasks:

- From an arbitrary position it should find a distinctively coloured starting mark and reach it.
- From that point, it should follow an arbitrary black line on the floor until it reached a distinctive coloured ending mark
- It should be able to detect and avoid any obstacles placed in its path. Two separate objectives were to avoid the obstacles either based only on vision or only on IRs.
- If it detects a ball on its path, it should kick it.

This application was the opportunity to test most facilities of the robotic platform, except for the laser, whose server is anyway the simplest one. The vision and motion & actions server were heavily tested by this project and either performed very well or pointed out the improvements that should be made.

The perceptional part of the application used the panoramic camera and the rectification facility of the server. The rectified image was processed to extract the interesting features based on colour information. The vision server performed flawlessly at all times, with multiple clients accessing the cameras. The best proof for the proper operation of it was the fact that very often the applications were quitting ungracefully, forcing the server to recover. All recovering actions were performed as designed and neither the server nor other applications accessing the same camera devices were harmed by such erratic programs. The CPU utilisation of the server program was so low as to be not measurable by the `top` command. In Figure 7.1 4 frames taken from the panoramic camera while running the detection algorithm are shown. The floor is drawn white, the ball is indicated by a black crosshair and the line by a red one.

The action part of the application performed also well, yet unveiled many subtle implementation issues on the motion server that were quickly solved. I was not surprising that the behaviour that relied solely on vision to move the robot did not reveal

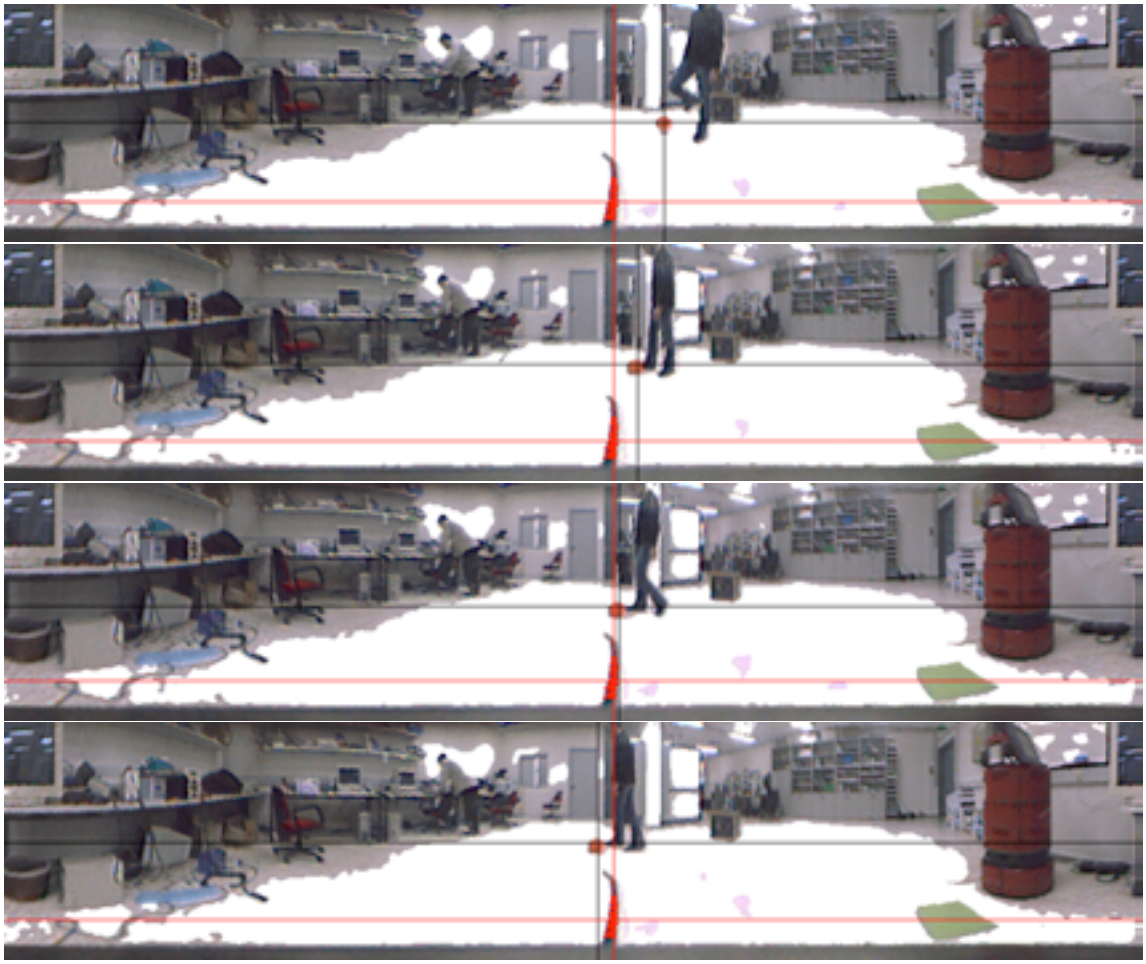


Figure 7.1: Ball detection

any serious deficiencies. It was the one that relied on sensory information from the IR sensors that really stressed the motion server. The requests to it were much more often and many small bugs relative to synchronisation became apparent. The particular program made very little and fast data processing and operated on a loop, so it was sending a constant stream of different commands to the server and almost as fast as they could be processed. These are probably the harshest operating conditions possible for the motion server. However, the logging and error handling facilities built into it helped correct the emerged bugs almost immediately and gave hints about future improvements.

As a byproduct of this project, a ball following application was also implemented. Using some of the already available algorithms of the project, the robot was programmed to chase the ball and kick it when in range, while avoiding the walls by the use of the infrareds. The motion it exhibited was satisfactorily smooth for the crude movement algorithm used, since the controllers managed to make the appropriate corrections.


Figure 7.2: Robot avoiding an obstacle by IR

## 7.2 Conclusions

Apart from the complex tasks involved, as much as it concerns the server programs only, that were tested and verified by the project, the platform proved to be an easy to use application environment and only a little supervision and instruction at the begin-

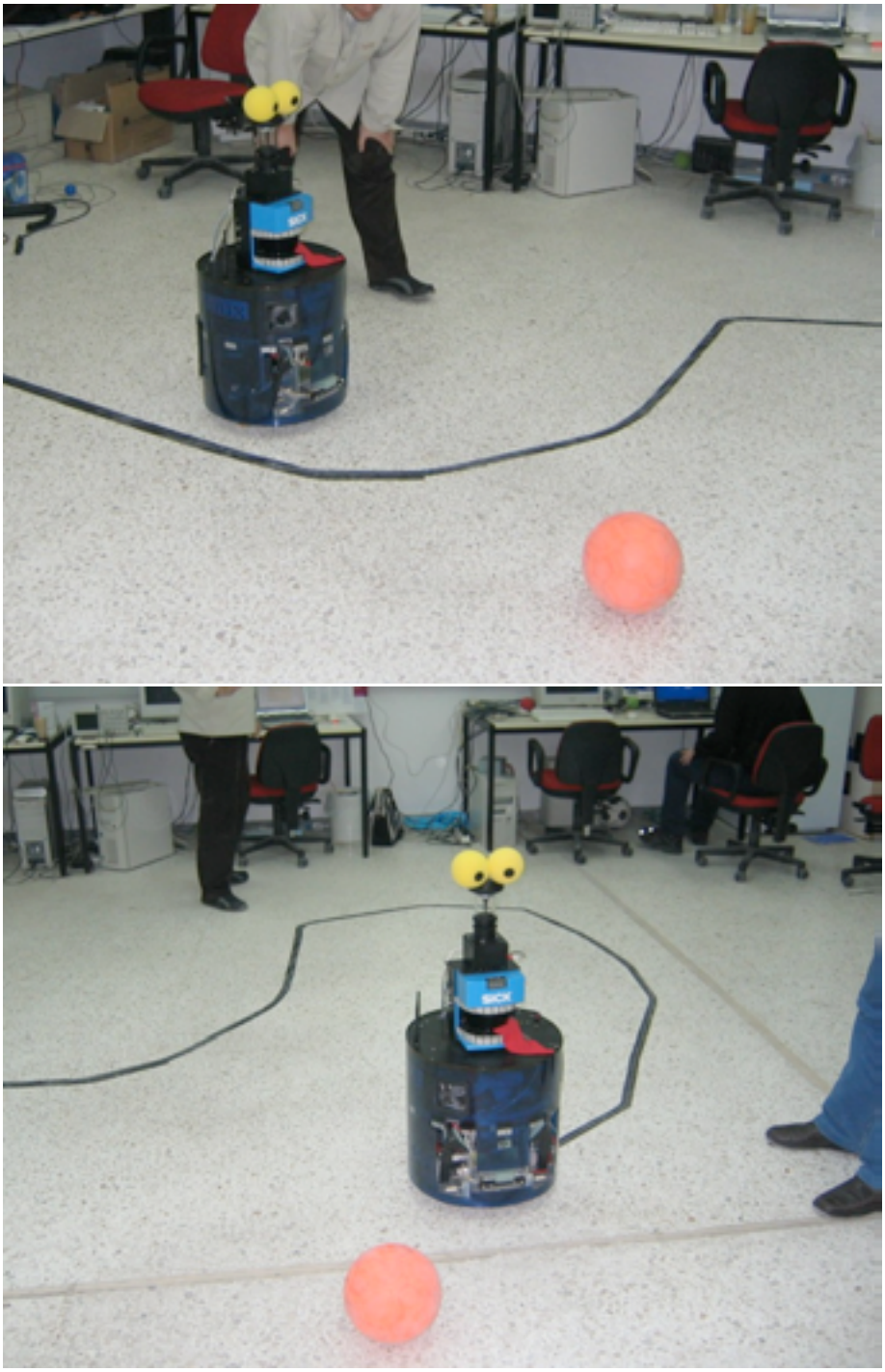


Figure 7.3: Robot chasing the ball

ning was necessary. The development of applications requires only an X-server on the user's machine and this only to examine the pictures taken by the camera, if used. The platform provided all other tools and even a desktop environment. Some small helper scripts were added to make the network facilities even easier to use, either wireless or cabled.

However, one important issue was encountered. The programmers were not aware of some of the more useful motion facilities of the robot. Their use demanded a change in philosophy and programming style, which is a little different than what was used for the main research robot of the institute. This particular one provides very limited and primitive commands for motion, so even a simple smoothly accelerated trajectory requires many commands and intermediate checks on behalf of the applications. The solution to this was the programming of an intermediate server that takes care of that, yet this uses much CPU power for the simplest tasks. In view of that, our platform was designed to offload the application programs from such concerns and encourages more sparse trajectory corrections, nevertheless these features remain to be accustomed to.

> This was fun, thought Ford, giving Colin a friendly pat. Colin was about the first genuinely useful robot Ford had ever encountered. Colin bobbed along in the air in front of him in a lather of cheerful ecstasy. Ford was glad he'd named him after a dog.
>
> THE HITCHHIKER'S GUIDE TO THE GALAXY

# Epilogue

The design and implementation of the robotic platform from scratch has been a tedious but rewarding procedure. It provided an insight into numerous topics of electrical, electronic, mechanical and software engineering , while keeping always in mind the demands, inherent problems and distinctive characteristics of robotics. The Foundation for Research and Technology provided all the necessary equipment and facilities to make hard things easy. Almost all the staff and the researchers in the Computational Vision and Robotics Laboratory in FORTH gave their invaluable suggestions and help to make the platform real. And for al these material and immaterial things I am deeply thankful.

The idea behind this prototype was to be able to build a team of robots to participate in the RoboCup competition and through that to research the field of cooperative embodied agents. During the development it became apparent that the platform should have a broader range of applications and become a companion to the research robots already used in the institute. The experience gained by researchers by working for years with them was very important so as to try to design something that would suit their needs better.

The objectives have been met to the extent possible, yet there is always room for improvement. The platform should now be used for what it was designed for and in the course get improved or become obsolete. Its major advantage over any commercial robot is the complete code and blueprint availability, so that any changes can be made at any level, either hardware, firmware or software. And even if not used as a whole, the modular design of the components allows for some functional blocks to be reused for other purposes. For example, the vision and the laser servers are completely autonomous and can be used on any other computer, while all the CAN bus modules

are readily completely functional through a conventional RS-232 serial port.

However, the robot was designed as a whole, and because of that it stands a good chance that it will be robust and perform well in its role as a quick and easy application environment for robotics research. I hope that it will get the opportunity to prove it and I believe that it will or else explode trying.

"Documentation is like sex; when it's good, it's very, very good, and when it's bad, it's better than nothing."

DICK BRANDON

# Appendix A

# Software

## A.1 Vision Server

### A.1.1 Configuration Files

Upon launching, the vision server searches in the directory `/etc/visionserver` for files containing the configuration parameters of each camera. The files are named `parameters-port-`**`m`**`-cam-`**`n`**, where **m** is the FireWire controller port number and **n** the camera number. If a file for a camera is missing, default values are loaded. Their format is plain ASCII and each one contains in any order some or all of the following options, terminated by newlines.

`angle_forward` *`float`*

Defines the angle between the upwards direction of the panoramic image and the forward direction of the robot. *f* is a floating point value in the set [0,360] expressing degrees and it is used to calibrate the rectified image so as to make its middle point on the horizontal axis coincide with the forward direction.

`center_x_coeff` *`float`*
`center_y_coeff` *`float`*

Defines the centre of the panoramic image. The arguments *`float`* are floating point values in the set [0,1] expressing the fraction of each coordinate, (1,1) being the bottom-right point and (0,0) the top left.

`radius_coeff` *`float`*
`radius_in_coeff` *`float`*

Defines the inner and outer radius of the useful part of the panoramic image. The arguments are floating point values in the set [0,0.5] expressing the radiuses of the circles as fractions of the picture height.

`brightness` *`integer`*
`exposure` *`integer`*
`gain` *`integer`*
`gamma` *`integer`*
`saturation` *`integer`*
`sharpness` *`integer`*
`shutter` *`integer`*
`white_bal` *`integer integer`*

These values define the default configuration parameters of the camera. They are all positive integers including zero and their interpretation is camera specific. The special value of $-1$ selects the automatic mode for a setting, if the camera provides it.

Next follows the configuration file of the currently installed Fire-i panoramic camera adjusted to cold artificial (fluorescent) light as a sample. Any lines beginning with a # are treated as comments by the server program.

```
#camera 1 parameter file
center_x_coeff 0.519
center_y_coeff 0.518
radius_coeff 0.475
radius_in_coeff 0.1
angle_forward 45
brightness -1
exposure -1
sharpness -1
saturation 90
gamma 1
shutter 1
gain 90
white_bal -1 -1
```

### A.1.2 Software Library

A C library is provided as the interface of the application programs to the vision server. As pointed out by the architecture of the server and its facilitation of shared memory, the application must reside in the local machine. There are 3 distinct stages that must be followed to initiate an image stream from a device: Connection to the server, configuration of camera device, and request for acquisition start. Two custom data types are defined and used to pass on option sets to the library functions. Some of the type members are set by the user and some by the library routines. All functions return 0 on success or a negative integer indicating an error condition. Some of the error conditions are only useful for debugging purposes so they are not described here, but are documented in the header files. A brief overview of the data types and the commands in order of their use is given below:

```
typedef struct
{
   int sockfd;
   int port;
   char *address;
}vconnection;
```

The `vconnection` type contains all the information to establish and maintain a TCP connection to the server. The user must define the IP or DNS address and the port of the server in this structure before connecting to the server. This may be a little superfluous, since the port is always the same and the IP address the local machine, yet this implementation allows for extension of the program to also handle remote clients in the future without altering the function calling scheme. Only one instance of this structure is needed by each application.

```
int vserver_connect(vconnection *)
int vserver_disconnect(vconnection)
```

This `vserver_connect` function establishes a connection to the server. Upon success, the application can configure a camera device.The custom data `shared_memory` type must be filled out with the required information before passing it to the function. The `vserver_disconnect` function disconnects from the vision server and terminates the connection handling thread.

```
typedef struct
{
```

```
\\These variables are read and written by the application program
   int device;
   float fps;
   int color_space;
   int x;
   int y;
   bool rectify;
\\These variables are only read by the application program
   unsigned char *offset;
   time_t timestamp
\\These variables are used by the library functions only
   unsigned char *shm;
   key_t key;
   int shmid;
   int flag_offset ;
   float w;
   int noi;
}shared_memory;
```

The application must define the device number, the frame rate and size and the colour-space of the stream. The variables `offset` and `timestamp` are used by the acquisition functions to pass on the address of the most recent frame to the application. One such structure is needed for each camera an application uses.

The use of these two types for each application to identify the connection it controls is a slight risk for the stability of the libraries. The application could modify by mistake the variables of the structures used internally and confuse the error handling routines. However, the server thoroughly check the incoming commands to ensure that it can't crash by commands issued by an erratic application. On the software library side, a more consistent implementation would be to identify each connection and device by pointer handles and store the library variables internally. The structure of the programs allows such an expansion easily when the architecture has proven itself to run smoothly and efficiently.

```
int configure_acquisition(vconnection, shared_memory *)
```

This function applies the configuration settings pointed out in the `shm`, if possible, and initialises the shared memory segment that will contain the image data. The values applied are not necessarily the ones requested by the user. Closest match values are applied to the size and frame rate settings if they are not supported by the camera,

while they remain unchanged if the device is already configured and used by another application. When rectification is requested, the x,y variables are set to the size of the rectified image. If the function exits with no other errors, the structure will hold the valid configuration of the requested device and the address of the shared memory segment. After this point , the acquisition can start. The address of the most recent frame is though not yet valid.

```
int start_aquisition(vconnection, shared_memory *)
int stop_aquisition(connection, shared_memory *)
```

Once a device is properly configured and the data types up to date, these functions are used to start and stop the image stream. These commands have no effect if the acquisition has already started and if another application still uses the device respectively.

```
int give_me_a_picture(vconnection,shared_memory *)
int release_buffer(vconnection, shared_memory *)
```

After the acquisition has begun, the first function validates the `offset` and `timestamp` variables in the `shared_memory` structure so that they point at the most recent image captured by the camera indicated by the device number. The buffer containing the image is reserved and will not be written to by the server until a subsequent image request or until the application explicitly releases the buffer by using the second function.

```
int change_camera_settings(vconnection, shared_memory *,
   int setting, int value)
int get_camera_settings(vconnection, shared_memory *,
   int setting, int *value)
```

These functions can be used at any time after a device has been selected and while it is acquiring or not. They control the same set of camera parameters that is defined in the configuration files, i.e. brightness, contrast etc. As regards the interpretation and the range of the values, the same rules[1] as in the configuration files apply.

## A.2 Motion Server

Although the appropriate software library is provided, the motion server can also operate completely independent of it. It provides a complete interface for remote command-line operation and users can connect to it by means of a telnet client and issue

[1] or absence of them

mnemonic commands to control the robot. RFC-1184 compliant clients like the ones found in almost all UNIX flavours have a complete console functionality, while not compliant ones like all Windows native implementations have everything but backspace functionality.

### A.2.1 Command Line Interface

The same limitations as to applications apply, so either many users can connect to the secondary TCP port 3971 or one to the primary port 3970 only if no other user or application is using it. Only on the second occasion has the user access to the privileged motion commands. Both the common and the the privileged set are replied using the same scheme. All replies have one of the following prefixes:

`ack:` This is the standard reply after each command is received, but before its syntax or access permissions have been checked.

`rep:` This is the prefix for replies to commands that request information.

`msg:` This is the prefix for error messages caused by the last issued command. These include syntax and access errors.

`err:` This is the prefix for error and warning messages issued asynchronously. These include voltage and current warnings and all critical errors.

The `err:` messages are broadcast to all connected clients automatically and indicate critical errors and warnings about the current state of the robot. Some of them are for debugging purposes and some can be useful to applications. These include:

`err:bat low` The battery voltage has dropped below 11.5 Volts. This is just a warning since the robot can still continue to function, but the voltage drop is very rapid below this point and some precaution actions should be taken.

`err:overcurrent` The current on one of the motors has exceeded 10 Amperes and the protective circuit has been activated. It indicates an error of medium importance and can happen rarely under normal circumstances if the acceleration value of the robot motion is very high and the motors are forced to output high torque. If this happens constantly then the motor gears may be blocked by obstacles.

`err:high current` This is a warning indicating that the motors are operating for more than 1 second at 90% of their maximum allowed power.

`err:motor` A critical error indicating that one motor is experiencing a problem that cannot be automatically recovered. When this error condition occurs, the motion controller is shut down and does not accept any commands. Normally caused by unconnected cables.

### A.2.1.1 Common Command Set

`help`

Prints an overview of all[2] available commands.

`kill`

This command stops the robotic platform immediately using the highest possible deceleration and suppresses the execution of all subsequent motion commands.

`enable`

Cancels a kill condition and re-enables the execution of motion commands.

`get bat`

Requests the battery voltage indication. The reply is of the form: `rep:bat=`*XX.X*, where *XX.X* is the voltage expressed in Volts.

`get pos`

Requests the odometry indication. The reply is of the form: `rep:x=`*X.XXX* `y=`*Y.YYY* p=θ.θθ, where x, y are the coordinates expressed in metres and p the orientation of the robot expressed in degrees.

`get vel`

Requests a velocity measurement. The reply is of the form: `rep:tang=`*T.TTT* `ang=` *A.AAA*, where `tang` is the tangential velocity of the robot expressed in m/sec and `ang` is the angular velocity expressed in degrees/sec.

`get cur`

Requests a motor current measurement. The reply is of the form: `rep:mcur=`*M.MM*

[2] toute? non! un petit...

`scur=S.SS`, where `mcur`, `scur` are the indications of the master and slave controller respectively expressed in Amperes.

| `get temp` |
|---|

Requests the temperature indication of the motion controller power electronics. The reply is of the form: `rep:mtemp=M.MM stemp=S.SS`, where `mtemp`, `stemp` are the indications of the master and slave controller respectively expressed in C$^\circ$ .

| `quit` |
|---|

Terminates the connection to the motion server.

### A.2.1.2 Privileged Command Set

#### Motion Commands

The following commands instruct the robot to follow a certain trajectory. They accept as arguments some of the following quantities in floating point notation:

*tangv*: Tangential velocity expressed in m/sec

*angv* : Angular velocity expressed in degrees/sec

*angle*: Angle expressed in degrees

*dist* : Distance expressed in metres

The combination of tangential and angular velocities define both the maximum velocity and the trajectory. The robotic platform will try on all occasions to accelerate on the defined path until it reaches the designated velocity, unless this is physically impossible or it leads to excess electromechanical stress taking into account its current motion.

| `mov(tangv angv)` |
|---|

Instructs the robot to move continuously at the given velocity.

| `mov fdist(tangv angv dist)` |
|---|

Instructs the robot to move at the given velocity, until it has covered the desired distance.

| `mov fang(tangv angv angle)` |
|---|

Instructs the robot to move with the given velocity, until it has covered a distance on its circular path whose arc equals the indicated angle.

`mov str(tangv dist)`

Instructs the robot to move straight until it has covered the desired distance. This command is a shorthand for `mov fdist(tangv 0 dist)`

`mov rot(angv angle)`

Instructs the robot to rotate in place (pivot) for *angle* degrees. This command is a shorthand for `mov fang(0 angv angle)`

`stop`

Instructs the robot to stop in a controlled fashion, meaning that it will try to decelerate smoothly on a straight path using the currently set acceleration/deceleration value. It is a shorthand for `mov(0 0)`.

`brake`

Instructs the robot to stop in a non controlled fashion. This command puts the motors in braking mode, transforming their kinetic energy into heat. This way both the electronics and the mechanical parts are stressed less, but the braking force is usually less than in controlled mode. Also there are no guaranties about the path that the robot will follow during braking.

`limp`

This command puts the wheels in low resistance mode. Useful when moving the robot by hand.

### Parameter Changing Commands

`set acc(accel)`

Sets the value for acceleration and deceleration. *accel* is a floating point value expressed in m/sec$^2$.

`set pos(x y θ)`

Resets the odometry to the given values. All parameters are floating point values, whereas *x,y* are expressing metres and θ degrees.

```
autocol on
autocol off
```

Turns on or off the simple automatic collision avoidance feature built into the motion controller. When activated and a collision is detected by the IR sensors, the robot stops immediately.

#### Ball Handling Commands

```
arm up
arm down
```

Raises or lowers the ball handling arm of the robotic platform.

```
kick(duration power)
```

Activates the kicker piston. Both parameters are floating point values ranging from 0 to 1. *duration* is expressed in seconds while *power* is the fraction of the maximum power.

### A.2.2 Software Library

The C library provided for communication with the motion server can be used by applications running on either the local or remote machines. The implementation is quite straightforward and corresponds completely to the command line functionality. Once initialised, a thread is launched to handle the connection, i.e. the transmission of the commands and the interpretation of replies and error messages. Errors and warning messages that occur asynchronously, like in case of low battery voltage or high current, are handled by default functions, which can be overridden by user-supplied ones. All functions are protected by watchdog timed mutexes as to avoid hangs by incomplete communication procedures.

The following data types are defined for use with the supplied functions:

```
typedef int IR;
```

The data type IR is used for holding the IR sensors' status. It is simply an alias for the type `int` and it uses the lower 12 bits only, each one representing the state of a sensor, either 0 when it's free or 1 when detecting an object. The numbering order is defined

with the lowest bit corresponding to the frontal IR and counting in a clockwise order when the IR belt is viewed from the top (See also 4.2.2).

```
typedef struct
{
   float x;
   float y;
   float pose;
}Pos;
```

The data type Pos is used to pass and get information about odometry to and from the functions. x, y are expressing metres and pose degrees.

```
typedef struct
{
   float lin_vel;
   float ang_vel;
}Vel;
```

The Vel data type is used hold the velocity readings. lin_vel expresses m/sec, while ang_vel degrees/sec.

As pointed out by the architecture of the server there are two sets of functions, one available only to the primary connection and a common one. The calling scheme is the same for both types of connections, not accessible commands are ignored by the server and an error message is printed. All functions return 0 on success or a negative integer indicating the error encountered. The error codes are defined and explained in the header files and are mainly for debugging.

### A.2.2.1 Common Command Set

```
int mserver_connect(char *host,int port)
int mserver_disconnect()
```

These two functions are used to connect to and disconnect from the motion server. After the connection is established, a handler thread belonging to the application is launched and is terminated by the mserver_disconnect function. The arguments to the mserver_connect function are a string containing the IP address or DNS name of the robot computer and the port number. The port number can be either PRIMARY or SECONDARY, which are aliases for 3970 and 3971 respectively and defined in the header file.

```
int kill_robot()
int enable_robot()
```

The `kill_robot` function stops the robot immediately and suppresses the execution of all motion commands. It is the only function available to the secondary connection that allows it to override the primary one. `enable_robot` resumes normal operation.

```
int getIR(IR *)
int getPos(Pos *)
int getVel(Vel *)
int getBattery(float *)
int getCurrent(float *master,float *slave)
int getTemp(float *master,float *slave)
```

These functions are used to request various readings from the server. They are completely analogous to the ones described in the command line interface and their arguments express the same quantities and scale.

### Error Functions

```
void set_motor_error_action(void (*action)(void))
void set_highcurrent_action(void (*action)(void))
void set_overcurrent_action(void (*action)(void))
void set_battery_low_action(void (*action)(void))
```

The functions stated above are used to change the default error handlers. They are invoked automatically each time one of the error conditions stated in A.2.1 occurs. The default handler for the critical motor error is to terminate the application while for the other warnings/errors is to print an appropriate message to `stderr`. The handler functions supplied by the user must be of type `void` *`functionname`*`()`.

#### A.2.2.2 Privileged Command Set

The following commands are available only to applications connected to the primary port. If the calling program uses a secondary port, the functions will return with EPERM.

##### Motion Commands

```
int move(float lin_vel, float ang_vel)
int move_fxd_dist(float lin_vel, float ang_vel, float distance)
int move_fxd_angle(float lin_vel, float ang_vel, float angle)
int move_straight(float lin_vel, float distance)
int rotate_inplace(float ang_vel, float angle)
int brake()
int stop()
int limp()
```

These functions are used for moving the robot. Each of them corresponds exactly to a motion command of the terminal interface.

##### Parameter Changing Commands

```
int setMaxAcc(float acceleration)
int setCurPos(Pos position)
int autostop(int on_off)
```

Used to set the acceleration/deceleration, reset the odometry and turn the auto-stopping feature on and off. The argument to the `autostop` function is either AUTOSTOP_ON or AUTOSTOP_OFF, defined in the header file as 1 and 0 respectively.

##### Ball Handling Commands

```
int move_arm(int state)
int kick(float duration,float power)
```

The ball handling commands are also completely analogous to the ones of the command line interface and accept the same values and quantities. The `move_arm` function takes as its argument the constants ARM_UP and ARM_DOWN, defined in the header file as 0 and 1 respectively.

## A.3 Laser Server

### A.3.1 Software Library

The C library that interfaces the applications to the laser server is similar is similar in function and use to the one used for the vision server, with the difference that the calling program needn't reside on the local computer. Here also, there are 3 distinct stages that must be followed to initiate an laser measurements stream: Connection to the server, configuration of the laser device, and request for measurements' start. Two custom data types are defined and used to pass on option sets to the library functions. All functions return 0 on success or a negative integer indicating an error condition and documented in the header file. The functions and data types are documented below in order of their use.

```
typedef struct
{
   int sockfd;
   int type;
   char *address;
}lconnection;
```

The `lconnection` type contains all the information to establish and maintain a TCP connection to the server. The user must define the IP or DNS address and the type of connection to establish, which is either SHARE_MEMORY or SOCKET.

```
int lserver_connect(connection *)
int vserver_disconnect(connection)
```

This `lserver_connect` function establishes a connection to the server. Upon success, the application can configure the laser measuring device. The custom data type `ldevice` type is used to hold the appropriate settings and must be filled out prior to starting the measurement stream. The `lserver_disconnect` disconnects from the vision server and terminates the connection handling thread.

```
typedef struct
{
\\These variables are read and written by the application program
   int angle_of_view;
   int angular_resolution;
   int distance_resolution;
```

```
\\These variables are only read by the application program
   unsigned short int *mtable;
   time_t timestamp
\\These variables are used by the library functions only
   unsigned short int *offset;
   unsigned  short int *shm;
   key_t key;
   int shmid;
   int flag_offset ;
   int noi;
}lconf;
```

The application must define the angle of view and the angular and distance resolution of the laser measurements. The angle of view can take two values, ANGLE_100 and ANGLE_180, for 100° and 180° field of view respectively. The angular resolution can be RES_1_DEG, RES_0_5_DEG or RES_0_25_DEG for 1° , 0.5° and 0.25° resolution respectively. The 0.25° resolution can be selected only in conjunction with the 100° field of view. The distance resolution can be MODE_MM or MODE_CM, which set the the granularity of the measurements to millimetres or centimetres respectively. The mtable variable when validated points to a table containing a set of subsequent distance measurements from right to left. Each one is two bytes long and the size of the table is $\frac{\text{angle}}{\text{angular\_resolution}} + 1$. The values are range from 0 to 8191 and express either millimetres or centimetres depending on the effective distance resolution.

```
int configure_laser(lconnection, lconf *)
```

This function applies the configuration settings pointed out in the lconf struct, if possible, and either initialises the shared memory segment that will contain the laser data or allocates the required memory when a socket connection is used. If the function exits with no errors, the structure will hold the address of the measurement table in the pointer mtable, which will be either a shared or a private memory segment, depending on the type of connection. After this point , the measurement stream can be initiated. The application must check the configuration settings before using them, as they are not applied if the device is already configured by another application and streaming.

```
int start_scan(lconnection, lconf *)
int stop_scan(lconnection, lconf *)
```

Once a device is properly configured and the data types up to date, these functions are

used to start and stop the measurement stream. These commands have no effect if the laser scan has already started and if another application still uses the device respectively.

```
int give_me_a_scan(lconnection,lconf *)
int release_buffer(lconnection, lconf *)
```

After the data stream has been initiated, the first function validates the `mtable` and `timestamp` variables in the `lconf` structure so that they point at the most recent distance measurement data set received. The buffer containing the data set is reserved and will not be written to by the server until a subsequent request or until the application explicitly releases the buffer by using the second function.

"...It's just an arbitrary set of rules like chess or tennis or, what's that strange thing you British play?"
"Er, cricket? Self-loathing?"
"Parliamentary democracy. The rules just kind of got there. They don't make any kind of sense except in terms of themselves..."

THE HITCHHIKER'S GUIDE TO THE GALAXY

# Appendix B

# RoboCup Middle Sized League Competition Rules

## Preamble

### Rules Philosophy:

1. RoboCup rules should not in any way describe the behaviour of how the game is played. Rules should only ensure that a fair competition takes place, and encourage both technical and creative development.

2. RoboCup rules should avoid to constrain the design of robots, including their mechanical construction, their use of sensory systems, communication equipment, etc., unless the constraints seem necessary to foster scientific progress or to ensure a fair competition.

   Example constraint: Global vision systems are not permitted in the middle size league.

### Design Philosophy:

1. Each team should design their robots without making interpretations or placing expectations on how the environment around the field will look like, about spectators, what other teams will do, what robots should look like, or how they will behave. Expecting the environment around the field, spectators or other teams to comply with your own interpretations should be avoided.

2. Each team is under no obligation to accommodate modifications to their own robots to suit other teams. Any such modification is by mutual consent only.

# B.1 RoboCup Law 1 – The Design of Robots

## B.1.0 Preliminaries: Design Guidelines

Robots for playing soccer should be designed such that they are **both robust and safe.**

- **Robust** means that the physical integrity of the robot is not endangered by incidental, accidental, or intentional collisions with objects of the field or other robots, and that the robot's sensing systems and software can handle potentially significant levels of noise caused by other sources, such as other robots, game officials, team members, spectators, or the media.
- **Safe** means that robots do not damage other robots. In particular, the design of the robots should ensure that "Robot Fouls and Misconduct" are avoided. Robots that threaten to seriously damage opponents may be excluded from play in a tournament by the league organizing committee.

## B.1.1 Size of Robot Players

The size of each robot player must obey the following two constraints :

1. The robot must possess a configuration of its actuators, where the projection of the robot's shape onto the floor fits into a square of size 50 cm $\times$ 50 cm.
2. Every robot may not have configurations of its actuators, where the projection of the robot's shape onto the floor does not fit into a square of size 60 cm $\times$ 60 cm.

   The robot should be in the configuration that fits within the 50 cm $\times$ 50 cm square for the majority of play time, and only occasionally (for instance, when kicking) exceed this limit (up to the 60 cm $\times$ 60 cm limit).
3. The robot height must be at least 30 cm and at most 80 cm.

## B.1.2 Weight of Robot Players

The maximum weight of a robot is 80 kg.

## B.1.3 Shape of Robot Players

Any shape is allowed as long as the following two constraints are met:

1. The size restrictions in Clause B.1.1-2 are not violated.

2. The robot passes the following checking procedure:
   Put an elastic strip (or string) around the robot body. Take a piece of paper obtained by cutting a paper circle, of the same diameter as the ball, perpendicularly to a diameter, at 1/3 of the distance from the border (e.g., for a 25 cm ball, cut at 8.3 cm). If there is a way of inserting the piece of paper, kept parallel to the ground, between the elastic strip and the robot body, with the circular part in contact with the robot body, then the robot shape is not admitted.

   If a robot can change its shape, the size AND shape constraints should never be violated.

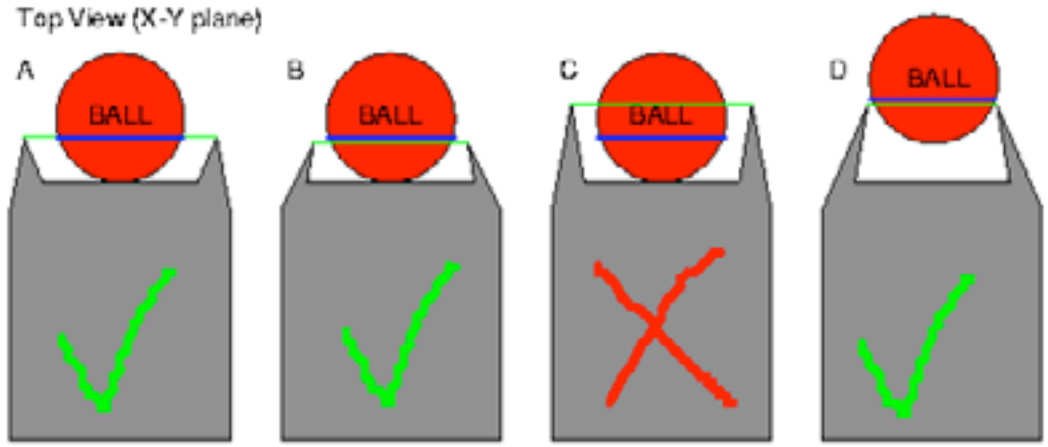



Figure B.1:

- Situation A shows, in symbolized form, a robot shape used by various teams. This will be okay, if the concavity in the front of the robot is small enough such that at most one third of the ball's diameter (thick blue line) is inside of the robot's convex hull (thin green line).
- Situation B illustrates that the form of the concavity does not matter. For example, fingers may point inwards.
- Situation C shows a robot that does not meet shape restrictions. It will be excluded from playing in tournaments.
- Situation D illustrates that the actual depth of a concavity is irrelevant. All that matters is how deep the ball will fit into a concavity.

- Situation K shows a legal shape for a circular robot with a concavity.
- Situation L is obviously not permitted.

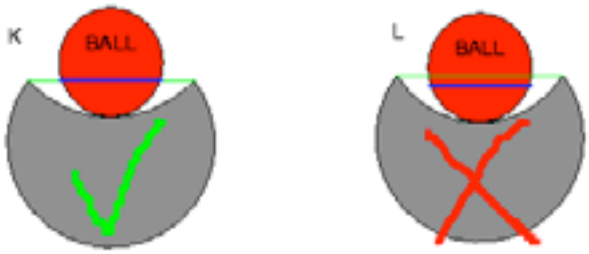



Figure B.2:

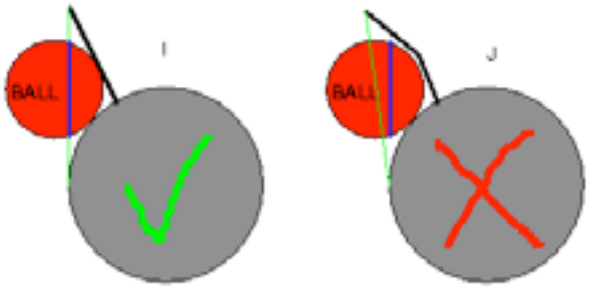



Figure B.3:

- Situation I illustrates how the concavity will be measured on a circular robot with an arm-like extension. This robot will be okay.
- Situation J violates shape restrictions.

- Situation E illustrates the shape situation from a side view and shows a robot that will be permitted.
- Situation F demonstrates an illegal situation.

  This figure illustrates a robot with a somewhat ambiguous shape.

- Situation G is obviously okay.
- Situation H will be interpreted as ball holding. The usual sanctions for ball holding will apply, including the sending-off penalty (red card) for committing this offense repeatedly. Furthermore, a robot that repeatedly gets into this situation may be excluded from further play in the tournament.

### B.1.4 Color of Robot Body

*(Omitted)*

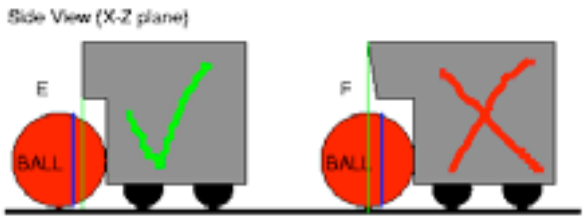



Figure B.4:

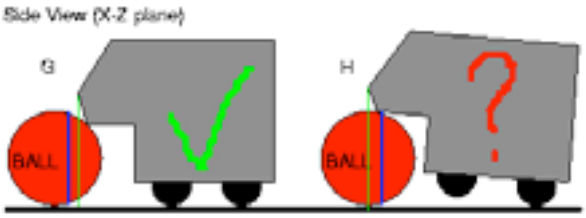



Figure B.5:

## B.1.5 Markers of Robot Players

*(Omitted)*

## B.1.6 Ball Handling Mechanisms

The robot players may exert force onto the ball only by direct physical contact with one of their body parts.

A robot may travel with ball (dribble) as long as the contact point between robot and ball is constantly changing.

In any situation, another robot can easily take possession of the ball when a robot controls the ball.

Rule B.3.2 Ball Holding defines which kind of activity can be done with the ball. In any case, shape and dimension constraints should be respected.

## B.1.7 Communications

Communication with or between the robots of a team and remote computer systems is allowed.

Wireless communication equipment shall follow legal regulations of the country where the tournament is held.

Wireless communication equipment satisfying IEEE 802.11 specifications is allowed.

Radio communication in the 418 MHz and 433 MHz bands are explicitly disallowed.

For any other kind of equipment, permission for use of that equipment must be obtained by the Middle Size League Technical Committee at least three months prior to a tournament.

Global cameras, other global sensing systems, and human intervention are not allowed.

Note: A team using a wireless communication is recommended to contact the tournament committee. In order to avoid interference, a team is recommended to select two carrier frequencies before the match, or to get other teams' information of frequencies from the tournament committee.

### B.1.8 Global Vision System

Global vision systems cannot be used to provide input to machines (either robots or ground stations) involved in the game during the game.

### B.1.9 Sensing Systems

Any sensing system is allowed as long as the following constraints are met:

1. All parts of the sensing system (i.e .the actual sensing device and, if applicable, a signal emitting device) must be on the robots.
2. There may be no manipulation of the environment, such as placing specific markers as landmarks.

   Note: Robot players may safely use as landmarks the objects of the field, including any coloring and texture of goals, corner poles and border poles around the field.

### B.1.10 Technical Inspection and Modification of Robots

*(Omitted)*

### B.1.11 Infringements/Sanctions

A robot that violates any of the constraints set forth in this law will be excluded from participation in the tournament.

Before the start of a tournament, the organizing committee will ensure that all participating robots meet the constraints set forth in this law.

In exceptional cases, the organizing committee may permit robots violating particular constraints, provided that all team leaders agree.

## B.2 RoboCup Law 3 - Robot Fouls and Misconduct

The following actions by robot players are forbidden.

### B.3.1 Damage to the Field, the Ball, or Other Robots

No robot may threaten spectators and/or damage the field and/or other robots.

A robot causing damage will be removed from the field for the remainder of the game.

A robot that caused damage may be excluded from further participation in the tournament by the organizing committee.

### B.3.2 Ball Holding

Holding a ball means taking full control of the ball by removing all its translational degrees of freedom; typically, by fixing a ball to the body or surrounding a ball using the body to prevent accesses by others. Other robots must be able to easily take possession of the ball when a robot controls the ball.

Stopping the ball means holding the ball for less than one second.

Stopping the ball is allowed. Holding the ball for more than one second is not allowed for any robot.

Refer also to "Clause B.1.6 Ball Handling Mechanisms".

Robots that are designed such that ball holding is certain or very likely to take place will not be permitted to play. If a robot accidentally gets into a situation where it holds the ball during play, a foul will be called, the player will be cautioned (shown the yellow card), and a free kick will be awarded to the opponent. When holding the ball repeatedly, a player will be shown the red card and removed from the field for the rest of the game.

### B.3.3 Prohibited Behavior

*(Omitted)*

### B.3.4 Charging (Attacking Other Robots)

At all times, the behavior of robots must be such that damage to other robots is avoided.

All robots should feature equipment to detect situations of contact with other robots (direct charging situations). The obligation to detect charging situations includes indirect contact with another robot through the ball.

*(Omitted)*

### B.3.5 Jamming

During the match any robot shall never jam communication and sensor system of opponents. The usage of equipments which may cause interference of communication or sensors should be negotiated between two teams before the match. If a team uses

communications and sensors other than those previously declared to the tournament committee and/or the opponent, the game may be forfeited; the tournament committee will take this decision.

Half-read books and magazines nestled amongst piles of half-used towels. Half pairs of socks reclined in half-drunk cups of coffee...

THE HITCHHIKER'S GUIDE TO THE GALAXY

# Bibliography


[1] Rolf Pfeifer and Christian Scheier. *Understanding Intelligence*. MIT Press, 1999.

[2] International Standards Organisation. *Road vehicles – Controller area network (CAN) – Part 1: Data link layer and physical signalling*, 2003.

[3] CiA (CAN in Automation). *CAN Specification 2.0, Part A*, 2001.

[4] MSL Technical Committee. *Middle Size Robot League Rules and Regulations* , 2004.

[5] Ansgar Bredenfeld et al. Description of the GMD 2000 RoboCup team. Technical report, GMD – Institute for Autonomous Intelligent Systems (AiS), 2000.

[6] Thilo Weigel et al. CS freiburg – doing the right thing in a group. Technical report, Universität Freiburg, 2000.

[7] Mansour Jamzad. CS-sharif ROCS99 team in middle-sized robots league. Technical report, Sharif University of Technology, Iran, 1999.

[8] Walter H. Johnson and Rob Franklin. Effective ball handling and control in robot soccer. Technical report, Dept. of Electrical and Computer Engineering, Brigham Young University, 2002.

[9] Bernard Friedland. *Control System Design: an Introduction to State Space Methods*. McGraw–Hill, 1986.

[10] King Sun Fu. *Robotics: Control, Sensing, Vision and Intelligence*. McGraw–Hill, 1987.

[11] Yoram Koren. *Robotics for Engineers*. McGraw–Hill, 1985.

[12] Peter H. Stone. *Layered Learning in Multiagent Systems: a Winning Approach to Robotic Soccer*. MIT Press, 2000.

[13] Robert L. Hoekstra. *Robotics and Automated Systems*. Cincinatti: South–Western Publishing Company, 1986.

[14] Albert Kloss. *A Basic Guide to Power Electronics*. John Wiley & Sons, 1984.

[15] Kjeld Thorborg. *Power Electronics*. Prentice Hall, 1988.

[16] Richard Valentine. *Motor Control Electronics Handbook*. McGraw–Hill, 1998.

[17] Andrew S. Tannenbaum. *Operating Systems Design and Implementation*. Prentice-Hall, 1987.

[18] Philips Semiconductors. *The* $I^2C$*-bus and how to use it (including specifications)*, 1995.

[19] Brian W. Kenighan and Dennis M. Ritchie. *The* C *Programming Language*. Prentice Hall, 1988.

[20] Kirk Zurell. C *Programming for Embedded Systems*. R&D Books, 2000.

[21] Jean J. Lambrosse. *Embedded Systems Building Blocks*. R&D Books, second edition, 2000.

[22] Maurice J. Bach. *Design of the* UNIX *Operating system*. Prentice Hall, 1986.

[23] Gary Nutt. *Kernel Projects for* Linux . Addison Wesley, 2001.

[24] Piyush K. Rai, Kamal Tiwari, Prithwijit Guha, and Amitabha Mukerjee. A cost-effective multiple camera vision system using FireWire cameras and software synchronization. In *10th Annual Conference on High Performance Computing*, 2003.

[25] Microchip Technology Inc. *Application Notes 212, 215, 228, 234, 578, 588, 660, 713, 733, 734,735, 736, 738, 739, 826, 849, 853, 851, 910*.

[26] Microchip Technology Inc. PICmicro 18C MCU *Family Reference Manual*, 2000.

[27] Microchip Technology Inc. PIC18FXX8 *Data Sheet*, 2003.

[28] SGS Thomson. *Load Current Sensing in Switchmode Bridge Motor Driving Circuits*, 1995.

[29] SGS Thomson. *Driving* DC *Motors*, 1995.

[30] Fairchild Semiconductor. *Introduction to Power* MOSFET*s and their Applications*, 1998.

[31] Robert Bringhurst. *The Elements of Typographic Style*. Hartley & Marks Publishers, second edition, 2002.

Typeset with XeLaTeX using the Myriad, Futura, Univers, Utopia
and Lucida Sans Typewriter font families