Fast Autonomous Mobile Robot Platform

by

JC Wolf

A report submitted to the University of Plymouth

In partial fulfilment for the degree in

B Eng (Hons) Robotics and Automated Systems

2003-2004

SUMMARY

This report gives an introduction to the theory of fast mobile robots and focuses on problems involved and gives different approaches to practical implementation. In order to complete the project, an autonomous mobile robot was designed, built and tested. The designed robot has the approximate dimensions of a wheelchair, to enable it to navigate around buildings. It provides a platform for future development on navigation systems. It has the capabilities to be turned into a complete service robot. Future areas of development have been identified. An obstacle avoidance system has been successfully implemented in order to demonstrate these. Its maximum speed is 1.5 meter per second.

An investigated into different areas of research, such as fast sensing techniques for ultra sonic distance measurement, trajectory generation for fast movements and obstacle avoidance, was carried out. Tests have been completed, in order to confirm the trajectory generation theory.

The report goes into detail about the circuit design. For the completion of the project a modern micro controller module was designed. Due to the flexibility of the design, the module provided the base for other student projects, as well this one.
The power electronics design provides the DC Motors with up to 30 Amperes per channel. This maximum of power for a circuit that does not require expensive specialised components.

The project was successfully completed in time. It is part of the final year, in the undergraduate degree course BEng Robotics & Automated systems.

ACKNOWLEDGEMENTS

The author is grateful to many of his fellow students who indirectly contributed to this work.

Appreciation also goes to my supervisors Dr. Guido Bugmann, Paul Robinson, Dr. Genhua Pan and lecturers Alan Simpson, Dr. Mansel Davies, Dr. Phil Culverhouse who supported me the last 3 years. Furthermore, I received excellent technical support from David and Rodger Bowers, Bob Bray, Dipl. Ing. Peter Luther, Phil Hall and Phil Brown.

Finally the author wishes to pay tribute to his family. My parents (Bernd and Maria Wolf), who supported me financially and encouraged me in many ways. Special thanks go to my beautiful wife Soo-mi Yang, for her support and tolerance in many ways. Without her moral support this project could **never** have been accomplished.

J.C. Wolf
University of Plymouth
April 2004

TABLE OF CONTENTS

1. INTRODUCTION

Service robots are not yet off-the-shelf products. Why ? Because there are subjects that need researching an improving until the first real service robot can go into production.

One requirement of service robots would be to integrate into a human environment by moving and navigating at the same speed as humans do. Current autonomous mobile robots are usually moving slow or even stop to take measurements. That would be a hindrance for other people passing. Slow robots are inefficient or even useless, since the user might rather do the job himself, if it is quicker. This project is about proposing a robot platform that can navigate fast and therefore making mobile robots more useful for real world applications.

The author gained experience and knowledge about mobile robots in many different projects including: Robot football, The Melexis trophy and others. More information about these projects can be found on the homepage Wolf J C (2004): www.swrtec.de. When appropriate this knowledge of these projects contributes to this report. Especially robot football, where fast movements are essential, theory can be developed and tested and transferred to natural environment problems.

2. THEORY OF FAST MOBILE ROBOTS

In this chapter the underlying theory and principles for mobile robots will be explained. Particular focus will be given to advantages and disadvantages that arise when the theory is applied to fast robots.

2.1 Digital Controllers

Most robotic application use digital controllers, therefore the basics of digital control should be given and the impact of its advantages discussed.

Some of the most important advantages of digital control: Kuo B.C. (1992)

- Digital components are less susceptible to aging and environmental variations.
- They are less sensitive to noise and disturbance.
- Digital processors are more compact and lightweight. (fit inside mobile robots)
- They are more flexible and powerful.
- The costs of Microprocessors and Digital Signal Processors are continuously going down.

However there are also disadvantages:

- Limitations on computing speed and signal resolution due to the finite wordlength of the digital processor. (In contrast, analogue controllers operate in real time and the resolution is theoretically infinite.)
- The finite wordlength often translates into system instability.
- The limitations on computing speed causes time delays in the control loop which may cause instability in closed-loop systems.

2.1.1 Sampling and quantization

Most control systems contain digital as well as analogue signals. Therefore analogue to digital conversion (AD) and vice versa (DA) play an important role. A robot might have servos with A to D converter or shaft encoders on the wheels. In this cases sampling and quantization is required.

2.1.2 AD Conversion

An analogue value is taken at a time instant.

Since the length of a digital number is finite, the analogue value has to be rounded. This process is called quantization. The minimum step height or the closest value it can round to (1 bit), is called the quantization level. The output of a A to D converter stays constant for a time T until a new value has been converted and is taken by the controller. The time T is called sampling time. The processor usually computes an output by going through the control algorithm within one sampling period. This means that the output is at least T delayed with respect to the input.

2.1.3 DA Conversion

A digital word is converted into a corresponding proportional analogue level. The minimum step with of one quantization level is limiting the accuracy again. Fortunately digital to analogue conversion can happen almost instantly. However, the output of a DA converter is usually held until a new value is converted. This process is modelled as a zero-order hold in analysis and block diagrams.

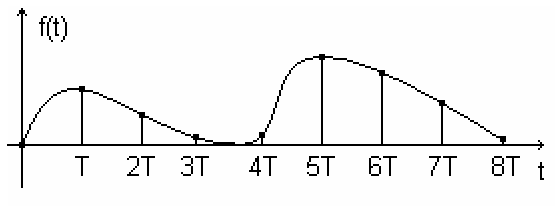


Fig 1. Sampled data at instants of kT

2.1.4 Sampling frequency

The sampling frequency ω_s must be chosen in a way that the controller can respond to changing in input values ω_v, disturbances ω_w and the controller and plant function bandwidth ω_b. The Nyquist-Shannon sampling theorem states that the sampling frequency must be at least twice the signal frequency. In practice this would not give a smooth controller response. Therefore, as a rule of thump, ω_s should be chosen to satisfy the following equations:

$$T < \frac{T_{SE}}{10} \tag{1}$$

$$\omega_s > 10\omega_n \tag{2}$$

Where T_{SE} is the settling time, ω_s the sampling period and ω_n the natural frequency of the system.

Usually controllers are used to control motors in mobile robots. The mechanical motor time constant is related to ω_n and in the order of a few hundred milliseconds. Therefore the practical sampling frequency is not very high. (say 50Hz – 1000Hz). If the robot is controlled through vision this becomes a problem, since computers are (at present) not powerful enough to provide a vision systems with a frame rate of 50-1000Hz. Other sensors, such as shaft encoders however do not take much CPU power and the sampling frequency can easily be achieved.

2.1.5 Delay

Delays occur during signal transmission and processing of sensor information in a digital control system. Delays can lead to instability and must be kept to a minimum. If a delay can not be reduced by improving the CPU power or other hardware, predictive control methods can compensate for delays to some extend. Predictive control methods often require a model of the controlled process. This causes the controller to be more complex and that is the reason why it is not so common. In the case of fast robots, delays are critical. A decrease of delay increases controller gains and can therefore improve the transient response.

2.2 Trajectory Generation

There are many ways to generate a trajectory (path) for a robot to follow. Usually the input data to a trajectory generator is an initial position, a target position and possibly intermediate waypoints or obstacles. A trajectory generator is a guidance system. Guidance systems are usually a subsystem of a navigation system concerned about guiding a robot to a target point, given to it by the navigation system.

Fast mobile robots are required to move as fast as possible. Therefore the vehicle speed and controller gains are pushed to the limits. If the demanded trajectory is beyond the systems capabilities the following problems may occur:

- On this huge input demand non-linear ties, such as slip and saturation, can make the system unstable and the robot spins out of control
- the robot can not follow the required path and may hit an obstacle

Therefore there is a need to generate a trajectory that matches the vehicle dynamics.
Here is an overview of trajectory functions that have been investigated their basic concepts will be introduced and the advantages and disadvantages discussed.

2.2.1 Arcs

Arcs are part of a circle. A path can be generated by calculating a circle where initial and target positions are tangential points of the circle. The calculated radius is directly related to the wheel speed and steering of the robot. The resulting velocities of each wheel have fixed ratio, which is an advantage because it simplifies calculations.
The tangential acceleration a_t can be easily limited within the trajectory generation algorithm, which would otherwise happen afterwards in the robot controller. This increases the accuracy of the path following.

$$a_t = -\omega^2 r \tag{3}$$

$$\omega = \frac{v}{r} \tag{4}$$

combined:

$$|a_t| = \left| -\frac{v^2}{r} \right| \tag{5}$$

$$v = \sqrt{|a_t| r} \tag{6}$$

The radius r is known from the trajectory generator. The acceleration $|a_t| \leqq K$ is limited to K and the forward velocity of the robot v is usually a robot controller input and can be set as required. If the robot is required to move in a straight line the radius would be infinite. In this case the algorithm must make an exception.

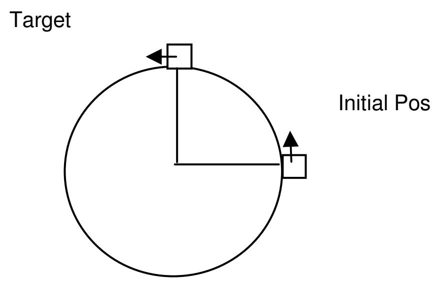


Fig 2: Arc trajectory

Here is an example where the author uses a circle for approaching a target point from 0 degree in order to continue in a horizontal trajectory after. The example is taken from robot football. The robot first aims for i1 with line of sight guidance. After passing i1 the robot is tangential to the calculated circle and can enter the circle at i2. It will exit the circle when colliding with the ball and continue with straight line motion and a slight acceleration in order to dribble the ball towards the goal.

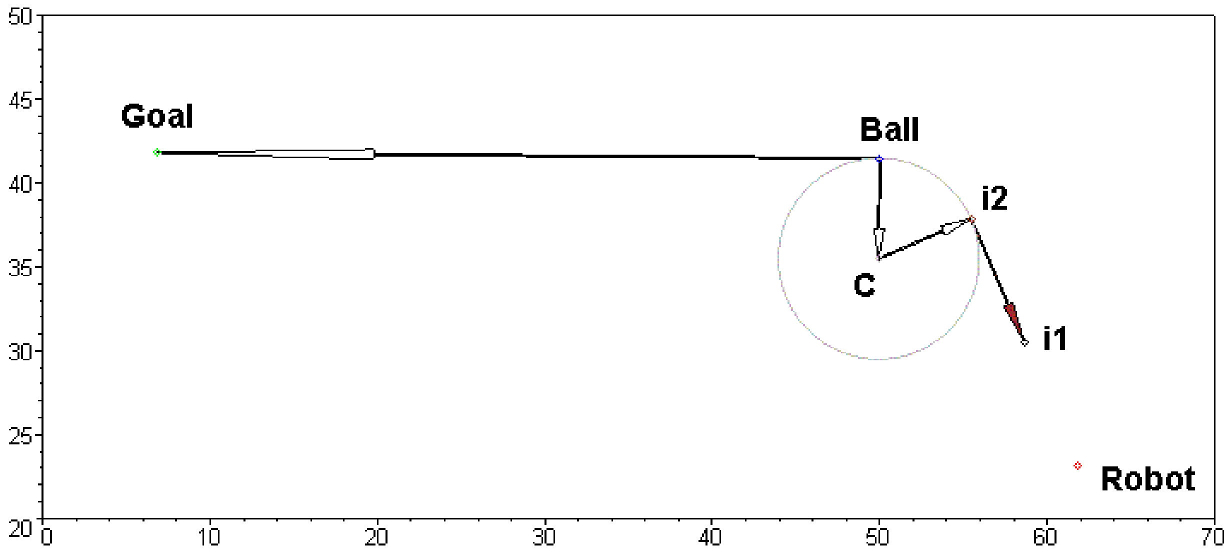


Fig 3: Circle and straight line move trajectory applied to robot football

This strategy has been implemented in one of the worlds most successful robot football strategies: University of Korea and KAIST, see B-J Lee et al (1999). It has also been tested by the author. Figure 3 shows data from a MAPLE file containing data from a real game situation. The mayor disadvantage of this method is that obstacle avoidance and other changing factors in a dynamic environment are difficult to combine with the rigid straight-circle-straight structure. The advantages are the simplicity, making trajectory-robot dynamics matching easy and prediction of movements accurate.

2.2.2 Bezier curves

A Bezier curve can be defined as a curve that is tangential to two straight lines. The first straight line is chosen to go through the initial robot position and the second is chosen to go through the target position. The first line has the orientation of the robot. The orientation of the second line will determine from which side the robot approaches the target (final posture). This is very useful property of this kind of trajectory generation. The major disadvantage is the mathematical complexity of Bezier curves, making it difficult to create trajectories that a robot can follow at high speed. The author conducted experiments with Bezier curves where the robot was requested to aim for waypoints along the curve. As soon as the robot comes near to the waypoint, the next waypoint will be selected.

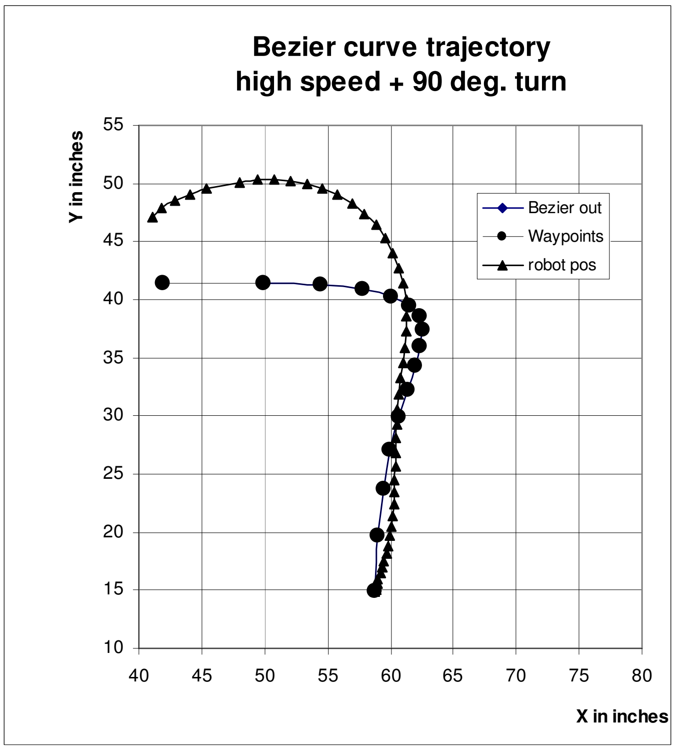


Fig. 4: Bezier curve trajectory.

The robot is required to turn 90 degree during its move. It starts off at (x,y) = (57,15). The line with the round dots defines the generated trajectory. If the relationship between a sudden change in direction and robot velocity is not balanced, then the robot will overshoot too much.

The path that the robot actually went is shown with small triangles.

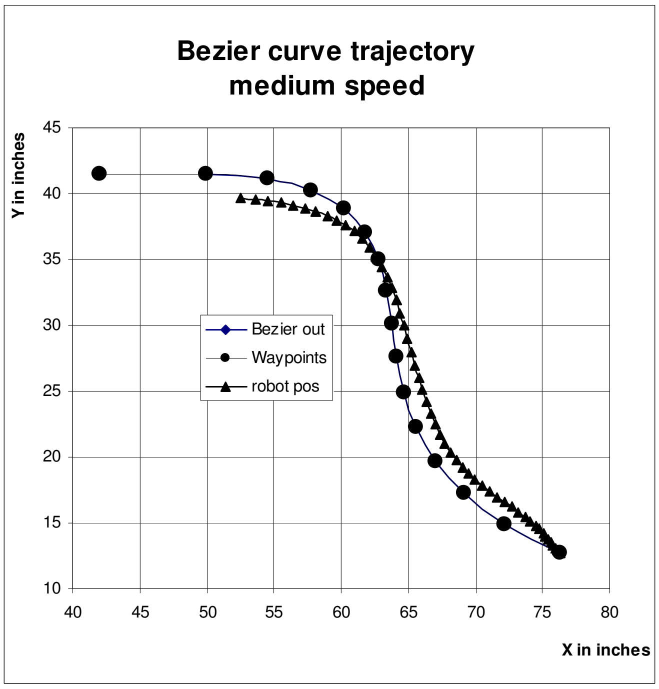


Fig. 5: Bezier curve trajectory

In this experiment the difference between its starting point orientation and final orientation is smaller than in the first experiment. The robot starts off at (x,y) = (77,12) The speed has been reduced. Now the robot has no problems following the requested trajectory quite accurately.

2.2.3 Straight line guidance

A robot can move to a target point by turning towards it and then move to it in a straight line. Assuming a wheeled robot, turning without a forward velocity is only possible with differential steering. The main advantage of the system is that it is simple and accurate.

The major disadvantage is the speed, since the robot must stop to turn.

2.2.4 Line of sight Guidance

Also called missile guidance is related to straight line guidance, but this time the robot moves while lining up with the target direction.

If the vector $\overset{\rightarrow}{rt}$ is pointing from robot towards target, then its angle $\theta = \angle \overset{\rightarrow}{rt}$ indicates the heading angle the robot should be going towards.

2.2.5 Vector fields

A vector-field is essentially a 2-Dimentional field with vectors. A vector consists of magnitude and angle, which represent importance or speed and demanded heading angle respectively.

The magnitude interpretation as a an importance is useful when vector fields are combined by addition of each vector. The more important vector is longer, and therefore the resulting heading angle will be more into the direction of the more important vector.

Some vector field generators, such as basic potential field methods, are not concerned about the magnitude. In this case the magnitude is often normalized.

2.2.5.1 Potential fields

The theory of potential fields as trajectories is derived from an electrical field of a sphere in physics.

The formulae for an attractive field is as follows:

$$\vec{V_t} = \frac{\vec{T} - \vec{R}}{\left|\vec{T} - \vec{R}\right|} \qquad (7)$$

where

$\vec{T}$ is a vector from the origin to the target position

$\vec{R}$ is a vector from the origin to the robot

$\vec{V_t}$ is the resultant vector with normalized length, indicating the direction of the field at the robots current position.

The resultant is pointing towards the target. The attractive potential field is therefore related to line of sight guidance.

A repulsive field is generating vectors pointing away from T. The formulae is

$$\vec{V_t} = \frac{\vec{R} - \vec{T}}{\left|\vec{R} - \vec{T}\right|} \qquad (8)$$

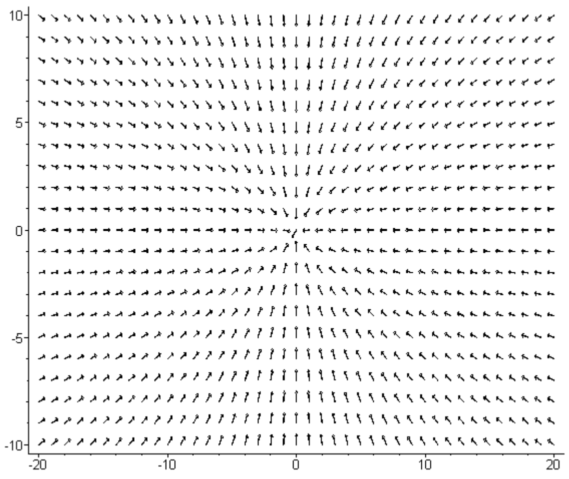


Fig 6: attractive potential field

Figure 6 shows a attractive potential field with a target point at T(0,0) . In an application usually only the vector at the current position of the robot is calculated, for demonstration graphs such as Figure 6 the robot is assumed to be in every possible position in the field, and therefore generating the vectors at each point.

2.2.5.2 Limit cycle based vector fields

Limit cycles are part of nonlinear control theory. However the properties of a graph representing a limit cycle, Figure 7, can be adopted for path generation. For further reading see D-H Kim (2000). The limit-cycle characteristics of the 2nd order nonlinear function can be represented as a vector field containing a unit circle. Vectors outside the circle will be directed tangentially onto the circle. It can be seen as an arc/circle trajectory generator that lines up the robot coming in from any direction automatically. The resulting vector-field can be used like a arc trajectory generator or for obstacle avoidance.

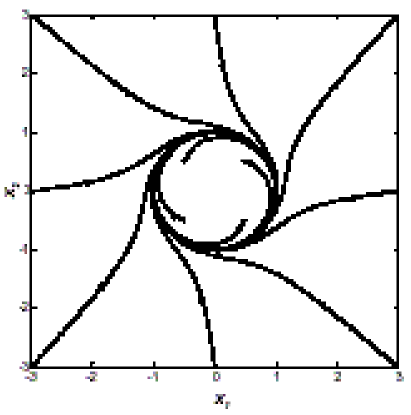


Fig 7: limit cycle

The disadvantage of the limit cycle method is that once the robot crosses the unit circle, the vector pointing towards a singularity in the centre. Therefore, a practical implementation is not easy, since the robot is likely to overshoot the circle border slightly when arriving at the circle.

A modification of the field within the circle is a proposed solution to the problem.

2.2.5.3 Vector field fusion

All discussed vector field methods can be applied at the same time. The author developed a way of combining (fusing) vector fields, which is published in Robinson P. (2004).

Constraints and requirements:

- Two or more vector fields are given
- These vector fields contain normalized vectors

The method is best described in an example. A typical combination of vector-field shall be analyzed where a Robot R avoids and obstacle Robot O on the way to a target point T. See figure 8 below.

A weighting function is required to fuse the vector fields together. Experiments have shown that the Gaussian normal distribution function is an acceptable method of combining these fields. (A cylinder or cone would create a sudden change in heading angle and excites instability.)

The angle α is the difference between the instant heading angle θ of the robot and the vector $\overset{\rightarrow}{ro}$ which points from robot to obstacle.

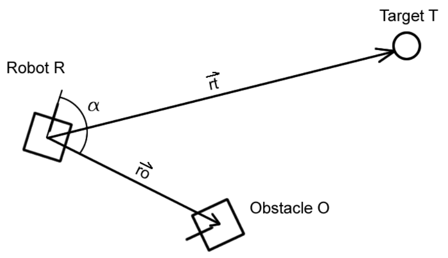


Fig 8: avoidance scenario

$$\alpha = a\tan 2(\overset{\rightarrow}{ro}) - \theta \qquad (9)$$

Thus α is an indication of how much the robot is on collision course with the obstacle. The smaller the angle, the more it is on collision course and the importance to avoid the obstacle is high.

The mission of the robot is to go to T. In order to take into account the obstacle on its way towards the target it must consider how close the obstacle is. The distance to the obstacle is defined as $\vec{ro}$. A smaller distance to an obstacle means that is more important to avoid it.

An avoidance vector field $\vec{V_O}$ shall be defined which is normal tot he mission vector field $\vec{rt}$. The normalized target vector is $\vec{V_T}$.

$$\vec{V_T} = normalize(\vec{rt}) \tag{10}$$

Suppose two vectors $\vec{V_T}$ and $\vec{V_O}$ are added together – 'fused' - with a Gaussian weighting function m*G(d).

$$\vec{V_{MT}} = \vec{V_T} + mG(|ro|, \mu, \sigma)\vec{V_O} \tag{11}$$

Where:

$\vec{V_{MT}}$ is the resultant modified target vector

m is a additional constant weighting factor

$G()$ is the Gaussian distribution function.

μ is the offset of the Gaussian hat

σ is the distribution of the Gaussian hat

We just learned that there are essentially two factors that define how important it is to avoid the obstacle.

α and $|ro|$.

The author will base the principle of vector field fusion by relating the length of each vector to importance towards the mission at a particular point in the field.

Thus α and $|ro|$ can be modelled as follows to influence the length of $\vec{V_O}$.

$$\mu = -r_1 2(1 - \frac{1}{1 + e^{-\frac{\alpha}{\tau}}}) \tag{12}$$

r_1 is the maximum offset that α can cause.

τ is steepness of the slope (relationship of μ and α)

A larger τ will result in higher angles already to be considered as important.

And the distance of the robot to the obstacle $\vec{ro}$ is modelled as the position parameter in the Gaussian function.

Finally, the resultant vector field $\vec{V}_{MT}$ indicates the new instant heading angle for the robot.

Test results at different speeds with a robot football robot. The maximum speed is 100% corresponding to 3.0 m/sec. The coordinate system is in inches.

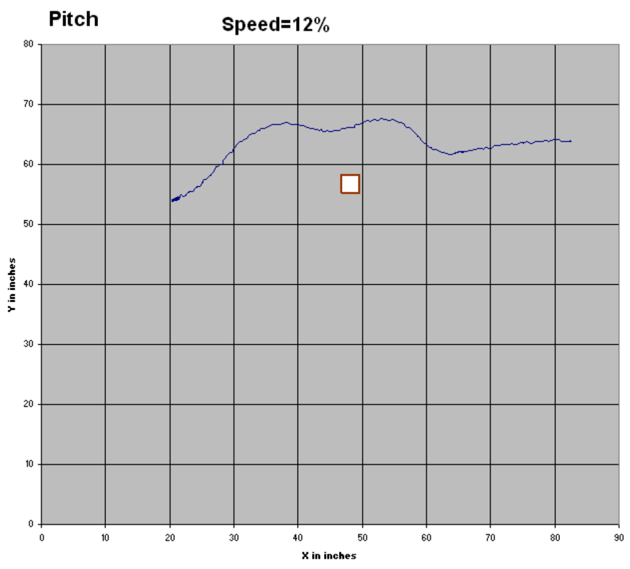


Fig 9: avoidance path at 0.36 m/sec

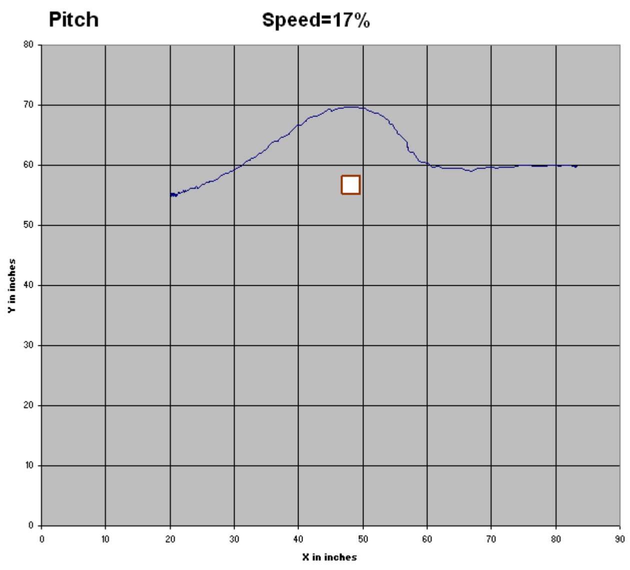


Fig 10: avoidance path at 0.51 m/sec

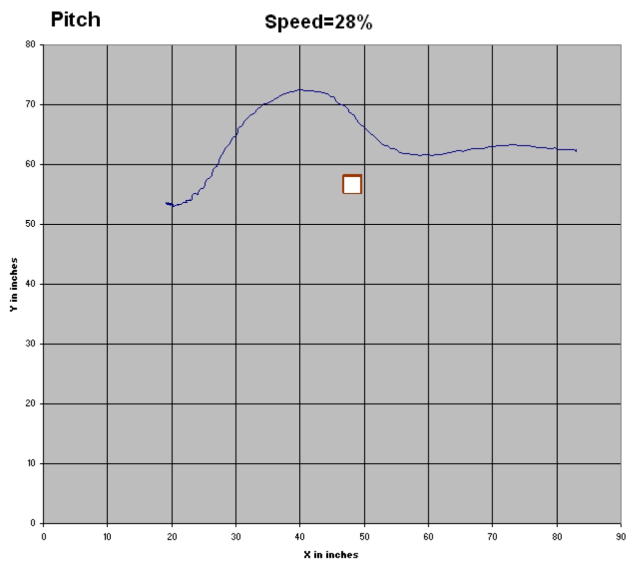


Fig 11: avoidance path at 0.84 m/sec

2.2.6 Matching the trajectories to the dynamic model of mobile robots

A current attempt of the author is to compare a path through a potential field with the robots dynamics model in order to determine if the robot can follow it. This can be done in frequency domain, by comparing the bandwidth of the robot plus controller model to the bandwidth of the input signal when trying to follow the path. This approach can be taken further. This could provide a basis of matching a vector-field by design to the robot's bandwidth.

2.3 Modelling mobile robots

This chapter is concerned with developing and understanding models of mobile robot kinematics and the control of each individual motor actuating the links within the kinematic model. Further reading is available in McKerrow P J (1991) chapter 8.1 which references to Muir P F and Neuman C P (1986). Muir and Neuman introduced a way of modelling wheeled mobile robots. It is related to modelling the kinematics of robot arms (manipulator kinematics).

Differential driven Robot

Differential driving is one of the simplest methods of modelling a mobile robot. This is probably why it is so common. The robot consists of 2 diagonally opposing wheels, see Fig. 12. If both wheels have the same velocity, the robot will go straight. If one wheel goes faster than the other the robot will follow a circular trajectory. If one wheel turns in the opposite direction of the other but with the same magnitude in speed, the robot will turn around its centre, “on the spot”.

The wheel Jacobian matrix is given and can be used as follows:

$$\dot{p} = \begin{bmatrix} v \\ \omega \end{bmatrix} = \begin{bmatrix} \frac{V_R + V_L}{2} \\ \frac{V_R - V_L}{L} \end{bmatrix} \tag{13}$$

$$p = J * \dot{p} = \begin{bmatrix} \cos\theta & 0 \\ \sin\theta & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix} = \begin{bmatrix} v\cos\theta \\ v\sin\theta \\ \omega \end{bmatrix} \tag{14}$$

Where v is the velocity forward of the centre of the robot and ω is the angular velocity around the centre of the robot, see Fig 12. $\dot{p}$ is related to p through the wheel Jacobian. p is the posture of the robot. The posture gives information about how the robot moves with respect to the floor.

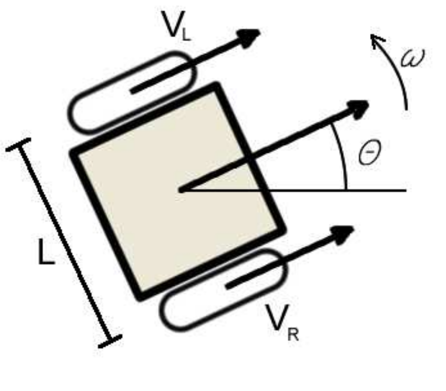


Fig 12: Differential driven Robot

θ indicates the instant heading angle of the robot.

Assuming no slip, the direction the vehicle is facing towards, is the same as the direction of the velocity vector (at and instant in time). An advantage of this fact, it simplifies calculations. A disadvantage however is that it can not move side wards.

3 DESIGN AND IMPLEMENTATION

3.1 Specification

for fast autonomous mobile platform:

- faster than 1m/sec
- large enough for real world application, such as picking up goods
- space for a onboard laptop
- enough sensors for autonomous movements
- battery life for several hours
- inexpensive (< £1000)

3.2 Mechanical Design

Every part of the mechanical design is build from basic materials, only the caster wheels are a ready made construction. One focus of the project was to build the mechanical construction rather than buy a ready made gearbox and frame. As a benefit the authors machining skills has improved.

3.2.1 Frame

The robot body consists of a steel frame that is welded together forming a box. Initially the frame was screwed together until the design was fully developed. Then the screws and brackets have been replaced by welded joints. The top rectangle can be taken of in order to do repair work. A large orange plastic sheet is mounted on top as a base for the circuit boards and the notebook. The battery is placed on top of the bottom frame. The key point is here that the bottom frame is lower than the wheel axis. It is placed just 2 cm above ground to prevent the robot from toppling at high speed.

3.2.2 Steering

The steering consists of 2 links, i.e. 2 wheels.

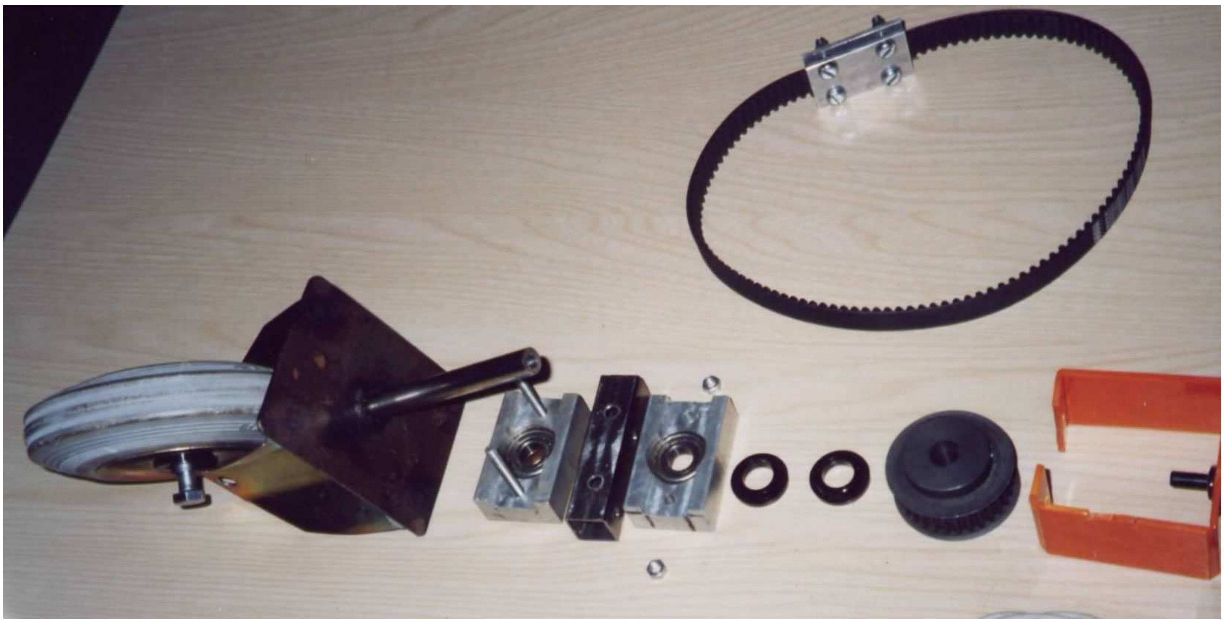


Fig 13: Explosion picture of one steering link

One steering link consists of a medium duty caster wheel that has been welded to a plate. The plate and the underlying caster-wheel have a 12 mm shaft welded on in order to enable steering of the wheel. The wheel is not offset its centre, unlike on a shopping trolley for example. Therefore it must be controlled by active steering to line it up with the direction of movement. Both steering shafts are driven by a motor-gearbox combination (gear-ratio 1:50) over a belt system (ratio 1:2). The motor is a 12 Volt DC Motor. A potentiometer on the top of one shaft is read by a microcontroller to determine the current steering angle. The overall system is a servo system, since it has positional feedback, see section 3.3.5 for a description of the control.

The above design, is the finally implemented one, the initial design had a stepper motor with controller circuit. However, the stepper motor was not powerful enough to turn the steering on rough surfaces. The implemented system responds quick and accurate within a fraction of a second to any angle.

There are 3 ball bearings per link: one in the axis of the wheel and two in line with the 12mm steering shaft. This two ball-bearings shift the weight of the robot onto the wheel. One steering link is designed to carry a weight of 120Kg. One could argue that axial-ball bearings would have been better, but the axial load of the radial ball-bearings chosen is much higher than the maximum weight that the robot will ever experience. The two ball-bearings are placed in a machined aluminium housing. All the machining for the slot and the place to fit the bearing was done with a lathe and a milling machine.

3.2.3 Gearbox

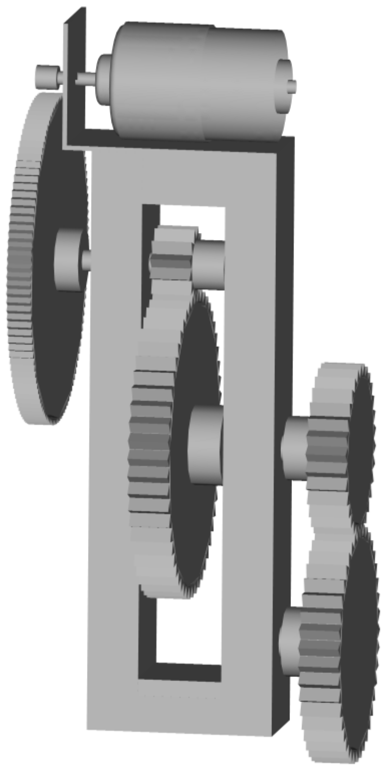


Fig 14: Gearbox in AutoCAD

The two gearboxes are constructed out of 4 solid aluminium bars each, which are bolted together. On the bottom bar two slots are milled out, increasing the accuracy of their alignment with the other bars. During construction the bars where clamped together, in order to align the shaft holes of both bars precisely. The surfaces of the bars have been milled straight at the beginning, to have accurate reference during construction.

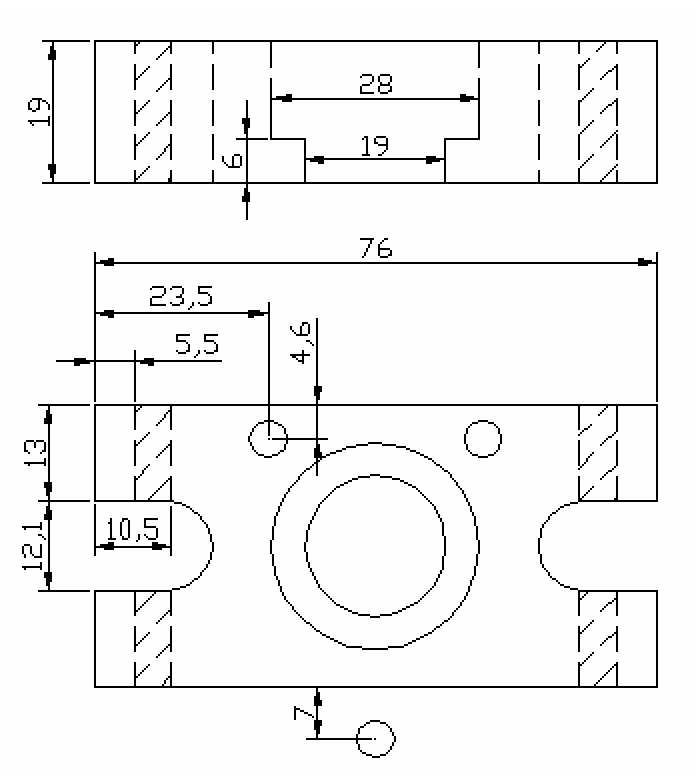
The gearbox has 2 ball bearings on the shaft that is connected to the wheel. The other two shafts are for transmission gears. Each shaft has sleeves to adapt to the different diameters of the gears.

The gear ratio is:

Transmission Name	**Shaft diameters**	**Gear diameters**	**Gear ratio**
Motor-Gear to Top-Shaft	5mm to 12mm	15mm and 120mm	1:8
Top-Shaft to middle-Shaft	8mm to 14mm	24mm and 110mm	1:4.58
Middle-Shaft to Wheel-Shaft	12mm to 12mm	60mm and 84mm	1:1.4
		Product\≈	**51.33**

n.b. Wheel diameter = 125mm

Wheel circumference ≈392.7mm



A further ball bearing with housing is mounted onto the frame. Thus the frame is connected to the housing and the housing to the gearbox.

The holes marked with stripes in figure 15 are for fixing frame an housing together.

Fig 15: Housing with 3 holes for gearbox-mount

3.2.4 Accuracy

For the construction of the gearbox, only machine tools such as a lathe and a milling machine can achieve the accuracy. A stand drill is already problematic. The machines should be calibrated with a dial indicator. A dial indicator is a dial gauge that can measure distance in fractions of millimetres. It is mounted onto the lathe or milling machine to align the tool with the work piece.

3.3 Electronic Hardware Design

Every circuit in the robot has been designed from basic principles. The design consists of two modular Microcontrollers, the power electronics and the ultrasonic sensors.

3.3.1 Power Supply circuit

The robot runs of a 12Volt battery. In the centre of the frame is place to strap on a car battery or motor-cycle battery. With a car battery, the robot runs approximately 3-4 hours in constant action. The power is split up into signal power and motor power from the battery onwards to minimize noise distribution. The motor power goes through an emergency stop button before being fed to the electronics board. All circuits can be switched of through a lever switch added next to the emergency stop. A bipolar capacitor with 4700uF is placed on the power electronics board. Each power regulator is surrounded by capacitors as well. The larger electrolytic capacitors are always accompanied by a bipolar 10nF or 100nF ceramic capacitor. The tracks on the power electronics board have a diameter of 6mm. The motor power cables have a diameter of 4.4mm. The cable is originally designed for speakers. The noise amplitude on the 12Volt rail is less than 100mV.

3.3.2 Microcontroller Module

The modular microcontrollers was designed to be an improvement from the popular robot football circuit, which is used by many students at the university. Unfortunately the chip used in the old circuit (90S8515) is discontinued and the new generation, the Atmel Mega series usually comes as surface mount device). At a development stage, surface mount is a problem. Firstly, it is not easy to unsolder a surface mount chip and secondly, a surface mount chip can not be stuck into a breadboard to do a quick design check.

The module was designed with the following specification in mind:

- similar amount of ports as the 90S8515
- only a bare minimum on components on board
- serial and programming connector (Robot football compatible)
- Avoid extra features such as test LEDs, I2C connector etc. since they are application dependant
- Power LED for quick confirmation
- Crystal with build-in capacitors
- Plug-in design with a Pin distance usable for bread-boards

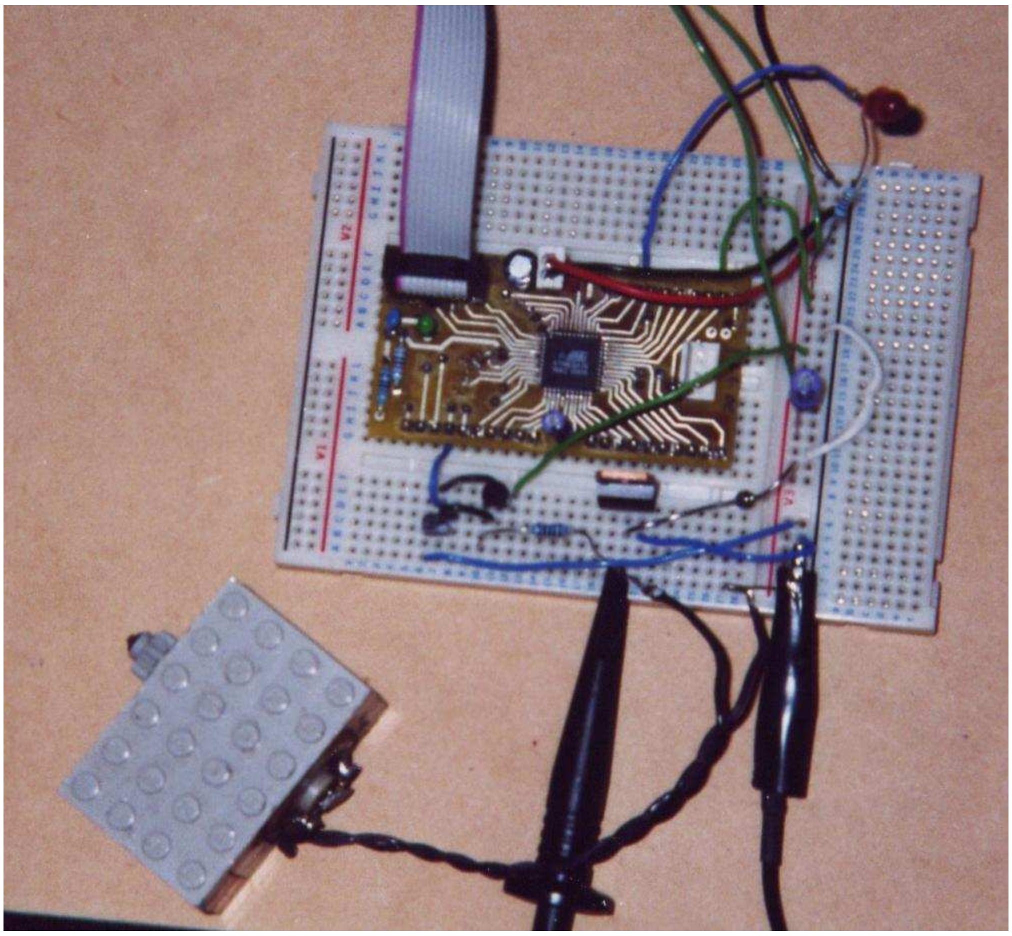


Fig 16: Atmel Mega16 Microcontroller board used for designing the motor controllers

The specification is appreciated by the technicians and other students of the University.
Several other students already applied this design to their final year project, which proves the flexibility of the design. The author is currently writing a guide on how to develop with an AtmelMega and the new gcc 3.X compiler. A draft version of the guide can be found in the Appendix.

Technical Details of the Microcontroller Module

- Atmel Mega16-AI in TQFP package (Atmel Package Code 44A)
- 16 MHz Crystal
- Atmel ISP Programming connector (IDC10, right angled)
- Robot Football 4-Pin Molex Serial Port connector
- 3x 10Pin Single-in-Line connectors for IO-Ports

3.3.3 Ultra Sonic Sensors design

The final design of the sensor is more simple than the original. The flexibility has increased since modulation and signal decoding is part of the software. Faster sensing is made possible through the changes. However, it demands more computing time.

Features of the new design include:

- frequency can be set by software
- signal can be coded
- reliable range 1.3 m

The transmitter consists of a software running in a timer at 76 to 84 kHz and toggling the transistor Q1. The toggling divides the frequency by two. Unfortunately none of the timer frequency settings match the resonance frequency of the transducers. Therefore, the timer frequency must be programmed to sweep from a few kilohertz under the resonance frequency to a few over the resonance frequency.

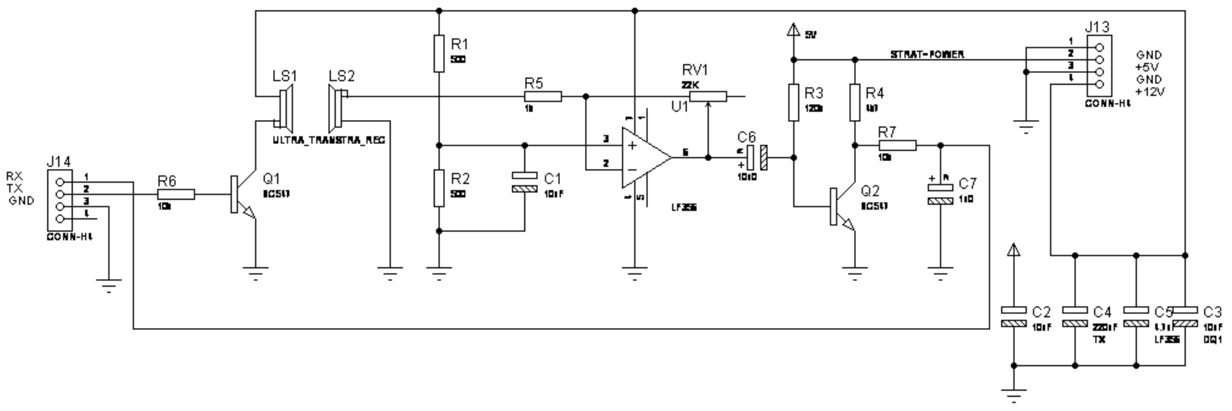


Fig 17: Ultra sonic distance measurement electronics

The receiver end consists of a operational amplifier for signal boosting, a transistor Q2 for level shifting (12V to 5V) and a low pass filter R7,C7.

The OpAmp is configured with a only positive rail at 12V. The positive input is clamped to 6V. Feedback resistor RV1 is a 47KOhm potentiometer in the final version, thus creating a variable gain from 1 to 48.

Practically gain values over about 30 amplify noise created by the transmitter over the power rail. Even the extensive use of capacitors could not remove this problem. The sensor can detect flat objects, such as walls and boxes up to 3 meters away. Reliable detection of humans can only be achieved within 1.3 meters.

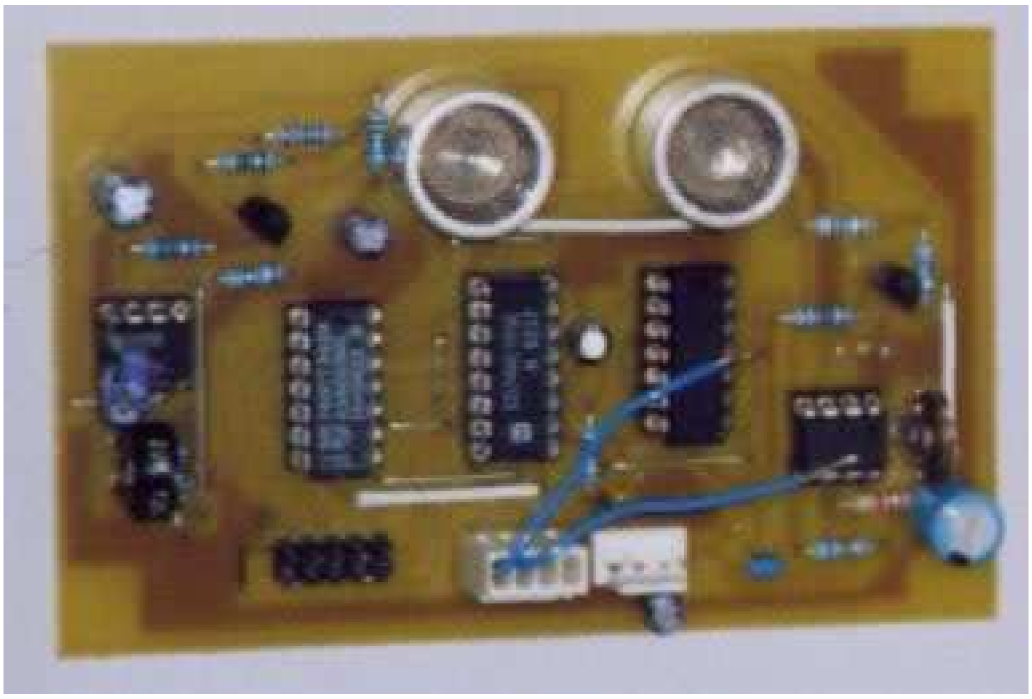


Fig 18: design of a ultra sonic distance sensor with 8-bit bus connector (original design)

Low pass Filter

The micro controller recognizes a logical high at 3.5V and above, Atmel (2003), on an digital IO pin. The filter must be matched to give this voltage at the maximum acceptable frequency. Experiments show that, the a design with the 3dB point at 42KHz (Transducer frequency) has not enough safety margin and the micro controller does not always recognise the signal as high when it should be. Therefore the 3dB point is set to 49KHz.

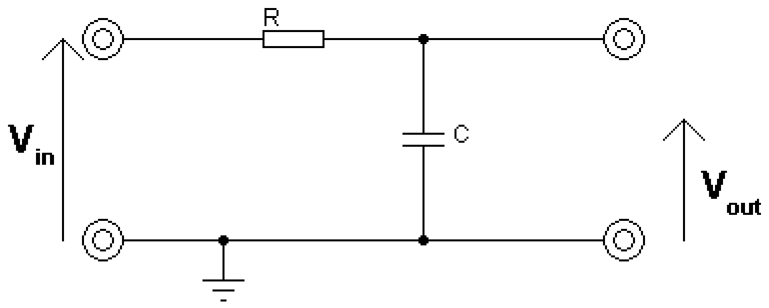


Fig 19: low pass filter (used in ultra sonic circuit)

The question is which R and C values to choose in order to have 3.5 Volt at the output at 49 kHz.

Initial formulae

$$\frac{V_{OUT}}{V_{IN}} = \frac{X_C}{R + X_C} \qquad (15)$$

rearranged for R.

$$R = \frac{\sqrt{V_{IN}^2 - V_{OUT}^2}}{2\pi f C V_{OUT}} \qquad (16)$$

Vin comes from the level shift transistor and is 5Volt.

Vout is 3.5Volt

C can be arbitrary, decided C=150pF

f= 49KHz (42KHz + margin)

using (16), then **R = 22KΩ**.

n.b. the output impedance of the transistor circuit has been neglected, since it is lower than the low-pass circuit. The input impedance of the microcontroller is much higher than the one of the low-pass circuit, and the impedance can be neglected in the calculation again.

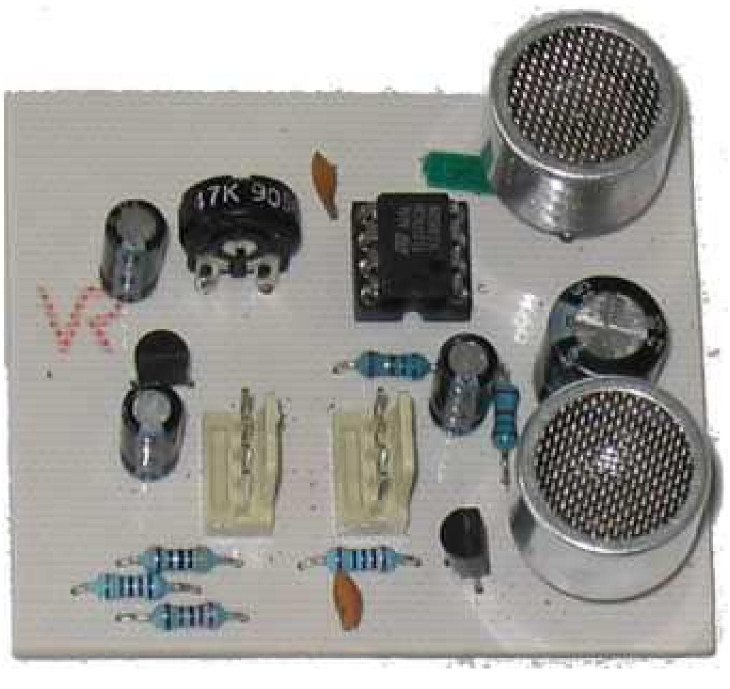


Fig 20: Ultrasonic sensor electronics (final design)

3.3.4 Shaft Encoders

The author designed two shaft encoder systems for the completion of the project. The second one is the one finally implemented into the robot.

3.3.4.1 Initial design

This design is based on a Microsoft Intelli Mouse. The encoder wheels and the photo transistor and infra-red LED was taken out and put into a separate circuit. Microsoft Intelli mouse encoders are quadrature encoders. The electronic components and the circuit diagram had to be identified by reverse engineering:

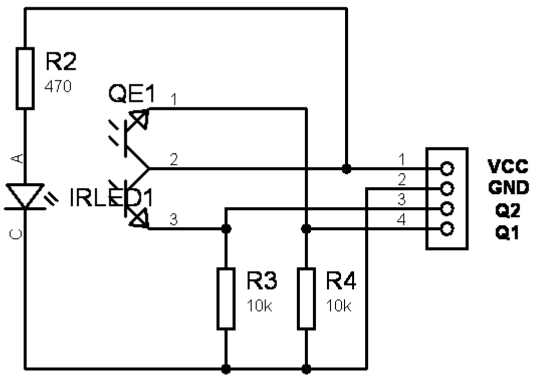


Fig 21: Shaft encoder circuit

QE1 is a Kingbright NPN Phototransistor L-610MP4BT/BD

IRLED1 is a Kingbright IR-LED (Gallium Arsenide) AM4457F3C

The circuit is supplied with 5 Volt. The outputs Q1 and Q2 will give the typical offset quadrature square waves. The alignment of the encoder wheel with the beam is very sensitive and must be done with precision tools. And here lies the disadvantage of this design. If the alignment tools are not available or the axis where the wheel is mounted on is not running perfectly smooth, the signal from Q1 and Q2 is distorted. The shaft encoder should be mounted directly onto the motor shaft, in order to increase the resolution. An Intelli mouse encoder wheel has only got 36 teeth.

Hence a second design was necessary, since the author encountered the described problems.

3.3.4.2 Final shaft encoder design

The final design uses an ISTS708 reflective opto switch manufactured by IsoCom. It consists of a infra-red LED and phototransistor sensor housed in plastic package responding to radiation from the diode when a reflective object is placed in the field of view.

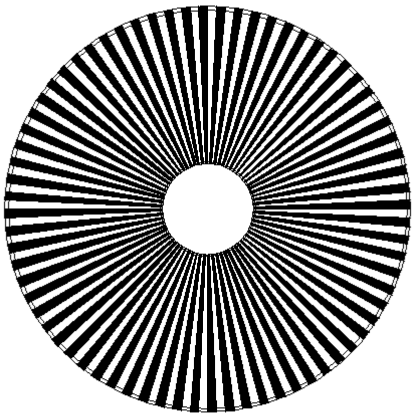


Fig 22: Encoder wheel

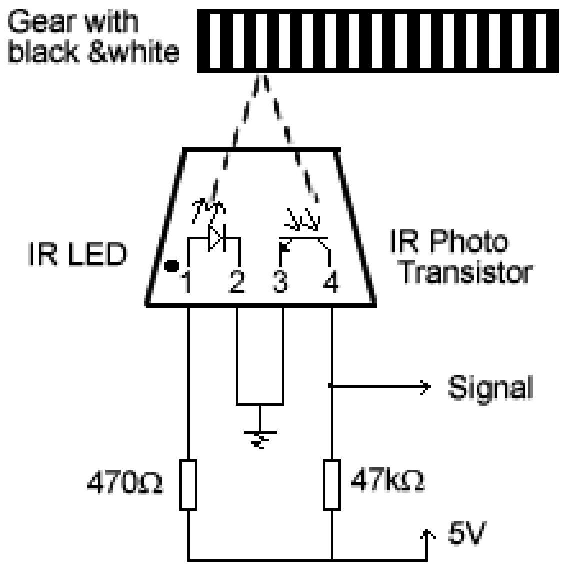


Fig 23: ISTS708 opto switch with circuit

The sensor is mounted in front of one the gears from the gearbox. The gear has radial stripes printed onto its side. White stripes reflect the light of the infra-red LED, black stripes don't. A 470 Ohm resistor is in series with the LED for current limiting, and a 47KOhm resistor is in series with the photo transistor to serve as potential divider, Figure 23. The value 47KOhm has been chosen by experiment to match the base current which is in fact daylight coming into the phototransistor.

3.3.5 Servo Controller

Line 1-2: The positional feedback is retrieved from a potentiometer mounted on a steering axis. The value is converted into a positive value between 0 and 255. Where 128 is the centre, making the robot go straight.

Line 4 is the Control Law, in this case a simple proportional gain called STEERING_GAIN = 26.

The steering gain is K_p = 2.6 but it appears as 26 in the equation and the whole equation is divided by 10. This increases the resolution and avoids floating point operations, which are time-consuming on a 8-bit CPU.

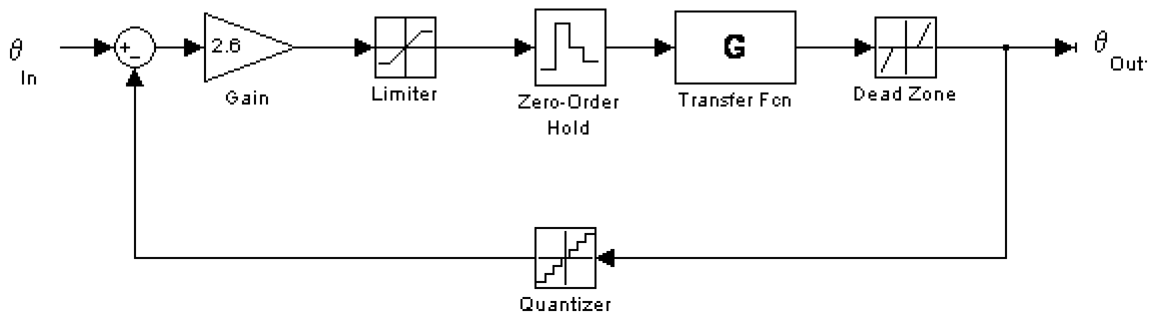


Fig 24: Steering controller model

Control law

$$\theta_{out} = K_p(\theta_{in} - \theta_{FB}) \tag{17}$$

The sign of θ_{out} determines which direction the motor must turn. Therefore the relays are switched accordingly, see code line 9 to 14.

Non-linear controller components:

Line 16 and 17 introduce a saturation limit, to protect the system from too much power.

Line 19 takes the absolute value of θ_{out} since the sign of θ_{out} has already been processed by switching the relays.

Line 22 adds a hysterisis. This is used for noise rejection. An error smaller than 4 will be ignored (rejected). Noise can cause the relays switching very quickly.

Line 23 to 25 is a dead band compensator. If error is too small to cause the motor to move the drive signal is increased.

In line 44 θ_{out} finally becomes the PWM ratio but before a software limit switch has a final decision if the servo is driven into the limits (< +-90 degree)

Line 29-45:

The software limit switch effectively does not allow the servo to drive more than +-90 degrees. At 90 degrees the wheels are positions such that the robot can turn around its centre on the spot. The reference radius is therefore 0 and a smaller radius can not be achieved. Thus it makes no sense to turn the steering more than 90 degrees.

If the steering wheel went more then 90 degrees due to overshooting, line 34 and 41 drive it out from the limit.

Steering Software

```
1 SteeringFeedback =  (int)A2D();
2 if(SteeringFeedback < 0){SteeringFeedback+=256;}              // make it look unsigned
3
4 Theta_out = ((Theta_in - SteeringFeedback)* STEERING_GAIN)/10;
5
6 // Steering Controller equation
7 Theta_out_old=Theta_out;
8
9 if(Theta_out < 0){                                            // is negative
10              PORTB&=~0x02;                                   //    relay SDIR (logic 0 , relay
released)
11 }else if(Theta_out > 0){                                             // is positive
                 PORTB|=0x02;                                   //    relay SDIR (logic 1 , relay
pulled)
12 }else{                                                               // is around 0
13      Theta_out=0;                                            //    delete error
14 }
15
16 if(Theta_out > 126) {Theta_out = 127;}                               // Max Saturation
17 if(Theta_out < -126){Theta_out = -127;}
18
19 if(Theta_out < 0) {Theta_out = -Theta_out;}                  // Take absolute value
20
21
22 if(Theta_out < 4) {Theta_out = 0;}
23 if((Theta_out >= 4)&&(Theta_out < DEADBAND_OFFSET)){                 // considerable minimum
                                                                        error
24                                                                      //but within motor dead
                                                                        band
25                      Theta_out = DEADBAND_OFFSET;
26 }
27
28
29 if(SteeringFeedback < MIN_ANGLE){
30 if(Theta_out_old < 0){                       // wrong direction to get out from the limit
31              Theta_out=0;
32              STEERING_PWM=0;
33      }else{  // right direction to get out
34              STEERING_PWM=DEADBAND_OFFSET+10;
35      }
36 }else if(SteeringFeedback > MAX_ANGLE){
37       if(Theta_out_old > 0){                 // do not allow to drive the servo more into the limit
38              Theta_out=0;
39              STEERING_PWM=0;
40      }else{
41              STEERING_PWM=DEADBAND_OFFSET+10;
42      }
43 }else{
44       STEERING_PWM=Theta_out;
45 }
```

3.3.6 Power Electronics

The power electronics drive the motors of the robot and communicate with the notebook through serial. The board also forwards all sensor information to the onboard computer. A photo of the power electonics can be found in the appendix.

board features:

- 3 channel PWM , 4 Quadrant drive
- max 12 Volt and 30 Ampere per channel
- current sensing for 2 of the PWM channels
- A to D converter for servo (8-Bit)
- 2 quadrature Shaft encoder Inputs (interrupt controlled)

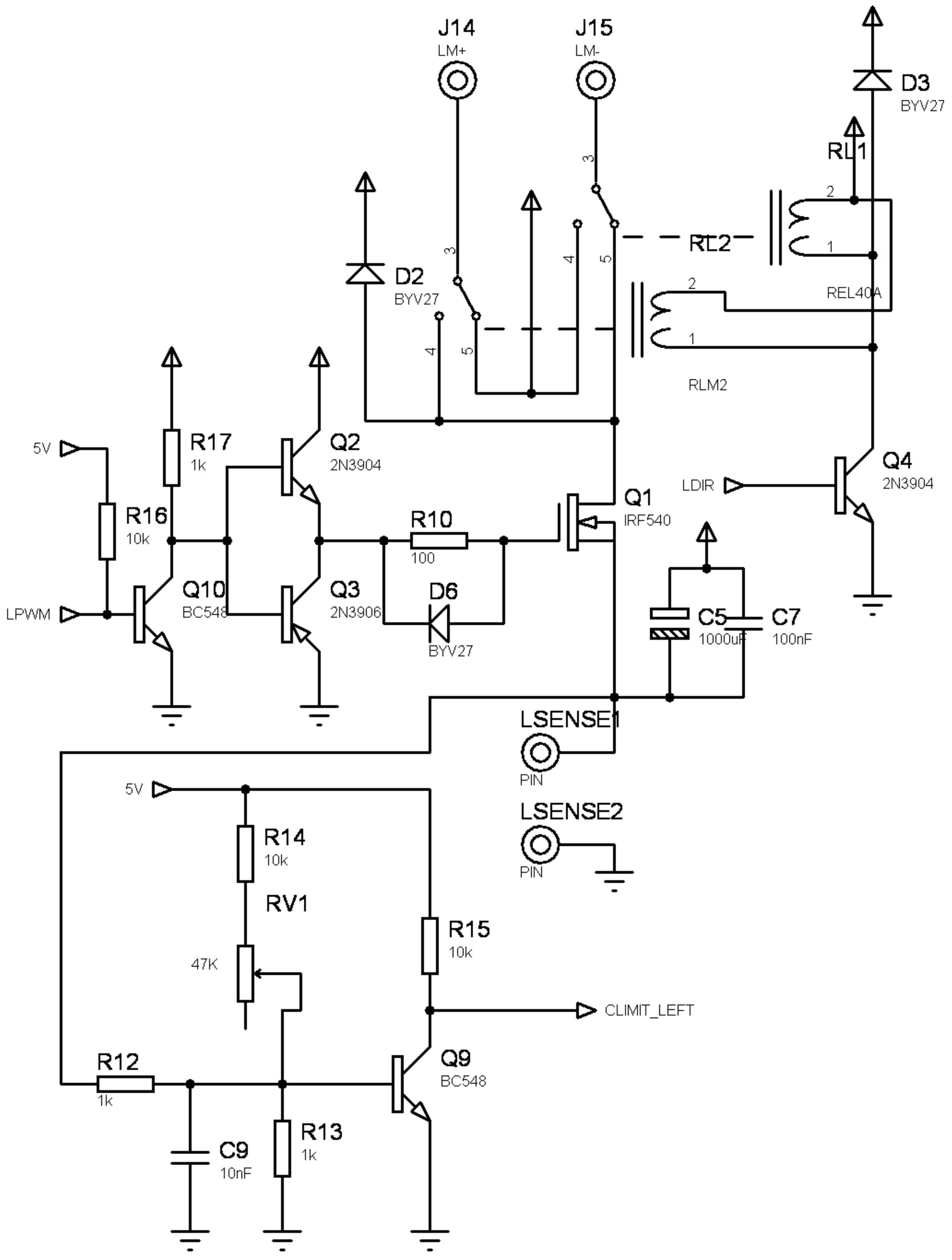


Fig 25: Left channel of the drive system electronics.

3.3.6.1 Circuit inputs

5 and 12 Volt	Power
LPWM	5 volt square wave from the microcontroller
LDIR	Direction for the motor
RV1	User can set the hardware current limit with the Potentiometer

Circuit outputs

LM+,LM-	Connector for motor
LSENSE1/2	Connector for Current Sensing Resistor
CLIMIT_LEFT	Current limit flag for microcontroller

3.3.6.2 Circuit functionality

The PWM signal is level shifted from 5 to 12 Volt through transistor Q10. R16 is a pull-up resistor which comes into action if the pin is virtually disconnected from the microcontroller. There are three possibilities when this can occur:

- The microcontroller module has been taken out, but the power is on
- The microcontroller is hold on reset during programming
- The microcontroller pin is configured as an input

The so called “glitch-free” PWM of a Atmel microcontroller can not be interrupted during one PWM cycle. Changing the pin with a “outp()” command will be ignored from the microcontroller. However, for current limiting it is essential to switch off during one cycle, as soon as the current hits the limit. The only way to achieve this, is to configure the pin as an input when the current limit is reached, which will cause the power to cut out, since the pull-up resistor R16 will switch Q10 through.

Fast MOS-FET switching:

The Q2,Q3,R10 and D6 are responsible for charging and discharging the gate of Q1 as fast as possible. During the switching of Q1 its internal resistance changes from a fraction of a ohm (when switched through) to infinite ohms (when open). Therefore power is dissipated during switching. The slower the switching the more power will be dissipated. Experiments have shown that, without this circuit, at switching frequencies higher than 500 Hz all power will be dissipated for switching. Switching on will cause Q2 to conduct through R10. When switching off the gate discharges through D6 and Q3.

D2 is the free-wheel diode for the motor. Smoothing capacitors C5 and C7 are placed as close as possible to the MOS-FET. An additional unipolar capacitor with 4700 uF was placed at the power connector of the PCB.

Direction control

The direction of the motor is changed by applying a signal to Q4, which switches the two automotive relays RL1 and RL2. D3 is a free-wheel diode for the relay coils.

3.3.6.3 Current Limiter design

Based upon measurements, the armature resistance is varying with temperature between 0.15Ω and 0.18Ω.According to the datasheet, the stall current is 85A.

$$I = \frac{V}{R_a} \frac{12V}{0.15\Omega} = 80A \qquad (18)$$

Equation 18 confirms the datasheet.

Although a Power MOSFET is in series with the motor, in a complete circuit, the maximum continuous current that can occur is still too high for an inexpensive MOSFET. Current limiting is required.

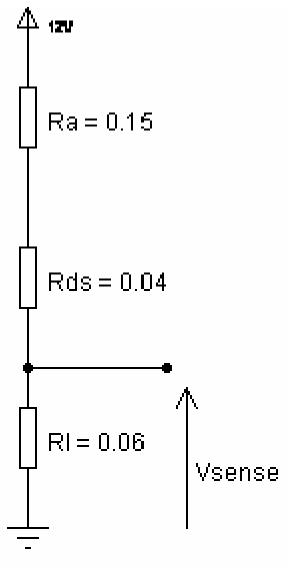


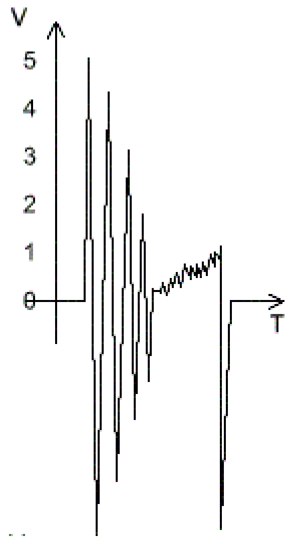
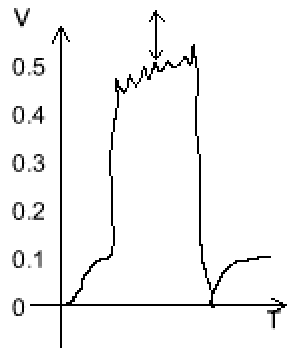
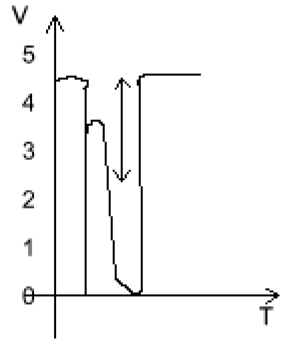
Fig 26: current path

The circuit on the left hand side, figure 26, describes a model of the motor, MOSFET and current sensing resistor in series.

$$I = \frac{V}{R_a + R_{DS} + R_L} = 48A \qquad (19)$$

The continuous drain current through a BUZ11 MOSFET, which was chosen for the design, is 30A. Within a PWM cycle the current rises up. And with it the voltage Vsense. Vsense going into a potential divider with low-pass filter, see figure 25, R12,R13, C9. RV1 sets the bias voltage for the transistor and R14 is for safety, so the user can not apply the full 5Volt to the base of the transistor by turning RV1 to 0Ω.

Here are the CRO graphs of the signal path in the current sensing circuit.
One PWM cycle is shown.

 Figure 27	Figure 27: This graph represents Vsense, measured directly at the current sensing resistor. Initial switching noise can be seen, as well as the negative inductor voltage at the end of the cycle. The inductor voltage is cut out after the diode responds.
 Figure 28	Figure 28: Signal at the base of transistor Q9. The low-pass filter removed the switching noise. The signal can be shifted up and down (arrow) with RV1.
 Figure 29	Figure 29: Showing the output of Q9 going into the micro controller. The transistor inverts the signal. The more the voltage comes near to 0, the higher the current. The graph indicates that the current must be measured more towards the middle of the PWM cycle, in order to get correct reading.

The microcontroller is checking the current synchronized with the timer interrupt that also generates the PWM signal. If a 0 is detected (too high current) the software disables the PWM output pin.

Current limit source code
SIG_OVERFLOW1 is the timer that generates the PWM signal as well.

```
SIGNAL(SIG_OVERFLOW1){
        int i;
        for(i=0;i<22;i++){                  i++;i--;  }             // initial break
        if(PINC & 0x80 ){                                           // PC7 is high
                CurrentLimit&=~CLIMITLEFT;                          // Limit not reached , switch off
the flag
                DDRD|=0x10;
        }else{                                                      // PC7 is low
                                                                    // Current limit reached
                DDRD&=~0x10;                                        // switch the MOSFET off
                CurrentLimit|=CLIMITLEFT;                           // Flag on
        }
        if(PINC & 0x40 ){                                           // PC6 is high
                CurrentLimit&=~CLIMITRIGHT;                         // Limit not reached , switch off
the flag
                DDRD|=0x20;
        }else{                                                      // PC6 is low, Current limit
reached
                DDRD&=~0x20;                                        // switch the MOSFET off
                CurrentLimit|=CLIMITRIGHT;                          // Flag on
        }
}
```

3.4 Onboard communication

3.4.1 Serial Communication

Speed 38400 baud, 8 data-bits , no parity, one stop-bit

The serial protocol used is frame based. Square brackets indicate the start and end of a frame/packet. This is necessary to synchronize the information exchange. It enables the communicating machines to decide when the information is complete and ready to be processed. If synchronization is lost the communications software will wait for the next data start byte "[" to regain synchronization. Any data coming in before that will be ignored. The correctness of information transfer within the robot is not critical, since a packet gets out of date within milliseconds. Therefore there is no checksum or protocol extension for acknowledges etc. This protocol is a streaming protocol. If the serial bus would be disconnected and reconnected, everything would continue and re-synchronize automatically.

Command	**Explanation**
[LSx]	Left Motor Speed
[RSx]	Right Motor Speed
[LDC]	Left Motor Direction Clockwise
[LDA]	Left Motor Direction Counter-Clockwise
[RDC]	
[RDA]	
[Sx]	Steering Angle, where x is a signed 8 bit value And 0 degree represents: wheels straight ahead
[LG]	
[RG]	
[U]	Request sensor information

x is a 8-bit binary value from 0 to 255

The Sensor data Packet

When the Notebook (Master) requests sensor information from the Drive system micro controller, the drive-system microcontroller forwards the request via I2C bus to the sensor systems and collects the data. After that the drive system microcontroller will answer the request with the following packet:

[n G4 G3 G2 G1 H0 L0 H1 L1 H2 L2 H3 L3 H4 L4 H5 L5 H6 L6 H7 L7]

where:

n	is the number of sensor values being sent / in the system
G1 to G4	is the Global time-stamp in order to determine the sampling rate and how old the sample is. Where G4 is the most significant byte (MSB) of the 32-bit unsigned value.
Hx and Lx	are a 16-bit unsigned value of the sensor, which in this application represent the time-of-flight measured by the ultra-sonic sensors in unit time-stamps. Hx is the MSB.

3.4.2 I2C Communication

The I2C bus, pronounced "i squared c", was invented by Philips. It is a fast serial bus with two wires (Clock and Data) plus a ground reference. The AtmelMega 16 has special I2C registers and a interrupt flag. I2C has a protocol stack to manage device addresses and read/write operations. One device acts as a Master, the others are Slaves. Usually the master requests data from the slave.

In the current design of the robot, one microcontroller is connected to the onboard PC by RS232 link. In order for the PC to communicate with the other onboard devices such as the ultra sonic sensor microcontroller and the digital compass, the I2C bus is used.

The communication protocol used in the robot is described below.

The a single byte (Data) is requested from the slave.

Master	START	SLA+W		Address		RS	SLA+R			NA	STOP
Slave			A		A			A	Data		

START is a single bit indicating the start of a frame. SLA+W means, the master is going to transmit a byte onto the bus. A means acknowledge. Address; the Master transmits a slave address. This is used if more than just one slave is on the same bus. RS is a "repeated start", meaning that the master will continue transmitting SLA+R contains a register number that the master requests. Data is the actual data that the master requested. NA means not acknowledge, which is actually the correct reply of the Master on the end of a frame. STOP ends the transmission like a stop bit on the RS232 bus.

The following source code can be installed into the slave. It will generate the correct reply to a master. The slave has an array of registers that can be requested. Each array cell can contain data, such as a ultra sonic distance measurement.

```
volatile unsigned char Registers[REG_MAX];

SIGNAL(SIG_2WIRE_SERIAL)
{
   static unsigned char reg = REG_MAX;
```

```
  switch (TWSR) {
        /* SLAVE RECEIVER */
      case TW_SR_SLA_ACK:
      case TW_SR_ARB_LOST_SLA_ACK:
      case TW_SR_STOP:
      case TW_ST_DATA_ACK:
      case TW_ST_DATA_NACK:
        TWCR = _BV(TWINT) | _BV(TWEN) | _BV(TWEA) | _BV(TWIE);
        break;
      case TW_SR_DATA_ACK:
      case TW_SR_DATA_NACK:
        if (reg < REG_MAX) {
            Registers[reg] = TWDR;
            reg = REG_MAX;
        } else {
            reg = TWDR;
        }
        TWCR = _BV(TWINT) | _BV(TWEN) | _BV(TWEA) | _BV(TWIE);
        break;
      case TW_ST_SLA_ACK:
      case TW_ST_ARB_LOST_SLA_ACK:
        if (reg < REG_MAX) {
            TWDR = Registers[reg];
            reg = REG_MAX;
        }
        TWCR = _BV(TWINT) | _BV(TWEN) | _BV(TWIE);
        break;
  }
}


// Initialize I2C Bus in slave mode

void i2c_init(void){
      DDRC |= 0x03;           // I2C pins for Output
      PORTC &= ~0x03;                 // Output 0
      TWSR = 0x00;
      TWBR = 28;              // I2C Bus Speed,28,18 ?
      TWAR = 0xC0;            // Our I2C Slave Address
      TWCR = _BV(TWEN) | _BV(TWEA) | _BV(TWIE);
}
```

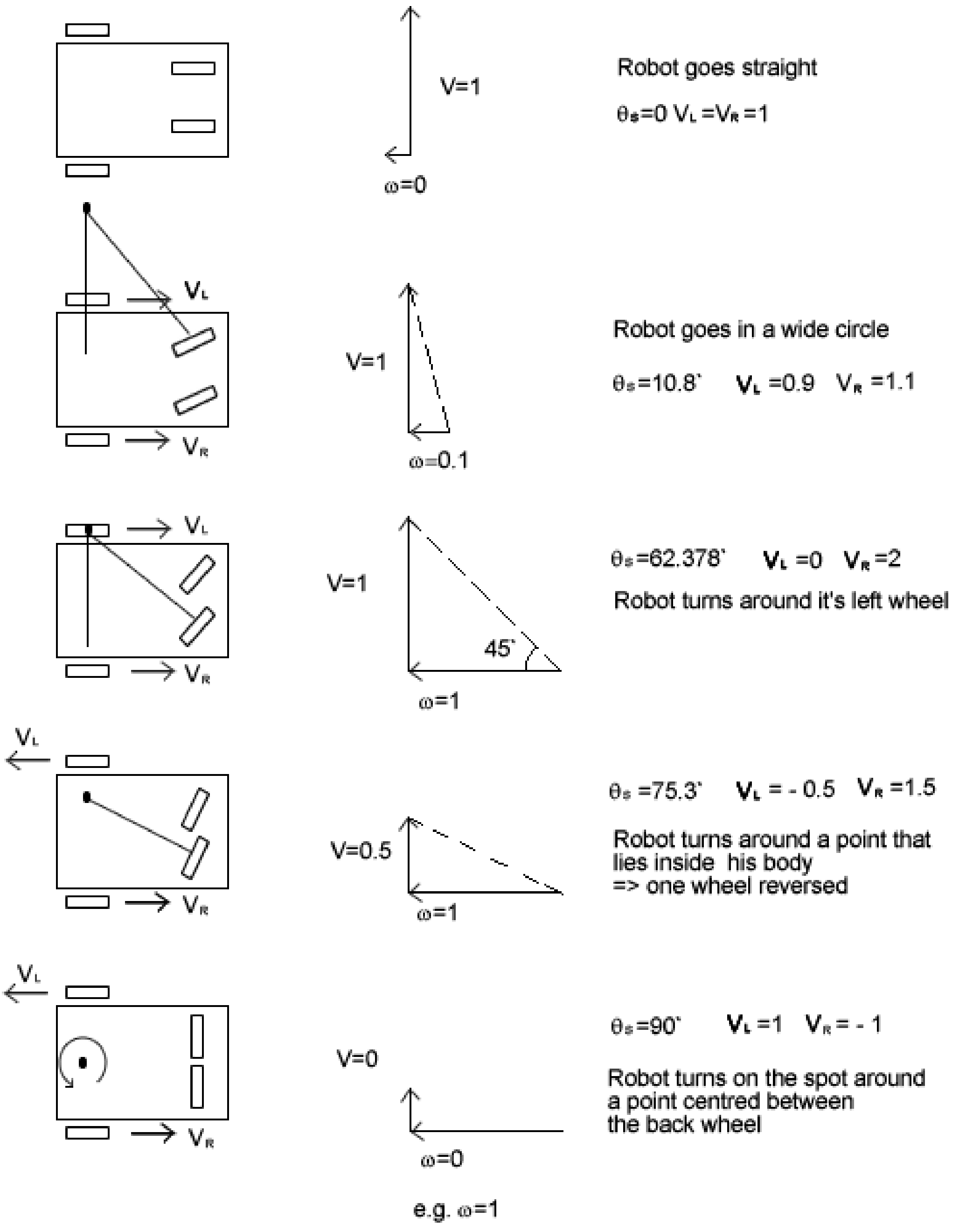
The i2c_init() function must be called once after reset of the micro controller, in order to initialize the micro as a I2C and set the baudrade to 100kHz.

4 INTELLIGENT CONTROL

4.1 Kinematic model

4.1.1 Inverse solution for vector based steering

The navigation system outputs a velocity vector to the inverse kinematics which calculate the steering angle and wheels speeds. The velocity vector consists of an angle which is relative to the robots current heading angle. The length of the vector is the speed. One can imagine a joystick control for the robot like a remote controlled car has, is the same principle. For example: Joystick more forward is faster; more left is gives smaller turning radius.


V=1
ω=0
Robot goes straight
θs=0 VL=VR=1
VL
VR
V=1
ω=0.1
Robot goes in a wide circle
θs=10.8° VL=0.9 VR=1.1
VL
VR
V=1
45°
ω=1
θs=62.378° VL=0 VR=2
Robot turns around it's left wheel
VL
VR
V=0.5
ω=1
θs=75.3° VL=-0.5 VR=1.5
Robot turns around a point that lies inside his body => one wheel reversed
VL
VR
V=0
ω=0
e.g. ω=1
θs=90° VL=1 VR=-1
Robot turns on the spot around a point centred between the back wheel

4.1.2 Inverse Kinematic algorithm

$$V_R = V_{in} + V_\omega \quad (20)$$

$$V_L = V_{in} - V_\omega \quad (21)$$

$$V = (V_R + V_L)/2 \quad (22)$$

$$\omega = \frac{V_R - V_L}{L} \quad (23)$$

if ω=0

then r = 20000 meter (seems infinite to an office robot)

else $r = \frac{V}{\omega}$ (24)

$$\vec{F} = \vec{M} - \vec{D} = \begin{bmatrix} m \\ 0 \end{bmatrix} - \begin{bmatrix} 0 \\ r \end{bmatrix} \quad (25)$$

if $V_R < V_L$

then $\theta_S = a\tan 2(F_y, F_x) - 90°$ (26)

else $\theta_S = a\tan 2(F_y, F_x) + 90°$ (27)

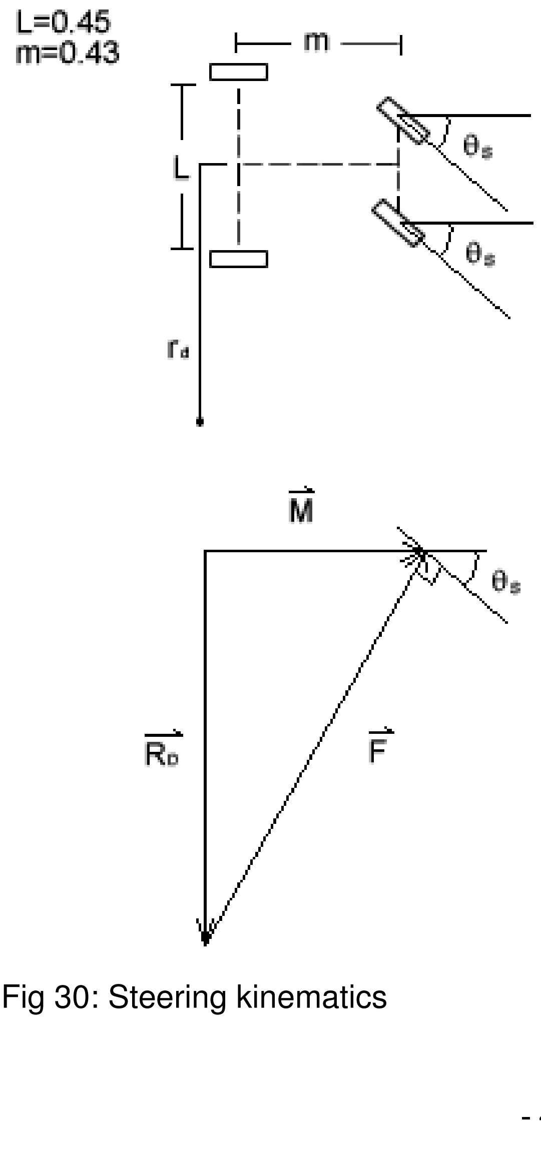


Fig 30: Steering kinematics

Given an velocity vector with the x and y components, corresponding to V_ω and V_{in} respectively, the output parameters: Steering angle θ_s and the driving wheel velocities Vr.Vl can be derived.

m is defined as the distance between front and rear wheels

L is defined as the distance between the two rear wheels

r is the radius about a virtual point which the robot is rotating about.

n.b. : r < L/2 then the point where the robot is rotating about lies within the robot

and if r=0 the robot rotates around its origin (on the spot)

4.2 Sensors for guidance

4.2.1 Position

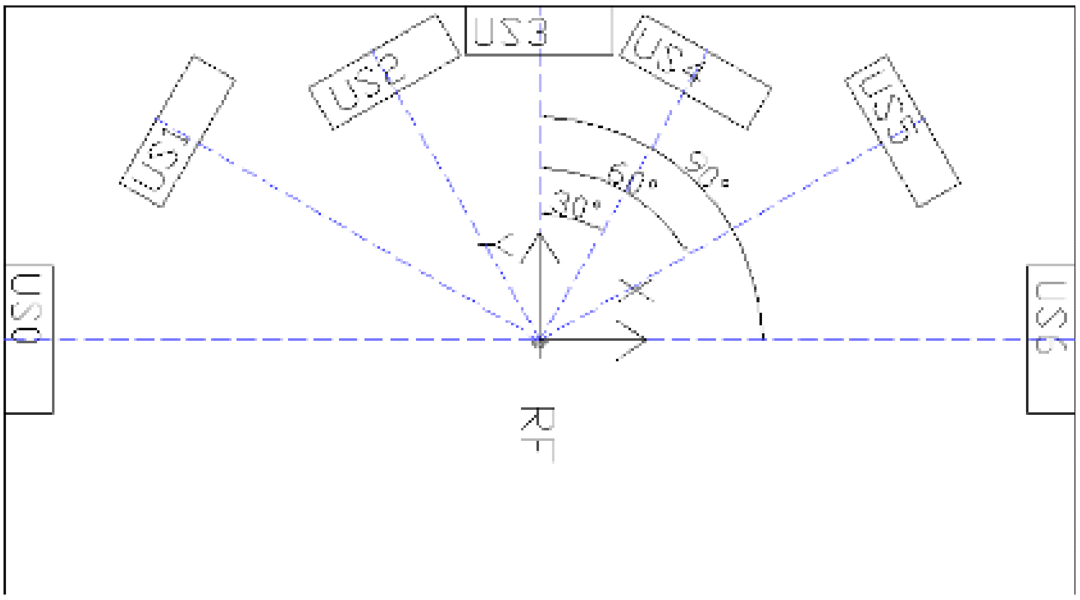


Fig 31: Position of the ultra sound transducers

The reference frame RF is the origin of the robot coordinate system. Is important to arrange the sensors in line with this reference frame. Otherwise the measured distance can not be converted directly into a vector pointing from robot to obstacle.

4.3 Investigation of sensing speed and reaction time

Assuming a worst case scenario can determine if the robot can stop or avoid an obstacle in its way.

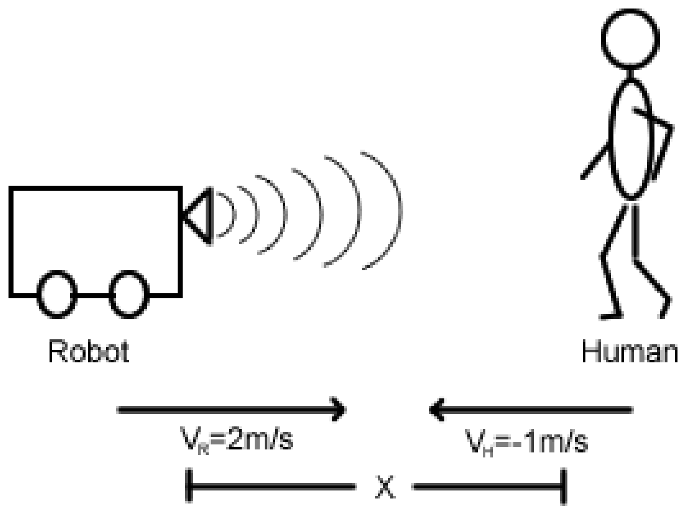


Fig 32: worst case collision scenario

Assumptions:

Robot and human coming towards each other:

$V_R = 2m/\sec$

$V_H = -1m/\sec$

$V_{Sound} = 300m/\sec$

2 measurements are required to determine the humans intentions.

The robot has a breaking distance of $d_{breaking} = 0.5m$

Further assumptions are:

Robot has a control delay for of 100ms (planning, communications…), which is equivalent to a distance of $d_{PLAN} = 0.2m$

The difference of the two velocities is the velocity between human and robot:

$$V = V_R - V_H = 3m/\sec \qquad (28)$$

This simplifies the model to a moving robot with Vr = 3m/sec and a standing human with 0m/sec.

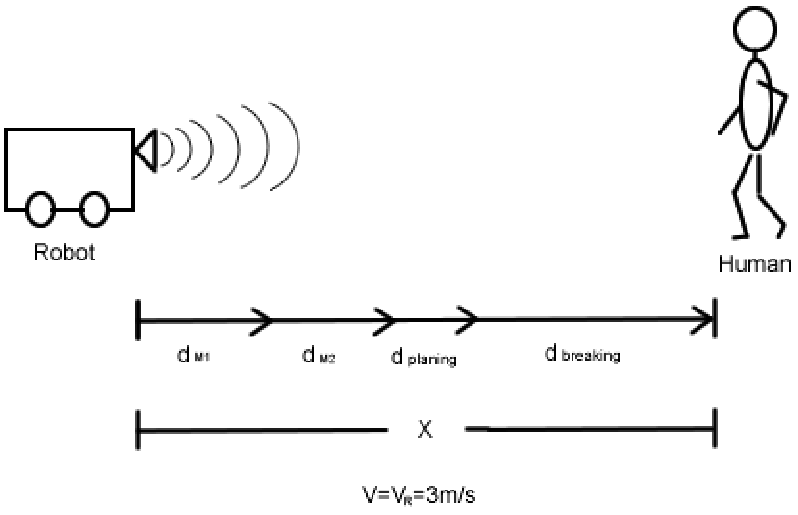


Fig 33: model of collision course scenario

Time until impact:

$$\Delta t_{\mathrm{Im}pact} = \frac{\sum d}{V_R} = \frac{x}{V_R} \qquad (29)$$

Distance until impact x:

$$x = d_{m1} + d_{m2} + d_{PLAN} + d_{breaking} \qquad (30)$$

Distance Measurement 1 (d_{m1})

The ultra sound wave travels x and then reflects on a human and travels back to x - d_{m1}, because the robot moved d_{m1} within the time of measurement.

$$x + x - d_{m1} = V_{sonic}\Delta t_{m1} \qquad (31)$$

Distance the robot travelled during the measurement

$$d_{m1} = V_R \Delta t_{m1} \qquad (32)$$

substituted into (31)

$$2x - d_{m1} = V_{sonic}\frac{d_{m1}}{V_R} \qquad (33)$$

rearranged

$$d_{m1} = \frac{2}{\frac{V_{sonic}}{V_R} + 1} x \tag{34}$$

Distance Measurement 2 (d_{m2})

Distance travelled by the ultrasound is x-d_{m1} towards the obstacle and x-($d_{m2} + d_{m1}$) back to the robot, which travelled d_{m2} during the time of measurement.

$$\Delta t_{m2} = \frac{2x - 2_{m1} - d_{m2}}{V_{sonic}} \text{ and } d_{m2} = V_R \Delta t_{m2} \qquad (35) , (36)$$

combined and rearranged

$$d_{m2} = 2x \frac{1 - \frac{2}{\frac{V_{sonic}}{V_R} + 1}}{\frac{V_{sonic}}{V_R} + 1} \tag{37}$$

(34) and (37) into (30)

$$x = 2x \frac{1}{\frac{V_{sonic}}{V_R} + 1} + 2x \frac{1 - \frac{2}{\frac{V_{sonic}}{V_R} + 1}}{\frac{V_{sonic}}{V_R} + 1} + d_{PLAN} + d_{breaking} \tag{38}$$

evaluated with Rv and Vsonic:

$$x = \frac{d_{PLAN} + d_{breaking}}{1 - 0.02 - 0.02} = \frac{d_{PLAN} + d_{breaking}}{0.96} \tag{39}$$

The equation shows that the measurement time of the ultrasonic sensor is so quick that it can usually be neglected.

Control delay and breaking distance are the limiting factors of robots maximum speed.

4.4 Navigation and Guidance system modelling

The Perception Model also called M^2P^2A ,see McKerrow P J (1991) is a way of transforming the physical world into a model that the robot can interpret. The Perception Model consist of the following transformation processes

1. MEASUREMENT
2. MODELLING
3. PERCEPTION
4. PLANNING
5. ACTION

This model, applied to has been applied to the example guidance System:

Sensor data such as Ultrasonic distance and direction is **measured** and **modelled** as vectors into the direction the sensor is facing. The robot **percepts** open space and obstacles by comparing these vectors. The default **plan** of the robot is to go straight ahead, obstacles cause a change of this plan into a alternative direction. The resultant direction is put into **action**, which means the robot is following it.

4.4.1 Demo Guidance System

In order to show the robot in action a simple guidance program was written. The mission of the robot is to go straight. The vector RobotVelocity determines the directions in which the robot will move.

The default mission of the robot is going straight: the vector is set to $V_{Robot} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$.

An obstacle on the side (90 degrees) of the robot causes it to correct its course slightly. (Vx=±0.15)

Obstacles located 30 and 60 degrees sideward call for more avoidance (x=±0.32,x=±0.55)

Obstacles ahead of the robot (Sensor US3 at 0 degrees)

```
void Avoidance(void){
        int i;
        double Timeout_distance = 1344.0 / 509.0909;   // 2.640 meter
        Vector2D Obstacle[7];
        Vector2D SensorRange[7];
        Vector2D ObstacleSum;
        Vector2D MissionVector;
        double isClose  =       0.50;

        // Go Straight
        RobotVelocity.x = 0.0;
        RobotVelocity.y = 1.0;

        if(vabs(Sensor[0])<(isClose-0.5)){              // Sensor 0 is located left at 90
                        RobotVelocity.x = 0.15;
                        RobotVelocity.y = 1.0;
        }else if(vabs(Sensor[1])<isClose){              // Sensor 1 is located left at 60
                        RobotVelocity.x = 0.32;
                        RobotVelocity.y = 1.0;
        }else if(vabs(Sensor[2])<isClose){              // Sensor 2 is located left at 30
                        RobotVelocity.x = 0.55;
                        RobotVelocity.y = 1.0;
        }

        else if(vabs(Sensor[4])<isClose){               // Sensor 4 is located right 30
                        RobotVelocity.x = -0.55;
                        RobotVelocity.y = 1.0;
        }else if(vabs(Sensor[5])<isClose){              // Sensor 5 is located right 60
                        RobotVelocity.x = -0.32;
                        RobotVelocity.y = 1.0;
        }else if(vabs(Sensor[6])<(isClose-0.5)){        // Sensor 6 is located right 90
                        RobotVelocity.x = -0.15;
                        RobotVelocity.y = 1.0;
        }


        // Sensor 3 is located at front

        if((vabs(Sensor[3])<0.9)||(vabs(Sensor[2])<0.6)||(vabs(Sensor[4])<0.6)){
                RobotVelocity.x = -0.0;
                        RobotVelocity.y = -1.0;
                        if((vabs(Sensor[0])<isClose)||(vabs(Sensor[1])<isClose)){
                                RobotVelocity.x = 0.40;                 //go back and right
                        }
                        if((vabs(Sensor[5])<isClose)||(vabs(Sensor[6])<isClose)){
                                RobotVelocity.x = -0.40;                        //go back and left
                        }
                        if((vabs(Sensor[1])<isClose)&&(vabs(Sensor[5])<isClose)){
                                RobotVelocity.x = 0.0;                          //go back straight
                                                                                // dead end
                                system("play sounds/txt1.wav &");
                        }
        }
}
```

If one of the sensors at the front (-30°,0° and +30°) measures a very close obstacle, the robot assumes that it can not continue to go forward. It will decide to put the wheels in reverse immediately, causing a sudden stop and a backward movement. During its move backwards the robot will look out for obstacles on its side. If an obstacle is on its side it should steer towards it, when going backwards.

Just like a car parking out of a car park space. The car would first drive in reverse gear towards the pavement (side obstacle) and then forwards away from it.

5. 3D MONITORING INTERFACE

For the study of behaviour of autonomous robots a human-machine interface is required that displays how the robot percept its environment. Debugging, simulation and navigation development becomes easier and therefore more efficient if the interface is used.

5.1 VRML

One of the simplest 3D programming languages is VRML.
Within the navigation system, information about the robots environment and sensors can be converted to 3D objects. This 3D objects are then send out to the monitoring PC via network. Objects are described as polygons, cones, boxes, cylinders or spheres.
The position and orientation can be described as 3D transforms. Transforms can have global or local reference frames. The way of describing transforms and kinematic relationships between VRML objects (nodes), is closely related to robotics.

5.2 Information exchange for remote monitoring

As shown in the system overview diagram (APPENDIX), the onboard notebook generates VRML objects from all data available to it. An onboard Apache web server contains a homepage consisting of a VRML plug-in and a Java-Applet. If a user downloads this homepage into his or her client PC over the network, the Java-Applet starts running and requests data from the robot. This data are the VRML objects. This enables the user to see a continuous update of the robots actions on screen. The VRML objects are sent through a separate TCP/IP socket, not over the web-server.

6 PROJECT MANAGEMENT

The interim reports give detailed information about the project management. The overall project size is more than the scheduled time of one double module, the author estimated that he could make up for the lack of time with experience in electronics, but he was wrong. Fortunately, the project is designed to be open ended. The initial scheduled time of 1 month for navigation system and 1 month for the report writing, is actually slack time. This slack time turned out to be essential.

7 CONCLUSION

The robot and the drive electronics have been constructed and tested successfully. The only part in specification that could not be met is the additional sensors. Future applications for the designed robot include tour guide or (blind) person's guide/porter, store house assistant. By coupling it with a trailer, it could transport bigger parcels/pallets around factories. Cleaning or guarding offices, is another possibility.

The project is about to provide a platform, in order to implement one of these applications in a future project. The approach of designing all components from the basic principles, rather than buying a gearbox or the drive electronics, etc, brought the author much valuable experience.

The author identified the key specification of an ideal fast mobile service robot

- a minimum speed >=1.5 m/sec
- enough sensors to detect even a small obstacle
  (maybe infrared + ultra sonics + camera + bumper switches)
- more than 3 meter sensor range
- A response time that allows it to stop if a person comes towards it
- enough size to implement a real world application onto it.
- not noisy
- not too expensive (< 1000 Pound)
- minimum control delays and responds fast
- data analysis tool for debugging of the navigation
- remote task assignment

By comparing this key specification to current development is science, including this project, it can be estimated that there will be a fast mobile **service** robot for sale within the next 10 years.

GLOSSARY

Definitions of expressions used in this report.
The definitions may be only valid for robotics.

MSB	Most significant Byte
Posture	a set of values defining position and orientation of a robot
TQFP	Thin Quad Flat Pack (a thin type of surface mount chip package)
Trajectory	a path plan for a vehicle
VRML	Virtual Reality Modelling Language (3D object based programming language, based upon a interpreter engine interpreting wrl-text files, simple and suitable for internet applications)

REFERENCES and BIBLIOGRAPHY

Kuo B C (1992). *Digital Control Systems*, 2nd.Ed. , Oxford University Press, New York

Wolf J C (2003a). *An Investigation of Digital Control*, ECAL326 Coursework, University of Plymouth, U.K.

B-J Lee, S-O Lee and G-T Park (1999).
Trajectory Generation and Motion Tracking Control for the Robot Soccer Game, Proceedings of the 1999 IEEE/RSJ International Conference on Intelligent Robots and Systems

D-H Kim, J-H Park and Jong-Hwan Kim (2000).
Limit-cycle Navigation Method for Soccer Robot, Dept. of Electrical Engineering and Computer Science, KAIST, Korea

Robinson P,Hall P, Wolf J, Phillips R,Peck C,Culverhouse P,Bray B & Simpson AJ (2004) *The Technology and Challenges of MiroSot Robot Football,* School of Computing, Communications and Electronics, University of Plymouth, U.K.

McKerrow P J (1991) . Introduction to Robotics,Addision-Wesley Publishers Ltd, Sydney, Australia

Muir P F and Neuman C P (1986) *Kinematic Modelling of Wheeled Mobile Robots*, Robotics Institute, Carnegie Mellon University

Web3D (2004) The WEB 3D Consortium (2004), *www.web3d.org*

Wolf J C (2004) Wolf J C, Yang S-M,Hall P ,Schulte C, *www.swrtec.de*
Homepage about projects, including **this one**

Atmel (2003) The Atmel Cooperation, *Atmel Mega 16 Datasheet*, doc2466, San Jose California, USA

APPENDIX

APPENDIX 1: UoP ATMega Handbook (DRAFT VERSION)

Joerg Wolf
Draft Version

The University of Plymouth ATMega Handbook

1. Introduction

If you are already confident with basic ANSI C programming (statements such as if, else, do, while, for, main and others) and you have a basic understanding of microcontrollers, this document should be easy to understand. A final version of this document can be downloaded from www.swrtec.de.

1.1 What do I need ?

- ATMega16 / ATMega16L
- avr-gcc version 3 (WinAVR)
- PonyProg 2.05a or 2.06c
- University ATMega16-Developper-Board
- Atmel Parallel Port downloading cable
- University Serial Level converter board

1.2 How do I know I have got the right software ?

Open a command prompt and enter: **avr-gcc –version**

The output should look something like this:

```
C:\data\atmel\soo-mi>avr-gcc --version
avr-gcc (GCC) 3.3.1
Copyright (C) 2003 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
C:\data\atmel\soo-mi>
```

Make sure that the version number is >= 3.0.0. If the version is e.g. 2.95 then you got the wrong software.

Start up PonyProg and check the version by the menu “?” , then “about”. The version number should be 2.05a or higher.

2. Get going

2.1 Software Installation

2.1.1 Windows : WinAVR 3.X

Just double click the setup….

2.1.2 Linux: gcc 3.x

Guide to install avr-gcc 3.3 or 3.4 under Linux No warranty and no support that this will work for every system. But I successfully compiled avr-gcc under Linux with the following packages:

```
avr-libc-1.0.3.tar.bz2
binutils-2.14.tar.bz2
gcc-core-3.4-20040310.tar.bz2
uisp-20040311.tar.bz2
```

The packages should be compiled in the following order:
1. binutils for the system
2. binutils for AVR-target
3. gcc-core
4. libc
5. uisp

The first step is to install the new binutils for the Linux gcc:

```
tar xvfj binutils-2.14.tar.bz2
cd binutils-2.14
./configure
make
make install
```

The next step is to install binutils again, but this time for AVR.
Everything will be installed to the target folder /usr/local/atmel

```
make distclean
./configure --prefix=/usr/local/atmel --target=avr --enable-languages=c --host=avr --disable-nls
make
make install
cd ..
```

Now the AVR-GCC can be compiled. The new avr-gcc is included in the original package from gnu.org.
No patches are required.

```
tar xvfj gcc-core-3.4-20040310.tar.bz2
cd gcc-3.4-20040310
./configure --prefix=/usr/local/atmel --target=avr --enable-languages=c --host=avr --disable-nls
make
make install
```

Now the AVR files should be renamed in order to avoid confusion if the system gcc and the avr-gcc are both in the path.

```
cd /usr/local/atmel/bin
```

```
mv gcc avr-gcc
mv objcopy avr-objcopy
mv ar avr-ar
mv as avr-as
mv cpp avr-cpp
mv ranlib avr-ranlib
...
```

Before compiling avr-libc the following environment variables must be exported:

```
export CC=/usr/local/atmel/bin/avr-gcc
export AR=/usr/local/atmel/bin/avr-ar
export AS=/usr/local/atmel/bin/avr-as
export RANLIB=/usr/local/atmel/bin/avr-ranlib
export PREFIX=/usr/local/atmel

cd "download-folder"
tar xvfj avr-libc-1.0.3.tar.bz2
cd avr-libc-1.0.3
./doconf
./domake
./domake install
cd ..
```

The last step, is to install the ISP-Programmer software.
I opened a new console, so there are none of the libc export variables around.

```
tar xvfj uisp-20040311.tar.bz2
cd uisp-20040311
./configure --prefix=/usr/local/atmel/
make
make install
```

Congratulations, you have successfully compiled avr-gcc
I must admit it is a bit more work than WinAVR.

2.1.3 PonyProg configuration:

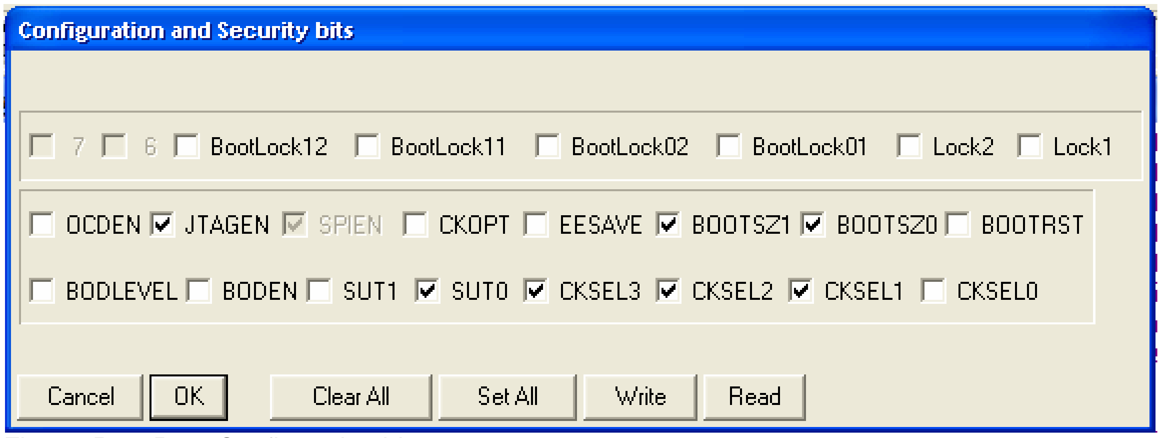
1. Open PonyProg and go to the menu point Setup / Interface Setup
2. Click on Parallel
3. Select “AVR ISP I/O
4. Click on LPT1
5. Click OK
6. Open menu point Setup / Calibration

It should say , calibration OK

2.2 Setting up the fuses

Start up PonyProg and go to the menu point Command / Security and Configuration bits and then press read.

When the micro comes from the factory the settings in PonyProg look like this :



☐ 7 ☐ 6 ☐ BootLock12 ☐ BootLock11 ☐ BootLock02 ☐ BootLock01 ☐ Lock2 ☐ Lock1

☐ OCDEN ☑ JTAGEN ☑ SPIEN ☐ CKOPT ☐ EESAVE ☑ BOOTSZ1 ☑ BOOTSZ0 ☐ BOOTRST

☐ BODLEVEL ☐ BODEN ☐ SUT1 ☑ SUT0 ☑ CKSEL3 ☑ CKSEL2 ☑ CKSEL1 ☐ CKSEL0

Fig 2.2 PonyProg Configuration bits

This setup means that the micro is running with its 1 MHz internal clock and it doesn't care even if you have soldered in a external resonator already. Be aware that the JTAGEN flag is set by factory default, which means that PORTC doesn't work for normal IO !
There are many possible settings depending on the required configuration. Setting up the fuse bits is a dangerous business, since you could loose control over the micro if they are set wrong.

2.2.1 1 MHz Internal Clock Setup

So the minimum to get going is to disable the JTAG interface by doing the following steps:

1. Press Read
2. remove the tick in front of the JTAGEN
3. press Write

2.2.2 16MHz External Clock Setup

1. Solder in a 16 MHz crystal with the capacitors
2. Press Read
3. remove all ticks ! (except SPIEN of course)
4. tick CKOPT
5. press Write

2.3 Setting up a Port

Example:
DDRA = 0x08;

2.4 Using a Port for Output

A port is a group of physical pins on the micro. Therefore setting a port like this **PORTA = 0xF0;** will actually make pins high and low. It is a good idea to represent the bits as a hexadecimal value. This is indicated through the 0x in front of the hexadecimal number F0. In this case the lower 4 significant bits are low (0) and the 4 higher ones are set to high (F).
The Atmel Mega16 has 4 ports called PORTA PORTB PORTC PORTD. Some bits are already used for special functions such as serial port, programming port others are general purpose.

Often it is necessary to change only one bit at a time. You might have two programs which control different pins on the same port.
Here is an example how to switch on bit 3 from PORTA.

```
PORTA = PORTA | 0x08;
```

This line of code takes the existing content of port A (the current status of highs and lows on the pins) and combines it with a logical OR and then stores the result in PORTA. The line can also be written as

```
PORTA |=   0x08;                // pin high
```

In Binary the bits look like:

```
PORTA before           0 1 0 0   0 0 0 1
0x08                   0 0 0 0   1 0 0 0
-------------------------------------------------------
PORTA new (OR)         0 1 0 0   1 0 0 1
```

Example:

```
PORTA &=~0x08;                  // pin low

( 0x08                 0 0 0 0   1 0 0 0 )


PORTA before           0 0 1 0   1 0 0 0
~0x08                  1 1 1 1   0 1 1 1
-----------------------------------------------------
PORTA AND ~0x08        0 0 1 0   0 0 0 0
```

2.5 Using a Port for Input

Example:

```
If( PINA & 0x01){
                    // The pin is high
}else{
                    // The pin is low
}
```

2.6 Complete Source

2.7 Setting up the Makefile

To edit the Makefile you can use:

- Windows Notepad
- Programmers Notepad
- MS Visual C++
- Windows WordPad

But you must **NOT** use:

- MS-DOS Edit

Editors like MS-DOS Edit do not use real tabs (ASCII 9), they substitute tabs with spaces (ASCII 32). This will confuse “make”.

2.8 Compiling the program

2.9 Downloading the program with PonyProg

3 Using Timers

PWM

4 Hardware Interrupts

Appendix

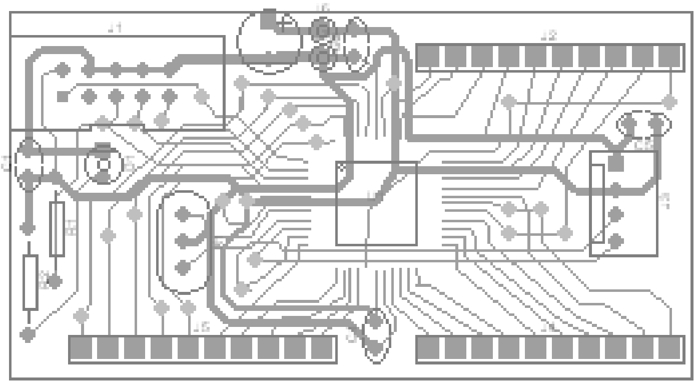


Fig A1 Layout

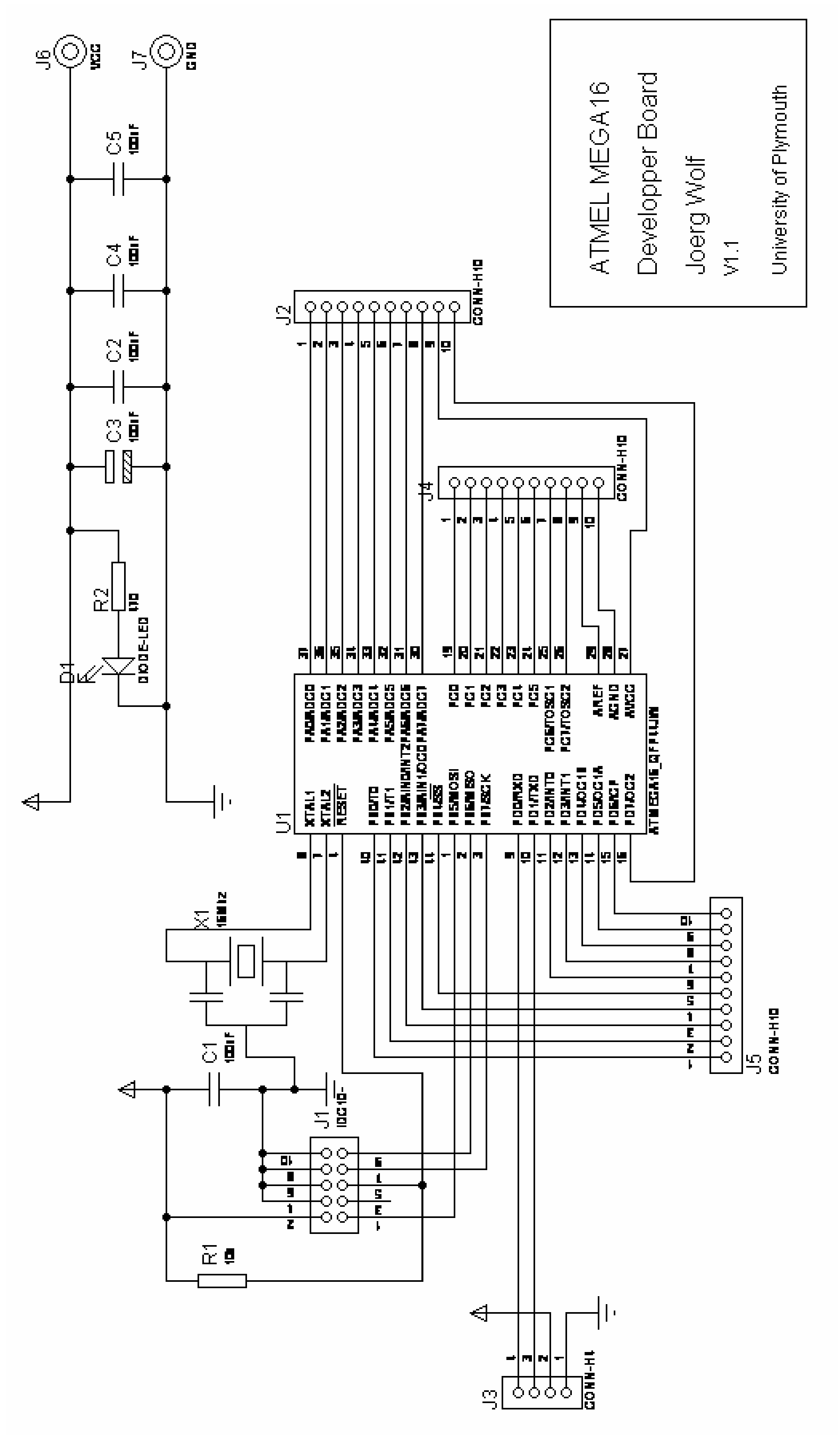
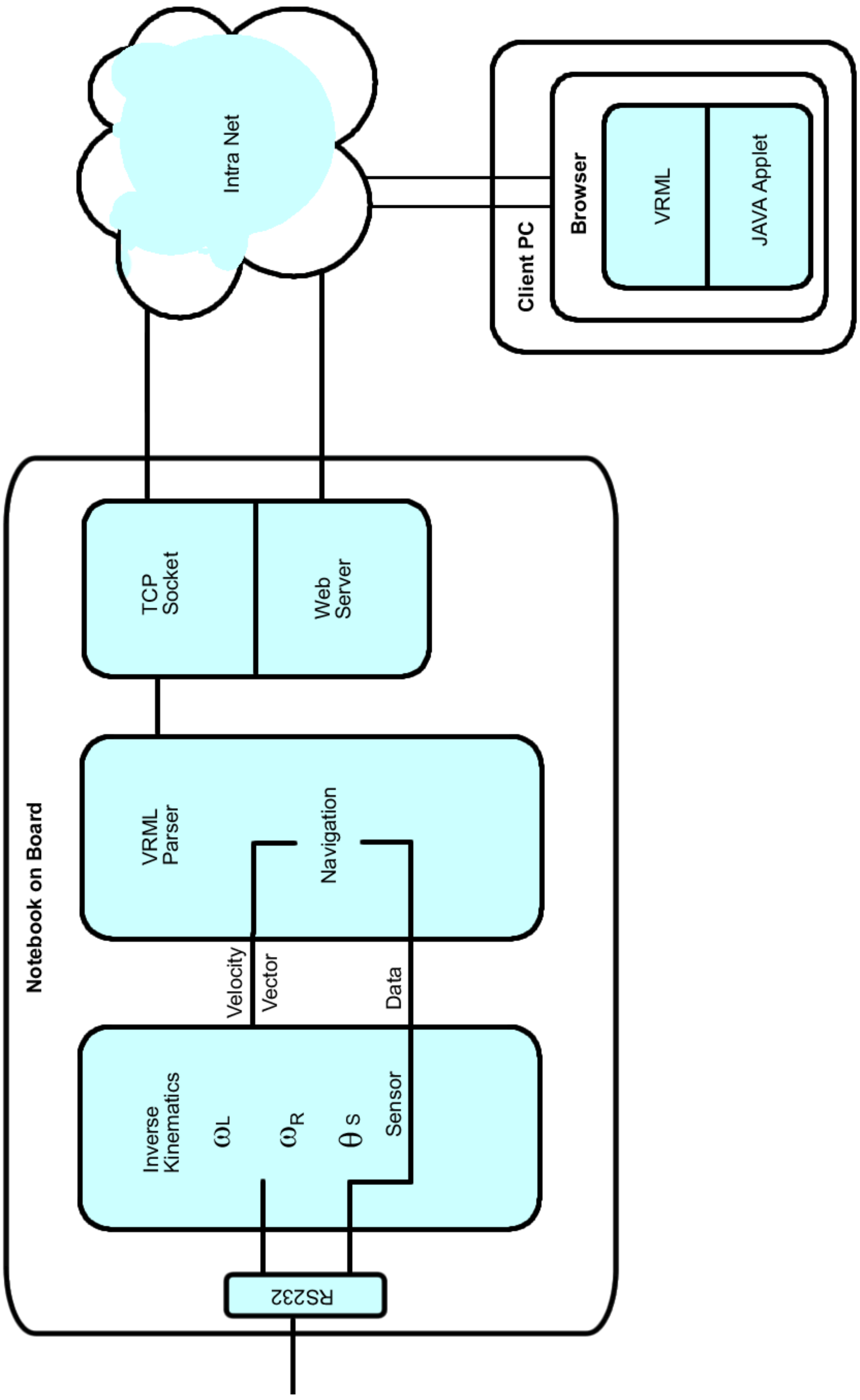
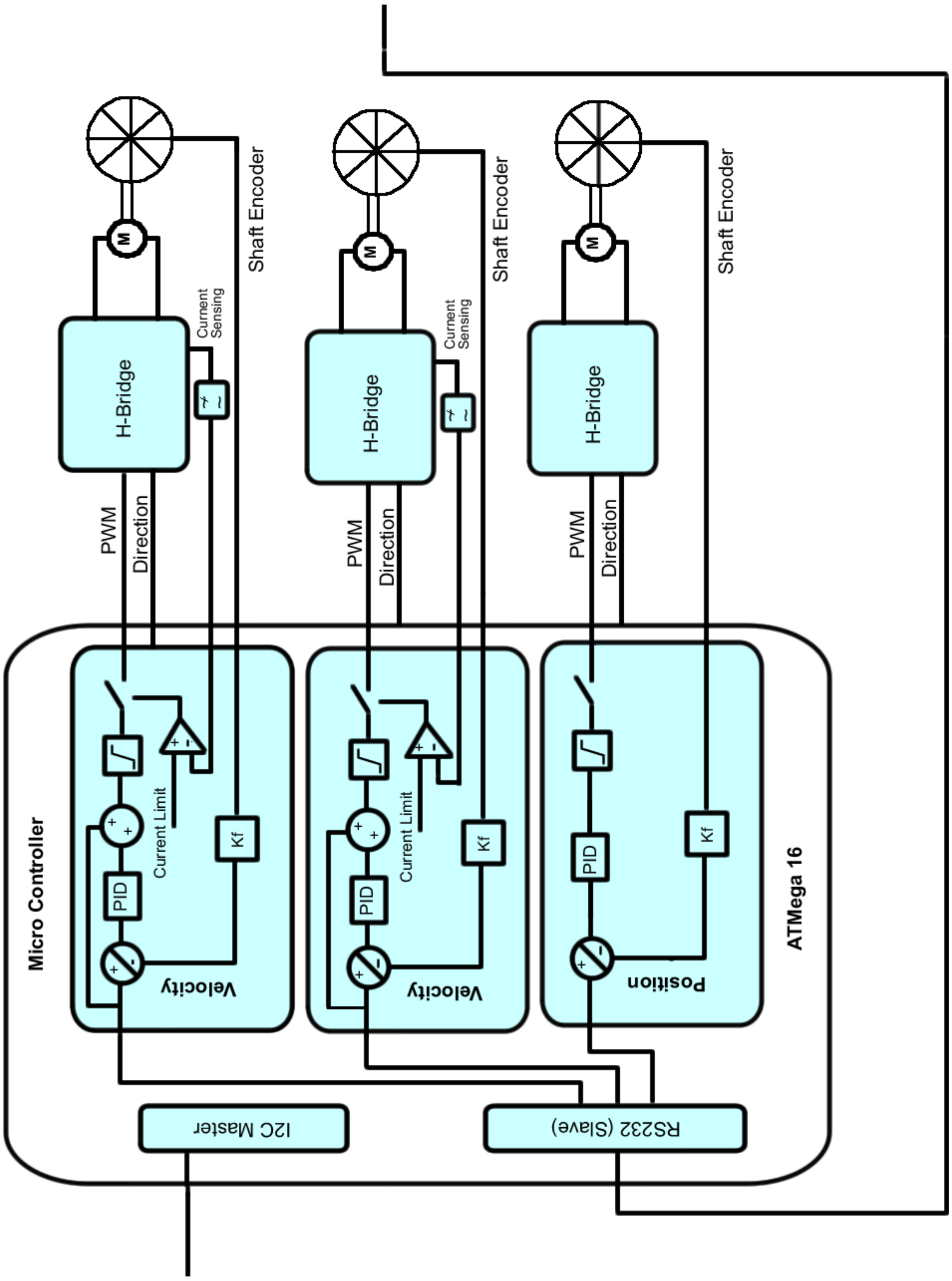


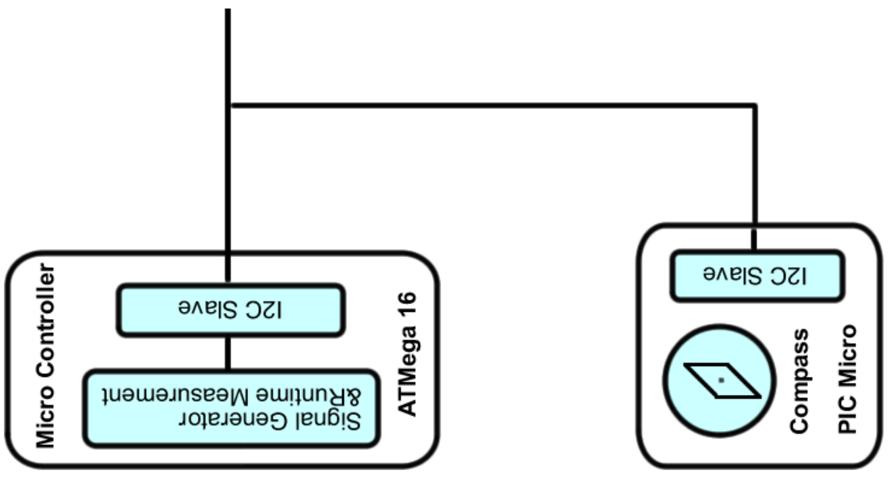
Fig A2 Circuit Diagram of the UoP module

APPENDIX 2: System overview block diagram

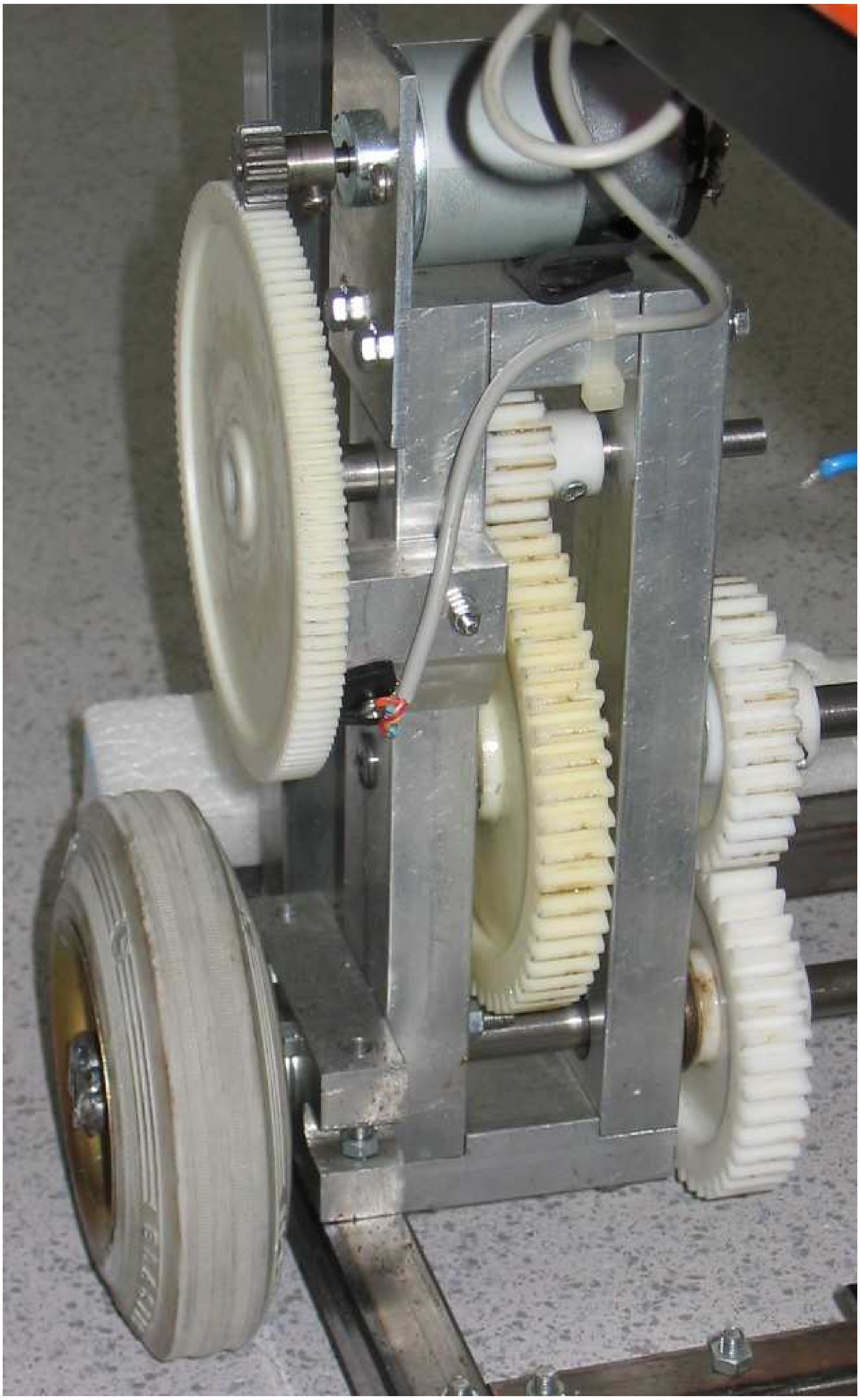




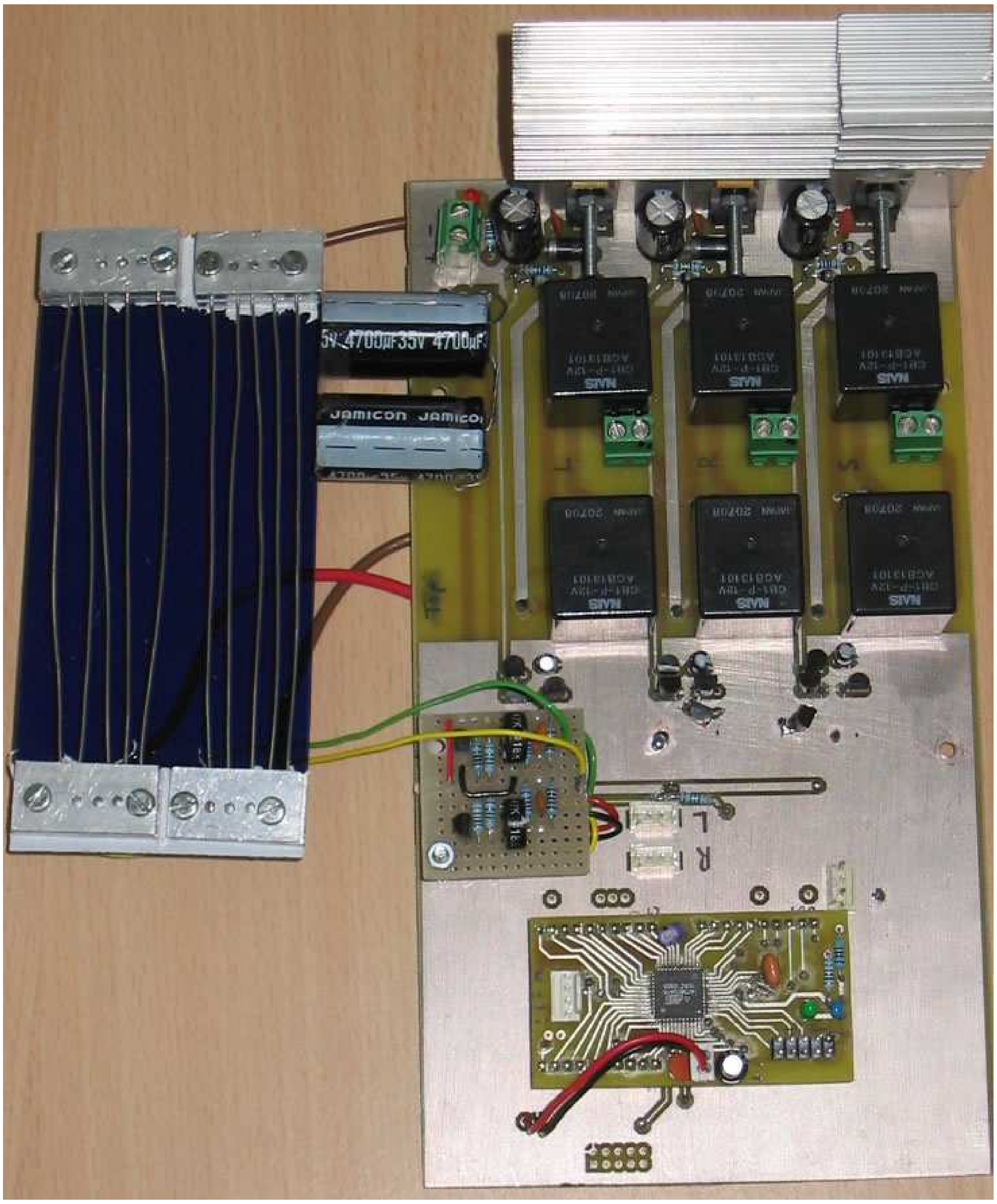
Micro Controller
ATMega 16
I2C Master
RS232 (Slave)
Velocity
Velocity
Position
PID
Kf
Current Limit
PWM
Direction
H-Bridge
Current Sensing
M
Shaft Encoder


Micro Controller
ATMega 16
Signal Generator
&Runtime Measurement
I2C Slave
I2C Slave
Compass
PIC Micro

Appendix : Gearbox Photo



APPENDIX: Photo of the Power Electronics



APPENDIX