

---

---

# Obstacle Avoidance for a Differential Drive Robot

---

---

Project Report

ED5-8

Aalborg University  
Electronics and IT

Copyright © Aalborg University 2016

This paper was written using LaTeX.

The modelling was performed using Matlab.

Simulation was performed using Matlab's Simulink environment.



**Electronics and IT**

Aalborg University

<http://www.aau.dk>

## **AALBORG UNIVERSITY**

### STUDENT REPORT

**Title:**

Obstacle Avoidance for a Differential Drive Robot

**Theme:**

Automation

**Project Period:**

Fall Semester 2016

**Project Group:**

ED5-8

**Participant(s):**

Philip Philev  
Mihkel Soolep

**Supervisor(s):**

Simon Pedersen

**Abstract:**

The rising development in automation has led to the almost unprecedented surge of electric autonomous vehicles. To better understand the technologies used in the fields of autonomous vehicles and the smaller scale mobile robotics, this project was carried out. A model describing and analysing the behaviour of a differential drive robot was created, alongside a theoretical PID controller. The application was conducted on a described set of hardware, which was later programmed using the Raspberry Pi platform. The development process includes basic software and leads to the implementation of obstacle avoidance

**Copies:** 0

**Page Numbers:** 48

**Date of Completion:**

June 12, 2017

*The content of this report is freely available, but publication (with reference) may only be pursued due to agreement with the author.*



# Contents

<b>Preface</b>	<b>ix</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Mobile robotics . . . . .	2
1.2 Motivation and summary of project . . . . .	3
<b>2 Chapter 2 name</b>	<b>5</b>
<b>3 Modeling</b>	<b>7</b>
3.1 Differential Drive . . . . .	7
3.1.1 Non-holonomic constraints . . . . .	8
3.2 Kinematics . . . . .	8
3.3 Dynamics . . . . .	9
3.4 Complete Model . . . . .	11
3.5 Dead reckoning . . . . .	11
3.6 System Parameters . . . . .	13
<b>4 Hardware</b>	<b>15</b>
4.1 Single board computer . . . . .	15
4.2 Distance measuring . . . . .	16
4.3 DC motors and chassis . . . . .	17
4.4 Motor driver . . . . .	17
4.5 Speed sensor . . . . .	18
4.6 Wi-Fi adapter . . . . .	19
4.7 Assembly . . . . .	19
<b>5 Development</b>	<b>21</b>
5.1 Ultrasonic sensors . . . . .	21
5.2 DC motors . . . . .	23
5.3 Raspberry Pi configuration and software . . . . .	25
5.3.1 Raspberry setup . . . . .	25
5.3.2 Software . . . . .	25

5.3.3	Ultrasonic sensor reading . . . . .	27
5.3.4	Movements . . . . .	28
5.3.5	Main loop . . . . .	30
5.4	N.B. . . . .	32
<b>6</b>	<b>Conclusion and Future Development</b>	<b>33</b>
<b>A</b>	<b>Driver Chip</b>	<b>35</b>

# Todo list

<div></div> I think this word is misspelled . . . . .	5
Figure: We need a figure right here! . . . . .	5
<div></div> Reference . . . . .	8
<div></div> insert vehicle figure . . . . .	8
<div></div> ref book Modelling.Planning and Control . . . . .	11
<div></div> ref Robotics book . . . . .	12
<div></div> reference to book . . . . .	12
<div></div> include all parameters . . . . .	13





# Preface

Aalborg University, June 12, 2017

---

Philip Philev  
<pphile14@student.aau.dk>

---

Mihkel Soolep  
<msoole14@student.aau.dk>



# Chapter 1

## Introduction

**The Future of Vehicle Automation** In recent years, a big emphasis has been put on the development of autonomous or semi-autonomous ground vehicles. It comes as no surprise considering it is no longer a question of *will* this technology be implemented, but rather *when*. The benefits of autonomous vehicle integration can be considered invaluable. Currently 90% of motor vehicle fatalities are estimated to be due to human errors, meaning that vehicle automation could result in substantial decrease of accidents. Furthermore, depending on the percentage of autonomous vehicles on the roads, a research concluded, a drastic reduction in traffic and congestions. [DriverlessCar]

Nonetheless, there is still much work to be done in perfecting the control as well as the sensing capabilities of autonomous ground vehicles, if they are to become the default means of automotive transportation. Some of the issues consist of environmental conditions, which may disturb the sensors accuracy; precise mapping awareness, such as live maps that update when there is ongoing maintenance of infrastructure etc.; improved sensing capabilities (e.g advanced lidars) that can differentiate road damage, liquid spills etc.; ethical choices (as when an accident cannot be avoided), choosing to minimize potential damage and avoid casualties. [DriverlessCar]

**Levels of Automation** Automated vehicles, as defined by the *National Highway Traffic Safety Administration* (NHTSA - USA), are ones in which at least some aspects of a safety-critical control function occurs without the operator's direct input. (e.g steering, throttle, braking etc.) As such they are classified by the NHTSA in five levels: [NHTSA]

- **Level 0 - No Automation**

Logically, this level does not include any direct automation functions, however it may include some warning systems such as blind spot monitoring. The operator has the complete control over the vehicle.

- **Level 1 - Function Specific Automation**

The system may utilize one or more control functions operating independently from each other, such as cruise control or dynamic brake support. Nevertheless the driver has over control and can limit the functions of the supported aid systems.

- **Level 2 - Combined Function Automation**

The system utilizes at least two primary control functions, intercommunicating with each other in order to allow the operator's disengagement from physical operation of the vehicle. An example of such is a combination between *adaptive cruise control* and *lane centering*. The driver is still responsible for monitoring the environment, even when automated operating mode is enabled.

- **Level 3 - Limited Self-Driving Automation**

The driver accepts to cede full control of all safety-critical functions under certain conditions, and rely completely on the vehicle to monitor the environment if a transition toward manual control is required. Such level of control is observed in automated or self-driving vehicles that conclude when the system is unable to handle an environment, such as road construction site, requiring specific manoeuvres. The driver is not expected to fully pay attention to the road, but is advised to pay attention to sudden changes.

- **Level 4 - Full Self-Driving Automation**

Vehicle is designed to solely operate all safety-critical functions and supervise road conditions. Apart from providing destination input, the driver is not expected to maintain control at any point of the trip.

Currently **Level 4** automation is in active development stage, which means it won't be long until every newly manufactured vehicle has an above level 1 degree of automation.[NHTSA]

## 1.1 Mobile robotics

While autonomous cars are becoming closer to "computer on wheels" rather than mechanical cars, as Elon Musk said in an interview, mobile robots could be viewed as the harbinger of this new era of automation.[ElonMusk] Most systems, present in autonomous cars, are a scaled-up version of what has been present in robotics for a considerable time. Furthermore, an error or a failure of a system in an autonomous car would have much larger consequences than a system failure in domestic cleaning robot per se. That is why, the general motivation for this project has been to model, analyse and develop a wheeled mobile robot, resulting in knowledge that could later be applied in the more "mature" industry of autonomous vehicles.

## **1.2 Motivation and summary of project**

This paper is concerned with the development of a feedback control and obstacle avoidance in a differential drive robot. The information is distributed in five chapter, which individually cover most of the information needed to recreate the results obtained in this paper. Further development would increase the accuracy of developed model, as well as the physical system.



## Chapter 2

# Modeling

This chapter is concerned with the mathematical model of the wheeled robotic vehicle and will serve as a pre-requisite for the later implemented feedback control algorithm. The dynamics part, describing the motor model, will be used in the cruise control derivation. Subsequently the kinematics model, which treats the robot as a point entity, will be used in the estimation for position control.

### 2.1 Differential Drive

Differential drive steering, a popular choice in mobile robots, is a design where two wheels on a same axle are controlled independently. It may or may not include a castor wheel, however this paper addresses the latter. Depending on the relative rotation of each wheel, the robot could be steered in a desired manner. For the sake of clarity, several basic cases of interest are presented along with the equations describing the linear and angular velocity of the vehicle.

$$v = \frac{Rw_r + Rw_l}{2} \quad (2.1)$$

$$W = \frac{Rw_r - Rw_l}{l} \quad (2.2)$$

Equation 3.1 describes the linear speed of the robotic vehicle, with respect to the angular speeds of each wheels ( $w_r$ ) and ( $w_l$ ) and the wheel radius ( $R$ ), while 3.2 describes the angular speed of the robot, with  $l$  being the axle length between the wheels. The equation could be rewritten in matrix form as:

$$\begin{pmatrix} v \\ \omega \end{pmatrix} = \begin{pmatrix} \frac{R}{2} & \frac{R}{2} \\ \frac{R}{l} & -\frac{R}{l} \end{pmatrix} \begin{pmatrix} \omega_r \\ \omega_l \end{pmatrix} \quad (2.3)$$

Analysis of the above equation leads to the consideration of three cases where certain behaviour is to be expected.

- $v_r = v_l$   
In this case scenario, both wheels have the same speed. The robot's speed from equation 3.13 is simply equal to the individual speed of each of wheel. On the other hand, the angular speed from equation 3.14 becomes 0, and the turning radius infinite. The robot is expected to perform straight linear motion.
- $v_r = 0$  or  $v_l = 0$   
In this case scenario, the turning radius becomes  $\frac{l}{2}$ . The robot is expected to perform rotation either about the right or the left wheel, with the center of rotation being the zero velocity wheel.
- $v_l = -v_r$   
In this case scenario, the turning radius and the linear speed become 0, while the angular speed is doubled. The robot is expected to perform rotation about it's midpoint, or simply put in-place rotation.

### 2.1.1 Non-holonomic constraints

A robot, using the differential drive steering design, has restricted local movements, while having no restrictions in global motion. That is similar to the restrictions on a car, where it is not possible to slide into a parallel position with respect to the current one. Nevertheless, the vehicle can still manoeuvre into the desired position, similar to parallel parking.

If we consider we can define the non-holonomic constraints of a differential drive vehicle as:

$$\dot{x}\sin(\theta) - \dot{y}\cos(\theta) = 0 \quad (2.4)$$

As observed from equation 3.4 and shown by figure 3.1, the robot can not have speed perpendicular to the sagittal axis. The equation can be further rewritten as:

$$\tan \theta = \frac{\dot{y}}{\dot{x}} = \frac{dy}{dx} \quad (2.5)$$

## 2.2 Kinematics

The kinematic model, similar to most rigid vehicles travelling on a two-dimensional plane, is of three degrees of freedom. From figure 3.1, we can identify  $x$  and  $y$  as the coordinates for the center of mass of the robot and  $\theta$  as the angle with respect

Reference

insert vehicle figure



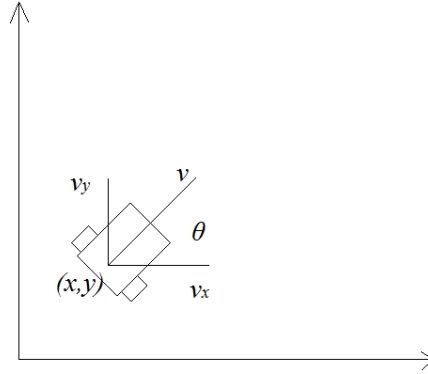


Figure 2.1: WMR orientation

to the x axis. From the constraints of the vehicle, the equations concerning the kinematic model are derived as:

$$\begin{aligned}\dot{x} &= v \cos(\theta) \\ \dot{y} &= v \sin(\theta) \\ \dot{\theta} &= \omega\end{aligned}\tag{2.6}$$

Where  $v$  is the linear and  $\omega$  is the angular speed of the vehicle. The equations could further be represented in matrix form.

$$\begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{pmatrix} = \begin{pmatrix} \cos \theta & 0 \\ \sin \theta & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} v \\ \omega \end{pmatrix}\tag{2.7}$$

## 2.3 Dynamics

In this section the equations used in the dynamics model of the motor are derived and described.

A DC motor produces mechanical torque applied to the shaft( $T_e$ ) linearly proportional to the armature current, the magnetic field and a torque constant( $K_t$ ). Assuming constant magnetic field, the produced torque is proportional to the torque constant( $K_t$ ) and the armature current( $I$ ). (Equation 3.1) Furthermore, the back Emf (electromotive force)( $E_b$ ) is equal to the angular velocity( $\omega$ ) of the shaft multiplied with the Back emf constant( $K_b$ ). (Equation 3.2)

$$T_e = IK_t \quad (2.8)$$

$$E_b = \omega K_b \quad (2.9)$$

Because the two constants  $K_t$  and  $K_b$  are equal in SI units, in further equations and simulations they will be denoted only as a motor constant  $K$ .

Using Kirchhoff's voltage law, the equations modelling the electrical part of the DC motor are derived as:

$$V = RI + L \frac{dI}{dt} + E_b \quad (2.10)$$

where ( $V$ ) is the applied voltage, ( $R$ ) the armature resistance, ( $L$ ) the armature inductance, ( $E_b$ ) the Back EMF. 3.4 Substitution ( $E_b$ ) with 3.8 yields:

$$V = RI + L \frac{dI}{dt} + K_b \omega \quad (2.11)$$

The mechanical part of the DC motor (mechanical part of figure ??) is described by Newton's second law for rotational motion as:

$$J\dot{\omega} = K_t I - b\omega \quad (2.12)$$

Where  $J$  is the load's inertia and  $b$  is the viscous friction in the motor's bearings. Further substitution in equation 3.4 with the derived back emf from 3.2 results in:

Equations 3.10 and 3.11 are the equations describing the electrical and mechanical part of the motor.

Applying the Laplace transform to the equations 3.10 and 3.11, we can relate the output angular velocity to the input voltage as described in the transfer function:

$$\frac{\Omega(s)}{V(s)} = \frac{K_t}{(Js + b)(sL + R) + K_t K_b} \quad (2.13)$$

The state space form representation of equations 3.10 and 3.11 is represented in the form  $\dot{x} = Ax + Bu$ , where the state variables are the current ( $I$ ) and the angular speed ( $\omega$ )

$$\begin{pmatrix} \dot{I} \\ \dot{\omega} \end{pmatrix} = \begin{pmatrix} \frac{-b}{L} & \frac{-K_t}{L} \\ \frac{-K_b}{L} & \frac{-R}{L} \end{pmatrix} \begin{pmatrix} I \\ \omega \end{pmatrix} + \begin{pmatrix} \frac{1}{L} \\ 0 \end{pmatrix} V \quad (2.14)$$

The Simulink block diagram describing the behaviour of the DC motor in continuous time is given in figure 3.2.

The motor model is a SISO system, where the input is Voltage and the output is angular velocity. Thus, classical control is sufficient to manipulate the behaviour of the system.

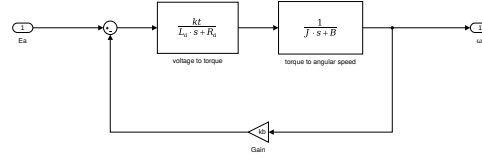


Figure 2.2: Motor block diagram

## 2.4 Complete Model

Combining the motor dynamics, the the robot kinematics yield the complete plant of the system.

## 2.5 Dead reckoning

If we consider the robot's configuration  $\mathbf{q}$ , previously described as  $(x, y, \theta)$ , a way of measuring the above mentioned configuration is required in order to implement planning and feedback control. Incremental encoders are common in wheeled robots as they are able to measure the rotation of the wheels. However, they do not provide direct measurements of the position and orientation of the robot, thus dead reckoning (aka. odometric localization) is an affordable way to estimate the vehicle's configuration  $\mathbf{q}$  by iterative integration of the kinematic model from 3.2. The major drawback of the method is the proneness to error, that is if there exists a small error in any step of the localization it will influence the calculation of the subsequent positions, leading to potentially large deviation between the expected position and the localization results. From that we can conclude that dead reckoning is less precise in large areas due to systematic errors( eg. limited encoder resolution, unequal wheel diameters, etc.), and external factors not considered in the kinematics due to non-systematic errors( eg. wheel slippage on floors, inclined surfaces etc.)

Assuming constant inputs  $v_k$  and  $\omega_k$  in the kinematic model 3.2 for discrete time  $[t_k, t_{k+1}]$  (zero-order hold), for the sampling step  $\mathbf{k} = 1, 2, \dots, n$ , the differential drive robot will traverse along an arc with a circle radius  $\mathbf{D} \left( \frac{v_k}{\omega_k} \right)$ . Furthermore if the starting position  $(x_k, y_k, \theta_k)$ , for  $\mathbf{k} = 1$ , and the velocities  $v_k$  and  $\omega_k$  are known, we can reconstruct the value of  $(x_{k+1}, y_{k+1}, \theta_{k+1})$  by forward integration of the kinematic

model at the sampling time  $t_{k+1}$ . Several methods for numerical integration could be used, among which are the Forward Euler method, Second-order Runge-Kutta method or exact integration. . Because of inadequate accuracy due to the constant orientation  $\theta_k$  in the Euler method, in this paper we will consider the Runge-Kutta approximation method and exact integration as described in the book "Robotics: Modelling, Planning and Control" by Bruno Siciliano.

ref Robotics book

$$x_{k+1} = x_k + T_s v_k \cos(\theta_k + \frac{\omega_k T_s}{2}) \quad (2.15)$$

$$y_{k+1} = y_k + T_s v_k \sin(\theta_k + \frac{\omega_k T_s}{2}) \quad (2.16)$$

$$\theta_{k+1} = \theta_k + T_s \omega_k$$

Equation 3.16 refers to the second-order Runge-Kutta method, where  $T_s$  is the duration of the sampling period  $T_s = t_{k+1} - t_k$ . It is important to note that the approximations for  $x_{k+1}$  and  $y_{k+1}$  use the average value of the unicycle orientation in the sampling period, compared to the less accurate Euler method with constant orientation.

The exact reconstruction of  $(x_{k+1}, y_{k+1}, \theta_{k+1})$ , assuming constant velocity inputs in the sampling interval is given by ??:

reference to book

$$x_{k+1} = x_k + \frac{v_k}{\omega_k} (\sin(\theta_{k+1}) - \sin(\theta_k)) \quad (2.17)$$

$$y_{k+1} = y_k + \frac{v_k}{\omega_k} (\cos(\theta_{k+1}) - \cos(\theta_k)) \quad (2.18)$$

$$\theta_{k+1} = \theta_k + T_s \omega_k$$

Note that the exact reconstruction is not defined for  $w_k = 0$  (motion in a line segment) thus, in implementation, a conditional statement has to handle the exception by using the Runge-Kutta method, which is still defined for  $w_k = 0$ .

Nevertheless, in practice it is convenient to estimate the input velocities using the incremental encoders and by calculating the relative displacement  $\Delta s$  and orientation  $\Delta \theta$  over the sampling period:

$$\begin{aligned} v_k T_s &= \Delta s \\ \omega_k T_s &= \Delta \theta \\ \frac{v_k}{\omega_k} &= \frac{\Delta s}{\Delta \theta} \end{aligned} \quad (2.19)$$

In the case of a differential drive robot with a castor wheel, from equation 3.1 and 3.2, we can define  $\Delta s$  and  $\Delta \theta$  as :

$$\begin{aligned}\Delta s &= \frac{r}{2}(\Delta s_r + \Delta s_l) \\ \Delta \theta &= \frac{r}{l}(\Delta s_r - \Delta s_l)\end{aligned}\tag{2.20}$$

Where  $\Delta s_r$  and  $\Delta s_l$  are the rotation of the right and left wheel, respectively, measured by the incremental encoders during the sampling period.

## 2.6 System Parameters

In this section the parameters used in the mathematical model of the robot are given. They were acquired by measurements and experiments on the physical system, prior to implementation.

Parameter	Description	Nominal Value
K	Motor constant	0.1838 V/(rad/s) Nm/amp
R	Armature resistance	11.5 $\Omega$
L	Armature inductance	0.1 H
$b_r$	Rotor damping	0.0221
$J_w$	Load inertia	2.8033e-5 KgM <sup>2</sup>
n	Gear ratio	1:48

**Table 2.1:** System parameters

include all parameters



## Chapter 3

# Control

In this chapter we will look more closely at the cruise controller developed for the robot. Cruise control is essential for the robot since it ensures, that it keeps following a straight line and in certain angle respect to the global co-ordinate axis. The motor model used in this chapter is the same one that we examined in the previous one. The before discussed motor model is completely linear, in what case we can apply classical control techniques in order to design the controller.

In the cruise control we want to manipulate the velocity and the angular speed of the robot. In order to do that we must be able to control each of the wheels angular speed namely  $\omega_r$  and  $\omega_l$ .

HERE GOES A FIGURE WHAT SHOWS ROBOT IN GLOBAL CO-ORDINATES

In the following figure you can see the closed loop block diagram for the motor speed control. The plant in this diagram is the same motor model as we looked in the previous chapter.

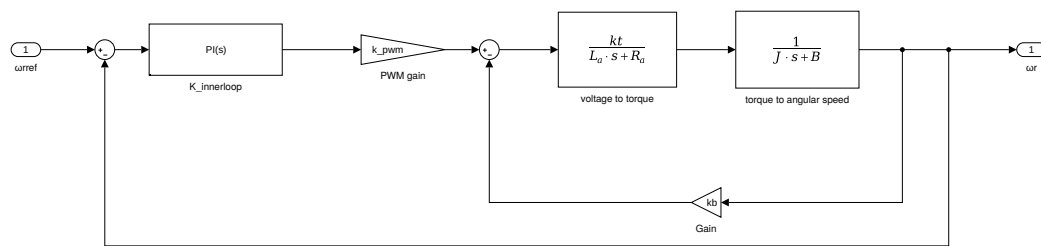
Now we will look more closely what is in that diagram. First we have the plant model what we are already familiar. Next we have the  $K_{innerloop}$ , what is the PI controller what we have designed to meet our desires. From the diagram we can see that we will have an  $P_{innerloop}$  what is the motor model and the PWM term  $k_{pwm}$  put together. As a result we get the following equation:

$$P_{innerloop} = \frac{k \cdot k_{pwm}}{k^2 + (sL_a + R_a)(sJ + B)} = \frac{3300}{(s + 1.02)(s + 5000)} \quad (3.1)$$

Next we will look at the  $K_{innerloop}$ . This is the PI controller what we have developed for the system in order to meet the desired specifications.

$$K_{innerloop} = \frac{g(s + z)}{s} * \left[ \frac{100}{s + 100} \right]^2 \quad (3.2)$$

Following we can see that the open loop transfer function is given by the following equation:



**Figure 3.1:** Motor speed control



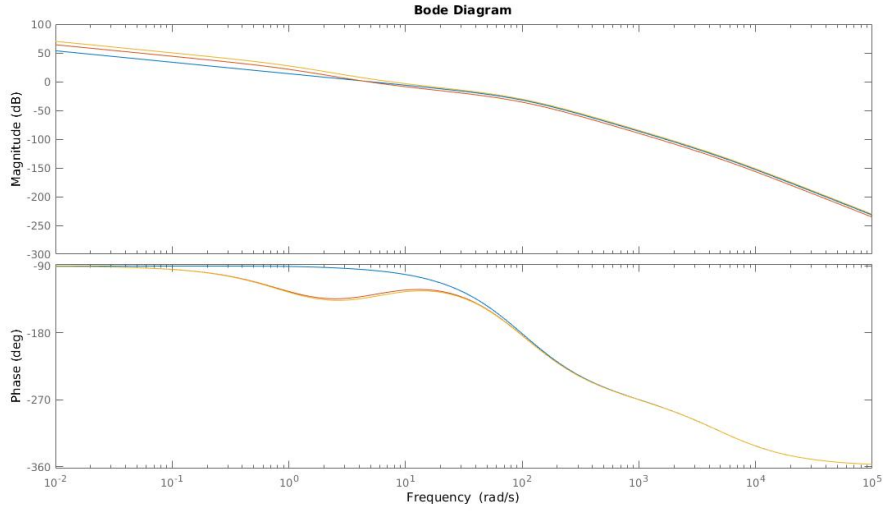


Figure 3.2:  $G_{innerloop}$  bode plot

$$G_{innerloop} = P_{innerloop} * K_{innerloop} \quad (3.3)$$

and the closedloop transfer function we get:

$$H_{innerloop} = \frac{G_{innerloop}}{G_{innerloop} + 1} \quad (3.4)$$

For the simulations we used three different values on the  $g$  and  $z$ . For the first simulation we used  $z=1$  and  $g=7$ . In the second simulation we used  $z=5$  and  $g=5$ . for the final one we used  $z=10$  and  $g=6$ .

The bode plots of the simulation can be seen below.

From the next figure you can see the step responses of the three different settings on controllers.

Previously we looked at the control design of the seperate angular velocities. In this section we will be taking a closer look on the complete architecture of the cruise control. The whole cruise control layout can be seen in figure ??.

In order to develop the architecture of the cruise control we first must look how can we obtain  $\omega_{rref}$  and  $\omega_{lref}$  by manipulating  $\omega$  and  $v$ .

$$\begin{pmatrix} \omega_{rref} \\ \omega_{lref} \end{pmatrix} = \begin{pmatrix} \frac{1}{R} & \frac{L}{2R} \\ \frac{1}{R} & -\frac{L}{2R} \end{pmatrix} \begin{pmatrix} v_{ref} \\ \omega_{ref} \end{pmatrix} \quad (3.5)$$

$$M^{-1} = \begin{pmatrix} \frac{1}{R} & \frac{L}{2R} \\ \frac{1}{R} & -\frac{L}{2R} \end{pmatrix} \quad (3.6)$$

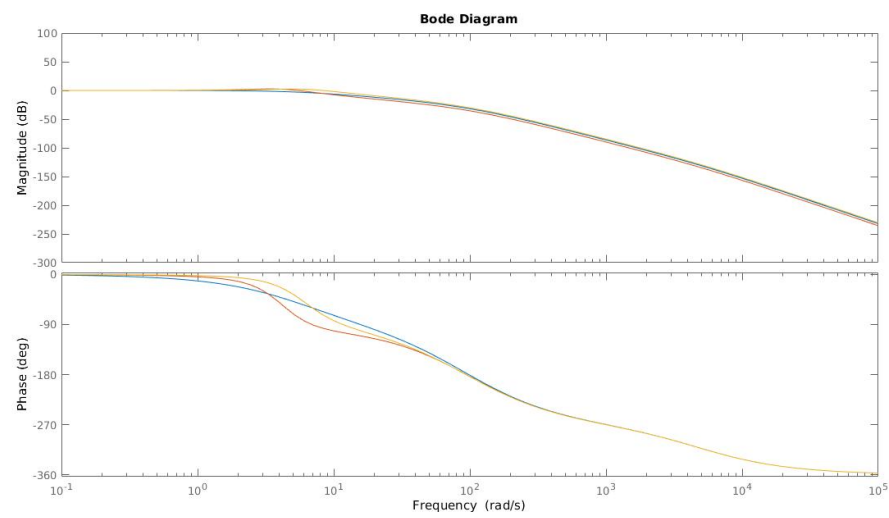


Figure 3.3:  $H_{innerloop}$  bode plot

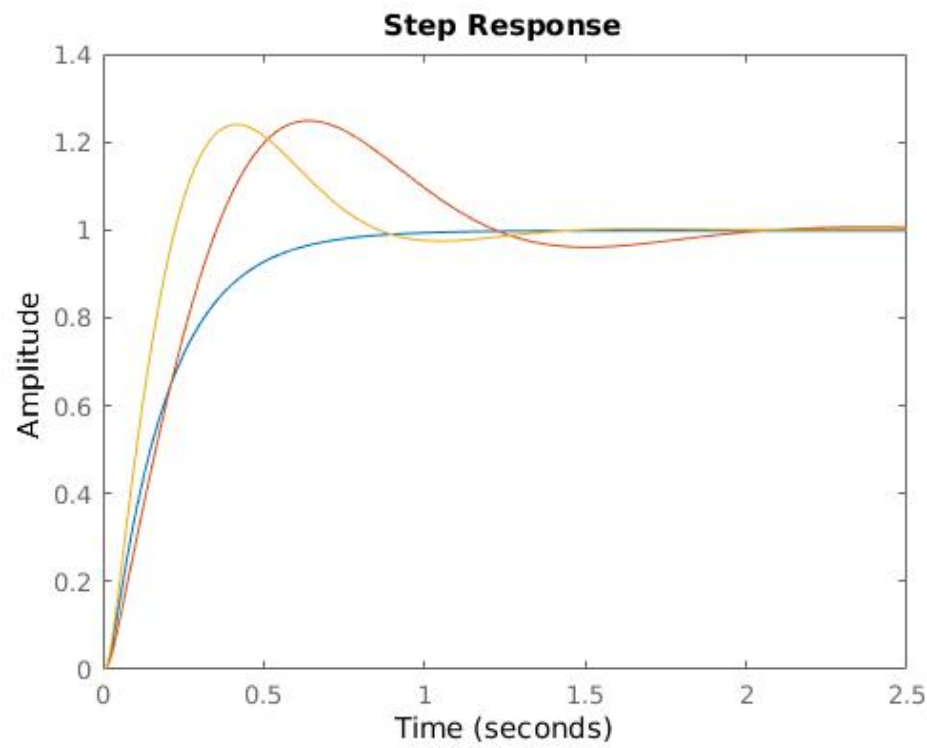


Figure 3.4: Step response

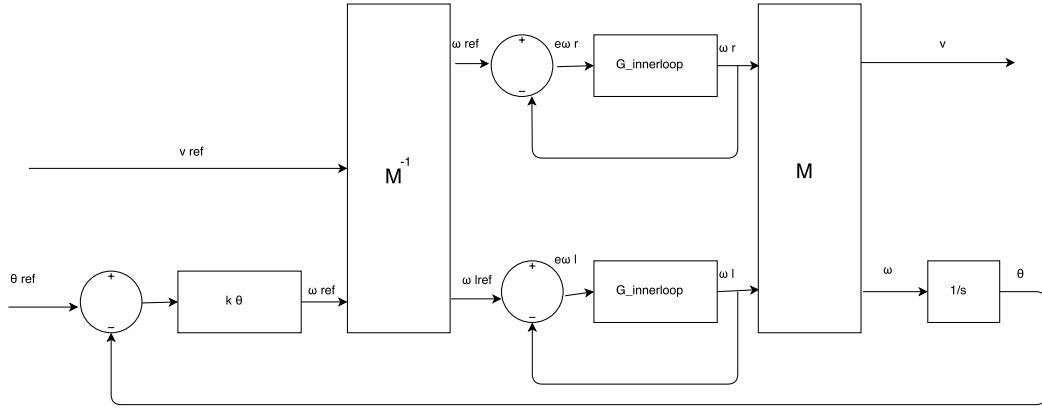


Figure 3.5: Cruise control full architecture

In the equations ?? and ?? we have  $\omega_{rref}$  and  $\omega_{lref}$  what are the separate wheels angular speeds what are commanded by the system.  $v_{ref}$  is the linear speed on the robot and  $\omega_{ref}$  is the desired angular speed of the whole robot.  $R$  is the radius of the wheels and  $L$  is the length between motorized wheels.

By analyzing the equations we can see that the transfer functions from reference linear speed( $v_{ref}$ ) to the linear speed ( $v$ ). Similarly is the transfer function from  $\omega_{ref}$  to  $\omega$ . This will tell us that the  $v_{ref}$  to  $v$  and  $\omega_{ref}$  to  $\omega$  time and frequency responses are the same as are the motor speed responses. Using that knowledge we can now obtain new equations?? and ??

$$\begin{pmatrix} v(s) \\ \omega(s) \end{pmatrix} = M^{-1} \begin{pmatrix} H_{innerloop}(s) & 0 \\ 0 & H_{innerloop}(s) \end{pmatrix} M \begin{pmatrix} v_{ref}(s) \\ \omega_{ref}(s) \end{pmatrix} \quad (3.7)$$

$$\frac{v(s)}{v_{ref}(s)} = \frac{\omega(s)}{\omega_{ref}(s)} = H_{innerloop}(s) \quad (3.8)$$

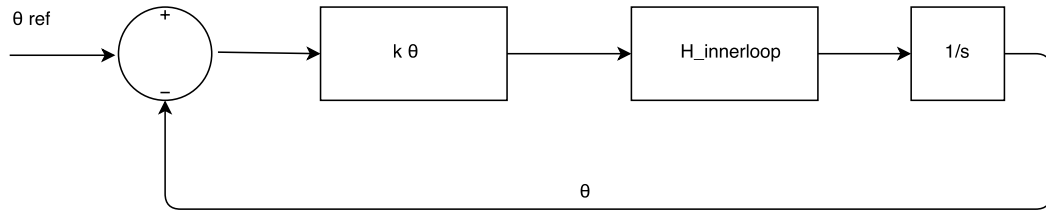
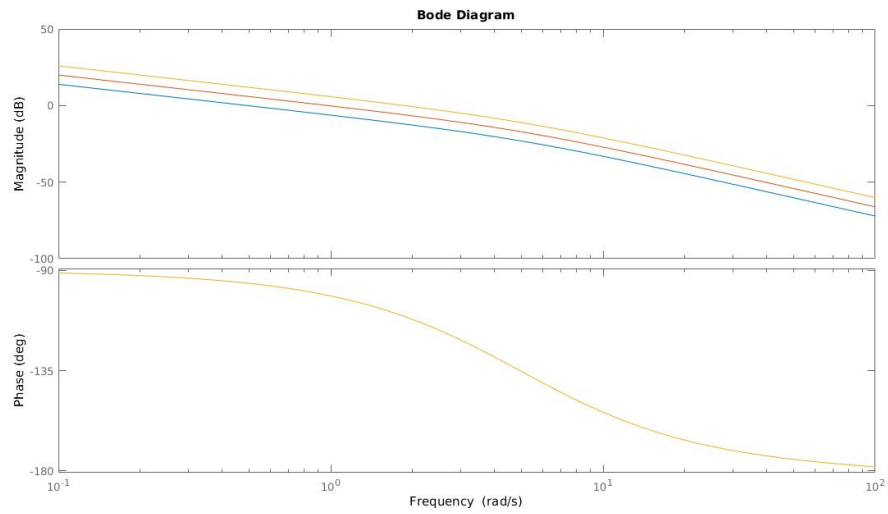
In the equations ?? and ?? the  $v_{ref}$  is the linear velocity command and the  $\omega_{ref}$  is the angular speed command for the robot.

Next part of the cruise control is to make the robot follow  $\Theta_{red}$  the commanded angle. Examining the figure ?? we can see that the plant for the angle control is given by

$$P_{\Theta} = \frac{H_{innerloop}}{s} = \frac{24570}{(s + 4995)(s + 4.957)s} \quad (3.9)$$

For simpler understanding we will simplify the angle control block diagram what can be seen in the figure ??

For controlling the  $\Theta$  we will use a proportional controller. The form of the controller  $k_{\theta}$  is shown in the equation ??.

Figure 3.6:  $\Theta$  controlFigure 3.7:  
 $G_\theta$  bode plot

$$k_\theta = g \quad (3.10)$$

Furthermore we can obtain the transfer function for the open loop loop ( $G_\theta$ ) what is given by

$$G_\theta = P_\theta k_\theta \quad (3.11)$$

From here we can get the closed loop loop transfer function, what we will call  $H_\theta$  shown in the equation ??

$$H_\theta = \frac{G_\theta}{G_\theta + 1} \quad (3.12)$$

In the figure ?? we can see the bode plot of the  $G_\theta$ . In figure ?? we have the bode plot of the  $H_\theta$ . And on the figure

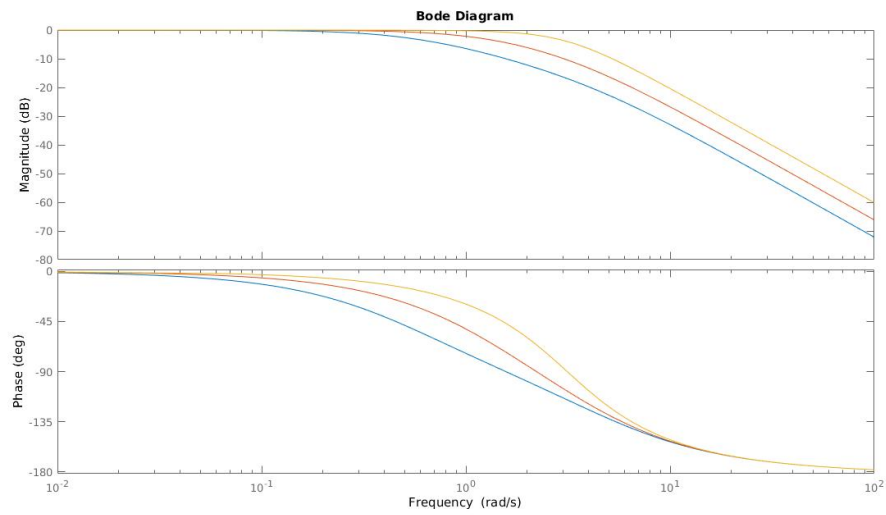


Figure 3.8:  $H_\theta$  bode plot

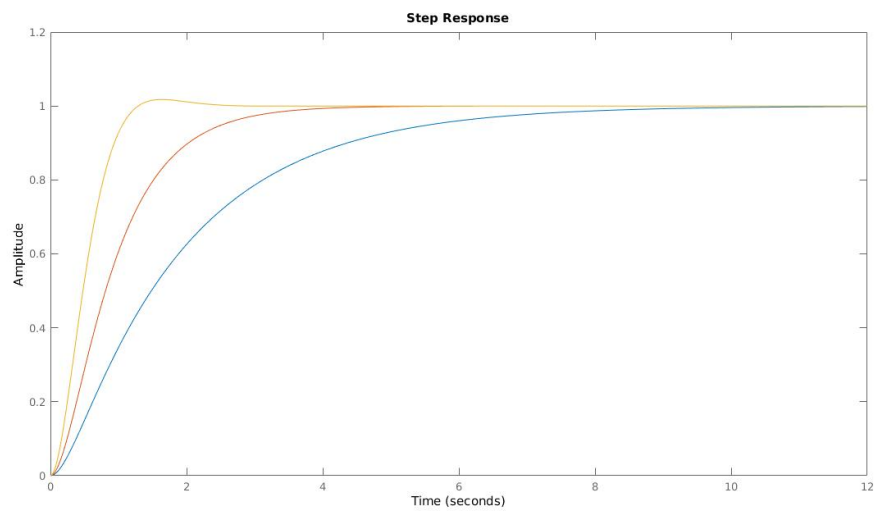


Figure 3.9:  $\Theta$  step response

A system is said to be controllable if there exists a control law  $u(\cdot)$  which can transfer the state of the system from any initial state  $x_0$  to any final state  $x_f$  within a finite amount of time. Otherwise the system is said to be uncontrollable  
[REFERENCE TO sAME BOOK HE HAD](#)

We will now look more closely the kinematics model and determine if it is controllable or not. The kinematic model is described with the following equations.

$$\begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\Theta} \end{pmatrix} = \begin{pmatrix} \cos \Theta & 0 \\ \sin \Theta & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} v \\ \omega \end{pmatrix} \quad (3.13)$$

Since we have the  $\sin \Theta$  and  $\cos \Theta$  inside the system it is non linear system.

To determine the controllability of the system we must check the sufficient conditions what is given by

$$\text{rank}(h_1, h_2, [h_1, h_2]) = \text{rank} \begin{pmatrix} \cos \Theta & 0 & \sin \Theta \\ \sin \Theta & 0 & -\cos \Theta \\ 0 & 1 & 0 \end{pmatrix} = n = 3 \quad (3.14)$$

where

$$[h_1, h_2] = \frac{\partial h_2}{\partial p} h_1 - \frac{\partial h_1}{\partial p} h_2 \quad (3.15)$$

Since we can see that  $m=n=3$  we can determine that the system is controllable. This shows us that indeed you can move to any position on the given plain with the robot.

Before we had the kinematic model in the non linear form. Now we will linearize the model about the equilibrium and we obtain the following linearized model of the kinematics.

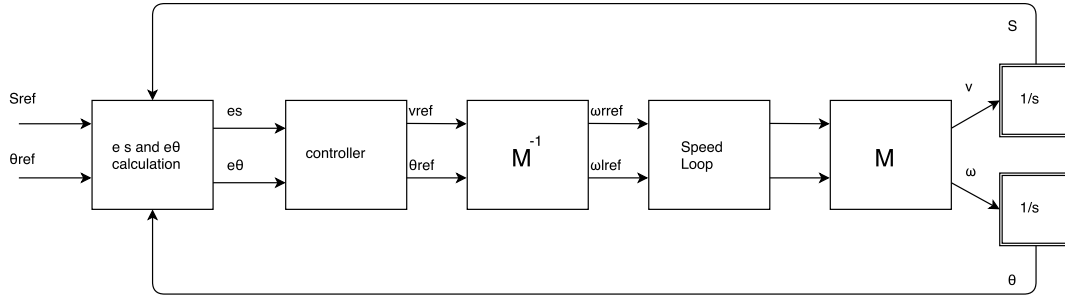
$$\begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\Theta} \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} v \\ \omega \end{pmatrix} \quad (3.16)$$

After the linearization we can see that the rank of the controllability matrix is less then  $n$ . From that we can only determine that the controllability of the system has been lost after the linearization.

Cartesian stabilisation control is necessary for the robot because it makes sure that the robot goes to the intended point. In the [ref to astolfi book](#) it has been explained the transformation method of the posture stabilization. We will be using the same method for the cartesian stabilization.

[next subchapter](#)

First we need to transform the kinematic model ?? in the terms of the angular and linear displacements.



**Figure 3.10:** Cartesian stabilization

$$\dot{s} = v\dot{\Theta} = \omega \quad (3.17)$$

Next we define a new state vector  $\rho$

$$\rho = \begin{pmatrix} s \\ \Theta \end{pmatrix} \quad (3.18)$$

Furthermore we will rewrite the state equations as  $\dot{\rho}$ .

$$\dot{\rho} = \begin{pmatrix} v \\ \omega \end{pmatrix} \quad (3.19)$$

The transformed system will be used in order to make the robot go towards the goal point. Figure ?? shows this system.

As we can see from the figure ?? we would need to get the value of  $S_{ref}$ . On closer examination we can say that the value of  $S_{ref}$  is useless for us and measuring the  $S$  is difficult. The problem can be dealt with some manipulations as it is explained in [Vieira, F.C.; Medeiros, A.A.D.; Alsina P.J.; Araujo A.P. REF THIS](#)





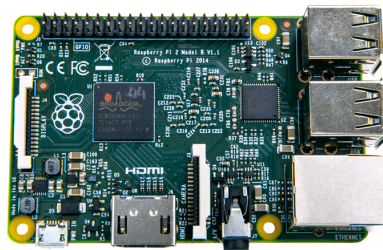
## Chapter 4

# Hardware

In the following chapter the hardware used to create the robot is discussed. The first decision made was what platform to use, based on the available resources and knowledge of use.

### 4.1 Single board computer

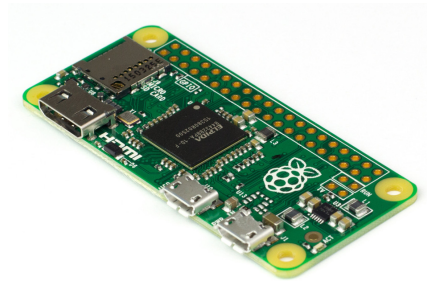
The Raspberry Pi is the platform of choice used in the development of this project. Several factors influenced the choice, the most prominent of which was the familiarity the single board computer offered. Previous usage of Raspberry Pi 2B, allowed for faster decision making, when exploring for potential resources.



**Figure 4.1:** Raspberry Pi 2B

The initial choice was a Raspberry Pi 2B, however, after consideration of the size and required power consumption, a choice was made to use Raspberry Pi Zero instead.

Performance of the two computers is almost similar, with the exception that the Zero model has less RAM compared to its 2B counterpart. Nevertheless, no significant impact was registered.



**Figure 4.2:** Raspberry Pi Zero

Specs of the Raspberry Pi Zero: 1Ghz, Single-core CPU 512MB RAM Mini HDMI and USB On-The-Go ports Micro USB power HAT-compatible 40-pin header Composite video and reset headers

## 4.2 Distance measuring

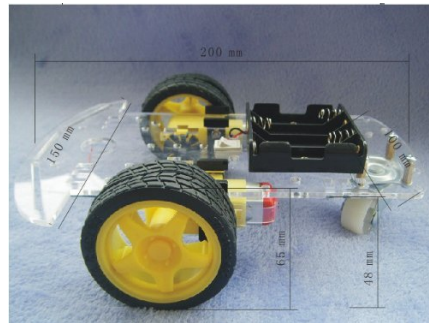
Since the main purpose of the device would be to avoid obstacles, distance measuring sensors are required. There are several choices when it comes to distance sensors, vastly ranging in price and accuracy. The optimal choice would be a Lidar, however, at the time of development of this paper, the prices for Lidar were too high for practical implementation in a consumer application. As a result, the choice was made to use a set of ultrasonic sensors, which scaled well according to price and accuracy of measurements. Three HC-SR04 ultrasonic sensors were used for distance measuring purposes in the device. The positioning of the sensors is on the front, and on the sides.



**Figure 4.3:** Ultra sonic sensor HC-SR04

### 4.3 DC motors and chassis

Considering the body of the robot, an inexpensive set of DC motors with a chassis was purchased and assembled. It comes with an attachable tachometer wheel for the motor, in order to incorporate a rotary encoder in the design, crucial for regulating the speed of the individual wheels.[Motorref]



**Figure 4.4:** Chassie and the two DC motors with a power supply

Motor specs:

Voltage: DC 3V DC 5V DC 6V Current: 100 MA 100MA 120MA Reduction rate:48:1 RPM (With tire):100,190,240 Tire Diameter:66mm Car Speed(M/minute):20,39,48

Motor Weight (g):50

Motor Size:70mm\*22mm\*18mm

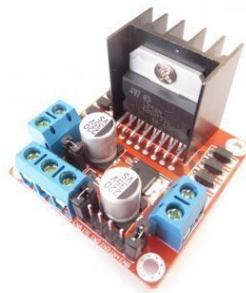
Noise:<65dB

The two motors are identical and are used to move, as well as steer, the device and to

### 4.4 Motor driver

At the beginning of the project a L9110S DC Stepper Motor Driver H-Bridge was used, capable of controlling the direction of rotation of the wheels, but it was soon discovered the suggested driver was not capable of regulating the motor speeds. To control the speed of the robot, a replacement with a L298N driver had to be performed. The second driver was capable of using the PWM to regulate the speed of the motors.

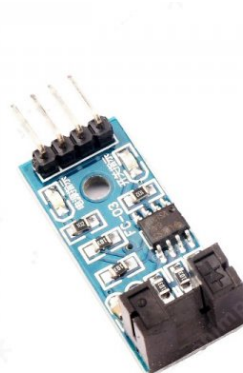
Driver specs: Working mode: H bridge (double lines) Control chip: L298N (ST) Logical voltage: 5V Driving voltage: 5V-35V Logical current: max 36mA Driving current: 2A (max single bridge) Maximum power: 25W Storage temperature: -20 C +135 C Periphery dimension: 43 x 43 x 27mm(L x W x H)



**Figure 4.5:** The L298N driver

## 4.5 Speed sensor

The speed of the wheels is measured by the LM393 IR speed sensors attached to the tachometer wheel of the motor.[SpeedSensor]



**Figure 4.6:** The LM393 IR speed sensor

The speed sensor is used to estimate the error of the wheel speed. Features:

Working voltage: 3.3V 5V Weight: 8g Dimensions: Approx.3.2 x 1.4 x 0.7cm  
5mm Groove width Using wide voltage LM393 comparator Application: Widely  
used in dynamo speed detecting, pulse counting, etc Output form: Digital switch  
output (0 and 1) and Analog for Sensitivity.

## 4.6 Wi-Fi adapter

In order to perform wireless monitoring, as well as extend the connectivity, an Edimax EW-7811Un Wi-Fi adapter, was used.



**Figure 4.7:** Edimax EW-7811Un Wi-Fi adapter

## 4.7 Assembly

In this section, the schematics of the assembled components has been analysed and given in figure 4.8.

In the middle section of the schematics, the Raspberry Pi is present. Unfortunately, the software used to make this schematic did not have the Raspberry Pi Zero layout, so the model 2 was used instead. Regarding the project there is not a big difference, as the pins are the same as Zero.

On the right side of the system three ultrasonic sensors could be seen, connected to the Raspberry Pi. The Vcc and GND pins are all connected to the Raspberry, allowing the sensors to draw power from the microcontroller itself. The trigger pins are all connected to the same Raspberry pin, meaning that when one of the sensors is triggered all of them will emit sonic impulse. The triggers are all on the same pin to not limit the number of available pins on the Raspberry. However there is not a significant difference, as the echo pins of each of the sensors go into separate ports on the Raspberry and each of them has a voltage divider connected to the GND.

The driver is present in the bottom left of the schematics, alongside a battery pack connected to the driver and providing power to both motors. It is connected to the Raspberry by six pins, four responsible for the direction of the motors (2 for each motor) and 2 enable pins, used for controlling the speed of the motor by PWM.

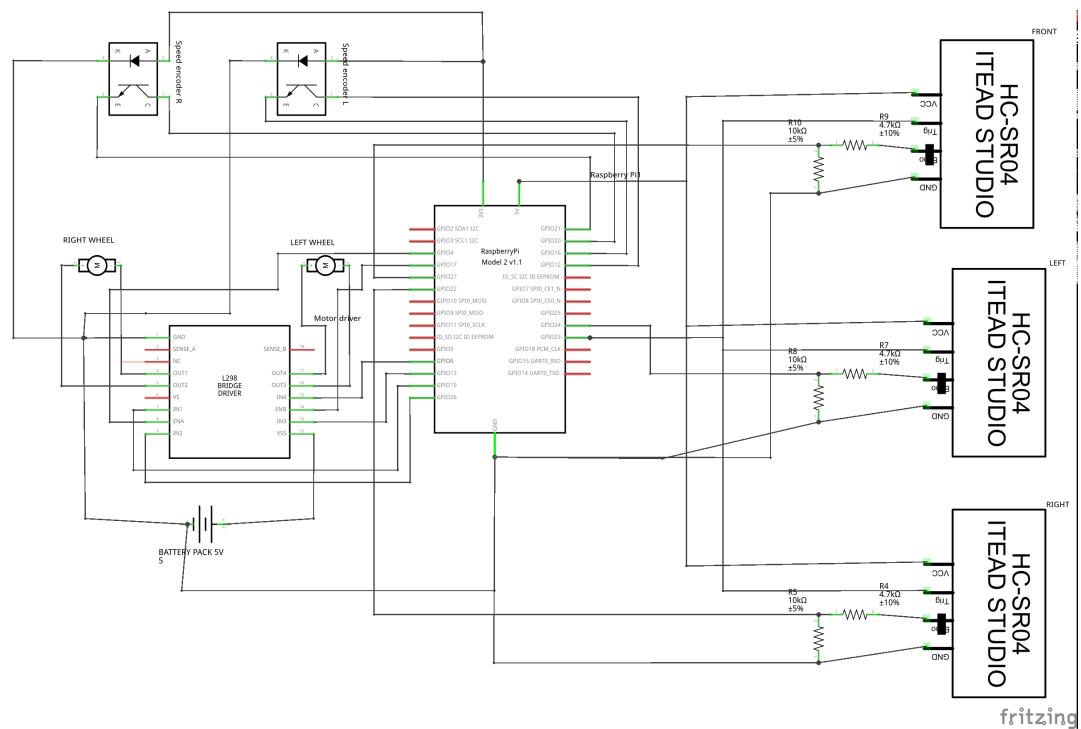


Figure 4.8: Schematics of the device

The motors are connected directly to the driver, by two wires. The two encoders, responsible for monitoring the speed of the wheels, are present in the top left of the schematics. An external power source was used for the Raspberry Pi, which is not present in this schematics.

## Chapter 5

# Development

### Hardware testing

#### 5.1 Ultrasonic sensors

The first part of the hardware outline was testing of the ultrasonic sensors. For each of the sensors a voltage divider was build as seen on figure /refvoltage1.

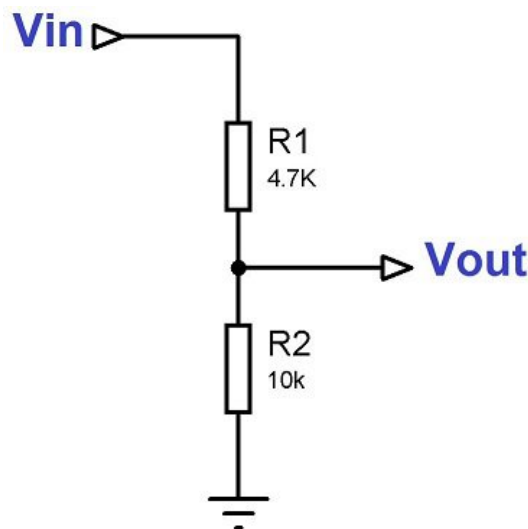


Figure 5.1: Voltage divider

The values of the resistors are calculated by the following equation:

$$V_{out} = V_{in} * R_2 / (R_1 + R_2) \quad (5.1)$$

All sensors were tested out separately by connecting them individually to the Raspberry pi and running the test code below

```

import RPi.GPIO as GPIO
import time
GPIO.setmode(GPIO.BCM)

TRIG = 23
ECHO = 24

print "Measuring distance"

GPIO.setup(TRIG, GPIO.OUT)
GPIO.setup(ECHO, GPIO.IN)

while True:
    GPIO.output(TRIG, False)
    print "Waiting for the sensor"
    time.sleep(2)

    GPIO.output(TRIG, True)
    time.sleep(0.00001)
    GPIO.output(TRIG, False)

    while GPIO.input(ECHO)==0:
        pulse_start = time.time()

    while GPIO.input(ECHO)==1:
        pulse_end= time.time()

    pulse_duration = pulse_end - pulse_start

    distance = pulse_duration * 17150
    distance = round(distance, 2)

    print "Distance:%d", distance

```

Each of the sensors worked as expected while connected separately, thus the logical progression was to test them out all at the same time. While taking the readings, it was noticeable that some of the values were random or distorted due to noise and could not be used for accurate measurement. Therefore, some filtering had to be included in order to scale the values to an acceptable limit. A moving average filter was used for the previously mentioned task by taking three readings at a time and calculating the average of the values.

```
def readsensor(PIN):
```



```

    for x in range(0, 2):
        read_time_start1 = time.time()
        GPIO.output(TRIG, True)
        time.sleep(pulse)
        GPIO.output(TRIG, False)

        while GPIO.input(PIN)==0:
            pulse_start = time.time()

        while GPIO.input(PIN)==1:
            pulse_end= time.time()

        pulse_duration[x] = pulse_end - pulse_start
        time.sleep(0.05-(time.time()-read_time_start1))

    distance = sum(pulse_duration)/measurment_count* SPEED_OF_SOUND
    distance = round(distance, 2)
    print distance
while True:
    readsensor(ECHOF)
    readsensor(ECHOR)
    readsensor(ECHOL)

```

From the code above, the application of the moving average filter could be seen. Importantly there was a noticeable improvement of the readings, which became much more precise and consistent. Furthermore, it can be seen that every reading takes less than 0.05 seconds. This is useful in making every cycle evenly long and predictable, when it comes to the total time that the program is required to run the full cycle. Therefore the time for the full sensor reading loop becomes  $3 \times 0.05s = 0,15s$ .

## 5.2 DC motors

To make sure the purchased DC motors were operational an initial testing was performed by driving each of them in forward and in backward gear through the used driver. This can be seen in script below.

```

GPIO.setmode(GPIO.BCM)
GPIO.setup(StepPinForward, GPIO.OUT)
GPIO.setup(StepPinBackward, GPIO.OUT)

def forward(x):

```

```

GPIO.output(StepPinForward , GPIO.HIGH)
print "forwarding running motor "
time.sleep(x)
GPIO.output(StepPinForward , GPIO.LOW)

def reverse(x):
    GPIO.output(StepPinBackward , GPIO.HIGH)
    print "backward running motor"
    time.sleep(x)
    GPIO.output(StepPinBackward , GPIO.LOW)

print "forward motor "
forward(5)
print "reverse motor"
reverse(5)

print "Stopping motor"
GPIO.cleanup()

```

As it can be seen from the code, one of the motors was initially driven forward for 5 seconds and subsequently backwards for 5 seconds. To apply the test for the second motor simply the pin numbers(StepPinForward and StepPinBackward) were changed.

Furthermore, Pulse Width Modulation (PWM) was utilized in order to regulate the separate speeds of the motors, as it was an initial condition to implement cruise control. in the final software what is ran on the device we have changed the forward() and reverse() so that we can change the speed of the motors at our desire.

```

def forward(forwardtime ,SPEED):
    print "REVERSE"
    GPIO.output(StepPinBackward1 , GPIO.HIGH)
    GPIO.output(StepPinBackward2 , GPIO.HIGH)
    PWML.start(SPEED)
    PWMR.start(SPEED)
    time.sleep(forwardtime)
    GPIO.output(StepPinBackward1 , GPIO.LOW)
    GPIO.output(StepPinBackward2 , GPIO.LOW)

```

As you can see from the code above we use the drivers PWM input to change the speed of the vehicle.

## 5.3 Raspberry Pi configuration and software

In this section, a description of the steps taken to configure the Raspberry Pi, was made.

### 5.3.1 Raspberry setup

Initially, after obtaining the Raspberry Pi Zero, an installation of the Raspbian OS was performed. Subsequently, all GPIO pins were enabled, followed by configuration of the Secure Shell (SSH) protocol in order to perform remote logins to the microcontroller. Furthermore, as mentioned in the Hardware chapter, due to its size, the Raspberry Zero has only one USB port, resulting in a space limitation. Thus, the logical choice was to connect the Wi-Fi adapter to that USB port, and configure the microcontroller remotely, through the SSH. Additionally, extra tools such as Tmux Multitab and Nano Text Editor were used to clarify the programming sessions.

### 5.3.2 Software

The software constituting the robot's behaviour is self-made with the addition of several external libraries

```
import sys
import time
import RPi.GPIO as GPIO
```

To be precise, only three external libraries were used at the time of the development of this paper. Nevertheless, as the system is still in active development the list may expand after the submission of the documentation.

- **Sys module**

This module provides a number of functions and variables that can be used to manipulate different parts of the Python runtime environment. [**sys\_module**]

- **Time module**

This module provides a number of functions to deal with dates and the time within a day. It is a thin layer on top of the C runtime library. A given date and time can either be represented as a floating point value (the number of seconds since a reference date, usually January 1st, 1970), or as a time tuple. [**time\_module**]

- **RPi.GPIO module**

This module is for functions that are connected to the GPIO pins.

The overall setup of the pins and the variables, with the different values used in the code, is present in the snippet below.

```
#PIN numbers
LetfPWM=16
RightPWM=20
StepPinForward1=26
StepPinBackward1=19
StepPinForward2=13
StepPinBackward2=6
ECHO_F=4
ECHO_L=27
ECHO_R=22
TRIG=17

#Values for reading the sensors
SPEED_OF_SOUND = 17150
measurement_count = 3
pulse = 0.00001
pulse_duration = [0,0,0]
sensorF_data=0
sensorR_data=0
sensorL_data=0

#navigation variables
reversetime=0
turningtime = 1
MAXSPEED = 1
MIDSPEED = 0.6
MINSPEED = 0.1

#GPIO setup for each pin
GPIO.setmode(GPIO.BCM)
GPIO.setup(StepPinForward1, GPIO.OUT)
GPIO.setup(StepPinBackward1, GPIO.OUT)
GPIO.setup(StepPinForward2, GPIO.OUT)
GPIO.setup(StepPinBackward2, GPIO.OUT)
GPIO.setup(ECHO_F, GPIO.IN)
GPIO.setup(ECHO_L, GPIO.IN)
GPIO.setup(ECHO_R, GPIO.IN)
GPIO.setup(TRIG, GPIO.OUT)
GPIO.setup(LetfPWM, GPIO.OUT)
```

```
GPIO.setup(RightPWM, GPIO.OUT)
```

```
#PWM channels and frequency
```

```
PWML=GPIO.PWM(16, 0.5)
```

```
PWMR=GPIO.PWM(20, 0.5)
```

### 5.3.3 Ultrasonic sensor reading

As previously mentioned, the ultrasonic sensors are the proposed way of implementing collision detection. The desired function is for them to measure the distance from the robot to the closest present obstacle, and return the data for further usage in the code. In subsection 5.1, it was mentioned of the potential problems with noise and the proposed filtering method, using the average of every three reading. Below is a section of the code, with the function for data acquisition after filtering the readings.

```
def readsensor(PIN):
    for x in range(0, 2):
        read_time_start1 = time.time()
        GPIO.output(TRIG, True)
        time.sleep(pulse)
        GPIO.output(TRIG, False)

        while GPIO.input(PIN)==0:
            pulse_start = time.time()

        while GPIO.input(PIN)==1:
            pulse_end= time.time()

        pulse_duration[x] = pulse_end - pulse_start
        time.sleep(0.05-(time.time()-read_time_start1))

    distance = sum(pulse_duration)/measurment_count* SPEED_OF_SOUND
    distance = round(distance, 2)
    print distance
    if PIN == ECHOF:
        sensorF_data=distance

    if PIN == ECHOR:
        sensorR_data=distance

    if PIN==ECHOL:
```

```
sensorL_data=distance
```

The data reading is straightforward. The **TRIG** pin emits an ultrasonic impulse from a each of the sensors in addition to a digital timestamp that records the exact time of the event. When the impulse reaches back the sensor, a second timestamp is recorded. The logic behind it is that every loop should take equal amount of time for execution. The first timestamp in the beginning of the loop is initialized with  $read\_time\_start1 = time.time()$ . and calculated afterwards in  $time.sleep(0.05 - (time.time() - read\_time\_start1))$ . The expected cycle length is 0.05 second and is done to ensure that reading are acquired in the sensing range 0.02m to 4m. The set time of 0.05 second is used, as it would give the range, limited by time.

$$330(m/s) * 0.05s = 16.5 \quad (5.2)$$

And since the impulse is going to the object in range and returning to the sensor, a division by 2 with give the max range for the time limitation

$$16.5/2 = 8.25m \quad (5.3)$$

The result is almost double the sensing range supported by the hardware.

Furthermore, from the line: for x in range(0, 2): the reading of the sensor is done 3 times in a row, thus the implementation of the moving average filter.

The last lines of code ensure that when data reading is initiated, it would pass the value to only one of the readings, avoiding data collision.

```
if PIN == ECHO_F:
    sensorF_data=distance

if PIN == ECHO_R:
    sensorR_data=distance

if PIN==ECHO_L:
    sensorL_data=distance
```

### 5.3.4 Movements

The expected movements of a differential drive robot was covered in section ???. In this subsection, a description is conducted on the pre made functions governing the behaviour of the physical system. It is important to mention

At the completion of this chapter, five different functions concerned with the movements, have been used.

The most basic function is the **stop()**, which as it's name suggests, halts the motion of the device

```
def stop():
    GPIO.output(StepPinForward1, GPIO.LOW)
    GPIO.output(StepPinForward2, GPIO.LOW)
    GPIO.output(StepPinBackward1, GPIO.LOW)
    GPIO.output(StepPinBackward2, GPIO.LOW)
```

All the movement enabling pins will be set to low, which prompts the device to halt any ongoing movements.

The second function is the forward motion.

```
def forward(SPEED):
    print "FORWARD"
    GPIO.output(StepPinForward1, GPIO.HIGH)
    GPIO.output(StepPinForward2, GPIO.HIGH)
    PWML.start(SPEED)
    PWMR.start(SPEED)
```

From the code snippet, it can be seen that when the forward function is executed, both of the motors start moving in the same direction and the speed is determined by the variable SPEED. It is important to mention that small fluctuations in the speed of each wheel will cause curved motion instead of the desired forward linear motion, as discussed in section ???. Thus appropriate control needs to be applied in order to maintain equal speed in both wheels.

How the speed is determined will be further discussed in subsection 5.3.5: main loop.

The third function is the reverse, or what is essentially the same as the forward function only that the triggered pins are the backward ones. As in the forward movements, appropriate control is needed to maintain backward linear motion.

```
def reverse(SPEED):
    print "REVERSE"
    GPIO.output(StepPinBackward1, GPIO.HIGH)
    GPIO.output(StepPinBackward2, GPIO.HIGH)
    PWML.start(SPEED)
    PWMR.start(SPEED)
```

The two movements: forward and backward are the essential functions for the robot's linear motion.

Next we have the turning functions left() and right(). It was decided that the turning functions should be made that the vehicle takes the least amount of space to turn. As it was explained in section 3.1, in a differential drive if the velocities of each wheel are the same but in a different direction, the robot will rotate around its middle point of the wheel axis, which essentially becomes the turning radius.

Below are the two functions needed for turning.

```

def right(turningtime ,SPEED):
    print "RIGHT"
    GPIO.output(StepPinBackward1 , GPIO.HIGH)
    GPIO.output(StepPinForward2 , GPIO.HIGH)
    PWML.start(SPEED)
    PWMR.start(SPEED)
    sleep(turningtime)
    GPIO.output(StepPinBackward1 , GPIO.LOW)
    GPIO.output(StepPinForward2 , GPIO.LOW)

def left(turningtime ,SPEED):
    print "LEFT"
    GPIO.output(StepPinForward1 , GPIO.HIGH)
    GPIO.output(StepPinBackward2 , GPIO.HIGH)
    PWML.start(SPEED)
    PWMR.start(SPEED)
    sleep(turningtime)
    GPIO.output(StepPinForward1 , GPIO.LOW)
    GPIO.output(StepPinBackward2 , GPIO.LOW)

```

As we can see both turns, right and left, are made with predetermined factor **turningtime**. This will ensure that the vehicle will not over or under turn. And since a decision was made to mount 3 ultrasonic sensors, the turning ratio on 90 degrees was found adequate. Alteration of the turningtime will allow the device to turn to any desired amount of degrees, however a requirement of more than three ultrasonic sensors and specific positioning might be necessary.

### 5.3.5 Main loop

In this subsection the main loop, initiated when the device starts, is analysed.

```

while True:
    readsensor(ECHOF)

    if sensorF_data > 200:
        forward(MAXSPEED)

    if 200 > sensorF_data > 100:
        forward(MEDSPEED)

    if 100 > sensorF_data > 20:
        forward(MINSPEED)

```



```

if 20>sensorF_data:
    stop()
    readsensor(ECHOR)
    readsensor(ECHOL)

while sensorR_data<15 and sensorL_data<15:
    reverse(MINSPEED)
    readsensor(ECHOR)
    readsensor(ECHOL)

if sensorR_data>sensorL_data== True:
    right(1,MINSPEED)
    break

if sensorL_data>sensorR_data==True:
    left(1,MINSPEED)
    break

```

The loop is initiated with reading of the front sensor **readsensor(ECHOF)**. Any further action is determined, based on the reading taken from it. If the distance to the closest object is over 2m the vehicle will start moving in a forward direction with full speed. When the distance drops down between 200 cm and 100 cm, the vehicle speed will be reduced to medium speed in order to prepare for a potential turning. Minimal speed is initiated when the distance from the front sensor drops down between 100cm and 20cm, allowing the robot to perform an instant stop. When the distance to the closet object falls under 20 cm,the vehicle will stop and will decide how to proceed based on 3 possible cases. The ultrasonic sensors on the left and right will estimate the distance to the closest object on both sides.

- Case 1  
If both sides fall under 15 cm of the closest object, the robot will perform reverse motion and constantly recheck if space becomes available on either side. Once that is true, the robot will perform appropriate turn.
- Case 2  
If one or both of the side sensors register distance to the closest object more than 15 cm, it will turn in the direction of the larger available space.
- Case 3  
If the robot performs a turn with a minimal acceptable distance reading, it will re-evaluate the possible turning options.

After the turning phase the loop will go back to the beginning and start all over again.

The loop is made as simple as possible to avoid any complications while running the code.

## **5.4 N.B.**

Unfortunately, due to shipping delays, the completion of this part was done without the acquisition of the optical encoders, necessary to implement the crucial wheel speed control. Obstacle avoidance was performed, relying on the similarity of the motors, and was sufficient to successfully test the above mentioned code. After the arrival of the encoders (which is after the completion of this paper), a thorough update would be performed on this chapter and presented in order at the examination date.

## Chapter 6

# Conclusion and Future Development

Throughout this project, knowledge was obtained in the field of mobile robotics, by developing a differential drive robot. Modelling and simulation was performed, in order to understand the behaviour of the system, as well as to design a theoretical controller for cruise control. The platform used for the implementation is a Raspberry Pi and the software was developed in Python. Through a set of ultrasonic sensors the robot was capable of navigating, yet avoiding obstacles. Due to project limitations, at the completion of this report, not all desired functions were implemented in the system. Namely, with the absence of working optical encoders, it was not possible to test the theoretical controller. Nevertheless, after the acquiring of the necessary components, an update would be performed on the development part of this paper.



## **Appendix A**

# **Driver Chip**

## Driver Chip

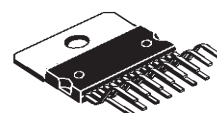

**L298**

### DUAL FULL-BRIDGE DRIVER

- OPERATING SUPPLY VOLTAGE UP TO 46 V
- TOTAL DC CURRENT UP TO 4 A
- LOW SATURATION VOLTAGE
- OVERTEMPERATURE PROTECTION
- LOGICAL "0" INPUT VOLTAGE UP TO 1.5 V (HIGH NOISE IMMUNITY)

#### DESCRIPTION

The L298 is an integrated monolithic circuit in a 15-lead Multiwatt and PowerSO20 packages. It is a high voltage, high current dual full-bridge driver designed to accept standard TTL logic levels and drive inductive loads such as relays, solenoids, DC and stepping motors. Two enable inputs are provided to enable or disable the device independently of the input signals. The emitters of the lower transistors of each bridge are connected together and the corresponding external terminal can be used for the con-

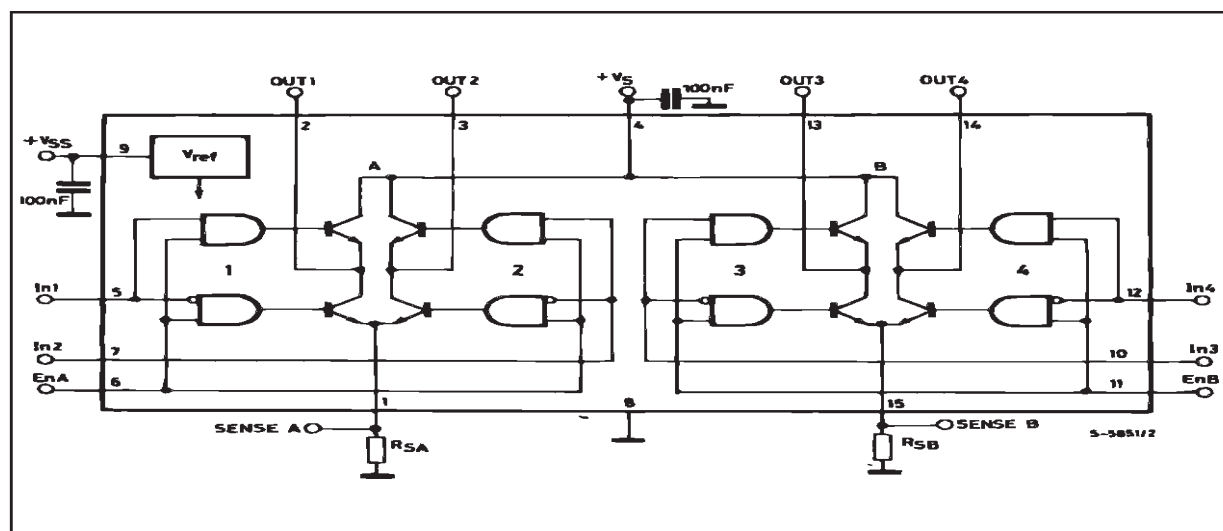

**Multiwatt15**

**PowerSO20**

**ORDERING NUMBERS :** L298N (Multiwatt Vert.)  
L298HN (Multiwatt Horiz.)  
L298P (PowerSO20)

nection of an external sensing resistor. An additional supply input is provided so that the logic works at a lower voltage.

#### BLOCK DIAGRAM



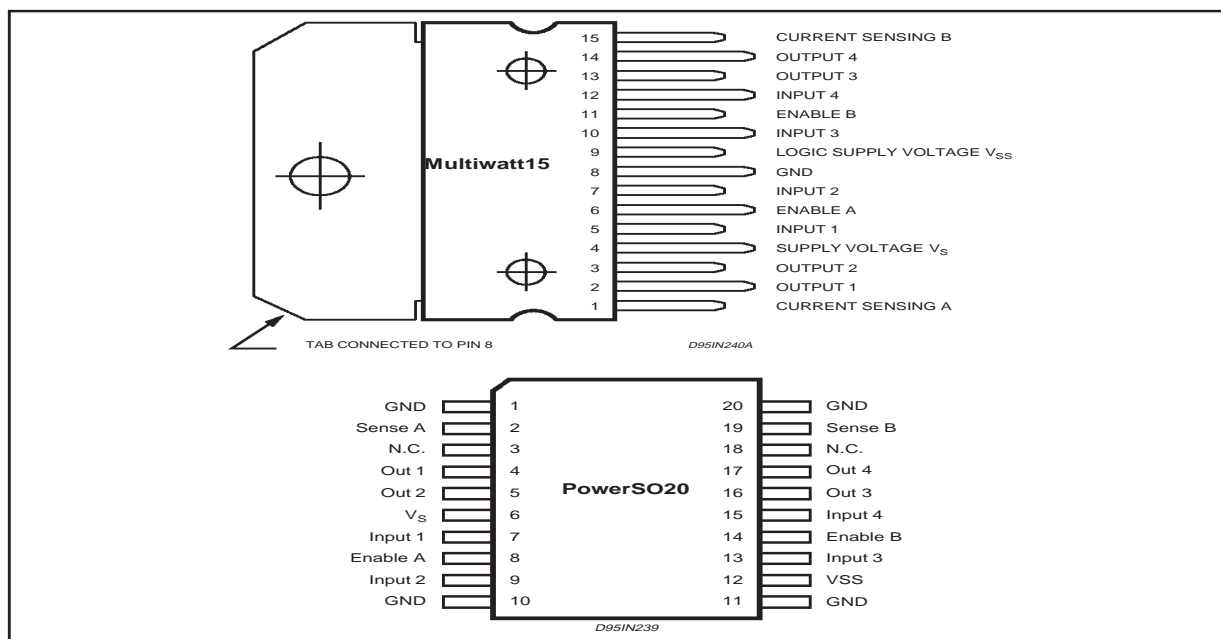
# Driver Chip

## L298

### ABSOLUTE MAXIMUM RATINGS

Symbol	Parameter	Value	Unit
$V_S$	Power Supply	50	V
$V_{SS}$	Logic Supply Voltage	7	V
$V_I, V_{en}$	Input and Enable Voltage	-0.3 to 7	V
$I_O$	Peak Output Current (each Channel) – Non Repetitive ( $t = 100\mu s$ ) – Repetitive (80% on –20% off; $t_{on} = 10ms$ ) – DC Operation	3 2.5 2	A A A
$V_{sens}$	Sensing Voltage	-1 to 2.3	V
$P_{tot}$	Total Power Dissipation ( $T_{case} = 75^\circ C$ )	25	W
$T_{op}$	Junction Operating Temperature	-25 to 130	$^\circ C$
$T_{stg}, T_J$	Storage and Junction Temperature	-40 to 150	$^\circ C$

### PIN CONNECTIONS (top view)



### THERMAL DATA

Symbol	Parameter	PowerSO20	Multiwatt15	Unit
$R_{th j-case}$	Thermal Resistance Junction-case	Max. –	3	$^\circ C/W$
$R_{th j-amb}$	Thermal Resistance Junction-ambient	Max. 13 (*)	35	$^\circ C/W$

(\*) Mounted on aluminum substrate

## Driver Chip

L298

### PIN FUNCTIONS (refer to the block diagram)

MW.15	PowerSO	Name	Function
1;15	2;19	Sense A; Sense B	Between this pin and ground is connected the sense resistor to control the current of the load.
2;3	4;5	Out 1; Out 2	Outputs of the Bridge A; the current that flows through the load connected between these two pins is monitored at pin 1.
4	6	V <sub>S</sub>	Supply Voltage for the Power Output Stages. A non-inductive 100nF capacitor must be connected between this pin and ground.
5;7	7;9	Input 1; Input 2	TTL Compatible Inputs of the Bridge A.
6;11	8;14	Enable A; Enable B	TTL Compatible Enable Input: the L state disables the bridge A (enable A) and/or the bridge B (enable B).
8	1,10,11,20	GND	Ground.
9	12	V <sub>SS</sub>	Supply Voltage for the Logic Blocks. A 100nF capacitor must be connected between this pin and ground.
10; 12	13;15	Input 3; Input 4	TTL Compatible Inputs of the Bridge B.
13; 14	16;17	Out 3; Out 4	Outputs of the Bridge B. The current that flows through the load connected between these two pins is monitored at pin 15.
—	3;18	N.C.	Not Connected

### ELECTRICAL CHARACTERISTICS (V<sub>S</sub> = 42V; V<sub>SS</sub> = 5V, T<sub>j</sub> = 25°C; unless otherwise specified)

Symbol	Parameter	Test Conditions	Min.	Typ.	Max.	Unit
V <sub>S</sub>	Supply Voltage (pin 4)	Operative Condition	V <sub>IH</sub> +2.5		46	V
V <sub>SS</sub>	Logic Supply Voltage (pin 9)		4.5	5	7	V
I <sub>S</sub>	Quiescent Supply Current (pin 4)	V <sub>en</sub> = H; I <sub>L</sub> = 0 V <sub>i</sub> = L V <sub>i</sub> = H		13 50	22 70	mA mA
I <sub>SS</sub>	Quiescent Current from V <sub>SS</sub> (pin 9)	V <sub>en</sub> = L V <sub>en</sub> = H; I <sub>L</sub> = 0 V <sub>i</sub> = L V <sub>i</sub> = H V <sub>en</sub> = L V <sub>i</sub> = X		24 7	4 36 12	mA mA mA
V <sub>iL</sub>	Input Low Voltage (pins 5, 7, 10, 12)		−0.3		1.5	V
V <sub>iH</sub>	Input High Voltage (pins 5, 7, 10, 12)		2.3		V <sub>SS</sub>	V
I <sub>iL</sub>	Low Voltage Input Current (pins 5, 7, 10, 12)	V <sub>i</sub> = L			−10	μA
I <sub>iH</sub>	High Voltage Input Current (pins 5, 7, 10, 12)	V <sub>i</sub> = H ≤ V <sub>SS</sub> −0.6V		30	100	μA
V <sub>en</sub> = L	Enable Low Voltage (pins 6, 11)		−0.3		1.5	V
V <sub>en</sub> = H	Enable High Voltage (pins 6, 11)		2.3		V <sub>SS</sub>	V
I <sub>en</sub> = L	Low Voltage Enable Current (pins 6, 11)	V <sub>en</sub> = L			−10	μA
I <sub>en</sub> = H	High Voltage Enable Current (pins 6, 11)	V <sub>en</sub> = H ≤ V <sub>SS</sub> −0.6V		30	100	μA
V <sub>CEsat</sub> (H)	Source Saturation Voltage	I <sub>L</sub> = 1A I <sub>L</sub> = 2A	0.95	1.35 2	1.7 2.7	V V
V <sub>CEsat</sub> (L)	Sink Saturation Voltage	I <sub>L</sub> = 1A (5) I <sub>L</sub> = 2A (5)	0.85	1.2 1.7	1.6 2.3	V V
V <sub>CEsat</sub>	Total Drop	I <sub>L</sub> = 1A (5) I <sub>L</sub> = 2A (5)	1.80		3.2 4.9	V V
V <sub>sens</sub>	Sensing Voltage (pins 1, 15)		−1 (1)		2	V



# Driver Chip

## L298

### ELECTRICAL CHARACTERISTICS (continued)

Symbol	Parameter	Test Conditions	Min.	Typ.	Max.	Unit
$T_1 (V_i)$	Source Current Turn-off Delay	$0.5 V_i$ to $0.9 I_L$ (2); (4)		1.5		$\mu s$
$T_2 (V_i)$	Source Current Fall Time	$0.9 I_L$ to $0.1 I_L$ (2); (4)		0.2		$\mu s$
$T_3 (V_i)$	Source Current Turn-on Delay	$0.5 V_i$ to $0.1 I_L$ (2); (4)		2		$\mu s$
$T_4 (V_i)$	Source Current Rise Time	$0.1 I_L$ to $0.9 I_L$ (2); (4)		0.7		$\mu s$
$T_5 (V_i)$	Sink Current Turn-off Delay	$0.5 V_i$ to $0.9 I_L$ (3); (4)		0.7		$\mu s$
$T_6 (V_i)$	Sink Current Fall Time	$0.9 I_L$ to $0.1 I_L$ (3); (4)		0.25		$\mu s$
$T_7 (V_i)$	Sink Current Turn-on Delay	$0.5 V_i$ to $0.9 I_L$ (3); (4)		1.6		$\mu s$
$T_8 (V_i)$	Sink Current Rise Time	$0.1 I_L$ to $0.9 I_L$ (3); (4)		0.2		$\mu s$
$f_c (V_i)$	Commutation Frequency	$I_L = 2A$		25	40	KHz
$T_1 (V_{en})$	Source Current Turn-off Delay	$0.5 V_{en}$ to $0.9 I_L$ (2); (4)		3		$\mu s$
$T_2 (V_{en})$	Source Current Fall Time	$0.9 I_L$ to $0.1 I_L$ (2); (4)		1		$\mu s$
$T_3 (V_{en})$	Source Current Turn-on Delay	$0.5 V_{en}$ to $0.1 I_L$ (2); (4)		0.3		$\mu s$
$T_4 (V_{en})$	Source Current Rise Time	$0.1 I_L$ to $0.9 I_L$ (2); (4)		0.4		$\mu s$
$T_5 (V_{en})$	Sink Current Turn-off Delay	$0.5 V_{en}$ to $0.9 I_L$ (3); (4)		2.2		$\mu s$
$T_6 (V_{en})$	Sink Current Fall Time	$0.9 I_L$ to $0.1 I_L$ (3); (4)		0.35		$\mu s$
$T_7 (V_{en})$	Sink Current Turn-on Delay	$0.5 V_{en}$ to $0.9 I_L$ (3); (4)		0.25		$\mu s$
$T_8 (V_{en})$	Sink Current Rise Time	$0.1 I_L$ to $0.9 I_L$ (3); (4)		0.1		$\mu s$

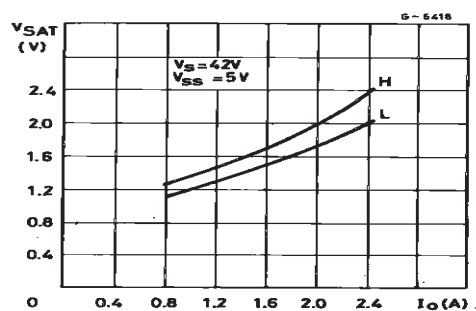
1) Sensing voltage can be  $-1 V$  for  $t \leq 50 \mu s$ ; in steady state  $V_{sens} \min \geq -0.5 V$ .

2) See fig. 2.

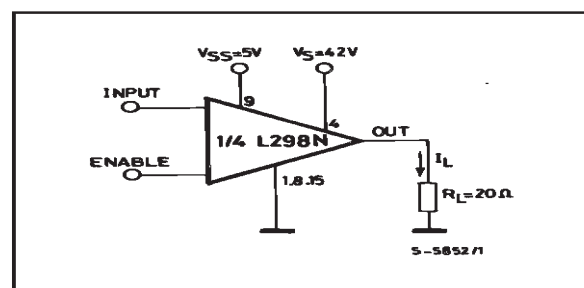
3) See fig. 4.

4) The load must be a pure resistor.

**Figure 1** : Typical Saturation Voltage vs. Output Current.



**Figure 2** : Switching Times Test Circuits.



Note : For INPUT Switching, set EN = H  
For ENABLE Switching, set IN = H

Driver Chip

L298

Figure 3 : Source Current Delay Times vs. Input or Enable Switching.

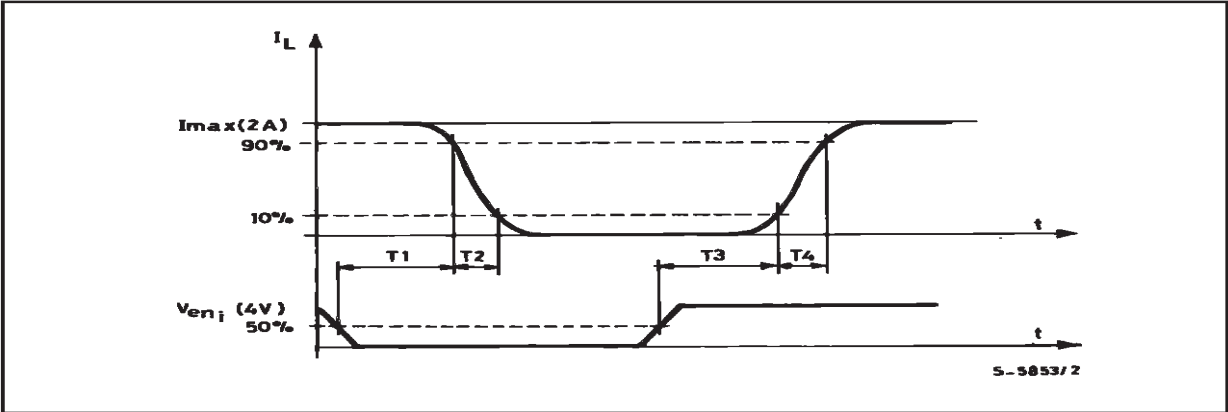
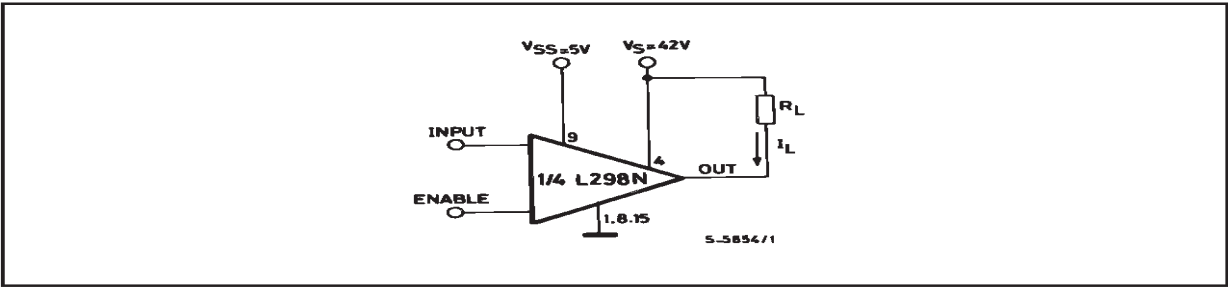


Figure 4 : Switching Times Test Circuits.



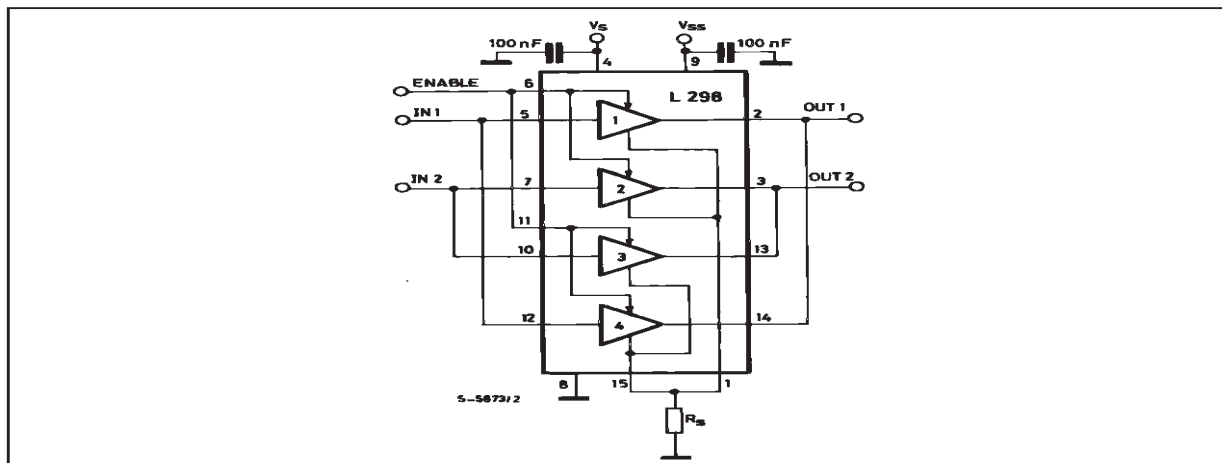
Note : For INPUT Switching, set EN = H  
For ENABLE Switching, set IN = L



## Driver Chip

L298

**Figure 7 :** For higher currents, outputs can be paralleled. Take care to parallel channel 1 with channel 4 and channel 2 with channel 3.



### APPLICATION INFORMATION (Refer to the block diagram)

#### 1.1. POWER OUTPUT STAGE

The L298 integrates two power output stages (A; B). The power output stage is a bridge configuration and its outputs can drive an inductive load in common or differential mode, depending on the state of the inputs. The current that flows through the load comes out from the bridge at the sense output: an external resistor ( $R_{SA}$ ;  $R_{SB}$ ) allows to detect the intensity of this current.

#### 1.2. INPUT STAGE

Each bridge is driven by means of four gates the input of which are  $IN_1$ ;  $IN_2$ ;  $EN_A$  and  $IN_3$ ;  $IN_4$ ;  $EN_B$ . The  $IN$  inputs set the bridge state when The  $EN$  input is high; a low state of the  $EN$  input inhibits the bridge. All the inputs are TTL compatible.

### 2. SUGGESTIONS

A non inductive capacitor, usually of 100 nF, must be foreseen between both  $V_S$  and  $V_{SS}$  to ground, as near as possible to GND pin. When the large capacitor of the power supply is too far from the IC, a second smaller one must be foreseen near the L298.

The sense resistor, not of a wire wound type, must be grounded near the negative pole of  $V_S$  that must be near the GND pin of the I.C.

Each input must be connected to the source of the driving signals by means of a very short path.

Turn-On and Turn-Off : Before to Turn-ON the Supply Voltage and before to Turn it OFF, the Enable input must be driven to the Low state.

### 3. APPLICATIONS

Fig 6 shows a bidirectional DC motor control Schematic Diagram for which only one bridge is needed. The external bridge of diodes D1 to D4 is made by four fast recovery elements ( $t_{rr} \leq 200$  nsec) that must be chosen of a  $V_F$  as low as possible at the worst case of the load current.

The sense output voltage can be used to control the current amplitude by chopping the inputs, or to provide overcurrent protection by switching low the enable input.

The brake function (Fast motor stop) requires that the Absolute Maximum Rating of 2 Amps must never be overcome.

When the repetitive peak current needed from the load is higher than 2 Amps, a paralleled configuration can be chosen (See Fig.7).

An external bridge of diodes are required when inductive loads are driven and when the inputs of the IC are chopped; Schottky diodes would be preferred.

## Driver Chip

### L298

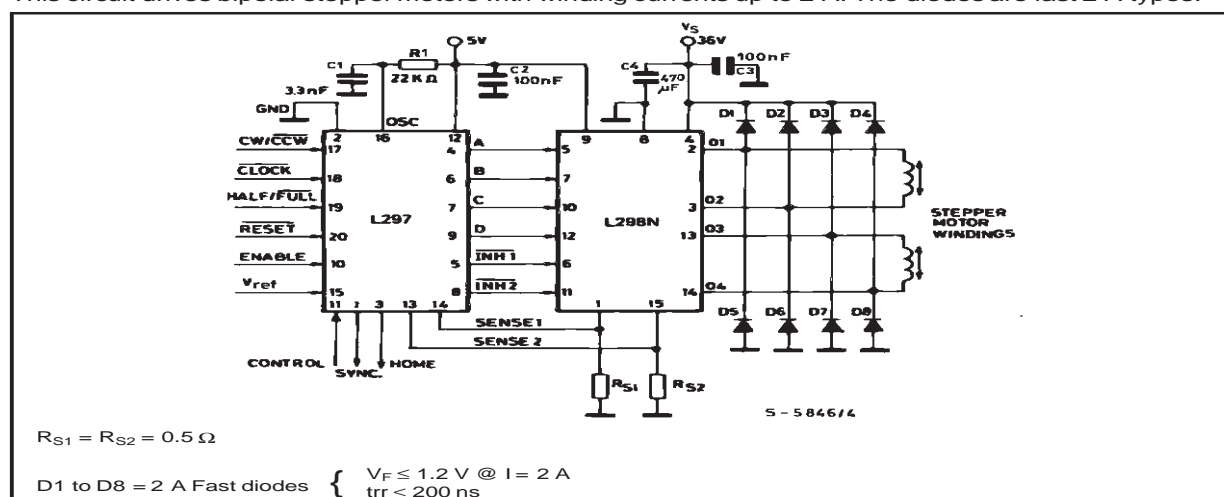
This solution can drive until 3 Amps In DC operation and until 3.5 Amps of a repetitive peak current.

On Fig 8 it is shown the driving of a two phase bipolar stepper motor ; the needed signals to drive the inputs of the L298 are generated, in this example, from the IC L297.

Fig 9 shows an example of P.C.B. designed for the application of Fig 8.

**Figure 8** : Two Phase Bipolar Stepper Motor Circuit.

This circuit drives bipolar stepper motors with winding currents up to 2 A. The diodes are fast 2 A types.



Driver Chip

L298

Figure 9 : Suggested Printed Circuit Board Layout for the Circuit of fig. 8 (1:1 scale).

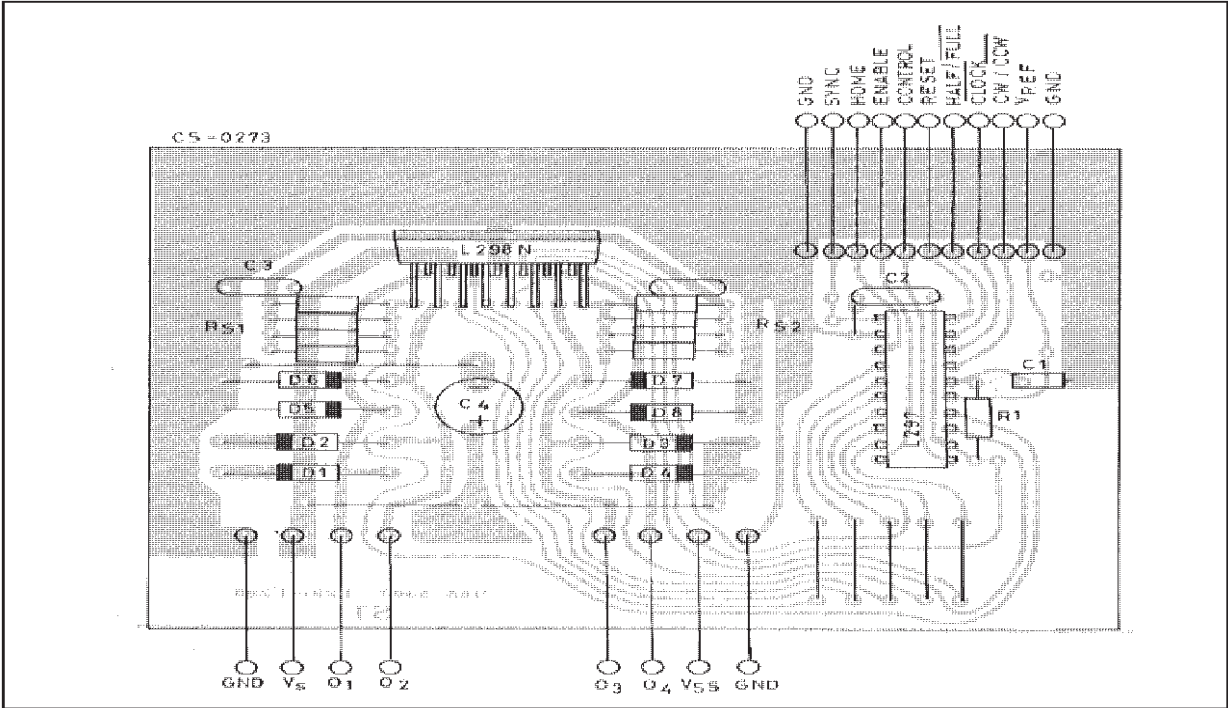
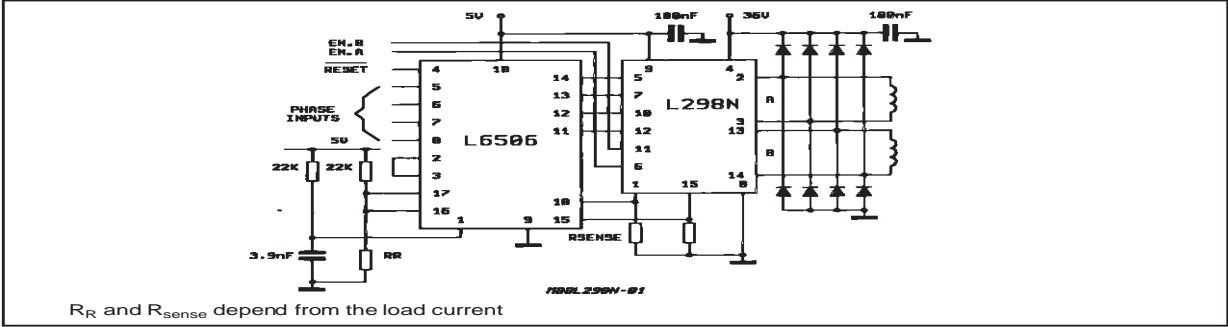


Figure 10 : Two Phase Bipolar Stepper Motor Control Circuit by Using the Current Controller L6506.

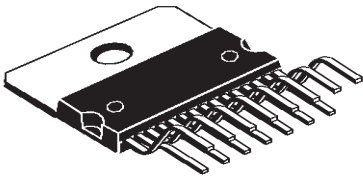


Driver Chip

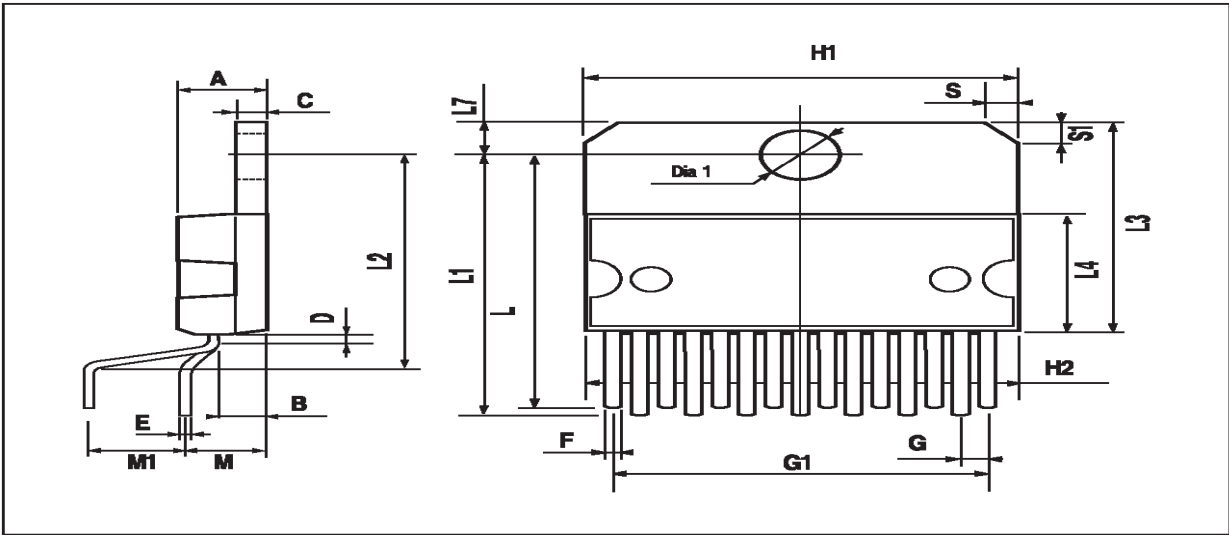
L298

DIM.	mm			inch		
	MIN.	TYP.	MAX.	MIN.	TYP.	MAX.
A			5			0.197
B			2.65			0.104
C			1.6			0.063
D		1			0.039	
E	0.49		0.55	0.019		0.022
F	0.66		0.75	0.026		0.030
G	1.02	1.27	1.52	0.040	0.050	0.060
G1	17.53	17.78	18.03	0.690	0.700	0.710
H1	19.6			0.772		
H2			20.2			0.795
L	21.9	22.2	22.5	0.862	0.874	0.886
L1	21.7	22.1	22.5	0.854	0.870	0.886
L2	17.65		18.1	0.695		0.713
L3	17.25	17.5	17.75	0.679	0.689	0.699
L4	10.3	10.7	10.9	0.406	0.421	0.429
L7	2.65		2.9	0.104		0.114
M	4.25	4.55	4.85	0.167	0.179	0.191
M1	4.63	5.08	5.53	0.182	0.200	0.218
S	1.9		2.6	0.075		0.102
S1	1.9		2.6	0.075		0.102
Dia1	3.65		3.85	0.144		0.152

OUTLINE AND MECHANICAL DATA



Multiwatt15 V

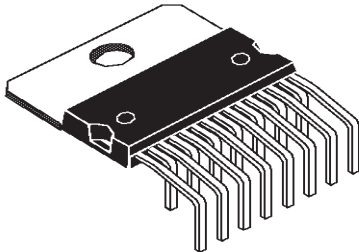


Driver Chip

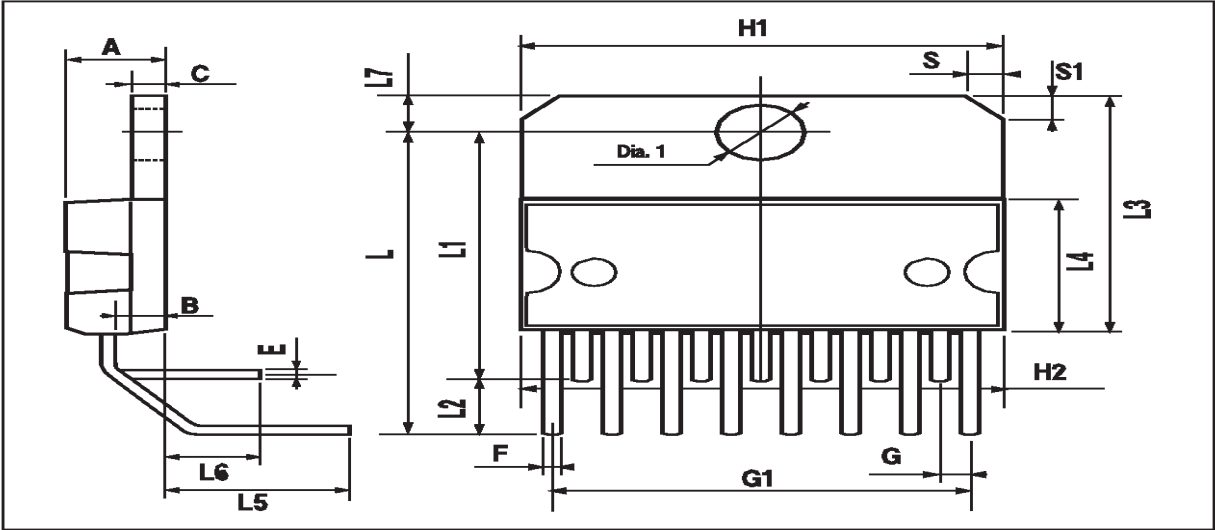
L298

DIM.	mm			inch		
	MIN.	TYP.	MAX.	MIN.	TYP.	MAX.
A			5			0.197
B			2.65			0.104
C			1.6			0.063
E	0.49		0.55	0.019		0.022
F	0.66		0.75	0.026		0.030
G	1.14	1.27	1.4	0.045	0.050	0.055
G1	17.57	17.78	17.91	0.692	0.700	0.705
H1	19.6			0.772		
H2			20.2			0.795
L		20.57			0.810	
L1		18.03			0.710	
L2		2.54			0.100	
L3	17.25	17.5	17.75	0.679	0.689	0.699
L4	10.3	10.7	10.9	0.406	0.421	0.429
L5		5.28			0.208	
L6		2.38			0.094	
L7	2.65		2.9	0.104		0.114
S	1.9		2.6	0.075		0.102
S1	1.9		2.6	0.075		0.102
Dia1	3.65		3.85	0.144		0.152

OUTLINE AND MECHANICAL DATA



Multiwatt15 H





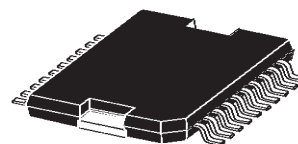
# Driver Chip

## L298

DIM.	mm			inch		
	MIN.	TYP.	MAX.	MIN.	TYP.	MAX.
A			3.6			0.142
a1	0.1		0.3	0.004		0.012
a2			3.3			0.130
a3	0		0.1	0.000		0.004
b	0.4		0.53	0.016		0.021
c	0.23		0.32	0.009		0.013
D (1)	15.8		16	0.622		0.630
D1	9.4		9.8	0.370		0.386
E	13.9		14.5	0.547		0.570
e		1.27			0.050	
e3		11.43			0.450	
E1 (1)	10.9		11.1	0.429		0.437
E2			2.9			0.114
E3	5.8		6.2	0.228		0.244
G	0		0.1	0.000		0.004
H	15.5		15.9	0.610		0.626
h			1.1			0.043
L	0.8		1.1	0.031		0.043
N	10° (max.)					
S	8° (max.)					
T		10			0.394	

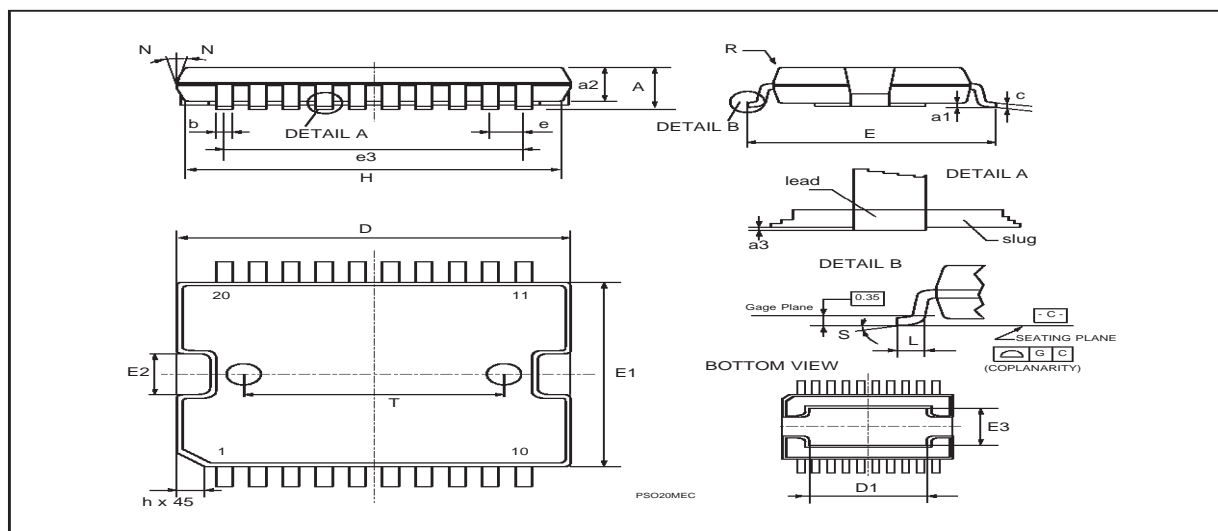
(1) "D and F" do not include mold flash or protrusions.  
 - Mold flash or protrusions shall not exceed 0.15 mm (0.006").  
 - Critical dimensions: "E", "G" and "a3"

## OUTLINE AND MECHANICAL DATA



JEDEC MO-166

**PowerSO20**



## Driver Chip

---

**L298**

Information furnished is believed to be accurate and reliable. However, STMicroelectronics assumes no responsibility for the consequences of use of such information nor for any infringement of patents or other rights of third parties which may result from its use. No license is granted by implication or otherwise under any patent or patent rights of STMicroelectronics. Specification mentioned in this publication are subject to change without notice. This publication supersedes and replaces all information previously supplied. STMicroelectronics products are not authorized for use as critical components in life support devices or systems without express written approval of STMicroelectronics.

The ST logo is a registered trademark of STMicroelectronics  
© 2000 STMicroelectronics – Printed in Italy – All Rights Reserved  
STMicroelectronics GROUP OF COMPANIES  
Australia - Brazil - China - Finland - France - Germany - Hong Kong - India - Italy - Japan - Malaysia - Malta - Morocco -  
Singapore - Spain - Sweden - Switzerland - United Kingdom - U.S.A.  
<http://www.st.com>

