Obstacle Avoidance for a Differential Drive Robot

Project Report

ED5-8

Aalborg University
Electronics and IT

This paper was written using LaTeX.
The modelling was perform 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

Copies: 0

Page Numbers: 74

Date of Completion:
June 10, 2017

Abstract:

The rising development in automation has lead 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

Contents

Preface

Aalborg University, June 10, 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 .[8]

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.[8]

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:[5]

- **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.[5]

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.[3] Most systems, present in autonomous cars, are a scaled-up version of what has been present in robotics for a considerate 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

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.

2.1 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.

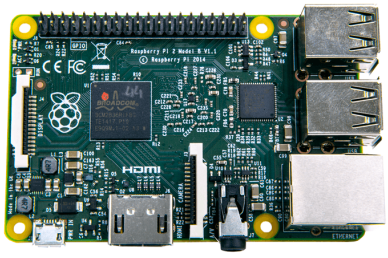


Figure 2.1: Raspberry Pi 2B

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 2.2: 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

2.2 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 to 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.

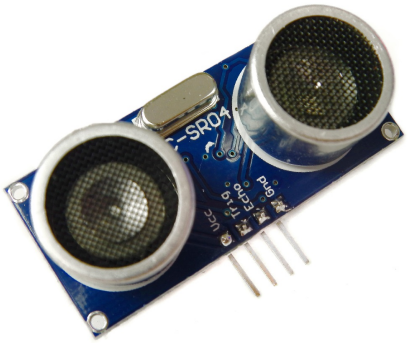


Figure 2.3: Ultra sonic sensor HC-SR04

2.3 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.[4]

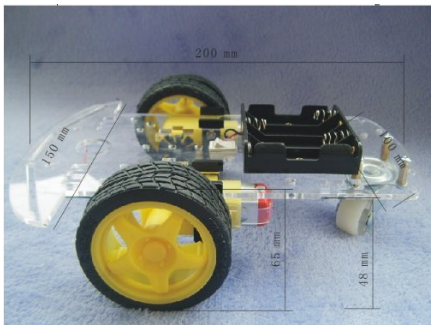


Figure 2.4: Chassie and the two DC motors with a power supply

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

2.4 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)

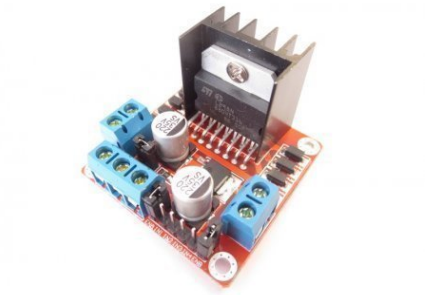


Figure 2.5: The L298N driver

2.5 Speed sensor

The speed of the wheels is measured by the LM393 IR speed sensors attached to the tachometer wheel of the motor.[9]

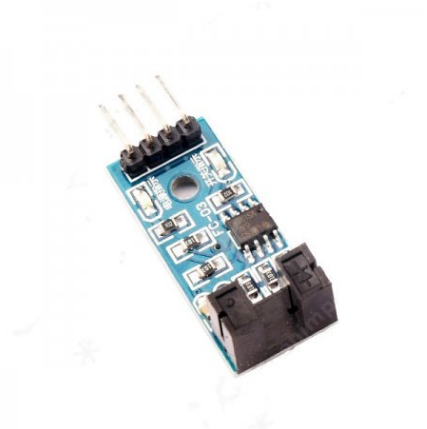


Figure 2.6: The LM393 IR speed sensor

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.

2.6 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 2.7: Edimax EW-7811Un Wi-Fi adapter

2.7 Assembly

In this section, the schematics of the assembled components has been analysed and given in figure 2.8.

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.

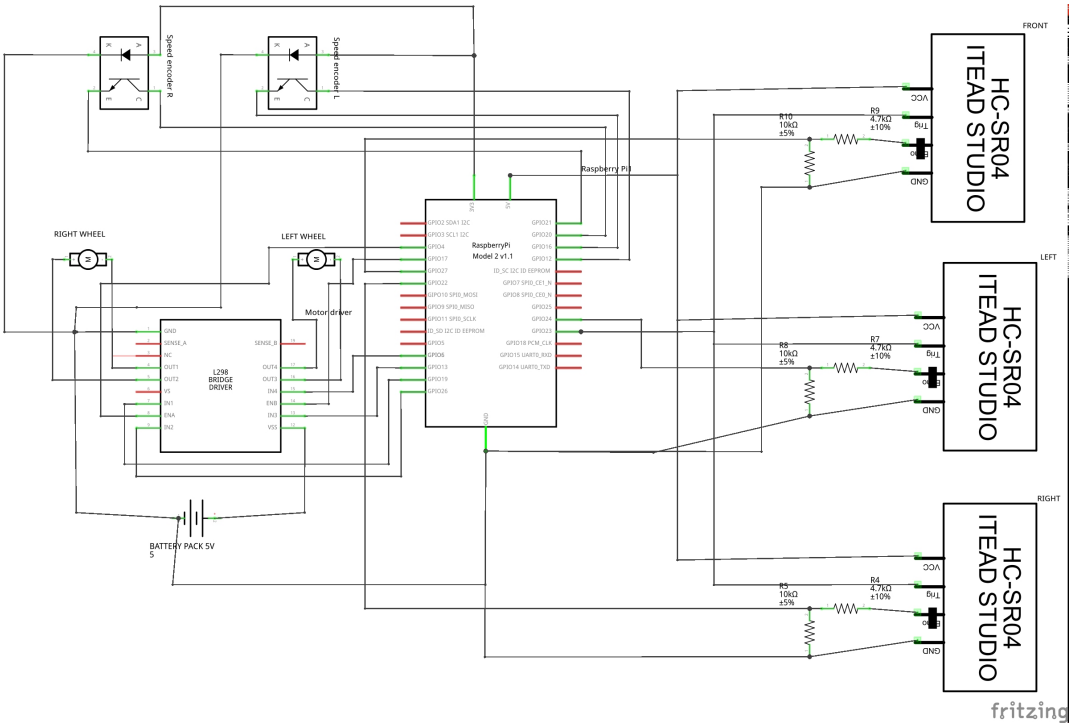


Figure 2.8: Schematics of the device

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 3

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.

3.1 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.

$$v = \frac{Rw_r + Rw_l}{2} \tag{3.1}$$

$$W = \frac{Rw_r - Rw_l}{l} \tag{3.2}$$

Equation 3.11 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 3.12 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:

$$\begin{bmatrix} v \\ W \end{bmatrix} = \begin{bmatrix} \frac{R}{2} & \frac{R}{2} \\ \frac{R}{l} & \frac{R}{l} \end{bmatrix} \begin{bmatrix} w_r \\ w_l \end{bmatrix}$$

3.1.1 Non-holonomic constraints

A robot, using the differential drive steering design, has restricted local movements, while having no restrictions in global motion

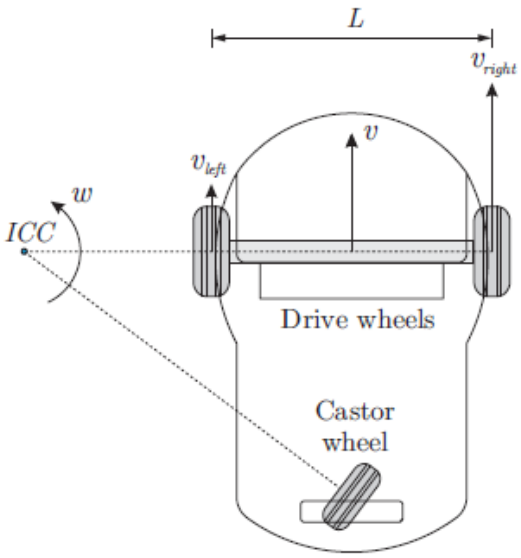


Figure 3.1: Differential drive overview[1]

The linear speed, previously derived from chapter 3.2.2, is represented as v_r and v_l, for each of the wheels. The speed of the robot is taken as the average speed of each wheel.

$$v = \frac{v_r + v_l}{2} \tag{3.3}$$

The angular speed of the robot (or turning speed) is based on the linear speeds of each wheel and the distance between the wheels. It is denoted as **W** in equation 3.4.(not to be confused with ω, the rotational speed of the motor)

$$W = \frac{v_r - v_l}{l} \tag{3.4}$$

The relation between the linear speed and the angular speed of the robot is similar to equation 3.22.

$$v = WD \tag{3.5}$$

Where D is the turning radius, from the midpoint of the wheels to the ICC. Solving for the turning radius, yields:

$$D = \frac{v}{W} \tag{3.6}$$
$$= \frac{l}{2}\frac{v_r + v_l}{v_r - v_l}$$

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 3.3 is simply equal to the individual speed of each of wheel. On the other hand, the angular speed from equation 3.4 becomes 0, and the turning radius infinite. The robot is expected to perform straight linear motion.

- $v_r = \mathbf{0}$ or $v_l = \mathbf{0}$
  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.

It is important to mention that the wheels , present in the system , carry some **nonholonomic** constraints. That is, the robot's local movements are restricted, while no restrictions are present in the global navigation. We can further extend the idea with the use of generalised coordinates.

$$q = (x, y, \theta) \tag{3.7}$$

Equation 3.7 could be seen as a point on a two dimensional Cartesian coordinate system, where x and y are the axis and θ is the angle between the x axis and the point.

Figure 3.2 shows the robot representation based on the generalised coordinates.

We can picture the constrains for the wheels by setting the sideways velocity to zero. That is, the robot can not perform sideways movements like slipping or sliding, but is not limited from manoeuvring in that position, whatsoever.

$$\dot{x}sin(\theta) - \dot{y}cos(\theta) = 0 \tag{3.8}$$

Equation 3.8 is a common nonholonomic constraint in mobile robotics.[6]

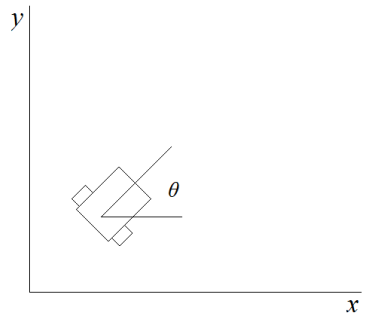


Figure 3.2: Robot representation in Cartesian coordinates

Particularly if the robot is viewed as a point, the Kinematics equations in Cartesian space can be derived as:

$$\begin{aligned} \dot{x} &= v cos(\theta) \\ \dot{y} &= v sin(\theta) \\ \dot{\theta} &= \omega \end{aligned} \tag{3.9}$$

In the case of a differential drive robot, substitution of the linear and angular velocities, **v** and ω, with the previously derived in equation 3.3 and 3.4 average robot speed and angular robot speed (with respect to the center of rotation between the wheels), will results in the kinematics equations for locomotion of a differential drive.

$$\begin{aligned} \dot{x} &= \frac{v_r + v_l}{2} cos(\theta) \\ \dot{y} &= \frac{v_r + v_l}{2} sin(\theta) \\ \dot{\theta} &= \frac{v_r - v_l}{l} \end{aligned} \tag{3.10}$$

In equation 3.10 we will have a change in the robot's position (x,y,θ) when the velocity of each wheel is controlled.

3.2 Dynamics

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

Parameter	Description	Unit
K	Motor constant	V/(rad/s) Nm/amp
R	Armature resistance	Ω
L	Armature inductance	H
b_r	Rotor damping	Nms
J_w	Load inertia	KgM^2

Table 3.1: Motor parameters

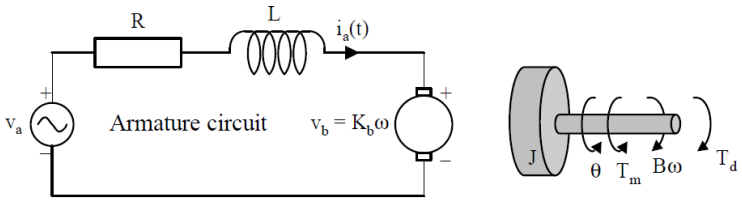


Figure 3.3: Electrical circuit and free-body diagram of DC motor [2]

3.2.1 Mechanical part

A DC motor produces mechanical torque applied to the shaft($\boldsymbol{T_e}$) linearly proportional to the armature current, the magnetic field and a torque constant($\boldsymbol{K_t}$). Assuming constant magnetic field, the produced torque is proportional to the torque constant($\boldsymbol{K_t}$) and the armature current(**I**).(Equation 3.11)Furthermore, the back Emf (electromotive force)(E_b) is equal to the angular velocity(ω) of the shaft multiplied with the Back emf constant(K_b).(Equation 3.12)

$$T_e = IK_t \tag{3.11}$$

$$E_b = \omega K_b \tag{3.12}$$

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.

$$K_t = K_b = K \tag{3.13}$$

Furthermore, from figure 3.3, using Kirchhoff's voltage law, we can derive the equations governing the electrical part of the DC motor, where the applied voltage (**V**) is proportional to the voltage drop through the armature resistance(**R**) and inductance(**L**), and the back electromotive voltage($\boldsymbol{E_b}$). 3.14

$$V = RI + L\frac{dI}{dt} + E_b \tag{3.14}$$

The mechanical part of the DC motor(mechanical part of figure 3.3) is derived from the equations, where the mechanical torque(T_m) is the difference between the electromagnetic torque(T_e) and the rotational losses (T_b). 3.15

$$T_m = T_e - T_b \tag{3.15}$$

Using Newton's second law for rotational motion and substituting from equation3.11, we can rewrite equation 3.15 as:

$$J\dot{\omega} = KI - b\omega \tag{3.16}$$

Where **J** is the load's inertia and **b** is the viscous friction in the motor's bearings. Further substitution in equation 3.14 with the derived back emf from 3.12 results in:

$$V = RI + L\frac{dI}{dt} + K\omega \tag{3.17}$$

Equations 3.16 and 3.17 are the combined equations of motion for the DC motor.

Applying the Laplace transform to the equations, we can derive the transfer function of the DC motor.

$$\begin{aligned} sJ\Omega(s) + b\Omega(s) &= KI(s) \\ sLI(s) + RI(s) &= V(s) - K\Omega(s) \end{aligned} \tag{3.18}$$

$$\Downarrow$$

$$\begin{aligned} \frac{\Omega(s)(sJ+b)}{K} &= I(s) \\ I(s)(sL+R) + K\Omega(s) &= V(s) \end{aligned} \tag{3.19}$$

Substituting with **I(s)** in the second part of equation 3.19, and setting the angular velocity($\boldsymbol{\Omega(s)}$) as output and the voltage (**V(s)**) as input results in the transfer function for the DC motor.(3.20)

$$\frac{\Omega(s)}{V(s)} = \frac{K}{(Js+b)(sL+R) + K^2} \tag{3.20}$$

3.2.2 Simulink Model

In this subsection, the previously derived equations are represented in a block diagram using Matlab's Simulink environment. There are several possible ways to

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

Table 3.2: Motor parameters

arrange the blocks governing the DC motor, thus in this paper a familiar approach is considered.

As evident from equation 3.20, the voltage is the input of the system, while the angular velocity is the output. In order to accurately apply the equations, while attaining the desired result, a modification of equations 3.16 and 3.17 was made.(3.21)

$$\begin{aligned} \frac{dI}{dt} &= \frac{1}{L}(V - RI - K\omega) \\ \frac{d\omega}{dt} &= \frac{1}{J}(KI - b\omega) \end{aligned} \tag{3.21}$$

The block diagram representation in figure 3.4 has the integrals of the rotational acceleration and the rate of change of the armature current considered as outputs based on equations 3.21.

The inclusion of the gear ratio (**n**) and the radius of the wheel (**r**) products to the angular velocity in the end of the block diagram, results in model scaling for the linear velocity (**v**) of the wheel. (3.22)

$$v = r\omega \tag{3.22}$$

Performing integration on the derived linear velocity results in obtaining the linear displacement of the wheels, later to be used with the kinematics model.

To summarise, the goal was to relate the voltage to the speed. The input of the block diagram is the voltage of the motor (**V**) while the outputs are the linear speed caused by wheel rotation and the linear displacement, obtained from integrating the speed. The blocks comprising the upper and lower part of the block diagram, directly correspond to equation 3.21 (upper part correspond to the electrical part of the motor; lower part correspond to the mechanical part of the motor).

Furthermore, as this paper is concerned with the development of a differential drive robot, the block diagram in figure 3.4 is solely a subsystem of the complete

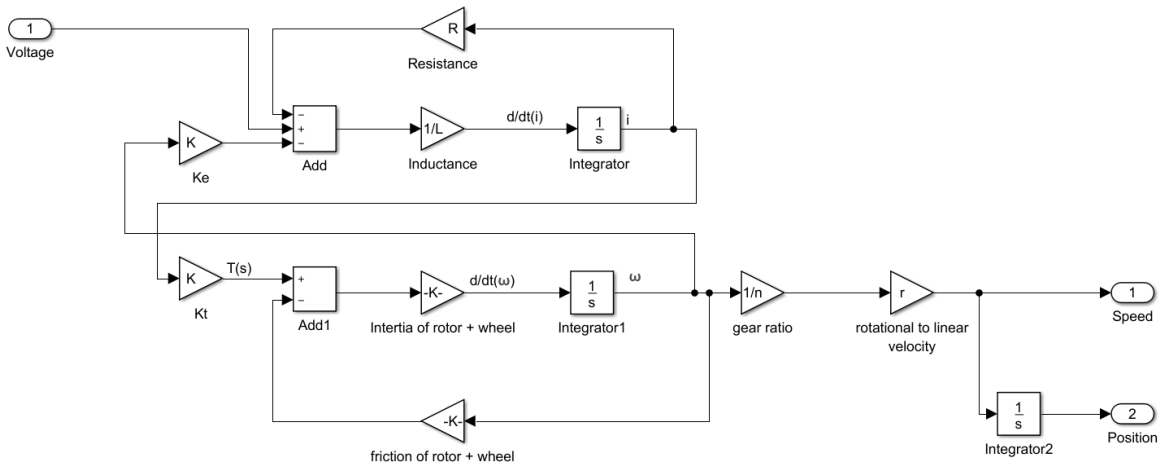


Figure 3.4: DC Motor Block Diagram

kinematics model. That is, two DC motor subsystems are required in order to describe the complete motor/wheel dynamics.

3.2.3 Kinematics in simulink

Using equations 3.3 and 3.4 a simulink block diagram is constructed in figure 3.5.

The inputs are the individual wheel velocities, arranged to reflect the previously mentioned equations. The middle output is the turning radius **D**, which is governed by equation 3.6. The derived average linear speed is to be further used in equation 3.10 to estimate the position of the robot based on it's angle through time, where the angle (θ) is obtain from integrating $\dot{\theta}$, which itself is the angular speed of the robot.

In figure 3.6 the inputs are the the average speed and the orientation of the robot, while the outputs X and Y from equation 3.10 are fed in a graph to observe the robot's trajectory through time.

The subsystem from figure 3.7 is composed of blocks arranged to reflect equation 3.10. The outputs $\dot{x}$ and $\dot{y}$ are further integrated to graph the trajectory of the robot.

3.3 Complete model

In this section the complete system model is derived. It includes motor dynamics and robot kinematics. That is, the behaviour of the robot is analysed and compared with the expected performance.

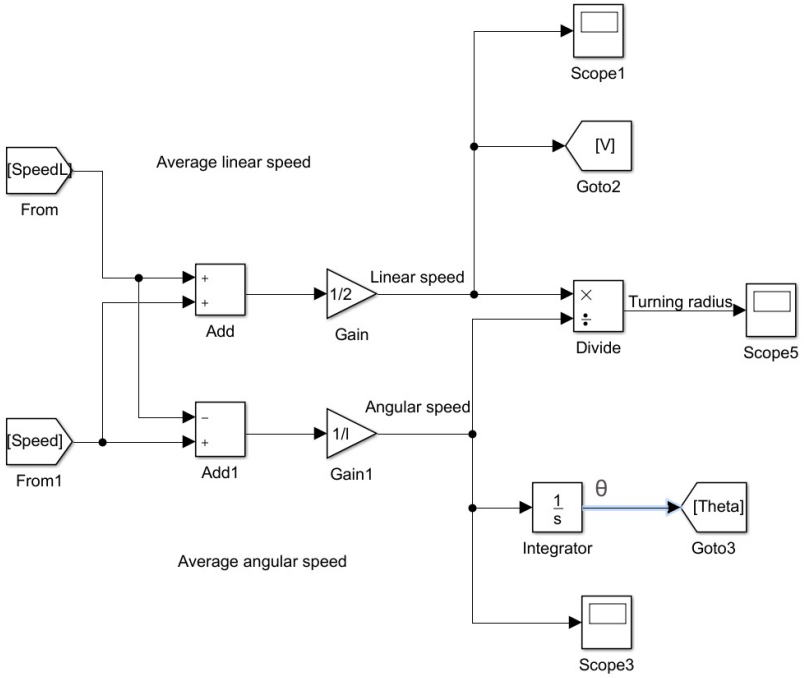


Figure 3.5: Differential drive kinematics

In figure 3.8, the complete system model for a differential drive robot could be observed. As previously mentioned, a differential drive consist of two DC motors, thus to accurately model the relations, two motor subsystem as described in section 3.2.2, are used.

3.3.1 DC motors model analysis

The two DC motors are in the top left corner of the model in figure 3.8. A step function block is used to simulate the voltage applied to the motor, alongside a saturation block that limits the model from computing with values greater than the maximum allowed voltage in the physical motor.

The motor model proposed in section 3.2.2 is constructed using only linear blocks, thus the open-loop response could be observed by providing a unit step input and observing the response through a scope block.

Figure 3.9 is consistent with the expected step response of a DC motor. When 1 Volts is applied as a step input, the motor achieves maximum speed of 0.06 cm/s (after conversion to linear speed) However to further understand the results, linear analysis on the subsystem has been performed.

From figure 3.10, it is clear that the open-loop subsystem has two real poles in the left hand plane. That is ,no oscillations or overshoot present as it could be seen in the step response. Furthermore, the slower of the two poles will dominate the

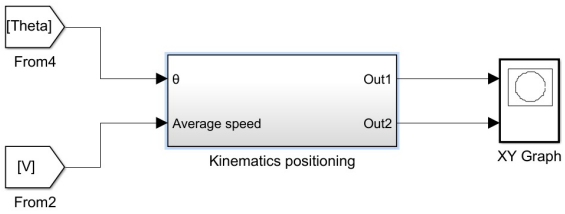


Figure 3.6: Positioning subsystem

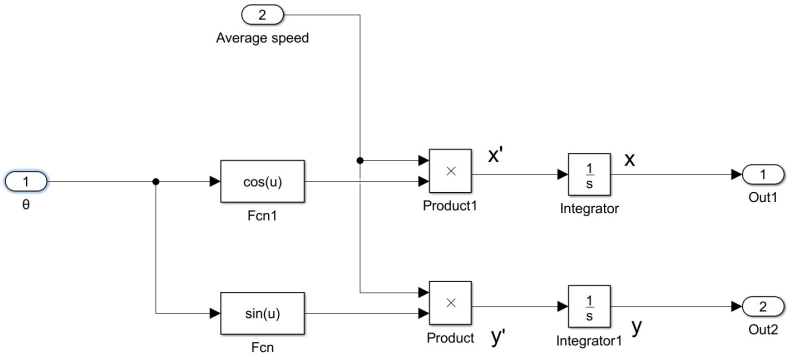


Figure 3.7: Positioning subsystem

dynamics of the system, prompting the system to behave as it was first-order.

Obviously, both motors' models behave the same while applying the same step input. Nevertheless, it is important to understand that in real life this may not be the case, as the constants used in the model may not reflect accurately both real-life counterparts.

3.3.2 Kinematics model analysis

We discussed in subsection 3.1 how small fluctuation in the speed of each motor, will results in a change of the trajectory of the whole robot. It is important to verify that the cases laid in the above mentioned subsection hold true.

In figure 3.11a, when both wheel velocities match ($v_r = v_l$), the trajectory the robot partakes is a straight line.

As to be expected, the turning speed in figure 3.11b remains zero for as long as the velocities of each wheel match.

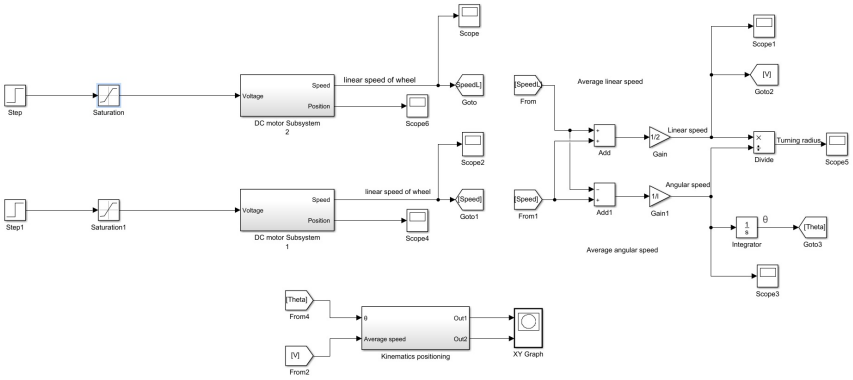


Figure 3.8: Complete system model

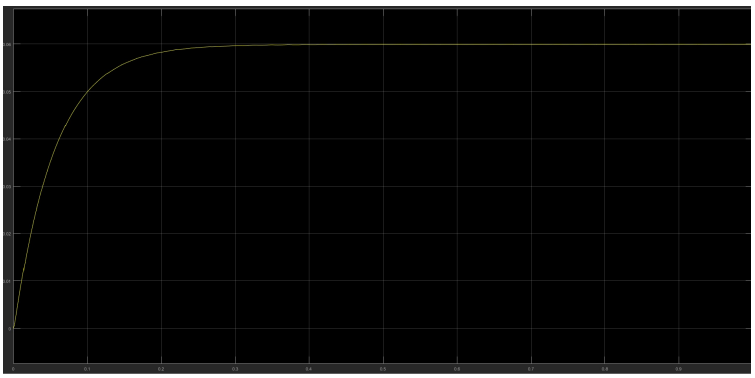


Figure 3.9: Step response of DC motor

The other case scenario is when one of the wheels has zero velocity (the other wheel's velocity can not be equal to zero).

The expected behaviour is rotation where the ICC is positioned at the zero velocity wheel. Furthermore, the turning speed should be constant, while the turning radius equal to $\frac{l}{2}$.

In figure 3.13a it takes 1.5 seconds to reach constant turning speed, while in figure 3.13b, the turning radius is exactly 0.052 cm, which is exactly half of the distance between wheels. This satisfies the expected behaviour.

However to truly picture why necessary control needs to be applied in order to maintain constant speed for both wheels, a simulation was performed where the inputs of the step function differ only by 0.1 V. That is, $v_r \approx v_l$.

In figure 3.14 we can clearly see that the trajectory the robot partakes is similar to an arc. That is, even small fluctuation in the speed of one motor, will result in a circular motion with albeit a relatively large turning radius (3.14b.

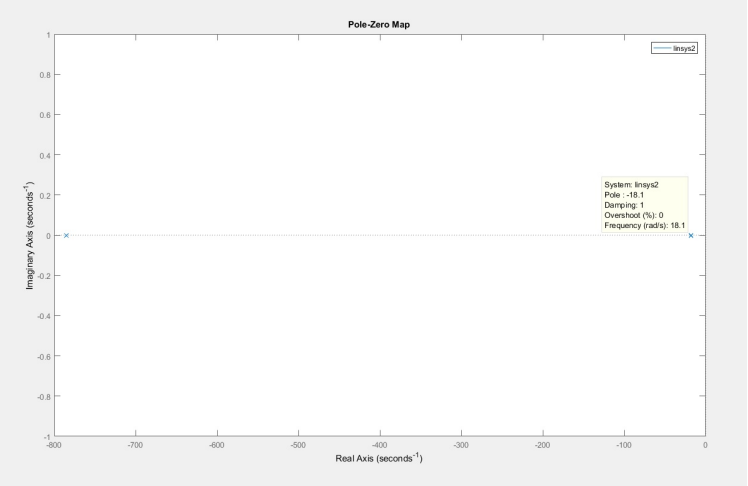
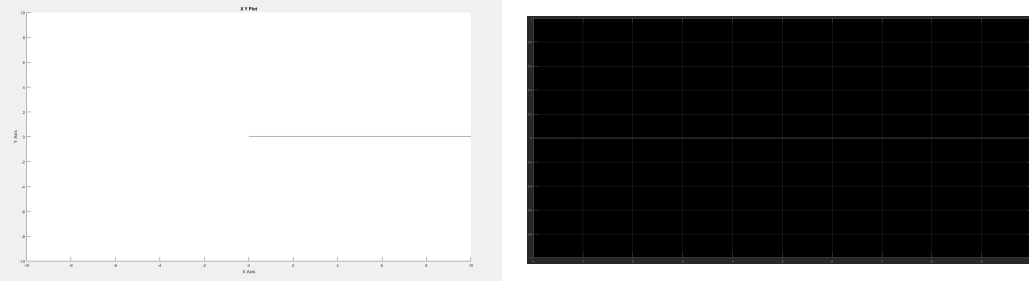


Figure 3.10: Pole-Zero map of DC motor



(a) XY-graph

(b) Turning speed

Figure 3.11: For equal velocities

Clearly, a controller is necessary to maintain constant speed in both wheel, when the robot is required to follow a linear trajectory.

3.4 Cruise Control

In this section, the problem of necessary control is addressed.

3.4.1 PID controller

As it was previously discussed, small fluctuation in the individual speeds of the wheels would cause distortion in the linear movements. Differential drive robots are a perfect example of the necessity of a control algorithm.

First of all, it needs to be mentioned that in a real life scenarios, several tools would be used in order to provide basic navigation. The simplest way to limit the motor's speed to a certain percentage would be to use Pulse Width Modulation

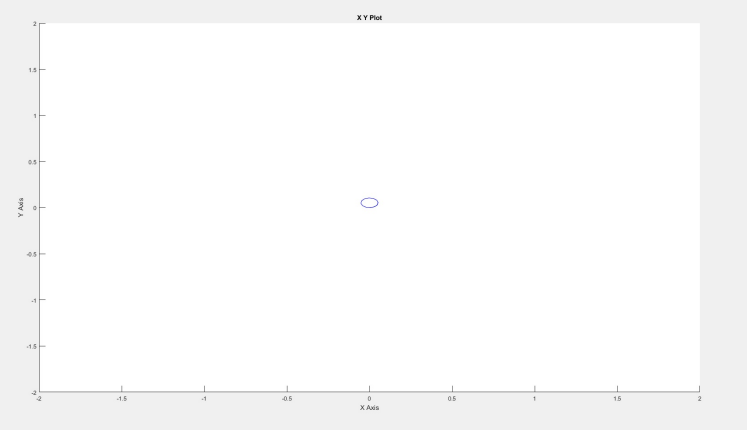
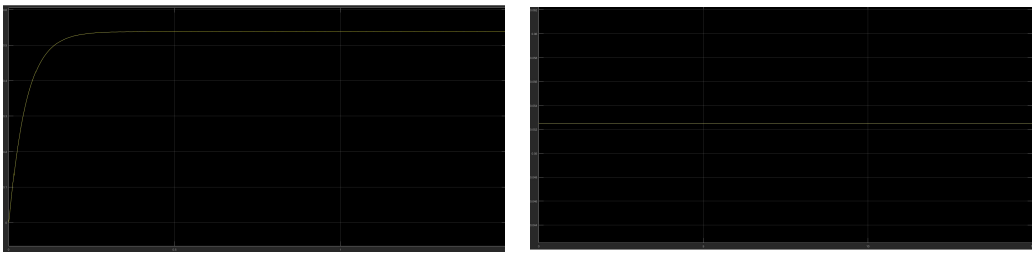


Figure 3.12: XY-graph for zero velocity left wheel



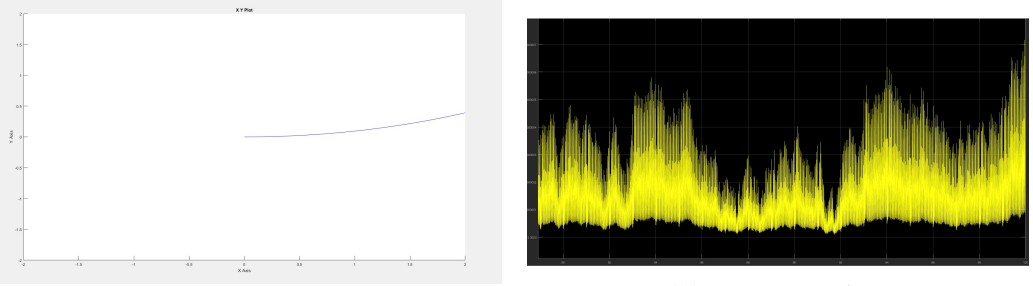
(a) Turning speed **(b)** Turning radius

Figure 3.13: For zero velocity left wheel

(PWM). Nevertheless, it would remain an open loop system, that is, no information would be provided to verify that the individual speeds of the wheels match. In order to create a closed loop system, some feedback is needed. The most common way would be with an encoder. This way, the individuals speed of each wheel could be compared and theoretically adjusted. Proportional integral derivative (PID) controller is potential solution. In a differential drive robot, the idea would be to have a PID controller for both motors and feedback from the encoders. For example, if the goal is to maintain constant speed (which would ensure linear movement), a setpoint value would be needed, as well as constant feedback from the encoders. The two values would be compared and an error value will be generated:[7]

$$error = setpoint - encoder\,reading \tag{3.23}$$

The error value would be used to calculate the amount of alterations required on the motor speed to reduce the error to a theoretical zero. That is, a plain Proportional controller, would feed the error value to the PWM output after multiplying it by the proportional gain constant.



(a) XY-graph

(b) Turning radius

Figure 3.14: For different velocities

$$C = eK_p \tag{3.24}$$

Where C is the control variable, e is the error and K_p the proportional gain. However, if a large K_p value is used, it may cause overshoot followed by oscillations that would result in potential jittering as robot's behaviour. This could be solved by using a PD controller, where the rate of the change of the error is taken into account as well ($\frac{de}{dt}$).

$$C = eK_p + \frac{de}{dt}K_d \tag{3.25}$$

Where K_d is the derivative gain. In general a PD controller might be sufficient enough to maintain constant speed in the motor, nevertheless adding an Integral control could improve steady state performance by integrating all previous errors and detect accumulating errors. Last the complete equation of a PID controller:

$$C = eK_p + K_i \int_0^t e\,dt + \frac{de}{dt}K_d \tag{3.26}$$

3.4.2 Simulink Closed-loop Response

Understanding how a PID controller could potentially remove fluctuations in the motors' speed, was necessary to apply it to the motors model from section 3.2.2.

In figure 3.15, the closed-loop control system is shown. The system consist of a PID controller, a gain block to represent the PWM output (in percentage), and a negative feedback from the angular speed of the motor (represented by an encoder in real life).

In figure 3.16, the yellow curve represents the open-loop step response, while the blue curve the closed-loop step response. The final value in the step function is set to 1, and it is obvious that steady state error is quite large in the open-loop case and lower but not gone in the close-loop case. Thus to further adjust the PID values, a set of design criteria had to be met.

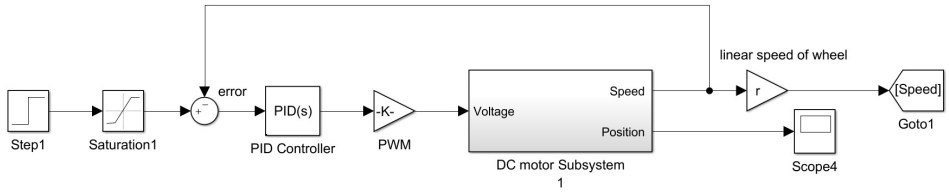


Figure 3.15: Closed-loop motor subsystem

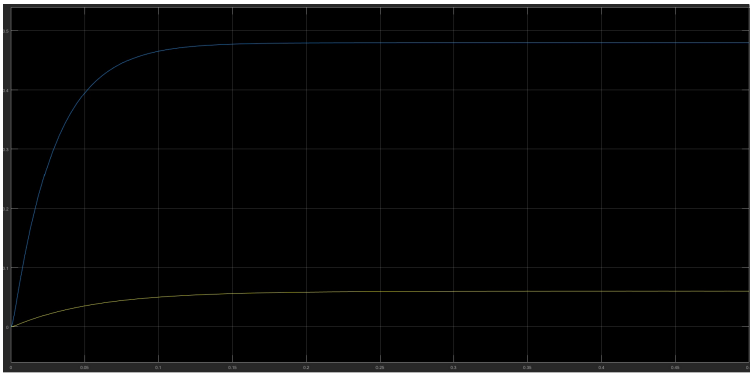


Figure 3.16: Comparison between open-loop and closed-loop step response

- Settling time < 1 sec
- Overshoot < 5 %
- Steady-state error < 1%

Using Matlab PID tuner, the design criteria are met when the values correspond to those in figure 3.17.

Although the tuned values cover the design criteria, a more practical approach such as Ziegler–Nichols method has to applied when designing a controller for the physical robot system.

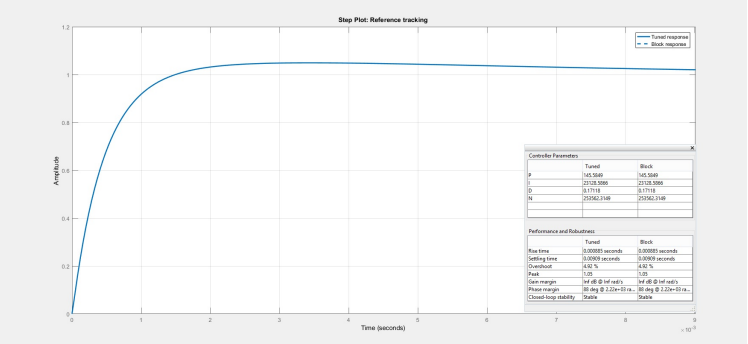


Figure 3.17: Matlab PID tuner and gains adjustment

Chapter 4

Development

Hardware testing

4.1 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.

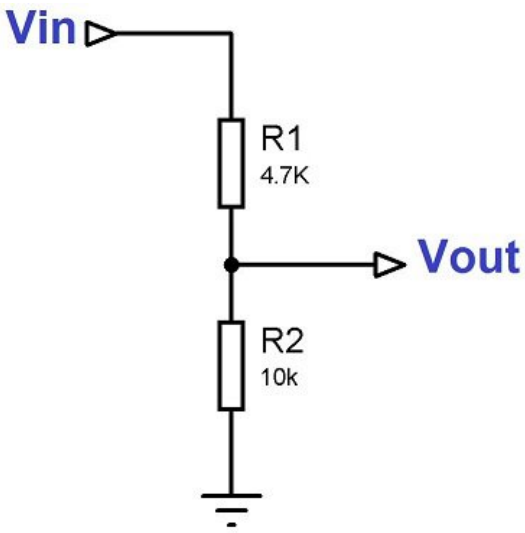


Figure 4.1: Voltage divider

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

$$V_{out} = V_{in} * R_2/(R_1 + R_2) \tag{4.1}$$

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 where 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 reading 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 $3 \times 0.05s = 0,15s$.

4.2 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.

4.3 Raspberry Pi configuration and software

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

4.3.1 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 it's 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.

4.3.2 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. [10]

- **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. [11]

- **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
ECHOF=4
ECHOL=27
ECHOR=22
TRIG=17

#Values for reading the sesnsors
SPEED_OF_SOUND = 17150
measurment_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
MEDSPEED = 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(ECHOF, GPIO.IN)
GPIO.setup(ECHOL, GPIO.IN)
GPIO.setup(ECHOR, 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)
```

4.3.3 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 4.1, 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.

$$330(m/s) * 0.05s = 16.5 \tag{4.2}$$

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

$$16.5/2 = 8.25m \tag{4.3}$$

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 == ECHOF:
        sensorF_data=distance

if PIN == ECHOR:
        sensorR_data=distance

if PIN==ECHOL:
        sensorL_data=distance
```

4.3.4 Movements

The expected movements of a differential drive robot was covered in section 3.3.2. 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 3.3.2. 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 4.3.5: 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.

4.3.5 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.

4.4 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 5

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

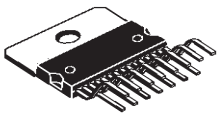
Driver Chip

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)



Multiwatt15

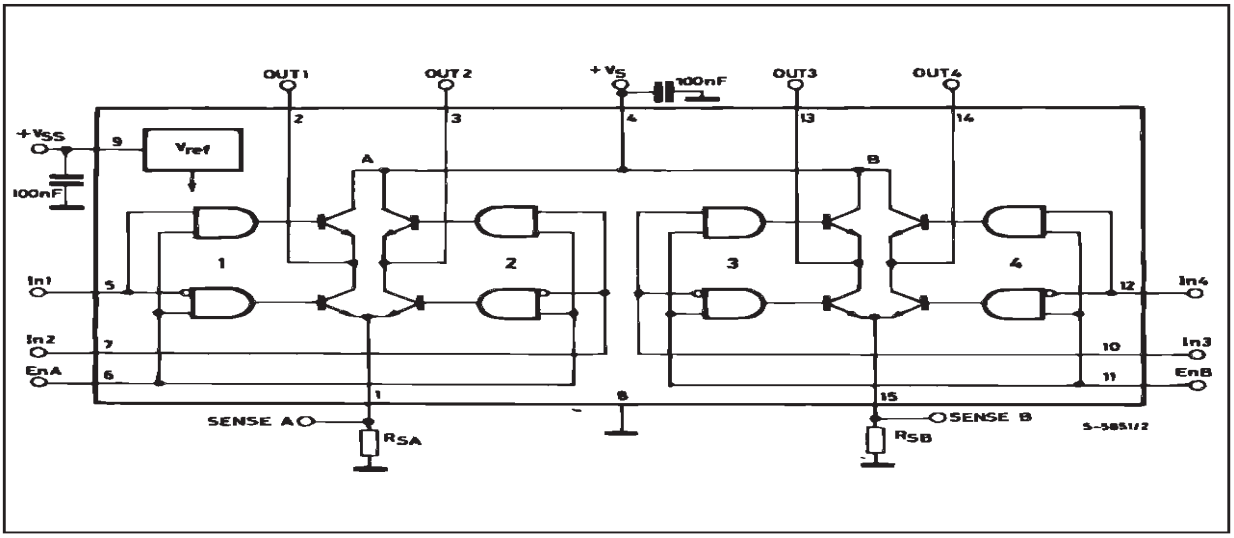
PowerSO20

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

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

BLOCK DIAGRAM



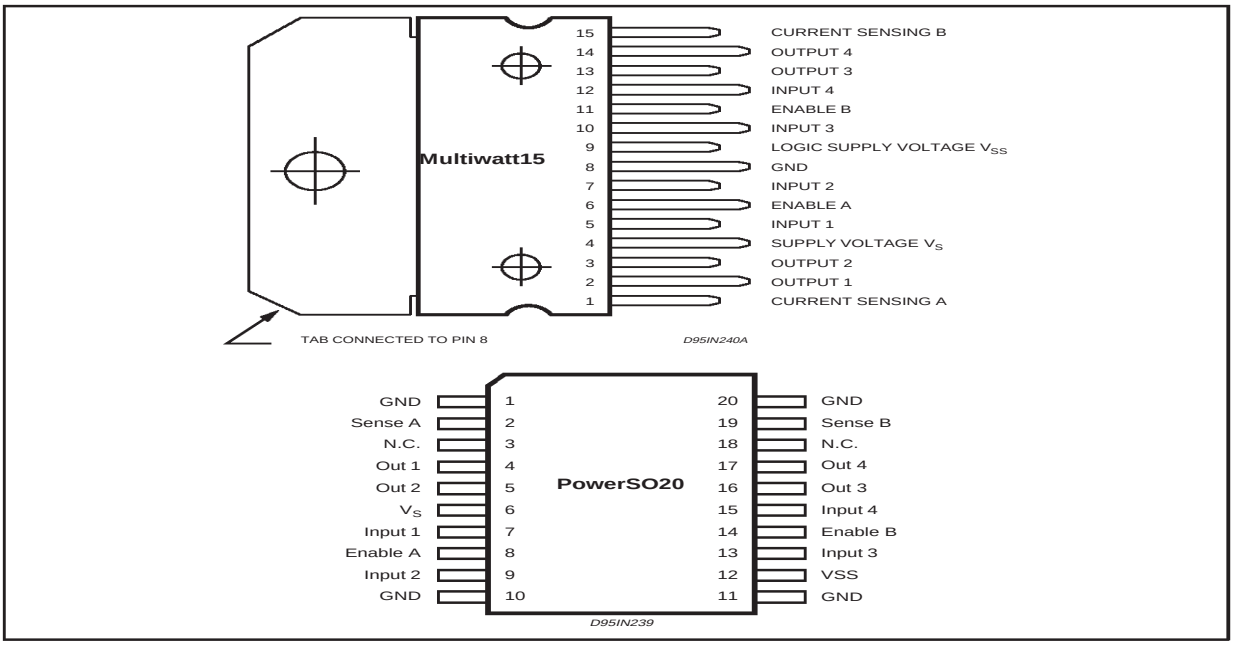
Driver Chip

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 = 100µs) –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°C)	25	W
T_{op}	Junction Operating Temperature	–25 to 130	°C
T_{stg}, T_j	Storage and Junction Temperature	–40 to 150	°C

PIN CONNECTIONS (top view)



THERMAL DATA

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

(*) Mounted on aluminum substrate

Driver Chip

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_S	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	VSS	Supply Voltage for the Logic Blocks. A100nF 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_S = 42V; V_{SS} = 5V, T_j = 25°C; unless otherwise specified)

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

Driver Chip

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		µs
T_2 (V_i)	Source Current Fall Time	0.9 I_L to 0.1 I_L (2); (4)		0.2		µs
T_3 (V_i)	Source Current Turn-on Delay	0.5 V_i to 0.1 I_L (2); (4)		2		µs
T_4 (V_i)	Source Current Rise Time	0.1 I_L to 0.9 I_L (2); (4)		0.7		µs
T_5 (V_i)	Sink Current Turn-off Delay	0.5 V_i to 0.9 I_L (3); (4)		0.7		µs
T_6 (V_i)	Sink Current Fall Time	0.9 I_L to 0.1 I_L (3); (4)		0.25		µs
T_7 (V_i)	Sink Current Turn-on Delay	0.5 V_i to 0.9 I_L (3); (4)		1.6		µs
T_8 (V_i)	Sink Current Rise Time	0.1 I_L to 0.9 I_L (3); (4)		0.2		µs
fc (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		µs
T_2 (V_{en})	Source Current Fall Time	0.9 I_L to 0.1 I_L (2); (4)		1		µs
T_3 (V_{en})	Source Current Turn-on Delay	0.5 V_{en} to 0.1 I_L (2); (4)		0.3		µs
T_4 (V_{en})	Source Current Rise Time	0.1 I_L to 0.9 I_L (2); (4)		0.4		µs
T_5 (V_{en})	Sink Current Turn-off Delay	0.5 V_{en} to 0.9 I_L (3); (4)		2.2		µs
T_6 (V_{en})	Sink Current Fall Time	0.9 I_L to 0.1 I_L (3); (4)		0.35		µs
T_7 (V_{en})	Sink Current Turn-on Delay	0.5 V_{en} to 0.9 I_L (3); (4)		0.25		µs
T_8 (V_{en})	Sink Current Rise Time	0.1 I_L to 0.9 I_L (3); (4)		0.1		µs

1) 1)Sensing voltage can be –1 V for t ≤ 50 µsec; in steady state V_{sens} min ≥ –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.

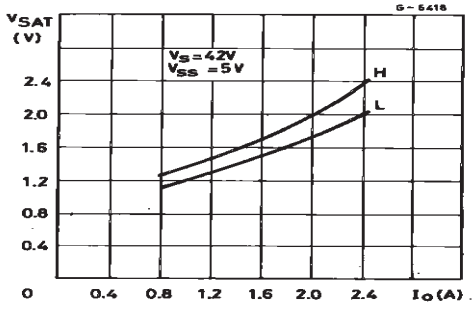
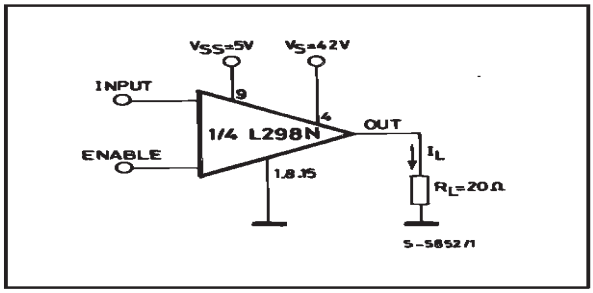


Figure 2 : Switching Times Test Circuits.



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

Driver Chip

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

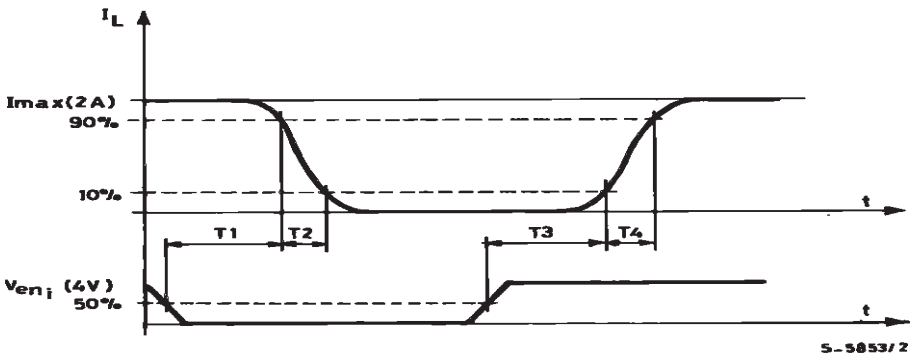
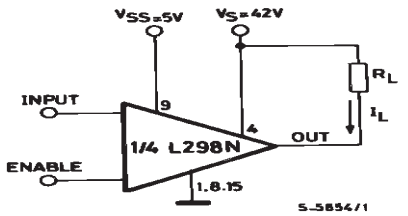


Figure 4 : Switching Times Test Circuits.



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

Driver Chip

Figure 5 : Sink Current Delay Times vs. Input 0 V Enable Switching.

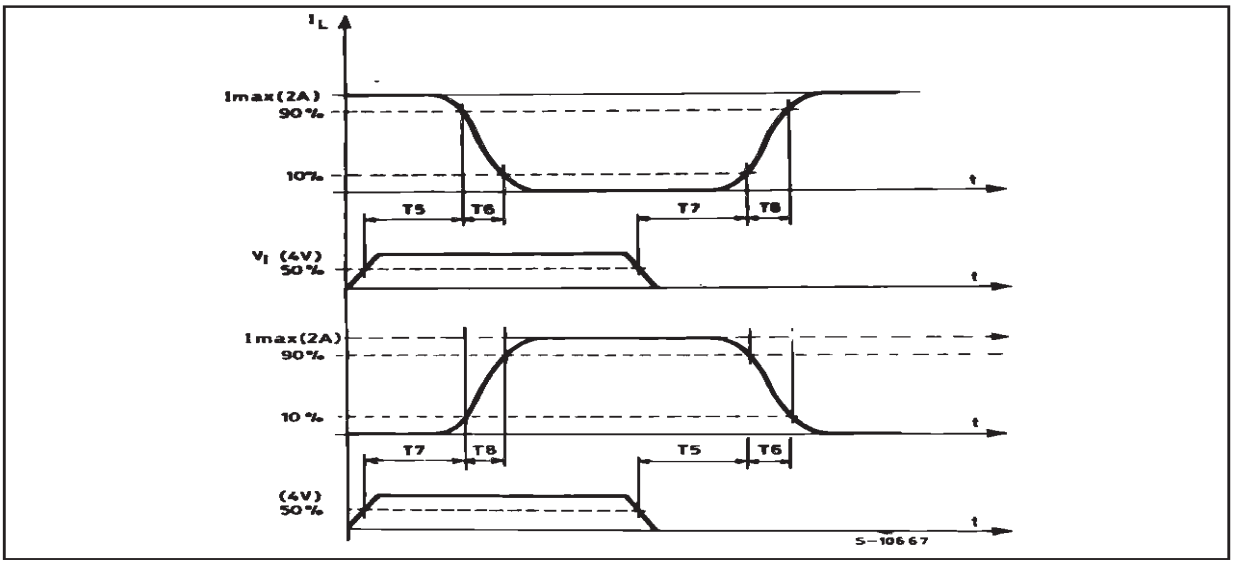
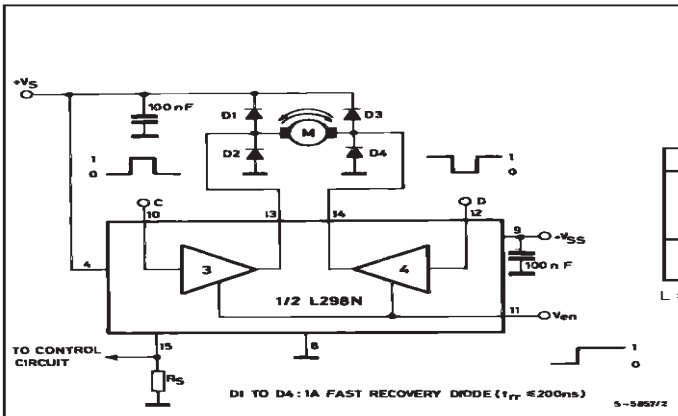


Figure 6 : Bidirectional DC Motor Control.

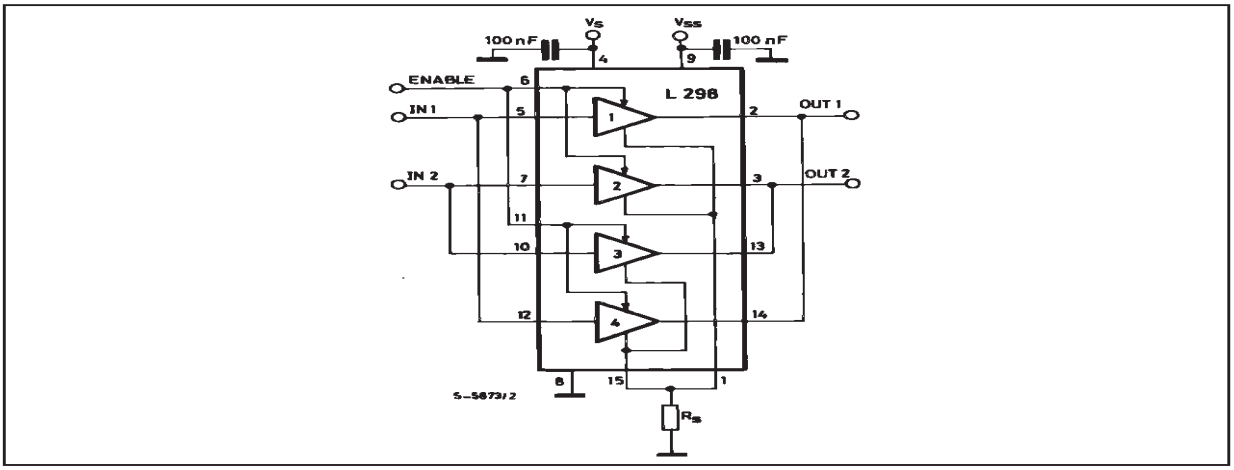


Inputs		Function
V_{en} = H	C = H ; D = L	Forward
	C = L ; D = H	Reverse
	C = D	Fast Motor Stop
V_{en} = L	C = X ; D = X	Free Running Motor Stop

L = Low H = High X = Don't care

Driver Chip

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 differenzial 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 In1 ; In2 ; EnA and In3 ; In4 ; EnB. 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 Vs and Vss, 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 Vs 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 (trr $\leq$ 200 nsec) that must be chosen of a VF 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; Shottky 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.

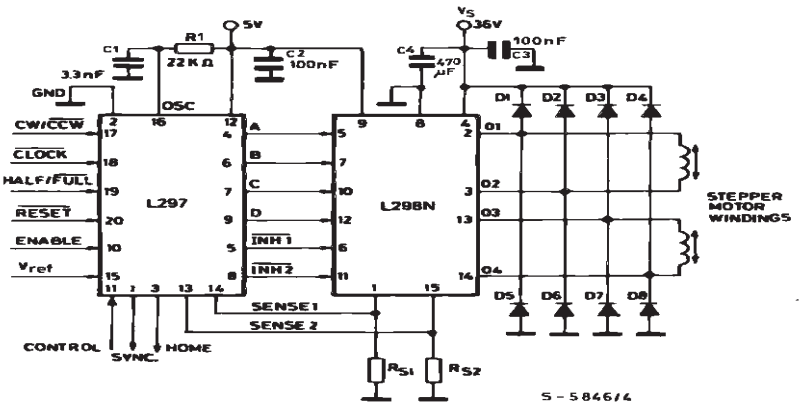
On Fig 8 it is shown the driving of a two phase bipolar stepper motor ; the needed signals to drive the inputs of the L298 are generated, in this example, from the IC L297.

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

Fig 10 shows a second two phase bipolar stepper motor control circuit where the current is controlled by the I.C. L6506.

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.



$R_{S1} = R_{S2} = 0.5\ \Omega$

D1 to D8 = 2 A Fast diodes { $V_F \leq 1.2$ V @ I = 2 A
trr ≤ 200 ns

Driver Chip

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

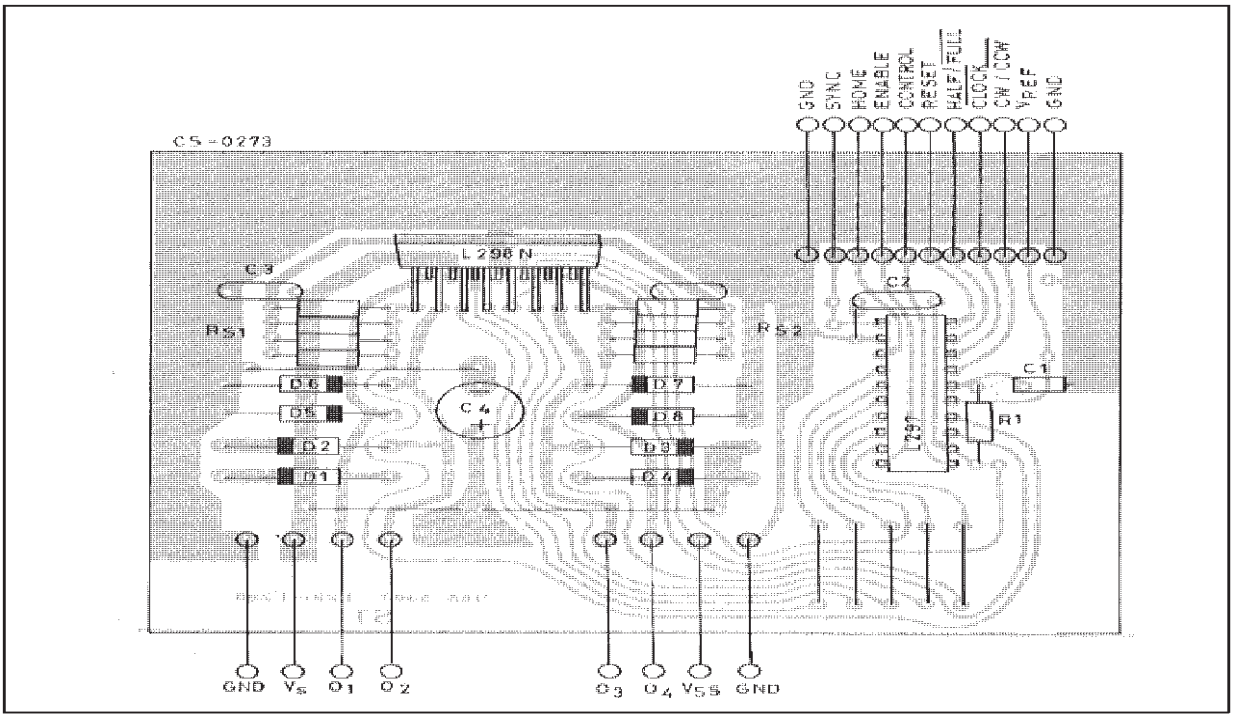
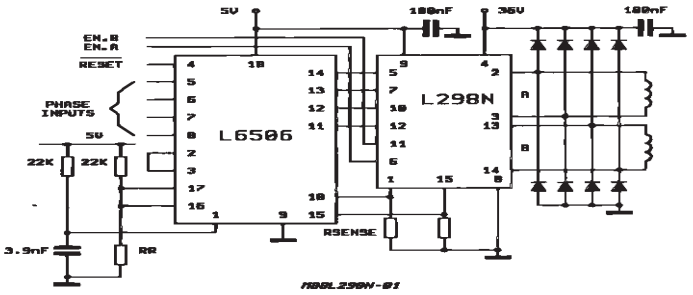


Figure 10 : Two Phase Bipolar Stepper Motor Control Circuit by Using the Current Controller L6506.



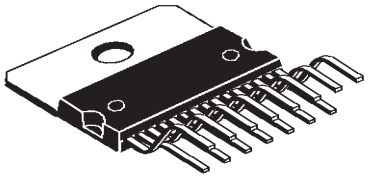
R_R and R_{sense} depend from the load current

Driver Chip

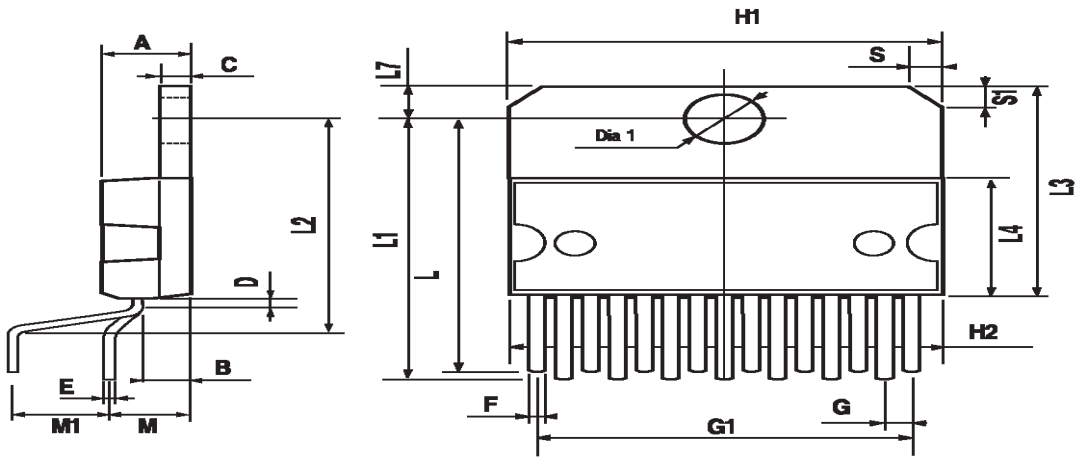
L298

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



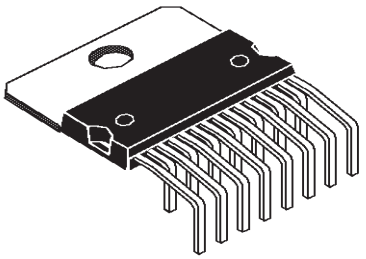
Multiwatt15 V



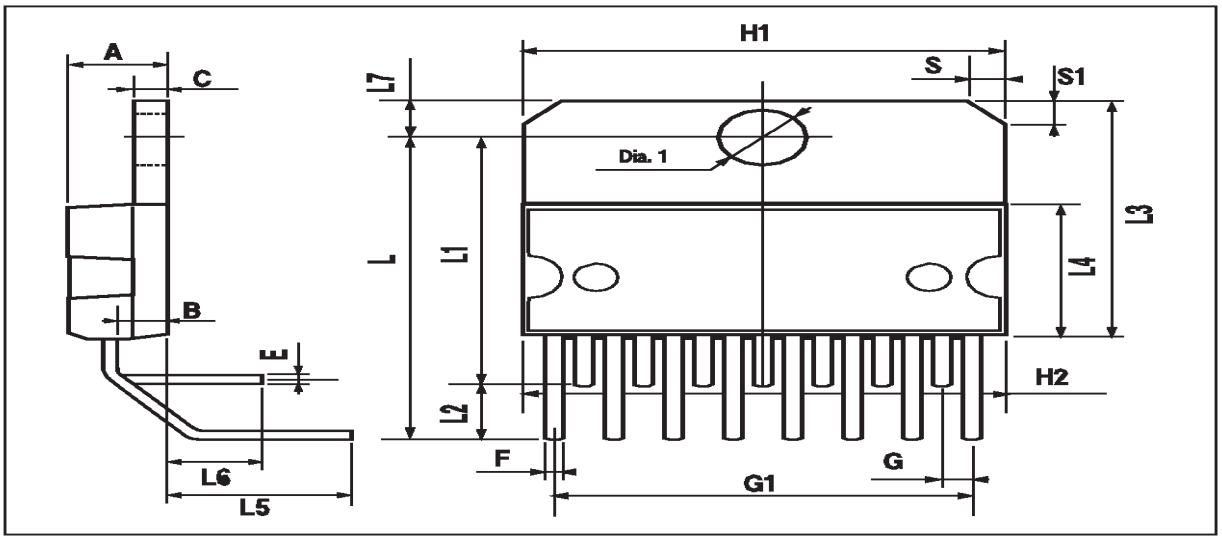
Driver Chip

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

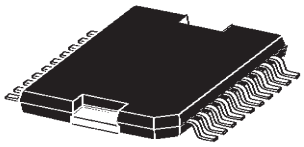


Driver Chip

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	

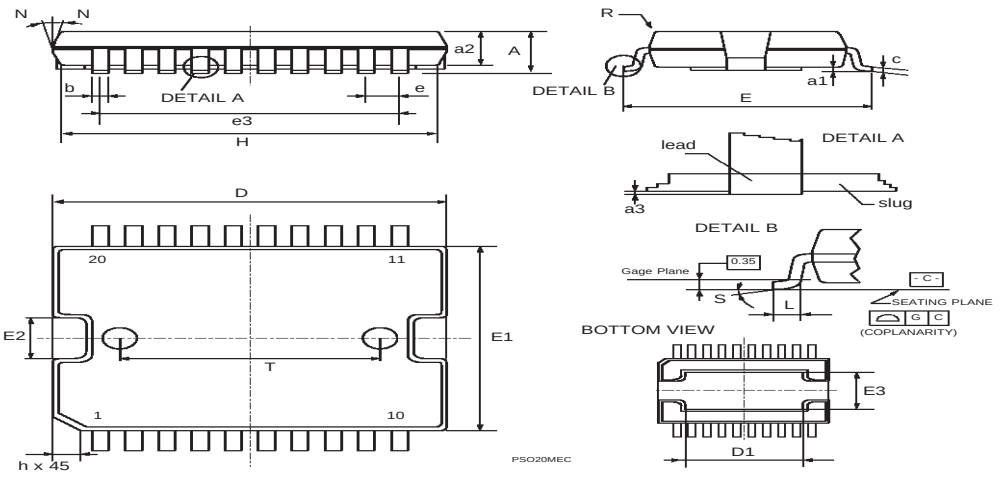
(1) "D and F" do not include mold flash or protrusions.
- Mold flash or protrusions shall not exceed 0.15 mm (0.006").
- Critical dimensions: "E", "G" and "a3"

OUTLINE AND MECHANICAL DATA



JEDEC MO-166

PowerSO20



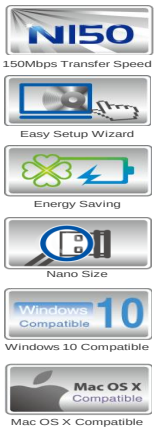
Driver Chip

Appendix B

Wi-Fi Adapter

Wi-Fi Adapter

EDIMAX
NETWORKING PEOPLE TOGETHER



EW-7811Un

N150 Wi-Fi Nano USB Adapter
Ideal for Raspberry Pi

EW-7811Un is a nano USB wireless adapter that supports maximum range and speed. Despite the size, this tiny USB adapter supports higher data rate of up to 150Mbps when connecting with wireless 802.11n device which is 3 times faster than your normally 11g connection. You can just plug it into computer's USB port and enjoy incredible high-speed wireless network access. This is for sure the trendiest piece of upgrade you can make to your wireless network.

Complies with wireless 802.11n standards with data rate up to 150Mbps
EW-7811Un complies with wireless IEEE802.11b/g/n standards. As being built with the latest wireless technology, the EW-7811Un can increase your wireless coverage up to 3 times* as much and greatly help reduce "dead spots" compared to your old 11g adapter. The transmission data rate can go up to 150Mbps when connected to a 802.11n device, with signal travel further and connection a lot more stable compared to your previous 802.11g network .

Supports Green WLAN
EW-7811Un has adapted a clever protocol in smart transmission power control. The adapter smartly adjust transmission output by distance and CPU offload to help reduce power consumption when wireless is idle. With the green WLAN technology, the power consumption can be reduced up to 20% ~ 50%. The adapter does not just provide you with the best wireless technology in the world, it also care for the environment by using less energy and your pocket by saving up on your electric bills.

Supports WPS (Wi-Fi Protected Setup)
Wi-Fi Protected Setup (WPS) is a standard for easy and secure establishment of a wireless network. This wireless USB adapter supports software WPS-compatible configuration. By pressing 2 buttons, your wireless network is immediately secured providing your wireless router will also support this great easy function.

Supports multi-language easy setup wizard
The included 16 languages easy setup wizard and friendly UI will walk you through configuring EW-7811Un to your wireless network.

Portable and compact design
EW-7811Un is currently the smallest wireless adapter. The compact design of EW-7811Un is convenient to carry for all mobile users. It also means you will not be able to accidentally snap your USB adapter anymore as it is hidden very well once it is plugged into the USB port.

*Maximum performance, actual data rates, and coverage will vary depending on network conditions and environmental factors. Product specifications and design are subject to change without notice.

www.edimax.com

Wi-Fi Adapter

EDIMAX
NETWORKING PEOPLE TOGETHER

www.edimax.com

N150 Wi-Fi Nano USB Adapter
Ideal for Raspberry Pi

EW-7811Un

FEATURES AND TECHNICAL SPECIFICATIONS

- Complies with wireless 802.11b/g/n standards with data rate up to 150Mbps
- Green Power Saving : supports smart transmit power control and auto-idle state adjustment
- Increases wireless coverage 3 times* further
- Includes multi-language easy setup wizard
- Supports 64/128-bit WEP, WPA , WPA2 encryption and WPS-compatible
- Supports QoS-WMM, WMM-Power Save mode

TECHNICAL SPECIFICATIONS

HARDWARE INTERFACE	STANDARD	FREQUENCY BAND
1 USB 1.0/2.0 Type A Internal Antenna	IEEE802.11b, 802.11g, 802.11n	2.4000~2.4835GHz (Industrial Scientific Medical Band)
DATA RATE	**INSTALLATION**	**LED & DIMENSION**
11b: 1/2/5.5/11Mbps 11g: 6/9/12/24/36/48/54Mbps 11n (20MHz): MCS0-7 (up to 72Mbps) 11n (40MHz): MCS0-7 (up to 150Mbps)	Multi-language easy setup wizard with auto run configuration	Link/Activity 7.1(H) x 14.9 (W) x 18.5 (D) mm
SECURITY	**OUTPUT POWER**	**HUMIDITY & TEMPERATURE**
WEP 64/128, WPA and WPA2 Software WPS configuration	11b:17 ±1.5dbm 11g:b:15 ±1.5dbm 11n:14 ±1.5dbm	Operating : 10~90% (Non Condensing) Storage : Max. 95% (Non Condensing) Operating : 32~104°F (0~40°C) Storage : -4~140°F (-20~60°C)
SYSTEM REQUIREMENTS	**RECEIVE SENSITIVITY**	**CERTIFICATIONS**
Windows XP/Vista/7/8/8.1/10 Linux & Mac OS	11n(20MHz)@MCS7: -68dBm±2dBm 11n(40MHz)@MCS7: -64dBm±2dBm 11g@54Mbps: -71dBm±2dBm 11b@11Mbps: -81dBm±2dBm	CE, FCC , WiFi

APPLICATION DIAGRAM

An example of how the EW-7811Un can be setup:

- Connects the EW-7811Un wireless adapter to your computer.
- Setups the wireless connection by running the multi-language easy setup wizard.
- Connects to 802.11b/g/n wireless access point or broadband router.

Ideal for Raspberry Pi/Pi 2
Plug & Play!

RoHS FC CE

www.edimax.com

EDIMAX
NETWORKING PEOPLE TOGETHER

Edimax Technology Co., Ltd
6F., No.3, Wu-Chuan 3rd Road, Wu-Gu,
New Taipei City 24891, Taiwan
Email: sales@edimax.com.tw

Edimax Technology Europe B.V.
Fijenhof 2, 5652 AE Eindhoven,
The Netherlands
Email: sales@edimax.nl

Edimax Computer Company
3350 Scott Blvd., Bldg.15 Santa Clara,
CA 95054, USA
Email : sales@edimax.com

Appendix C

Ultrasonic Sensor

Ultrasonic Sensor

Tech Support: services@elecfreaks.com

Ultrasonic Ranging Module HC - SR04

Product features:

Ultrasonic ranging module HC - SR04 provides 2cm - 400cm non-contact measurement function, the ranging accuracy can reach to 3mm. The modules includes ultrasonic transmitters, receiver and control circuit. The basic principle of work:
(1) Using IO trigger for at least 10us high level signal,
(2) The Module automatically sends eight 40 kHz and detect whether there is a pulse signal back.
(3) IF the signal back, through high level , time of high output IO duration is the time from sending ultrasonic to returning.
Test distance = (high level time×velocity of sound (340M/S) / 2,

Wire connecting direct as following:

- 5V Supply
- Trigger Pulse Input
- Echo Pulse Output
- 0V Ground

Electric Parameter

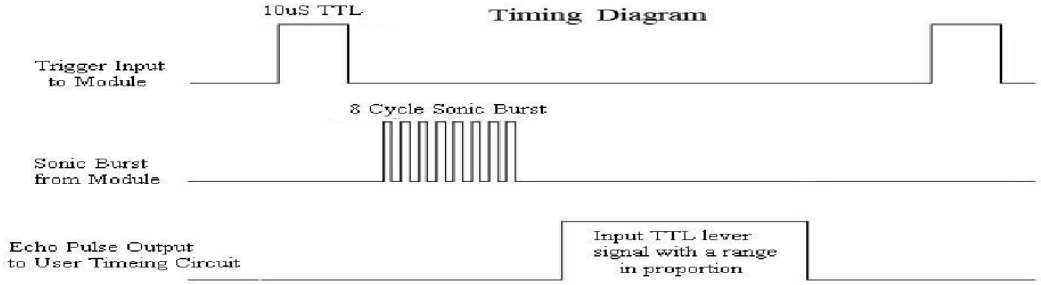
Working Voltage	**DC 5 V**
Working Current	**15mA**
Working Frequency	**40Hz**
Max Range	**4m**
Min Range	**2cm**
MeasuringAngle	**15 degree**
Trigger Input Signal	**10uS TTL pulse**
Echo Output Signal	**Input TTL lever signal and the range in proportion**
Dimension	**45*20*15mm**

Ultrasonic Sensor

Vcc Trig Echo GND

Timing diagram

The Timing diagram is shown below. You only need to supply a short 10uS pulse to the trigger input to start the ranging, and then the module will send out an 8 cycle burst of ultrasound at 40 kHz and raise its echo. The Echo is a distance object that is pulse width and the range in proportion .You can calculate the range through the time interval between sending trigger signal and receiving echo signal. Formula: uS / 58 = centimeters or uS / 148 =inch; or: the range = high level time * velocity (340M/S) / 2; we suggest to use over 60ms measurement cycle, in order to prevent trigger signal to the echo signal.



Ultrasonic Sensor

Attention:

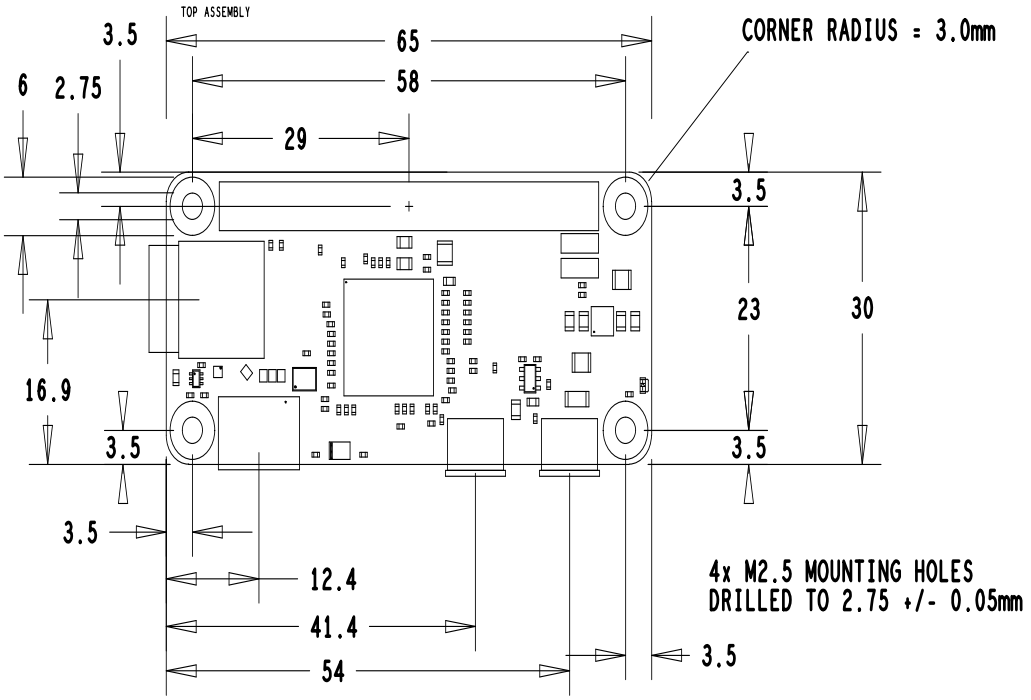
- The module is not suggested to connect directly to electric, if connected electric, the GND terminal should be connected the module first, otherwise, it will affect the normal work of the module.
- When tested objects, the range of area is not less than 0.5 square meters and the plane requests as smooth as possible, otherwise ,it will affect the results of measuring.

www.Elecfreaks.com

Appendix D

Raspberry Pi

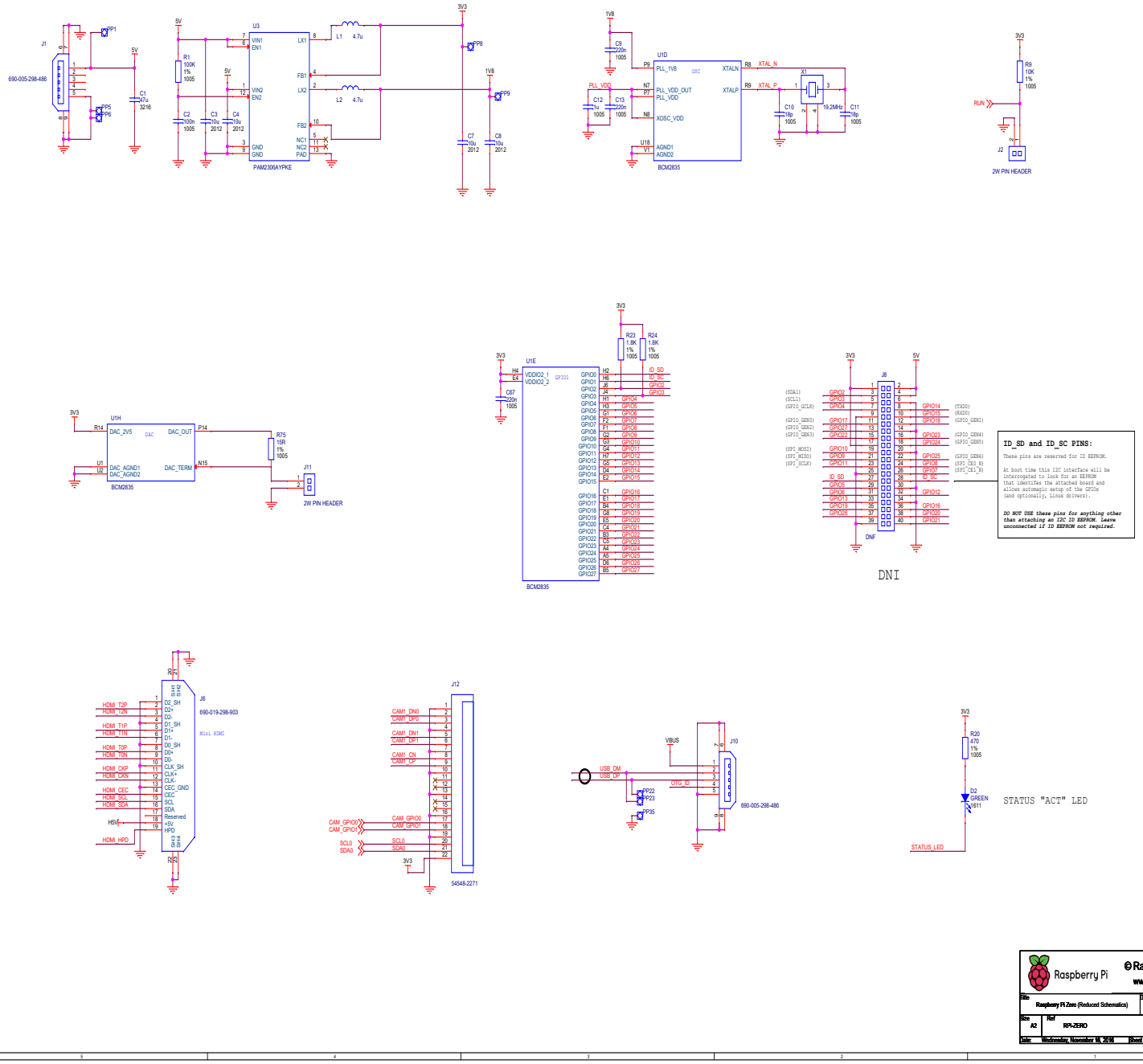
Raspberry Pi



Raspberry Pi
www.raspberrypi.org
© Raspberry Pi 2015

TITLE	RASPBERRY PI ZERO		
DATE	23/09/2015	REF	RPI-ZERO-V1_2
DRAWN	Mike Stimson	APVD	James Adams

Raspberry Pi



Appendix E

Source Code

Source Code

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



#PIN numbers
LetfPWM=16
RightPWM=20
StepPinForward1=26
StepPinBackward1=19
StepPinForward2=13
StepPinBackward2=6
ECHOF=4
ECHOL=27
ECHOR=22
TRIG=17

#Values for reading the sesnsors
SPEED_OF_SOUND = 17150
measurment_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
MEDSPEED = 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(ECHOF, GPIO.IN)
GPIO.setup(ECHOL, GPIO.IN)
GPIO.setup(ECHOR, 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)

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))
```

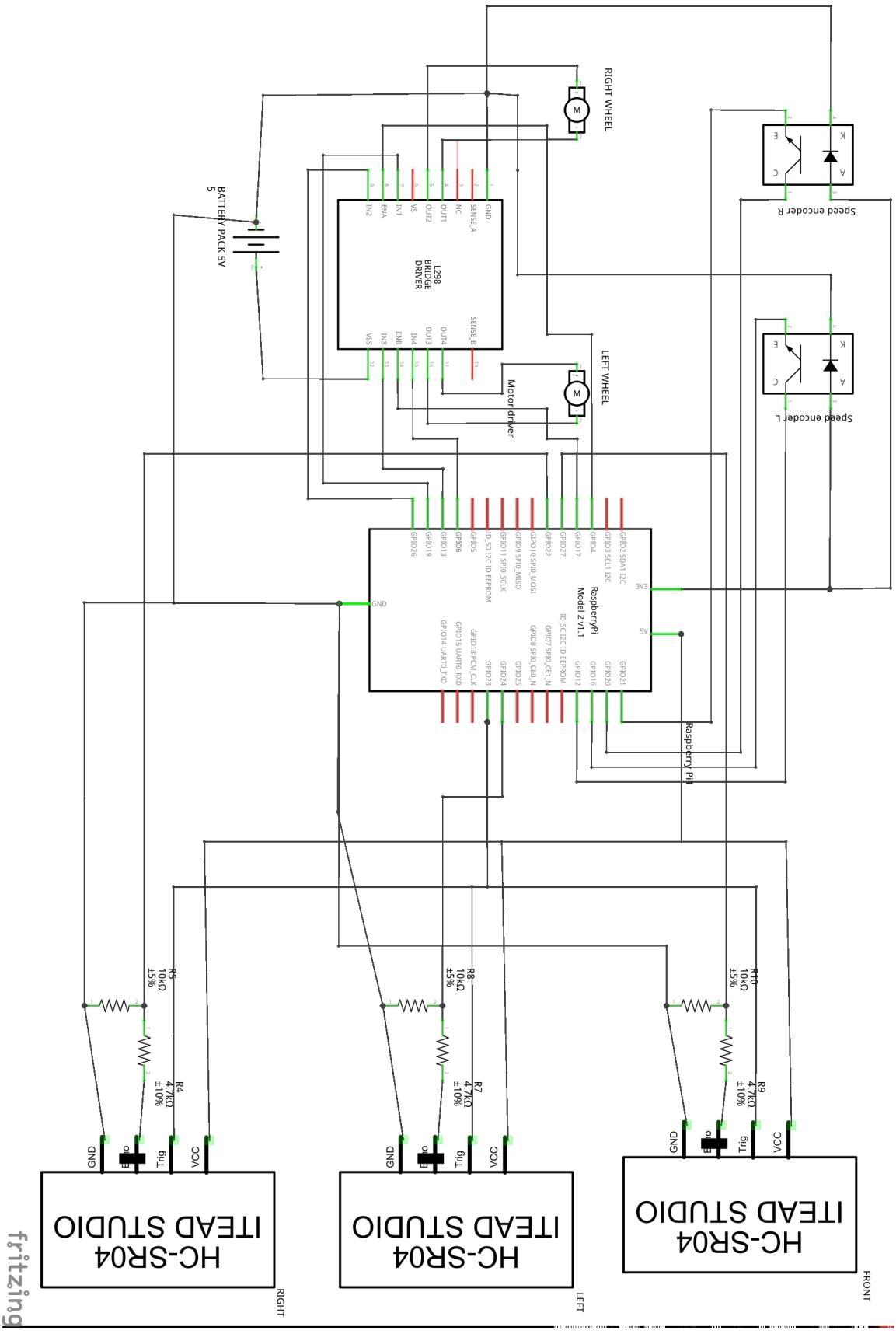
```
        distance = sum(pulse_duration)/measurment_count* SPEED_OF_SOUND
        distance = round(distance, 2)
        print distance
        if PIN == ECHOF:
                sensorF_data=distance

        if PIN == ECHOR:
                sensorR_data=distance

        if PIN==ECHOL:
                sensorL_data=distance

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


def reverse(reversetime,SPEED):
        print "FORWARD"
        GPIO.output(StepPinForward1, GPIO.HIGH)
        GPIO.output(StepPinForward2, GPIO.HIGH)
        PWML.start(SPEED)
        PWMR.start(SPEED)
        sleep(reversetime)
        GPIO.output(StepPinForward1, GPIO.LOW)
        GPIO.output(StepPinForward2, GPIO.LOW)

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)

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)

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
```

Appendix F

Schematics


RIGHT WHEEL
LEFT WHEEL
BATTERY PACK 5V
L298 BRIDGE DRIVER
Motor driver
Speed encoder R
Speed encoder L
RaspberryPi Model 2 v1.1
Raspberry Pi
HC-SR04 ITEAD STUDIO
RIGHT
LEFT
FRONT
fritzing