

This page was intentionally left blank.

This page was intentionally left blank.

Scaled Test Bed for Automotive Experiments: Evaluation of Electronic Stability Control Schemes

Master thesis of

Diomidis I. Katzourakis

diomidis@systems.tuc.gr, diomkatz@gmail.com

ECE, Technical University of Crete, Greece

Supervising Committee

Master's Advisor

Dr. Yannis Papaefstathiou

ygp@mhl.tuc.gr

Assistant Professor, Microprocessor & Hardware Lab.

ECE, Technical University of Crete, Greece

Dr. Apostolos Dollas

dollas@mhl.tuc.gr

Professor, Microprocessor & Hardware Lab.

ECE, Technical University of Crete, Greece

Dr. Michail G. Lagoudakis

lagoudakis@intelligence.tuc.gr

Assistant Professor, Intelligence Systems Lab.

ECE, Technical University of Crete, Greece

This page was intentionally left blank.

This page was intentionally left blank.

Abstract

Evaluation and testing of electronic stability control systems in automotive vehicles, in the real environment confronts with cost and safety concerns leading to an overdue prototyping of the actual system. This master thesis, addressees the implementation of a scaled test bed for automotive experiments, based on 1:5 scaled model car especially designed for rapid system prototyping. A single gyroscope electronic stability control scheme has been formatted, post - process simulated and evaluated on the test bed. The improvement on the vehicle's stability is noticeable.

The custom developed model car acting as experimental platform is fully equipped with sensors, actuators, a controller to collect data and a Linux based computer system to process data. By using a scaled model car we introduce realistic simulation dynamics and disturbances. The reference model for stabilization is based upon the dynamics of the Bicycle Model. The Stability Control System issues commands to individual vehicle's brakes in order to reduce the error between the actual and desired response of the car.

The thesis is organized into six sections:

- I. Introduction
- II. Vehicle Dynamics and Stabilization Algorithms
- III. Similar work
 - i. Existing ESPs
 - ii. Published ESPs
 - iii. Comparison between published ESPs and our system
- IV. System Implementation
 - i. Mechanical modifications
 - ii. Computing hardware
 - iii. Software
- V. Real Environment Evaluation
- VI. Conclusions and Future Work
- VII. Appendix
 - i. Host codes
 - ii. Microcontroller Codes
 - iii. Matlab Codes

The author would like to dedicate this document to his parents and to Elli Barla for their long lasting patience and support.

Υπό Κλίμακα Πλατφόρμα για Αυτοκινητιστικά Πειράματα: Αξιολόγηση Συστημάτων Ηλεκτρονικού Ελέγχου Ευστάθειας

Μεταπτυχιακή Διατριβή του

Κατζουράκη Ι. Διομήδη

diomidis@systems.tuc.gr, diomkatz@gmail.com

HMMY, Πολυτεχνείο Κρήτης, Ελλάδα

Εξεταστική Επιτροπή

Επιβλέπων

Δρ. Παπαευσταθίου Γιάννης

ygp@mhl.tuc.gr

Επίκουρος Καθηγητής, Εργαστήριο Μικροπεξεργαστών & Υλικού

HMMY, Πολυτεχνείο Κρήτης, Ελλάδα

Δρ. Δόλλας Απόστολος

dollas@mhl.tuc.gr

Καθηγητής, Εργαστήριο Μικροπεξεργαστών & Υλικού

HMMY, Πολυτεχνείο Κρήτης, Ελλάδα

Δρ. Λαγουδάκης Γ. Μιχαήλ

lagoudakis@intelligence.tuc.gr

Επίκουρος Καθηγητής, Εργαστήριο Ευφών Συστημάτων

HMMY, Πολυτεχνείο Κρήτης, Ελλάδα

This page was intentionally left blank.

This page was intentionally left blank.

Περίληψη

Η αξιολόγηση και η δοκιμή συστημάτων ηλεκτρονικής ευστάθειας σε αυτοκινούμενα οχήματα στο πραγματικό περιβάλλον έρχεται αντιμέτωπη με τα θέματα του υψηλού κόστους και την ανάγκη για ασφάλεια, τα οποία οδηγούν στην καθυστερημένη προτυποποίηση του τελικού συστήματος. Η μεταπτυχιακή αυτή διατριβή αποδίδει την υλοποίηση ενός υπό κλίμακα μοντέλου αυτοκινήτου για αυτοκινητιστικά πειράματα, που βασίζεται σε ένα μοντέλο αυτοκινήτου κλίμακας 1:5, ειδικά σχεδιασμένο για ταχεία προτυποποίηση. Έχει σχεδιαστεί, εξομοιωθεί και αξιολογηθεί στην πράξη ένα σύστημα ηλεκτρονικής ευστάθειας που βασίζεται σε γυροσκόπιο. Η βελτίωση της οδικής συμπεριφοράς του μοντέλου είναι αξιοπρόσεκτη.

Το ειδικά μετασκευασμένο μοντέλο, είναι πλήρως εξοπλισμένο με αισθητήρες, ενεργοποιητές, έναν ελεγκτή για την συλλογή δεδομένων και ένα υπολογιστικό σύστημα βασισμένο στο λειτουργικό σύστημα Linux για την επεξεργασία των δεδομένων. Χρησιμοποιώντας το μοντέλο, μπορούμε να εφαρμόσουμε - εξομοιώσουμε ρεαλιστικά την δυναμική συμπεριφορά και τις διαταραχές που δέχεται ένα πραγματικό σύστημα αυτοκινήτου. Το μοντέλο αναφοράς για την ευστάθεια του συστήματος, βασίζεται στο δυναμικό μοντέλο του "ποδηλάτου". Το σύστημα ευστάθειας στέλνει εντολές σε κάθε φρένο του μοντέλου ξεχωριστά για να μειώσει το σφάλμα μεταξύ της πραγματικής και της επιθυμητής απόκρισης.

Η μεταπτυχιακή αυτή διατριβή είναι οργανωμένη σε έξι ενότητες:

- I. Εισαγωγή
- II. Δυναμικό Μοντέλο Αυτοκινήτου και Αλγόριθμοι Ευστάθειας
- III. Αντίστοιχη Εργασίες
 - iv. Υπάρχοντα ESPs
 - v. Δημοσιευμένα ESPs
 - vi. Σύγκριση μεταξύ δημοσιευμένων ESP και του συστήματος μας
- IV. Υλοποίηση του Συστήματος
 - vii. Μηχανολογικές Μετατροπές
 - viii. Υπολογιστικό Υλικό
 - ix. Λογισμικό
- V. Αξιολόγηση σε Πραγματικό Περιβάλλον
- VI. Συμπεράσματα και Μελλοντική Εργασία
- VII. Παράρτημα
 - x. Πηγαίος Κώδικας Υπολογιστή
 - xi. Πηγαίος Κώδικας Μικροελεγκτή
 - xii. Πηγαίος Κώδικας Matlab

Ο συγγραφέας θα ήθελε να αφιερώσει αυτό το έγγραφο στους γονείς του και στην Μπάρλα Έλλη για την αμέριστη στήριξη και την υπομονή που έδειξαν για την ολοκλήρωση αυτού του μεταπτυχιακού.

Table of Contents

1.	Introduction	15
2.	Related work	17
2.1	Existing ESPs	17
2.1.1	Historical and Commercial background of the ESC... ..	17
2.2	Published ESPs.....	20
2.3	Scaled Implementations.....	29
2.4	Comparison between published ESPs and our system.....	32
2.5	Attainment to Academic Community.....	33
3.	Vehicle Dynamics and Stabilization Algorithm	34
3.1	Oversteer and Understeer	34
3.1.1	Abstractional Behaviour.....	34
3.1.2	Counteracting Oversteer and Understeer	37
3.2	Yaw rate control with individual wheel braking	39
3.2.1	Vehicle dynamics	39
3.2.2	Yaw rate stabilization algorithm	42
	Single gyroscope Electronic Stability Control Algorithm.....	43
	Analytic presentation of the algorithm.....	47
3.3	Single Accelerometer Electronic Stability Control	49
	Single Accelerometer ESC algorithm	50
3.3.1	Real Environment Evaluation for the Single Accelerometer ESC	51
4.	System's Implementation.....	52
4.1	Mechanical modifications	52
4.2	Computing hardware	57
4.2.1	Sensors	58
	Front and Rear Axle Accelerometers: ADXL311	58
	ADXL311 $\pm 2g$ interface with the microcontroller.....	60
	Central Accelerometer: ADXL213	60
	ADXL213 $\pm 1.2g$ interface with the microcontroller.....	62
	ADXL311 and ADXL213 bandwidth selection	65
	Observations, problems and possible improvements on the interface between the accelerometers and the microcontroller	67
	ADXL311	67
	ADXL213.....	68

Side by side comparison of ADXL311 and ADXL213	68
Steering Angle Estimation	69
Wheel angular velocity.....	71
Yaw rate estimation	77
Driver's commands	80
4.2.2 Actuators.....	82
4.2.3 Power Supply	83
4.2.4 Microcontroller: Schematics, PCB and Development Tools.....	86
Schematics and PCB	86
Central PCB for the microcontroller.....	87
Development Tools	93
Development Board.....	93
Development Software	94
Programming the device	95
4.3 Software at SBC	98
4.3.1 Single Board Computer; Ubuntu Linux.....	98
Third Party Software	100
4.3.2 Custom developed source code.....	100
A Brief outline	100
Daemon!	103
Serial port initialization, reading and writing.....	110
Evaluation of the information from the binary data.....	112
Auto generated experiments function.....	117
4.3.3 Stabilization routines	119
4.3.4 Scripts	121
4.4 Firmware at the Microcontroller	122
4.4.1 Main loop.....	123
4.4.2 Normal Routines.....	126
4.4.3 Interrupt Routines:	129
4.5 How to operate the system	135
5. Real Environment Evaluation	136
Real Environment Experiments	136
Bird-Eye-View.....	139
6. Conclusions and Future Work	143
6.1 Conclusions.....	143

6.2	Future work and potentials extensions	145
7.	Acknowledgements.....	146
8.	References	147
9.	Appendix.....	155
9.1	Host code	155
9.1.1	daemon.....	155
	async.cpp	155
	headers.h	169
	init_serial.c	170
	time_etc.cpp.....	171
	servos.h.....	172
	stabilization_routine1.c.....	172
	stabilization_routine2.c.....	174
	stabilization_routine3.c.....	176
9.1.2	Scripts	178
	compile.....	178
9.1.3	index.html.....	178
9.2	Microcontroller codes	179
9.2.1	ESC_new.c	179
9.2.2	adc_accelerometers.c.....	183
9.2.3	interrupt_routines.c.....	186
9.2.4	servos.c	192
9.2.5	header.h.....	192
9.3	MATALAB codes.....	194
9.3.1	plotdata	194
9.3.2	gwnies.....	197
9.3.3	Bird's eye view.....	197
9.4	Typical Log file output	201

This page was intentionally left blank.

This page was intentionally left blank.

1. Introduction

Electronic Stability Control (ESC) is a closed loop computer based system which helps the driver to maintain control of the vehicle and prevent skidding under highly demanding situations by applying individual wheel braking and/or readjusting the engine delivered torque. The driver can be modelled as a high gain system whom reactions are cursory and boorish and might worsen a situation of instability. Even an experienced driver in a panic situation might try to counteract the effect of oversteer (or understeer) in a rear wheel drive vehicle by applying the brakes, an action that will increase the violence of the effect. The loss of handling in such a scenario is likely to result to a fatal accident. Several studies since the wide mass introduction of ESC, in the year 1998, have showed the system's effectiveness ([1]). At least 40% of fatal accidents are triggered by skidding and the global installation of ESC could reduce skidding accidents by even 80% ([2]). The undisputed benefits from ESC led the European Union to launch a campaign called "Choose ESC" at the Bridgestone European Testing Ground, on 8th May 2007. The aim of this campaign was to spark people's awareness towards ESC and promote the active safety market in automotive industry with the installation of ESC on all vehicles in European ground. It is prominent that in year 2007, in Germany, Denmark, Austria, and Italy, as ESC system is standard fit for almost all brands. The goal of the "Choose ESC" campaign is to halve the road fatalities by 2010.

Extensive research towards Stability and Yaw Control has been conducted by several authors and different approaches have been proposed ([4], [5], [8]) with BOSCH GmbH being the pioneer and thereafter leader in ESC. The stabilization of the vehicle is accomplished by individual wheel braking ([7]), active steering ([5], [6]) and hybrid methods ([9]) combining the precedents along with dynamic engine's torque distribution. Challenging position on modelling about stability and desired path tracking on rally driving techniques has also been addressed by [10] and [11], where manoeuvres commonly used for high speed cornering, like pendulum turn and trail braking have been analyzed. Similar to previous systems ([10] and [11]) should be adapted by automotive manufacturers for their fast fleet cars as an optional cachet in order to increase vehicle's fun to drive side.

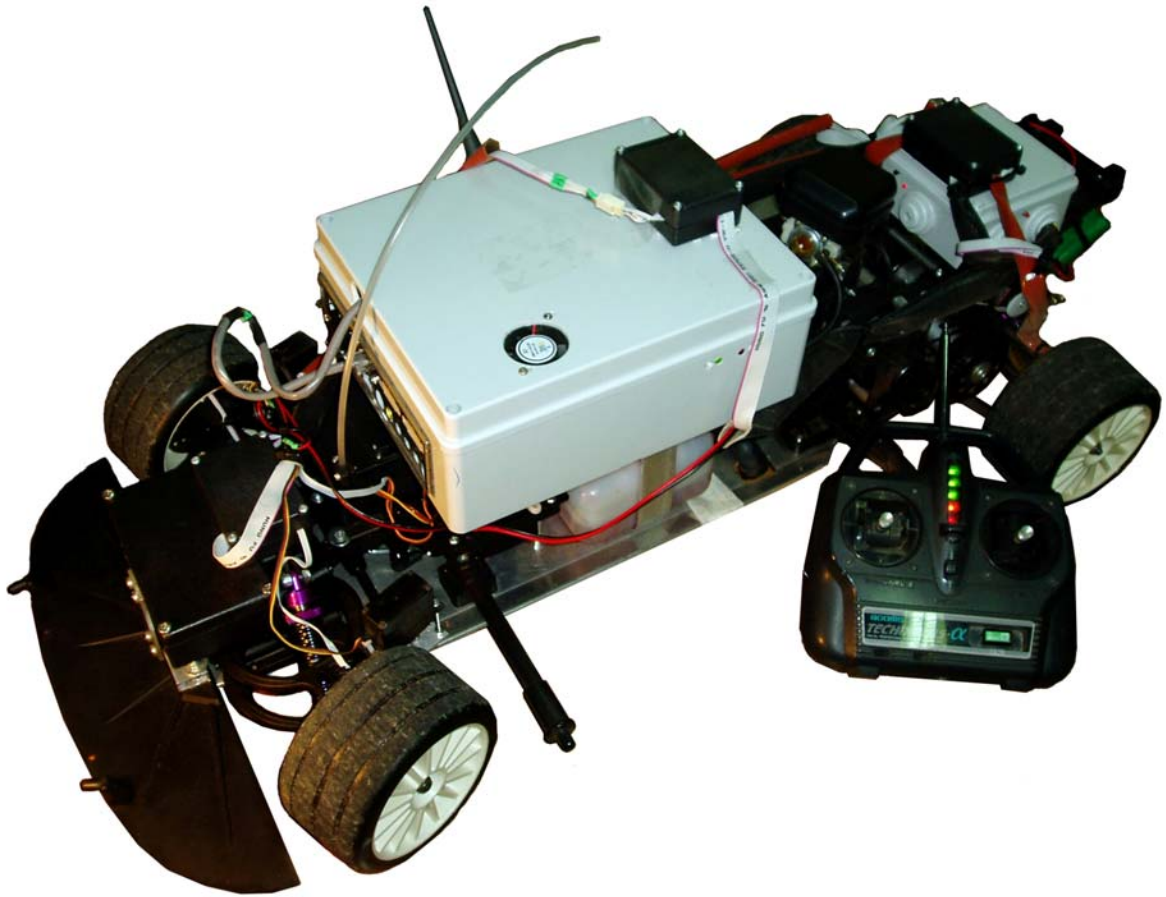


Fig. 1.1 Scaled Model Car.

ESC course along with fondness towards automobiles and high speed driving, challenged us to develop a 1:5 scaled model car to be used as test bed for experiments in vehicle dynamics and control (Fig. 1.1) ([58]). Similar work has been done by [3] and [13].

2. Related work

The benefits and solicitude of Electronic Stability Control towards safety and dynamics – control correspondingly and of course the wide commercial acceptance among vehicular manufactures has triggered an ESC research trend. The proposed approaches for control are many, regardless of the fact that the state and inputs for every vehicular system are similar. Yet the goal is always the same; stabilize the vehicle, even in situations where the driver worsens the instability due to his stampede reactions and driving errors (exaggerated steering). According to [65] a normal driver is not capable of estimating safe critical information, such as grip reserves on the tires or the friction coefficient.

Inputs for an ESC system are the driver commands (steering and throttle or brake command), velocity, engine delivered torque, brake torque on each wheel etc. In most cases the state of the system is the angular velocity (yaw rate) around the Z axis, the slip angle of the vehicle and in some cases the longitudinal or lateral acceleration at the center of gravity of the vehicle. An ESC system has to deal with unknown parameters and disturbances (the travelling vehicle itself and the environment affecting the vehicle) like the friction coefficient, road bank angle, side winds, unbalanced loading etc.

This chapter is dedicated to related work concerning ESC systems available for studying in the academic community. The different approaches proposed are divided into three main categories: those where the proposed stabilization schemes have never been implemented on a real vehicle, but are mainly simulated; the fortunate Research Institutes in close collaboration with the industry, for whom their research is based primarily on experimenting on real vehicles; and the smallest category of all, those whose research is based on scaled vehicles, which is the category we belong to. We shall outline all three categories and present some key stabilization schemes. **We must clarify that certain sentences inside “*quotes in italic fonts*” couldn’t be paraphrased and are collocated with small changes from the following cited publication.**

2.1 Existing ESPs

2.1.1 Historical and Commercial background of the ESC...

The first vehicle with installed ESC is the trailblazing pioneer of its times, Mercedes-Benz S-Class sedan in 1995. The installed ESC system was supplied by Bosch ([65]). The Mercedes-Benz S-Class also was the first vehicle that would introduce airbags as a Supplemental Restraint Systems (SRS) ([66]), ABS, Adaptive Cruise Control and many more innovations. Since 1995,

Bosch has produced tenth's of million systems worldwide, marketed as ESP - Electronic Stability Program. The main commercial ESP manufactures nowadays (May 2008) include ([67]):

- Robert Bosch GmbH ([68])
- TRW ([69])
- Continental Automotive Systems ([70])
- Delphi, USA ([71])
- Aisin Advics ([72])
- Nissin Kogyo ([73])
- Hitachi, Japan ([74])
- Mando Corporation ([75])
- Bendix Corporation ([76])
- WABCO ([77])

The market is governed by American, Japanese and German brands. The most famous of the above manufacturer is BOSCH which has the lion's share in ESC market, for both European and USA vehicles. Today, BOSCH ESC is installed in many cars like BMW X5, BMW X3, Cadillac Escalade, Chrysler Pacifica, Chevy Silverado, Dodge Durango, GMC Acadia and Toyota Camry ([78]) etc.

As manifested by a BOSCH report ([65]), the maximum side slip angles (the angle between the longitudinal axis of the vehicle and the actual direction of the vehicle; see Fig. 3.1 where α_{cg} denotes the slip angle for the bicycle model) where the vehicle is still steer able, are dependent on the road friction coefficient. As proclaimed by BOSCH ([65]) the slip angle on dry asphalt has typical peak values of about $\pm 12^\circ$ (Fig. 2.1), whereas on polished ice it is in the range of $\pm 2^\circ$. In all day traffic situations *"side slip angle values of typically not more than $\pm 2^\circ$ "* ([65]).

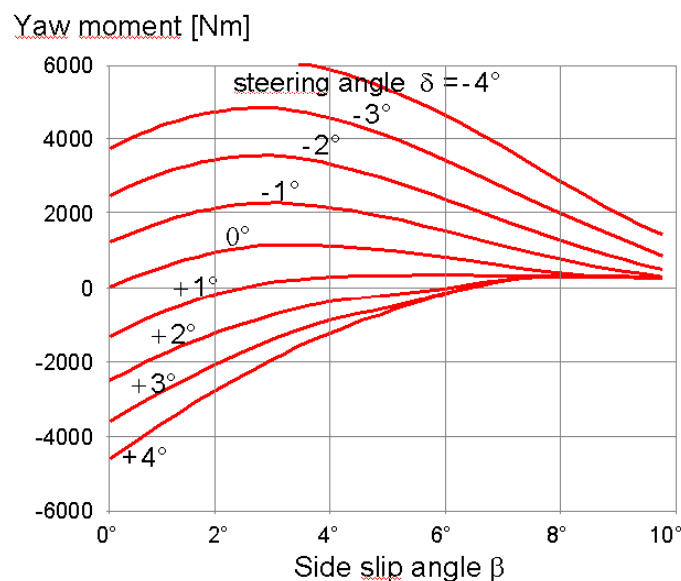


Fig. 2.1 Influence of side slip angle on yaw moment for different steering angles at high tire-road friction. Plot taken from [65].

The proposed ESP from BOSCH ([65]) is based on controlling the value of the slip at each wheel (towards the longitudinal axle of the wheel). They have used the “bicycle” model to build the desired behaviour of the vehicle (coming from the driver), where the model uses a linear relationship between the slip and the lateral forces on the tires and a state observer for the missing state variables. The “bicycle” model is used in cases where the car’s velocity is constant and the slip angle is small, whereas in full braking situation they use the observer to estimate the slip angle of the vehicle. The observer is a high complexity system which uses dynamic equations from the full vehicle (four wheel model), which was “*rearranged and discretized*” for becoming the model for a Kalman filter ([65]). The outputs of their controller are the nominal slip values for each tire.

The author of [65] declares that the vehicle slip angle is not accurate enough. Thereupon the vehicle’s dynamic controller also uses a linear estimation for the nominal yaw rate ($\dot{\psi}_{\text{NOM}}$) of the vehicle. Therefore, after the determination of the nominal values for the slip angle and the yaw rate, when the controller detects a deviation between the nominal and actual values above a dead zone, intervenes to generate the correct yaw moment to compensate the effect using the preinstalled electro hydraulic brake system. They have also used the concept “*suspicion of failure*” for the system, where in a likely event of failure the system increases the dead zone before it intervenes.

Automotive manufacturer Ford ([8]) has used the triad: Responsiveness, consistency, and smoothness, as guidelines principles for the development of their stability control system. The proposed ESP (ESP: Electronic Stability Program; similar abbreviation to the ESC) takes into account an accurate interpretation of the driver’s future intention for the vehicle motion for providing additional directional control (within physical limitations) as a driver’s aid. The intention for the driver can be extracted partially from angular position of the steering wheel and the turning direction.

The driver’s directional intention is determined by a desired yaw rate function based upon the simplified “bicycle” model (with appropriate limitations, linearization and constraints). The controller proposed by Ford will then use the above information to determine the need for intervention and try to regulate the difference between the target and the real response of the vehicle. Ford’s approach ([8]) the problem in two-stages; firstly the determination of the target yaw rate and secondly limiting the error of the target yaw rate based upon the lateral acceleration in a smooth manner. Under the assumptions that the longitudinal velocity is known via wheel speeds, the lateral tire forces can be approximated by the Dugoff model ([64]), they try to approximate the front and rear slip angle using small angle assumptions. After those assumptions and approximations they create a dynamic calculated yaw rate target based on the driver’s steering input and vehicle speed which is tuned firmly based on real experiments.

For the tuning of the “bicycle” model they also take into account some vehicle parameters like vehicle mass, center of gravity (CG) location with respect to the front and rear axle, rotational moment of inertia in the yaw axis, the front and rear cornering compliances, and the maximum vehicle’s yaw rate.

They have also built a theoretical limit for the yaw rate which they consider achievable at the current tire/road and through this limit they assert that it manages to obtain smooth transitions between the target’s yaw rates under the road’s friction coefficient transitions. In their implementation the theoretical road surface coefficient isn’t directly calculated.

A very nicely written paragraph from the authors of [8] has been cited, which is a general rule for every ESC system: While the target deviation from nominal response needs to be progressive in order not to provoke a harsh reaction from the driver, the target deviation also needs to be large enough to inform the driver that “*the cornering capability of the vehicle has been approached or adverse driving conditions exist*”. Without sufficient feedback reaching to the driver, the latter is alienated from critical information necessary for adapting to the driving conditions.

As in most ESP systems, the proposed stability control systems rely on the correct treatment of the brake pressure distribution and engine/drivetrain torque. For robust control of the vehicle’s sideslip angle they demand a controller insensitive to sensor dc offset, drift, or environmental conditions. So they propose a method for correcting the corroded sensors measurements. In retrospect, the methods for stabilizing, although weren’t cited very analytically, were similar with that proposed by BOSCH at [65]. Both ESC systems (BOSCH and Ford) are evaluated of course on real vehicles and are market products.

2.2 Published ESPs

Jürgen Ackerman is one of the top researchers in automotive control and is cited in almost every publication, since he was a pioneer of his times.

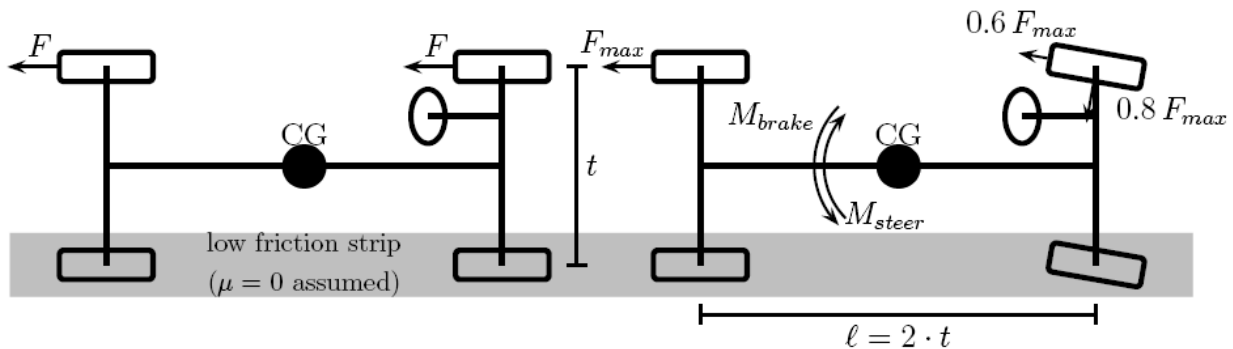


Fig. 2.2 Extreme μ -split braking (courtesy of [5]).

In the publication [5], the potential of active steering as an alternative or in combination with an active braking system is discussed under the following assumptions: the total force F_{max} which is delivered by the tires does not depend on the direction in which the force acts; the center of gravity (CG) is assumed to be midway between the front and rear axles of the vehicle and the wheelbase is two times longer than the track width (Fig. 2.2).

Ackerman et al ([5]) assert that with the help of steering we can easily compensate for torques caused by asymmetric braking (μ – split braking) through a corrective torque. At Fig. 2.2 which derives from [5], an extreme μ – split braking situation is presented for showing the advantages of active steering. The writers assert that the combination of braking and steering manages a better balance of torque and supports his assumption with a simple but illuminating example. Finally, they conclude that the use of active steering can influence efficiently a vehicle's yaw and roll dynamics. Although individual wheel braking based ESC systems have low cost, they have inherent drawbacks and limitations which can be compensated with the hybrid combination of active steering.

In another publication Ackerman et al ([4]) proposes a robust car steering method for yaw control. The main idea of this publication is a feedback control law that actuates on the steering of the vehicle and decouples the yaw mode from the lateral mode of the front axle. This paper is also based upon the dynamics of the “bicycle” model for the vehicle.

The implementation of the above system utilizes the yaw rate measured by a gyroscope and the steering wheel angle. The writer assumes a longitudinal mass distribution and shows that a compensator with unity transfer function and an integrating actuator is able to achieve the above mentioned decoupling between the front axle lateral acceleration and the second order system with state variables, the steering angle and the yaw rate. The novelty on this scheme is that the steering wheel input controls lateral acceleration through a simple first order transfer function with only a minor bondage on operating conditions. The writer proves his assertion that the unobservable mode is always stable. Also they propose a state observer adaptive at its gain for estimating the vehicle's lateral velocity under sensor noise and bias, road surface coefficient uncertainty, as well as vehicle model uncertainty.

The whole publication has more theoretical value than practical, since the main result of this paper is a robust compensator/actuator design for all cars and all operating conditions, for something that doesn't seem to have practical applicability and the conclusion is that the theory of observable / uncontrollable subspaces can be used to improve the dynamic behaviour of a car. Nowhere inside this publication is mentioned anything about the applicability of the proposed control law, neither about simulation or real test cases.

In a third publication ([6]) of the same author, we can see an improved version of the decoupling control law for active steering that was proposed in the previously mentioned paper. The vehicle model for car steering model is a simplified version of the single track model. To achieve his goal, the author introduces the sideslip angle at the front mass as a state variable. After some linearizations and assumptions he reaches to a relationship where he shows that the state variable he chose does not depend directly on the forces at the rear tires. He will try to decouple through the corresponding control law, the indirect coupling between the yaw rate and on the forces at the rear tires. The robust decoupling, as declared, is achieved by a feedback control law which has the following properties: the coupling from the yaw mode into the lateral acceleration has been removed. Thereby the steering transfer function is a first order function of the dimensionless variable he used earlier (the steering wheel input correlated with the transmission ratio of the steering gear).

The author proclaims that this law has disadvantages, such as yaw damping reduction at high velocity and no immediate reaction of the lateral acceleration after a step command at the steering input caused by the integration in the control law. Those drawbacks make the driver to feel that the vehicle is not reacting as swiftly as a conventional car. Therefore he proposes a modified control law where there is a direct connection between the inputs at the steering wheel to the front wheel steering angle.

The author of [6] signifies that the use of a dimensionless representation of a system's equations is beneficial through introduction of dimensionless similarity numbers. The most interesting part with respect to robust control theory is the potential of reduction of the system's uncertain parameters. Thereupon, the analysis and design of controlled systems are simplified. In the paper, the author utilizes Buckingham's theorem frequently, succeeding to put all uncertain parameters of the conventional car into one similarity number. Therefore, the novelty this system achieves in its dimensionless form is that all possible operating points can be represented by one single number; while the other numbers are invariant for a specific car due to construction (tire properties).

After some simulation on the properties of a real vehicle, the author concludes that a car designed to host the proposed controller, can be considered as a "*compromise*" between a typical car which has poor attenuation properties concerning the yaw disturbance and the decoupled car which "*exhibits poor yaw damping at higher velocities*". A car equipped with the above control system can return smoothly to the steady-state behaviour of the typical car.

Jong Hyeon Park and Chan Young Kim have proposed ([79]), in the "Vehicle System Dynamics" journal, a very well organized and highly applicative scheme to enhance vehicle lateral stability with a traction control system during cornering and lane changes. The specific writer will be cited again in this thesis, for a second stability scheme he has proposed, because his work is

transparent, easily comprehensible and unambiguous. The proposed scheme ([79]), controls wheel slip during cornering by varying the slip ratio as a function of the slip angle assuming the existence of traction control system acting on the engine throttle.

Jong Hyeon Park and Chan Young Kim, authors of [79] initiate their publication with an introduction on the Traction Control System (TCS) signifying that when a vehicle is accelerated on a slippery road, the resulting wheel slip reduces the traction force and increases the instability of the vehicle. A TCS is maximizing the longitudinal friction coefficient by maintaining an appropriate slip ratio. He also notifies that a vehicle in order to be able to turn without lateral slippage (maintain stability), two different forces are demanded for the desired behaviour of a vehicle driving straight and during cornering; longitudinal forces and lateral forces respectively.

The main idea of this publication is based on the typical characteristics of the tires, which show that the longitudinal friction coefficient decreases as the slip angle increases and also the increase on the slip angle shifts the peak of longitudinal friction coefficient ([79]). The authors of [79] also use the established ([14], [64]) statement that the peak value of the lateral friction coefficient occurs when there is absolute no slip ratio and increases as the slip angle increases. Thus, there is the potential of building a controller that will impose a value for the slip ratio during cornering lower than when driving straight, in order to achieve optimal balance of those two situations. Therefore the authors propose a proportional, integrating control method to control wheel slip for optimization of the longitudinal traction and the lateral force where the target slip ratio is given as a function of the wheel slip angle, estimated from vehicle's measurements (velocity, the yaw rate, and the steering angle). The whole control scheme was simulated for a front-wheel-driven vehicle, derived from an eight degree of freedom vehicle model (rotations of 4 wheels, yaw, roll, longitudinal and lateral motions). Concluding, the key idea is the reduction of the slip ratio of the wheels when the slip angle is large, in order to increase the lateral force producing this ways a Variable Slip – Ratio Control (VSRC) ([79]).

Although the control scheme looks robust in a control manner, it has a flaw in the author's (of this thesis) opinion. The control algorithm needs the lateral velocity in order to be implemented and the integration of lateral acceleration isn't the optimal method for estimating the lateral velocity. A state observer for this purpose should be constructed instead. Besides this flaw, the idea is very nice, the presentation explicative and the simulated results from the scheme look really promising.

Another interesting approach for yaw stabilization was proposed by Soheli Anwar ([60]). Anwar presents the theoretical development and experimental results of his method for yaw control based on generalized prediction. The idea is straightforward and derives directly from the title of the paper that the objective of the controller is to track the desired yaw rate by minimizing future yaw

rate errors based on the present errors between the actual and desired yaw rate. This writer, although his figures show a four wheel vehicle model, he has used the common “bicycle” model. The backbone of the process for the generalized predictive control algorithm is the utilization of Diophantine type discrete mathematical identities for obtaining a prediction of the future output of the plant which is to be controlled. The author of [60] asserts that his approach has been shown to be robust against modelling errors and external disturbances. In the presented work ([60]) an electromagnetic brake-by-wire based yaw control has been used, which energises selectively the EM brake at each wheel, for controlling the yaw moment of the vehicle. The author also declares in his publication that braking one wheel instead of more for generating the desired yaw rate torque is more effective. For example, as he states, the best action to counteract the effect of an oversteer condition is braking the front wheel is, while braking the rear wheel “*has been found*” to be a thriving action in an understeer condition. Finally, the author concludes his paper by citing the experimental results, which as he asserts show that the predictive controller can stabilize the vehicle even in an oversteer/understeer condition on a packed snow surface.

Electronic stability program along with steering intervention, based on steer-by-wire system, for heavy duty vehicles is a different approach to ESC proposed by P.Koleszar et al ([80]). The vehicle dynamic control is based on the combination of braking and steering actions using electronic or electro-hydraulic steering. Three actuators self equipped with electronic controllers have been used for their implementation: brake, steering system and the engine throttle. The vehicle model used is the all time classic “bicycle” model under simplifications like constant forward speed, neglected self aligning torque of the tires, tire lateral forces proportional to the sideslip angle of the tire etc. The authors of that paper declare that the measurable output signals of the model should be available on a real vehicle, therefore yaw rate and lateral acceleration of the vehicle are chosen as output.

P.Koleszar et al [80] assume a “*virtual*” model which is the reference for the real vehicle, using a 3 degree of freedom model. Because of the fact that the sideslip angle of the vehicle is necessary for controlling their model, they have used a simple proportional controller. When their controller energises, depending on the behaviour of the vehicle, the dynamic controller brakes individual wheel and steers the front axle for stabilization. The brake based controller has a defined “*dead band*” in order to diminish the sensitivity of the system and prevent untimely interaction, “*which results in a rather harsh intervention*”. The authors of the paper state that they have tested their stabilizing algorithm both, in a simulator and a real prototype vehicle and after, some experiments, like μ – split braking, lane change, braking in a turn and normal ESP situations, they derived to the conclusion that ESP combined with a steer by wire system can be very fruitful. Although they assert that their algorithm improves performance, something clear from the figures, they were very poor concerning the explication on the exact way their control system works.

An H_∞ controller for integrated side slip angle, roll and yaw controls have been proposed by Kazuya Kitajima et al ([82]). The H_∞ filter, also called minimax filter, is used in H_∞ control, where all the necessary information about the noise of the system is unavailable, minimizes the worst case estimation error and is used to design robust to unknown noise sources, control systems ([83]).

The authors of [82] have proposed two control algorithms, a feed – forward integration and H_∞ control algorithm. The proposed system is highly complicated and involves the coordination VDC (Vehicle Dynamic Control), active suspension and 4WS (four wheel steering). The idea of decoupling, also used by Ackerman at [4] and [6], is additionally used for the feed – forward integration method. On the other hand the H_∞ control algorithm tries to minimize a cost function under driver's steering and braking actions which are dealt as disturbances and simultaneously design the chassis control functions. The authors of this paper ([82]) have utilized a 4WS steering vehicle for their algorithms because as they claim, it reduces vehicle side slip angle and the phase difference between yaw rate and lateral acceleration while increasing lateral tire force, in the effort to unload the saturation from the other control inputs. In the feed – forward integration design, one vehicle control input (front and rear steering angles) is designated for each output (yaw rate and lateral velocity) and the others control inputs are treated as disturbances. For the evaluation of their algorithms, the authors developed a Matlab/Simulink simulator which realizes vehicle longitudinal, lateral, roll, yaw and each tire rotational motion. The concept of “*relative authority*” has been used for the feed – forward integrator where i.e. VDC affects the yaw rate relatively more than other vehicle states (i.e., side slip, roll angle, slip ratio). On the other hand, the H_∞ integration measures front steering angle and regards it as a disturbance input, whose effect is to be rejected by the control signal.

For the evaluation of the performance with and without vehicle yaw control, the authors put their implementation under test, similar to maneuvers used by NHTSA. Maneuvers like J-turn, where the vehicle is excited to roll and spin motions under sudden turns onto a sharp ramp and fishhook maneuvers, where the vehicle is provoked to two – wheel lift – off by a drastic turn and then turning back in the opposite direction ([82]). Simulations results of [82] showed that un-coordinated chassis control only improves slightly from the no-control case, while the H_∞ integrated case was found to perform significantly better and their control scheme can improve vehicle stability in most situations, under some imposed limitations of inputs.

Another paper incorporating an H_∞ controller has been proposed by the aforementioned Jong Hyeon Park for his paper “Wheel Slip Control in Traction Control System for Vehicle Stability” ([79]). The second proposed H_∞ controller for Yaw Moment Control With brakes ([59]) has been a guideline for the development of the ESC controller of this thesis test bed and will be cited again and explained extensively at “Yaw rate control with individual wheel braking” chapter. This paper

([59]) introduces a H_∞ yaw moment control scheme for improving vehicle stability especially in high speed driving using individual wheel braking. Similar to previous cited paper by the same author ([79]), steering angles are dealt as disturbances to the system and the controller tries to minimize the error between the desired and actual response of the vehicle.

The author of [59] uses a two degree of freedom linear model, representing lateral and yaw motions of a vehicle for the design of his H_∞ controller, while the vehicle dynamics used for simulation derives from an 8 DOF non linear model. The desired yaw moment is generated by applying braking torque to just one wheel, since as the authors claim, some wheels are more effective in generating the desired yaw moment than others, whereas braking only one wheel at a time, decelerates the vehicle less than braking more than one wheels. The authors assert that they evaluated the performance of the proposed controller through a series of computer simulations which were based on an 8 DOF vehicle model and nonlinear tire model and the results showed that their controller demonstrates efficient performance and robust stability.

The coordinated action of steering and individual wheel braking approach for ESC has also been proposed by [9]. The authors of this paper uphold the belief that the most effective approach is the coordinated and combined use of both methods for generating the corrective yaw moment. The approach they propose is based upon a revised model regulator which imposes coordinated steering and individual wheel braking for the better performance of the vehicle's stability around the Z axis. For the generation of their regulator, they have used the simplest model that can accommodate steering and individual wheel braking, the two track model which neglects roll and pitch motions. Their presentation is organized in three steps; the steering based ESC, the individual wheel braking ESC and the combined action of the previous two. The basic method of the combined controller is to regulate the error between the actual and desired yaw rate, through coordinated action for the actuators of steering and wheel braking. The authors claim that after realistic simulations and road tests of their model regulator, their approach was successful.

Continuing our related work presentation, we shall proceed with some “state of the art” ESC systems. An excellent presented paper which can also be used as tutorial for ESC and Control Allocation is proposed by Leo Laine and Johan Andreasson at [81]. This paper ([81]) combines control allocation with an optimization formulation for the implementation of on – line electronic stability control system for a conventional road vehicle. It is based on control allocation, methods which is used for systems, where the degrees of freedom controlled are less than the actuators (“over-actuated systems”). The driver manages the steering and the control allocator actuates on the engine and the four mechanical brakes to compensate any undesired behaviour. The whole idea concerning the control allocator is that it provides “*automatic redistribution*” when one actuator saturates in position or in rate, where its effectiveness is limited from this point and on. The

authors assert that the proposed algorithm is upgradeable and could be used in future vehicles where the actuators would be more than in the current implementation. Since we found their work very interesting we shall cite some parts of their publication. The desired path $r = [v_x \ v_y \ w_z]^T$ ([81]) is calculated by the motion controller and then the path controller tries to follow the desired path by correcting the global forces and yaw torque $v = [F_x \ F_y \ M_z]^T$. The controller, then distributes the correction over the actuators from v to u , where $u = [t_{ice} \ t_{mb1} \ t_{mb2} \ t_{mb3} \ t_{mb4}]^T$ and $\text{rank}(v) < \text{rank}(u)$. We have cited the control illustration from [81] in Fig. 2.3.

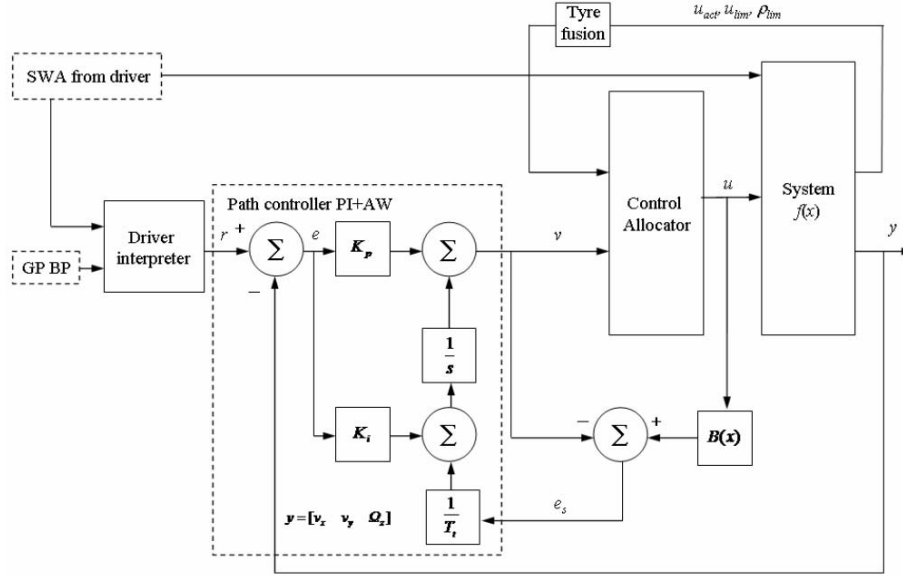


Fig. 2.3 Control Design with a focus on control law for vehicle motion and its path controller, a PI controller with Anti-Windup strategy. Abbreviations: Gas Pedal(GP), Brake Pedal (BP), and Steering Wheel Angle (SWA). Figure derives directly from [81].

The control is divided in two steps. First, the formulation of a control law for the net effort v , and second the design of the control allocator that maps virtual control input of the net to true control input, $v(t)$ mapping to $u(t)$. The necessary desired lateral velocity v_y and yaw rate w_z are predicted through a linear “bicycle” model. The whole implementation utilizes a path controller that is based on feedback linearization which transforms the nonlinear system into a linear one in order to have the ability to use linear techniques.

The next step in the control design for [81] was the creation of the control allocator, where the most important part is the selection of the control input set “ u ” from all possible combinations. The authors at this point, state that their proposed control allocation optimization is also suitable for fuel cell vehicles and hybrid electric vehicles. The ESC system was simulated on Matlab/Simulink, where the vehicle system models were implemented as S – functions and was tested against the proposed test procedure, Sine with Dwell, for ESC systems from the National Highway Traffic System Administrations, which did pass. After an excellent presentation, and

highly documented results the authors conclude that their ESC systems can be considered as a native characteristic of the motion control system for over-actuated road vehicles and that it is suitable for real time implementation.

A second publication utilizing control allocation for ESC using multi parametric nonlinear programming has been proposed by P. Tøndel and T. A. Johansen at [61]. This publication is nicely written and can also be used as tutorial paper for automotive tire dynamics (Fig. 2.4). The control scheme being introduced by the authors allows the control task to be divided into modules, so that the higher level designates the desired behaviour of the vehicle, while the lower level, the control allocator distributes the appropriate commands to the actuators so as to accomplish its goal. Since the control allocation is defined ([61]) as a non linear optimization situation, therefore computationally demanding, the piecewise linear approximate solution is computed off – line in order a realistic implementation in real time of the proposed system be achievable. Solving the problem in real time isn't applicable for time critical tasks.

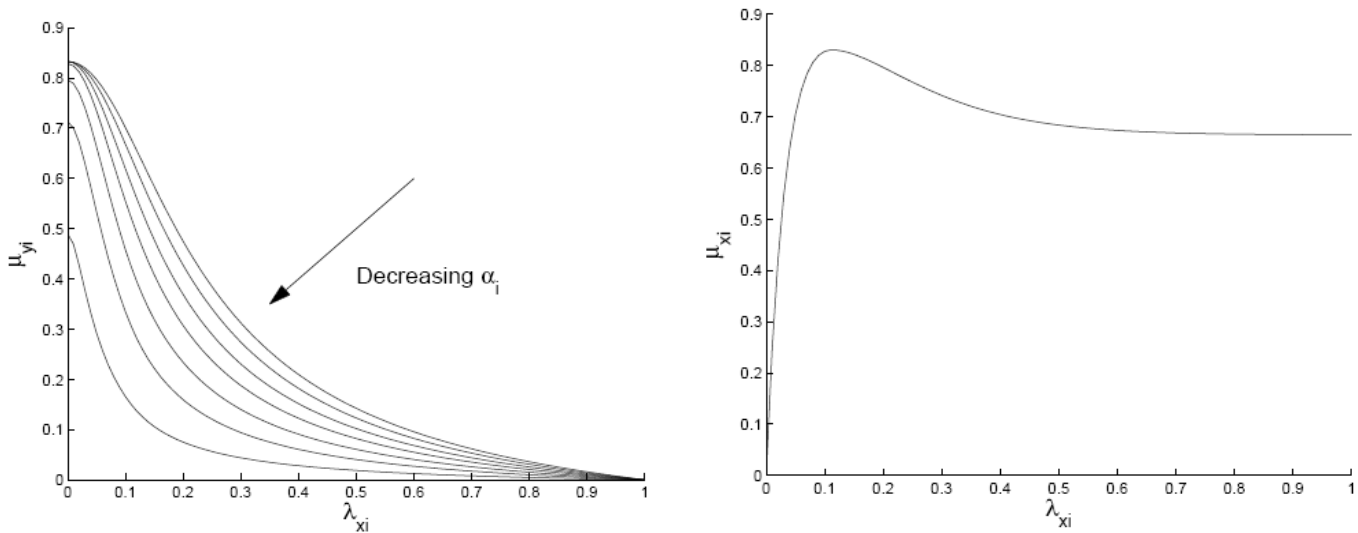


Fig. 2.4 Lateral friction coefficient as a function of the longitudinal wheel slip on the left and longitudinal friction coefficient as a function of the longitudinal wheel slip on the right. The figures are taken by [61].

Continuing the presentation of [61], a double track model is used by to describe the dynamics of the vehicle, while the vehicle is assumed to be ABS equipped with embedded wheel slip controller. This controller is assumed to apply the commands for the desired longitudinal wheel slips individually. The state vector of the proposed system consists of the longitudinal velocity, the yaw rate and the slip angle, where the first two are directly measured by sensors, while the slip angle is estimated through an observer. The control allocation scheme is a static mapping between the commanded moment around the Z axis of the vehicle, and the control inputs, where the dynamics of the vehicle are unconcerned. These dynamics are adjusted by a higher level control system.

The yaw stabilization is conducted through constraining the vehicle side slip angle and yaw rate with the upper/lower bound for its current state. The authors declare through their simulations that their control allocation algorithm works as desired and tracks the problem very accurately. The authors claim that on a case where the vehicle loses steerability, then under the application of the proposed controller, the manoeuvre remains stable.

An adaptive optimizing dynamic control allocation algorithm for yaw stabilization using brakes is proposed by J. Tjønnas and T. A. Johansen ([7]) and derives partially from author of the previous presented paper. This approach for ESC is similar to the previous cited in the sense that it utilizes modularization in the control design. It consists of (similarly to the previous paper) a high and low level module, allowing portability of the implementation, in the sense of software. The high level module deals with motion control, consisting of the yaw rate reference generator and an interface between the two levels and a low level module that deals with the actuation, through commanding individual brakes and the longitudinal clamping force, such that the vehicle meets the control objective (desired yaw rate). The adaptive dynamic control allocation algorithm is responsible for generating the appropriate commands delivered to brakes. A plane two-track model is then used for the stabilization design.

The actuator model depends on the unknown friction coefficient. Thus, the authors of [7] developed an adaptive law for estimating the maximal value of the friction coefficient. The allocation algorithm harnesses the parameter estimate and implements a certainty equivalent update law. The validation of the ESC scheme by [7] was conducted on the Daimler Chrysler's proprietary simulation environment CASCaDE (Computer Aided Simulation of Car, Driver and Environment) for MATLAB, which is a realistic nonlinear simulation environment ([7]). The authors declare that their adaptive algorithm is expedient for improved performance in terms of reference trajectory tracking and leans to be both more prosperous and robust compared to a non – adaptive one in terms of meeting the control objective ([7]).

2.3 Scaled Implementations

Building a scaled model of a real vehicle and/or its environment bridges the gap between solely simulation based implementations and real experiments. Development and evaluation of ESC systems in full scale vehicles confronts with safety and cost issues. It is difficult to experiment with the factory pre – installed sensors and actuators and even for a trial run, a racing track is necessary. Therefore some researchers are tempted by the potential of valid results while having diminished the above unflattering facts by building scaled automotive systems. It is most likely that we are the first building a self contained scaled automotive vehicle incorporating the necessary sensors,

actuators and computer systems for the implementation of a fully functional ESC system. However scaled test beds have been developed by some other researchers too.

A very interesting approach utilizing scaled test beds for automotive experiments has been introduced by [85]. Their main objective was to compare simulation data along with real data gathered from their custom developed scaled test bed, in order to estimate the potentials of ESC as vehicle properties change and particularly to specify vehicle's stability threshold at those changes. The implemented ESC system is simple; if the model exceeds some certain thresholds for stability, known from simulation, intervene to throttle and steering. MATLAB is once more used here for simulation purposes. The dynamic model they have used has 4 degrees of freedom, and the dynamic motions taken into consideration are longitudinal, lateral, yaw and roll. Their work is interesting, and has the potential for giving realistic results concerning vehicle dynamics and could be used for educative purposes too.

A radio controlled model car as vehicle dynamics testbed has been developed by Paul Yih ([3]) from Stanford University. The model car is 1:5 scale, gas powered and has an onboard single board computer which interfaces with the radio receiver, servo motors, and sensors through a microcontroller and digital input/output ports analog input/output ports. The software for their radio control model was developed with MATLAB Real-Time Workshop which generates C code from the appropriate developed Simulink model. This software is executed at the single board computer and is responsible for data acquisition and servo motor actuation. Although, this test bed was developed about dynamic inspection control, nothing was mentioned about dynamics or control for the vehicle.

Probably the most known and highly utilized facility for scaled testbeds, is the Illinois Roadway Simulator (IRS) ([13], [86]). This Roadway is an experimental platform for scaled vehicles that are simulated running on a road surface. The road surface moves relative to the vehicle, while those are held fixed through an articulated mechanism with respect to inertial space. The advantages, as claimed by the authors of [86] are many; cheap construction and design for new vehicles, economic in tests, which can be repeated multiple times with little or no cost etc. The roadway has the potentials to offer "*considerable sensing and actuation flexibility*" ([86]). The moving surface is a treadmill that can reach a top speed of 24km/h. Of course, the roadway simulator consists of all the necessary computing hardware and software for its operation which is managed via Wincon, a Windows-based control program running real-time code generated by the previous mentioned Real-Time Workshop of MATLAB.

The vehicle's dynamics are described by the "bicycle" model, while roll and pitch motions are neglected, allowing only two degrees of freedom, lateral position and yaw angle. The vehicle is fixed on a coordinate system to the center of gravity for the needs of modelling. The authors claim

that scaled vehicles showed similar dynamical behaviour to full scale vehicles. The roadway simulator looks like a perfect mean for real control emulation.

The Illinois Roadway simulator, is an excellent idea for experimenting. Many publications, like the previous two mentioned, and also [90], [91], [92], along with two Master Thesis have been derived from the results of IRS ([87], [89]). The Master Thesis of Sean N. Brennan ([88]), “Modelling and control issues associated with scaled vehicles”, is one of the best documented and written with unambiguous results the author of the thesis you are reading now has seen. The work his has done is a Masterpiece for the interested in building scaled vehicle and learn both basic and advanced dynamics in vehicles. It is nicely written, along with many valid citations and could be used as a tutorial on advanced modelling, dynamics, embedded systems etc.

In his thesis, he analyses the development of the roadway simulator the development of scaled vehicles, critical tasks the bottlenecks they found in the development of the roadway, and many experiments they did on fine tuning for the roadway and the scaled test beds, along with experiments on the dynamic behaviour of the vehicles ([91], [13]). In the “Driver Assisted Yaw Rate Control” publication ([91]) written by Mr. Brennan, he has proposed a yaw rate control method, based on model reference control. Mr. Brennan has constructed a yaw rate reference and tries to minimize the error between the reference and the plant, where the plant is the real vehicle. He uses a proportional – differential controller that acts on the rear wheels steering angle (4WS test bed). It is a nicely written work utilizing the benefits from the roadway simulator.

A 1:10 radio control car has been used by William E. Travis et al ([87]) as test bed, for validation of simulated experiments. The authors have modified the vehicle, so they can change the properties of the vehicle like center of gravity, spring stiffness and roll center, in order to be able to view the changes into the dynamic behaviour of the vehicle. The vehicle is equipped with an inertial measurement unit, consisting of a GPS, two gyroscopes and a two – axis gyroscope. The authors claim that the simulation followed the dynamic behaviour of the scaled car, in a real good manner. They conclude their publication by declaring the connection between experimentally observed dynamics with simulation results for full sized vehicles and claim that scaled vehicles can be profitable used for some forms of rollover research.

A 1:10 radio control vehicle has also been by R. Holve and P. Protzel ([93]) used for his reverse parking of a model car with the use of Fuzzy Control. Their fuzzy controller lacks any kinematics or modelling for the vehicle and the algorithm is very simple, but as the authors claim, it works nicely. Another very simple implementation utilizing the use scaled models for educational purposes, as the authors assert, is written by Manuel C.G. Silva, Mário L.O.S. Mateus & João M.S. Cruz .Their scaled model doesn’t do anything but is simple has an odometer and logs the data to a computer through a commercial software. A third scaled vehicle for traction control

study, was the submitted for Master of Science Thesis by Mark A. Morton ([95]). His thesis covers the design of platform and car and the application of traction control system on the vehicle (1:10 scaled car) and on the platform (Lego build treadmill). He cites the result from experiments while claiming that he managed a small error between mathematical simulations and physical tests.

In his master thesis, Mr. Richard D. Henry ([96]) developed an automatic ultrasonic headway control for a scaled car. His thesis was about the development a platform for rapid prototyping of systems similar to adaptive cruise control. A scaled test – bed was also built by Seung kook Jun and Daniel O. Gott ([97]). The scaled Test Bed is a radio control truck with a PC/104 based computer and various embedded sensor - and actuator. The goal of their work was to build a test – bed for studying several concepts like instrumentality for complex robotic controlled by human, stoutness of the control in the presence of varying grades of communication and “*multi-user shared teleoperation*”. Their test bed, as the authors assert, can also be used for studying human computer interaction, examining issues related to ground traffic control and issues related to drive-by-wire system. The system is based on the XPC target toolbox from Mathworks (Matlab). Nothing is mentioned about a control application in the publication.

2.4 Comparison between published ESPs and our system

Apart from the analytically presented publications towards ESC, there are hundreds different approaches for vehicle modelling, reference generation, safety margins and simulation – emulation of the different proposed vehicle stability control schemes. We can clearly notice that with the evolution of technology and the wide availability of controllers in vehicles they can execute computationally demanding algorithms in real time. If we look at origins of the research being conducted for ESC and its evolution to present, we can definitely see the exponential growth in control complexity. The origins of the research extend from the linear “bicycle” model of 1995, to the 8 degree of freedom nonlinear complicated models, employing nonlinear curves for the tire modelling, embedded in a 32 – bit ESC controller.

There are hundreds of different publications, under the appellation Electronic Stability Control, Yaw Control, Electronic Stability Program, Vehicle Dynamics Control etc. For most of them the objective is to stabilize the vehicle around the Z – axis and diminish the effects of oversteer or understeer. The control methods proposed are based on the same principles: a gyroscope and an accelerometer at the Center of Gravity, a reference for Yaw Rate and Slip angle, where the latter is evaluated from a state observer in industrial applications, like those in BOSCH ([12], [65], [78]) and Ford ([8]). Key points from the previous two manufactures, like fine tuning of their

algorithms and the construction of parts like the state observer for estimating the slip angle, are not published. But most published ESPs (see the section “Published ESPs”) do have a detailed description of their system’s operation.

Our ESP system (see section “Vehicle Dynamics and Stabilization Algorithm”) is governed by similar principles of those presented in the literature. We utilize only one gyroscope without a linear accelerometer, and by using a function of the velocity of the vehicle and the steering angle and after some experimenting we used a yaw reference function from [62], chapter 8 (p. 230). Our ESP is similar to [59], something clearly mentioned in the corresponding section. Our ESP is using a simple and straightforward method, using only a single gyroscope, which proved in real life experiments.

We know that our linearized ESP system isn’t the state of the art for controlling such complex system, but did manage to prove the usefulness of the vehicle testbed and the reason it was build for.

2.5 Attainment to Academic Community

The attainment to academic community for this thesis is clear. It is a complete construction manual for building an open source, test bed for automotive experiments, along with the evaluation of an ESP algorithm. It provides a description for the vehicle for a simple “stabilization” scheme and presents key literature for those who want to start studying in vehicle ESC.

This thesis is a tutorial on building from zero a very low cost test platform, illuminating key points and key aspects for the future developer, giving a detailed guidance for what he/she should be careful.

The most important part of this thesis is not the ESP algorithm proposed, but the fact that every software used is free and its location is clearly cited in the following chapters. We have used the most economic hardware available and none of the hardware used is industrialised or requires any specific licence. Everything inside this thesis could be bought from local stores, and all the software is available at the Internet. So we wish good luck to the potential researcher who plans to develop a similar test bed and he/she should read this document first, since it would save him/her quite a lot of time. It has citations everywhere needed, is documented and leaves no wonders or speculation for any subject.

3. Vehicle Dynamics and Stabilization Algorithm

3.1 Oversteer and Understeer

3.1.1 Abstractional Behaviour

The single track model will be used for a reduced complexity dynamical behaviour analysis for an automotive vehicle that is derived from the mathematical model considered by [14] for steady state cornering. The model takes into account tractive and inertial forces around the yaw axis, and neglects roll and pitch motion. Nevertheless, the complete dynamics of a real vehicle are highly non – linear and difficult to control. Interesting approaches have been proposed for non – linear systems with measurable state by [16], where an unknown non – linear system is controlled, with the usage of Recurrent High Order Neural Networks. The geometry of our single track model is shown at Fig. 3.1. The dynamics equations are given by 3.1, 3.2, 3.3 and 3.4:

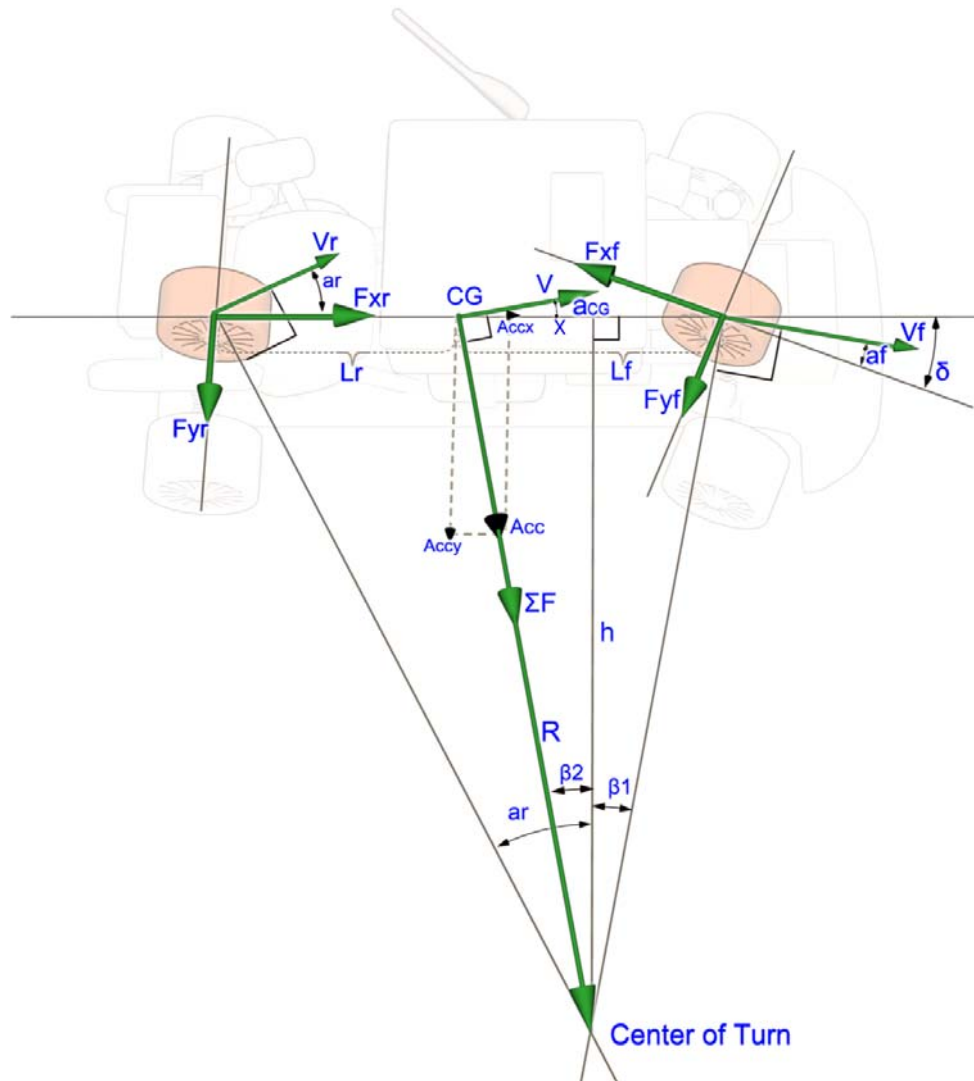


Fig. 3.1 Single track vehicle Model.

Applying Newton's Second Law in the lateral direction of the Vehicle, we can derive to the necessary dynamics equations. For a vehicle travelling with a speed V , the sum of the lateral forces (ΣF_L) originating from the tires acting on the vehicle are equal to the Centripetal Force ([14]).

$$\sum F_L = F_{Lf} + F_{Lr} = M \frac{V^2}{R} \quad (3.1)$$

Where M the mass of the vehicle, R : radius of turn, F_{Lr} and F_{Lf} the sum of the lateral forces (vertical from each wheel's direction of travel) acting on the rear and front axle correspondingly. Applying Newton's second law around the Center of Gravity (CG), if we consider the vehicle to be in an equilibrium moment ([14]):

$$F_{Lf} \cdot l_f - F_{Lr} \cdot l_r = 0 \quad (3.2)$$

Where l_f and l_r are the distance of the front and rear axle from the CG correspondingly. If we substitute equation 3.2, back to 3.1:

$$M \cdot \frac{V^2}{R} = F_{Lr} \cdot \left(\frac{l_r}{l_f} + 1\right) = F_{Lr} \cdot \left(\frac{l_r + l_f}{l_f}\right) = F_{Lr} \cdot \left(\frac{L}{l_f}\right) \quad (3.3)$$

$$\left. \begin{aligned} F_{Lr} &= M \cdot \left(\frac{l_f}{L}\right) \cdot \frac{V^2}{R} \\ F_{Lf} &= M \cdot \left(\frac{l_r}{L}\right) \cdot \frac{V^2}{R} \end{aligned} \right\} \quad (3.4)$$

Finally through Fig. 3.1(F_{xf} is considered positive), Gillespie ([14]) concludes:

$$F_{Lf} = F_{yf} \cdot \cos(a_f + \delta) + F_{xf} \cdot \sin(a_f + \delta) = M \cdot \left(\frac{l_r}{L}\right) \cdot \frac{V^2}{R} \quad (3.5)$$

$$F_{Lr} = F_{yr} \cdot \cos(a_r) + F_{xr} \cdot \sin(a_r) = M \cdot \left(\frac{l_f}{L}\right) \cdot \frac{V^2}{R} \quad (3.6)$$

Key parameters and symbols are defined in Table 3-1.

M	Mass of the Vehicle
V	Forward speed
l_f, l_r	Distance of front, rear axle from the center of gravity
L	Wheelbase ($L = l_f + l_r$)
F_{Lf}, F_{Lr}	Sum of lateral forces for front and rear axle.
F_{yf}, F_{yr}	Cornering forces: front, rear axle
F_{xf}, F_{xr}	Tractive forces: front, rear axle
a_f, a_r	Slip angles: front, rear axle
δ	Steering Angle
C_{af}, C_{ar}	Cornering stiffness: front, rear axle
R	Radius of Turn

Table 3-1 Variables and parameters for the Single Track Model.

Cornering forces F_{yf}, F_{yr} , are linear w.r.t. slip angle at low slip angles according to [14] and [64]. The relationship between forces and slip angle is:

$$\left. \begin{aligned} F_{yr} &= C_{ar} \cdot a_r \\ F_{yf} &= C_{af} \cdot a_f \end{aligned} \right\} \quad (3.7)$$

For angles less than 20 degrees, the error for a first order approximation for sine and cosine ($\sin a \approx a$ and $\cos a \approx 1$), is less than 6.5%. From fig.3, assuming small angles, we can derive the following approximations:

$$\left. \begin{aligned} \tan a_r &= \frac{l_r + x}{h} \xrightarrow{a_r : \text{small}} a_r = \frac{l_r + x}{h} \\ \tan(\beta_1) &= \frac{l_f - x}{h} \xrightarrow[\beta_1 = \delta - a_f]{a_f : \text{small}} \delta - a_f = \frac{l_f - x}{h} \\ \cos(\beta_2) &= \frac{h}{R} \xrightarrow[\beta_2 = a_r - \delta + a_f \approx 0]{\beta_2 : \text{small}} \frac{h}{R} = 1 \end{aligned} \right\} \quad (3.8)$$

Where x is distance from CG to the projection of Center of Turn on the longitudinal axis of the vehicle. Adding the first two and substituting the 3rd, yields:

$$a_r + \delta - a_f = \frac{l_r + l_f}{h} \xrightarrow{h=R} \delta = \frac{L}{R} + a_f - a_r \quad (3.9)$$

If we substitute equations 3.7 into 3.5 and 3.6, we get:

$$C_{af} \cdot a_f \cdot \cos(a_f + \delta) + F_{xf} \cdot \sin(a_f + \delta) = M \cdot \left(\frac{l_r}{L}\right) \cdot \frac{V^2}{R} \quad (3.10)$$

$$C_{ar} \cdot a_r \cdot \cos(a_r) + F_{xr} \cdot \sin(a_r) = M \cdot \left(\frac{l_f}{L}\right) \cdot \frac{V^2}{R} \quad (3.11)$$

Again, assuming small angles:

$$\cos(a_f + \delta) = \cos a_f \cos \delta - \sin a_f \sin \delta \approx 1 - a_f \cdot \delta$$

$$\sin(a_f + \delta) = \cos a_f \sin \delta + \sin a_f \cos \delta \approx \delta + a_f$$

Using the above relationships into 3.10 and 3.11:

$$C_{af} \cdot a_f \cdot (1 - a_f \cdot \delta) + F_{xf} \cdot (a_f + \delta) = M \cdot \left(\frac{l_r}{L}\right) \cdot \frac{V^2}{R} \rightarrow$$

$$C_{af} \cdot a_f - C_{af} \cdot a_f^2 \cdot \delta + F_{xf} \cdot a_f + F_{xf} \cdot \delta = M \cdot \left(\frac{l_r}{L}\right) \cdot \frac{V^2}{R}$$

$$\xrightarrow[\text{if } a_f, \delta = 20^\circ \rightarrow a_f^2 \cdot \delta = 0.04]{a_f^2 \cdot \delta \approx 0}$$

$$C_{af} \cdot a_f + F_{xf} \cdot a_f + F_{xf} \cdot \delta = M \cdot \left(\frac{l_r}{L}\right) \cdot \frac{V^2}{R} \quad (3.12)$$

And

$$C_{ar} \cdot a_r \cdot 1 + F_{xr} \cdot a_r = M \cdot \left(\frac{l_f}{L}\right) \cdot \frac{V^2}{R} \quad (3.13)$$

Solving 3.12 and 3.13 w.r.t. a_f and a_r yields:

$$a_f = \frac{M \cdot l_r \cdot V^2}{C_{af} \cdot (1 + \frac{F_{xf}}{C_{af}}) \cdot R \cdot L} - \frac{F_{xf} \cdot \delta}{C_{af} \cdot (1 + \frac{F_{xf}}{C_{af}})} \quad (3.14)$$

$$a_r = \frac{M \cdot l_f \cdot V^2}{C_{ar} \cdot (1 + \frac{F_{xr}}{C_{ar}}) \cdot R \cdot L} \quad (3.15)$$

If we substitute equations 3.14, 3.15 into 3.9, and solve w.r.t. R we get:

$$\begin{aligned} \frac{\delta \cdot \left(1 + 2 \cdot \frac{F_{xf}}{C_{af}}\right)}{\left(1 + \frac{F_{xf}}{C_{af}}\right)} &= \frac{1}{R} \cdot \left(L + \frac{M \cdot l_r \cdot V^2}{C_{af} \cdot (1 + \frac{F_{xf}}{C_{af}}) \cdot L} - \frac{M \cdot l_f \cdot V^2}{C_{ar} \cdot (1 + \frac{F_{xr}}{C_{ar}}) \cdot L} \right) \Leftrightarrow \\ R &= A \cdot \left(L + \left(\frac{V^2 \cdot M}{L} \right) \cdot (B - C) \right) \\ A &= \frac{1}{\delta} \cdot \frac{\left(1 + \frac{F_{xf}}{C_{af}}\right)}{\left(1 + 2 \cdot \frac{F_{xf}}{C_{af}}\right)} \quad B = \left(\frac{l_r}{C_{af} \cdot (1 + \frac{F_{xf}}{C_{af}})} \right) \quad C = \left(\frac{l_f}{C_{ar} \cdot (1 + \frac{F_{xr}}{C_{ar}})} \right) \end{aligned} \quad (3.16)$$

If we consider a steady steering angle δ , we can analyze the behaviour of the radius in a turn for each term (A, B and C) of the above equation.

3.1.2 Counteracting Oversteer and Understeer

When a vehicle oversteers tends to narrow the radius of turn. On the other hand, when a vehicle understeers, tends to widen the radius (Fig. 3.2). Applying the right inputs when the effect of oversteer or understeer is detected, the ESC system, can counteract the undesired effects within the physical limits of the system. With a closer look at equation 3.16 we can determine which would be the appropriate inputs for each case.

Oversteer:

- In order to counteract oversteer; we have to increase the radius of turn:
 - Reduction on the magnitude of a positive F_{xf} , or better a negative F_{xf} , increases both A and B terms term, consequently R . This is translated as releasing throttle on a FWD vehicle or applying brakes at the front axle.
 - Increase on the magnitude of a positive F_{xr} reduces C terms, therefore increases R . This accounts for applying throttle on RWD vehicle.

Understeer: In order to counteract understeer, we have to decrease radius of turn:

- Increase on the magnitude of a positive F_{xf} reduces both the A and B terms, therefore reduction on R . This is translated as applying throttle on FWD vehicle.
- Reduction on the magnitude of a positive F_{xr} , or better a negative F_{xf} , results to an increased C term, therefore reduction on R . This accounts for releasing throttle on a RWD vehicle, or applying brakes at the rear axle (tailbrake).

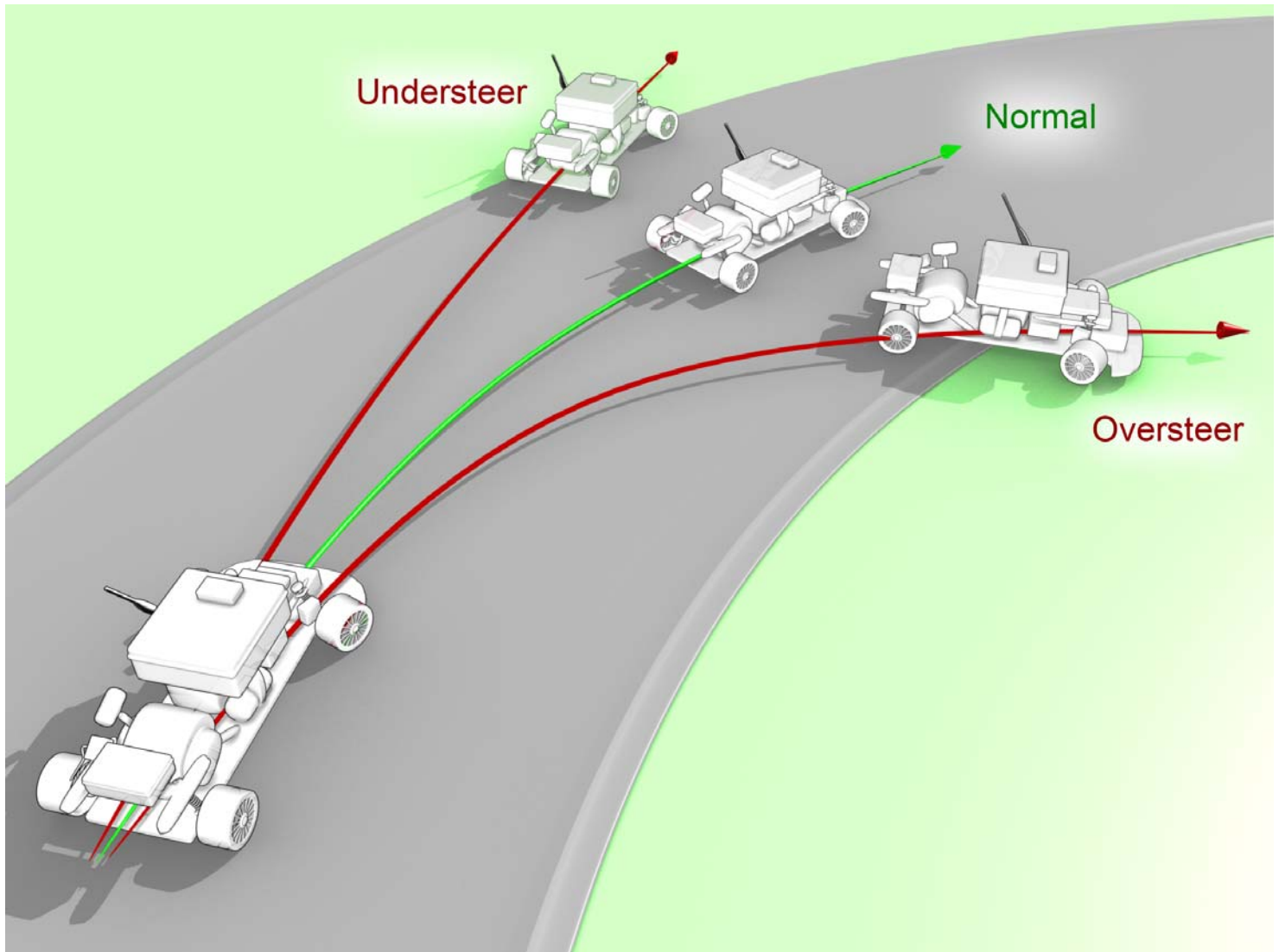


Fig. 3.2 Oversteer, Understeer and Neutral Steer.

Although the above analysis is simple, gives a clear case of what would be the response of applying throttle or brake at the radius of turn of a vehicle. An experienced driver counteracts impulsively with the above techniques in order to control the vehicle.

3.2 Yaw rate control with individual wheel braking

Yaw rate control systems usually apply individual wheel braking and/or readjusting the engine delivered torque, where some systems also use active steering or hybrid methods combining both: differential braking, torque distribution and active steering. So the best approach which would limit the impact of the ESC system at the original velocity of the vehicle would be to apply brakes on the most effective wheel at each time. This wouldn't slow down the vehicle very much and would be as much effective, as those mentioned above at the "Oversteer and Understeer Effects Section".

A very nice and readable approach on the above stabilization scheme (differential wheel braking) has been presented by [59] and most of the dynamical equations and mathematical formulas that will be used in this section have derived from this paper. The dynamical model of the vehicle is based upon the dynamics of the double track model, is real close to a real vehicle, and is quite popular to automotive researchers, for example the [60] and [61].

The double track model is a two degree of freedom linear vehicle model constituting of the lateral and yaw motion of the vehicle.

3.2.1 Vehicle dynamics

Using the longitudinal and lateral axis of the vehicle at Fig. 3.3 as coordinat axis we derive to following dynamic equations, which are taken from [59].

Applying Newton law with respect to the longitudinal axis (X axis) of the vehicle:

$$M \cdot (\dot{V}_X - V_Y \cdot \omega) = F_{X1} \cdot \cos \delta_1 + F_{X2} \cdot \cos \delta_2 - F_{Y1} \cdot \sin \delta_1 - F_{Y2} \cdot \sin \delta_2 + F_{X3} + F_{X4} \quad (3.17)$$

Applying Newton law with respect the lateral axis (Y axis) of the vehicle:

$$M \cdot (\dot{V}_Y + V_X \cdot \omega) = F_{X1} \cdot \sin \delta_1 + F_{X2} \cdot \sin \delta_2 + F_{Y1} \cdot \cos \delta_1 + F_{Y2} \cdot \cos \delta_2 + F_{Y3} + F_{Y4} \quad (3.18)$$

Applying Newton law with respect the vertical axis (Z axis) of the vehicle around the center of gravity:

$$I_Z \cdot \dot{\omega} = t_f \cdot (F_{X1} \cdot \cos \delta_1 - F_{X2} \cdot \cos \delta_2 - F_{Y1} \cdot \sin \delta_1 + F_{Y2} \cdot \sin \delta_2) + t_r \cdot (F_{X3} - F_{X4}) + L_f \cdot (F_{X1} \cdot \sin \delta_1 + F_{X2} \cdot \sin \delta_2 + F_{Y1} \cdot \cos \delta_1 + F_{Y2} \cdot \cos \delta_2) - L_r \cdot (F_{Y3} + F_{Y4}) \quad (3.19)$$

Where:

- $F_{X1} - F_{X4}$: Longitudinal forces acting on the tires.
- V_X : Longitudinal velocity along the X axis
- $F_{Y1} - F_{Y4}$: Lateral forces acting on the tires.
- V_Y : Lateral velocity along the Y axis
- ω : Angular rate around Z axis.
- $\dot{\omega}$: Angular acceleration around Z axis.

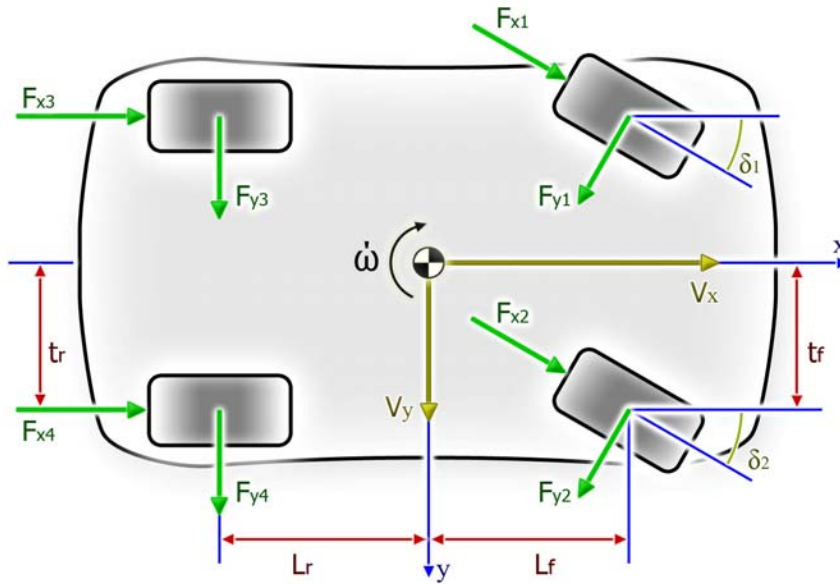


Fig. 3.3 Double track vehicle model.

All the parameters and symbols that will be used for the double track model are centralized in Table 3-2 ([59]).

Parameter		Unit
M	Mass of the Vehicle	kg
I_z	Moment of Inertia around the Z axis center of gravity (CG)	kgm^2
l_f, l_r	Distance of front, rear axle from CG	m
L	Wheelbase ($L = l_f + l_r$)	m
$F_{X1} - F_{X4}$	Longitudinal forces acting on the tires.	N
$F_{Y1} - F_{Y4}$	Lateral forces acting on the tires.	N
V_X	Longitudinal velocity along the X axis	m/sec
V_Y	Lateral velocity along the Y axis	m/sec
a_1, a_2, a_3, a_4	Slip angles: front left, front right, rear left, rear right	rad
β	Slip angle at CG	rad
δ_1, δ_2	Steering angle: Front left wheel, Front right wheel	rad
C_{af}, C_{ar}	Cornering stiffness: front, rear axle	N/rad
t_f, t_r	Half of the space between wheels: front and rear axle	m
ω	Angular rate around Z Axis at CG	rad/sec
$\dot{\omega}$	Angular acceleration around Z Axis at CG	rad/sec^2

Table 3-2 Parameters for the double track model.

Wheel slip angles for the front left, front right, back left and back right are expressed respectively by ([59]):

$$\begin{aligned}
 a_1 &= \delta_1 - \tan^{-1} \left(\frac{V_Y + L_f \cdot \omega}{V_X + t_f \cdot \omega} \right), \\
 a_2 &= \delta_2 - \tan^{-1} \left(\frac{V_Y + L_f \cdot \omega}{V_X - t_f \cdot \omega} \right), \\
 a_3 &= -\tan^{-1} \left(\frac{V_Y - L_r \cdot \omega}{V_X + t_r \cdot \omega} \right), \\
 a_4 &= -\tan^{-1} \left(\frac{V_Y - L_r \cdot \omega}{V_X - t_r \cdot \omega} \right)
 \end{aligned} \tag{3.20}$$

The above relations are directly taken from [59], but are quite popular to other publications concerning vehicle dynamics. Proof for the slip angles can be found at [87] in pages 46 to 48. The slip angle of the vehicle at the center of gravity is defined as the angle between the vector of the actual velocity of the vehicle and the longitudinal axis of the vehicle (X Axis).

$$\beta = \tan^{-1} \left(\frac{V_Y}{V_X} \right) \tag{3.21}$$

The authors of [59], under some assumptions derive to a relationship really helpful to us. The assumptions are:

- The longitudinal and lateral tire forces F_X and F_Y , although they are highly non – linear functions of the slip angle, slip ratio, velocity of the tires etc, are simulated with the Dugoff tire model ([59],[64]), same as at the “Abstractional Behaviour” section, as linear functions of the slip angle. The relationship between forces and slip angle is ([14],[59],[64]): $F_{Yi} = C_{ai} \cdot a_i \quad i = 1, 2, 3, 4$ (3.22)

$$V_X \gg V_Y \rightarrow \beta \approx V_Y / V_X \tag{3.23}$$

$$\delta_1 \approx \delta_2 = \delta_f \ll 1$$

$$t_f \approx t_r = t$$

$$V_X \gg t_f \cdot \omega \text{ and } V_X \gg t_r \cdot \omega$$

- Assuming quasi – static moment balance at the wheels about the rotational centers:

$$F_{Xi} = T_{Bi} \cdot R_W \quad i = 1, 2, 3, 4, \tag{3.24}$$

Where T_{Bi} is the braking torque and R_W is the effective radius of each wheel.

Under the above assumptions and combining the relationships 3.17 – 3.24, the author of [59], reaches to the following state representation of the simplified vehicle model, with state variables, the yaw rate ω and the slip angle .

$$\begin{bmatrix} \dot{\beta} \\ \dot{\omega} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \cdot \begin{bmatrix} \beta \\ \omega \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 & 0 \\ b_{21} & b_{22} & b_{23} & b_{24} \end{bmatrix} \cdot \begin{bmatrix} T_{b1} \\ T_{b2} \\ T_{b3} \\ T_{b4} \end{bmatrix} + \begin{bmatrix} c_1 \\ c_2 \end{bmatrix} \cdot \delta_f$$

Where

$$\begin{aligned} a_{11} &= \frac{-C_{af} + C_{ar}}{M \cdot V_x} & a_{12} &= -1 + \frac{C_{ar} \cdot L_r - C_{af} \cdot L_f}{M \cdot V_x^2} \\ a_{21} &= \frac{C_{ar} \cdot L_r - C_{af} \cdot L_f}{I_z} & a_{22} &= -\frac{C_{ar} \cdot L_r^2 + C_{af} \cdot L_f^2}{I_z \cdot V_x} \\ b_{21} = b_{23} &= -\frac{t}{I_z \cdot R_w} & b_{22} = b_{24} &= \frac{t}{I_z \cdot R_w} \\ c_1 &= \frac{C_{af}}{M \cdot V_x} & c_2 &= \frac{C_{af} \cdot L_f}{I_z} \end{aligned} \quad (3.25)$$

The above relationships come from [59]. The control scheme we decided to use for stabilizing the vehicle is based upon the relationship 3.25. It is simple but straightforward and proved quite effective. It is presented in the next section.

3.2.2 Yaw rate stabilization algorithm

The yaw rate of the vehicle can be affected by the brake torque on the wheels as denoted by the equation 3.25. Nevertheless, all wheels don't have the same effect in generating torque. As proved in section "Counteracting Oversteer and Understeer" and the relationship 3.16, applying brakes at the front axles reduces oversteer, which is translated as widening the radius of turn of a vehicle, which means that decreases the yaw rate. Applying brakes on the rear axle has the contrary effect; increase on yaw rate of the vehicle. The above proof, agrees to the author of [59] where he asserts that, "All wheels are not equally effective in generating the yaw moment".

Thereafter, we have followed the same route as the [59] and modern ESP systems; applying brakes only at one wheel at a time, based on the decision of oversteer or understeer. Thereupon, we used an appropriate fitted reference yaw rate function of velocity and steering angle. If the error between the reference and the actual yaw rate measured from the gyro exceeded a certain amount then the system would detect oversteer or understeer and would apply brakes on the most effective wheel to counteract the effect.

To determine understeer or oversteer, we collate the measurement from an inertial measurement unit, a gyroscope at the center of gravity of the vehicle with the desired yaw rate. The control scheme is the following:

- Detection of Oversteer (measured yaw rate > desired yaw rate)

- Positive yaw rate measured from gyro scope
 - Apply brakes at the front left wheel.
- Negative yaw rate measured from gyros scope
 - Apply brakes at the front left wheel.
- Detection of Understeer (measured yaw rate < desired yaw rate)
 - Positive yaw rate measured from gyro scope
 - Apply brakes at the front left wheel.
 - Negative yaw rate measured from gyros scope
 - Apply brakes at the front left wheel.

The desired yaw rate function has derived from, [62] from the “Electronic Stability Control” chapter and is presented at Fig. 3.4 along with the complete algorithm with the sensitivity parameter. The parameters used are explained at Table 3-3.

Single gyroscope Electronic Stability Control Algorithm

- 1) Evaluate the reference yaw rate (ω_{ref}) the car should

experience according to current state and actual yaw rate (ω) from the gyro.

$$\omega_{ref} = \frac{V}{L + \frac{M \cdot V^2 \cdot (l_r \cdot C_{ar} - l_f \cdot C_{af})}{2 \cdot C_{af} \cdot C_{ar} \cdot L}} \cdot \delta = \frac{V}{L + \frac{M \cdot V^2 \cdot K}{2 \cdot L}} \cdot \delta$$

- 2) if $abs(\omega) \geq 10$

if $(abs(\omega)) \geq (abs(\omega_{ref}) \cdot (1/S) + 3)$ // oversteer

if $(abs(\omega)) > 0$, brake front left wheel;

else, brake front right wheel

if $(abs(\omega)) < ((abs(\omega_{ref}) \cdot S) \& \& (V > 2))$ // understeer

if $(abs(\omega)) > 0$, brake rear right wheel;

else, brake rear left wheel

else, do nothing; //normal steering

- 3) Go to step 1 (repeat forever)

S: Sensitivity $0 < S < 1$, best results with $S=0.9$

$K : \frac{(l_r \cdot C_{ar} - l_f \cdot C_{af})}{C_{af} \cdot C_{ar}}$ understeer gradient, best results with $K=0.004$

Fig. 3.4 Single gyroscope Electronic Stability Control Algorithm.

Parameter		Unit
M	Mass of the Vehicle	kg
l_f, l_r	Distance of front, rear axle from CG	m
L	Wheelbase ($L = l_f + l_r$)	m
δ	Steering angle: mean of both front wheels steering angles	rad
V	Longitudinal velocity along the X axis, average of longitudinal speed measured for each wheel	m/sec
C_{af}, C_{ar}	Cornering stiffness: front, rear axle	N/rad
ω	Angular rate around Z Axis at CG	rad/sec
ω_{ref}	Angular rate reference around Z Axis at CG	rad/sec

Table 3-3 Parameters for Single gyroscope Electronic Stability Control Algorithm.

When the brake torque is applied only at one wheel at a time, just like the above control scheme, from the relationship 3.25 we derive to relationship 3.26 ([59]), where the index i is set according to the single gyroscope ESC algorithm (Fig. 3.4):

$$\begin{bmatrix} \dot{\beta} \\ \dot{\omega} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \cdot \begin{bmatrix} \beta \\ \omega \end{bmatrix} + \begin{bmatrix} 0 \\ b_{2i} \end{bmatrix} \cdot T_{bi} + \begin{bmatrix} c_1 \\ c_2 \end{bmatrix} \cdot \delta_f \quad i = 1, 2, 3, 4 \quad (3.26)$$

A key point in the above algorithm is the determination of the reference yaw rate function. The determination, was done after we had gathered real data from the vehicle and some simulations on the Matlab. Thereafter, we had to build heuristically the most suitable reference function that would accomplish the best results for the ESC system.

The most important part of the above process was the determination of the understeer gradient “K”, at the reference function. The recipe was to gather data and collate different built reference functions (with different understeer gradient values) along with the real yaw rate the vehicle would measure. After that process, we would be able to substitute the most appropriate understeer gradient to the ESC reference function for the desired behaviour. The following figures (Fig. 3.5, Fig. 3.6) show real data gathered from the onboard sensors on the vehicle.

At the upper subplot of both Fig. 3.5 and Fig. 3.6, we can see the yaw rate measured by the gyroscope (red line) in parallel comparison with different understeer gradient values ($K=1$, $K=0.1$, $K=0.01$, $K=0.001$), substituted into the reference function mentioned at the ESC (Yaw Rate Des: Fig. 3.5) algorithm we used at Fig. 3.4. At the middle subplot we can see the turning angle and at the lower subplot, individual wheel speed. It is clear that the best results for the reference function would derive for an understeer gradient between 0.01 and 0.001. For $K=0.001$, at Fig. 3.5, we can see that the reference yaw rate, tracks the actual yaw rate measured by the gyroscope very closely.

It loses it at about Time= 1.8 sec, where the vehicle totally loses control, where the back left wheel starts spinning unbridled and produces a large amount of oversteer. So the reference yaw rate follows the actual yaw rate in a region that is undesired.

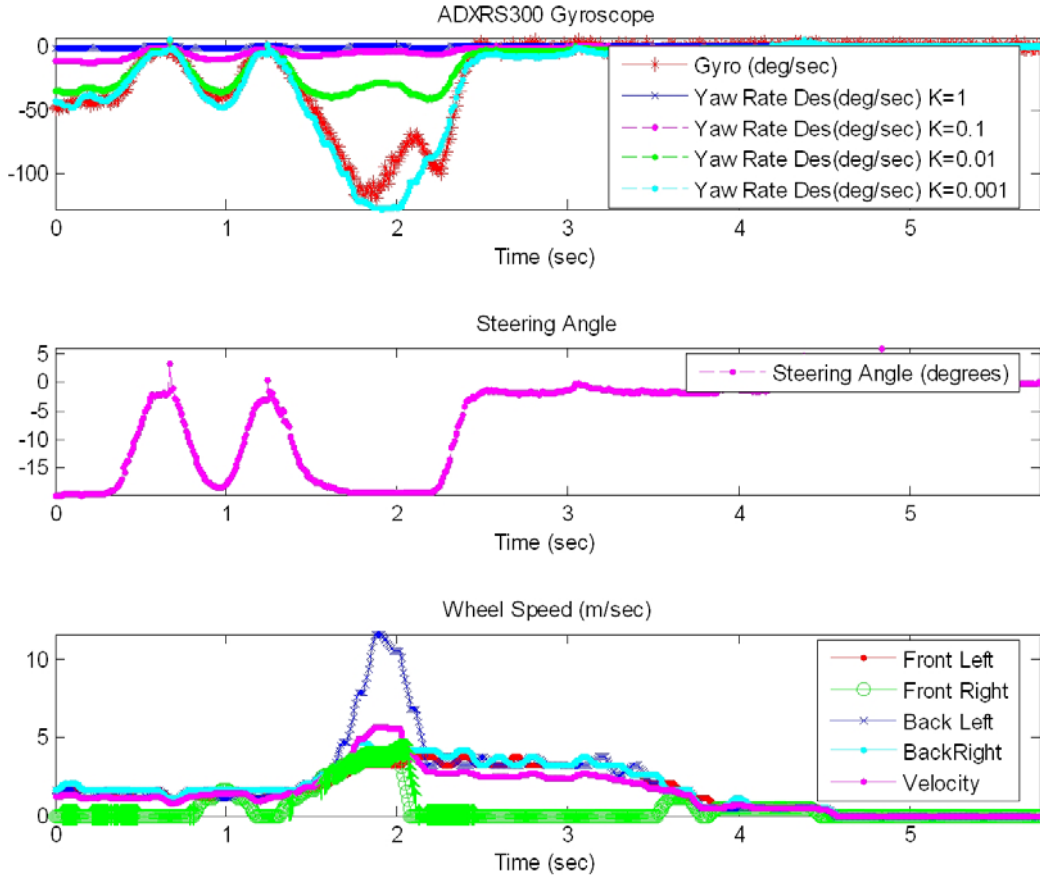


Fig. 3.5 Actual yaw rate collated with the reference for $K=1$, $K=0.1$, $K=0.01$, $K=0.001$.

At Fig. 3.6, we have plotted the same data as those at Fig. 3.5, but only with $0.01 < K < 0.001$ for $K=0.008$, $K=0.006$, $K=0.004$, $K=0.002$. After some trial and error tests, we decided that the value for the understeer gradient K that would be more appropriate for a not too aggressive neither too neutral ESC system would be $K = 0.004$ (Fig. 3.4).

Both figures, Fig. 3.5 and Fig. 3.6 are the same instance from a real experiment with the ESC fully functional. At Fig. 3.7 it is presented just the final yaw reference used, along with the decision and commands of the system. The green circles at the upper subplot denote the detection of the system for oversteer, and the arrow (green) along with individual wheel speed at the lower subplot, denotes the command the specific wheel to brake.

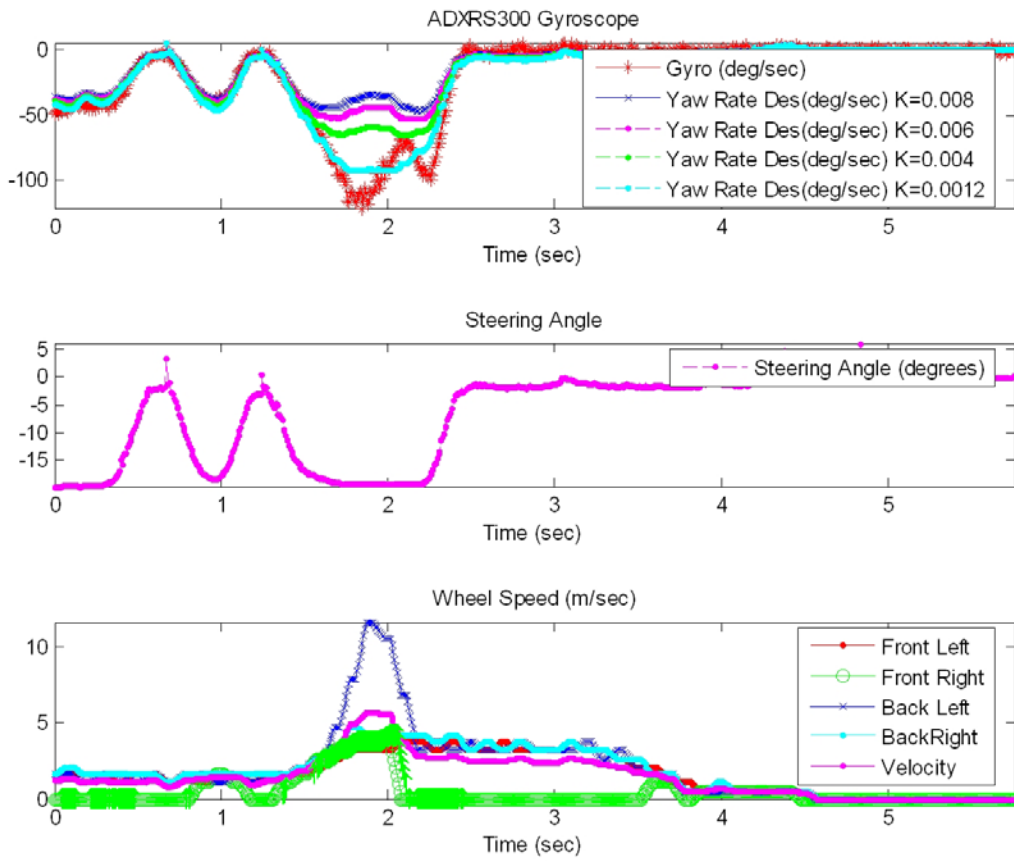


Fig. 3.6 Actual yaw rate collated with the reference for $K=0.008$, $K=0.006$, $K=0.004$, $K=0.002$.

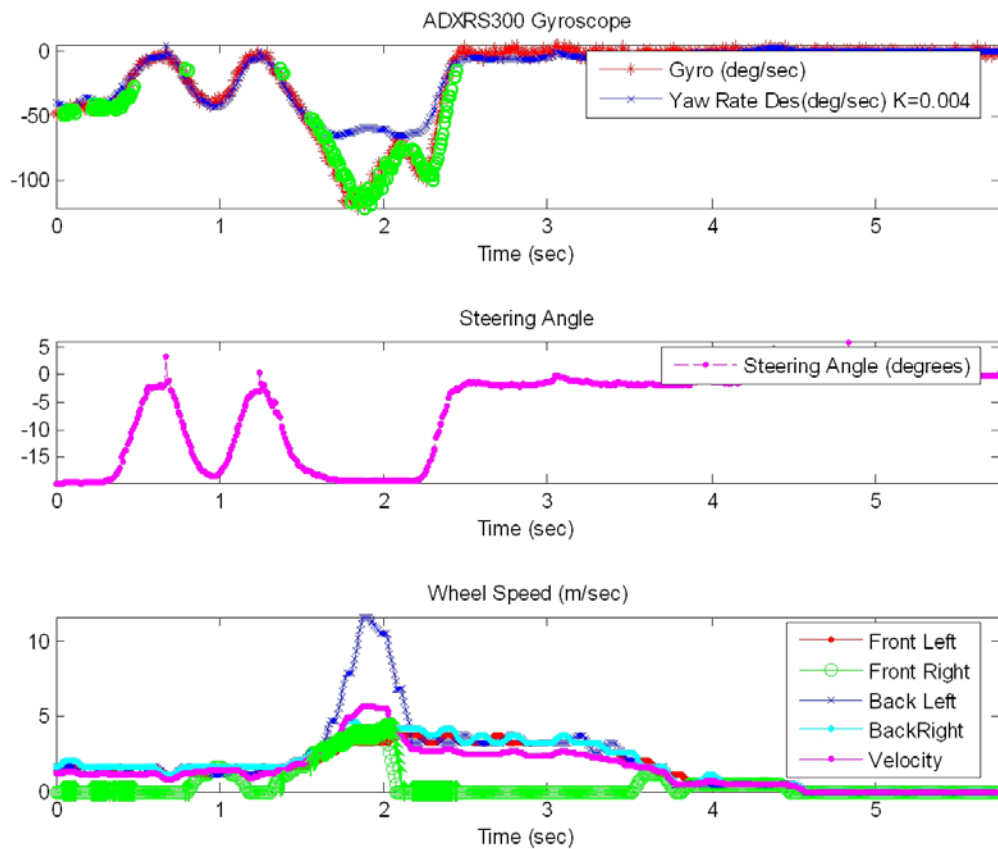


Fig. 3.7 Actual yaw rate collated with the reference for $K=0.004$. The green circles at the upper subplot denote the detection of the system for oversteer, and the arrow (green) along with individual wheel speed at the lower subplot, denotes the command the wheel is accepting from the system to brake.

The algorithm, although it is simple, proved valid. At the section “Real Environment Evaluation”, different scenarios will prove the effectiveness of the system. A first typical sample, can be obtained from Fig. 3.7, where the system detects oversteer (upper subplot on green circles) and counter acts the effect, by braking the appropriate wheel (which is the front right for this specific experiment). This action, causes an excessive amount of oversteer, which is shown as the immediate correction at the actual yaw rate, and finally stabilizes the vehicle while retaining the similar velocity as before the effect took place.

Another parameter this algorithm needs is the sensitivity of the system, which is translated as “what can be the error margin between the reference yaw rate and the actual yaw rate”. It is something that has to do with the driving style. The most convenient value for this parameter was the 0.9. The declaration of the parameter is shown at Fig. 3.4.

Analytic presentation of the algorithm

There are certain aspects concerning the ESC algorithm shown at Fig. 3.4 that must be clarified. Thus, in that section, we shall unravel each part of the algorithm analytically. Most constants inside the algorithm have come up from test runs and data inspection.

The first part of the algorithm ((1), Fig. 3.4), is the determination of the yaw rate reference function which was explained at the previous section.

The second part of the algorithm consists of seven parts:

- i. The condition “if $abs(\omega) \geq 10$ ”.
 - a. In order the system to act on brakes, we demand the vehicle to experience more than 10 degrees/sec yaw rate. This value came up from the collected data. We don’t want the system to intervene when it is unnecessary and the 10degrees/sec is a very low yaw rate for the vehicle to experience an undesired effect.
 - b. We have used the absolute into the “if” condition, because for the system entering the following “if”, where it will be determined if it understeers or oversteers and the direction of travel, we don’t care if the yaw rate is positive (right turn) or negative (left turn).
 - c. This specific testbed (because of the rear wheel drive and most of the load straining the rear axle, has a strictly oversteering behaviour) doesn’t have the slightest possibility experiencing less than 10degress/sec where it should experience more (understeering).

- ii. The condition “ if $(abs(\omega)) \geq (abs(\omega_{ref}) \cdot (1/S) + 3)$ ”

- a. At this condition the system determines whether the vehicle's yaw rate, stays inside the predetermined margin of yaw rate for the oversteer case.
 - b. It is examined if the yaw rate ($abs(\omega)$) is greater than the absolute yaw rate reference ($abs(\omega_{ref})$) multiplied by a constant factor greater than 1 ($0 < S < 1 \rightarrow (1/S) > 1$). This multiplication prevents the system from acting early and unnecessarily. Thus, it gives the driver the potential to determine sensitivity for the system.
 - c. The "+3" term came up from the experimental data. Because of measurement errors, we have $\pm 2^\circ / \text{sec}$ fluttering from the gyro. Thus, in low speed and a high sensitivity, let's say 1, the vehicle might detect falsely oversteer. For example, let's say the measurement from the gyro is 52deg/sec, when the real yaw rate of the vehicle is 50deg/sec. Also let's say the reference yaw rate generated from the function from Fig. 3.4 is 50deg/sec (the same as the real yaw rate). In case we have set the sensitivity 0.97, without this correction term (+3), the vehicle would detect oversteer and act on the brakes ($(abs(\omega)) \geq (abs(\omega_{ref}) \cdot (1/S) + 3) \rightarrow 52 \geq (50 \cdot (1/0.97)) = 51.54$), where there is no reason to do that. Thus this correction term is crucial for cases of high sensitivity and reference close to actual yaw rate. After some experimenting, we came up that the value "+3" was enough, while it wouldn't cause late acting on oversteer cases, because of its small value.
- iii. The term "*if $\omega > 0$, brake front left wheel;*"
- a. In case the vehicle is experiencing a positive yaw rate (right turn), greater than desired, then we should try to decrease it in order the vehicle to stabilize. Thereafter, through relationship 3.19, a negative F_{x1} force, which is acting on front left wheel (front left wheel braking), would decrease the first derivative of the yaw rate (angular acceleration around the Z axis), and thereupon would decrease the rate of change for the yaw rate of the vehicle.
- iv. The term "*else, brake front right wheel*"
- a. This is the inverse from the iii) part. In case the vehicle is experiencing a negative yaw rate (left turn), more negative than desired (left oversteer), then we should try to increase it in order for the vehicle to reach the reference. Again through relationship 3.19, a negative F_{x2} force, which is acting on front right wheel (front right wheel braking), would increase the first derivative of the yaw rate provoking the yaw rate to increase.
- v. The term "*if $(abs(\omega)) < ((abs(\omega_{ref}) \cdot S) \& \& (V > 2))$* ".

- a. At this condition the system determines whether the vehicle's yaw rate, stays inside the predetermined margin of yaw rate for the understeer case.
 - b. It is examined if the yaw rate ($abs(\omega)$) is lesser than the absolute yaw rate reference ($abs(\omega_{ref})$) multiplied by a constant factor less than 1 ($0 < S < 1$). This multiplication also prevents the system from acting early and unnecessarily, the same as the oversteer case.
 - c. The “($V > 2$)” term, which means “do not intervene, unless the velocity of the vehicle is higher than 2m/sec”, also came up from experimenting. It is likely for small velocities, that the vehicle detects understeer. This is because the term $\frac{M \cdot V^2 \cdot K}{2 \cdot L}$ from the denominator of the yaw rate reference function, because of the low velocity, doesn't dominate intensely to the term $V \cdot \delta$ at the nominator, although it is a second order term of velocity. This has as an outcome, at low velocities, the yaw rate reference get bigger, in analogy of that in higher velocities. This remark can also be extracted from the plotted data at Fig. 3.5, Fig. 3.6 and Fig. 3.7. Also the velocity of 2m/sec for the vehicle is not that big, so that the driver loses control of the vehicle.
- vi. The term “if $\omega > 0$, brake rear right wheel;”
- a. In case the vehicle is experiencing a positive yaw rate (right turn), less than desired, then we should try to increase. Thereupon, through relationship 3.19, a negative F_{x4} force, which is acting on rear right wheel (rear right wheel braking), would increase the first derivative of the yaw rate (angular acceleration around the Z axis), and thereafter would increase the rate of change for the yaw rate of the vehicle.
- vii. The term “else, brake rear left wheel”
- a. This is the inverse from the (vi) part. In case the vehicle is experiencing a negative yaw rate (left turn), less negative than desired (left understeer), then we should try to decrease it in order the yaw rate to reach the reference. Again through relationship 3.19, a negative F_{x3} force, which is acting on rear left wheel (rear left wheel braking), would decrease the first derivative of the yaw rate provoking the yaw rate to decrease.

3.3 Single Accelerometer Electronic Stability Control

A second stabilization scheme was designed in order to be simple to implement, easy to adjust and would be cost effective on a real vehicle, considered an already ABS system installed. It could

be implemented with a single dual axis accelerometer and a simple microcontroller with an embedded CAN Bus core, in order to cooperate with the ABS controller and the ECU of the vehicle.

The radius of turn for a vehicle travelling with low speed, with absence of oversteer or understeer, assuming small angles will be equal to $R_m = L / \delta$ [14]. This is the inverse of Ackerman steering angle, for a desired radius of turn. We can use the inverse of Ackerman's angle as a reference model for driver's desired radius of turn. The actual turning angle can be determined from a dual axis accelerometer, installed at the CG of the vehicle. The accelerometer can measure lateral acceleration towards the center of turn. The relationship is:

$$Acc = \frac{V^2}{R} \quad (3.27)$$

Through the accelerometer, we can determine the magnitude of slip angle for CG. The direction of travel for the vehicle at CG is vertical to the vector of lateral acceleration. The dual axis accelerometer is oriented so that can measure both lateral (Acc_y) and longitudinal (Acc_x) acceleration of the vehicle. Therefore, slip angle a_{CG} at CG (Fig. 3.1) can be determined by:

$$a_{CG} = \tan^{-1} \left(\frac{Acc_x}{Acc_y} \right) \quad (3.28)$$

Since the only available speed for measurement is towards the vehicle's longitudinal axis (V_x), we can compute V via: $V = V_x \cdot \cos(a_{CG})$ (3.29)

Thus, through equations 3.27, 3.28 and 3.29 we can determine the radius of turn relatively well. The stabilization algorithm is built for a RWD vehicle. The actuation is performed on the brakes of front or rear axle. A sensitivity parameter S has been implemented in the algorithm for an adjusted desired understeer or oversteer behaviour of the vehicle.

Single Accelerometer ESC algorithm

- 1) Evaluate Driver's Desired Radius of Turn $R_m = \frac{L}{\delta} \cdot V \cdot k$, δ : steering input in radians
 - 2) Estimate Actual Radius of Turn $R = \frac{V^2}{Acc} \left\{ \begin{array}{l} V = V_x \cdot \cos(a_{CG}) \\ a_{CG} = \tan^{-1} (Acc_x / Acc_y) \end{array} \right\}$
 - 3) If $R < S \cdot R_m$ ^{*1} Apply Instantaneous Brakes on Front Axle
 Else if $R > (1/S) \cdot R_m$ Apply Instantaneous Brakes on Rear Axle
 Else Do nothing
 - 4) Repeat Forever
- * k: non linear coefficient ^{*1}S: Sensitivity $0 < S < 1$

Fig. 3.8 Single Accelerometer ESC algorithm

3.3.1 Real Environment Evaluation for the Single Accelerometer ESC

The stabilization algorithm proposed at above and was summed at Fig. 3.8, although it looks promising, hadn't the expected results when evaluated on the test bed. Real environment simulation showed that the use of a single accelerometer for electronic stability control is inhibitory for a scaled vehicle. The limited functionality of the stabilization algorithm lies in the physical operation of the accelerometer. Vibration from the engine (two stroke single cylinder engine) and the anomalies from the ground in combination with small size of the vehicle corroded the measurements from the accelerometer (see explanation at the fourth chapter, "ADXL311 and ADXL213 bandwidth selection" section). The outcome from the miss measurements was a miss calculated actual radius of turn. In most cases, the algorithm detected understeering, thus applied brakes on the rear axle, provoking the vehicle to oversteer.

Another drawback of the algorithm that was clarified from experimental data is that the reference radius used in the algorithm is valid only for very slow speed turning. Measurements from the rest of the sensors revealed that the necessary centripetal force for the vehicle to follow the desired radius of turn is difficult to be produced in real environment. This is the reason that we used the term $R_m = (L/\delta) * V * k$ at Fig. 3.8. Initially this term didn't include the velocity, but only the non – linear coefficient (first results published at [58]). Neither this major correction helped the applicability of the algorithm.

From the data, we can also derive to the maximum yaw rate, before the vehicle loses control, with respect to individual wheel speed and steering angle. Loss of control can be determined from the behaviour of vehicle's yaw rate. That means if we have a constant steering angle and small variations at the speed of front wheels (back wheels might be spinning, thus are bad candidates for remarks) and we experience great variations on yaw rate the vehicle oversteers. On the other hand, if we have an increase at the speed of vehicle's front wheels and/or increase at steer angle, and experience little or decrease in magnitude of the yaw rate, the vehicle understeers.

From the above results, we concluded that the applicability of the Single Accelerometer ESC algorithm is rather poor and wouldn't even worth testing on a real vehicle. Comparing it with the performance of the Single Gyroscope algorithm presented earlier we can conclude that the use of gyroscope is probably one way solution (until now, September 2008) for building an ESC.

An alternative approach for abolishing the use of the gyroscope would be to estimate the forces acting on the wheels directly. Lateral and longitudinal force could be measured from the ball bearings the wheels seat, an innovative and robust method that the author of this thesis will research intensively in the near future with the greatest ball bearing manufacturer, the Swedish SKF for building a fault tolerant ESC system for real vehicles.

4. System's Implementation

The Test Bed is based on a XRC 1:5 scale remote control car. The model is a 2 rear wheel drive car, with a single cylinder 23cc two stroke air – cooled engine. It has a centrifugal clutch for transmission, a single disk brake for the rear axle and two independent disk brakes for each of the front wheels. It has a standard 2 channel FM radio with one servo for steering and another for throttle – brake with 18kg·cm and 5.5kg·cm torsion correspondingly with response time of 0.48s/60°. In order to meet our experimental standards, the platform had to be completely self – contained without equipment or machinery off the vehicle, had to be inexpensive, independent from special hardware and commercial software and comply with the terms of GNU General Public Licence. The model is equipped with all the necessary sensors, actuators and computing power for data fusion, dynamics modelling and control. The main processing unit of the system is a mini – ITX VIA Computer running Linux. The data logging and actuation control is imploded trough an ATMEL 8 – bit microcontroller which communicates through the serial port with the computer. The majority of the software for the microcontroller and the computer is written in C/C++ and all the necessary hardware is built from scratch. The system has a wireless LAN for remote access through a laptop with IEEE 802.11.

4.1 Mechanical modifications

The model at its original state was far beyond from an appropriate platform for experimenting. The mechanical flaws, the low quality of materials used for its construction and the malfunctioning installed electronic system, demanded an extensive rebuild and in some cases totally new mechanical and electronic parts. The model, also had to carry all the necessary equipment mentioned above, which had as an outcome the rearrangement of some vehicle's components and the mounting of substructures that would support the cases of computer systems, sensors, actuators and peripherals. At the figure below (Fig. 4.1) one can have a general perspective of the hardware placement.

The most important part of the system is the central processing unit, the single board computer, which is located inside a plastic insulated case mounted at the centre of the vehicle. This case was originally an electrical case suitable for outer use. The case is mounted on an aluminium rod screwed on the chassis, which allows a vibration in lower frequency than the typical frequency of operation of the engine which has a range of 15Hz – 100Hz (900rpm – 6000 rpm respectively).

With the above method and not directly connecting the computer to the rigid body of the vehicle we can expand the life expectancy of the computer.

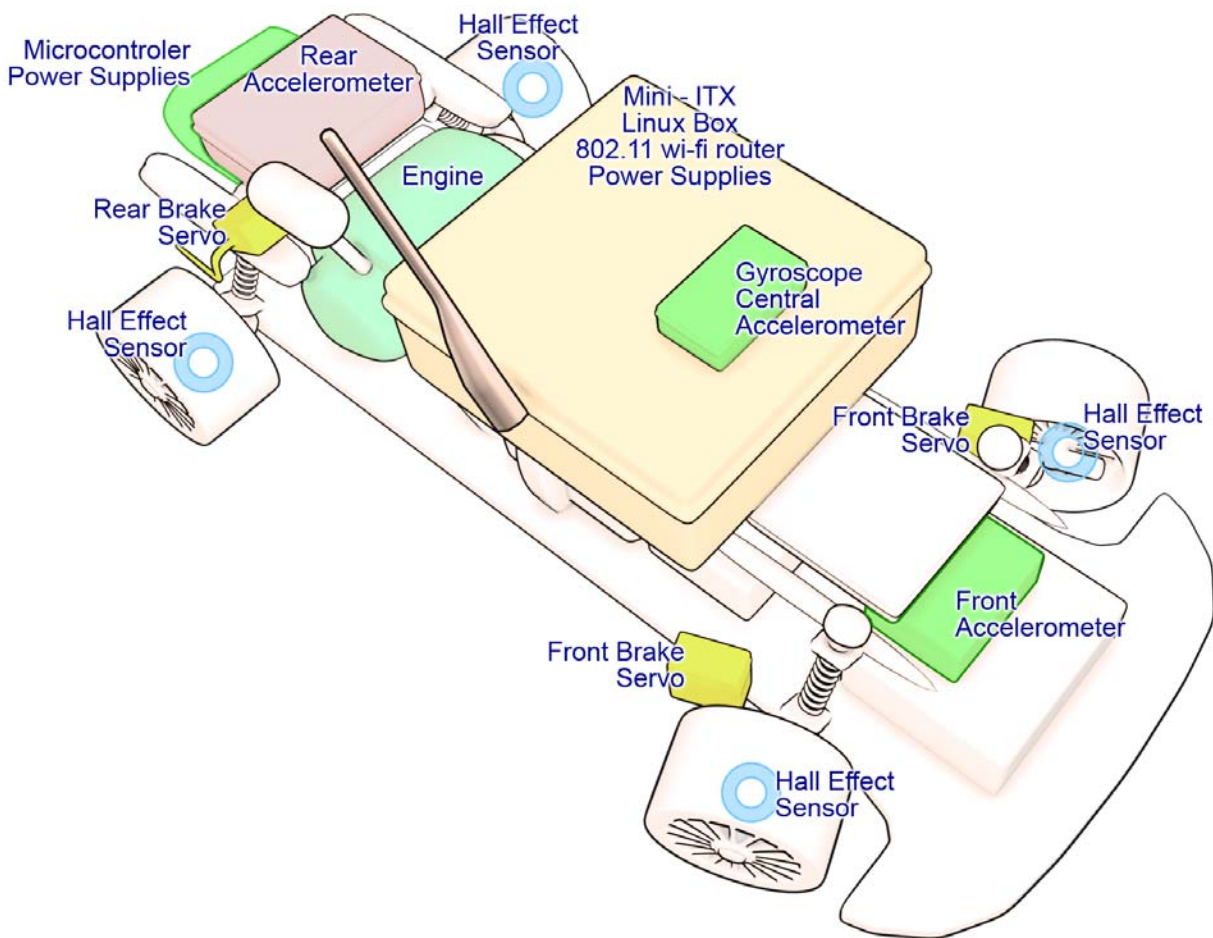


Fig. 4.1 Hardware Placement.

The SBC board computer inside the plastic case has been attached on rubber taps for further reduction of vibration delivered to the computer from the engine and of course the road. The case has also been modified to host a wireless compact router connected to the SBC via the Ethernet port and the power source of the whole system, which is 12V 4Ah Nickel – Metal Hydride battery pack, the custom built power supply for the router and a small fan (Fig. 4.2).

One extensive mechanical modification was on the front brakes. We completely removed the non functional built in front braking system, which was controlled by the throttle servo. At its prime state, both brakes were connected to a rod attached to the throttle servo through two Bowden cables and would brake all wheels (including those at the rear axle) at the same time. Most of the power was lost on the poor quality cables, which also locked when they were pulled and wouldn't return to their normal position. We left the original braking disk and pincer which we improved (added spring to unlock the pincer etc) but we used different braking actuators for each wheel since an ESP system demands individual wheel braking.

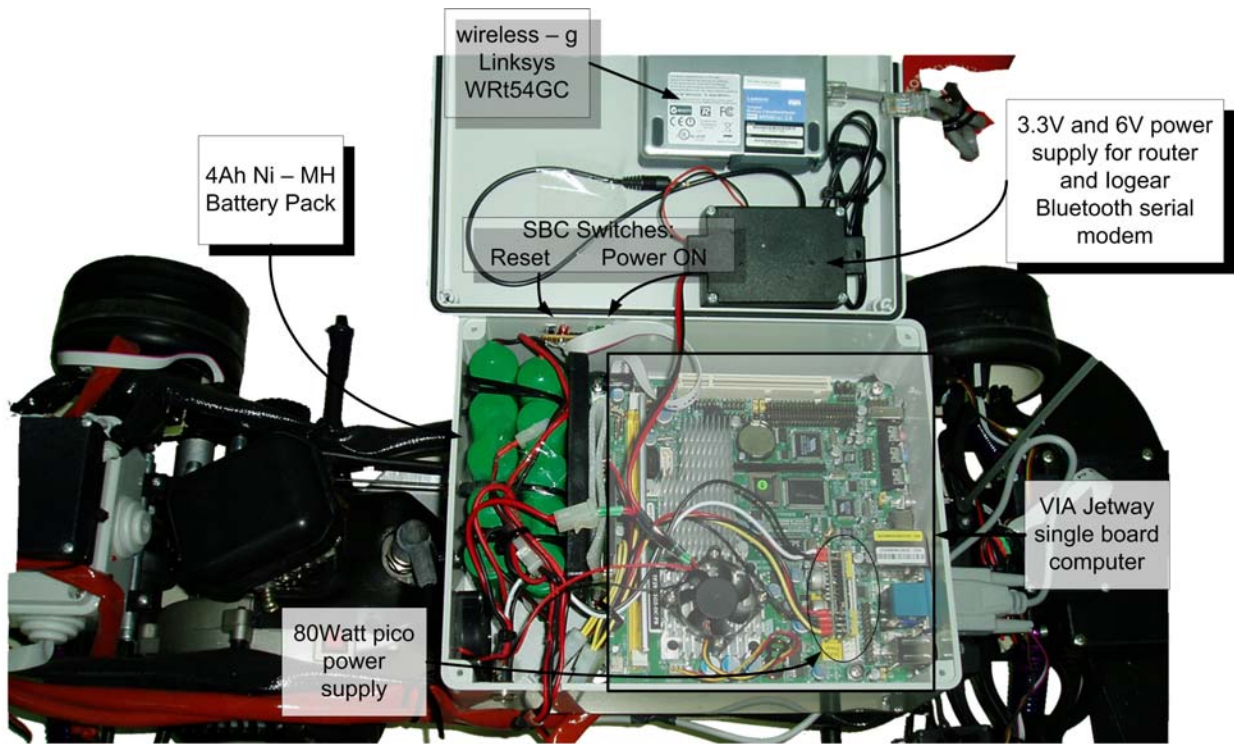


Fig. 4.2 Central case; host computer and peripherals.

Our low cost specifications demanded standard model servos as actuators and not hydraulic brakes that are used on expensive 1:5 scale model cars. In order to diminish the latency time between the braking command and the braking action, we decided to place the actuators as close as possible to the brakes themselves.

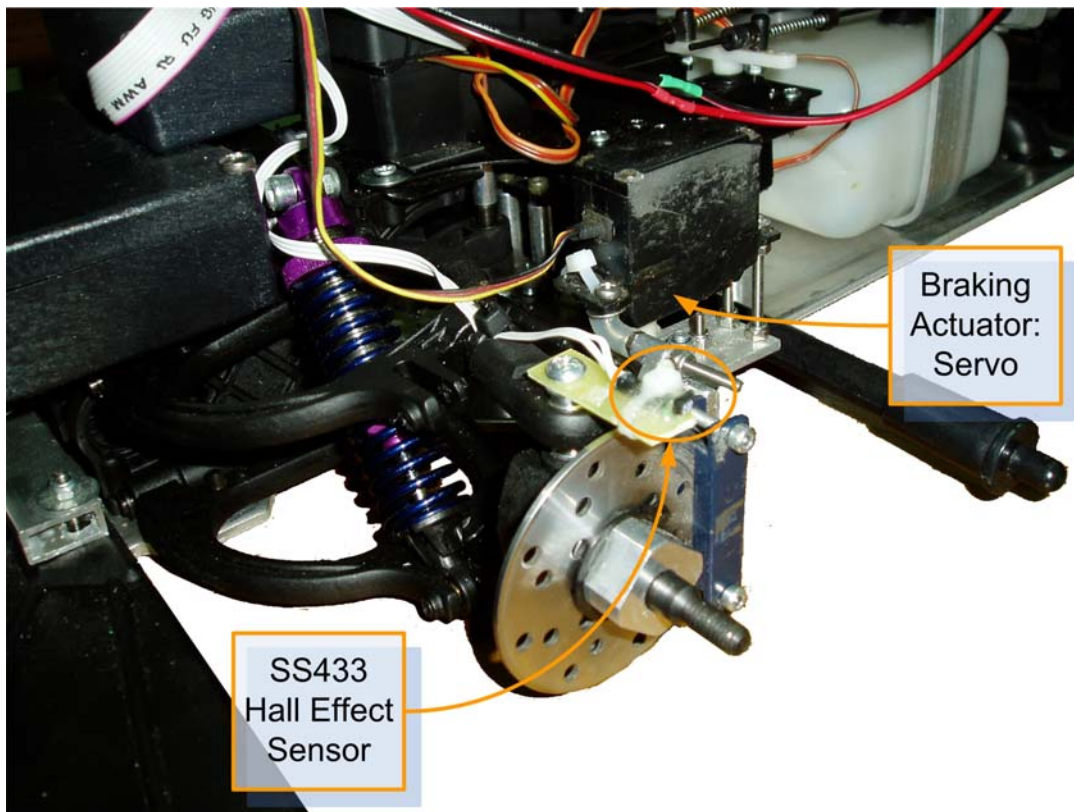


Fig. 4.3 Front braking system.

We also tried to use as few as possible transmission mechanisms for the motion derived from the servos, in order to reduce back – lash and elastic effects. Thereafter, we placed the servos upside – down on each fellow (Fig. 4.3) and used an iron articulated mechanism to deliver the braking action. They were set up so as to take only few degrees of servo turn between releasing and braking the wheel. You can also notice at Fig. 4.3 the Hall Effect sensors used. The Hall Effect sensor faces the rim, which has been attached with 8 reed relay magnets polarized wisely so that every time a magnet passes in front of the sensor, sinks its output.

For the rear axle brakes we went through a simpler solution. We used the already installed braking system which is a disk brake attached to the differential of the vehicle and a pincer controlled from the throttle servo, in parallel with our ESP controlled brake (Fig. 4.4). This method was followed for safety purposes, so that in case of ESP's malfunction we have a redundant brake. We don't have individual wheel braking, since the vehicle didn't have the provisions for installing a system relative to the front axle and the braking command acts to both rear wheels. The effect from braking both wheels isn't as rewarding as braking one wheel but it is effective too and due to the strictly oversteering behaviour this recession doesn't affect the system.

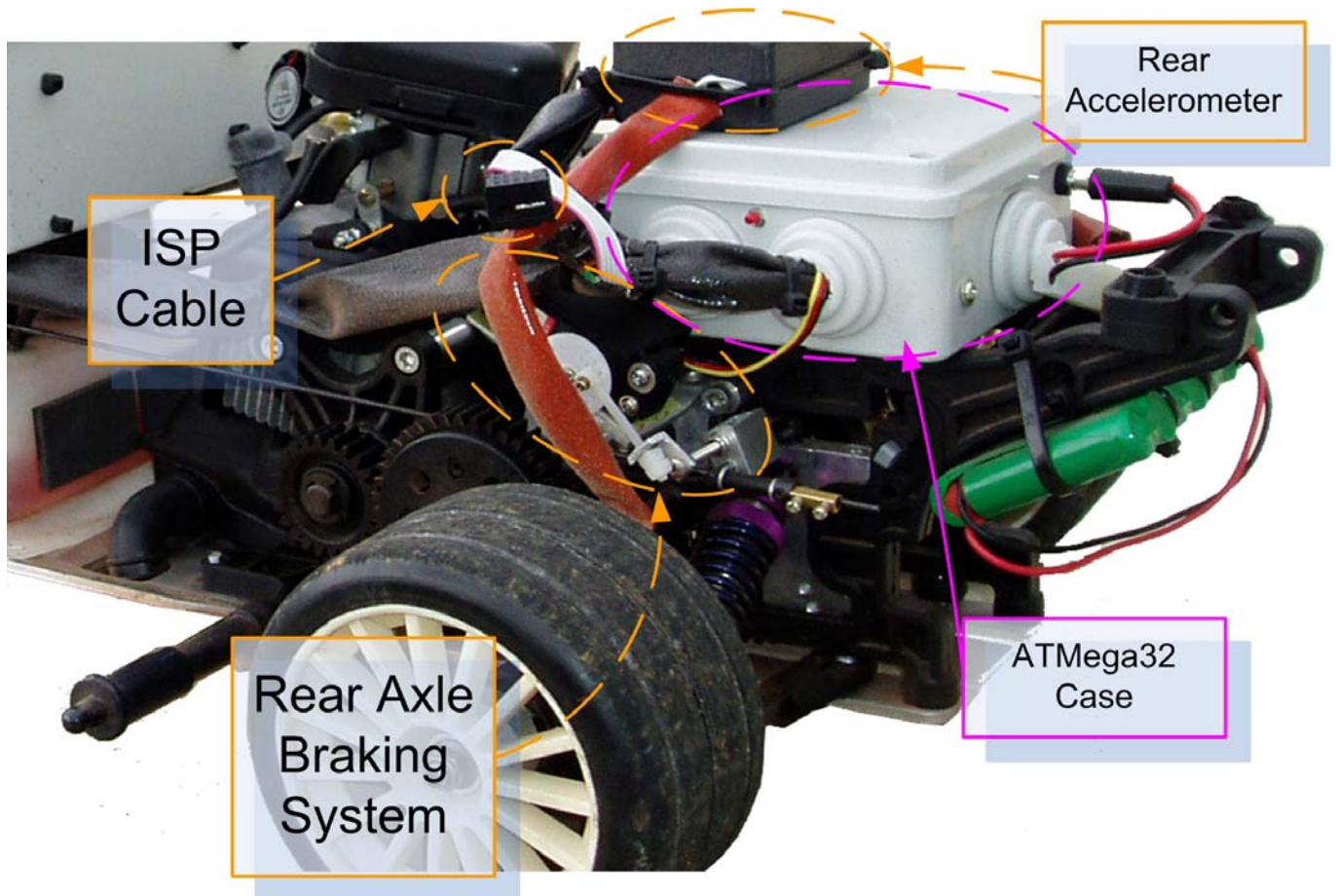


Fig. 4.4 Rear axle braking system and case mountings.

At Fig. 4.4 we can also see the mounted white plastic case which houses the printed circuit board with the microcontroller, the power supplies for the peripherals, such as actuators and sensors and finally the sockets for all the wiring system that delivers the power and samples for and from the sensors, and the power and commands for and from the actuators. We can see the top view of the plastic case and the organization of the hardware inside on the next figure (Fig. 4.5).

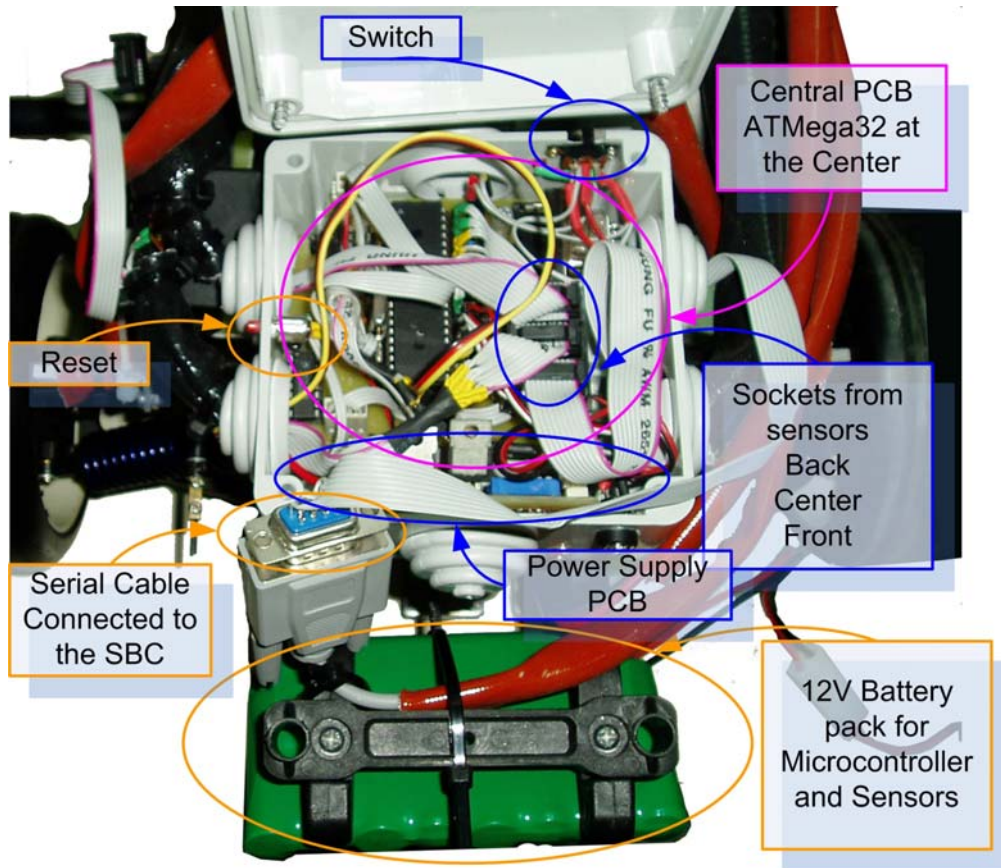


Fig. 4.5 Plastic case with the ATmega32.

The microcontroller and the sensors are powered by a 600mAh, 12V Nickel – Cadmium battery pack, capable for delivering power for more than 10 hours, through the power supply PCB, which we have mounted vertically from the ground inside the plastic case (Fig. 4.5). The power supply has regulators for producing different voltages: 5V, 6V and an adjusted voltage source from 2V – Input V. The whole system was built very carefully in order to have the less power dissipation possible (more analytical in the next two sections). Finally, the steering angle from the model is estimated through an articulated mechanism, which moves the shaft of a potentiometer, with the motion derived from the steering axon (Fig. 4.6).

The output of the potentiometer is sampled through an Analog to Digital Converter (ADC). Using the value of the voltage and through a fourth degree polynomial, we can estimate the steering angle. The potentiometer is insulated for heat, dust and moisture, similar to every exposed wire on the model.

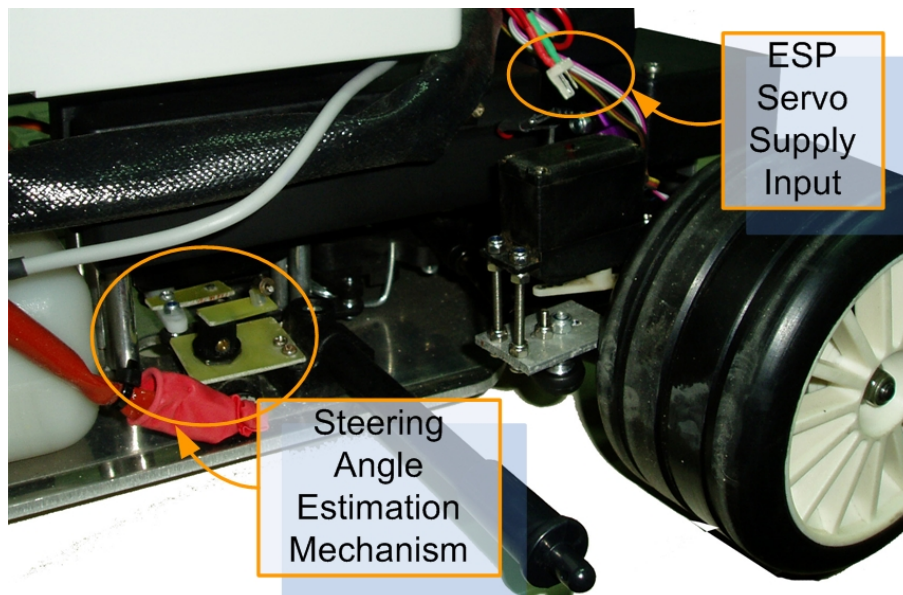


Fig. 4.6 Steering angle estimation mechanism.

4.2 Computing hardware

As mentioned previously, our goal was to build a low cost experimental platform with the potentiality to alter and publish software and results, without any special licence. We concluded that a mini – ITX (form factor 17cm*17cm) PC with a free operating system would be appropriate. Hence the system is based on VIA Jetway single board computer (SBC), with onboard VIA C7 1.5GHz nanoBGA2 Processor, an integrated graphics card, 400MHz FSB, 512 MB DDR2 400MHz RAM, 1 PCI slot, ATA 100/133 & 2 SATA ports support, IEEE 1394 firewire, 8 USB 2.0 ports & 2 COM ports and VIA 10/100 Ethernet, running an Ubuntu 6.10 Linux distribution, complaint under the GNU general public licence. There has also been installed a wireless – g Linksys WRt54GC compact router connected to the SBC via the Ethernet port. The SBC is powered through 12V 80Watt pico power supply and the power source is 12V, 4Ah Ni – Mh battery pack. Data fusion and actuation control is conducted by a versatile custom designed 2 – layer PCB. The PCB has an attached AVR ATmega32 RISC microcontroller connected to the serial port of the SBC with 32Kb program flash and 2Kb RAM. USB communication was prohibitive, because it would consume the majority of computing power of the microcontroller who runs in 16MHz. ATmega32 doesn't have an embedded USB core, so the communication would have to be incorporated in software. The speed of data flow is 115.2Kbs which is more than adequate for our purpose, since the bottleneck of the system is the actuation and not the computing power. The microcontroller has been exploited to the limits, since it manages to control 6 actuators (servo) and log data from 17 sensor inputs, in a sum of 32 I/O ports with an astounding real time precision.

4.2.1 Sensors

Front and Rear Axle Accelerometers: ADXL311

Lateral and longitudinal acceleration in the front and rear axle of the vehicle is evaluated through two ADXL311 accelerometers from Analog Devices. ADXL311 is a low cost high accuracy $\pm 2g$ accelerometer with analog output, proportional to the measured acceleration ([17]). It is housed in Leadless Chip Carrier (Lcc) package with 8 output lines and can measure acceleration into two axes, X – Y. The output of ADXL311 (X and Y axis measurement) is fed to the microcontroller through an analog to digital converter (ADC) I/O line. The bandwidth of the measurements is easily adjusted through two capacitors C_X and C_Y at the X_{OUT} and Y_{OUT} pins (Fig. 4.7). The less the bandwidth the lower the measurement error. ADXL311 can measure static and dynamic acceleration proportional to the acceleration it experiences. The typical noise floor limit is $300\mu g / \sqrt{Hz}$ providing the capability of measurement with 2mg accuracy, at the low bandwidth of 10Hz.

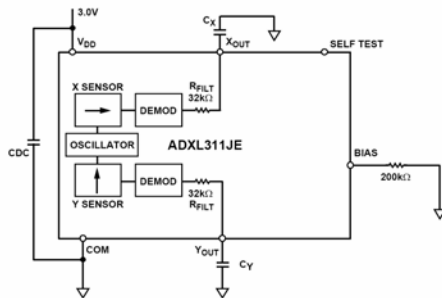


Fig. 4.7 Functional diagram of ADXL311 from data sheet [17].

The theory of operation of the accelerometer is the following: The sensor is a surface micro – machined structure of polysilicon built on top of the wafer suspended by polysilicon springs that provide a resistance against acceleration forces ([17]). This structure is stressed relative with the acceleration it experiences. The deflection of the structure is estimated by a differential capacitor ([18]), which consists of the stators, seating on the polysilicon surface and the rotor which lies on top of the structure, which is mounted on the springs. The stators of this differential capacitor, are driven by squares waves which are 180° out of phase. The deflection because of the acceleration will result to a square wave proportional to the acceleration. Afterwards, with some demodulation procedures based on the phase of the wave, can be determined the direction of the acceleration.

ADXL311, has provisions for bandlimiting the noise, acting as low pass filter while diminishing the noise. The equation for the -3dB bandwidth is: $F_{-3dB} = 1/(2 \cdot \pi \cdot (32k\Omega) \cdot C_{(X,Y)})$. The next table has the typical values of capacitors for determining the bandwidth.

Bandwidth (Hz)	Capacitor (uF)
10	0.047
50	0.01
100	0.05
200	0.027
500	0.01
5000	0.001

Table 4-1 Capacitor (C_x , C_y) selection for bandwidth determination.

Noise from the ADXL311, has characteristics of white additive noise and contributes equally to all outputs. The output of this sensor is analog and for $V_{dd} = 5V$, the sensitivity is approximately 312mV/g. For 0g acceleration, the output equals with $V_{dd}/2$ for every voltage of operation.

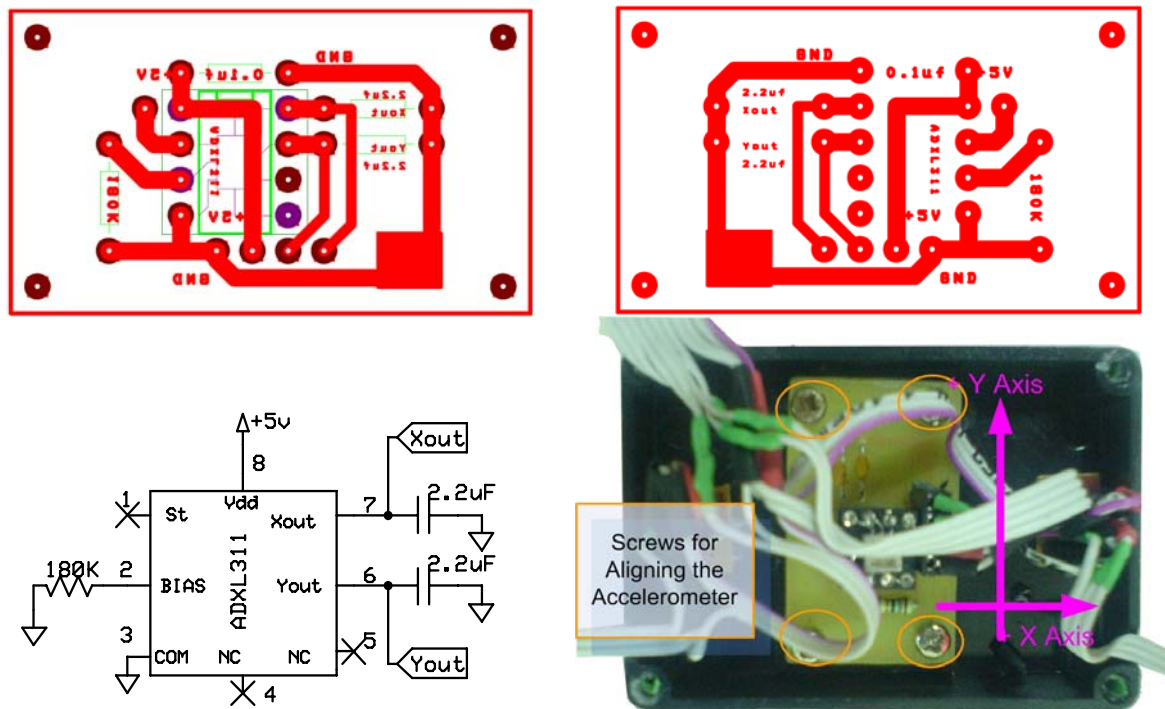


Fig. 4.8 PCB (top left) top layer, face of the copper at bottom layer (top right), schematic of the circuit used (bottom left) for ADXL311 and the sensor placed inside it's housing (bottom right).

Analog Devices Corporation has announced that this specific accelerometer is not recommended for new design since it will be withdrew from manufacturing and proposes as alternative choice the ADXL320. We could agree that this withdrawal is normal, since the noise measured from the accelerometer deviates from the data sheet and also because of accelerometer's nature of operation,

analog output voltage depending on the power supply, is prone to mismeasurements. The acceleration axis for ADXL311, are illustrated at Fig. 4.8, at the bottom right subfigure. The orientation and direction of the arrows, shows the positive acceleration. At Fig. 4.8 we can also see the PCB that the accelerometer has been mounted on and the schematic diagram of the circuit. More details for all those mentioned on this paragraph at “Observation, problem and possible improvements on the interface between the accelerometers and the microcontroller” section. At the bottom right subfigure of Fig. 4.8 one can observe system with the four screws for the aligning of the accelerometer with the plane.

ADXL311 ±2g interface with the microcontroller

The output of the ADXL311 accelerometer is an analog voltage proportional to the acceleration. Thereafter, the interface between the microcontroller (ATMega32) was simple. We have used 4 out of the 8 ADC ports from ATMega32 for measuring, X and Y axis of both front and rear axle ADXL311 accelerometers.

The ATMega32 is a RISC microcontroller, capable for 1MIPS per MHz, thus we have the execution of a single instruction per clock pulse ([20]). It has been set to work with the frequency of 16MHz. The built in analog to digital converter of the microcontroller has maximum resolution of 10bits and completes a conversion every 13 ADC clock cycles (almost). At the firmware running, we have used as clock source for the ADC the clock from the AVR, with the ADC clock prescaled at 64. That means that for every AVR clock pulses, we have 1 ADC clock pulse. This is translated as a conversion every $\frac{13 \cdot 64}{16 \cdot 10^6} \text{ sec} = 52 \cdot 10^{-6} \text{ sec} = 52 \mu \text{ sec}$. The conversion for each port is computed in cyclic mode.

We have used 7 ADC ports. That means that we have a new sample every almost $7 \cdot 52 \mu \text{ sec} = 364 \mu \text{ sec} \approx 0.4 \text{ msec}$. The “almost” at the equation arises because of the fact, that we don’t initiate a new conversion right after the previous finishes, but we do have some latency from new call of the routine that is responsible for this cyclic succession of conversions. The previous routine was big enough to be embedded into the ADC interrupt routine, so it is called from the endless loop of the firmware. More details on the firmware section.

Central Accelerometer: ADXL213

Lateral and longitudinal acceleration for the center of gravity of the car, is evaluated via one ADXL213 ±1.2g accelerometer from Analog Devices housed into an LCC package. It can measure acceleration in two axes, X – Y, with a range ±1.2g range of measurements. The output of this sensor is digital signals, duty cycled modulated. The output is proportional to the

acceleration and it is evaluated from the ratio of pulse width to the period (30% /g of acceleration). To typical noise floor limit is $160\mu g / \sqrt{Hz}$. The user selects the bandwidth through capacitors C_X and C_Y at X_{filt} and Y_{filt} pins respectively (Fig. 4.9).

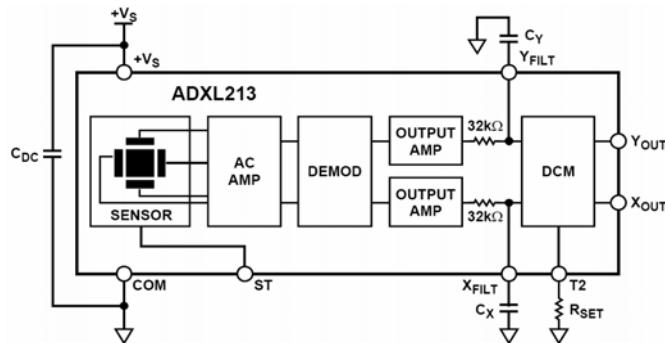


Fig. 4.9 Block diagram from ADXL213 data sheet [19].

The theory of operation is the same as ADXL311, with the main the difference that the output of this sensor is pulse width modulated and the acceleration can be realized from the duty cycle of the signal. When the output passed through the lowpass filter, the duty cycle controller, modulates appropriately the signal. For 0g acceleration, we have 50% modulation Fig. 4.10.

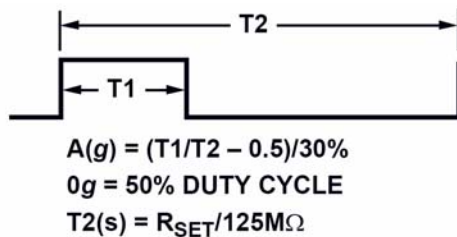


Fig. 4.10 Duty Cycle Modulated output for 0g acceleration from data sheet [19].

The acceleration can be realized by estimating the width of the positive pulse $T1$ (Fig. 4.10) and the period $T2$. The formula for estimating the acceleration is ([19]):

$$\text{Acceleration} = \frac{\left(\frac{T1}{T2} - \text{Zero g Bias} \right)}{\text{Sensitivity}}$$

Where:

Zero g Bias=50%=0.5 nominal

Sensitivity= $\frac{30\%}{g}$ =0.3 nominal

$T2 = R_{set}/125M\Omega$ and $R_{set}=1M\Omega$

From the above formula, it is clear that the period of the signal is specified by the resistor R_{set} (Fig. 4.11). Bandwidth is determined exactly the same way as ADXL311, but the Peak to Peak and RMS measurement error is much less than that of ADXL311.

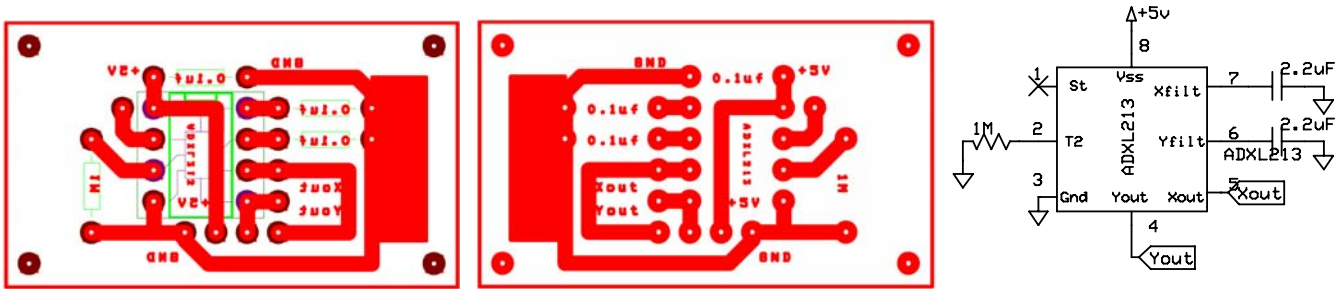


Fig. 4.11 PCB top layer (left), face of the copper at bottom layer (center), schematic of the circuit used (right) for ADXL213.

ADXL213 $\pm 1.2g$ interface with the microcontroller

The computing of the duty cycle for both of X and Y axis for the ADXL213, is conducted through an external interrupt port. ATmega32 has only 3 external interrupt ports and they are quite valuable to be wasted for the measurement of the values of just one accelerometer. For this reason we built a smart provision that would allow us to use one external interrupt port for both axes through a XOR gate. We have programmed the external interrupt to be executed at any logical change of the port. Whenever the microcontroller realises a rising edge at the external interrupt, starts counting the width of the logical “one” pulse with the help of the *Timer Counter Overflow routine*. The period of the PWM signal according to the data sheet for the ADXL213 is $T2 = R_{set}/125M\Omega$, and we have used $R_{set} = 1M\Omega$. This value was supposed to give us period of PWM signal with 125Hz frequency. But the real signal was 136Hz. We couldn’t determine the cause of a deviation with that magnitude. It is not referred to the data sheet, although it is a problem also presented to other engineers as we found out from a little search on the internet.

For the counting of the duty cycle we have used one of the two 8 bit Timer Counter of the Atmega32. This Counter has been programmed to signal an overflow interrupt and has Timer – Prescaler equals to 0 (the Timer Counter Increases by one with every clock cycle). Since the Counter is 8 bit length, an overflow occurs every 256 clock cycles. The clock cycle lasts 0.0625usec (16 MHz clock), that means we have an overflow every $256 \times 16 = 16\text{usec}$. The period of the pulse is 136Hz which means $T = (1/136) \text{ sec} = 0.00735 \text{ sec} = 7.35 \text{ msec}$. If we divide the period of the pulse with the sampling period we get $0.00735 \text{ sec} / 16\text{usec} = 460 \text{ samples per pulse}$.

From the formula for estimating the acceleration presented in the previous section, we saw that the sensitivity of the ADXL213 is 30% per g of acceleration. The maximum range of measurement is $2.4g$ ($\pm 1.2g$). So the 72% of the pulse represents a range of $2.4gs$, where the 100% of the pulse would represent $2.4g/0.72=3.333g$. That result divided with the number of samples we can have per sample gives a maximum resolution of $3.333/460 \approx 7.2$ mg, a value more than adequate for our application.

As mentioned above, the output for both axes of ADXL213 are measured through a XOR gate guided to an external interrupt. The output from each X axis and Y axis is directed to the 1st and 2nd input of the XOR gate and PORTC6 and PORTC7 of the microcontroller respectively (Fig. 4.12).

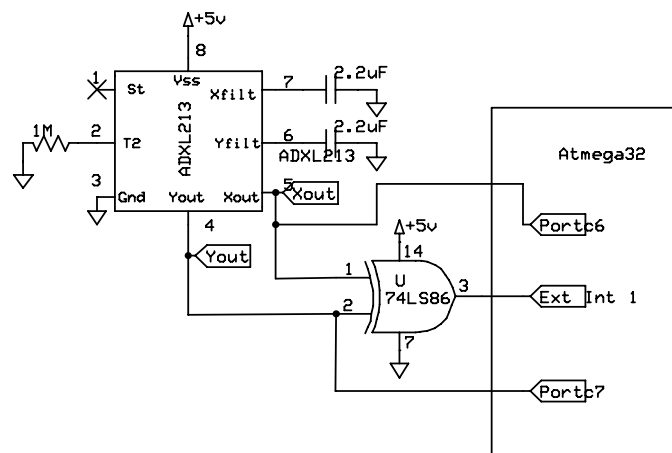


Fig. 4.12 ADXL213 interface with AVR.

The algorithm for counting the duty cycle is fast and economic in program resources. It consists of two parts. The first part is *External Interrupt* routine (Table 4-2) and the second the *Timer Counter Overflow* routine (Table 4-3), inside the firmware running at ATmega32. It uses the fewer variables possible. Since both X and Y axis pass through the XOR gate to the AVR external interrupt, any logical change to either of the outputs would change the output of the XOR gate, thus causing an external interrupt.

```
// External Interrupt Request 1, +1.2 Accelerometer
SIGNAL(INT1_vect)
{
    temp=PINC;
    if (temp&0b01000000){
        if(acc_12_X_ticks==0)
            acc_12_X_ticks++;
    }
    else{
        if(acc_12_X_ticks>0){
            if(TX_read!=3)
                acc_12_X=(acc_12_X_ticks-1);
            acc_12_X_ticks=0;
        }
    }
}
```

```

}...

```

Table 4-2 External Interrupt Request 1, routine (part of it)

```

/* Timer/Counter0 Overflow */
SIGNAL(TIMER0_OVF_vect){
    if(acc_12_X_ticks>0)
        acc_12_X_ticks++;
    if(acc_12_Y_ticks>0)
        acc_12_Y_ticks++;
    ...
    ...
}

```

Table 4-3 Timer Counter0 Overflow, routine (part of it)

When the External Interrupt request is issued the program checks for the logical state of each axis. Using of the same counter used to count the time quantum as a flag for the previous logical state of the Axis we can count the duty cycle. The flow chart of the counting operation is on the following flow chart (Fig. 4.13).

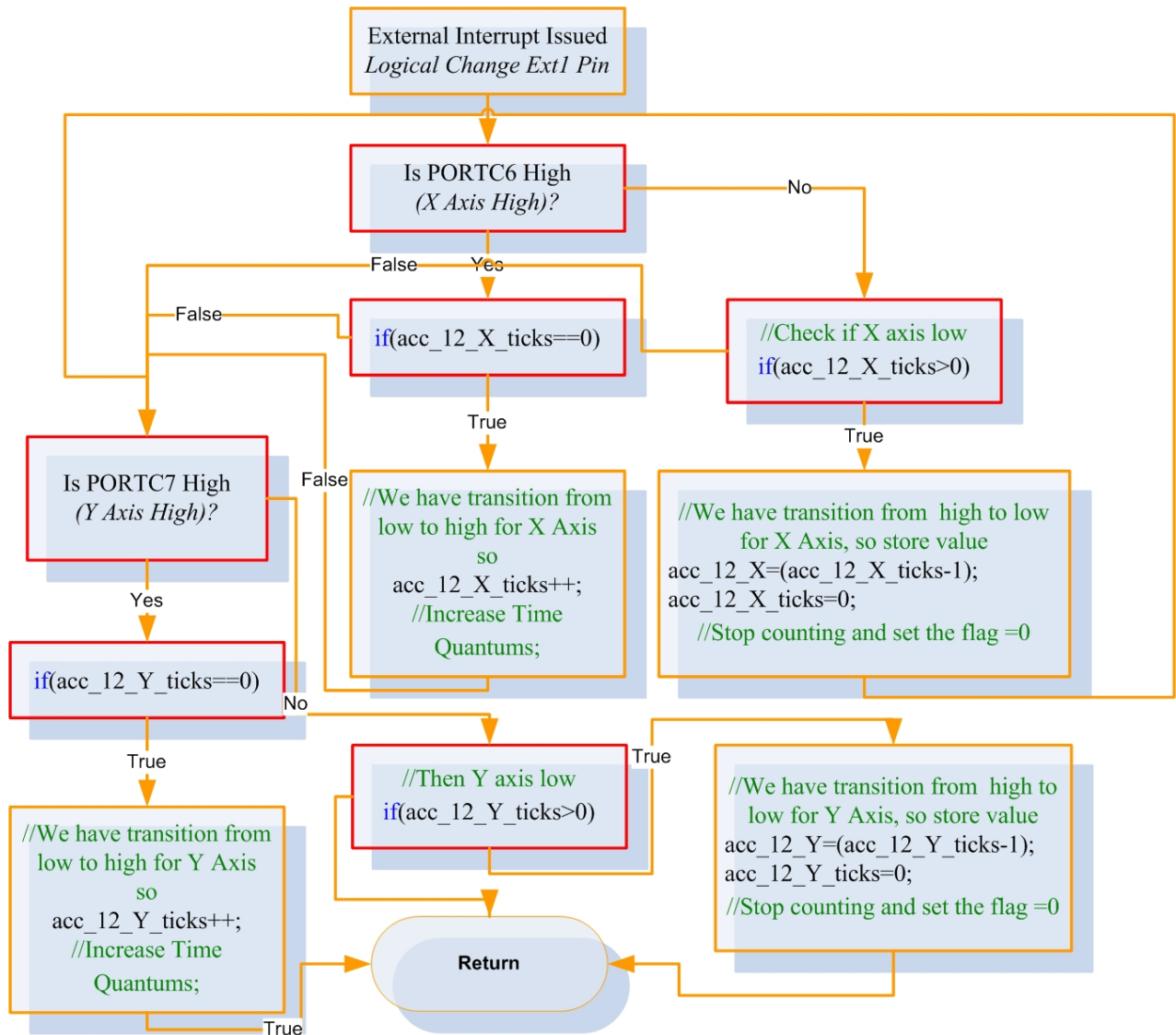


Fig. 4.13 Flow chart for measuring the duty cycle of each axis for the ADXL213 (rectangles in red line are for decision, more convenient than conventional rhombus).

The routine which is responsible for the increase of the time quantum of each counter is displayed in Table 4-3. That part of the routine is executed every 16usec as explained earlier. If the counters for the time quantum are greater than zero, which means that a counting sequence has been triggered by the External Interrupt routine (logical transition from low to high), then this would initiate their further increase. Whenever a logical transition from high to low takes place, the quantum counter is stored at a variable that is fed though the serial to the SBC.

ADXL311 and ADXL213 bandwidth selection

One major issue for the accelerometers was the selection of the bandwidth of operation. Initially, we had chosen the bandwidth of 50Hz in order to have high resolution variability while retaining the measurement error in a low lever. After our first real environment evaluation (on the vehicle with working engine) run, the acceleration measurements were very enlightening concerning the selected bandwidth. Vibration from the engine (two stroke single cylinder engine) and the anomalies from the ground in combination with small size of the vehicle corroded the measurements from the accelerometer. Although both types accelerometers have provisions for bandlimiting the measurements and can achieve low pass filtering for antialiasing and noise reduction they were physically constrained by the harsh environment of operation. Fig. 4.14 shows static vibration sampled from the ADXL213 w.r.t. time with the bandwidth of the accelerometers set at 50Hz (we had chosen 0.1uF capacitors). 50Hz is higher than the frequency operation of the engine in the lowest speed, which is about 800RPM which is translated as $80/6 \approx 13$ rotations per second, translated to 13Hz. It is clear from the figure that we should narrow the bandwidth. Thus, we set the bandwidth at 10Hz. The outcome was a lower vibration measured from the engine, but not satisfying at all.

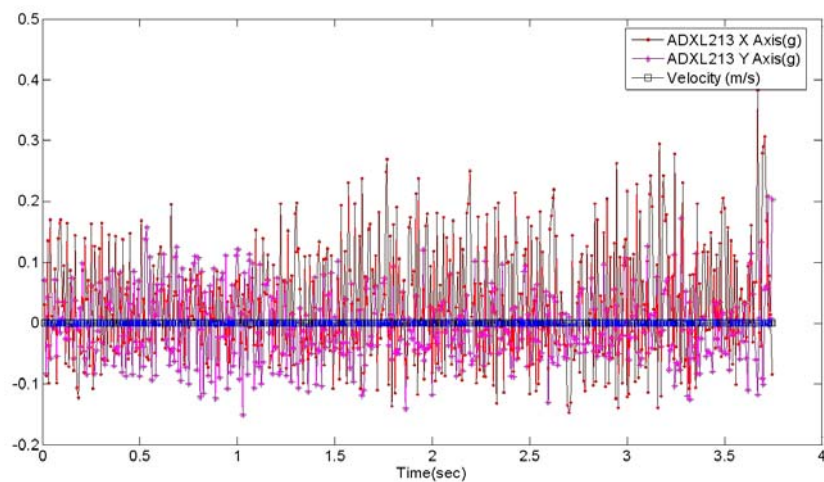


Fig. 4.14 Static acceleration measured by ADXL213 with engine working and bandwidth set at 50Hz.

We concluded that in order to have a valid measurement, we had to bandlimit lower than 3Hz. Therefore we used 2.2uF capacitors for the filtering which results in

$$F_{-3dB} = \frac{1}{2 \cdot \pi \cdot (32k\Omega) \cdot C_{(X,Y)}} = \frac{1}{2 \cdot \pi \cdot (32 \cdot 10^3) \cdot (2.2 \cdot 10^{-6})} \approx 2.26Hz$$
 bandwidth. A typical output of the ADXL213 accelerometer with 2.2uF capacitors is illustrated in Fig. 4.15.

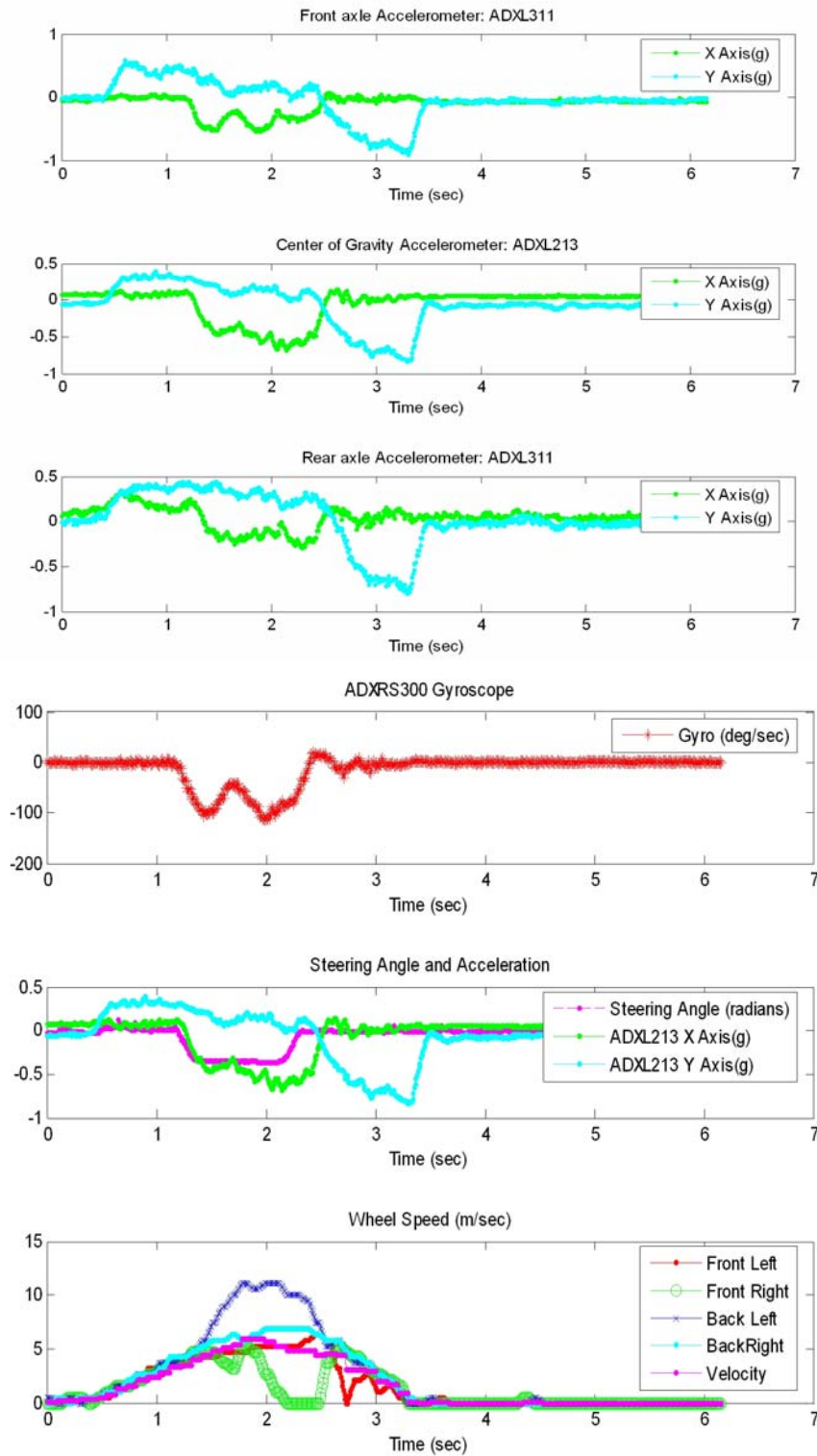


Fig. 4.15 Subplots of the same Instance (an auto generated experiment). Top three subplots are the acceleration measured by the front, center of gravity and rear accelerometer respectively. The bottom three subplots are the yaw rate measured by the gyroscope, the steering angle collated with the acceleration measured by the center of gravity accelerometer and finally, individual wheel speed respectively.

It is obvious from Fig. 4.15 that the noise from the engine is filtered whilst the resolution remains still in adequate level at the bandwidth of the 2.6Hz (Fig. 4.15). Edges on the measured acceleration still exists, but those are real moments of acceleration that the vehicle experiences and are due to bumps on the road, the vibration of the whole chassis from the movement etc. We could filtered the signal even further but the design trade – off between noise and resolution, would favour even further the low noise, but would increase the resolution at a level where the measurement wouldn't be any useful.

Observations, problems and possible improvements on the interface between the accelerometers and the microcontroller

ADXL311

The output values from the ADXL311 for static vehicle (engine shut off, nothing moving except for the fans of the computer and power supply) present an oscillation $\pm 0.03g$. This phenomenon is justified by:

- We do have $\pm 1/2$ bit quantization error for the analog to digital conversion operation. As voltage reference for the ADC we have set $V_{dd}=5V$ which is represented by 1023 or $2^{10}-1$ (10bit ADC resolution).
- The formula for the estimation of acceleration is $A = \frac{(V_{dd}/2) - V_{out}}{0.312}$ where $V_{dd}=5V$ ([17]). The variation from +2g into -2g is translated into analog output 1.876V and 3.124V respectively. That means that we have $3.124V - 1.876V = 1.248V$ to determine the range of 4g of acceleration, which is translated as 0.312V/g sensitivity. The maximum resolution with 10bit ADC is 5mV with 5V as voltage reference. Thus, maximum resolution for acceleration measurement is $(5mV/g)/(0.312V/g) \approx 16mg$.
- The voltage varies from 4.96V – 4.99V, although it is regulated and filtered and we have used this varying voltage as voltage reference. This variation decreases the resolution even further. Supposing the V_{out} from the accelerometer is constant while the V_{dd} varies, we would come to:

$$error = \left| \frac{(4.99/2) - V_{out}}{0.312} - \frac{(4.96/2) - V_{out}}{0.312} \right| = \left| \frac{(4.99/2) - (4.96/2)}{0.312} \right| = 0.048g$$

If we take the worst case scenario from the above we conclude to the oscillation presented is normal due to the ADC interface. In order to improve our measurements we have defined a function of weighted sums from the previous samples measured at the software running at the SBC (Table 4-4).


```
#define forgetting_sum(last,previous) (double)(last*0.4+previous[0]*0.3+previous[1]*0.15+previous[2]*0.15)
```

Table 4-4 Forgetting sum definition.

The weighted sum uses the current at the last three samples for smoothing out the variations from the whole process. Without deteriorating the real value of the samples, through this way we have diminished the oscillation from $\pm 0.03g$ to $\pm 0.02g$.

Outlining the above process, although the interface of ADXL311 with the AVR was easy, it would have been better, if it used a communication protocol less prone to mismeasurements and sensitive to voltage variation, like Serial Peripheral Interface (SPI). A precision voltage reference as voltage source for the accelerometer would improve the situation, but wouldn't eliminate the poor communication. In applications like ESC, the precision of the measurements is marginally sufficient and more robust algorithmic solutions should be found, that would take into account this uncertainty, like an adaptive ESC algorithm.

ADXL213

The ADXL213 accelerometer, whereas much more expensive than ADXL311 and theoretically much more precise didn't perform the maximum of its capabilities, due the restraints of the implementation. That is, the pulse T2 (Fig. 4.10) that we considered constant, in reality is not constant and should be measured too. The variation was smaller than the maximum resolution (time quantum equals to 16usec) but does exist and does worsen our measurements. We have an oscillation here too, which is about $\pm 0.02g$. We were able to shorten the time quantum but this would consume even more resources from our microcontroller and since the resolution was good enough for our application, it was left this way. Of course, a more robust communication protocol (like SPI or I2C) would be welcome here too.

Side by side comparison of ADXL311 and ADXL213

The interface between both accelerometers ADXL311 and ADXL213 was quite interesting and at the same time very demanding. The output of the sensors, analog and duty cycle modulated respectively made the whole process a trade off between speed, precision and resources of the microcontroller. Although the sensors do have divergence between the measured data and the characteristics proclaimed in the data sheets (might be just because of our implementation) they are quite functional, do have very good documentation and are really robust both in operation and construction. Although the LCC packages that the accelerometers were housed were soldered directly to a DIP8 socket, they remained intact, despite the fact that they are really tiny.

Steering Angle Estimation

Steering angle estimation is conducted via variable resistance. An articulated mechanism moves the shaft of a potentiometer, with motion derived from the steering axon (Fig. 4.6 and Fig. 4.16 left subfigure). The output of the potentiometer is driven to the ADC4 I/O line of the AVR (Fig. 4.16 right subfigure). Because the voltage generated is not proportional of the actual steering angle ([14]), we have sampled enough voltage values with respect to the actual angle of each wheel. These values are presented on Table 4-5.

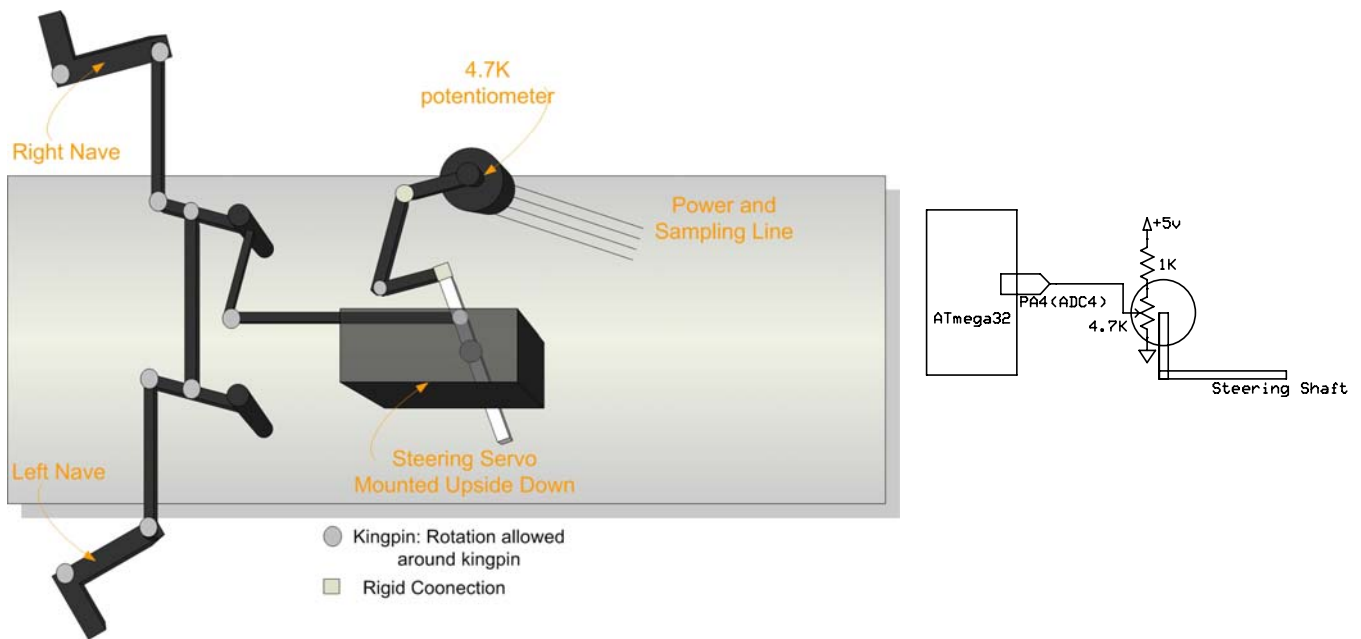


Fig. 4.16 Provision for estimating the steering angle.

Steering Angle Estimation															
Voltages (Volt)	1,924	1,943	1,953	1,987	2,04	2,051	2,09	2,104	2,139	2,163	2,183	2,212	2,246	2,266	2,3
Steering Angle (degrees)	21	19	18	14,5	10,5	8	4	0	-3	-5	-8	-10	-12,5	-14,5	-19
Real values															
Estimated Values by polynomial	20,99	19,2	18,2	14,7	8,8	7,5	3,1	1,4	-3	-5	-7	-10,4	-14	-15,32	-18
Final Values Used	19,99	18,2	17,2	13,7	7,8	6,5	2,1	0,4	-4	-6	-8	-11,4	-15	-16,32	-19

Table 4-5 Voltages measured by the system for steering angle estimation.

With the built in function of MATLAB, *polyfit* (uses LMS to fit values to a polynomial), we have derived to a third degree polynomial of angle with respect to (w.r.t.) voltage. The actual and estimated angles are quite close (Fig. 4.17 and Table 4-5). The polynomial is the following:

$$steering(volts) = 10^3 \cdot \left(\begin{array}{l} 0.31570260254613 \cdot volts^3 - 1.97189807334590 \cdot volts^2 + \\ 3.98997966117411 \cdot volts - 2.60471114838461 \end{array} \right)$$

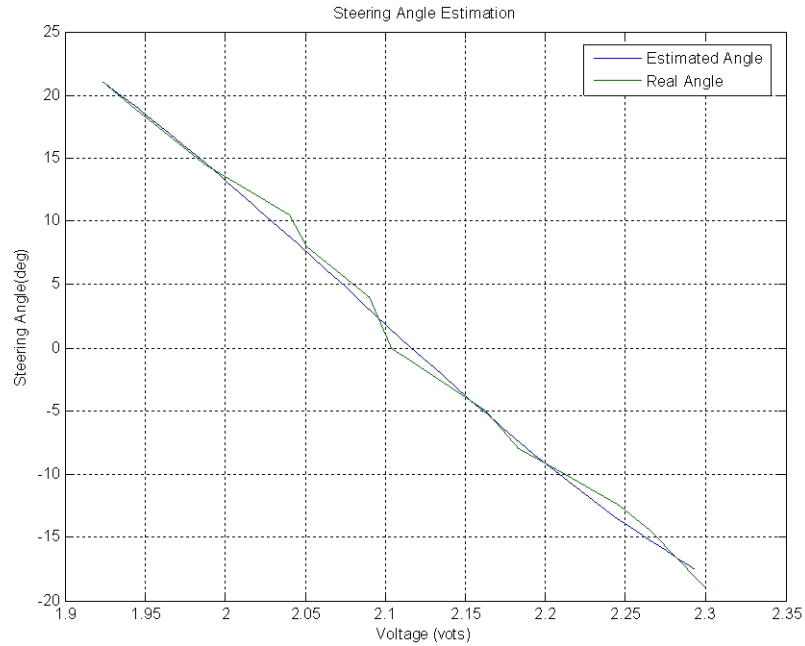


Fig. 4.17 Comparison between the real angle and the estimated by the 3rd degree polynomial.

The use of variable resistance for estimating the steering angle is not the best solution. The main drawback is the robustness of the system. The resistor with time passing will wear out and change its value. The resistance also depends on the temperature. So the outer temperature can quite easily give us two different measurements for the same crack shaft position. The steering angles presented on Table 4-5 are the average turn of both wheels. For a vehicle turning with low speed, in order not to experience reversely forces on the tires, the inner of the turn wheel must be turned by $\delta_{inner} = \frac{R}{L - h/2}$ and the outer by $\delta_{outer} = \frac{R}{L + h/2}$ ([14]), where R is where the desired radius of turn, L is the wheelbase of the vehicle and h is the space between wheels. The average of those two angles is the Ackerman Steering Angle.

Different solutions more credible than a potentiometer have been proposed and are used by automotive industry. The most widespread solution are optical and magnetic type sensors ([21],[22]). An optical sensor can detect very small angles, but the cost for installing such a sensor is high and is very sensitive to harsh environment. That means it requires very good shielding for temperature and stain ([21]). Fig. 4.18 from the presentation of [23] shows the principles and disadvantages of typical angle sensors.

Another very common provision for rotational angle estimation is imploded with the use magnetic sensors. They can achieve contact less position sensing providing the capability for measuring distance, angle, rotational speed etc ([23]). The main advantage is that they work even under the most unfriendly, grubby environmental conditions ([21], [23]). The previous mentioned basic advantages have resulted in the widespread use of magnetic sensors in contact less position detection ([23]). Typical magnetic sensors are conventional magneto resistors and Hall Effect sensors. Both of them have high environment resistance and are prone to air gap deviations ([21], [23]). So different methods for increased accuracy have been proposed like, multipole magnet sensors ([21]) or sensors based on the giant magneto resistance (GMR) effect ([23]).

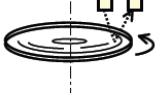
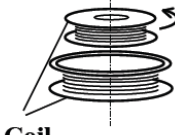
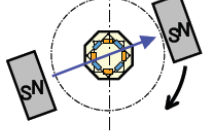
	Optical sensor	Resolver	2-phase type MRE
Principle	 <ul style="list-style-type: none"> * Barcode pattern reading 	 <ul style="list-style-type: none"> * Induced voltage * Two-phase detection 	 <ul style="list-style-type: none"> * Magnetic field rotation angle * Two-phase detection
Disadvantage	<ul style="list-style-type: none"> * Sensitive to high temperatures * Axis dislocation * Sensitive to staining * Expensive 	<ul style="list-style-type: none"> * Large size * Expensive 	<p>Impossible to detect 180 to 360 degree rotation angles</p>

Fig. 4.18 Principles and disadvantages of angle sensors, from presentation of [22].

The reason we used a simple potentiometer as an angle sensor was to achieve our goal to limited cost. A better resolution for measurements wouldn't benefit our design, since our main problem was mechanical lush which provokes a typical deviation of more than 1 degree in angle. This is due to the fact this is a scaled model with poorer assembly than a typical vehicle, where the magnitude of the mechanical lush, is multiplied by the scale of the vehicle. That means, if we have 2 mm lush between the steering system and the steering shaft, on a full scale vehicle it would provoke a (lets say) 0.2° wobble on the steering wheel where it would provoke 1° on the model.

Wheel angular velocity

Wheel angular velocity is computed with four SS433 series Hall effect sensors from Honeywell. More specific we have used the SS443A unipolar Hall effect sensor. When exposed to the appropriate magnetic flux, SS443A sinks its output ([24]).

The digital output of the Hall Effect sensor is open collector type. That means that the output signal is applied to the base of an internal NPN transistor whose collector is connected (open) to a load ([25]). In our implementation this load is the power supply through a 10K pull up resistor, in order to limit the current sinking to the ground (Fig. 4.20 left). The emitter of the NPN transistor is connected grounded. The current flows from the load into NPN transistor (Fig. 4.19). When the Hall Effect sensor is exposed to the appropriate flux, leads the internal NPN transistor to an on state, so the output of the sensor, the base of the transistor, is conducting current to the ground. That means that we have a logical zero. On the other hand, when the internal transistor is off, the output is connected to the load.

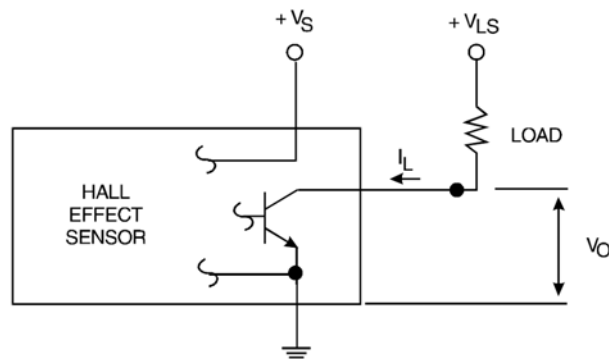


Fig. 4.19 NPN output Hall Effect Sensor. Figure from [25].

So in order to limit even further the current flowing from the load to our sampling circuit and to restore the voltage at the output back to original TTL levels we have used a XOR 74LS86 gate (Fig. 4.20).

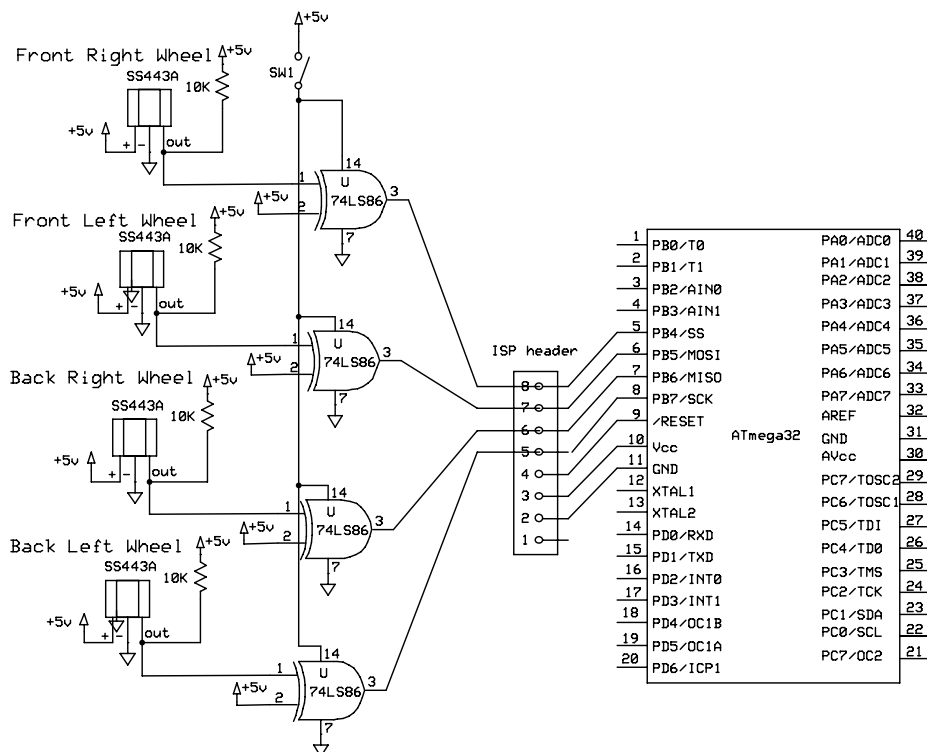


Fig. 4.20 Hall Effect circuitry

This circuitry provision was designed that way, in order to limit the current flowing from the source to the ground (limit short circuit as much as possible). The connections to the microcontroller from the Hall Effect Sensors through the XOR gate are exactly as illustrated on the above figure (Fig. 4.20). The most inquisitive part of the above figure is the switch (SW1) for the power supply of the XOR gate. The explanation is simple. The output from the XOR gate is fed to PB4 – PB7 pin from PORTB of the ATmega32. Those specific ports have dual purpose. The first is that they are general purpose digital I/O lines and the second is that they are the pins used for the In Circuit Programming (ISP) of the microcontroller. ATmega32 has the capability to be programmed in circuit using SPI Serial Downloading to both the program flash and EEPROM memory. The serial interface consists of pins SCK/PB7 (serial clock), MOSI/PB5 (input) and MISO/PB6 (output) ([20]) which in our case are driven by the XOR gate. To ease our development and not have to remove the microcontroller in order to put into the development board for programming (STK500) and then back to the PCB, we did put a pin header connecting necessary pins for the ISP which is conducted through a cable coming from the development board. The isolation between the XOR gate and ISP cable was necessary. Therefore, in order to program the microcontroller we have to manually power off the gate. More details about the programming on the “Microcontroller: Schematics, PCB and Development Tools” section.

To estimate the angular velocity we have attached 8 reed relay magnets at each rim (Fig. 4.21). Whenever a magnet passes close a Hall Effect sensor sinks its output. At Fig. 4.20 we can see that the output is directed at the first input of a XOR gate. The second input is hardwired to a TTL logical “One” level. So we have two states:

- Magnet in front of SS443A
 - Output sinked ➔ Output of XOR gate driven from sensor at logical “One” (One input of XOR gate is “One” and the other is “Zero”).
- Magnet away from SS443A
 - Output at logical “One” ➔ Output of XOR gate driven from sensor at logical “Zero” (Both inputs of XOR gate are at logical “One”).

The outputs from the XOR gates at Fig. 4.20, besides to the I/O lines of the ATmega32 PB4 – PB7 are fed to second circuitry of XOR gates. The main reason we did this, is because we had to measure the logical transitions for each Hall Effect Sensor through the XOR gate provision illustrated at Fig. 4.20 within a certain amount of time. The best way to do that was to build a similar to ADXL213’s measuring scheme (Fig. 4.12), using one I/O external interrupt and three XOR gates.

The angular velocity is updated every 100msec, delivering $2\pi/(8 \cdot 100\text{ms})$ rad/sec or 0.4625m/s (longitudinal speed with 11.7cm wheel diameter) resolution for each wheel. A timer counter interrupt measures the logical transitions for every sensor within a certain amount of time.

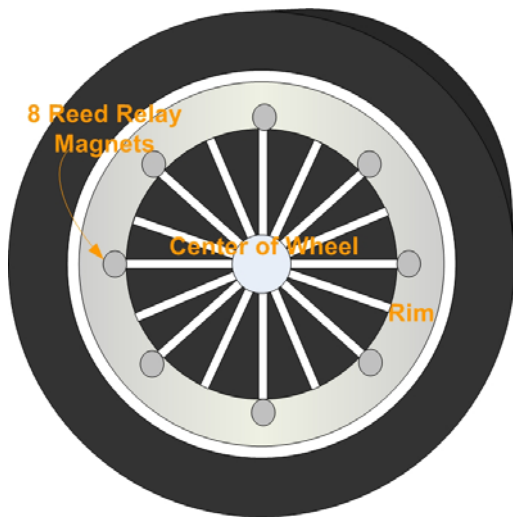


Fig. 4.21 Eight Reed Relay Magnets attached at inner side of the rim.

The final circuitry with external interrupt driven input for the microcontroller is schematized at Fig. 4.22.

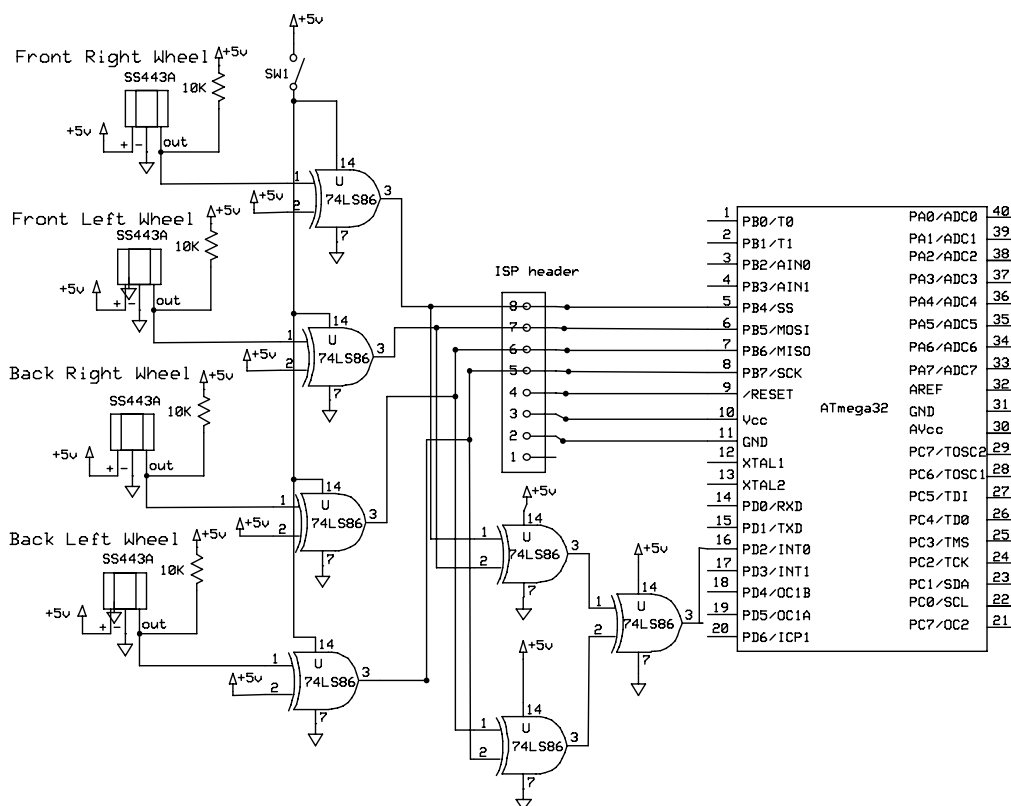


Fig. 4.22 Complete circuit for estimating angular velocity of individual wheel.

We have programmed the microcontroller to signal an Interrupt at any logical transition of the External Interrupt 0, PD2/INT0 pin (Fig. 4.22). Thus, whenever we have a state change at the output of the XOR gate driven by the Hall Effect sensor, we also do have a state change at the XOR gate which is fed at the PD2/INT0. When a interrupt is signalled, the routine responsible for estimating the rotational speed checks the inputs at ports PB4 – PB7, compares it with the previous state (when the last interrupt was signalled) and if it detects a state change to any of the inputs, increases a counter. At Table 4-6 we can see part of the code responsible for the estimation. The following code is well commented and the name of the variables show their origin and use.

```
SIGNAL(INT0_vect)
{
    temp=PINB;
    if (temp&0b00010000){
        if ((previous_state&0b00010000)==0)
            front_right_ticks++;
    }

    if (temp&0b00100000){
        if ((previous_state&0b00100000)==0)
            front_left_ticks++;
    }

    if (temp&0b01000000){
        if ((previous_state&0b01000000)==0)
            back_right_ticks++;
    }

    if (temp&0b10000000){
        if ((previous_state&0b10000000)==0)
            back_left_ticks++;
    }

    previous_state=temp;
}
```

Table 4-6 External Interrupt 0 routine. Estimation of state change at the wheels.

The only purpose the above routine checks for the previous state and doesn't increase the clock "ticks" at TTL high voltage level, is because a magnet might stop in front of a sensor thus producing a continuous logical "one", which means no revolution. The flow chart diagram of the routine responsible for the handling of the external interrupt is presented at Fig. 4.23.

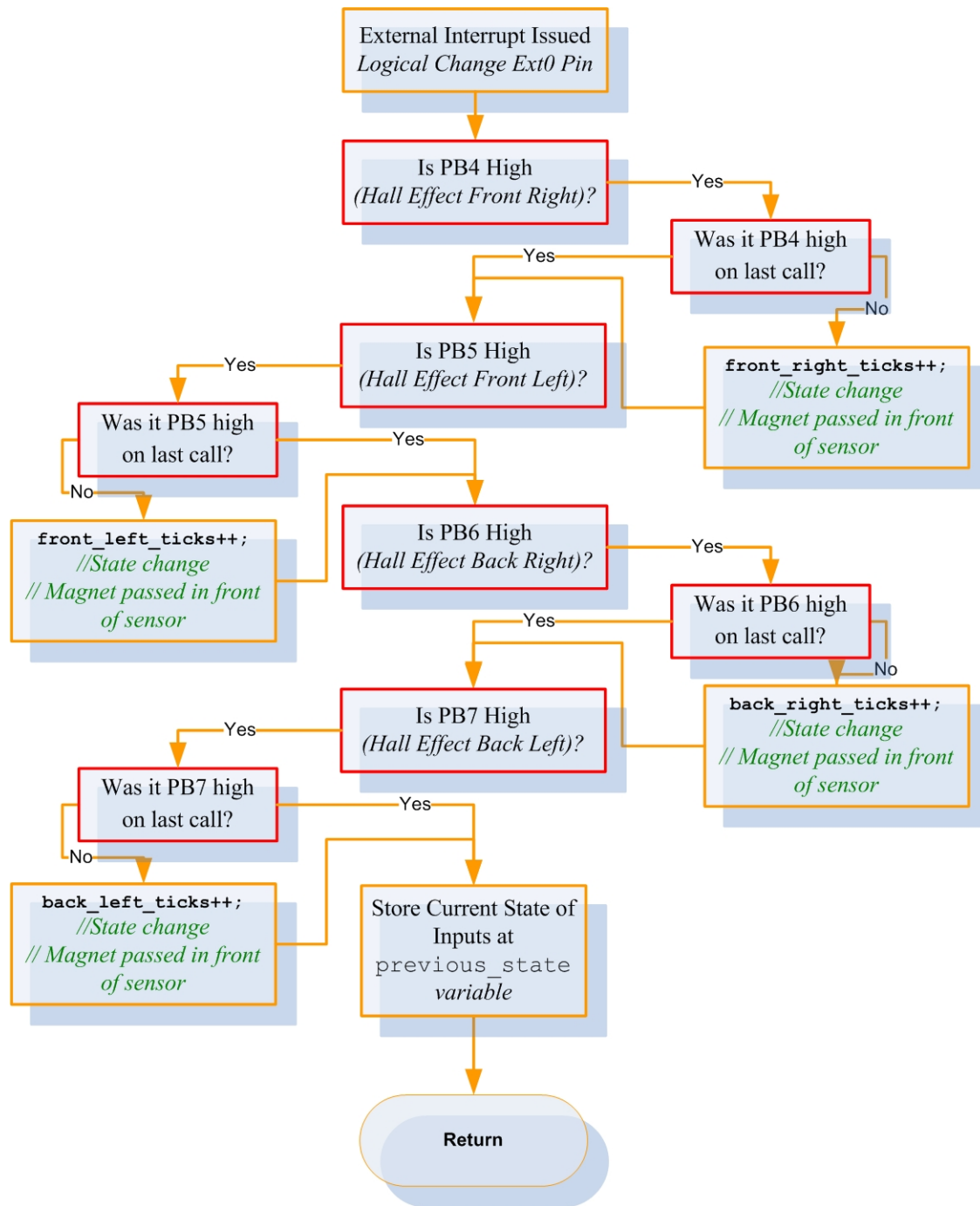


Fig. 4.23 Flow chart of External Interrupt handling routine 0.

After 100msec the interrupt driven routine (Table 4-7) based on timer counter 2 checks how many “ticks” we had for each wheel, stores the values and nullifies the counters to start over the process.

```

/* Timer/Counter2 Overflow */
SIGNAL(TIMER2_OVF_vect){
    wheel_big_counter++;
    if(wheel_big_counter>781){
        781*128us=100000usec=0.1sec
        front_left_speed=front_left_ticks;
        passed 1562*128us=200000usec
    }
}

```



```

front_right_speed=front_right_ticks;
back_left_speed=back_left_ticks;
back_right_speed=back_right_ticks;
wheel_big_counter=0;                                     // Start Counting Again
front_left_ticks=front_right_ticks=back_left_ticks=back_right_ticks=0;
}
}

```

Table 4-7 Timer/Counter2 overflow routine (part of it).

More details on the function of the routines presented and general the firmware, on the software section.

Yaw rate estimation

Yaw rate estimation is conducted through an ADXRS300 gyroscope from Analog Devices. It is a $\pm 300^\circ/\text{s}$ yaw rate gyro with signal conditioning. It uses a surface-micromachining process from Analog Devices Embedded with all the required electronics on the chip in order to be a fully functional low cost angular rate sensor ([26]). ADXRS300 has analog output, a voltage proportional to the angular rate about the z axis to the top surface of the BGA package (Fig. 4.24) and has $5\text{mv}/^\circ/\text{sec}$ sensitivity. The output is fed directly to an ADC line for estimation.

The user can set the bandwidth by an external capacitor and lower the scale of the measurements with the use of a single resistor. In order to compensate for the miss measurement and adjust the readings, the gyro provides a precision voltage reference and a temperature output too. That means, that the user may read both, the output from the sensor and the precision voltage which has a known precise value and through those two fix the real value. It also has an electromechanical provision (self test) that excite the sensor for proper operation ([27]).

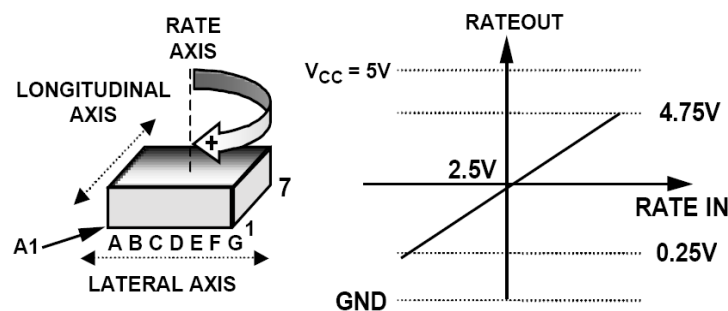


Fig. 4.24 ADXRS300 rate axis orientation from data sheet ([27]).

The operation on most MEMS gyroscopes is based on Coriolis force. The typical provision for angular rate in MEMS gyroscopes consists of an inertial element mounted on elastic suspension with two degrees of freedom (Fig. 4.25, [28]). The element, often called sensitive element, mounted on the springs is driven to oscillate on the primary axis, (Fig. 4.25) with prescribed

amplitude thus a known impulse response. When the sensitive element rotates around a particular axis, which in the case of the ADXRS300 is the z axis to the top surface of the BGA package, the oscillating mass experiences coriolis force, which causes the mass to oscillate on the secondary axis thereupon in different mode. So the yaw rate information lies in these oscillations in contrast with electromechanical gyroscopes where the yaw rate information is extracted from nonharmonic or angular displacements. The primary motion that the sensitive element driven, might be rotary and not necessarily oscillatory ([28]).

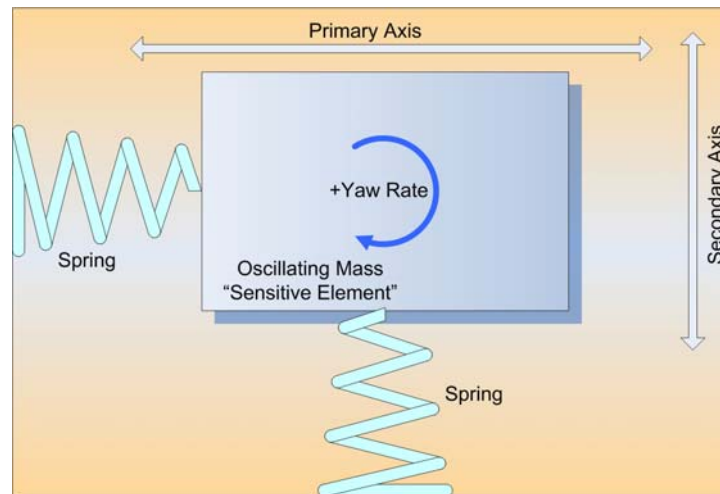


Fig. 4.25 Theory of operation for vibrating MEMS gyroscopes [28].

The ADXRS300 through built in provisions has the ability to preserve signal integrity and with the usage of a dual sensor regime is able to reject external g – forces and vibrations. At Fig. 4.26 we can see the recommended by the manufacturer connections for the ADXRS300 from the data sheet. The bandwidth is set by the capacitor C_{OUT} and resistor R_{OUT} (Fig. 4.26). R_{OUT} has been trimmed during manufacturing to be $180K\Omega \pm 1\%$ ([26]).

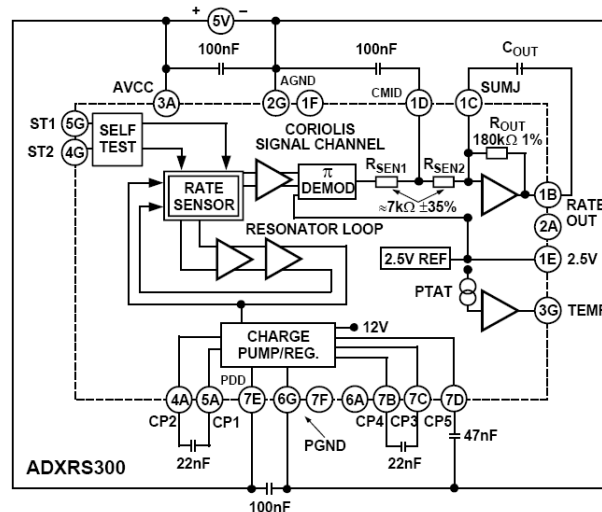


Fig. 4.26 Recommended connections for the ADXRS300 from the manufacturer's data sheet ([26]).

The formula for setting the bandwidth is $F_{-3dB} = \frac{1}{2 \cdot \pi \cdot R_{OUT} \cdot C_{OUT}}$ ([26]). As mentioned above the original package of the ADXRS300 is a tiny BGA (bald grid array), thus we had to use a breakout board. We chose to buy a breakout board from Sparkfun Electronics ([29]) which was suitable for our application (Fig. 4.27).

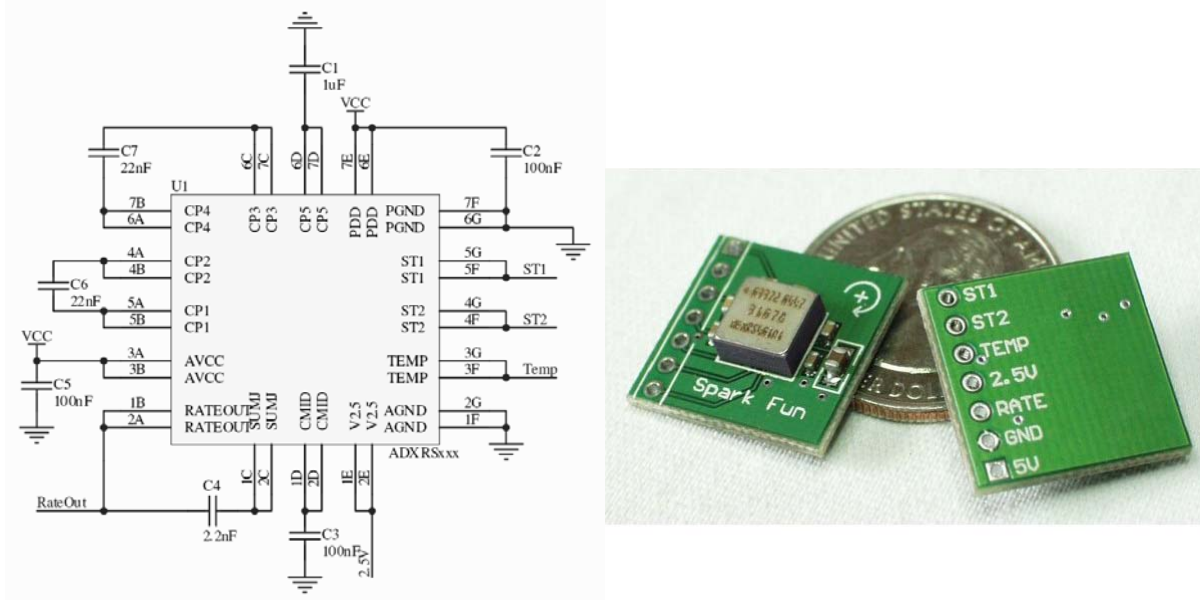


Fig. 4.27 ADXRS300 breakout board from SparkFun Electronics. Schematic from data sheet left ([27]) and picture from website right ([29]).

The C_{OUT} capacitor was originally set at the value of 2.2nF (Fig. 4.27) which is the recommended value from the manufacturer too, and with the formula presented on the previous paragraph results to 400Hz bandwidth.

The output from the gyro is analog, proportional to the yaw rate and has 5mv/°/sec sensitivity. The 0°/sec represented as 2.5V output. That means that the maximum rate measured, 300°/sec is represented by an offset from the 2.5V of magnitude which is: $300 \text{ °/sec} \cdot 5 \text{ mv/°/sec} = 1.5 \text{ V}$, resulting to a 4V output.

The functionality of the accelerometer is exemplary. The characteristics were met exactly as asserted by the manufacturer. The only paradox with the sensor was that it didn't suffer a drift, except in the case of limited available current. Even though the gyroscope wasn't current consuming did suffer from drift when it was supplied together from the same power source along with another device that would draw high current, for example the single board computer. The previous is the main reason that we used a different battery (600mAh, 12V Nickel – Cadmium, see Fig. 4.5) for the microcontroller and the sensors. This phenomenon did also occur when we had low voltage on the battery, which was below the nominal (less than 11V) which obvious means that the battery was with very little power. Although the rest of the system would continue

working until it would go completely out of power, the gyroscope provision was the first to suffer from the current drain. We can see a typical output from the sensor at Fig. 4.15.

Driver's commands

Driver's Command from the transmitter are delivered to the receiver and sampled from the microcontroller. We have used a standard FM 2 channel radio system from Acoms. The output of receiver is a pulse width modulated 50Hz signal. It is the common interface used by standard model radio system ([30]). The position of the servo is determined by the duty cycle of the signal (Fig. 4.28). The output of the receiver is sampled in a satisfactory manner, every 32us. The period of the pulse is 20ms. Thereupon, the 32us sampling period have as an outcome $20\text{ms}/32\mu\text{s}=625$ time quantums. Since the information about the position lies between in the duty cycle of the signal and the length of the high to low pulse fluctuates between 1ms (-90° position) and 2ms (90° position) we have $1\text{ms}/32\mu\text{s} \approx 31$ time quantums to encode a 180° angular position which is translated as 6° angular accuracy.

The connection scheme used between the receiver, the microcontroller and the servos (throttle and steering) is illustrated at Fig. 4.29.

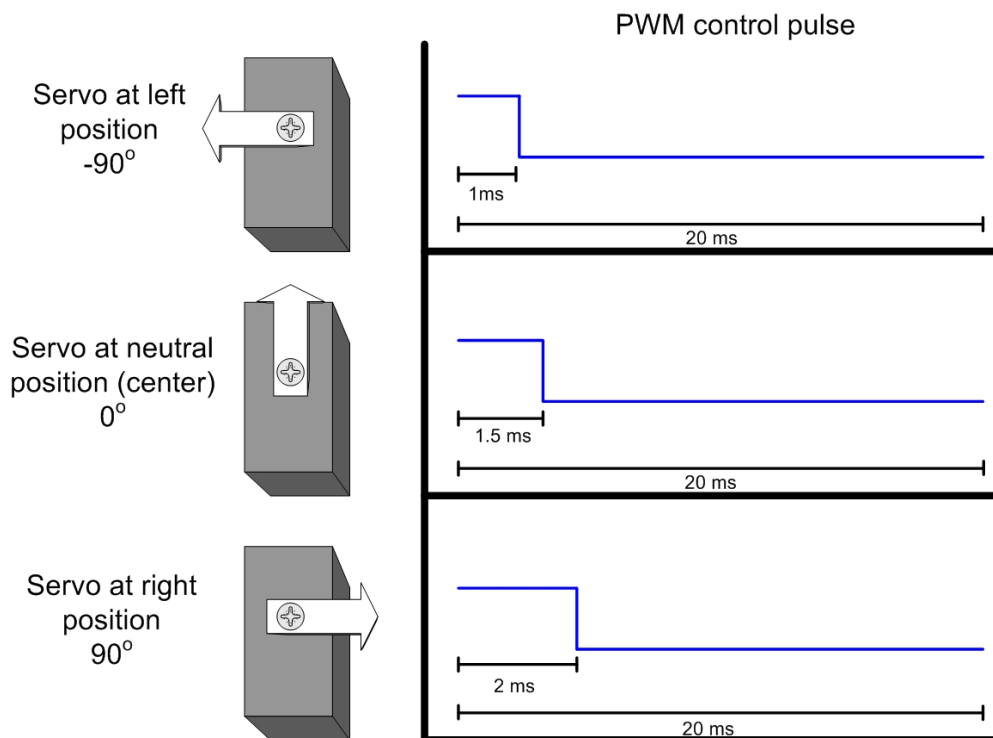


Fig. 4.28 Standard model servo PWM control.

The signal from the FM receiver is sampled by the microcontroller at PB2 and PB3 I/O pins of PORTB (Fig. 4.29) for the throttle and steering servos respectively. The final control signal

arriving at the servos comes either directly from the receiver or from the processed signal originating from the AVR. The user can manually switch the origination of the signal. This double signal feeding is done for safety and practicality.

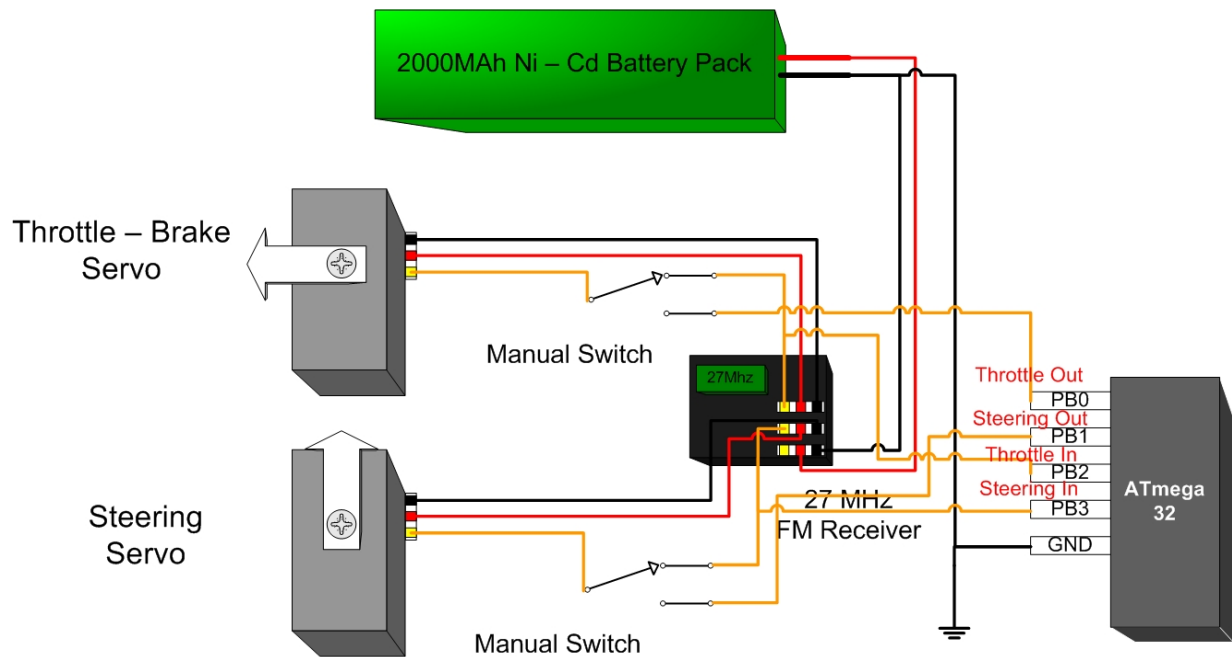


Fig. 4.29 Radio system connection scheme.

Why safety? Consider the situation that we want to conduct an automating driven experiment. For example; we can program the model to steer left after it reaches a certain velocity and hold that steering for a programmed amount of time. In a custom made system, many things can go wrong, like the program controlling the vehicle from the SBC to crash or the regulator supplying the microcontroller to fail (has happened to the author at the past) and many other undesired situations. So it would be better for the user to keep at least one channel of operation totally manual. In the example of the automatic experiment just mentioned, the steering servo must be controlled by the program and the throttle – brake servo can be controlled directly from the user. Thereafter, the user can brake the vehicle in case something goes wrong. In a different experiment, the program might just need to use the throttle servo. Or finally we might need to conduct experiments that would need one, both or none steering or throttle servos to be controlled from the program. So the easiest and safer way was to use the wiring scheme of Fig. 4.29, since the user can change at will the signal source of the servos. And since the model originates from toy it is pretty easy to switch the source to the receiver, take off the SBC case and trim the vehicle or play without the slightest fear that something could go wrong.

At Fig. 4.29, we can see that we have used a different power supply from the microcontroller and the SBC. The explanation is simple. The servos have 18kg·cm and 5kg·cm and can draw more than 8A power from the battery and which can produce voltage spikes that can totally destroy the readings from the ADC of the microcontroller. Beside the fact that we have used power regulated power supplies it is generally recommended to use a different power source for situations like the one mentioned.

Whether the final signal arriving at the throttle – brake and steering servos comes directly from the FM transmitter or a processed version from the receiver, the driver's command, that means the signal coming out from the receiver, is fed to microcontroller sampled and is send to SBC with many more readings. The sampling scheme is easy and will be presented into the firmware section.

4.2.2 Actuators

Standard model servo mechanisms are the actuators for the platform. There are two servos attached to the nave of each of the left and right wheel correspondingly and another one for the rear axle brake; exclusive of the two factory installed servos. The servos at the front axle are two ACOMS AS -12 standard servos with 3.0kg·cm at 4.8 Volt. The servo responsible for braking the rear axle is a Hitec HS – 311 with 3.7kg·cm with rotational speed 60°/0.15sec at 6V supply (Fig. 4.30).



Fig. 4.30 Hitec HS – 311 standard model servo used from the braking of the rear axle.

One major drawback of the design is the usage of standard servos for the braking system. A real automobile, with ABS installed has hydraulic valves as actuators and the reaction time is tens time less than an electro mechanical servo ([31], [32]). The Fig. 4.31 is the portrayal of the typical braking system installed on a real vehicle with ABS. When the driver pressures the brake pedal, a brake servo mechanism (or more common the brake pump) increases the pressure on the hydraulic system delivering high pressured brake fluids to the brake pistons which are attached to the brake clippers. The more the pressure from the driver, the higher the pressure from the brake servo. In the case a wheel locks because of the braking and the electronic unit responsible for the

antilocking decides that this is an undesired effect, takes action by sending the command to release the brake pressure of this specific wheel. This is operation done by the relief valve which recycles some of the fluids, back to the central fluid tank which has as an outcome the wheel to unlock.

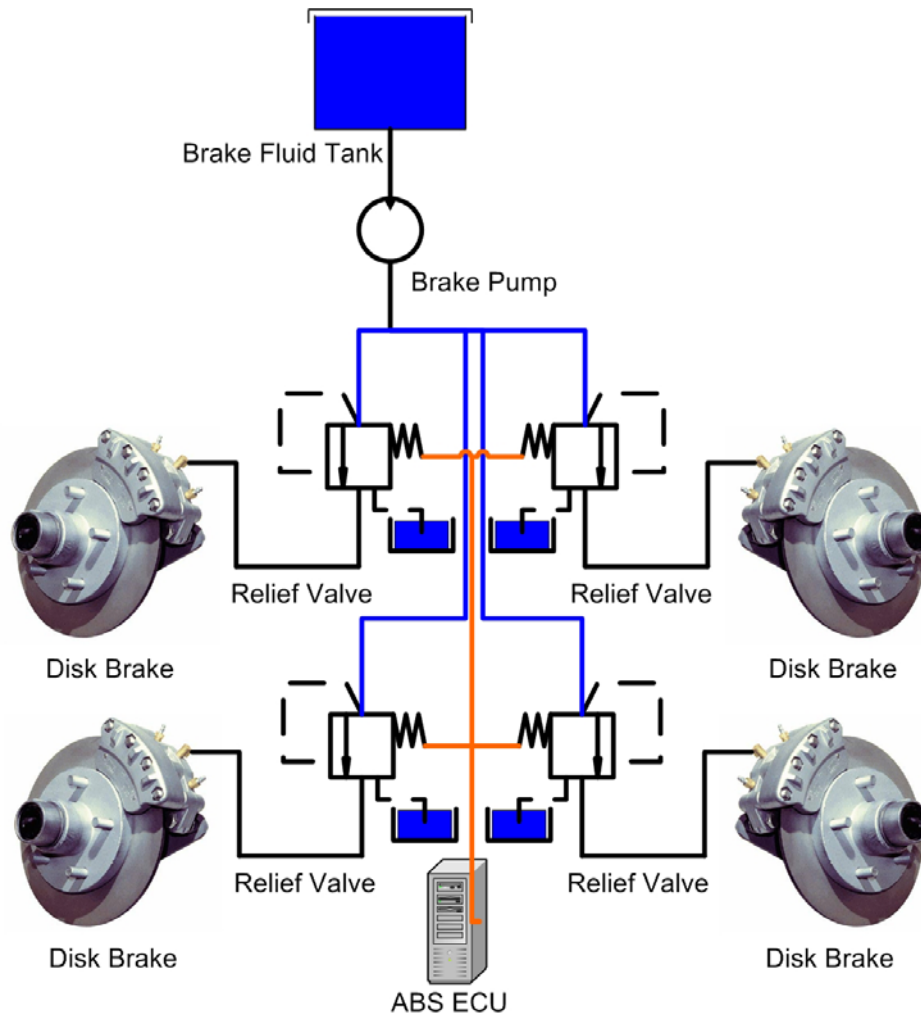


Fig. 4.31 Typical braking system of a real vehicle with ABS system.

The typical reaction time of a relief valve is some tens of milliseconds. Thus, it is obvious that an ESP system acting on real vehicle's hydraulic brakes has a great advance towards our implementation. The development and installation of a respective system on the model was more beyond our specifications and would raise the cost very high. So we had to suffice our needs with the servos, which finally did work fine and didn't let us down.

4.2.3 Power Supply

The schematic diagram for the power supply board installed on the rear plastic case of the model (Fig. 4.5) is illustrated below at Fig. 4.32. We have used one LM7806 and two LM317 voltage regulators and their required circuitry for operation. We have also attached a DIP switch on the PCB build in case we want to sollicitude a regulator or a pin header from power for power saving purposes, since their idle current is not negligible.

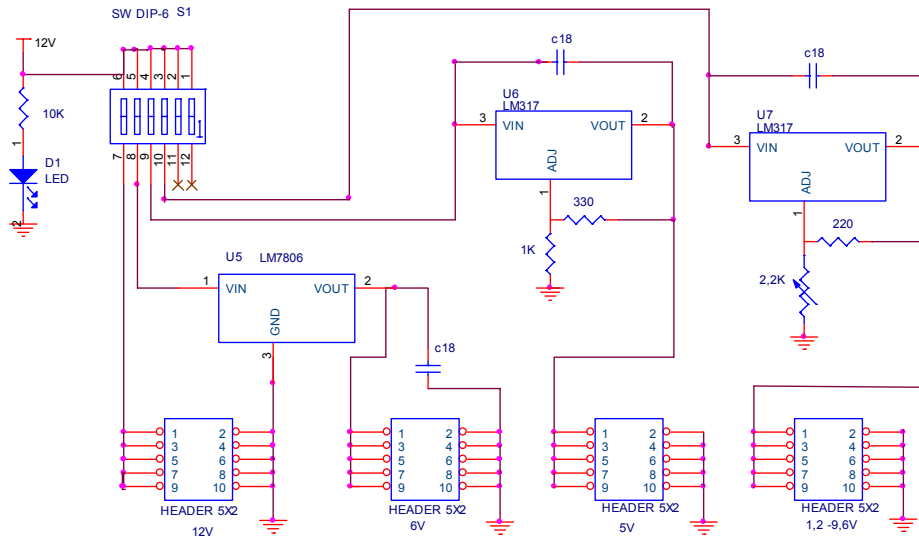


Fig. 4.32 Power supply schematic.

As illustrated on the above schematic we have one input source (a 12V battery) and 4 different voltages as outputs. The first output is directly fed from the battery, thus it has the same voltage level as the battery. The second derives from an LM7806 delivering 6V ([33]). The third output comes from an LM317. The output of the LM317 is adjusted ([34]) with the use of some resistors. Just following the directions from the data sheet we adjusted it so that we had 5V output, using one 1K and one 330Ω resistor. From this regulator and the 5V output respectively, we supply the microcontroller and the sensors. The fourth and last voltage output derives also from an LM317, but through a trimmer (onboard potentiometer) and by adjusting the resistance it delivers from 1.2V to battery voltage (12V). It has been implemented for potential future use.

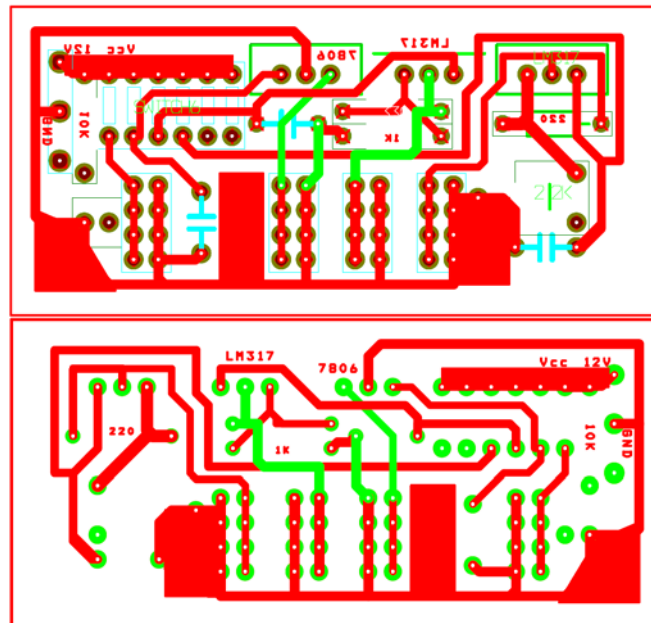


Fig. 4.33 PCB board for the central power supply. Top layer (up) and face of the copper – bottom layer (down).

The PCB built to house the above circuitry is shown at Fig. 4.33. The green lines appearing on the bottom layer are the second routing layer used. Those lines were drawn by hand as simple

wires. The use of second layer for routing was obligatory and it was easier drawing the wires manual than building a two layer copper PCB.

We have also used a second PCB for power supplies attached in the central plastic case housing the single board computer (Fig. 4.2). From this second power board, we supply the current for the Linksys router and also an Iogear Bluetooth serial modem device, which can be used when we remove the plastic case with SBC, router batteries etc from the vehicle and we need the data to be delivered wirelessly. The router requires a voltage of 3.3V and 2A current. A typical LM317 can supply current in excess of 1.5A. So we decided to use two LM317 paired together delivering 3.3V and current more than 3A. Because of the fact that the current drawn was high, we also used a light alloy heat sink and a small typical computer fan driving the hot air from the regulators away from the plastic case of the PCB out to the environment.

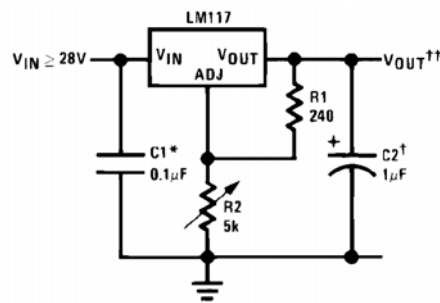


Fig. 4.34 Typical connection for the LM317 from data sheet ([34]).

The typical connection for an LM317 is illustrated on Fig. 4.34. The output for the above connection scheme is $V_{out} = 1.25 \cdot (1 + R_1/R_2) + I_{ADJ}(R_2)$, where the $I_{ADJ}(R_2)$ is almost zero. Therefore we chose $R_1 = 330\Omega$ and $R_2 = 560\Omega \rightarrow V_{out} = 1.25 \cdot (1 + 330/560) = 3.375$ Volts. The reason for choosing the previous values is because we wanted commercial available carbon resistors.

The schematic of the second power supply board is displayed at Fig. 4.35. For building the PCB, we did use a simple perforated prototype board, where we drew the wires manually.

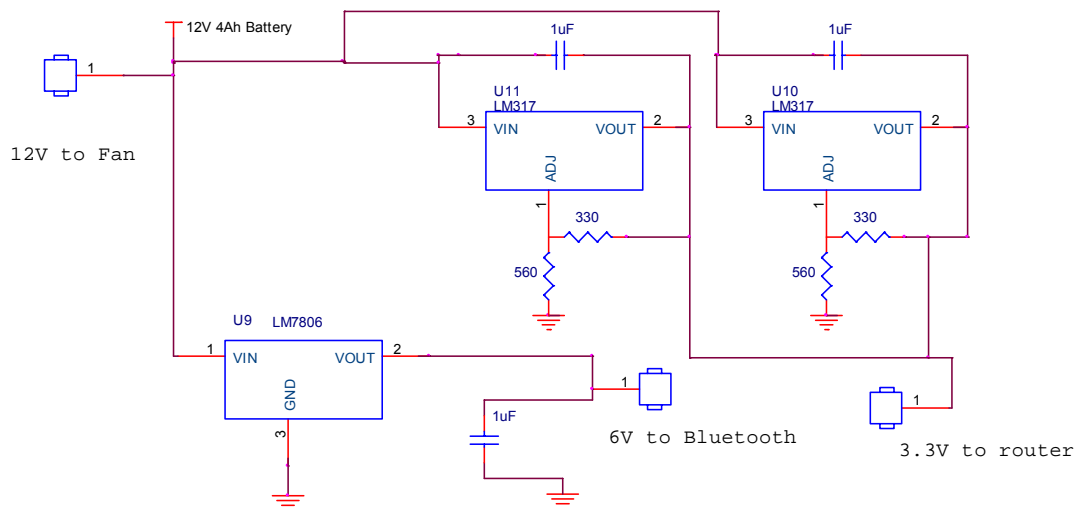


Fig. 4.35 Schematic of the router and Bluetooth serial modem power supply.

4.2.4 Microcontroller: Schematics, PCB and Development Tools

Schematics and PCB

The conductor of our whole implementation is an AVR ATmega32 (Fig. 4.36) microcontroller from ATMEL. ATmega32 is Reduced Instruction Set Computer (RISC) microcontroller connected to the serial port of the SBC; with 32Kb program flash and 2Kb RAM ([20]). The built in AVR core of the ATmega32 combines a rich instruction set with 32 directly connected to the Arithmetic Logic Unit (ALU), general purpose working registers. This architecture allows within one clock single instruction execution, two independent registers to be accessed, providing the capability of achieving ten times more throughput than the conventional architecture used in CISC microcontrollers ([20]).

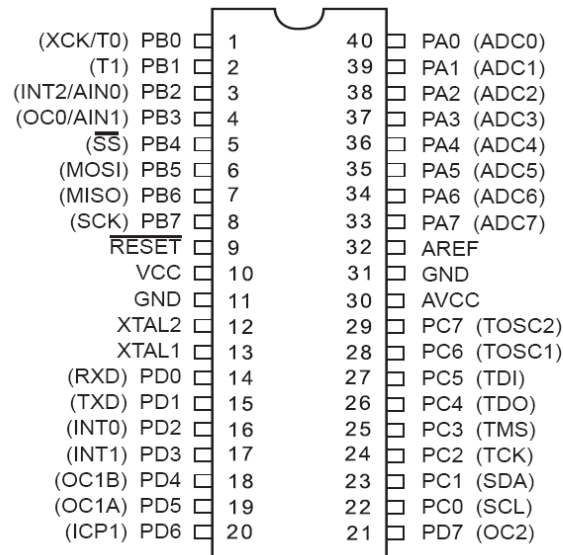


Fig. 4.36 Pinout Atmega32, PDIP package (data sheet: [20]).

The ATmega32 has the following provisions ([20]):

- 32K bytes of In-System Programmable Flash Program memory with Read-While-Write capabilities
- 32 general purpose I/O lines
- 32 general purpose working registers
- 2Kbyte SRAM
- 1024 bytes EEPROM
- A JTAG interface
- On-chip Debugging support and programming
- Three flexible Timer/Counters with compare modes
- Internal and External Interrupts

- Serial programmable USART
- Byte oriented Two-wire Serial Interface
- 8-channel, 10-bit ADC with optional differential input stage with programmable gain
- Programmable Watchdog Timer with Internal Oscillator
- SPI serial port
- Six software selectable power saving modes.

The ATmega32 AVR is supported with a full suite of program and system development tools including: program debugger/simulators like AVR studio ([36]), C compilers like Winavr ([15]), macro assemblers, in-circuit emulators, evaluation kits and development kits ([35],[36],[20]). More details on the programming and development section.

Central PCB for the microcontroller

When we started designing our implementation, our main goal was to build a central PCB for housing the microcontroller, that would be completely self depended from the rest of the sensors and power supplies, thereupon could be reusable for another project. Also this specification did leave us with the potential to make errors and mistakes on the peripherals and wouldn't require the remake of the whole system. Consequently the best way to achieve our goal was to break the design into many different peripherals that would be portable to every vehicle and could be evaluated even on a real vehicle too with some additions only. Most of the peripherals have already been outlined into the previous section. At this section we will present the most important part of the system that is the intermediate between sensing, decision and actuation part of the process.

The route that we followed for the design was to place the microcontroller on a PCB, with just the necessary electronics from operation and use dedicated sockets and connectors for the rest of the system. At Fig. 4.37 we can see the wiring of the board. The main parts of the board are: the microcontroller, the XOR gate for the Hall Effect sensors and the ADXL213 interface with the external interrupts, the 16MHz crystal with the necessary capacitors for oscillation, a MAX232 chipset for TTL to RS232 level conversion, a DSUB DB9 female adapter for the serial cable. We have also placed some voltage dividers in case we want to evaluate the level of a voltage above the 5V which is the voltage reference for ADC conversion. Almost every pin from the microcontroller is accessed through pin headers, so we can change the routing at no time. One important aspect that is not clear in the schematic, but is on the PCB board, is that we have used jumpers for certain connections. To be more specific, we might have designed the routing arriving at a certain pin on the microcontroller, but the final connection is not hard wired, but is done through a

simple jumper through a VIA originating out under the bottom layer of the PCB. So we have limited the amount of wires hanging above the microcontroller while retaining the possibility of placing another input directly to the headers. This whole design proved very convenient.

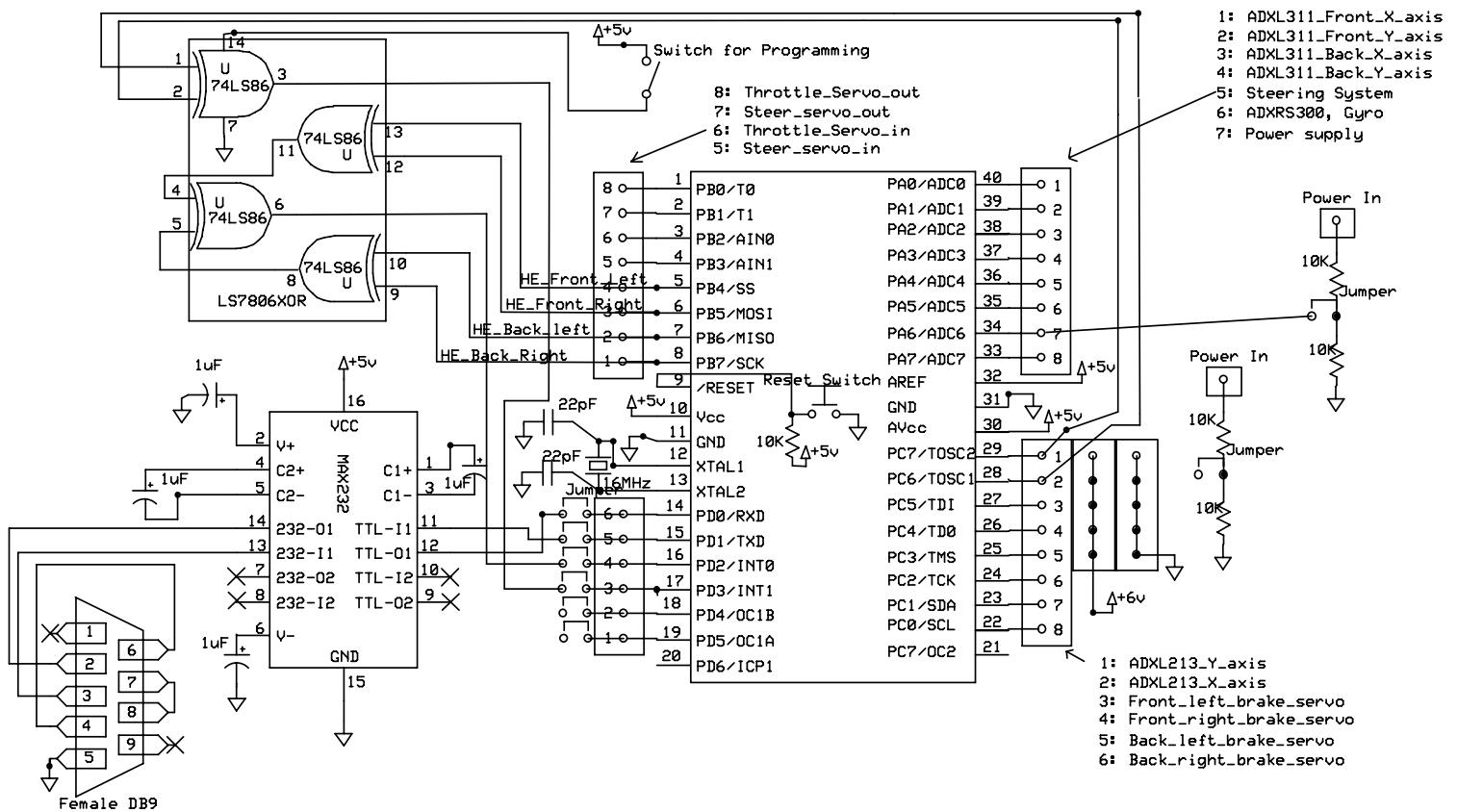


Fig. 4.37 Central board with ATmega32 schematic

In order to connect the peripherals (sensor circuits or actuators) the user just places the input to the right header. There is a brief description, of which signals arrives or leaves a specific pin from the microcontroller. Something that needs to be spelled out is the fact the wherever we have a +5V voltage supply illustrated into the schematic, comes from the regulated power supply presented earlier. That means that the +5V volts reaching to the Vcc of the ATmega32, the AVcc (ADC power supply) and AREF (ADC voltage reference) pins are regulated and retain a quite stable level. The oscillation of the voltage is very limited. Although we had the possibility to use a precision voltage reference chipset on the AREF pin, we decided to use the power supply of the microcontroller itself. The reason we did it is simple and straightforward. Since the voltage arriving at the Vcc, Avcc and AREF pin of the ATmega32 and the voltage at the sensors originate from the same source (the LM317 of the power supply board), then the oscillation of the voltage at the ADC process would be compensated by the fact that if the voltage would increase, it would

happen the same at the output of those sensors which their output is ratiometric of the input voltage, like:

- The steering angle estimation provision. It is a voltage divider → output proportional to input.
- The ADXL311 accelerometer, for which is clearly mentioned on the datasheet that the output and the zero g bias are ratiometric. That means that the zero g output is equal to $V_{cc}/2$ ([17]).
- The ADXL213 accelerometer, for which is clearly mentioned on the datasheet that the output and the zero g bias are also ratiometric. That means that the zero g output is equal to $V_{cc}/2$ for every supply voltage ([19]).

The only sensory provision in the system that is not ratiometric is the ADXRS300 gyroscope. The output is non ratiometric ([26]), but has an internal voltage reference of 2.5V which can be used for compensating the voltage oscillation error ([37]).

From the above we can conclude that the error between the real analog value and the value estimated from the conversion is less when both the analog value and the reference suffer the same offset by the voltage oscillation. For example, lets consider a situation where the accelerometer ADXL311 with ratiometric output experiences zero g acceleration and the voltage V_{cc} drops at 4.95V instead of the 5V that is considered to be nominal.

- The output of the accelerometer would be $V_{cc}/2$. The output for the nominal value (5V) would be $V_{cc}/2=5/2=2.5V$. In our situation the output would be 2.475V.
- The voltage at the AREF pin of the microcontroller also drops at 4.95V.
- The 10bit estimated value from the ADC of the ATmega32([20]) is $ADC=1024*V_{in}(\text{at } V_{cc} \text{ pin})/V_{ref}(\text{at AREF pin})= 1024*2.475/4.95= 512$.
- When we receive this value at the SBC, the conversion from the decimal number back to voltage will be given by $V_{in}=ADC*V_{ref}/1024$, where V_{ref} is nominal the reference voltage (5V) at AREF pin of the microcontroller for the ADC conversion.
- At the opposite operation, conversion back to voltage, as V_{ref} we are using the nominal value which is 5V. That yields to → $V_{in}=512*5/1024=2.5V$, which would be the nominal value. So we have absolute no error (at this specific situation were we don't have quantization error).

More examples and thorough explanation will be given at the firmware section. The PCBs created to mount the circuit of Fig. 4.37 are presented at Fig. 4.38 and Fig. 4.39, top and bottom illustration respectively.

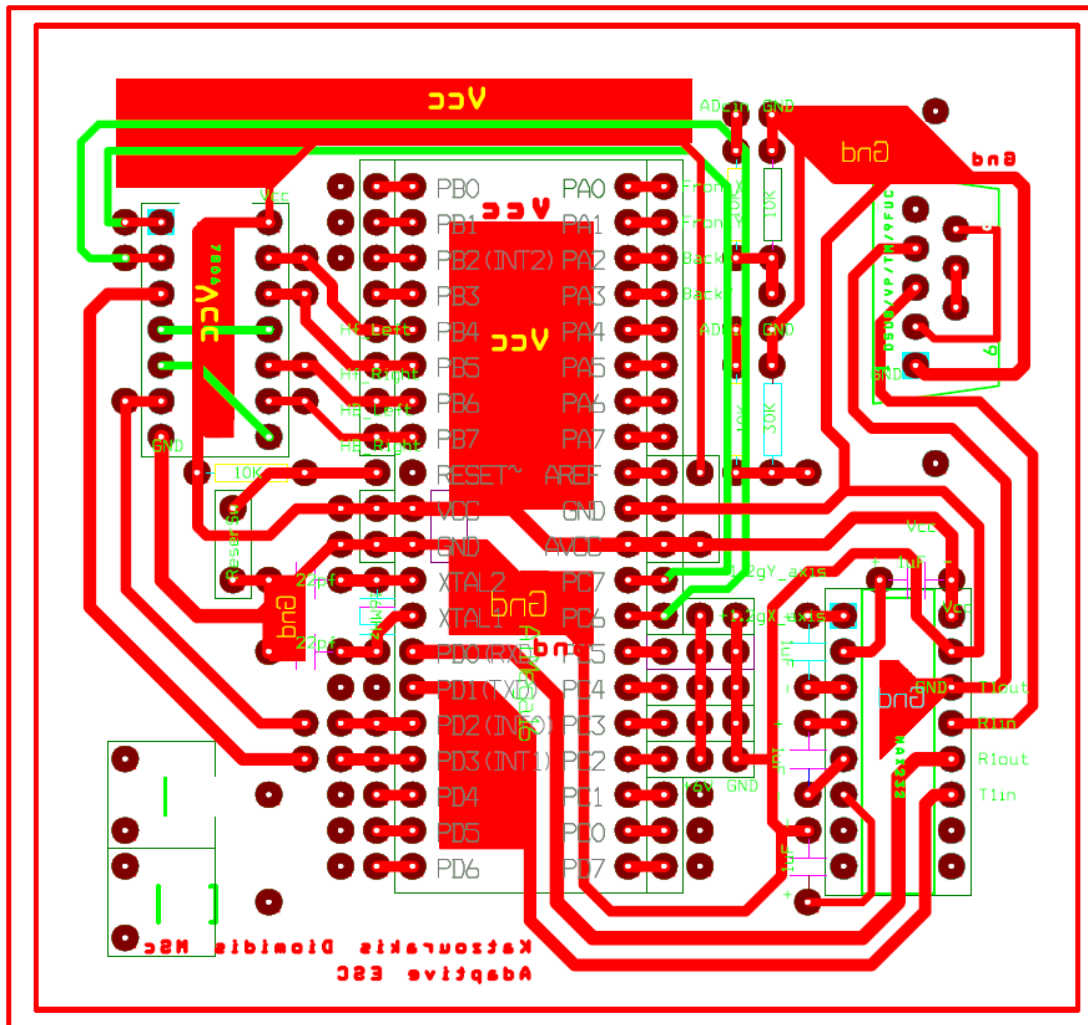


Fig. 4.38 Central PCB: Top layer. The green wires are the second routing layer drew manually with conventional wires.

On the actual PCB used of Fig. 4.38, at the right part of the board we have placed three 10 pin ribbon cable connectors and through them we are connecting the sensors and actuators with the microcontroller while keeping the chaotic routing tidy and easily accessible.

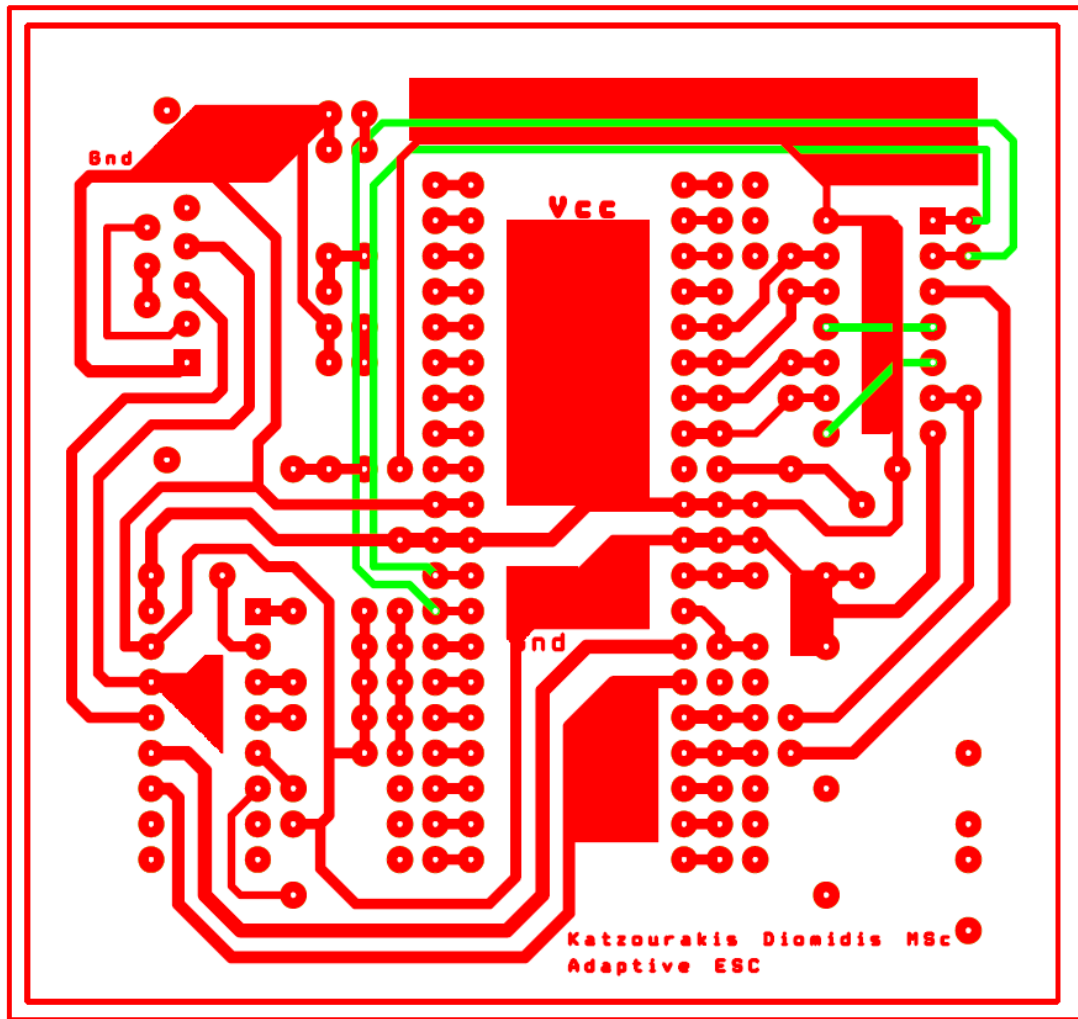


Fig. 4.39 Central PCB: Bottom layer (copper layer). The green wires are the second routing layer drew manually with conventional wires.

The final outcome of the PCB was presented at Fig. 4.5. Everything was easily accessed, very stable and robust. The environment of operation is very harsh and wouldn't leave any margin for mediocre constructions or soldering.

One important aspect of our design was the ability of the microcontroller to be able to communicate with host. The best solution was to use the built – in USART (Universal Asynchronous Receiver Transmitter) of the AVR to communicate with the SBC through the serial port. The serial port of the PC uses the typical RS – 232 levels, while the microcontroller uses classical TTL levels.

RS-232 is a standard that defines the voltage levels that correspond to logical one and logical zero levels and has range of -3 to -15volts and 3 to 15 volts for the logical one and zero respectively while the typical TTL levels are 0 and 5 Volts for the logical zero and one respectively ([38]). For a logical connection between the RS – 232 serial port of the SBC and the USART of the ATmega32 two we have to use a typical level converter; a MAX232 chip ([39]).

For the serial communication, we have used the minimal 3-wire required RS-232 connection consisting only of transmit data, receive data, and ground (Fig. 4.40). This communication was enough since we didn't need any more facilities of the RS-232 standard. We haven't used any flow control on since it wasn't necessary, so we had to short circuit the Request to Send (RTS) with Clear to Send (CTS) lines (necessary for handshaking), and the Data Terminal Ready (DTR) with Data Set Ready (DSR) line for both the microcontroller and the host computer (Fig. 4.37). For their connection we used a simple null modem cable wire. A null modem cable is a simple cable where the receive and transmit line for microcontroller are crosslinked → the transmit from the ATmega32 connects to the receive of the SBC and the transmit of the SBC connects to the ATmega32. One key part of this process that should raise attention, is that in case the wire is fixed and is a null modem cable wire, the RTS with CTS and DTR and DSR should be short circuited manually on the host two (Fig. 4.40). This did cause us some trouble, since we thought that all nine wires from the DB9 connector of the SBC would reach to the short circuit at the PCB board. But this proved to be wrong, since only three wires were travelling inside the shielding from the ready to use cable.

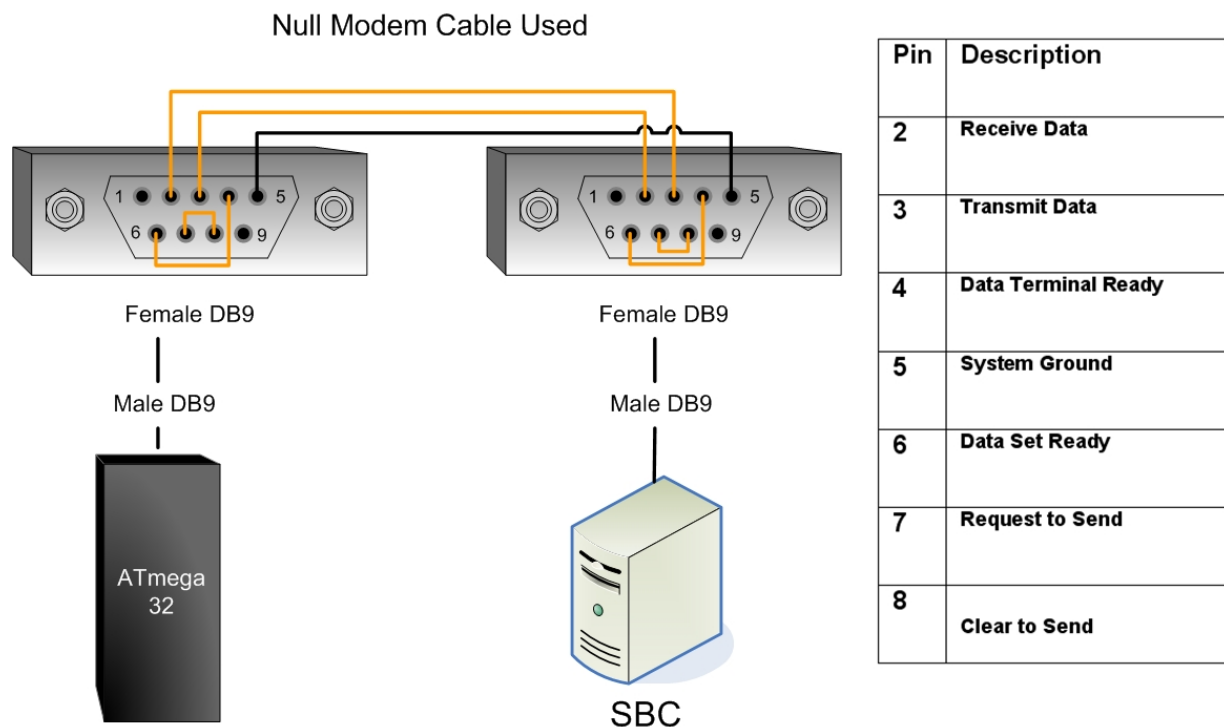


Fig. 4.40 Serial connection between the microcontroller and the SBC

Development Tools

The greatest advantage of using ATMEL AVR microcontrollers is the wide availability of development tools for programming and prototyping. The hardware development tools for prototyping are very economic and can be obtained with very little cost, or can be self-built. There are numerous free designs for AVR programmers on the web accompanied with a very nice documentation and software. In case someone wants to use a commercial available development tool from the ATMEL Corporation itself, with several peripherals for prototyping, an excellent documentation and guarantee for easy to use device can use the STK500 ([35]). The AVR® STK500 (Fig. 4.41) is a low cost complete starter kit and development system for the AVR Flash Microcontroller from ATMEL Corporation.

Development Board

It has been designed in a simple manner so that to give electronics systems designers an easy and quick way to start developing code on the AVR microcontrollers, both for prototyping and testing([35]). STK500 has many features, such as:

- Serial interface to the host computer.
- Regulated power supply input from DC power.
- Serial In System Programming for AVR devices
- Sockets for various sizes of AVR DIP Devices (8-pin, 20-pin, 28-pin, and 40-pin).
- 8 push buttons.
- 8 switches.
- 8 LEDs.
- Connectors for prototyping, giving access to all AVR I/O
- On-board 2-Mbit DataFlash® for Nonvolatile Data Storage ([35]).
- An RS-232 level converter to TTL level.
- Provision for Parallel and Serial High-voltage Programming.
- In-System Programmer for Programming AVR Devices in External Target System.
 - Very convenient feature that we really did appreciate.
- And the most important feature is 100% compatibility with the free AVR Studio®.

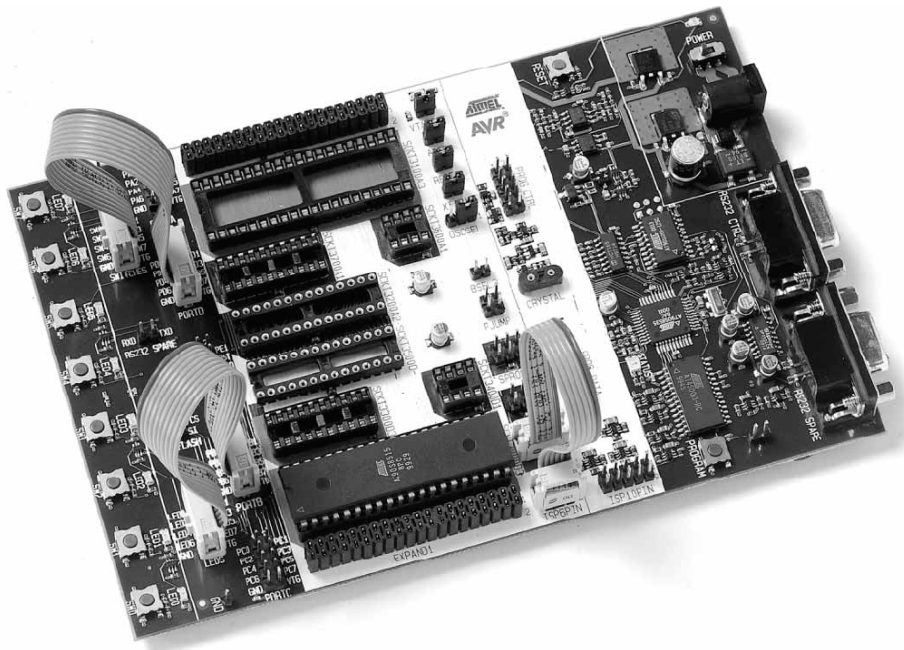


Fig. 4.41 STK500 Starter Kit from ATMEL. Image from ([35]).

The STK500 development board is controlled from AVR Studio (version 3.2 and higher) ([35],[36]). On the other hand AVR Studio is an integrated development environment (IDE) for developing and debugging AVR applications.

Development Software

AVR Studio is easy to work and user friendly. It provides a project management tool, source file editor for C/C++ programming, simulator, in circuit emulator interface and programming interface for STK500. The most important feature of AVR studio is the simulator. It has a built in simulator for most of the AVR devices and provides a full inspection of the I/O inside the AVR device ([36]) (see Fig. 4.42). The user can simulate his microcontroller application step by step; watch the values for variables in both assembly and high-level languages and use the comprehensive help manual embedded into the AVR Studio ([36]).

Most part of the project was developed at the AVR Studio version 4. 13. The firmware for the microcontroller was developed in C language, with WinAVR [15] a suite of executable, open source software development tools for the Atmel AVR series of RISC microcontrollers. WinAVR includes the GNU GCC compiler for C and C++. The software in total is free and available in WWW. WinAVR automatically installs it's assembler into the AVR Studio. So, at the creation of a new project the user has the ability to choose between the AVR assembler from ATMEL contained into the studio, where the typical language for development is assembly. Of course we chose C language instead of AVR assembly for a number of reasons ([42]):

- Reduced development time.
- Portability.
- The GCC compiler is known for the excellent machine and fully optimized machine.
- A number of routines and libraries are available for different AVR devices.
- The debugging is much easier. The code is easily readable.
- Most of AVR developers are also using C, so it is easier in case of a wonder we had the ability to ask for help in a forum.
- And many more...

The only superiority of the assembly programming is that the user can have complete inspection of the code to the last clock cycle. So in case of a very time sensitive application the assembly language is obligatory. There is also the opportunity for the user to mix assembly and C language using a different high level language development tool, like the IAR C – compiler ([40]).

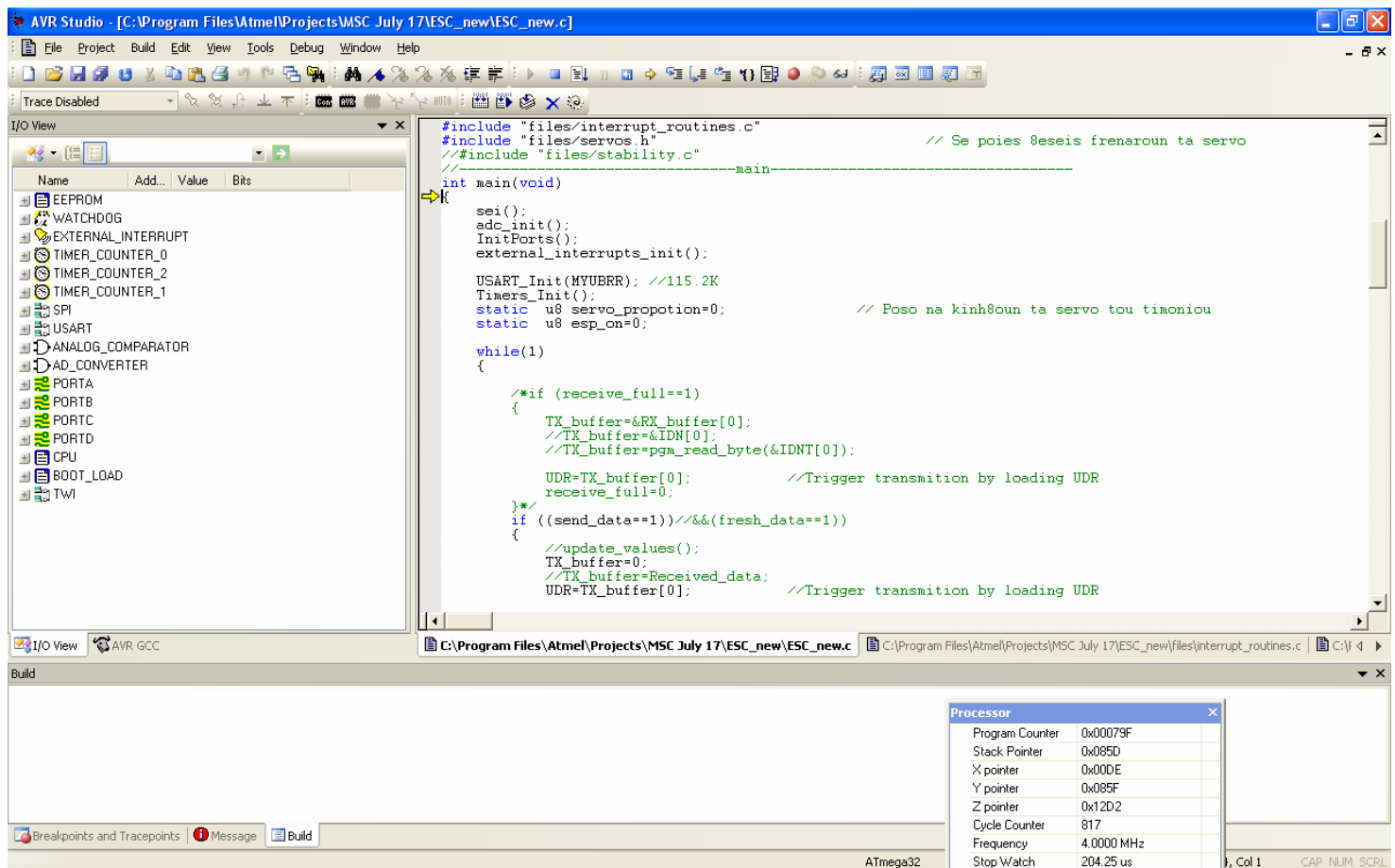


Fig. 4.42 AVR Studio 4.13; project simulation.

Programming the device

To program the firmware we have developed into the AVR device we first compile the program and to produce a hex file. Afterwards, we connect the STK500 into the computer and select

“Connect” from the “Tools” menu in AVR Studio. Thereinafter we select the AVR target which in our case is the ATmega32 and we locate the hex file from the compilation. We choose the ISP (In Circuit Programming) mode and press the program button. Another important part from the programming process is the fact that we must also program some fuses in order the device to work in the desired manner. The settings for the fuses are (Fig. 4.43):

- Uncheck the On – Chip Debug.
 - We didn’t have the necessary equipment for the On – Chip debug, which would be the AVR® JTAGICE mkII from Atmel® ([41]).
- Uncheck the JTAG Interface.
 - The JTAG interface is at PC1 – PC5 pins of the microcontroller, which have been used for interfacing the braking servos. In order this port to be functional we have to disable the interface.
- Check the fuse for using the “External Crystal Resonator, High frequency”.
 - By default, ATmega32 uses the built in clock which is a 1MHz. In order to be clocked from the crystal we have placed at the XTAL1:2 pins we have to check this fuse.

ATmega family of microcontrollers have the provisions to be programmed in circuit without the need to remove the device, attach to the development board and put it back to the circuit. The STK500 can be used as a programmer for external target systems.

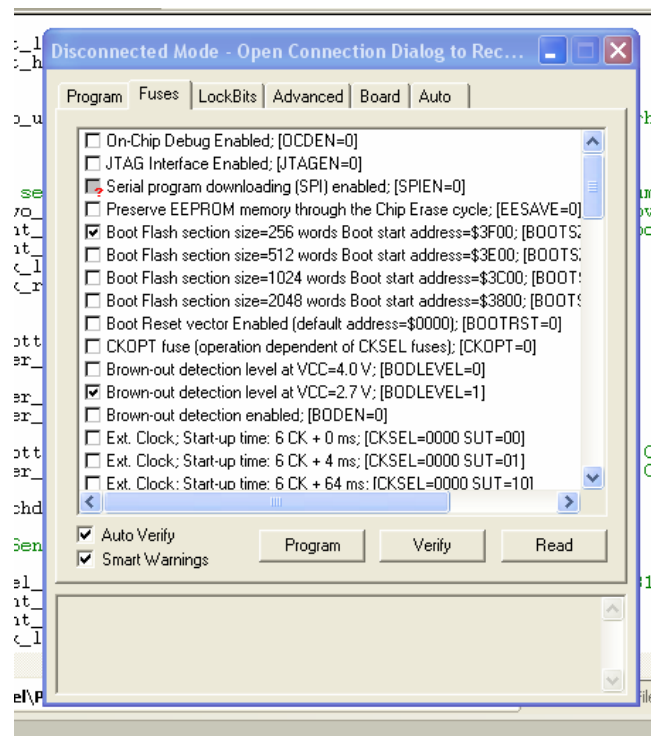


Fig. 4.43 Setting the fuses for the ATmega32.

ISP programming requires only VCC, GND, RESET and three signal lines for programming. Therefore, our approach for maintaining the AVR attached to the chip while programming was simple. We used one of the two ISP connectors of the STK500 by connecting the appropriate pins of the connectors and the ATmega32. The connection is illustrated at Fig. 4.44.

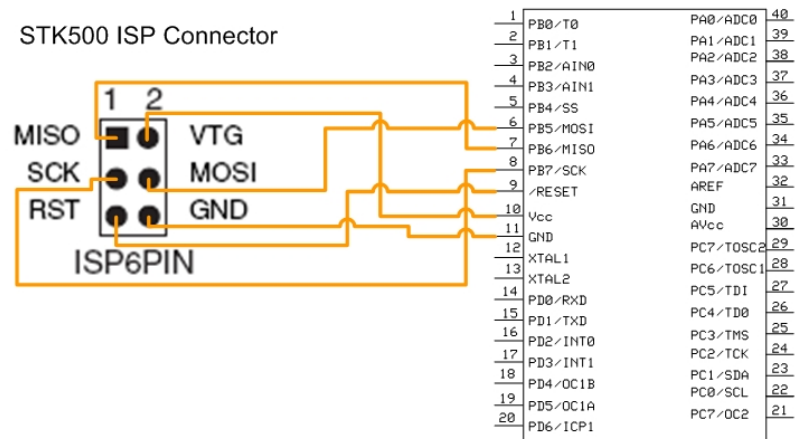


Fig. 4.44 In circuit serial connection for programming from the STK500 6 PIN ISP connector.

We have to underline the fact that in order to program the microcontroller we have to isolate the pins PB5:7 from every active component. In our design, the output from a XOR gate is delivered at those specific ports, so we just had power off the gate (Fig. 4.37). We could follow a more sophisticated solution that would do the process automatically but the installation of a simple switch was more than enough.

4.3 Software at SBC

4.3.1 Single Board Computer; Ubuntu Linux

The operating system installed is Linux Ubuntu 6.10. Ubuntu is a community developed, Linux-based open source free operating system compliant under the GNU general public licence. It contains all the applications a typical operating systems has; graphical user interface to the operating system through Gnome or KDE, a web browser, document and spreadsheet software, and much more. It has a graphical installer which gives the user the opportunity to “get up and running quickly and easily” ([44]) so a standard installation to a nowadays typical computer takes less than 25 minutes. Ubuntu is and always will be “free of charge” ([44]). The user doesn’t need to pay any licensing fees. It is free and has a number of potentialities and as true Linux – based operating system can be customized to the extent of the capabilities of the developer.

The philosophy around Ubuntu Linux and general public license (GPL) is software freedom that aims to distribute around the globe and brings the benefits of software to everyone, while promoting software improvement, cooperation between developers and maintaining the prices of commercial software reasonable, since there is high quality alternatives ([46]). The idea concerning free software and where Ubuntu is a member is that every computer user should have the freedom to download, run, study, share, alter and improve their software for any purpose ([46]). Especially for the academic community, open source software has set the foundations for the exponential growth of the number of software developers, the internet itself and generally has sparked many people to occupy themselves with a number of different topics that turned out to be very interesting and priceless to the academic community. The term “Open Source” doesn’t just mean access to the source code of the software, but has a broader definition ([49]).

On the other hand, the term “free software” is not directly refereed to the price ([46], [48]) although it presumes no charge for the software. For Ubuntu, is used mainly in regard to freedom so that ([48]):

- Anyone can run the programme, for any purpose.
- Anyone can study the operation of the programme through the open source and the necessary nice documentation and adapt it at his needs.
- Anyone can redistribute the alterations he made or new programmes he developed so can help the rest of the developing community, to improve the programme and release his improvements so that everyone benefits.

The licensing concerning Ubuntu is a more complicated. Ubuntu, like most Linux based free operating systems, is collection of different computer software and files created by numerous individuals, teams and companies. It is most probable that each of these attempts come under a different license. There are some restrictions though at the software adapted by Ubuntu: All software included into the operating system's wider access database must include source code and allow modification and distribution of modified copies under the same license. Besides the fact the source code is obligatory, the users must have the potential to change it ([45]). More information concerning Ubuntu Licensing can be found at [45] and free software definition on [48] and [47].

After hundreds of hours of customization we finally reached the Ubuntu Linux installed at our specifications. We disabled whichever part of the operating system wasn't indispensable for the application and installed all the necessary software. We disabled the desktop environment, GNOME, which is the graphical user interface that controls programs which manages application launching, task management file handling in a window environment and is generally an interface for the programs running at the background, similar to working to Microsoft Windows.

Typical Microsoft Windows programs are self contained and do not need many external libraries to work, which are installed automatically with their installers, producing a redundancy and duplicates of libraries. On the other hand Linux based operating systems like Ubuntu, are dependant on external libraries to work. Usually in a Linux system, there exists only one copy of a specific library. While this thing saves a lot of space and keeps the libraries up to date for all the programs sharing the library it produces a great amount of dependencies. Thus in Ubuntu, similar to most modern Linux distribution, has a built in system responsible to deal with software packages and with all the necessary dependencies, without having scattered duplicate of programs or libraries ([50]). Therefore, installing the necessary software to Ubuntu, besides the fact that it is free isn't that complicated as it used to be in the past. There is a main repository for software packages supported by Ubuntu and are guaranteed to work easily. So the user has just to type the command, "apt-get install package" (where package is the name of the specific software he wants to install), while having his computer connected to the internet ([50]).

Unfortunately not all the required software is available from ready to install packages from the Ubuntu community, so the dependencies problems are very difficult to deal with and require a lot of manual work at patience.

Many packages come in source code form. That means that the Linux user has to compile them himself. In order to do this, first has to install some compiler tools. In Ubuntu, those required tools all come with the package build-essential, easily available with the "apt-get install *package*" command, where the *package* is the most updated available tools in the community ([50]).

The analytic description of the customisation took place, is beyond the scope of this document, and there exists some very nice tutorials, on how to set up Ubuntu Linux in a convenient manner. Something worth to be mentioned was the disabling of the IPV6 protocol for speed up the computer itself. The improvement was dramatic and a short description of the process can be found here [51].

Third Party Software

Additional software was installed on the system such as: an SSH server, the GNU GCC compiler, an FTP server, a Webcamserver and an Apache webserver. Everything was set to meet the necessary requirements for the robust function of the platform. The most difficult of all to setup was the webcam server. We have mounted on the model two regular Labtec webcams that worked out of the box with the built – in drivers of Ubuntu. Unfavourably the webcam server found here [52], that was installed manually from source code, was very difficult to set up. The webcam server projects a sequence of Jpegs pictures captured from the webcam to a user specified port, through a Java client program running on the web browser. The application was very stable and is highly recommended! However, the dependencies we met were really annoying and did consume a lot of time and patience until they were set up properly. We had to install the webcam server along with the libc6 package back and forth many time until we find a way to fix it. We tested the webcam server on the Ubuntu 7.10, “*Gutsy Gibbon*” edition with the same webcams and worked excellent. The future developer should just use a newer from our distribution (6.10) and will face no problem.

4.3.2 Custom developed source code

The presentation of the operation of the system will be conducted in three parts: first a brief outline of the different part of the programs developed, second a thorough and analytic description of each programme, routine, code or script and its operation, and finally setting the model to operate and access the gathered data.

A Brief outline

A daemon server was developed for data acquisition from the computer’s serial port. The daemon is written in C/C++ and besides the data collection calls the routine for the stabilization of

the vehicle, also written in C/C++. The program was altered in order to cooperate with a Java Server – Client program for graphical user interface through a common browser that was developed by Mr. Bill Hatzidiakos with contribution from the author (Fig. 4.45). With the usage of this program the user can define the method of stabilization, adjust some parameters, have access to log files and generally have access to a nice graphical interface of the collected data from the sensors and the control decision for the stabilization.

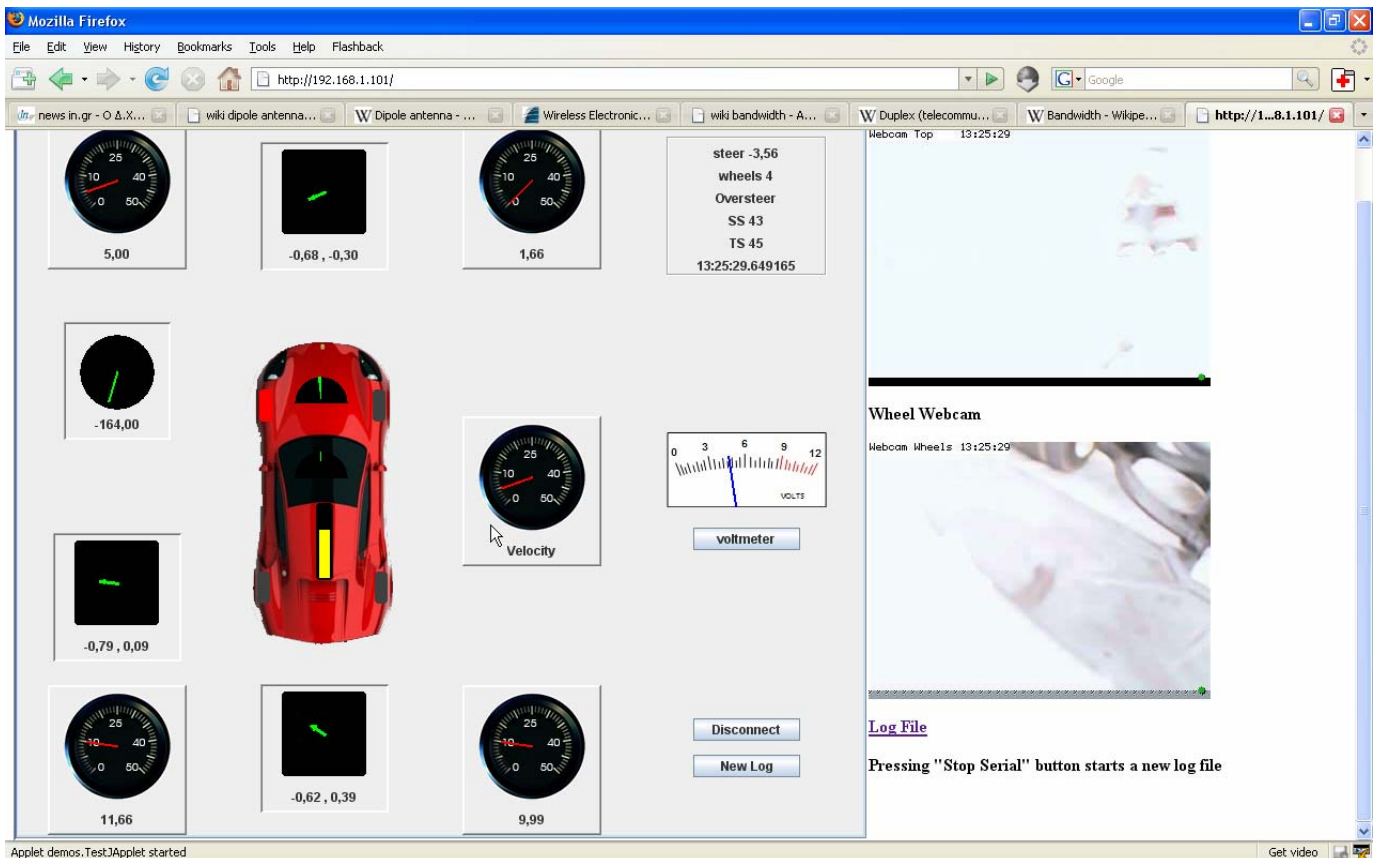


Fig. 4.45 The Java client GUI with the real time video streaming from the webcam server. Instance from an auto generated experiment. Left turn where the vehicle oversteers and brakes the front left inner wheel (with red).

For the operation of the ESP system with the Java GUI through the browser there are four different parts – programs which cooperate to each other. The most important part of the program is the daemon server which was developed for data acquisition from the computer's serial port. This daemon opens the serial port, listens for data packets, processes the data and calls the stabilization routines. It also writes the commands back to the serial port and builds the log file.

This daemon is called by a socket server which is developed by Mr. Hatzidiakos. The socket server listens for incoming connections from the internet. When a connection is set through a web browser, the user can access the port 80 where the Apache webserver handles the

communication. We have placed the Java jar file for the client execute at the user's browser at the root directory of the Apache webserver which was left at its default: */var/www*. also developed by Mr. Hatzidiakos were the exchange between the user and the daemon takes place.

The communication is handled through sockets providing the potential for real time data exchange between the user and the system. Data is provided in the form of C pointers with the help of threads and mutexes. Initialization process:

- i. Connect the power cable and press the power push button to switch on the system.
- ii. Connect through a wireless connection to the Linksys router using Static IP Addressing.
- iii. Use an SSH client (PUTTY ([53])) to connect to the SSH server of the SBC. Connect to IP: 192.168.1.101.
- iv. Execute the socket server from the folder it is stored at the SBC.
- v. With a web browser, connect from the client computer; connect to the address 192.168.1.101. The client computer should have Java Runtime Environment Installed.
- vi. At web browser of the client computer, the index.html will be loaded by default. This HTML code initiates three Java applets (Table 4-8). The first is the Java GUI, and the other two are two Java Applets for the real time video streaming. The outcome is displayed at Fig. 4.45.
- vii. The user in order to connect to the socket server running at the SBC, must press the connect button and everything is ready for experimenting using different ESC schemes, adjusting parameters and fine tuning the ESC algorithms.

From this point on, at the SBC the socket server is exchanging data with the browser and is generating a log file with all the operation, from the data gathered to the commands issued, which can be accessed from the browser through a hyperlink.

```
<html>
<body>
<p><b>
<applet code=demos.TestJApplet archive=index.jar width="800" height="700" align="left">
Your browser does not understand Java.
</applet></b></p>
<p><b>Top WebCam</b></p>
<p>
<APPLET CODE = "WebCamApplet.class" archive="applet.jar" WIDTH = "320" HEIGHT = "240">
<param name=URL value="http://192.168.1.101:8888">
<param name=FPS value="10">
<param name=width value="320">
<param name=height value="240">
</APPLET></p>
<p><b>Wheel Webcam</b></p>
<p><APPLET CODE = "WebCamApplet.class" archive="applet.jar" WIDTH = "320" HEIGHT = "240">
<param name=URL value="http://192.168.1.101:8887">
<param name=FPS value="10">
```

```

<param name=width value="320">
<param name=height value="240">
</APPLET>
</p>
<p><b><a href="http://192.168.1.101/~diomidis/logmath.txt">Log File</a></b></p>
</body>
</html>

```

Table 4-8 Index.html: Loads the applets, HTML code.

Daemon!

The running socket server calls the daemon, which listens to the serial port, processes the data and controls the vehicle, in a function call. The daemon, as any programmed characterized this way, runs in the background, is active all the time while consuming very little process power and is invisible to the user. We can see an instance of the daemon printing the data on the standard output, at the SSH client window (Fig. 4.46).

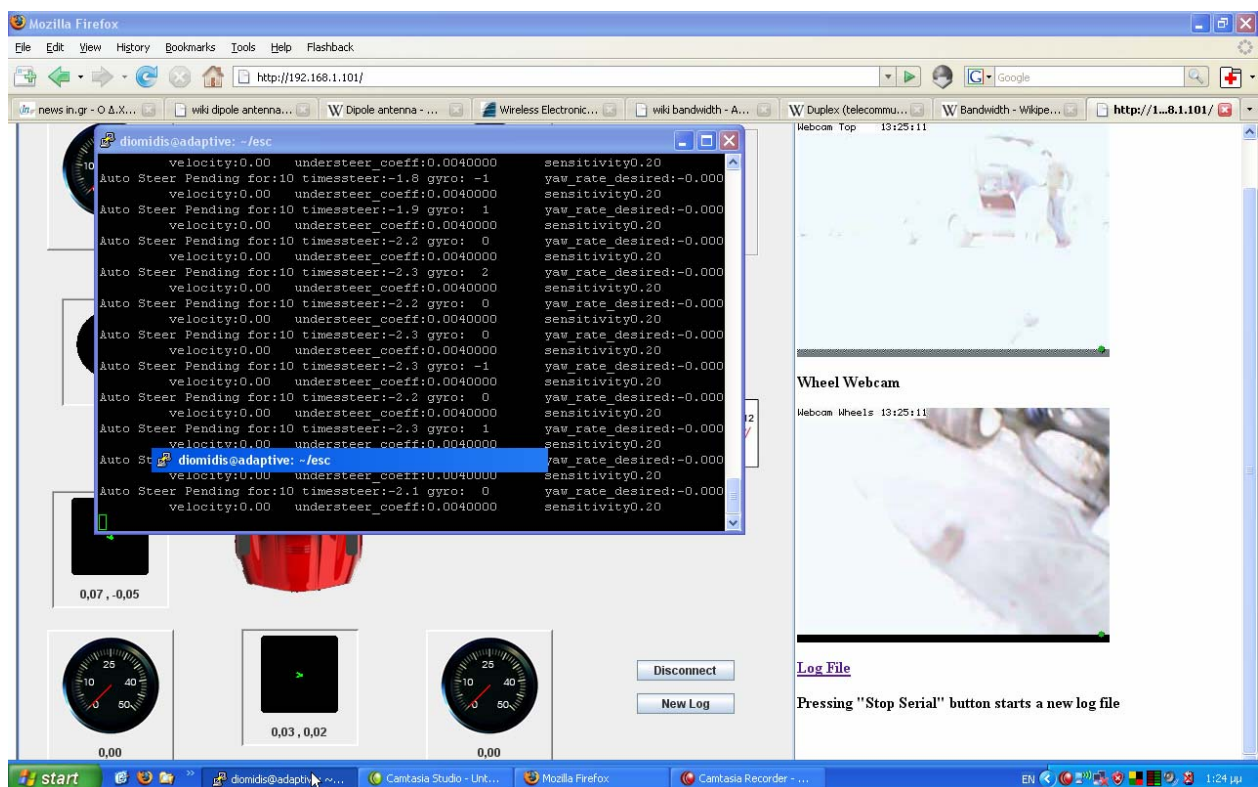


Fig. 4.46 Instance of the daemon printing some data on the standard output through the SSH client.

The program starts with the execution of the of the socketserver program also written in C/C++ by Mr. Hatzidiakos. Socketserver handles the threads creations and execution of the daemon through sockets. An abstractional flowchart of the process is illustrated on Fig. 4.47, where it

portrays only key parts of the procedure. We shall comment the software analytically from the point where the thread is created and after, since until the thread creation, the software has been developed by Mr. Hatzidiakos for his undergraduate thesis. The daemon as mentioned before is executed in a thread form named “serial_com” (Table 4-9).

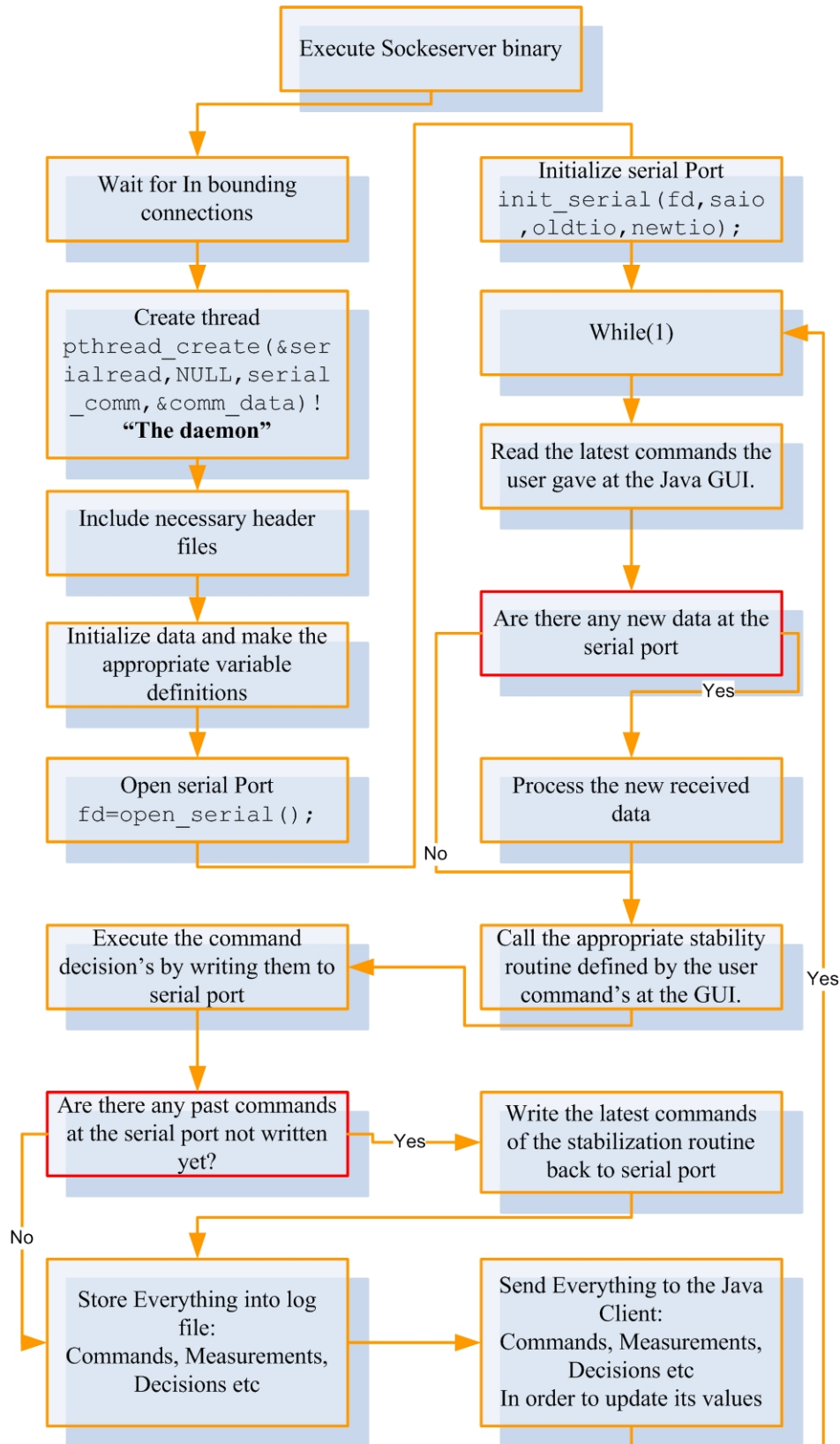


Fig. 4.47 Socketsserver abstractional flowchart.

The part of the process that initiates the thread creation is presented at Table 4-9. From this point on the software presented is developed from the author.

```
//run a serial comm thread

if(!comm_data.serial_running)
{
pthread_t serialread;
if(pthread_create(&serialread,NULL,serial_comm,&comm_data)!=0){
puts("Could not create serial thread");
}
else pthread_detach(serialread);
```

Table 4-9 Serial_com thread creation. Inside socketsserver.

The so – called “daemon”, the *serial_com* routine which is called inside a thread is the conductor the electronic stability control system. An analytical flow chart of the operations taking place at the *serial_com* is cited below on

Fig. 4.48. Because the flow chart would be very big, it is presented in sequential figures referring to each other.

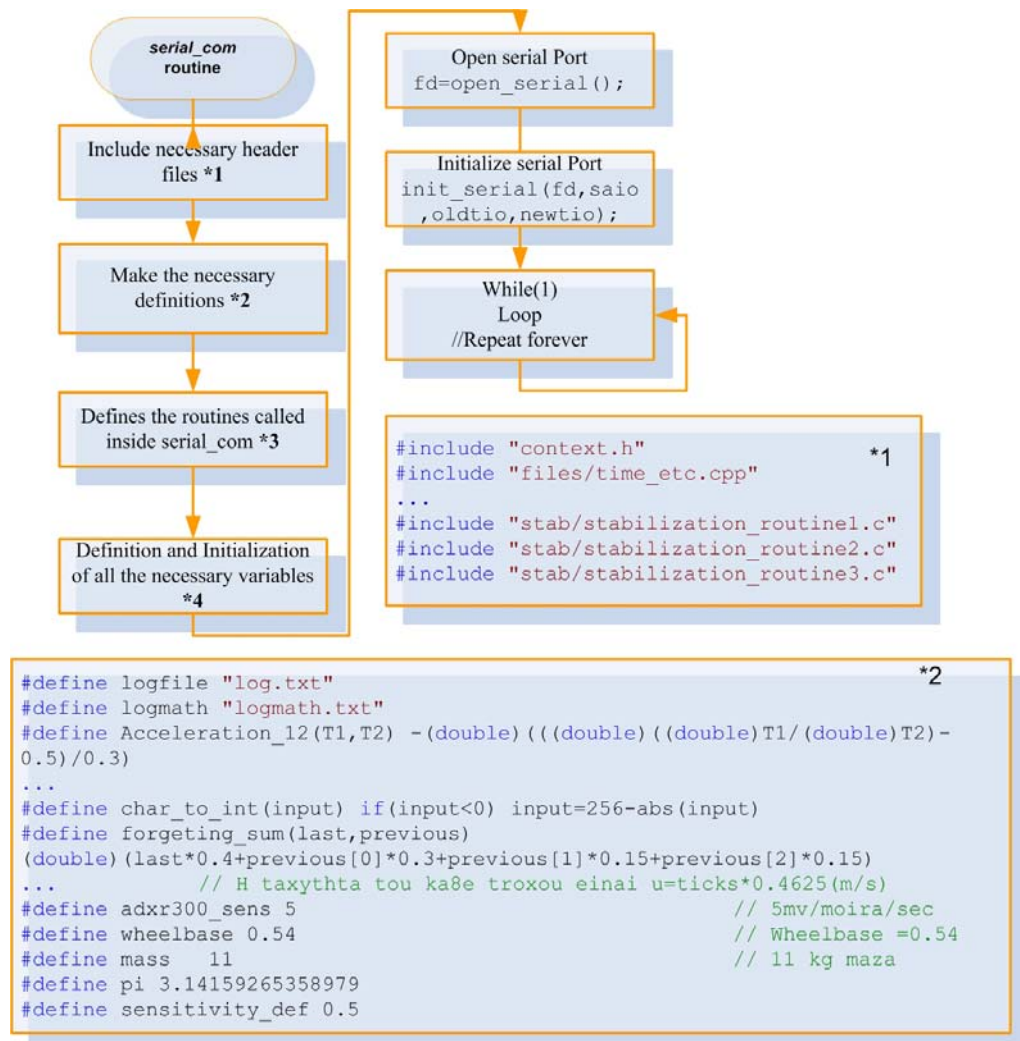


Fig. 4.48 Serial_com routine flow chart part A. File inclusion (*1) and definitions (*2).

```
void signal_handler_IO (int status);           *3           /* definition of signal handler */
char * return_date(void);
int * return_time(void);    //17_2_08 Dio

void shift_values(double last,double previous[]) {
    previous[3]=previous[2];
    previous[2]=previous[1];
    previous[1]=previous[0];
    previous[0]=last;
}
...
char * stabilization_1(double front_left_speed,double front_right_speed,double back_left_speed,double
back_right_speed,double gyro_degs_ana_sec,int throttle_servo,int steer_servo,double steer,int*
stab_int_parameters,double* stab_double_parameters,long time_passed);
char * stabilization_2(double front_left_speed,...
char * stabilization_3(double acc_12_X_gs,double acc_12_Y_g...
```

```
int fd,c, res,n;                               *4
struct termios oldtio,newtio;
struct sigaction saio;                         /* definition of signal action */
char buf[100];
int hr8e;
int posa;
int bytes=0,w_bytes=0;
long deigmata=0;
// Acceleration 1.2 g
//int acc_12_X=0,acc_12_Y=0;
int acc_12_X_high=0,acc_12_X_low=0;
int acc_12_Y_high=0,acc_12_Y_low=0;
double acc_12_X_gs=0,acc_12_Y_gs=0;
double acc_12_X_gs_buf[4],acc_12_Y_gs_buf[4];
...
struct timeval tp;                             //Apo http://cygwin.com/ml/cygwin/2002-01/msg01408.html
long time_start,time_end;
long time_passed=0,time_passed_sum=0;

gettimeofday(&tp,0);
time_start = ((unsigned)tp.tv_sec)*1000000+((unsigned)tp.tv_usec);
```

Fig. 4.49 Serial_com routine flow chart part B. Routine (*3) and variables (*4) definitions.

At Fig. 4.48 and Fig. 4.49 we can see the initialization of the routine *serial_com* until the point it reaches the *while(1)* “endless loop”. The routine is placed in C++ file called *async.cpp*, where it has some inclusions of another C, C++ or header files necessary for the software. The whole software has been divided at the most possible separate files in order to be more legible, simpler to debug, easily reusable and most of all easier to develop. So instead having a big chaotic file to work with, we are combining many different. Also all the routines are defined outside from the file C++ which includes the “endless loop”; for the same reasons as those cited above. The included files are:

- "files/headers.h"
- "context.h"
- "files/time_etc.cpp"
- "files/init_serial.c"
- "files/servos.h"
- "stab/stabilization_routine1.c"
- "stab/stabilization_routine2.c"
- "stab/stabilization_routine3.c"

The folder “files” and “stab” are directories for better organization of the files. Continuing with the *serial_com* routine from the

Fig. 4.48 we reach at the point where the program enters the “endless loop” (Fig. 4.50).

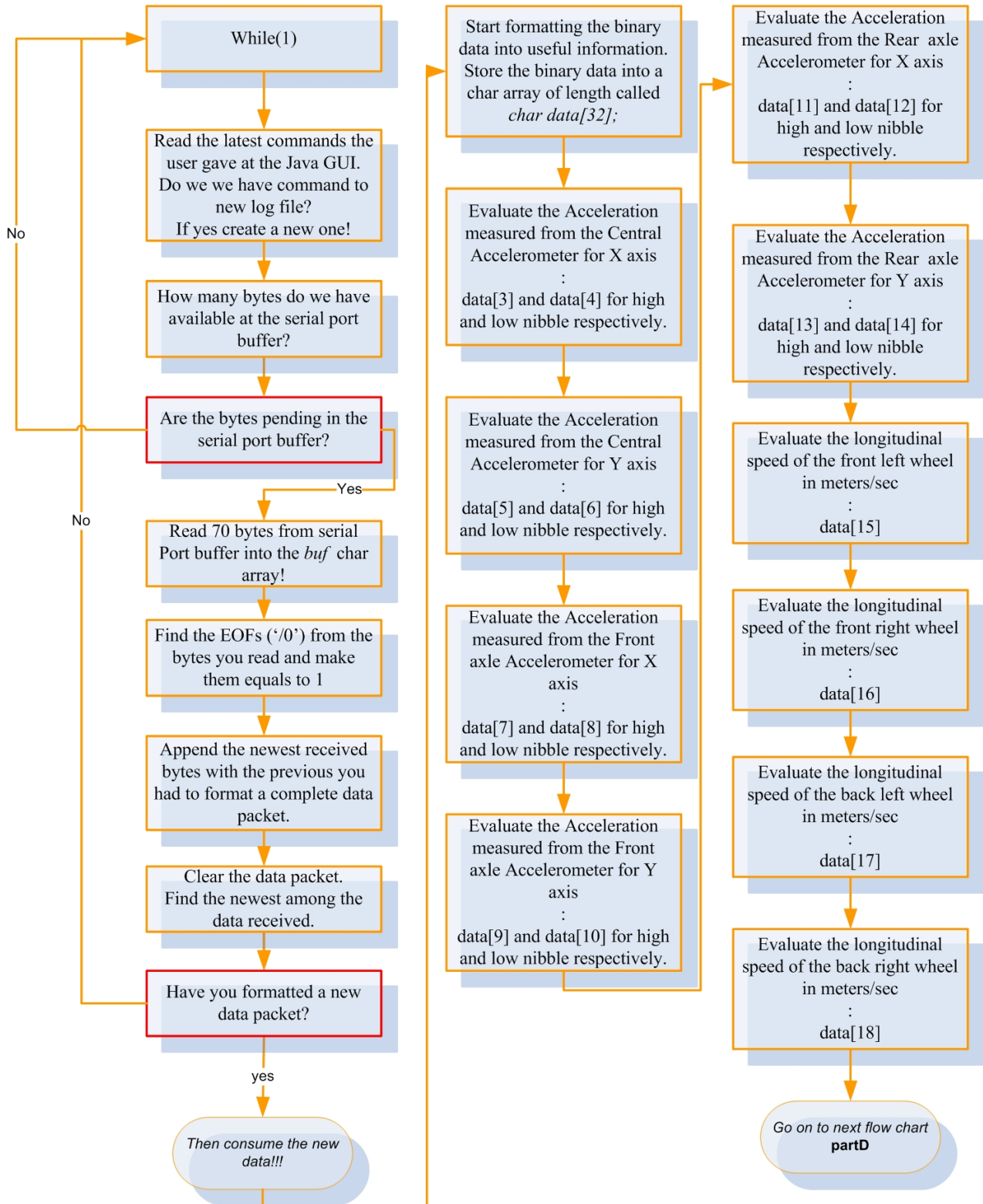


Fig. 4.50 *Serial_com* routine flow chart part C. “Endless loop”.

```

graph TD
    Start([Serial_com partD continue]) --> EvalSteer[Evaluate the steering angle  
:  
data[19] and data[20] for  
high and low nibble  
respectively.]
    EvalSteer --> EvalYaw[Evaluate the yaw rate from  
gyroscope  
:  
data[21] and data[22] for  
high and low nibble  
respectively.]
    EvalYaw --> EvalBatt[Evaluate the battery voltage  
:  
data[23] and data[24] for  
high and low nibble  
respectively.]
    EvalBatt --> EvalThrottle[Evaluate throttle input from  
the driver  
:  
data[25]]
    EvalThrottle --> EvalSteering[Evaluate steering input from  
the driver  
:  
data[26]]
    EvalSteering --> Switch{Which routine has the user asked you to  
call?  
switch (bill_local_command)}
    Switch -- 1 --> CallStab1[Call stabilization_1, pass the necessary evaluated data from microcontroller  
and the user commands from the GUI.  
stab_commands=stabilization_1( front_left_speed, front_right_speed,  
back_left_speed, back_right_speed,  
gyro_degs_ana_sec, throttle_servo,  
steer_servo,steer,stab_int_parameters,stab_double_parameters,time_pass  
ed);]
    CallStab1 --> CallStab2[Call stabilization_2, pass the necessary evaluated data from microcontroller  
and the user commands from the GUI.  
stab_commands=stabilization_2( front_left_speed, front_right_speed,  
back_left_speed, back_right_speed,  
gyro_degs_ana_sec, throttle_servo,  
steer_servo,steer,stab_int_parameters,stab_double_parameters,time_pass  
ed);]
    CallStab2 --> CallStab3[Call stabilization_3, pass the necessary evaluated data from microcontroller  
and the user commands from the GUI.  
stab_commands=stabilization_3( acc_12_X_gs, acc_12_Y_gs,  
acc_2_front_X_gs,acc_2_front_Y_gs,acc_2_back_X_gs,acc_2_back_Y_gs,fron  
t_left_speed,front_right_speed,back_left_speed,back_right_speed,pont_v  
olts,gyro_degs_ana_sec,batt_volts,throttle_servo,steer_servo,steer,sta  
b_int_parameters,stab_double_parameters);]
    Switch -- default --> DoNothing[Do nothing!  
  
// Release all wheels  
stab_commands[2]=front_left_servo_release;  
stab_commands[3]=front_right_servo_brake;  
stab_commands[4]=back_left_servo_release;  
stab_commands[5]=back right servo release;]
    CallStab3 --> WriteLog[Write all evaluated data and commands at the log file described by the file descriptor called *billfptr  
:  
  
FILE *billfptr;  
if ((billfptr=fopen(logmath,"a+"))==NULL) // a+, anoigel h dhmiourgei ena neo arxeio gia  
prosarthsh, sel 358  
  
{ perror(logmath); kill(getpid(),0); }  
and  
sprintf(big_buffer_math,"%1.3f \t%1.3f \t%1.3f \t%1.3f \t%1.3f \t%1.3f \t%2.2f \t%2.2f \t%2.2f \t%2.2f \t%3.0f \t%2.2f \t%  
t%d \t%4.4f \t%4.4f \t%4.4f \t%d \t%d \t%d \t%d \t%d \t%d \t%d \t%d \t%  
t%d \t%d \t%d \t%d \t%d \t%d \t%d \t%d \t%d \t%d \t%  
n",acc_12_X_gs,acc_12_Y_gs,acc_2_front_X_gs,acc_2_front_Y_gs,acc_2_back_X_gs,acc_2_back_Y_gs,front_left_speed,front_right_s  
peed,back_left_speed,back_right_speed,gyro_degs_ana_sec,steer,bill_local_command,stab_double_parameters[0],stab_double_para  
meters[1],stab_double_parameters[2],auto_trig,auto_speed,auto_steer,stab_commands[2],stab_commands[3],stab_commands[4],stab  
_commands[5],bill_brake,deigmata,time_passed,(time_passed_sum/1000),hour,min,sec,usec);  
\\MX \\MY \\FX \\FY \\BX \\BY \\FLS \\FRS \\BLS \\BRS \\tGyro \\tSteer \\tMethod \\tSDP[0] \\  
tSDP[1] \\tSDP[2]\\tauto_trig \\tauto_speed \\tauto_steer \\tFLB \\tFRB \\tRLB \\tBRB \\tbill_brake \\  
tSamples \\tTime_passed: \\tTimeSum \\tHour \\tmin \\tsec \\tusec \\n"  
fputs(big_buffer_math,billfptr);]
    WriteLog --> End([Go to next flow chart  
partE])
  
```

108

Thus, instead one byte of information having the code value in decimal form just 0 – 9 (ASCII), we are using the whole 8 bit of information, except the 255 which is left for packet handling. The above means that in one byte we are coding 0 – 254 in decimal representation.

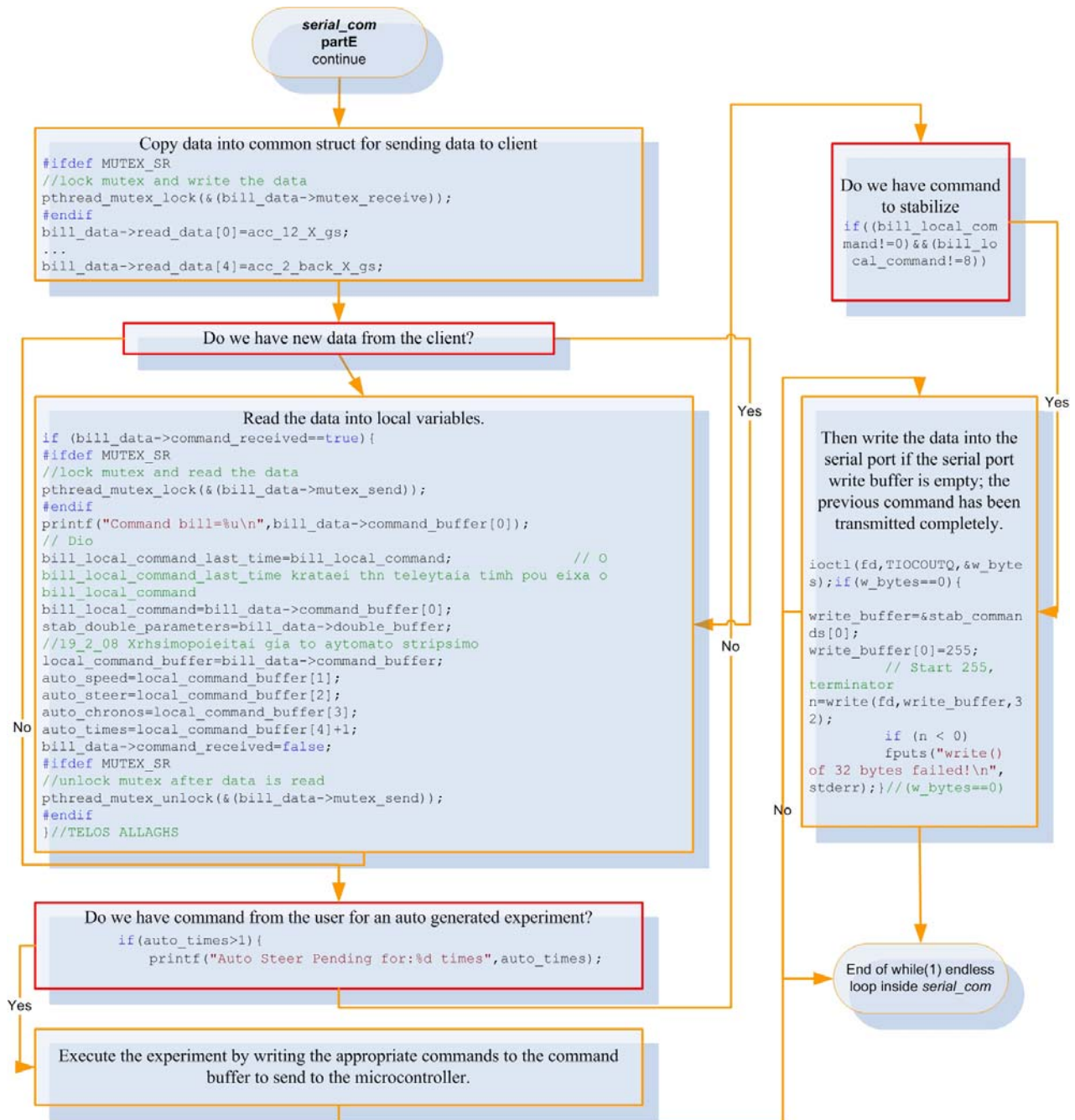


Fig. 4.52 Serial_com routine flow chart part E. "Endless loop", last part.

Fig. 4.52 shows the completion of the flow chart of the routine `serial_com`. Some key parts of the routine must be cleared out:

- Serial port is initialization and configuration through the routines at `init_serial.c` file.
- The way data are evaluated from the information of the binary data.
- The way the auto generated experiments work, first mentioned at Fig. 4.52.

Serial port initialization, reading and writing

At

Fig. 4.48 we can see the sequence of events to initialize and configure the serial port. Most of the settings and configuration has been done through guidance from [55] and the *tty_ioctl* manual page for Linux. The program first calls the *open_serial()* routine placed at the included files of *init_serial.c*. The *open_serial()* (

Table 4-10) returns a integer file descriptor (`int` `fd`) for opening the MODEMDEVICE defined as “`#define MODEMDEVICE "/dev/ttyS0"`” in the included file *settings.h*, in Read/Write open mode (`O_RDWR`) and with No TTY control (`O_NOCTTY`), with the `O_NONBLOCK` flag set, where `read()` returns immediately without reading any data if not available (doesn't blocks).

```
int open_serial(void)
{
    int fd;
    /* open the device to be non-blocking (read will return immediatly) */;
    fd = open(MODEMDEVICE, O_RDWR | O_NOCTTY | O_NONBLOCK);
    if (fd < 0) {perror(MODEMDEVICE); kill(getpid(),0); }
    return fd;
}
```

Table 4-10 Routine *open_serial()* at *init_serial.c*.

The next routine to be called is the *init_serial()* at the file *init_serial.c*. It is presented at Table 4-11 along with the signal handler “`void signal_handler_IO (int status)`” for the asynchronous operation. It accepts a number of inputs, where two of them are structs “`struct termios oldtio,newtio;`” for the old and new settings used for setting the serial port. The *sigaction()* function allows the calling process to examine the action to take: “`saio.sa_handler = signal_handler_IO;`” to be associated with a specific signal, which in our case in the SIGIO interrupt, “`sigaction(SIGIO,&saio,NULL);`” ([56]). The *fcntl()* function provides control of open file descriptors, “`fcntl(fd, F_SETOWN, getpid());`”, where the `F_SETOWN` flag sets the owning process ID, “`getpid()`”. The owning process or process group can receive SIGIO signals for readable data on communications ports (serial port in our case). The argument is the process ID or the negative of the process group ID for the owner, as a variable of type `pid_t`. The return value is 0 on success, or -1 on error.

To receive SIGIO signals, the process should establish a SIGIO handler, “`saio.sa_handler = signal_handler_IO;`”, prior to setting ownership of the file descriptor, and then must enable `O_ASYNC` with the `F_SETFL` command to *fcntl()*: “`fcntl(fd, F_SETFL, FASYNC);`”.

```

#include "headers.h"

/*****
 * signal handler. sets wait_flag to FALSE, to indicate above loop that
 * characters have been received.
 *****/

void signal_handler_IO (int status)
{
    //printf("signaled\n")    ;
    wait_flag = FALSE;
}

void init_serial(int &fd,struct sigaction &saio,struct termios &oldtio,struct termios &newtio)
{
    ////////////+++++++////////////////////////
    // install the signal handler before making the device asynchronous
    saio.sa_handler = signal_handler_IO;
    saio.sa_flags = 0;
    saio.sa_restorer = NULL;
    sigaction(SIGIO,&saio,NULL);

    // allow the process to receive SIGIO
    fcntl(fd, F_SETOWN, getpid());
    // Make the file descriptor asynchronous (the manual page says only
    // O_APPEND and O_NONBLOCK, will work with F_SETFL...)
    fcntl(fd, F_SETFL, FASYNC);

    tcgetattr(fd,&oldtio); // save current port settings

    //async1
    //
    // Set bps rate and hardware flow control and 8n1 (8bit,no parity,1 stopbit).
    // Also don't hangup automatically and ignore modem status.
    //Finally enable receiving characters.

    newtio.c_cflag = BAUDRATE | CRTSCTS | CS8 | CLOCAL | CREAD;

    newtio.c_iflag = IGNPAR; //Ignore bytes with parity errors and make terminal raw and dumb.

    newtio.c_oflag = 0;      //Raw output.

    //Don't echo characters because if you connect to a host it or your
    //modem will echo characters for you. Don't generate signals.

    newtio.c_lflag = 0;

    // blocking read until 1 char arrives
    newtio.c_cc[VMIN]=1;
    newtio.c_cc[VTIME]=0;

    // now clean the modem line and activate the settings for modem
    tcflush(fd, TCIFLUSH);
    tcsetattr(fd,TCSANOW,&newtio);
}

```

Table 4-11 Routine *init_serial()* at *init_serial.c*

A SIGIO signal is generated whenever there is data to be read on the file descriptor. So whenever we have data at the serial buffer, the *signal_handler_IO()* routine will be called making the “wait_flag = FALSE;”. Although we have set the software to signal on IO and this is the approach for most programmers seeking an asynchronous operation for their program, we decided to use a more secure and robust solution. The approach was to read how many bytes were at the serial port input buffer. If they were more than 0, we issued the read command. This was done with the function *ioctl()*. At Table 4-12 we can see the way our approach works. The main difference is that our packet needs to be reconstructed, since it is read with a random number of bytes every time. That means, that the program will issue a command for reading 70 bytes, will read those available at the buffer denoted by the “res” variable and will reconstruct the data packet with a little programming.

```
// Read serial port
ioctl(fd,FIONREAD,&bytes);
if(bytes>0){           // Append at serial_port_buffer to consume
    res = read(fd,buf,70);
}

ioctl(fd,TIOCOUTQ,&w_bytes);

// Write at serial port
if(w_bytes==0){
    write_buffer=&stab_commands[0];
    write_buffer[0]=255;           // Start 255, terminator

    n=write(fd,write_buffer,32);
    if (n < 0)
        fputs("write() of 32 bytes failed!\n", stderr);
}
```

Table 4-12 Read and write at the serial port at inside *serial_com()* routine.

Evaluation of the information from the binary data

The data arriving at the serial port are raw binary data needed to be translated into useful information. The first thing we have to do is convert them from char form (single bytes) into integers. The data are in two's complement representation. So a directed assign from their to binary value to integers is prohibited. We have used a directive definition “*#define char_to_int(input) if(input<0) input=256-abs(input)*”. So in order to translate a the char (byte) into integer the call we make is “acc_2_front_X_high = (*int*) (data[7]); char_to_int (acc_2_front_X_high);”. So we convert the char variable “data[7]” to integer through a direct integer assign, and then convert this two's complement integer into a real number integer with the

definition. Most of the data arriving at the serial port consist of two bytes. So we have to combine two chars (two single bytes) into an integer. The conversion is straightforward and is illustrated at

Table 4-13. The array “data[]”, is a char array which holds the bytes from the data packet transmitted from the microcontroller into the SBC through the serial port. At

Table 4-13 we have an example how to combine the 16 – bits measurement (two bytes) for the ADXL213 accelerometer. The lower nibble is “acc_12_X_low” the and the higher is “acc_12_X_high” which are finally combined into the integer “acc_12_X_high”. The if – else pattern at the conversion has to do with the fact that a “0” (00000000 in binary representation) in a char array is the end of file for the C language.

```
//-----1.2g Accelerometer-----
acc_12_X_high = (int)(data[3]);
char_to_int(acc_12_X_high);
    if(acc_12_X_high==254)
        acc_12_X_high=0;
acc_12_X_low = (int)(data[4]);
char_to_int(acc_12_X_low);
acc_12_X_high=(256*acc_12_X_high)+acc_12_X_low;
```

Table 4-13 Conversion from raw binary data arriving at the serial port into two bytes integers.

Thus, we must not transmit “0” in binary because it denotes the end of the array. So for some values, we know that the decimal “254” is not a valid representation, like the higher nibble from a 10 bit ADC conversion. So when we detect a decimal “254”, we know that it was an original decimal “0” at the microcontroller transmitted as “254” in order not signal the EOF for our data array. The problem is that we miss a bit for the lower nibbles, since we can’t distinguish the “0” from the “1”, but in ADC conversions where this it causes a $1/1024 \% \approx 0.1\%$ error, which is not catastrophic collated with the speed up our method for transmission gives.

Besides the conversion of raw binary data into integers, we have to transform each integer value into useful information. For example, the outcome from the analog to digital converter for the X axis of the front axle ADXL311 accelerometer (analog output) is an integer value between 0 and 1023 value in decimal representation. This value has to be translated firstly into voltage and after that into acceleration. We shall present one by one how we evaluate the information for all the data measured.

- Acceleration from the central accelerometer: ADXL213 $\pm 1.2g$ Accelerometer
 - X Axis:

```
//-----1.2g Accelerometer-----
acc_12_X_high = (int)(data[3]);
char_to_int(acc_12_X_high);
    if(acc_12_X_high==254)
        acc_12_X_high=0;
acc_12_X_low = (int)(data[4]);
char_to_int(acc_12_X_low);
```

```

acc_12_X_high=(256*acc_12_X_high)+acc_12_X_low;
acc_12_X_gs = Acceleration_12(acc_12_X_high,455);
acc_12_X_gs=forgeting_sum(acc_12_X_gs,acc_12_X_gs_buf);
shift_values(acc_12_X_gs,acc_12_X_gs_buf);

```

After we have evaluated the length of the T2 pulse (Fig. 4.10), we are passing the value into the pre-processor defined formula for evaluating the acceleration, the: “#define Acceleration_12(T1,T2) -(double) (((double) ((double)T1 / (double)T2)-0.5)/0.3)”. This definition takes as input the length of pulse T1 and T2 and returns the acceleration in Gs, according to methodology presented at the ADXL213 interface section. In order to reduce the effect from the noise of the error we are using a forgetting sum (Table 4-4) of the last four values measured stored into the array “acc_12_X_gs_buf”. After that we use simple routine “shift_values (acc_12_X_gs,acc_12_X_gs_buf);” which shifts the values of the array by one for the next evaluation.

○ Y Axis:

```

acc_12_Y_high = (int)(data[5]);
char_to_int(acc_12_Y_high);
if(acc_12_Y_high==254)
    acc_12_Y_high=0;
acc_12_Y_low = (int)(data[6]);
char_to_int(acc_12_Y_low);
acc_12_Y_high=(256*acc_12_Y_high)+acc_12_Y_low;
acc_12_Y_gs = Acceleration_12(acc_12_Y_high,458);
acc_12_Y_gs=forgeting_sum(acc_12_Y_gs,acc_12_Y_gs_buf);
shift_values(acc_12_Y_gs,acc_12_Y_gs_buf);

```

Same as X axis.

■ Acceleration from the front accelerometer: ADXL311 ±2g Accelerometer

○ X Axis

```

acc_2_front_X_high = (int)(data[7]);
char_to_int(acc_2_front_X_high);
if(acc_2_front_X_high==254)
    acc_2_front_X_high=0;
acc_2_front_X_low = (int)(data[8]);
char_to_int(acc_2_front_X_low);
acc_2_front_X_gs=(256*acc_2_front_X_high)+acc_2_front_X_low;
acc_2_front_X_gs=((acc_2_front_X_gs * 5)/ 1024); // Se volt
acc_2_front_X_gs=((acc_2_front_X_gs-2.5)/0.312);
acc_2_front_X_gs=forgeting_sum(acc_2_front_X_gs,acc_2_front_X_gs_buf);
shift_values(acc_2_front_X_gs,acc_2_front_X_gs_buf);

```

The analog to digital converter results into a 10 bit measurement, where the 1023_d represents the voltage reference, which in our case is 5V and ant 0_d represents 0 volts. The formula for estimating the analog value was presented at “Observations, problems and possible improvements on the interface between the accelerometers and the microcontroller” section. Therefore the “((acc_2_front_X_gs * 5)/ 1024);” give the measurement in volts and the “” gives the measurement in Gs. The justification about the “acc_2_front_X_gs = forgetting_sum (acc_2_front_X_gs, acc_2_front_X_gs_buf);” and the “shift_values(acc_2_front_X_gs,

acc_2_front_X_gs_buf);” was presented for the ADXL213 X axis and are valid here too.

- Y Axis

```
acc_2_front_Y_high = (int)(data[9]);
char_to_int(acc_2_front_Y_high);
if(acc_2_front_Y_high==254)
    acc_2_front_Y_high=0;
acc_2_front_Y_low = (int)(data[10]);
char_to_int(acc_2_front_Y_low);
acc_2_front_Y_gs=(256*acc_2_front_Y_high)+acc_2_front_Y_low;
acc_2_front_Y_gs=((acc_2_front_Y_gs * 5)/ 1024); // Se volt twra sel 214 ATmega16;
acc_2_front_Y_gs=((acc_2_front_Y_gs-2.5)/0.312);
acc_2_front_Y_gs=forgeting_sum(acc_2_front_Y_gs,acc_2_front_Y_gs_buf);
shift_values(acc_2_front_Y_gs,acc_2_front_Y_gs_buf);
```

Same as ADXL311 X axis.

- Acceleration from the rear accelerometer: ADXL311±2g Accelerometer

- X Axis

```
acc_2_back_X_high = (int)(data[11]);
char_to_int(acc_2_back_X_high);
if(acc_2_back_X_high==254)
    acc_2_back_X_high=0;
acc_2_back_X_low = (int)(data[12]);
char_to_int(acc_2_back_X_low);
acc_2_back_X_gs=(256*acc_2_back_X_high)+acc_2_back_X_low;
acc_2_back_X_gs=((acc_2_back_X_gs * 5)/ 1024); // Se volt twra sel 214
acc_2_back_X_gs=((acc_2_back_X_gs-2.5)/0.312);
acc_2_back_X_gs=forgeting_sum(acc_2_back_X_gs,acc_2_back_X_gs_buf);
shift_values(acc_2_back_X_gs,acc_2_back_X_gs_buf);
```

Same as ADXL311 X axis.

- Y Axis

```
acc_2_back_Y_high = (int)(data[13]);
char_to_int(acc_2_back_Y_high);
if(acc_2_back_Y_high==254)
    acc_2_back_Y_high=0;
acc_2_back_Y_low = (int)(data[14]);
char_to_int(acc_2_back_Y_low);
acc_2_back_Y_gs=(256*acc_2_back_Y_high)+acc_2_back_Y_low;
acc_2_back_Y_gs=((acc_2_back_Y_gs * 5)/ 1024); acc_2_back_Y_gs=((acc_2_back_Y_gs-2.5)/0.312);
acc_2_back_Y_gs=forgeting_sum(acc_2_back_Y_gs,acc_2_back_Y_gs_buf);
shift_values(acc_2_back_Y_gs,acc_2_back_Y_gs_buf);
```

Same as ADXL311 X axis.

- Front left wheel angular velocity

```
front_left_ticks = (int)(data[15]);
char_to_int(front_left_ticks);
if(front_left_ticks==254)front_left_ticks=0;
front_left_speed=front_left_ticks*taxythta_ana_ticks;
```

- The evaluation of the wheel angular velocity was presented at the “Wheel angular velocity” section. Every clock “tick” represents longitudinal speed of 0.4625m/s. Thereupon the definition “#define taxythta_ana_ticks 0.4625” multiplied by the “ticks” gives the longitudinal velocity.

- Front right wheel angular velocity

```
front_right_ticks = (int)(data[16]);
char_to_int(front_right_ticks);
if(front_right_ticks==254)front_right_ticks=0; // An htan 254 einai 0
front_right_speed=front_right_ticks*taxythta_ana_ticks;
```


Same as front left wheel.

- Back left wheel angular velocity

```
back_left_ticks = (int)(data[17]);
char_to_int(back_left_ticks);
if(back_left_ticks==254)back_left_ticks=0; // An htan 254 einai 0
back_left_speed=back_left_ticks*taxythta_ana_ticks;
```

Same as front left wheel.

- Back right wheel angular velocity

```
back_right_ticks = (int)(data[18]);
char_to_int(back_right_ticks);
if(back_right_ticks==254)back_right_ticks=0; // An htan 254 einai 0
back_right_speed=back_right_ticks*taxythta_ana_ticks;
```

Same as front left wheel.

- Steering angle

```
// Pontesiometro gia gwnia timoniou
pont_high = (int)(data[19]);
char_to_int(pont_high);
if(pont_high==254)
    pont_high=0;
pont_low = (int)(data[20]);
char_to_int(pont_low);
pont_volts=(256*pont_high)+pont_low;
pont_volts=((pont_volts * 5)/ 1024); // Se volt twra sel 214
ATmega16;
pont_volts=forgeting_sum_steer(pont_volts,pont_volts_buf);
shift_values(pont_volts,pont_volts_buf);
steer=1000*( (0.31570260254613*pow(pont_volts,3)) + (-
1.97189807334590*pow(pont_volts,2))+ (3.98997966117411 *pow(pont_volts,1)) -2.60471114838461);
steer-=1;
```

- The steering angle is evaluated through a third degree polynomial as mentioned into the corresponding section. Thus after we compute the voltage across the potentiometer we can insert that value into the polynomial and estimate the steering angle.

- Yaw rate

```
// Gyroskopio
gyro_high = (int)(data[21]);
char_to_int(gyro_high);
if(gyro_high==254)
    gyro_high=0;
gyro_low = (int)(data[22]);
char_to_int(gyro_low);
gyro_volts=(256*gyro_high)+gyro_low;
gyro_volts=((gyro_volts * 5)/ 1024); // Se volt twra sel 214 ATmega16;
gyro_volts=forgeting_sum(gyro_volts,gyro_volts_buf);
shift_values(gyro_volts,gyro_volts_buf);
// To gyroscopio exei 5mv/ana sec eyais8hsia
gyro_degs_ana_sec=floor(((gyro_volts-2.50)/0.005)); // Ta 2.866V einai se 0 moires/sec To
apotelesma einai se moires/sec
gyro_degs_ana_sec++;
```

- Battery health

```
// Mpataria
batt_high = (int)(data[23]);
char_to_int(batt_high);
if(batt_high==254)
    batt_high=0;
batt_low = (int)(data[24]);
char_to_int(batt_low);
batt_volts=(256*batt_high)+batt_low;
```



```
batt_volts=((batt_volts * 5)/ 1024); // Se volt twra sel 214 ATmega16;
batt_volts=batt_volts*2; //Brisketai se diaireth tashs
```

The battery health is just the analog voltage at the corresponding pin of the microcontroller. The only strange thing is that the value is multiplied by two and the explanation is simple; the voltage passes through a voltage divider (Fig. 4.37).

- Steer servo

```
// Ta throttle kai steer servo ta stelnw anapoda apo to atmega. 25, kai 26 antistoixa
// steer_servo
steer_servo = (int)(data[25]);
char_to_int(steer_servo);
if(steer_servo==254)
    steer_servo=0;
```

- Throttle servo

```
// throttle_servo
throttle_servo = (int)(data[26]);
char_to_int(throttle_servo);
if(throttle_servo==254)
    throttle_servo=0;
```

For some of the above conversions as we have the earlier mentioned, we can see the pattern “if(value==254) then value is 0”. For all those data we have placed this “if – else” we know that will never reach the value of 254_d and that whenever we come across it, it has derived from a binary “0” (EOF for C) that we didn’t want to transmit. For example the pulse of the throttle servo arriving at the receiver, sampled by the ATmega32 and transmitted to the SBC will never be zero, but will have a value between 20 and 70.

Auto generated experiments function

In order to have the potential to compare the behaviour of the car in a valid manner, with ESP system on or off and/or while adjusting some parameters we can program the car to perform the same experiment multiple times. For example, we might program the car to steer 10 degrees left when it would reach the velocity of 5m/s and hold than steering for 0.5 seconds. The command is given by the driver at the Java GUI on the browser, from where it is transmitted to the SBC and passed as an argument to the *serial_com()* routine. Something that hasn’t been mentioned so far is how the time is estimated. It is crucial for us to have an accurate measurement of time, while the *gettimeofday()* function returns time with 10msec resolution which is very low. At Table 4-14 we see a typical instance for time estimation in microseconds we have used ([57]). Through this simple code at Table 4-14 have accuracy of micro second for time readings.

```
struct timeval tp;
gettimeofday(&tp,0);
time_end = ((unsigned)tp.tv_sec)*1000000+((unsigned)tp.tv_usec);
```

Table 4-14 Time measurement in usec.

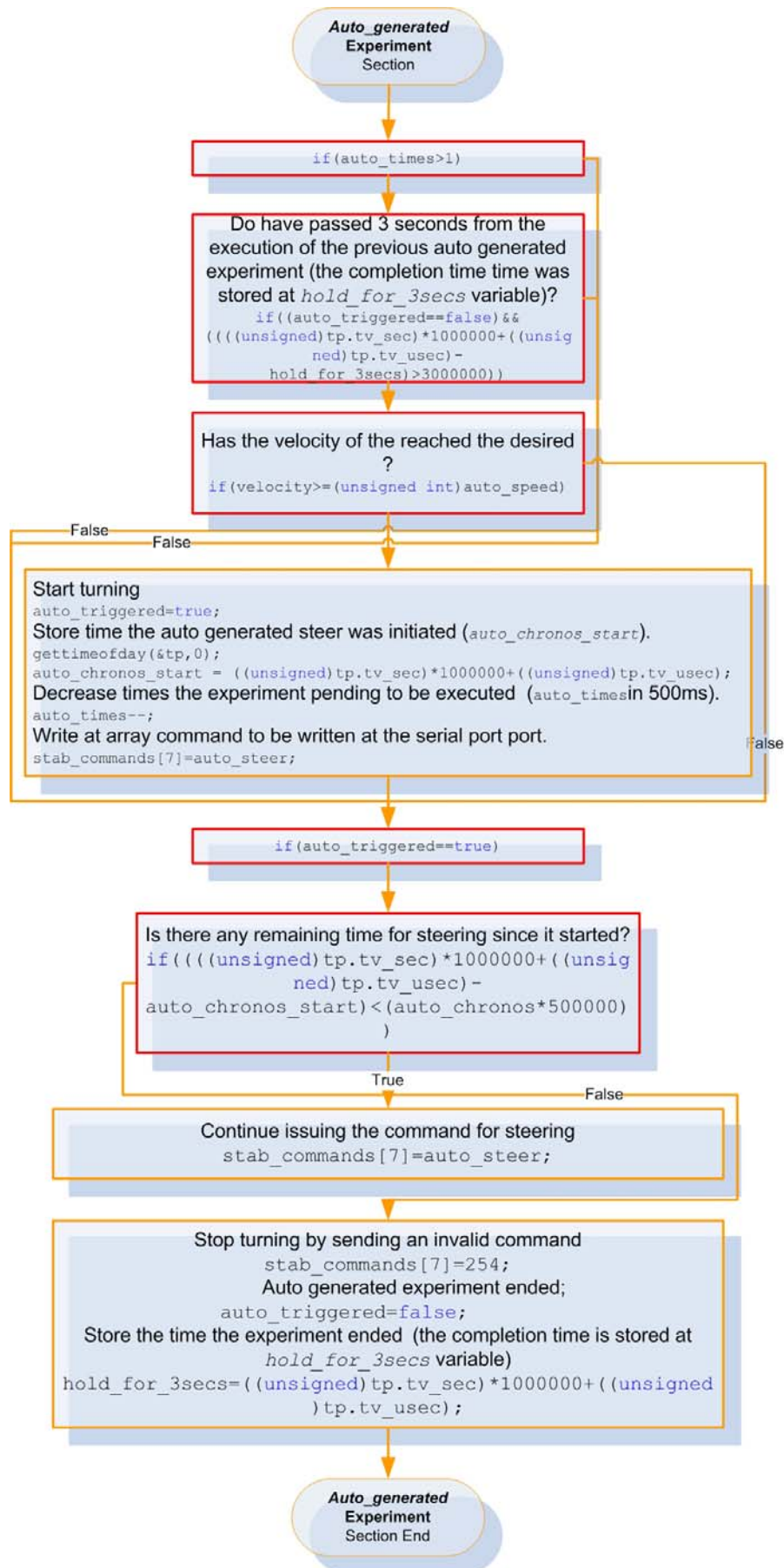


Fig. 4.53 Auto generated experiments flow chart.

The variables necessary for the program to conduct an experiment are:

- `unsigned char auto_speed=0;`
 - Starts turning when it reaches the *auto_speed*;
- `unsigned char auto_steer=0;`
 - Turn by *auto_steer* (in steer servo pulses, see Driver's Commands section).
- `unsigned char auto_chronos=0;`
 - Hold that steering *auto_chronos**500msec.
- `unsigned char auto_times=0;`
 - Repeat experiment for *auto_times* times.
- `long auto_chronos_start=0;`
 - Trigger time counting and hold that value at *auto_chronos_start*.
- `long hold_for_3secs=0;`
 - Wait *hold_for_3secs* seconds between a new experiment.
- `bool auto_triggered=false;`
 - Boolean flag.
- `int auto_trig=0;`
 - Flag.

When those values reach the point where the auto generated experiments are controlled, we reach at the flow chart for the auto generated experiments, which is illustrated at Fig. 4.53.

4.3.3 Stabilization routines

The user of the system can select up to eight different ESC stabilization algorithms from the Java GUI. In our implementation, we have used just three different routines where the one is based upon the ESC algorithm presented at the third chapter, the second is a single accelerometer ESC routine and the third is a hybrid method combining some characteristics from the previous two. Since the ESC algorithm we have evaluated most and was working unexceptionably, we shall present only this specific stabilization routine.

The routine is called "`char * stabilization_1(double front_left_speed ...)`", and takes as inputs most of the measurements transmitted from the ATmega32, like the yaw rate, individual wheel speed, steering angle, the sensitivity and understeer gradient set by the user at the Java GUI (by default are 0.9 and 0.004 respectively). The routine returns a pointer to a char array. This array

contains the commands for braking that will be transmitted to the microcontroller through the serial port. The flow chart of the routine is portrayed at Fig. 4.54.

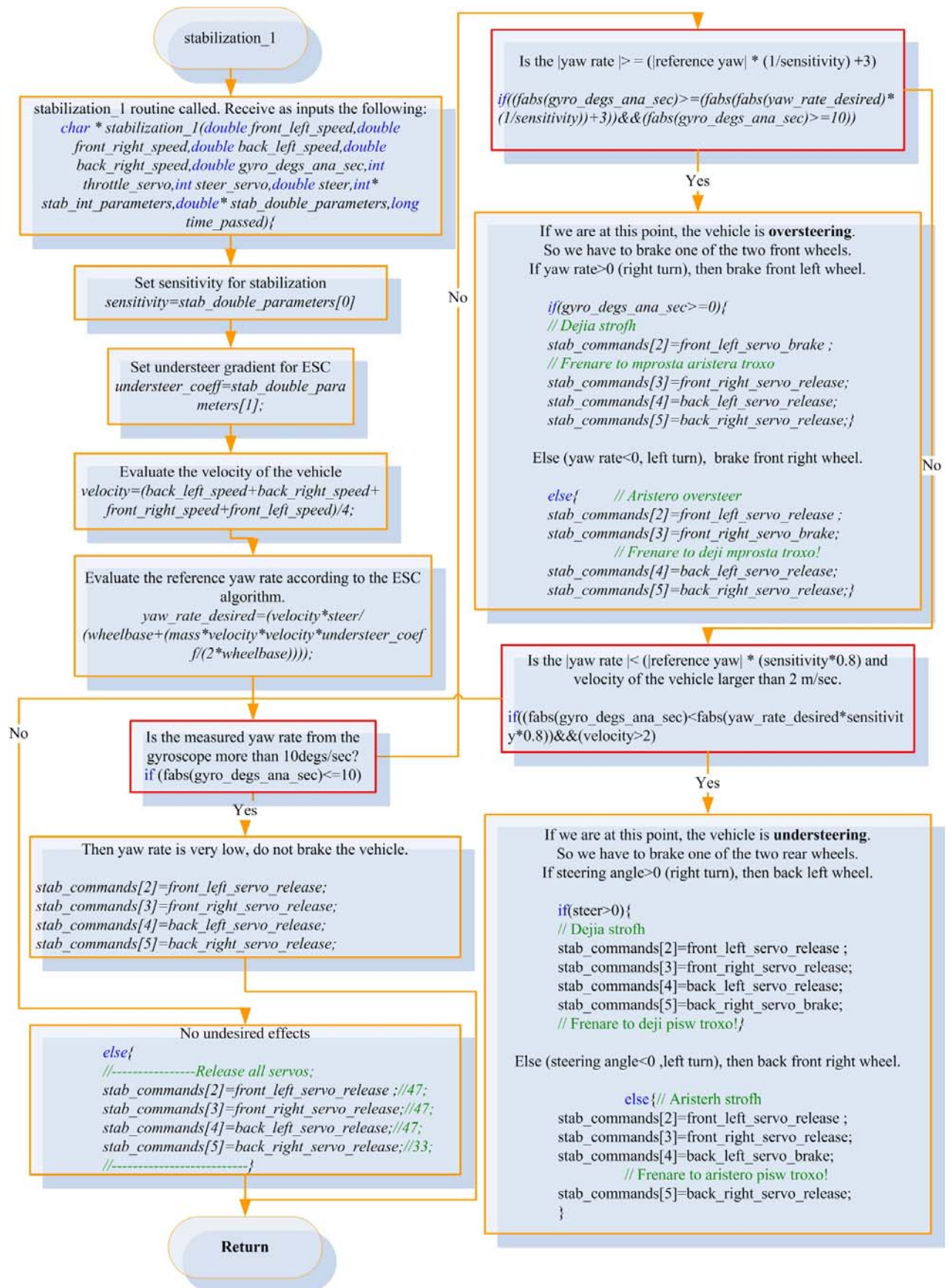


Fig. 4.54 Flow chart of the stabilization routine “*stabilization_1*”.

4.3.4 Scripts

In order to reduce the development time, for example compilation and linking of several programs and libraries, we have developed certain bash scripts that will do routine tasks. An example of a script that links the program the author developed along with the socketserver developed by Mr. Hatzidiakos for the Java interface, and executes the compiled program (after it prompts the user for this operation) is presented at Table 4-15. The only thing the user has to do, is do the script, named “*compile*” executable by changing its attributes with UNIX command “*chmod 755 compile*”.

```
#step 1
#set correct local address in defines.h //define LADDR "192.168.86.2"

#step 2
#to compile run
echo "Compiling: g++ -c async.cpp socketserver.cpp"
g++ -c async.cpp socketserver.cpp
echo "Linking: g++ async.o socketserver.o -lpthread -o socketserver"
g++ async.o socketserver.o -lpthread -o socketserver
echo "Done!"
echo "Do you want me to execute y or n?"
read choice
if [ $choice = "y" ]
    then
        clear
        echo "Yes sellected, Socketserver running!"
        ./socketserver
    else
        echo "Bye!"
fi

#step 3
#run
```

Table 4-15 Compilation script.

Scripts like the above have been used widely and are really helpful.

4.4 Firmware at the Microcontroller

The firmware for the microcontroller was developed in C language, with WinAVR ([15]) a suite of executable, open source software development tools for the Atmel AVR series of RISC microcontrollers. WinAVR includes the GNU GCC compiler for C and C++. The software in total is free and available in WWW.

The development of the software, was quite demanding, since the computing resources are limited, and had to deal with time sensitive applications, like PWM pulses, USART etc, so we had to be extremely careful with the usage of the whole system.

The motive for the development of the ATmega32 firmware was the same as the SBC software: source code packed into as many more different routines, spread to many different C files. The aim was to build an easily readable, easily expandable well documented source code that would be portable to any microcontroller after a few alterations. The firmware consisted of many different files. Those files are included from the C file, named “*ESC_new.c*”, where the “main” loop of the firmware lies:

- `#include <avr/io.h>`
- `#include <math.h>`
- `#include <avr/interrupt.h>`
- `#include <avr/pgmspace.h>`
- `#include <ctype.h>`
- `#include <avr/sfr_defs.h>`
- `#include "files/adc_accelerometers.c"`
- `#include "files/interrupt_routines.c"`
- `#include "files/servos.h"`
- `#include "header.h"`

The first six inclusions inside the tags (<>), are built in header files from the compiler, and the rest inside the quotes (“”), are the developed firmware. There are also some external dependencies files, which are included automatically inside the project of the AVR studio, when we select the ATmega32 as the target microcontroller. The most important file from those external dependencies, is the “*iom32.h*”, which is the mapping between the IO ports and control variables for the ATmega32, which in general are registers that have a certain address. It is really useful, since it has all the necessary definitions for accessing everything at the microcontroller. If we

hadn't used the mapping between the registers and names, the access to pin 5 from PORTA of the ATmega32, would have to be accessed like "\$1B&=~(1<<5);" contrary to "PORTA&=~(1<<PA5); // PA5 low", which is more legible.

4.4.1 Main loop

As mentioned at the previous section, the main loop of the firmware lies inside the file "*ESC_new.c*". The implemented system, is a Real Time system and its operation is based on the interrupts. Most of the processing and control is taken place inside the interrupt routines.

The whole idea around Real Time system and interrupts is that the firmware is running on a continuous loop doing basic operations, which are not time critical tasks. When an interrupt is signalled, then the program jumps immediately from the normal execution of the program, to the corresponding interrupt service routine. For example suppose we have an initiation at serial communication between the SBC and the ATmega32. Instead of polling (continuous asking) the corresponding bit of a register which indicates that a byte has been received, we have programmed the microcontroller to signal an USART received complete interrupt (*SIGNAL (SIG_UART_RECV)*). The real time interrupt driven method, secures that everything is executed and served within restraint time margins. We have asynchronous operation, and no execution is blocked waiting for another one to finish. If an interrupt is signalled when the microcontroller executes the ISR (interrupt service routine) of another, then this is held into the appropriate register (stored into a stack). Thus, when it completes the current execution, it jumps to corresponding routine of the new interrupt. The flow chart of Fig. 4.55 shows the progress of an interrupt execution instance.

We shall present the operation of the system in flow chart form, with samples of source code inside. The flow chart of the main loop is illustrated at Fig. 4.56.

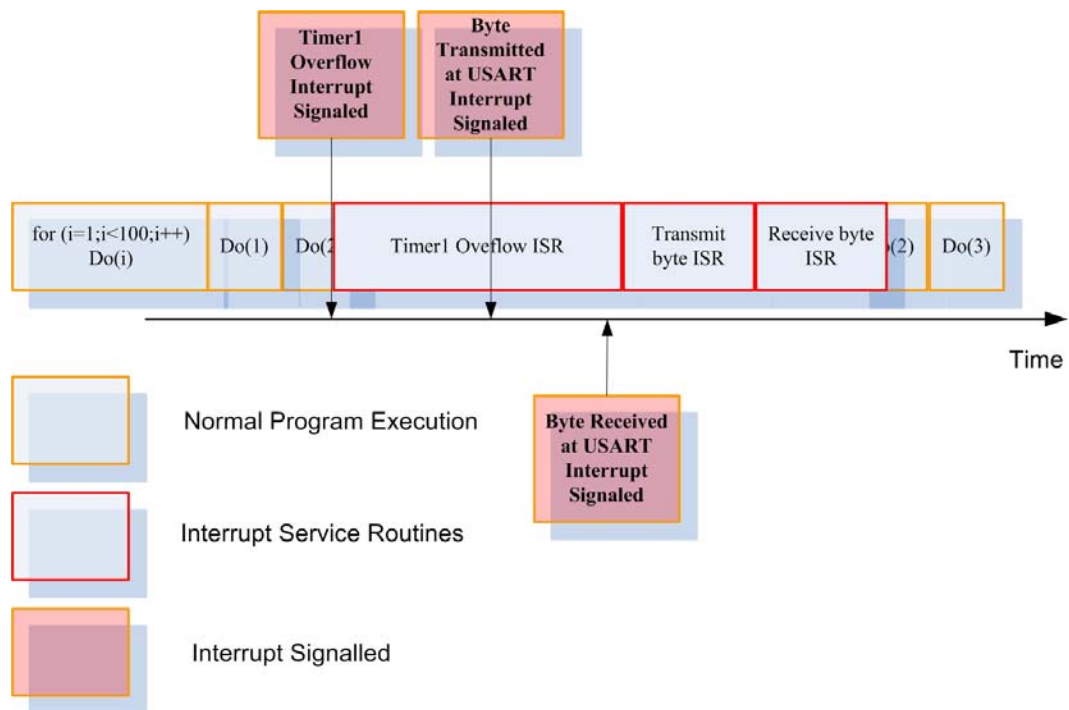


Fig. 4.55 Real time Interrupt driven firmware execution.

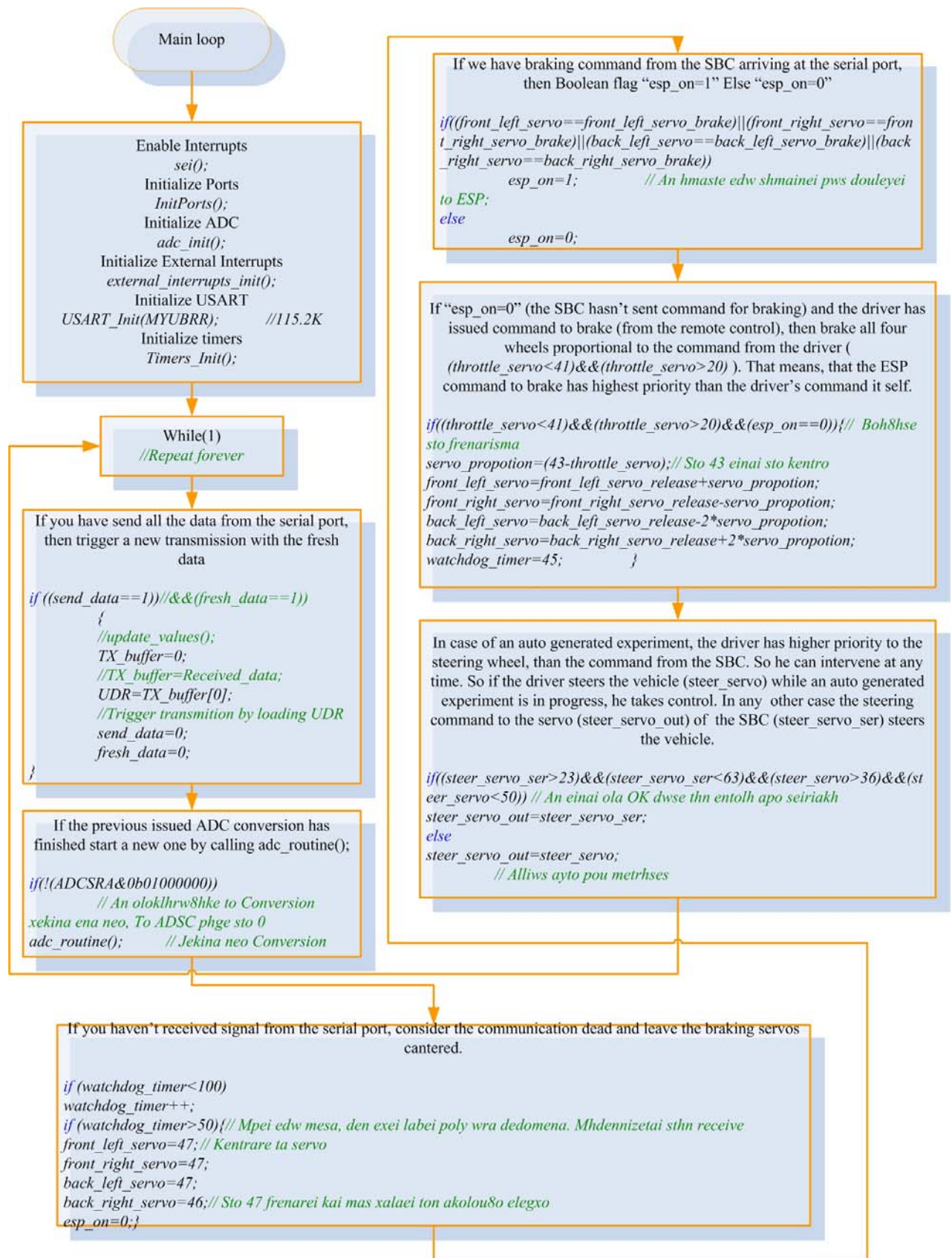


Fig. 4.56 Flow chart of “main loop” inside firmware.

As illustrated on both Fig. 4.55 and Fig. 4.56, the firmware depends on various routines; normal and Interrupt Service Routines. Most of the necessary initialization is held inside the routines. The normal routines inside the firmware are:

Normal Routines:

- `void InitPorts(void);`
- `void USART_Init(unsigned int ubrr);`
- `void Timers_Init(void);`
- `void external_interrupts_init(void);`
- `void adc_init(void);`
- `void adc_routine(void);`

4.4.2 Normal Routines

The operation of each routine will be explained with the corresponding flow chart. The first two to be explained are the InitPorts and USART_Init (Fig. 4.57).

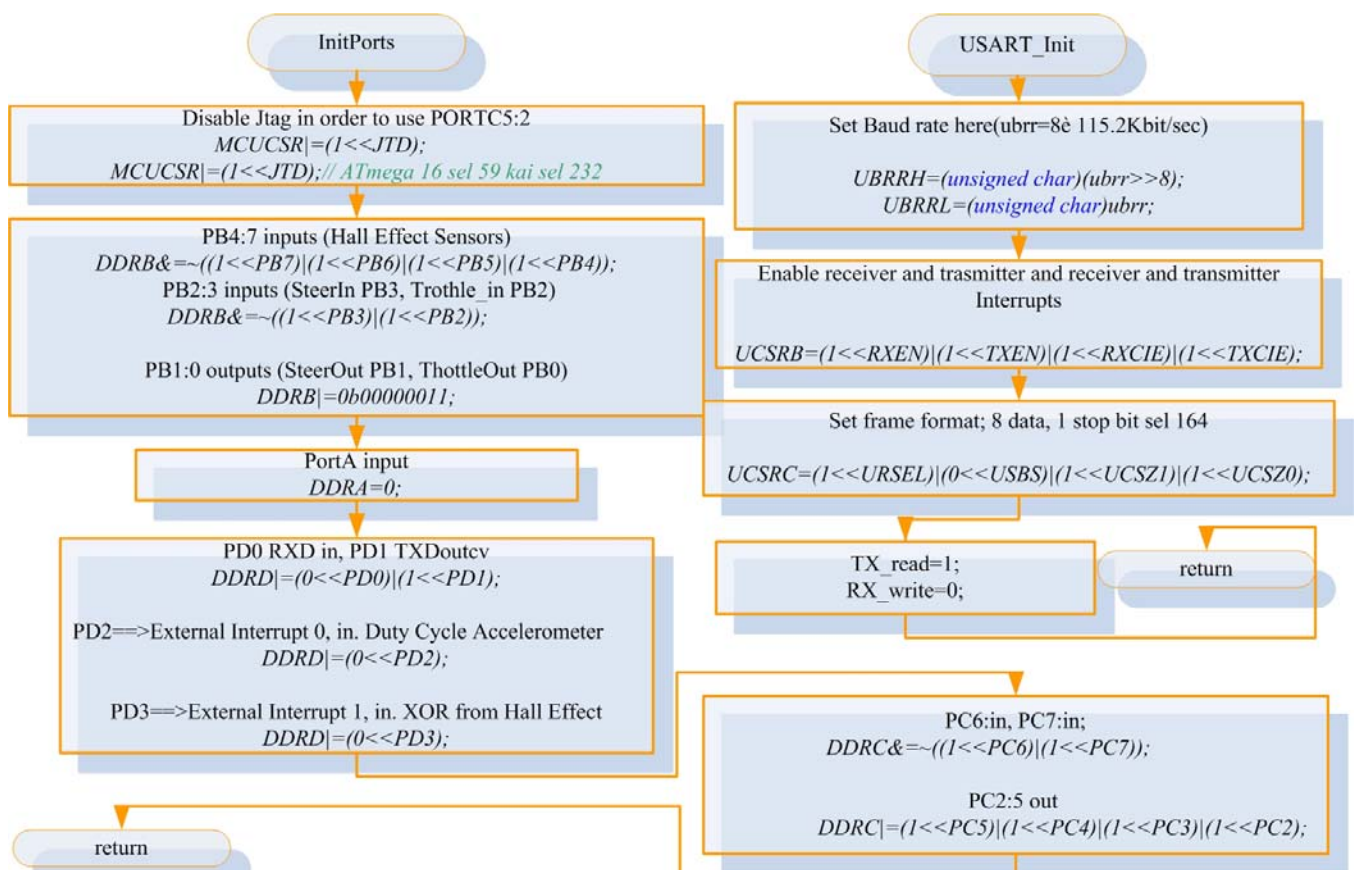


Fig. 4.57 InitPorts() routine (Initialization of ports) left and USART_Init() (Initialization of Universal Asynchronous Receiver Transmitter) right.

The *InitPorts()* routine, initializes the direction of the ports for the microcontroller (Fig. 4.57). The *USART_Init()*, initializes the universal asynchronous receiver transmitter of the ATmega32

for the serial communication with the SBC (Fig. 4.57). It sets the BAUD rate, enables receiver and transmitter interrupts and triggers a transmission.

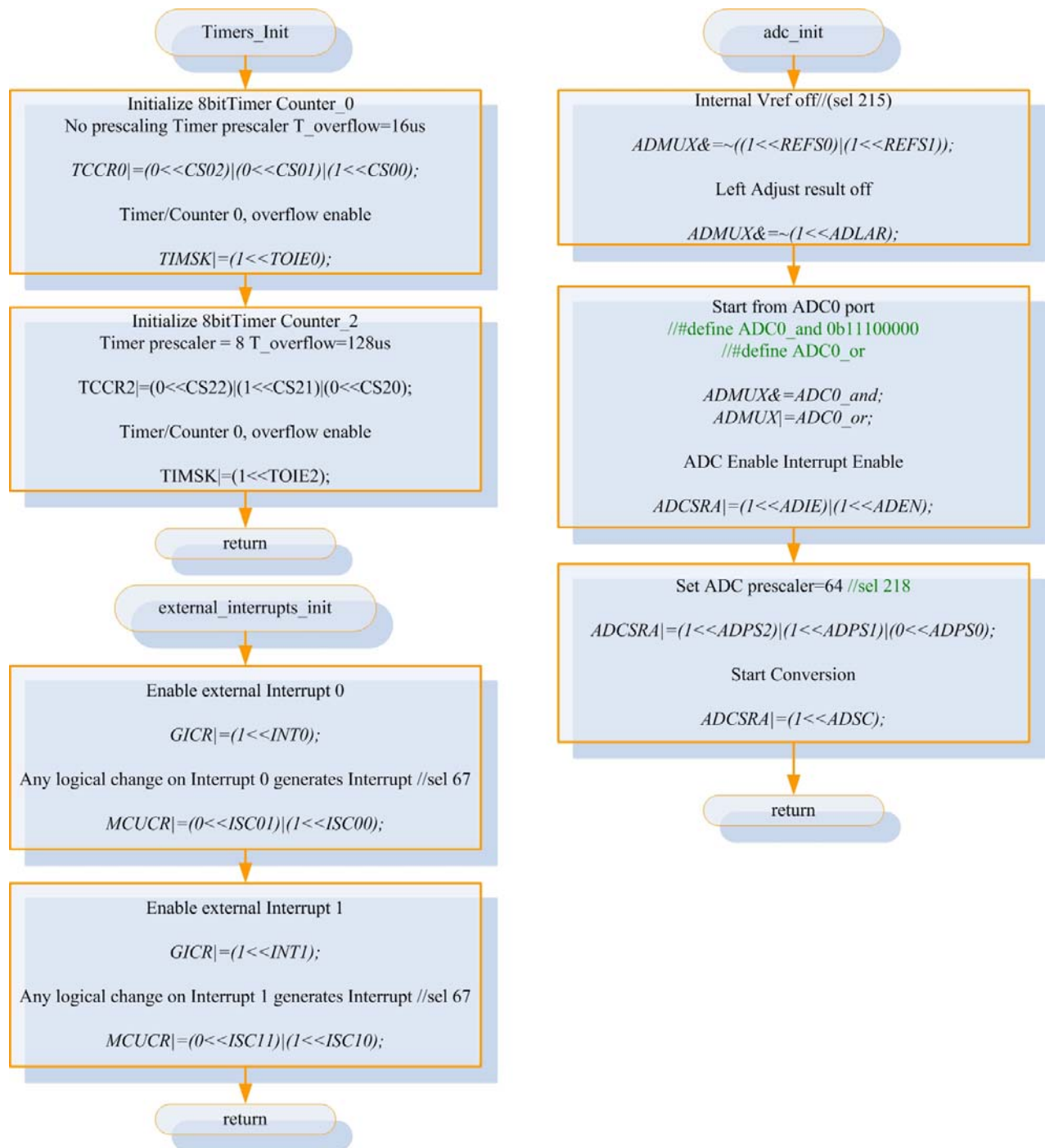


Fig. 4.58 Timers_Init(), external_interrupts_init() and adc_init() routines flow charts.

The *Timers_Init()*, initializes the 8 – bit Timer/Counter0 and Timer/Counter 2 counters of the microcontroller, using a 0 and 8 prescaling respectively and enables the timer overflow interrupt (Fig. 4.58). That means that we have an overflow interrupt every 256 clock cycles → 16usec for the Timer/Counter0 and 256*8 clock cycles → 128usec for the Timer/Counter1.

The *external_interrupts_init()*, enables both external interrupts, on PD2 and PD3 pin to be triggered with any logical change (Fig. 4.58). The *adc_init()*, commands the microcontroller to use as voltage reference, the analog voltage connected to the AREF pin of the ATmega32, align the result right, sets the source of analog voltage to sample (PA0 pin) with the ADMUX register and enables the ADC interrupt (Fig. 4.58). So whenever a conversion is complete we have an interrupt. It also arranges the ADC clock prescaler to 64 → we have an increase at the ADC clock every 64 clock cycles, at takes 13 ADC clocks to complete a conversion. Finally it initiates the first conversion.

We have dedicated a routine for the ADC conversion called *adc_routine()*. The function of this routine is to change cyclically the ADC input source from PA0 to PA6.

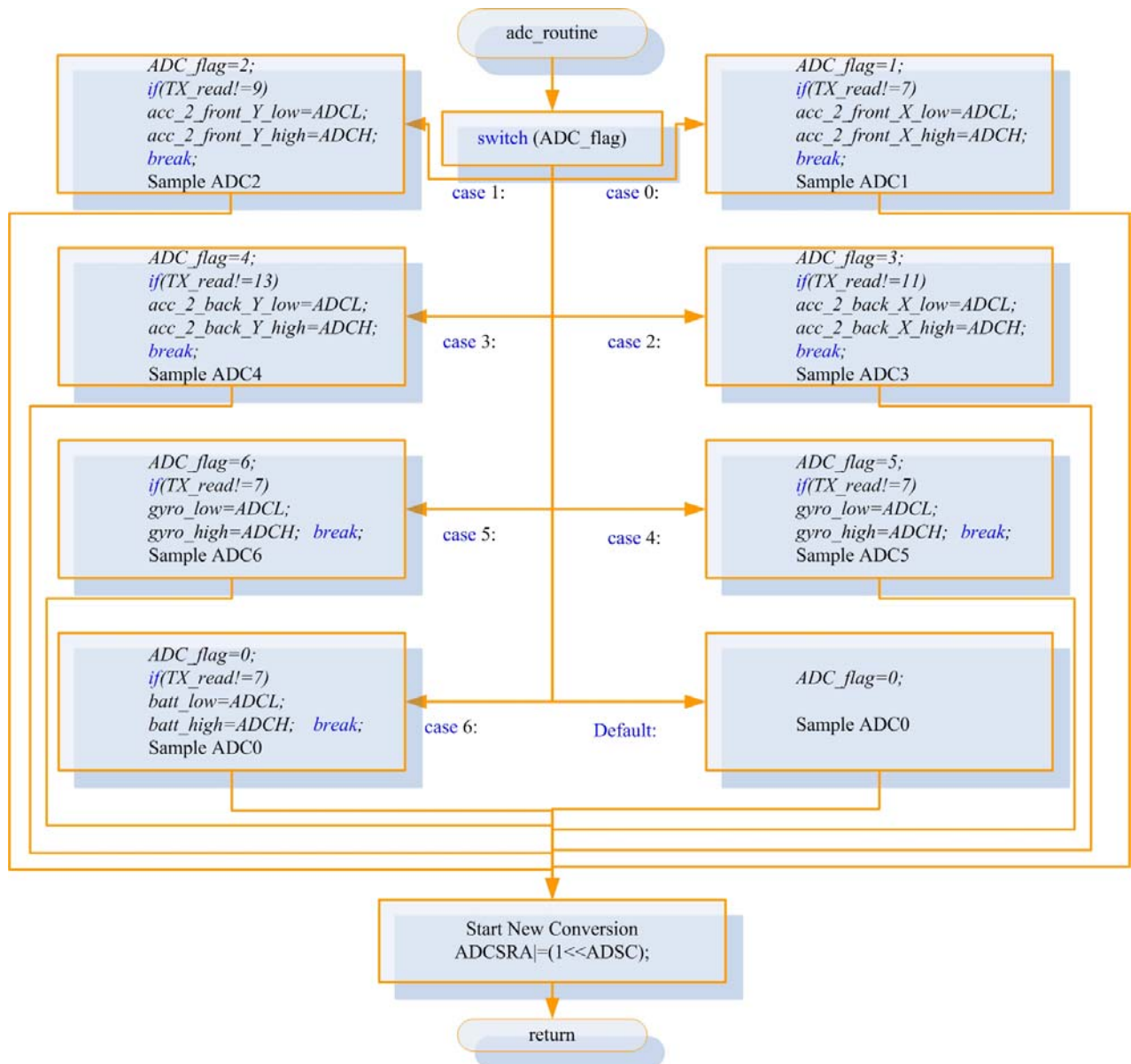


Fig. 4.59 Routine: *adc_routine()* flow chart.

The *adc_routine()*, also performs a check to the status of the bytes sent (*TX_read* at Fig. 4.58) at the serial port. So, when the higher nibble from the 10 – bit analog to digital conversion of a source has been sent, for example the voltage of the battery, then it doesn't change the value of the lower nibble and holds the previous value for consistency reasons. This doesn't happen often but in the occasion it does, it might ruin completely a measurement. Let's say that the higher nibble from an analog to digital conversion is in hexadecimal form 01_h and the lower 00_h. Suppose that on the next conversion the higher nibble is 00_h and the lower is FF_h. That means that the value changed 1 bit. If the old higher nibble was sent before the new update of value and the lower after the update, the value that would reach at the SBC through the serial would be 01FF_h➔ and not the 00FF which is more or less the double value. So a simple if – else pattern is enough to prevent the situation.

4.4.3 Interrupt Routines:

The *SIGNAL(SIG_UART_RECV)* is the interrupt routine signalled when a byte of information has been received at the USART of the microcontroller (Fig. 4.60). The information arriving from the SBC to the microcontroller is the desired position of each servo working on the system. The character FF_{hex} is used as terminator for the data packet. That means that a packet of info starts and ends with this byte. The flow chart is quite revealing concerning the operation of the routine (Fig. 4.60).

The *SIGNAL(SIG_UART_TRANS)* is the interrupt routine which is signalled every time a byte has been transmitted at the serial port (Fig. 4.61). The data packet transmitted is 32 bytes long and is initiated and terminated with the character FF_{hex}. The routine is responsible for the cycling rotation of the data to be transmitted, sending the data in the correct order. It also assures that the data byte transmitted is not the terminating character (FF_{hex}), changing the value in a way that can be detected by the receiving SBC and can be restored. It also reassures that it doesn't transmit the 00_{hex} because this is the end of file for the C language. One example is the wheel speed. In case the speed is 0_{dec}, it sends the 254_{dec} which is an invalid situation for the vehicle (it would denote the velocity of 400Km/hour). Thereafter, at reception, the program can restore the original value.

The *SIGNAL(TIMER0_OVF_vect)* is signalled every time we have overflow at the 8 bit Timer/Counter0 (Fig. 4.62). The counter increases with a ratio denoted by its corresponding prescaler (Fig. 4.58). For this counter, we have used 0 prescaling, thus we have overflow every 16usec. The operation of this routine is crucial. It is executed more often than every other routine in the firmware. It samples the pulse for the ADXL213, samples the commands from the driver

arriving at the receiver, and controls all six servos. It also performs check over the sampled duty cycle of the driver's command. If the duty cycle is invalid, it sends commands to the servos, that would put them into a centered position.

The *SIGNAL(TIMERC2_OVF_vect)* is the interrupt routine for the 8 bit Timer/Counter2 Overflow (Fig. 4.63). The prescaler for the Timer/Counter2 has been set that way, so that the routine is signalled every 128usec. The main operation for this routine is the determination of individual wheel speed. It stores the number of the logical transition for each wheel every 0.1sec into a dedicated variable and zero the counters for the following update. Also, the *SIGNAL(INT0_vect)*, is External Interrupt Request 0, dedicated for the wheel speeds (Fig. 4.65). A more detailed description for both routines has been presented at the section “Wheel angular velocity”.

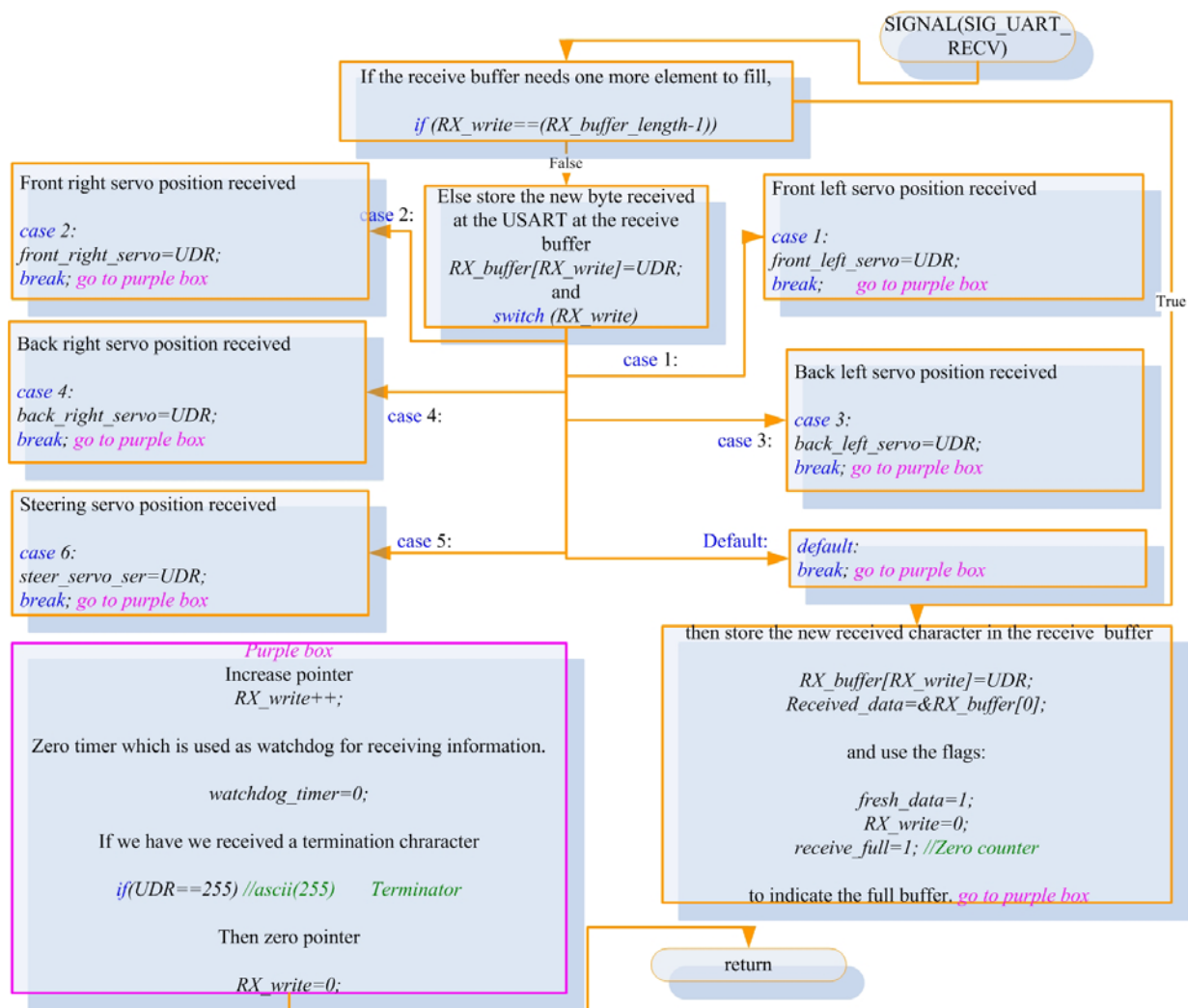


Fig. 4.60 Interrupt routine flow chart of SIGNAL(SIG_UART_RECV): USART, Rx Complete.

Finally, the interrupt routine *SIGNAL(INT1_vect)*, is the External Interrupt Request 1 which interfaces the ADXL213 accelerometer with the ATmega32 (Fig. 4.64). A detailed description has been cited at the section “ADXL213 $\pm 1.2g$ interface with the microcontroller”.



Fig. 4.61 Interrupt routine flow chart of SIGNAL(SIG_UART_TRANS): USART, Tx Complete.

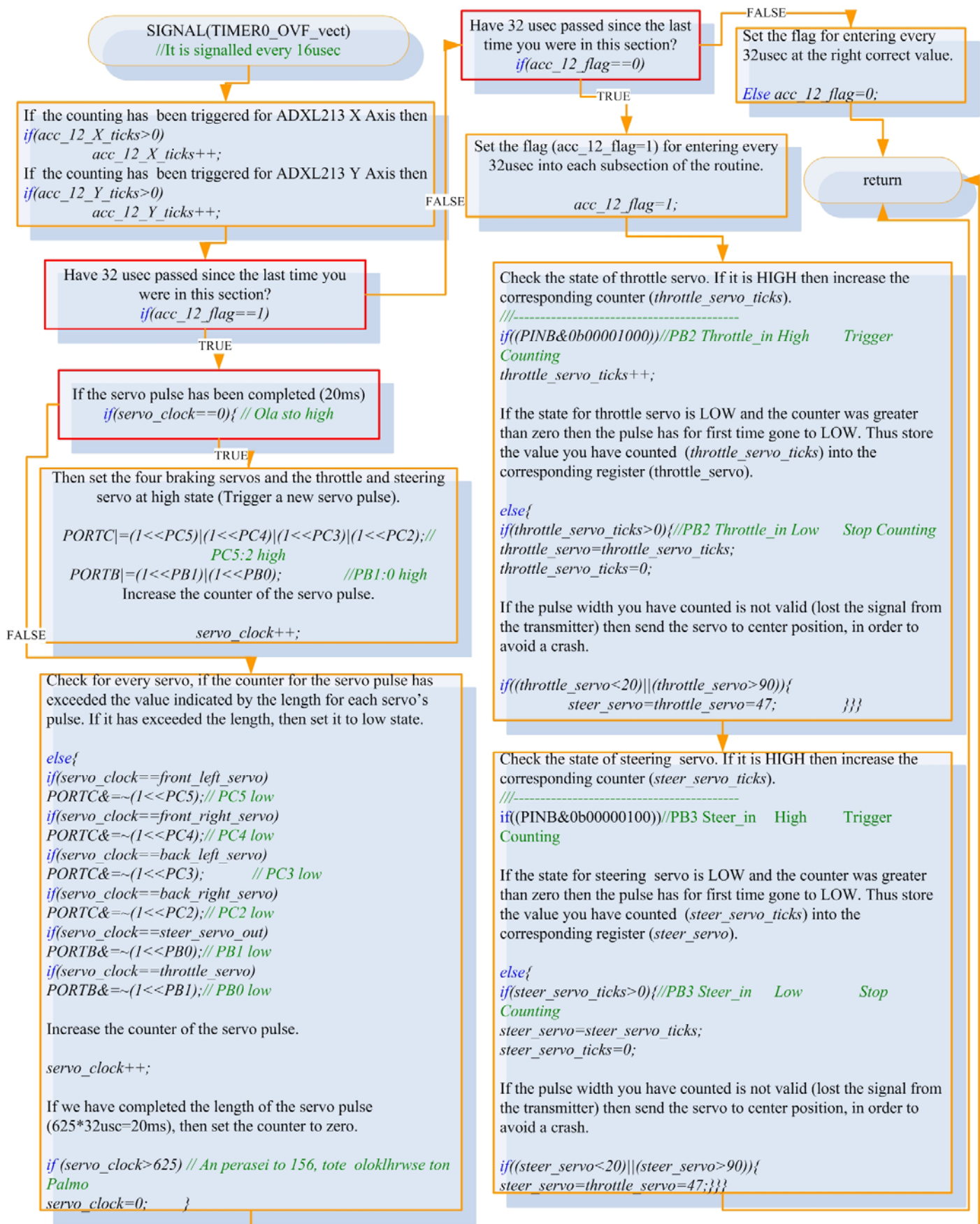


Fig. 4.62 Interrupt routine flow chart of SIGNAL(TIMERO_OVF_vect): Timer/Counter0 Overflow.

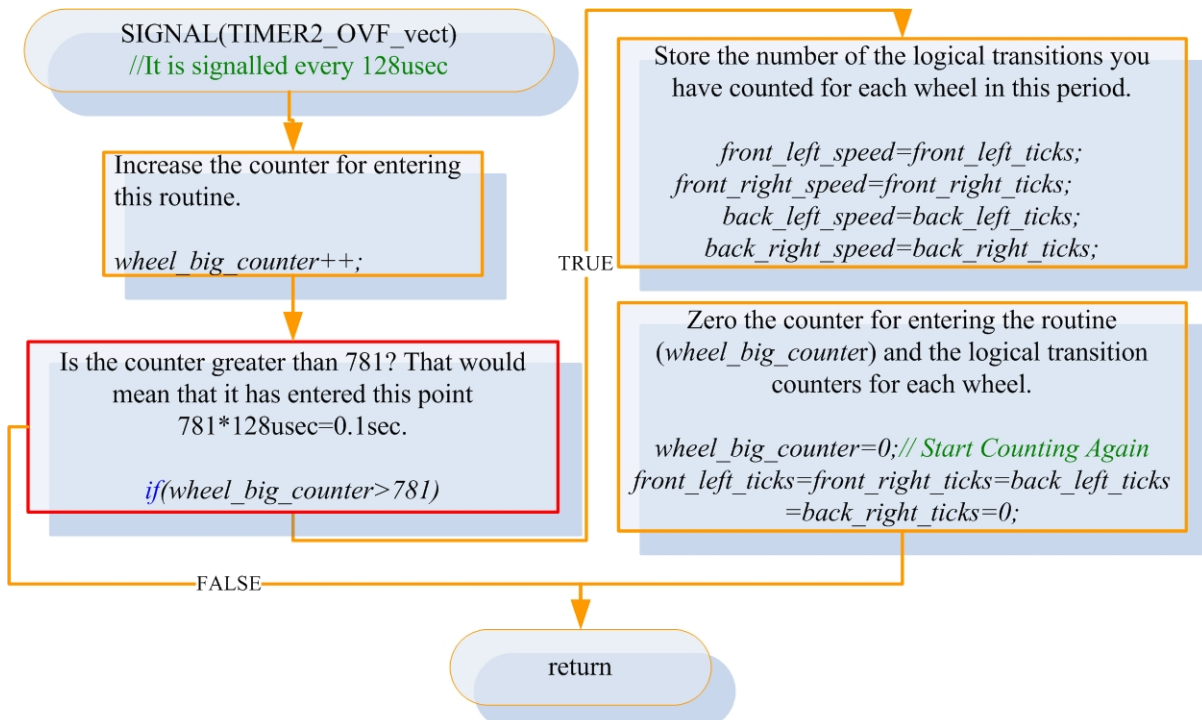


Fig. 4.63 Interrupt routine flow chart of SIGNAL(TIMER2_OVF_vect): Timer/Counter2 Overflow.

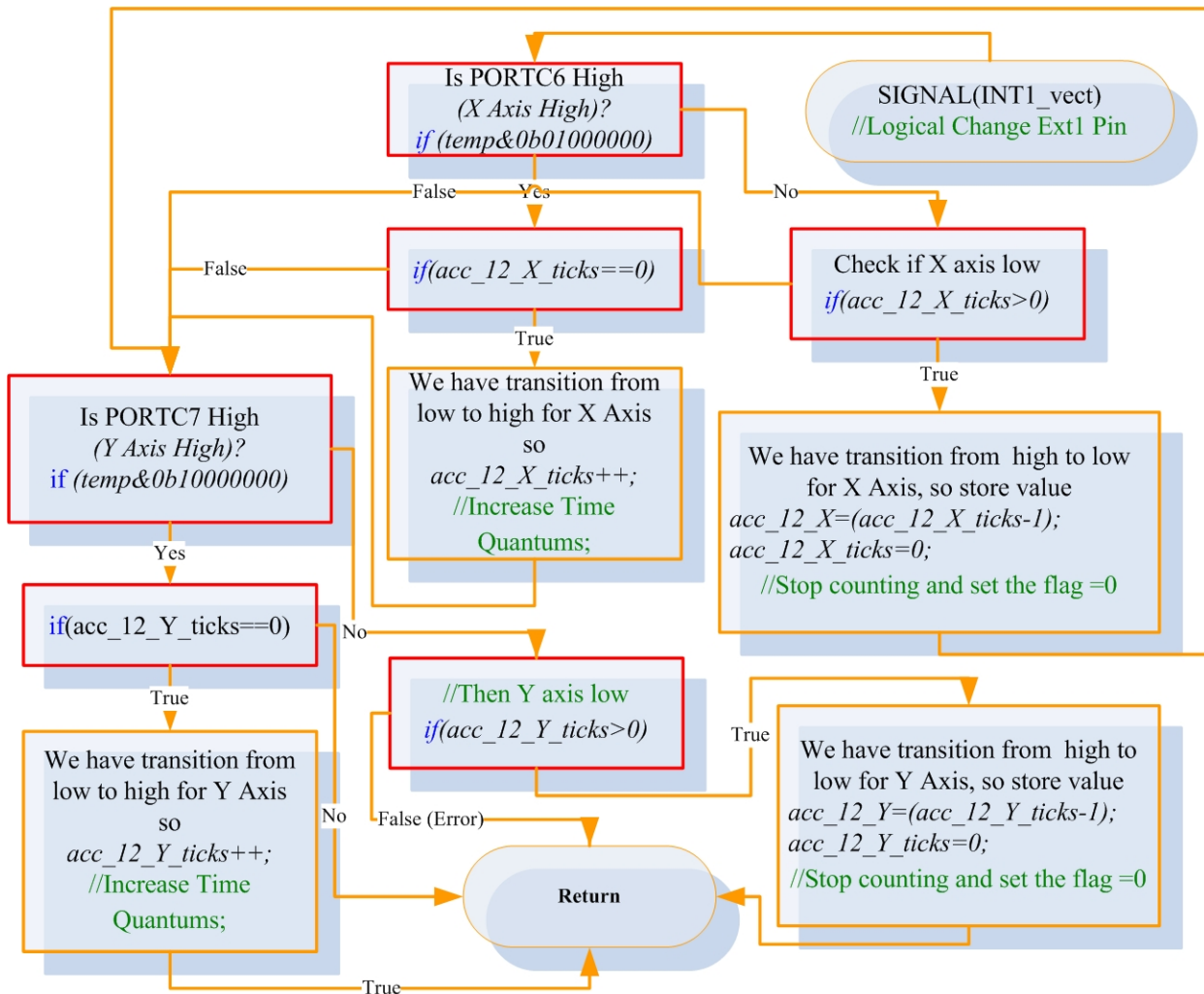


Fig. 4.64 Interrupt routine flow chart of SIGNAL(INT1_vect): External Interrupt Request 1 → ADXL213 Accelerometer.

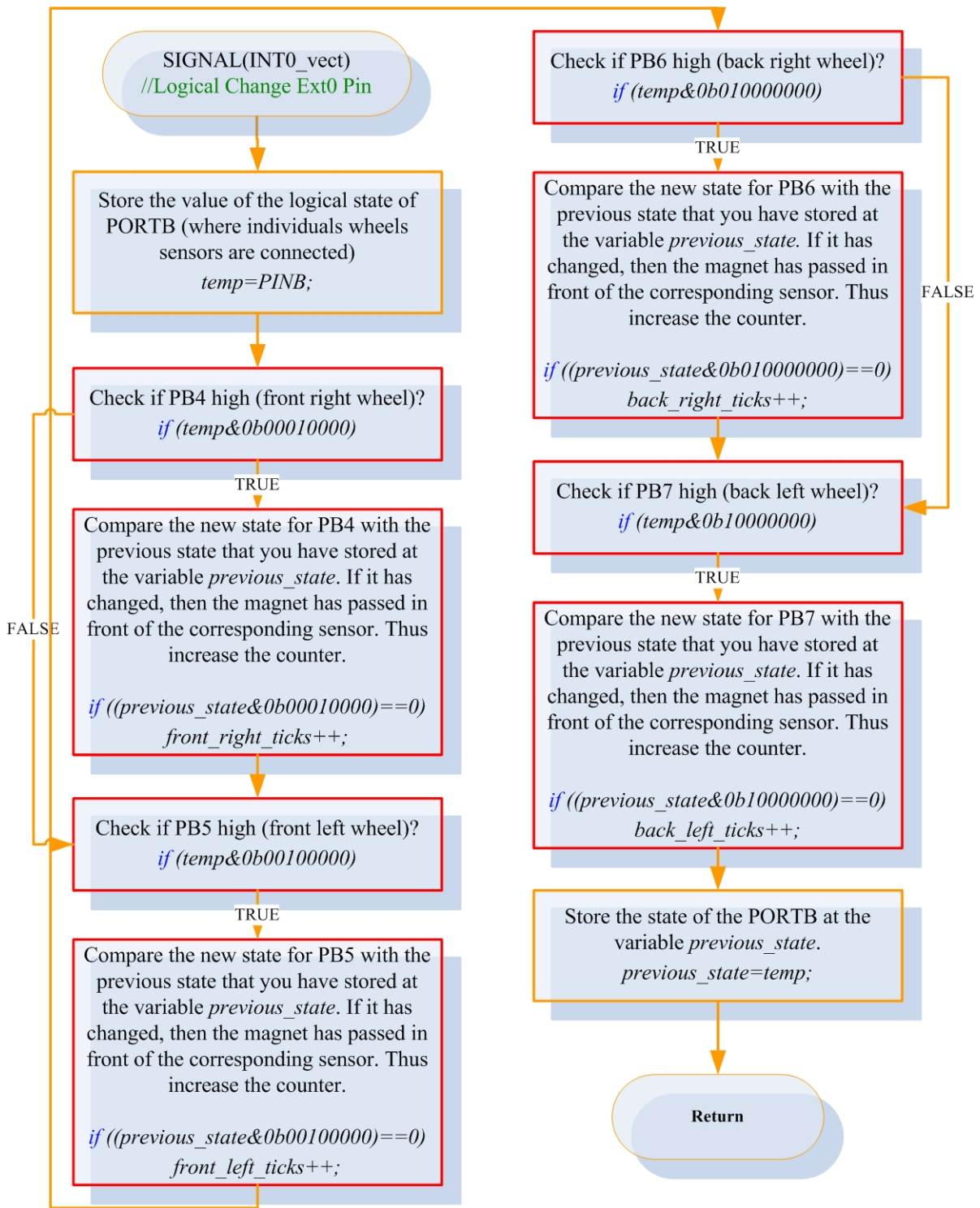


Fig. 4.65 Interrupt routine flow chart of SIGNAL(INT0_vect): External Interrupt Request 0 → Wheel speeds.

4.5 How to operate the system

The setup and running of the system has been scattered into different sections of this document. The “Custom developed source code” section has a wide description for the operation. At this section, we shall concentrate all the necessary steps from the powering up of the system, until the log file extraction.

1. Connect the power cable and press the power push button to switch on the system.
2. Connect through a wireless connection to the Linksys router using Static IP Addressing.
3. Use an SSH client (PUTTY ([53])) to connect to the SSH server of the SBC. Connect to IP: 192.168.1.101.
4. Execute the socket server from the folder it is stored at the SBC.
5. With a web browser, connect from the client computer; connect to the address 192.168.1.101. The client computer should have Java Runtime Environment Installed.
6. At web browser of the client computer, the index.html will be loaded by default. This HTML code initiates three Java applets (Table 4-8). The first is the Java GUI, and the other two are two Java Applets for the real time video streaming. The outcome is displayed at Fig. 4.45.
7. The user in order to connect to the socket server running at the SBC, must press the connect button and everything is ready for experimenting using different ESC schemes, adjusting parameters and fine tuning the ESC algorithms.

From this point on, at the SBC the socket server is exchanging data with the browser and is generating a log file with all the operation, from the data gathered to the commands issued, which can be accessed from the browser through a hyperlink.

5. Real Environment Evaluation

The improvement on the vehicle's behaviour with the ESC algorithm in function, based on a single Gyroscope we presented at Fig. 3.4, was downright. We used an appropriate fitted reference yaw rate function of velocity and steering angle (Fig. 3.4). After some trial runs, and the infallible method of trial and error, we managed to estimate nice values for the parameters of the algorithm, for increased performance. If the error between the reference and the actual yaw rate measured from the gyro exceeded a certain amount then the system would detect oversteer or understeer and would apply brakes on the most effective wheel to counteract the effect.

In a few words everything worked nice! Both hardware and software worked in a robust manner. In fact the battery for the SBC the router and some more peripherals, would last twice the battery of the author's laptop which became the bottleneck for the experiments. Besides the intense distress and harassment of the system, everything stayed in place and the VIA SBC deserves honourable mention.

We can concentrate the performance of the whole system, both testbed and the algorithm in a few bullets.

- Robust function for hardware, software, sensors and actuators.
- Tactile improvement on the vehicle's behaviour with the ESC system in function.
- There is potential application of the system into a real vehicle.
- The ESC algorithm was independent to road bank angle, bumps and engine vibration.
- Low cost implemented.
- Easily adjusted to every vehicle and to different driving styles.

Real Environment Experiments

Figures Fig. 5.1 and Fig. 5.2 are characteristic plots of yaw rate, individual wheel speed and steering angle with the single gyro ESP algorithm from auto generated experiments.

In order to have the opportunity to compare the behaviour of the car in a valid manner, with ESP system on or off and/or while adjusting some parameters we should program the testbed to perform the same experiment multiple times. Thereafter, we have programmed the vehicle to steer 20 degrees left when it reached the velocity of 3m/s, for both instances presented at Fig. 5.1 and Fig. 5.2.

Fig. 5.1 is a plot with the ESP system technically off, where we have set the sensitivity to intervene very late ($S = 0.2$, Fig. 3.4). At this instance, the vehicle totally loses control and experiences more than 200deg/sec yaw rate.

Fig. 5.2 shows relatively the same as the previous auto generated experiment, but with ESP fully active, where we have set the sensitivity equal to that of the ESC algorithm we have adopted (Fig. 3.4). If we look at the upper subplot of the Fig. 5.2, the green circles projected on the yaw rate assert that the vehicle detects oversteer. In the lowest subplot the up arrows (\uparrow) on individual wheel velocity assert the command from the ESP system to brake the specific wheel.

The vehicle on the same tarmac at the same turning velocity and the almost the same acceleration, experiences half yaw rate without the ESP system. When it first detects oversteer starts to brake the appropriate front wheel, producing a large amount of understeer which is also detected from the system (upper subplot inside black oval) and acts by braking the correct rear wheel, producing oversteer (upper and lower subplot inside red oval) and so on.

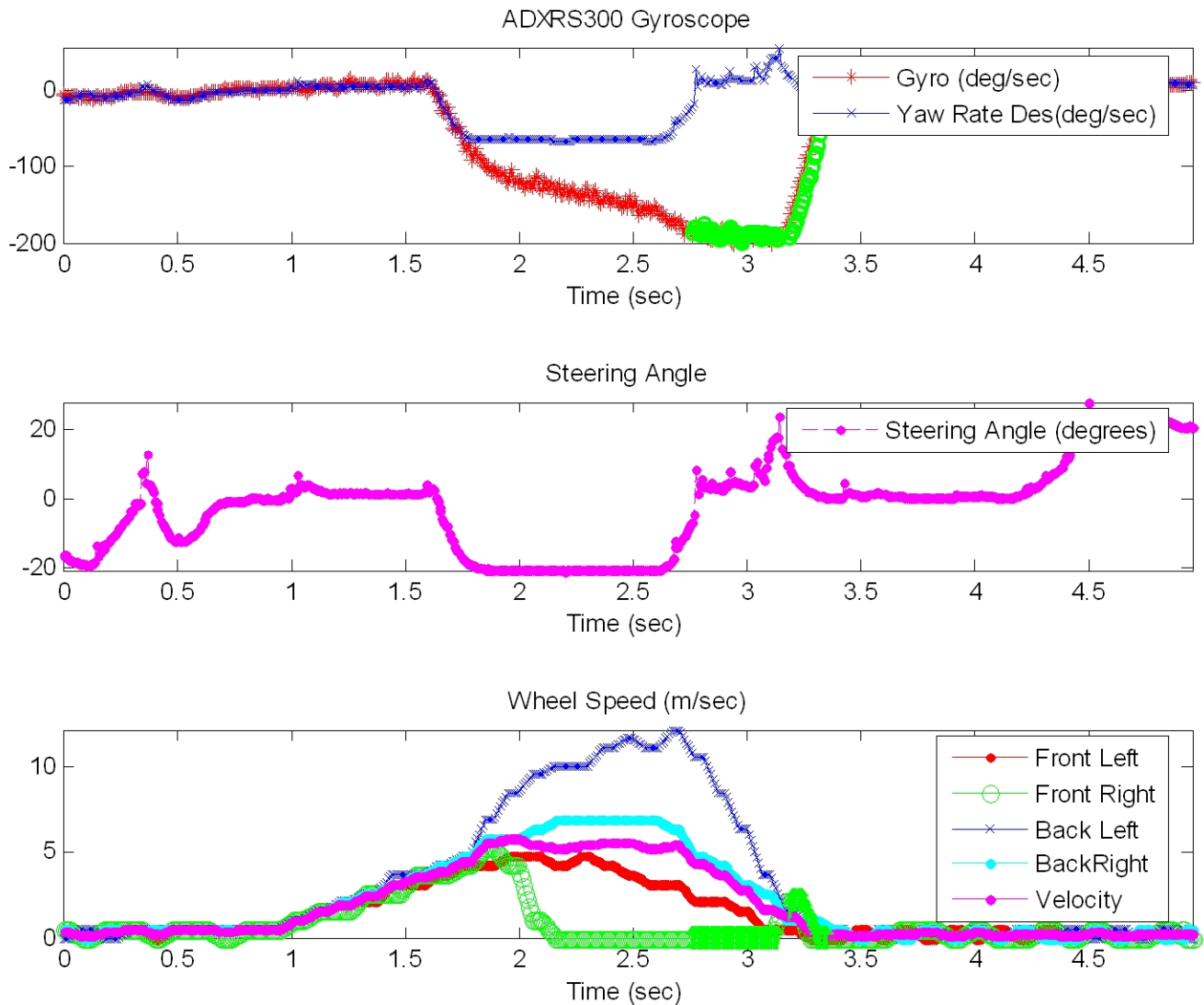


Fig. 5.1 Yaw rate, individual wheel speed and steering angle for the auto generated experiment with the ESP sensitivity set at 0.2 \rightarrow technically Off.

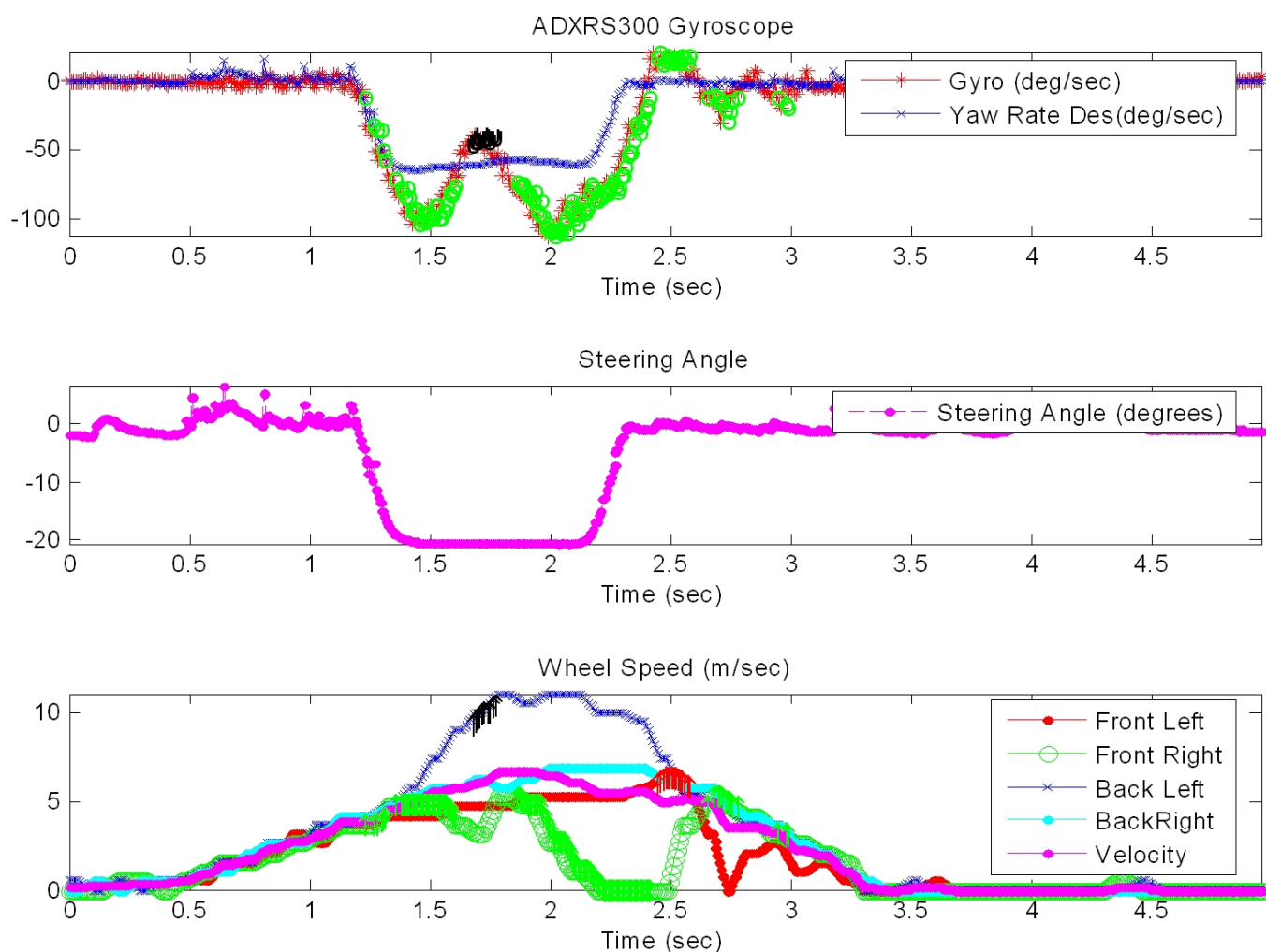


Fig. 5.2 Yaw rate, individual wheel speed and steering angle for the auto generated experiment with the ESP sensitivity set at 0.9 and the understeer gradient 0.004 (Fig. 3.4).

The improved performance comparing the previous two figures is obvious. The car retains a much higher velocity with regards the experiment with the ESP off. The greatest improvement can be observed in a slalom situation. With the ESP off, the vehicle totally loses control, because of its oversteering behaviour, even in very low velocity, whilst with the ESP in function the vehicle behaves much more stable. Unfortunately, it is not much useful to program an auto generated sequence of events for conducting slalom. Due to the extreme oversteering behaviour, the vehicle without the ESC system (in functional mode), isn't able to complete more than a single stiff turn without spinning out. To conduct more than one stiff turn in manual driving, the driver must always try to counteract the oversteering, by turning the steering wheel to the other side. Thus it is difficult to "build" a slalom situation and compare both conditions (with the ESC on and off) in a valid manner. The video accompanies this thesis is revealing for the improved behaviour of the vehicle with the ESC.

Ideally, for the illustration of the improved kinematics behaviour of the vehicle at exactly the same experiments as those cited previously, is the collation of the trajectories that the vehicle

follows for both experiments, in a bird-eye-view plot of traces. Of course, estimating the real trajectory is not an easy task and requires a sequence of pictures taken from a camera that would capture the kinesis of the vehicle and under vision techniques would reveal the path. Another possible way to do the same estimation is the usage of a GPS system onboard of the vehicle, but the error is also very high in that situation. Consequently, in order we have a bird-eye-view plot of traces, we use the collected data. Through the utilization of those data and simplified equations of motion we can derive to relatively realistic trajectory of the vehicle.

Bird-Eye-View

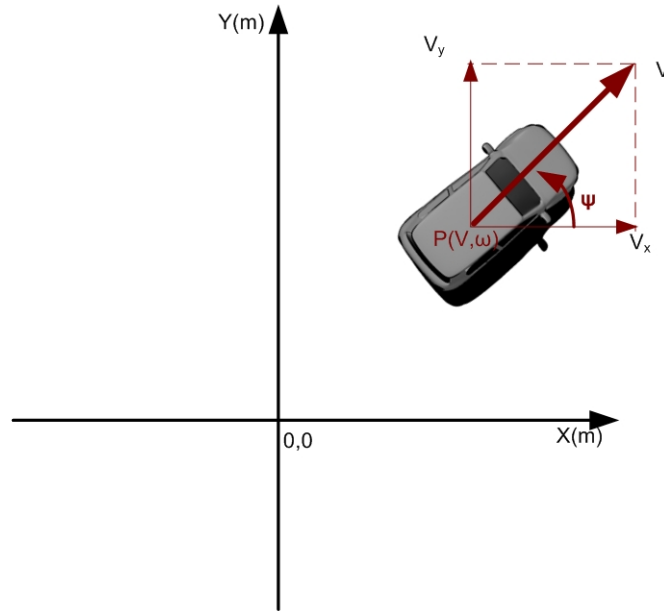


Fig. 5.3 Bird's eye view.

Suppose having the vehicle on a Cartesian fixed plane with coordinates (x, y). The position P of the vehicle (Fig. 5.3) is a function of the yaw rate (ω) of the vehicle and the velocity V, which are both functions of time. Suppose the vehicle starts at time = 0 sec at $P_0 = [0, 0]$ with an initial velocity V_0 and an initial yaw rate ω_0 . The position P of the vehicle at time equals "t" will be:

$$P(\vec{V}, \omega) = [x(t), y(t)]$$

and

$$P_0(\vec{V}_0, \omega_0) = [x(0), y(0)] = [0, 0] \quad (5.1)$$

Where:

$$\left. \begin{aligned} x(t) &= \int_0^t V_x(t) dt \\ V_x(t) &= V(t) \cdot \cos(\psi(t)) \end{aligned} \right\} x(t) = \int_0^t V(t) \cdot \cos(\psi(t)) dt$$

$$\left. \begin{aligned} y(t) &= \int_0^t V_y(t) dt \\ V_y(t) &= V(t) \cdot \sin(\psi(t)) \end{aligned} \right\} y(t) = \int_0^t V(t) \cdot \sin(\psi(t)) dt \quad (5.2)$$

And

$$\psi(t) = \int_0^t \omega(t) dt \quad (5.3)$$

If we consider that the slip ratio of each wheel is low, the magnitude of the velocity “V” of the vehicle at Fig. 5.3, can be estimated through with the following relationship (5.4):

$$|V(t)| = \frac{(V_{fl}(t) + V_{fr}(t)) \cdot \cos \delta + (V_{rl}(t) + V_{rr}(t))}{4}$$

V_{fl} : Longitudinal for front left wheel
 V_{fr} : Longitudinal for front right wheel
 V_{rl} : Longitudinal for rear left wheel
 V_{rr} : Longitudinal for rear right wheel

(5.4)

Combining the four previous relationships (5.1 - 5.4) and using the data collected from the vehicle, we can illustrate an approximation of the trajectory (Fig. 5.5 and Fig. 5.6) of the vehicle in the plane for the same auto – generated experiments presented above. The red arrow at the Fig. 5.5 and Fig. 5.6 shows the heading angle of the vehicle. The command for a wheel to brake is also illustrated on the figure, left and right of the trajectory of the vehicle. The time (character indicating time) is plotted with a time interval of 160 msec. At Fig. 5.7, we can see both figures Fig. 5.5 and Fig. 5.6 collated into the same plot.



Fig. 5.4 Vehicle's Velocity considering zero slip angles.

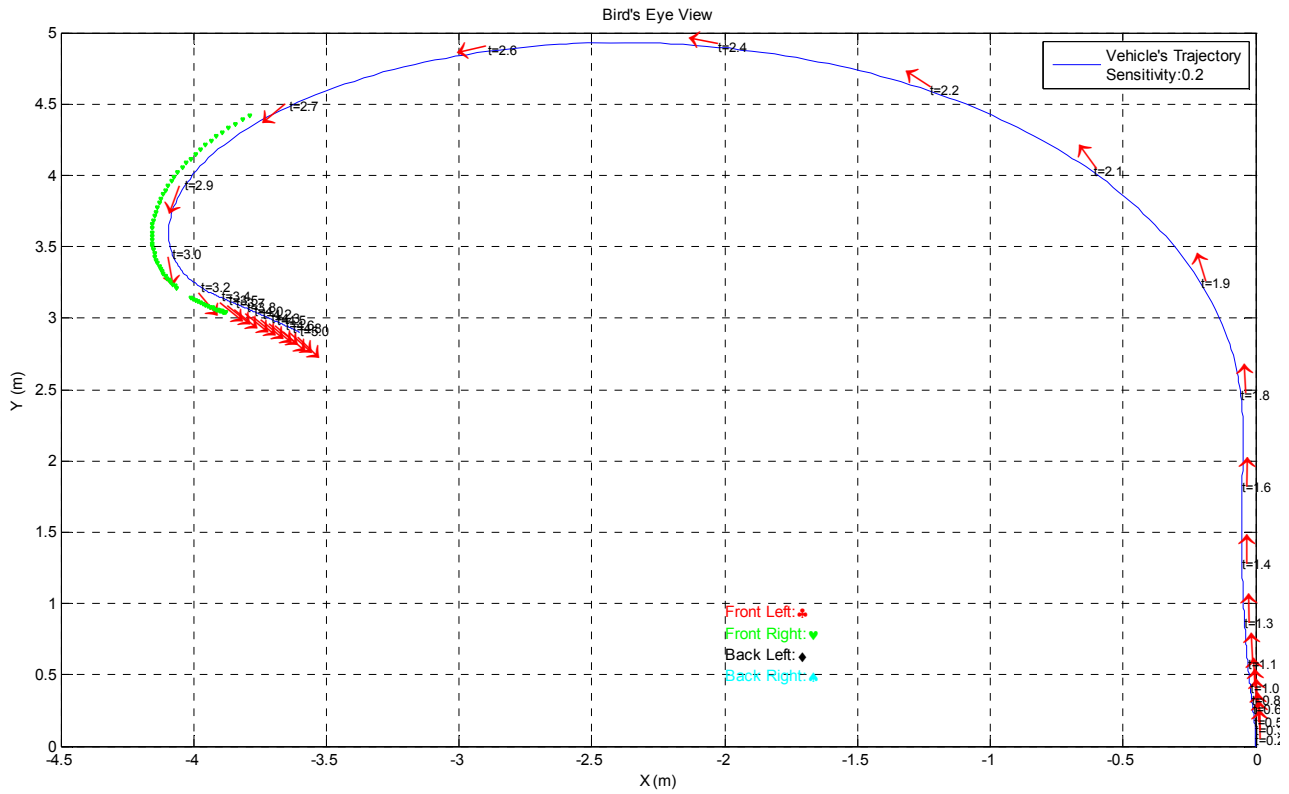


Fig. 5.5 Vehicle's bird-eye-view plot of traces from for the auto generated experiment shown at Fig. 5.1 with the ESP sensitivity set at 0.2 \rightarrow technically Off.

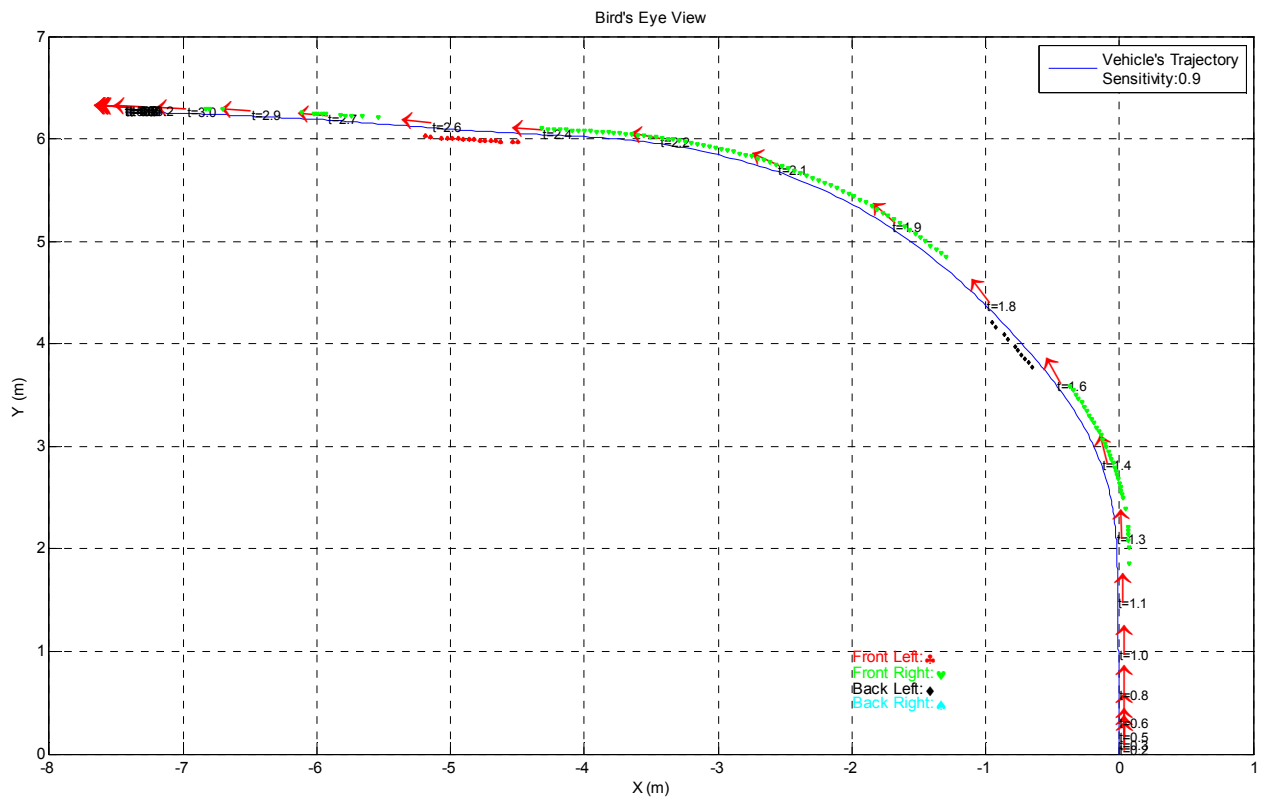


Fig. 5.6 Vehicle's bird-eye-view plot of traces from for the auto generated experiment shown at Fig. 5.2 with the ESP sensitivity set at 0.9 and the understeer gradient 0.004 (Fig. 3.4).

It is obvious, that the above presented method isn't impeccable. The flaws are due to the measurement errors. By integrating the values from the gyroscope, for estimating the direction of vector of the velocity, we accumulate the measurement errors from the gyroscope. The same is also true for the integration of the velocity over time, for the determination of the position in the plane. The most undermining factor of this method is the inner (of the turn) rear wheel which starts spinning when the vehicle oversteers. Thereafter, by the time the vehicle starts to oversteer, the real velocity of the rear inner wheel is less than the measured. This fact can be compensated (in case the ESC is active) by the fact that the longitudinal velocity of the wheel which is commanded to brake is higher than the measured, because of the moment of the vehicle and the high slip the wheel is experiencing.

The aforementioned problem for estimating a realistic trajectory is more intense in the case the ESC is off and the vehicle's behaviour is more unpredictable. The more intense the oversteer the higher the spinning, which is translated as a corroded measurement of velocity, thus as displacement at the real final position of the vehicle. Artlessly, if an experiment lasts little time, the errors accumulated are small, thus shorter experiments are desired.

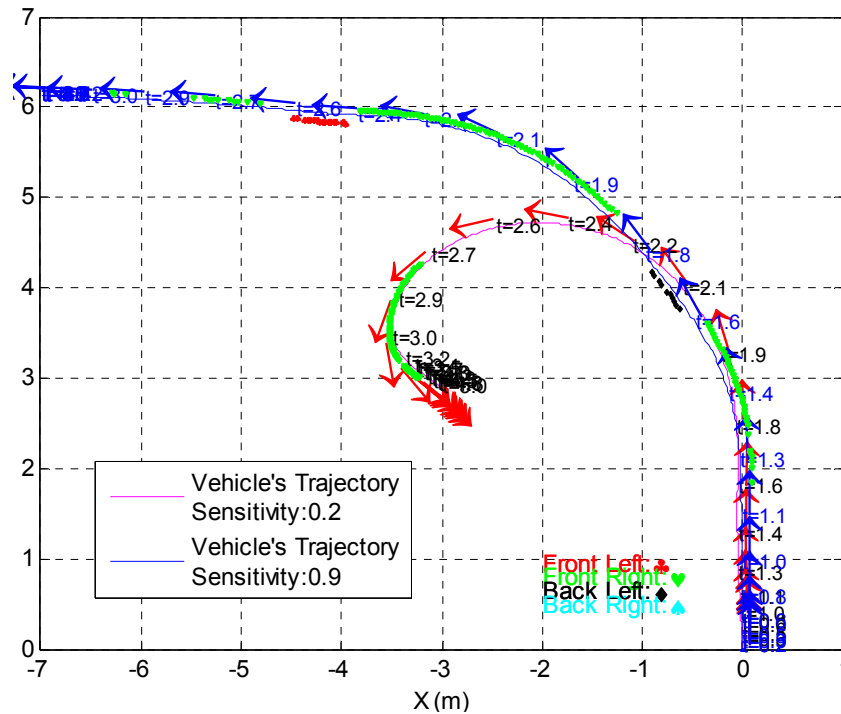


Fig. 5.7 Vehicle's bird-eye-view plot of traces from for the auto-generated experiment shown at shown at Fig. 5.2, for both inactive and active ESC tests with the ESC sensitivity correspondingly set to 0.2 (technically Off) for (magenta line trajectory) and ESC sensitivity set to 0.9 (blue line trajectory, ESC active).

6. Conclusions and Future Work

6.1 Conclusions

A scaled test bed was presented along with the evaluation of an electronic stability control scheme. The scaled testbed was built upon open source software and proved to be economic and robust in operation. It is highly recommended for the future researcher who plans to develop a similar testbed, to use a better base for the testbed (a better model car of higher quality) and a smaller form factor SBC. Enough time was spent in the upgrading of the model for becoming more credible and capable of hosting individual wheel brakes. A model with all the mechanical parts preinstalled from the factory, is recommended since the cost of buying a better model is lesser than the cost of upgrading a poor quality model. Also a smaller form factor SBC, like a PC-104 can have similar computing power as the one we used, but would dissipate less power, which means lighter batteries for the computer. A ready to use embedded system with preinstalled Linux that work out of the box, would be a nice solution too.

Lithium batteries are also proposed because their cost has dropped dramatically since the beginning of the construction of the scaled testbed (December 2006), are lighter and have similar characteristics in providing current. A modern lithium battery used for radio controlled models can provide up to 100 amperes (or more) of current at 14.8 Volts.

An important issue of the implementation was the single cylinder two stroke engine. It proved to be a real burden. In the market, there are 1:5 scaled radio controlled model cars that operate on battery and are as fast as those running on fuels. In case we redesigned the system, we would buy one of those. They are slightly more expensive, since they usually host brushless motors and brushless motor controllers, and high capacity lithium batteries which are all expensive equipment compared to an internal combustion engine and petrol respectively. However, the vibration from the engine would obliterate, we wouldn't have to mix petrol and oil for running the engine and it would not need any effort either for running the engine or for maintenance.

In terms of ESC; the applicability of the proposed ESC system was clear and useful results came up from the trial runs. First of all the use of gyroscope is one – way solution for an ESC system on a scaled test bed. All three accelerometers embedded in the scaled car, reported similar to ADXL213 sensitivity at measurements. There are several factors that can undermine the credibility of the samples. Vibration from the engine, shock from bumps on the road, inclination of the ground etc. The effects from the precedent would be reduced in a full scale vehicle, compared to a scaled car, but they wouldn't obliterate. The collected experimental data provide also the opportunity for a better understanding of understeering and oversteering effects.

Thereupon, we might be able to derive to more robust ESC schemes and heuristic algorithms. Through examination of velocity, steering angle and yaw rate we were able to determine the maximum yaw rate a vehicle can experience before loses handling. Thus, it was possible to construct off – line an appropriate reference function for yaw rate and discern when the vehicle experiences an undesired effect.

If we prove an explicit connection between the test bed and a real vehicle, the results and algorithms that would derive from the scaled car would be applicable on a real vehicle too. Therefore, the use of scaled test beds in automotive engineering could accelerate prototyping process and diminish the cost of development.

Using the testbed, we had the opportunity to run tests for different stabilization algorithms ([58]) that hadn't been proposed earlier, irrespective with the fact that those algorithms were not prosperous. After all, this is the reason that a scaled testbed is built for: testing in the real environment, easily and with low cost, new dynamic stabilization schemes and generally running experiments that can help us understand the application of theory.

One key outcome from this thesis was that the ESP should be adopted by every vehicle since it can improve dramatically even the behaviour of a scaled car where:

- the measurements errors are reversely proportional to the scale
- the reaction time is proportional to the scale
- and the mechanical actuators used are much more primitive than the conventional hydraulic brake system.

The system is a pure fun to experiment and implement. Scaled test beds could prove useful to automotive manufacturers and researchers.

6.2 Future work and potentials extensions

What would be the best sequel for the project?

- Develop, test and evaluate more sophisticated algorithms.
- Control the vehicle without the use of gyroscope, through a trained artificial neural network.
- Prove the behavioural and dynamical connection between the scaled model and a real vehicle, thus proving the utility of the model
- Add faster and more powerful actuators (higher torque) on the model.
- Manifest the connection between experiments in the test bed and a real vehicle.
- Use car dynamics based on double track model and Investigate vehicle's behaviour under high speed driving techniques [10], [11].
- Implementation of similar sensory provision on a real vehicle is desired.
- Use case specific computing hardware components, such as FPGAs, that can run high complexity and time consuming algorithms in real time.

7. Acknowledgements

The author is grateful to his advisors, Dr. Yannis Papaefstathiou, Dr. Michail Lagoudakis and Dr. Apostolos Dollas, from TUC for their excellent cooperation and supervision and most of all for their and support to obtain this Master degree. He would also like to thank Dr. Dionysios Pneumatikatos from TUC for his invaluable help to solve bothersome aspects during the development of this thesis and his brother Antonis Katzourakis from the Foundation of Research and Technology Hellas for the great 3D drawings he made for the completion of this thesis. The author is also thankful towards Professor Petros Ioannou from the University of Southern California who gave him academic access to one of the best institutes of the World, Professor Kostas Kalaitzakis from TUC for the donation of the accelerometers, graduate student Mr. Spiros Ninos at TUC for his tutorial help with Linux systems and administration and the undergraduate student Mr. Bill Hatzidiakos from TUC for his cooperation. The project was sponsored mainly by technical University of Crete and has received donations from Silicon Sensing Systems through Mr. Mike Barnes to whom the author is obliged.

8. References

- [1] European New Car Assessment Programme (EuroNCAP) and European Commission, "Choose ESC," 2007.
- [2] National Highway Traffic System Administrator, "Electronic Stability Control System," *FMVSS no.126*, March 2007.
- [3] Paul Yih, "Radio Control Car Model as Vehicle Dynamics Test Bed," Dynamic Design Lab, Stanford University, September 2000.
- [4] J. Ackermann, "Robust Car Steering by Yaw Rate Control," Conference Proceedings of the, *29th Conference on Decision and Control*, Honolulu, Hawaii, vol. 4, pp. 2033–2034, Dec. 1993.
- [5] J. Ackermann, T. Bunte, D. Odenthal, "Advantage of Active Steering for Vehicle Dynamics Control," German Aerospace Research Establishment, Institute for Robotics and System Dynamics, Oberpfaffenhofen, 1999.
- [6] J. Ackermann, T. Bunte "Automatic Car Steering Control Bridges Over the Driver Reaction Time," German Aerospace Research Establishment, Institute for Robotics and System Dynamics, Oberpfaffenhofen, 24 May 1995.
- [7] J. Tjønnas, T. A. Johansen, "Adaptive Optimizing Dynamic Control Allocation Algorithm for Yaw Stabilization of an Automotive Vehicle using Brakes," *Med's '06. 14th Mediterranean Conference on Control and Automation*, pp. 1 – 6, June 2006.
- [8] H.E.Tseng, B. Ashrafi, D. Madau, T. Allen Brown, D. Recker, "The development of Vehicle Stability at Ford," *IEEE/ASME Transactions on Mechatronics*, Vol. 4, pp. 223 – 234, Issue 3, Sept 1999.
- [9] B.A. Guvenc, T. Acarman, L.Guvenc, "Coordination of Steering and Individual Wheel Braking Actuated Vehicle Yaw Stability Control," *Intelligent Vehicles Symposium*, IEEE Proceedings, pp. 288 – 293, June 2003.
- [10] Velenis, E., Tsiotras, P., and Lu, J., "Aggressive Maneuvers on Loose Surfaces: Data Analysis and Input Parameterization," *15th IEEE Mediterranean Control Conference*, CD Proceedings, June 26-29, Athens, Greece.
- [11] Velenis, E., Tsiotras, P., and Lu, J., "Modelling Aggressive Maneuvers on Loose Surfaces: The Cases of Trail-Braking and Pendulum-Turn," *European Control Conference*, CD Proceedings, Kos, Greece, July 2-5, 2007.
- [12] T. Van Zanten *et al*, VDC, *the Vehicle Dynamics Control System of Bosch*, SAE 950759, 1995.

- [13] S. Brennan, A. Alleyne, "Using a Scale Testbed: Controller Design and Evaluation," *IEEE Control Systems Magazine*, pp. 15-26, June 2001.
- [14] T.D. Gillespie, *Fundamental of Vehicle Dynamics*, SAE, 1992.
- [15] WinAVR, Available: <http://winavr.sourceforge.net/>
- [16] G. A. Rovithakis, M. A. Christodoulou, *Adaptive Control With Recurrent High-order Neural Networks: Theory and Industrial Applications*, Springer 2000.
- [17] Analog Devices, ADXL311 accelerometer data sheet, "Low Cost, UltraCompact $\pm 2g$ Dual Axis Accelerometer," Analog Inc. 2003.
- [18] Walt Boyes, *Instrumentation Reference Book*, Online Book, Available: www.google.com , p. 132.
- [19] Analog Devices, ADXL213 accelerometer data sheet, "Low Cost, $\pm 1.2g$ Dual Axis Accelerometer," Analog Inc. 2004.
- [20] ATMEL Corporation, ATmega32 data sheet, "8-bit AVR Microcontroller with 32K Bytes In-System Programmable Flash", Atmel Corporation, 2006.
- [21] T. Ina, K. Takeda, Nippon Soken, Inc. A. Sawada, S. Fukaya, Denso Corp, "Micro – Rotational Angle Sensor with Integrated Hall IC," *Advanced Microsystems for Automotive Applications 2008*, pp. 229-237, Springer 2008.
- [22] T. Ina, K. Takeda, T. Nakamura, O. Shimomura, T. Ban and T. Kawashima, "360-degree Rotation Angle Sensor Consisting of MRE Sensors with a Membrane Coil," *Advanced Microsystems for Automotive Applications 2005*, pp. 447-457, Springer, 2005.
- [23] G. Rieger, K. Ludwig, J. Hauch and W. Clemens, Siemens AG, Corporate Technology, "GMR sensors for contact less position detection," *Sensors and Actuators A: Physical*, Volume 91, Issues 1-2, pp. 7-11, 5 June 2001.
- [24] Honeywell Inc., SS400 series data sheet, "Digital Position Solid State Sensors SS400 series".
- [25] Honeywell Inc., *Hall Effect Sensing and Applications*, Electronic Book, Available: <http://content.honeywell.com/sensing/prodinfo/solidstate/technical/hallbook.pdf>
- [26] Analog Devices, ADXRS300 data sheet, " $\pm 300^\circ/s$ Single Chip Yaw Rate Gyro with Signal Conditioning," Analog Inc., 2004.
- [27] SparkFun Electronics, "ADXRS300: Gyro Breakout Board v1.1", July 2005, Available: <http://www.sparkfun.com/datasheets/Sensors/ADXRS-Breakout.pdf>
- [28] V. Apostolyuk "Theory and design of micromechanical vibratory gyroscopes," MEMS/NEMS Handbook, Springer, 2006, Vol.1, pp.173-195.
- [29] SparkFun Electronics, Available: <http://www.sparkfun.com/commerce/categories.php>

- [30] Paul Sandin , *Robot Mechanisms and Mechanical Devices Illustrated*, McGraw-Hill/TAB Electronics; 1 edition, June 2003, pp. 20-23.
- [31] Kevin O'Dea: Delphi Corporation, "Anti-Lock Braking Performance and Hydraulic Brake Pressure Estimation", *SAE Technical Paper Series*, 2005-01-1061.
- [32] Th. Zamachoglou, G. Kapetanakis, P. Karampolas, G. Patsiavos, *Automotive Technology, Beyond 2000*, IDEA 2000.
- [33] Fairchild Semiconductor Inc., Data Sheet for: LM7805, LM7806, LM7808, LM7809, LM7810, LM7812, LM7815, LM7818, LM7824, LM7805A, LM7806A, LM7808A, LM7809A, LM7810A, LM7812A, LM7815A, LM7818A, LM7824A, "3 – Terminal 1A Positive Voltage Regulator", Data Sheet, Revised December 2005.
- [34] National Semiconductor Inc, LM117/LM317A/LM317 data sheet, "3-Terminal Adjustable Regulator", April 2007.
- [35] ATMEL, "AVR STK500 User Guide," Atmel Corporation, 03/2003, Available: http://www.atmel.com/dyn/Products/tools_card.asp?tool_id=2735, http://www.atmel.com/dyn/resources/prod_documents/doc1925.pdf
- [36] ATMEL, "AVR Studio: Integrated Development Environment," Atmel Corporation, 10/2007, Available: http://www.atmel.com/dyn/products/tools_card.asp?tool_id=2725, http://www.atmel.com/dyn/resources/prod_documents/doc2510.pdf
- [37] Harvey Weinberg: Analog Devices, "Using Absolute Output iMEMS® Gyroscopes with Ratiometric ADCs", Application Note.
- [38] Craig Peacock, "Interfacing the Serial / RS232 Port V5.0," Craig Peacock. 1998, Available: <http://www.beyondlogic.org/serial/serial.pdf>
- [39] Texas Instruments, MAX232 data sheet, "MAX232, MAX232L: Dual EIA – 232 Drivers/Receivers," Revised March 2004.
- [40] ATMEL, "AVR034: Mixing C and Assembly Code with IAR Embedded Workbench for AVR", ATMEL Corporation 2003.
- [41] ATMEL, "AVR JTAGICE mkII: Debug AVR Applications using system JTAG or Debugwire interface", ATMEL Corporation 2004. Available: http://www.atmel.com/dyn/resources/prod_documents/doc2489.pdf
- [42] OJ Svendsli, "Designing for Efficient Production with In-System Re-programmable Flash μ Cs", ATMEL, Available: http://www.atmel.com/dyn/resources/prod_documents/issue4_pg16_17_DesignC.pdf
- [43] Ubuntu Linux, Available: <http://www.ubuntu.com/>
- [44] Ubuntu Linux, What is Ubuntu, Available: <http://www.ubuntu.com/products/whatisubuntu>

- [45] Ubuntu Linux, Licensing, Available:
<http://www.ubuntu.com/community/ubuntustory/licensing>
- [46] Ubuntu Linux, Ubuntu Philosophy, Available:
<http://www.ubuntu.com/community/ubuntustory/philosophy>
- [47] Free Software Foundation, Available: <http://www.fsf.org/>
- [48] GNU Operating System, The free software definition, Available:
<http://www.gnu.org/philosophy/free-sw.html>
- [49] Open Source Initiative, Open Source definition, Available:
<http://www.opensource.org/docs/definition.php>
- [50] Simon Gray, How to Install Anything on Ubuntu, Available:
<http://monkeyblog.org/ubuntu/installing/>
- [51] Ubuntu forums, How to disable IPV6 to speed – up internet, Available:
<http://ubuntuforums.org/showthread.php?t=87798>
- [52] Ubuntu, Package: webcam-server (0.50-2), Available:
<http://packages.ubuntu.com/feisty/net/webcam-server>
- [53] Putty, Free SSH Client, Available:
<http://www.chiark.greenend.org.uk/~sgtatham/putty/>
- [54] David S.Lawyer, Serial HOWTO, The Linux Documentation Project, January 2007, Available: <http://tldp.org/HOWTO/Serial-HOWTO.html>
- [55] Michael R Sweet, Serial Programming Guide for POSIX Operating Systems 5th Edition, 6th Revision Copyright 1994 – 2005 by Michael R. Sweet, Available:
<http://www.easysw.com/~mike/serial/serial.html>
- [56] Available: <http://www.opengroup.org/>
- [57] Ralf Habacker, Mailing list of Cygwin Project, “gettimeofday() does not returns usec resolution”, Available: <http://cygwin.com/ml/cygwin/2002-01/msg01408.html>
- [58] Diomidis I. Katzourakis, Antonis I.Katzourakis, “Scaled Testbed for Automotive Experiments,” AMAA 2008, Berlin 2008, pp. 239-257.
- [59] Jong Hyeon Park and Woo Sung Ahn, “H_∞ Yaw-Moment Control with Brakes for Improving Driving Performance and Stability,” *Proceedings of the 1999 IEEE/ASME International Conference on Advanced Intelligent Mechatronics September 19-23*, pp. 747-752, Atlanta, USA, 1999.
- [60] Sohail Anwar, “Yaw Stability Control of an Automotive Vehicle via Generalized Predictive Algorithm,” *Proceedings of the American Control Conference 2005*, pp. 435-440, Portland, OR, USA, June 8-10, 2005.

- [61] P. Tøndel, T. A. Johansen, "Control Allocation for Yaw Stabilization in Automotive Vehicles using Multiparametric Nonlinear Programming," Hamilton Institute, Ireland.
- [62] Rajesh Rajamani, *Vehicle Dynamics and Control*, Springer 2006.
- [63] Petros A. Ioannou, Barış Fidan, *Adaptive Control Tutorial*, SIAM 2005.
- [64] H. Dugoff, P. S. Francher, and L. Segel, "An analysis of tire traction properties and their influence on the vehicle dynamic performance," SAE 700377, 1970.
- [65] E. K. Lieberman, K. Meder, J. Schuh, G. Nenninger, Robert Bosch GmbH, "Safety and Performance Enhancement: The Bosch Electronic Stability Control (ESP)", SAE, Paper Number 05-0471, 2004.
- [66] Wikipedia, The free Encyclopaedia, "The air – bag", Available: <http://en.wikipedia.org/wiki/Airbag>
- [67] Wikipedia, Electronic Stability Control, Available: http://en.wikipedia.org/wiki/Electronic_Stability_Control
- [68] Robert Bosch GmbH, Germany, Available: www.bosch.com
- [69] TRW, USA, Available: <http://www.trw.com>
- [70] Continental Automotive Systems, USA, Available: http://www.conti-online.com/generator/www/start/com/en/index_en.html
- [71] Delphi, USA, Available: <http://delphi.com/>
- [72] Aisin Advics, Japan, Available: <http://www.aisin.com/>
- [73] Nissin Kogyo, Japan, Available: <http://www.nissinkogyo.co.jp/>
- [74] Hitachi, Japan, Available: <http://www.hitachi.com/>
- [75] Mando Corporation, Korea, Available: <http://www.mando.com/>
- [76] Bendix Corporation, USA, Available: <http://www.bendix.com>
- [77] WABCO, USA, Available: <http://www.wabco-auto.com>
- [78] BOSCH Live, "Bosch celebrates major milestones," *The magazine*, September 2007 Available: http://www.bosch.us/content/language1/html/715_5859.htm
- [79] Jong Hyeon Park and Chan Young Kim, "Wheel Slip Control in Traction Control System for Vehicle Stability," *Vehicle System Dynamics*, pp. 263–278, 1999.
- [80] P. Koleszar, B. Trenčsni and L. Palkovics, "Development of an Electronic Stability Program Completed with Steering Intervention for Heavy Duty Vehicles," *Proceedings of the IEEE International Symposium on Industrial Electronics*, ISIE 2005. Vol. 1, pp. 379 – 384, June 20-23, 2005.
- [81] Leo Laine and Johan Andreasson, "Control Allocation based Electronic Stability Control System for a Conventional Road Vehicle," *Proceedings of the 2007 IEEE Intelligent Transportation Systems Conference Seattle*, pp. 514-521, WA, USA, Sept. 30 - Oct. 3, 2007.

- [82] Kazuya Kitajima, Huei Peng, "Control For Integrated Side Slip, Roll and Yaw Controls for Ground Vehicles," *Proceedings of AVEC 2000, 5th Symposium on Advanced Vehicle Control*, August 22 – 24, 2000, Ann Arbor, Michigan.
- [83] Dan Simon, "From here to Infinity," *Embedded Systems Programming*, October 2001, Available: <http://academic.csuohio.edu/simond/courses/eec641/hinfinity.pdf>
- [84] Jihua Huang, Jasim Ahmed, Aleksandar Kojic, Jean-Pierre Hathout, "Control Oriented Modeling for Enhanced Yaw Stability and Vehicle Steerability," *Proceeding of the 2004 American Control Conference Boston*, Vol. 4, pp. 3405 – 3410, Massachusetts June 30 -July 2, 2004.
- [85] Randy Whitehead, Ben Clark, Matt Breland, Kenny Lambert, David M. Bevly, George Flowers, "Scaled Vehicle Electronic Stability Control," *ESV International Collegiate Student Safety Technology Design Competition*, North American Regional Review, March 25, 2005.
- [86] Sean Brennan and Andrew Alleyne, "The Illinois Roadway Simulator: A Mechatronic Testbed for Vehicle Dynamics and Control," *IEEE/ASME Transaction On Mechatronics*, Vol. 5, pp. 349-359, No. 4, December 2000.
- [87] William E. Travis, Randy J. Whitehead, David M. Bevly, and George T. Flowers, "Using Scaled Vehicles to Investigate the Influence of Various Properties on Rollover Propensity," *Proceedings of the 2004 American Control Conference*, pp. 3381 – 3386, Boston, Massachusetts, June 30 -July 2004.
- [88] Sean N. Brennan, MSC Thesis, *Modelling and control issues associated with scaled vehicles*, Submitted in partial fulfilment of the requirements for the degree of Master of Science in Mechanical Engineering in the Graduate College of the University of Illinois at Urbana – Champaign, 1999.
- [89] DePoorter Mark. MSC Thesis, *Development, Experimentation and Control of Small Scale Vehicle Dynamics and Control Laboratory*. M.S. Mechanical and Industrial Engineering, University of Illinois at Urbana – Champaign, Urbana, 1997a.
- [90] DePoorter, M., S. Brennan and A. Alleyne, "Driver Assisted Control Strategies: Theory and Experiment," Paper read at 1998 *ASME International Mechanical Engineering Congress and Exposition*, pp. 721-725, at Anaheim, CA, Nov. 1998.
- [91] Sean Brennan Andrew Alleyne, "Driver Assisted Yaw Rate Control," *Proceedings of the 1999 American Control Conference*, Vol. 3, 2-4 June 1999, pp. 1697 – 1701.
- [92] Brennan, S., DePoorter, M., & Alleyne, A. "The Illinois Roadway Simulator - A Hardware-in-the-Loop Testbed for Vehicle Dynamics and Control," *Proceedings of the 1998 American Control Conference*, Philadelphia, PA, pp. 493-497, June 1998.
- [93] R. Holve*, P. Protzel, "Reverse Parking of a Model Car with Fuzzy Control", *Proceedings of the 4th European Congress on Intelligent Techniques and Soft Computing - EUFIT '96*, pp. 2171-

2175, Aachen, Germany, Sept. 1996.

[94] Manuel C.G. Silva, Mário L.O.S. Mateus & João M.S. Cruz, "Teaching kinematics and mathematics with a radio-controlled scale car", *World Transactions on Engineering and Technology Education* 2003, Vol.2, pp. 87-90, *UICEE No.1*, 2003.

[95] Mark A. Morton, *Traction Control Study for a Scaled Automated Robotic Car*, Thesis submitted to the Faculty of the Virginia Polytechnic Institute and State University In partial fulfillment of the requirements for the degree of Master of Science in Electrical Engineering, 2004.

[96] Richard D. Henry, *Automatic Ultrasonic Headway Control for a Scaled Robotic Car*, Thesis submitted to the Faculty of the Virginia Polytechnic Institute and State University In partial fulfillment of the requirements for the degree of Master of Science in Computer Engineering, Virginia Polytechnic Institute & State University, 2001.

[97] Seung kook Jun, Daniel O. Gott, "The Smart Car Project: Development and implementation of a modular scaled testbed," *Proceedings of DETC' 03 ASME2003 Design Engineering Technical Conferences and Computer and Information in Engineering Conference*, Chicago, Illinois, September 2 – 6, 2003.

Author: Diomidis I. Katzourakis (I.= Ioannis)

Address: Vernadaki 16B, Chania, Crete, Greece

E-mail: diomidis@systems.tuc.gr , diomkatz@gmail.com

Keywords: scaled test bed, model car, electronic stability control, sensors, gyroscope, ADXRS300, accelerometer, ADXL213, ADXL311, experiment, vibration, radio control, single board computer, microcontroller, AVR, ATmega32, remote access, hall effect sensor, SS443, single track model, bicycle, oversteer, understeer, yaw rate

This page was intentionally left blank.

This page was intentionally left blank.

9. Appendix

9.1 Host code

9.1.1 daemon

async.cpp

```
//OLES OI ALLAGES POU EXW KANEI EINAI ANAMESA SE SXOLIA TOU TYPOU
//ALLAGH APO BILL
//TELOS ALLAGHS

#include "files/headers.h"
#include <iostream>
using namespace std;

//ALLAGH APO BILL
// kanw include ena header mou gia to struct twn parametrwn (gia leptomereies des sto sygekrimeno header)
#include "context.h"
//TELOS ALLAGHS

#define logfile "log.txt"
#define logmath "logmath.txt"

void signal_handler_IO (int status);                                     /* definition of signal handler */
int wait_flag=FALSE;                                                    /* TRUE while no signal received */
char * return_date(void);

int * return_time(void);    //17_2_08 Dio

#include "files/time_etc.cpp"
#include "files/init_serial.c"                                           //serial port intialization, signal
handler                                                                   
                                     //to include ginetai meta
                                     //thn dhlwsh twn global metablhtwn

#define Acceleration_12(T1,T2) -(double)((((double)((double)T1/(double)T2)-0.5)/0.3)
#define Vref    5;
//#define char_to_int(input) if(input<0) input=255-abs(input) else if(input==0) input=255 else if(input==NULL)
input=0 // Metatrepei to 8 bit char se integer

#define char_to_int(input) if(input<0) input=256-abs(input)
//#define forgetting_sum(last,previous)
(double)(last*0.4+previous[0]*0.25+previous[1]*0.15+previous[2]*0.1+previous[3]*0.1)

//#define forgetting_sum(last,previous) (double)(last*0.8+previous[0]*0.1+previous[1]*0.05+previous[2]*0.05)
18_2_08
#define forgetting_sum(last,previous) (double)(last*0.4+previous[0]*0.3+previous[1]*0.15+previous[2]*0.15)

#define forgetting_sum_steer(last,previous) (double)(last*0.3+previous[0]*0.25+previous[1]*0.25+previous[2]*0.2)

#define taxythta_ana_ticks 0.4625 // H taxythta tou ka8e troxou einai u=ticks*0.4625(m/s)
#define adxr300_sens 5 // 5mv/moira/sec
```

```

/// New defintions-----
#define wheelbase 0.54 // Wheelbase =0.54
#define mass 11 // 11 kg maza
#include "files/servos.h"
#define pi 3.14159265358979
#define sensitivity_def 0.5

void shift_values(double last,double previous[]) {
    previous[3]=previous[2];
    previous[2]=previous[1];
    previous[1]=previous[0];
    previous[0]=last;
}

// Nees routines 8-10-200

char * stabilization_1(double front_left_speed,double front_right_speed,double back_left_speed,double
back_right_speed,double gyro_degs_ana_sec,int throttle_servo,int steer_servo,double steer,int*
stab_int_parameters,double* stab_double_parameters,long time_passed);

char * stabilization_2(double front_left_speed,double front_right_speed,double back_left_speed,double
back_right_speed,double gyro_degs_ana_sec,int throttle_servo,int steer_servo,double steer,int*
stab_int_parameters,double* stab_double_parameters,long time_passed);

char * stabilization_3(double acc_12_X_gs,double acc_12_Y_gs,double acc_2_front_X_gs,double
acc_2_front_Y_gs,double acc_2_back_X_gs,
double acc_2_back_Y_gs,double front_left_speed,double
front_right_speed,double back_left_speed,double back_right_speed,
double pont_volts,double gyro_degs_ana_sec,double batt_volts,int
throttle_servo,int steer_servo,double steer,int* stab_int_parameters,double* stab_double_parameters);

//Dio 10_2_08
#include "stab/stabilization_routine1.c"
#include "stab/stabilization_routine2.c"
#include "stab/stabilization_routine3.c"
//

//ALLAGH APO BILL
// gia na tre3ei san thread apo to diko mou programma , bgazw thn main

//---> main()

//kai bazw
void* serial_comm(void *comm_data)
{
// kanw cast to comm_data sto struct pou prepei na einai
struct context_struct* bill_data = (struct context_struct*) comm_data;

// twra mporei na ektelestei apo to diko mou programma kai na antalla3ei dedomena mesw tou bill_data
// h domh tou bill_data dld to context_struct orizetai sto context.h kai exei ena float array gia na pairnw
//ta dedomena sou kai ena unsigned char array sto opoio sou stelnw dedomena

// px bill_data->command_buffer[0] einai to stabilization method pou exei stalei

    int bill_wheel=0; //metabhlth gia to frenarisma se ka8e troxo ,gia ka8e
troxo pros8etoume thn antistoixh timh fl +8 fr +4 bl +2 br +1
    int bill_brake=0; //metabhlth gia to state tou brake decision: 0 neutral 1
over 2 under
    unsigned char bill_local_command=0; // Metabhlth gia to an 8a kanei h oxi stabilization, Diomidis, Sto
0 den grafei sta txt arxeia

```



```

    unsigned char bill_local_command_last_time=1;
//TELOS ALLAGHS

int fd,c, res,n;
struct termios oldtio,newtio;
struct sigaction saio;
char buf[100];
int hr8e;
int posa;
int bytes=0,w_bytes=0;
long deigmata=0;
// Acceleration 1.2 g
//int acc_12_X=0,acc_12_Y=0;
int acc_12_X_high=0,acc_12_X_low=0;
int acc_12_Y_high=0,acc_12_Y_low=0;
double acc_12_X_gs=0,acc_12_Y_gs=0;
double acc_12_X_gs_buf[4],acc_12_Y_gs_buf[4];

// 2g front Accelerometer
int acc_2_front_X_low=0,acc_2_front_X_high=0;
double acc_2_front_X_gs=0;
// X axis, High kai low nibble
// H
metrhsh se gs
double acc_2_front_X_gs_buf[4];
int acc_2_front_Y_low=0,acc_2_front_Y_high=0;
double acc_2_front_Y_gs=0;
double acc_2_front_Y_gs_buf[4];
// Y axis, High kai low nibble
// 2g back Accelerometer
int acc_2_back_X_low=0,acc_2_back_X_high=0;
double acc_2_back_X_gs=0;
double acc_2_back_X_gs_buf[4];
// X axis, High kai low nibble
// H metrhsh se gs
int acc_2_back_Y_low=0,acc_2_back_Y_high=0;
double acc_2_back_Y_gs=0;
double acc_2_back_Y_gs_buf[4];
// Y axis, High kai low nibble
// Wheel Speeds
int front_left_ticks=0, front_right_ticks=0,back_left_ticks=0,back_right_ticks=0; // Ticks
double front_left_speed=0, front_right_speed=0,back_left_speed=0,back_right_speed=0; // Speed, m/sec
// Pontentiometer gia gwnia stripsimatos
int pont_high=0;
int pont_low=0;
double pont_volts=0;
double steer=0;
double pont_volts_buf[4];
// Posa Volt blepei
// Poses moires strabei
// Gyro
int gyro_high=0;
int gyro_low=0;
int gyro_high_last_value=0;
int gyro_low_last_value=0;
double gyro_volts=0;
double gyro_volts_buf[4];
double gyro_degs_ana_sec=0;
// Posa Volt blepei
// Moires Ana sec
// Battery
int batt_high=0;
int batt_low=0;
double batt_volts=0;
// Posa Volt blepei
// throttle_servo
int throttle_servo=0;
// steer_servo
int steer_servo=0;

// Metablhtes gia stabilization

```

```

int command=0; // Apo aythn thn metablth
epilegetai to eidosis sta8eropoihs
int stab_int_parameters[32];
//double stab_double_parameters[32];
double *stab_double_parameters= new (nothrow) double[32];

//19_2_08 Xrhisimopoieitai gia to aytomato stripsimo
unsigned char *local_command_buffer=new (nothrow) unsigned char [10];
unsigned char auto_speed=0;
unsigned char auto_steer=0;
unsigned char auto_chronos=0;
unsigned char auto_times=0;
long auto_chronos_start=0;
long hold_for_3secs=0;
bool auto_triggered=false;
int auto_trig=0;

//19_2_08

///-----
// Metablhtes prin stabilization
static double sensitivity=sensitivity_def;//0.9;
double radius_des=0,radius_actual=0,slip_angle=0, velocity=0;
char *brake_decision;
brake_decision = new (nothrow) char[30];
if (brake_decision == 0){
    std::cout << "Error: memory could not be allocated";
    kill(getpid(),0);
}
///-----

char *stab_commands;
stab_commands =new (nothrow) char [32];
if (stab_commands == 0){
    std::cout << "Error: memory could not be allocated";
    kill(getpid(),0);
}

//-----

// Arxikopoihsh tw'n buffers gia to ADC
for(n=0;n<4;n++){
    acc_2_front_X_gs_buf[n]=0;
    acc_2_front_Y_gs_buf[n]=0;
    acc_2_back_X_gs_buf[n]=0;
    acc_2_back_Y_gs_buf[n]=0;
    acc_12_X_gs_buf[n]=0;
    acc_12_Y_gs_buf[n]=0;
    pont_volts_buf[n]=0;
}

// Servo Outputs
char front_left_servo=47; // Arxika htan 12 me TOVF=128us. Twra exw
TOVF=32us
char front_right_servo=47;
char back_left_servo=47;
char back_right_servo=47;

// 22-8-07

int start_address;
char serial_port_buffer[200];
//char testing[] = "nai ola ok";

```

```

bool consume_data=FALSE;
int last_append=0; // O deikths gia ths teleytaias
prosarthshs

// up to here 22-8-07

int start,end; //DIka mou
char *data,*previous_valid_data; //data=malloc(70*sizeof(char)); //
data = new (nothrow) char[70]; //Gia c++
if (data == 0){
    std::cout << "Error: memory could not be allocated";
    kill(getpid(),0);
}
char *write_buffer;
write_buffer = new (nothrow) char[32] ;
if (write_buffer == 0){
    std::cout << "Error: memory could not be allocated";
    kill(getpid(),0);
}

for (n=0;n<32;n++){
    write_buffer[n]='a';
}
char *pch;
char temp_char[20];
char *big_buffer;
big_buffer = new char [350];
if (big_buffer == 0)
{
    std::cout << "Error: memory could not be allocated";
    kill(getpid(),0);
}
char *big_buffer_new;
big_buffer_new = new char [250];
if (big_buffer_new == 0)
{
    std::cout << "Error: memory could not be allocated";
    kill(getpid(),0);
}

/*
FILE *fptr;
if((fptr=fopen(logfile,"w"))==NULL)
{
    perror(logfile); kill(getpid(),0); } //kill(getpid(),0) anti gia exit(-1) pou bgazei warning
big_buffer=return_date();
strcat(big_buffer,"\n\n");
fputs(big_buffer,fptr);
*/

//-----
/*
FILE *fptrmath; // Tab delimited
file
if((fptrmath=fopen("dummy","w"))==NULL)
{
    perror(logmath); kill(getpid(),0); } //kill(getpid(),0) anti gia exit(-1) pou bgazei warning
//
fputs(big_buffer,fptrmath); // Grapse thn hmeromhnia kai se ayto
to arxeio
*/// 15_2_08
//ALLAGH APO BILL
FILE *billfptr;
if ((billfptr=fopen(logmath,"a+"))==NULL) // a+, anoigei h dhmiourgei ena neo arxeio gia
prosarthsh, sel 358

```

```

{    perror(logmath); kill(getpid(),0);    }
big_buffer=return_date();
fputs(big_buffer,billfptr);
//TELOS ALLAGHS

char *big_buffer_math;
big_buffer_math =new char [300];
if (big_buffer_math == 0)
{
    std::cout << "Error: memory could not be allocated";
    kill(getpid(),0);
}
// 17_2_08==== O xronos se Integers Arxh
int *time_int;
time_int=new int[4];
time_int[0]=0;
time_int[1]=0;
time_int[2]=0;
time_int[3]=0;
int hour=0;
int min=0;
int sec=0;
int usec=0;
// 17_2_08==== O xronos se Integers telos

//-----
fd=open_serial();                                     //include init_serial.c
// H thn init_serial h ta akolou8a
init_serial(fd,saio,oldtio,newtio);                   //include init_serial.c
//////////+++++//////////
// Oti lepei einai sto init_stuff_serial.c
//////////+++++//////////

struct timeval tp;                                     //Apo http://cygwin.com/ml/cygwin/2002-
01/msg01408.html
long time_start,time_end;
long time_passed=0,time_passed_sum=0;

gettimeofday(&tp,0);
time_start = ((unsigned)tp.tv_sec)*1000000+((unsigned)tp.tv_usec);

printf("Debug1\n");
//ALLAGH APO BILL
bill_data->serial_running=true;
//TELOS ALLAGHS APO BILL

res = read(fd,buf,100);

printf("Debug2: %d res\n",res);
//ALLAGH APO BILL anti gia 1 bazw dih mou metablhth gia na elegxw pote 3ekinaei / stamataei
while(1)//while (!bill_data->serial_stop)    //16_2_08
//TELOS ALLAGHS
{
    // An einai true jekina neo log file, exw xrhsimopoihsei to serial_stop gia dhmiourgia neou log file
    if(bill_data->serial_stop){
        bill_data->serial_stop=false;
        fclose(billfptr);
        big_buffer=return_date();
        if ((billfptr=fopen(logmath,"w"))==NULL){
            perror(logmath);

```



```

time_passed=(long)(time_end-time_start);

gettimeofday(&tp,0);
time_start = ((unsigned)tp.tv_sec)*1000000+((unsigned)tp.tv_usec);

//-----1.2g Accelerometer-----
acc_12_X_high = (int)(data[3]);
char_to_int(acc_12_X_high);
if(acc_12_X_high==254)
    acc_12_X_high=0;
acc_12_X_low = (int)(data[4]);
char_to_int(acc_12_X_low);
acc_12_X_high=(256*acc_12_X_high)+acc_12_X_low;
acc_12_X_gs = Acceleration_12(acc_12_X_high,455);

acc_12_X_gs=forgeting_sum(acc_12_X_gs,acc_12_X_gs_buf);
shift_values(acc_12_X_gs,acc_12_X_gs_buf);

acc_12_Y_high = (int)(data[5]);
char_to_int(acc_12_Y_high);
if(acc_12_Y_high==254)
    acc_12_Y_high=0;
acc_12_Y_low = (int)(data[6]);
char_to_int(acc_12_Y_low);
acc_12_Y_high=(256*acc_12_Y_high)+acc_12_Y_low;
acc_12_Y_gs = Acceleration_12(acc_12_Y_high,458);

acc_12_Y_gs=forgeting_sum(acc_12_Y_gs,acc_12_Y_gs_buf);
shift_values(acc_12_Y_gs,acc_12_Y_gs_buf);
//-----

//-----2g Front Accelerometer-----
// -----X axis-----
acc_2_front_X_high = (int)(data[7]);
char_to_int(acc_2_front_X_high);
if(acc_2_front_X_high==254)
    acc_2_front_X_high=0;
acc_2_front_X_low = (int)(data[8]);
char_to_int(acc_2_front_X_low);
acc_2_front_X_gs=(256*acc_2_front_X_high)+acc_2_front_X_low;
acc_2_front_X_gs=((acc_2_front_X_gs * 5)/ 1024); // Se volt twra sel 214 ATmega16;
acc_2_front_X_gs=((acc_2_front_X_gs-2.5)/0.312);

acc_2_front_X_gs=forgeting_sum(acc_2_front_X_gs,acc_2_front_X_gs_buf);
shift_values(acc_2_front_X_gs,acc_2_front_X_gs_buf);
//-----

// -----Y axis-----
acc_2_front_Y_high = (int)(data[9]);
char_to_int(acc_2_front_Y_high);
if(acc_2_front_Y_high==254)
    acc_2_front_Y_high=0;
acc_2_front_Y_low = (int)(data[10]);
char_to_int(acc_2_front_Y_low);
acc_2_front_Y_gs=(256*acc_2_front_Y_high)+acc_2_front_Y_low;
acc_2_front_Y_gs=((acc_2_front_Y_gs * 5)/ 1024); // Se volt twra sel 214 ATmega16;
acc_2_front_Y_gs=((acc_2_front_Y_gs-2.5)/0.312);
acc_2_front_Y_gs=forgeting_sum(acc_2_front_Y_gs,acc_2_front_Y_gs_buf);
shift_values(acc_2_front_Y_gs,acc_2_front_Y_gs_buf);
//-----

//-----2g Back Accelerometer-----

```

```

// -----X axis-----
acc_2_back_X_high = (int)(data[11]);
char_to_int(acc_2_back_X_high);
if(acc_2_back_X_high==254)
    acc_2_back_X_high=0;
acc_2_back_X_low = (int)(data[12]);
char_to_int(acc_2_back_X_low);
acc_2_back_X_gs=(256*acc_2_back_X_high)+acc_2_back_X_low;
acc_2_back_X_gs=((acc_2_back_X_gs * 5)/ 1024); // Se volt twra sel 214 ATmega16;
acc_2_back_X_gs=((acc_2_back_X_gs-2.5)/0.312);

acc_2_back_X_gs=forgeting_sum(acc_2_back_X_gs,acc_2_back_X_gs_buf);
shift_values(acc_2_back_X_gs,acc_2_back_X_gs_buf);
// -----Y axis-----
acc_2_back_Y_high = (int)(data[13]);
char_to_int(acc_2_back_Y_high);
if(acc_2_back_Y_high==254)
    acc_2_back_Y_high=0;
acc_2_back_Y_low = (int)(data[14]);
char_to_int(acc_2_back_Y_low);
acc_2_back_Y_gs=(256*acc_2_back_Y_high)+acc_2_back_Y_low;
acc_2_back_Y_gs=((acc_2_back_Y_gs * 5)/ 1024); // Se volt twra sel 214 ATmega16;
acc_2_back_Y_gs=((acc_2_back_Y_gs-2.5)/0.312);

acc_2_back_Y_gs=forgeting_sum(acc_2_back_Y_gs,acc_2_back_Y_gs_buf);
shift_values(acc_2_back_Y_gs,acc_2_back_Y_gs_buf);
//-----

// Wheel speeds==> Einai oi palmoi pou blepoun ta hall effects mesa se 0.1 sec.
// To montelo me 8 hall effect sensors, gia ka8e palmo dianyei 4,625 cm.
// Seira pou erxontai. front_left_speed, front_right_speed, back_left_speed,

back_right_speed

//          int front_left_ticks=0, front_right_ticks=0,back_left_ticks=0,back_right_ticks=0;

// Ticks

//double front_left_speed=0, front_right_speed=0,back_left_speed=0,back_right_speed=0;

// Speed, m/sec

front_left_ticks = (int)(data[15]);
char_to_int(front_left_ticks);
if(front_left_ticks==254)front_left_ticks=0; // An htan 254 einai 0
front_left_speed=front_left_ticks*taxythta_ana_ticks;

front_right_ticks = (int)(data[16]);
char_to_int(front_right_ticks);
if(front_right_ticks==254)front_right_ticks=0; // An htan 254

einai 0

front_right_speed=front_right_ticks*taxythta_ana_ticks;

back_left_ticks = (int)(data[17]);
char_to_int(back_left_ticks);
if(back_left_ticks==254)back_left_ticks=0; // An htan 254 einai 0
back_left_speed=back_left_ticks*taxythta_ana_ticks;

back_right_ticks = (int)(data[18]);
char_to_int(back_right_ticks);
if(back_right_ticks==254)back_right_ticks=0; // An htan

254 einai 0

back_right_speed=back_right_ticks*taxythta_ana_ticks;

velocity=(front_left_speed+front_right_speed+back_left_speed+back_right_speed)/4;

// Pontesiometro gia gwnia timoniou

```

```

    pont_high = (int)(data[19]);
    char_to_int(pont_high);
    if(pont_high==254)
        pont_high=0;
    pont_low = (int)(data[20]);
    char_to_int(pont_low);
    pont_volts=(256*pont_high)+pont_low;
    pont_volts=((pont_volts * 5)/ 1024); // Se volt

twra sel 214 ATmega16;
    pont_volts=forgeting_sum_steel(pont_volts,pont_volts_buf);
    shift_values(pont_volts,pont_volts_buf);
    steer=1000*( (0.31570260254613*pow(pont_volts,3)) + (-
1.97189807334590*pow(pont_volts,2))+ (3.98997966117411 *pow(pont_volts,1)) -2.60471114838461);
    steer=1;

// Gyroskopio
    gyro_high = (int)(data[21]);
    char_to_int(gyro_high);
    if(gyro_high==254)
        gyro_high=0;
    gyro_low = (int)(data[22]);
    char_to_int(gyro_low);

    gyro_high_last_value=0;
    gyro_low_last_value=0;

    gyro_volts=(256*gyro_high)+gyro_low;
    gyro_volts=((gyro_volts * 5)/ 1024); // Se volt twra sel 214 ATmega16;
    gyro_volts=forgeting_sum(gyro_volts,gyro_volts_buf);
    shift_values(gyro_volts,gyro_volts_buf);
    // To gyroskopio exei 5mv/ana sec eyais8hsia
    gyro_degs_ana_sec=floor(((gyro_volts-2.50)/0.005)); // Ta 2.866V einai se 0 moires/sec To
apotelesma einai se moires/sec
    gyro_degs_ana_sec++;
    //printf("gyro_high:%d \t gyro_low:%d \t Gy:%3.0f
\n",gyro_high,gyro_low,gyro_degs_ana_sec);

// Mpataria
    batt_high = (int)(data[23]);
    char_to_int(batt_high);
    if(batt_high==254)
        batt_high=0;
    batt_low = (int)(data[24]);
    char_to_int(batt_low);
    batt_volts=(256*batt_high)+batt_low;
    batt_volts=((batt_volts * 5)/ 1024); // Se volt twra sel 214 ATmega16;
    batt_volts=batt_volts*2; //Brisketai se diaireth tashs

// Ta throttle kai steer servo ta stelnw anapoda apo to atmega. 25, kai 26 antistoixa
// steer_servo
    steer_servo = (int)(data[25]);
    char_to_int(steer_servo);
    if(steer_servo==254)
        steer_servo=0;

// throttle_servo
    throttle_servo = (int)(data[26]);
    char_to_int(throttle_servo);
    if(throttle_servo==254)
        throttle_servo=0;

// O xronos trexei parolo pou den ta grafei sta log arxeia!
////////////////////

```



```

time_passed_sum=time_passed+time_passed_sum;
if((bill_local_command!=0)&&(bill_local_command_last_time==0)){ // An htan sto 0
jekina na grafeis!!!

    big_buffer=return_date();
    strcat(big_buffer,"Start writing Again\n");
    //fputs(big_buffer,fptr);
    fputs(big_buffer,billfptr);
    bill_local_command_last_time=1;
    printf("bill_local_command:%d\n",bill_local_command,bill_local_command_last_time);
}

if(bill_local_command!=0)
{
    // Edw kalountai oi stabilization routines!
    switch (bill_local_command){
        case 1:
            //sensitivity=stab_double_parameters[0]
            //understeer_coeff=stab_double_parameters[1]
            stab_commands=stabilization_1( front_left_speed,
front_right_speed, back_left_speed, back_right_speed,
gyro_degs_ana_sec, throttle_servo,
steer_servo,steer,stab_int_parameters,stab_double_parameters,time_passed);
            break;
        case 2:
            //sensitivity=stab_double_parameters[0]
            //understeer_coeff=stab_double_parameters[1]
            stab_commands=stabilization_2( front_left_speed,
front_right_speed, back_left_speed, back_right_speed,
gyro_degs_ana_sec, throttle_servo,
steer_servo,steer,stab_int_parameters,stab_double_parameters,time_passed);
            break;
        case 3:
            //sensitivity=stab_double_parameters[0]
            //non_linear_coef=stab_double_parameters[1]
            stab_commands=stabilization_3( acc_12_X_gs, acc_12_Y_gs,
acc_2_front_X_gs, acc_2_front_Y_gs, acc_2_back_X_gs,
acc_2_back_Y_gs, front_left_speed,
front_right_speed, back_left_speed, back_right_speed,
pont_volts, gyro_degs_ana_sec,
batt_volts, throttle_servo, steer_servo,steer,stab_int_parameters,stab_double_parameters);
            break;
        case 8: // Do nothing but log data
            stab_commands[2]=front_left_servo_release ;
            stab_commands[3]=front_right_servo_brake;
            stab_commands[4]=back_left_servo_release;

            stab_commands[5]=back_right_servo_release;
            //printf("Log_data\n");
            break;
        default:
            //sensitivity=stab_double_parameters[0]
            //understeer_coeff=stab_double_parameters[1]
            stab_commands=stabilization_1( front_left_speed,
front_right_speed, back_left_speed, back_right_speed,
gyro_degs_ana_sec, throttle_servo,
steer_servo,steer,stab_int_parameters,stab_double_parameters,time_passed);
            break;
    }

    //*****//

```



```

// antigraph twndedomenwn sto koino struct gia apostolh se client
#ifdef MUTEX_SR
//lock mutex and write the data
pthread_mutex_lock(&(bill_data->mutex_receive));
#endif
bill_data->read_data[0]=acc_12_X_gs;
bill_data->read_data[1]=acc_12_Y_gs;
bill_data->read_data[2]=acc_2_front_X_gs;
bill_data->read_data[3]=acc_2_front_Y_gs;
bill_data->read_data[4]=acc_2_back_X_gs;
bill_data->read_data[5]=acc_2_back_Y_gs;
bill_data->read_data[6]=front_left_speed*3.6;
bill_data->read_data[7]=front_right_speed*3.6;
bill_data->read_data[8]=back_left_speed*3.6;
bill_data->read_data[9]=back_right_speed*3.6;
bill_data->read_data[10]=velocity*3.6;
bill_data->read_data[11]=gyro_degs_ana_sec;
bill_data->read_data[12]=batt_volts;
bill_data->read_data[13]=steer;

bill_wheel=0;
if(stab_commands[2]==front_left_servo_brake) bill_wheel+=8;
if(stab_commands[3]==front_right_servo_brake) bill_wheel+=4;
if(stab_commands[4]==back_left_servo_brake) bill_wheel+=2;
if(stab_commands[5]==back_right_servo_brake) bill_wheel+=1;

bill_data->read_data_int[3]=bill_wheel;
bill_data->read_data_int[2]=bill_brake;
bill_data->read_data_int[1]=throttle_servo;
bill_data->read_data_int[0]=steer_servo;
//enhmerwnoume metablhts gia na 3erei o server oti exoun er8ei kainourgia dedomena
bill_data->data_received=true;
#ifdef MUTEX_SR
//unlock mutex after data is written
pthread_mutex_unlock(&(bill_data->mutex_receive));
#endif

//paradeigma pws pairneis merika apo ta dedomena pou exoun stalei
//einai protimotero na antigrafoun ola me thn mia se topikes metablhtes
if (bill_data->command_received==true){
#ifdef MUTEX_SR
//lock mutex and read the data
pthread_mutex_lock(&(bill_data->mutex_send));
#endif
printf("Command bill=%u\n",bill_data->command_buffer[0]);

// Dio
bill_local_command_last_time=bill_local_command; // O
bill_local_command_last_time krataei thn teleytaia timh pou eixa o bill_local_command
bill_local_command=bill_data->command_buffer[0];

stab_double_parameters=bill_data->double_buffer;
//19_2_08 Xrhsimopoieitai gia to aytomato stripsimo
local_command_buffer=bill_data->command_buffer;
auto_speed=local_command_buffer[1];
auto_steer=local_command_buffer[2];
auto_chronos=local_command_buffer[3];
auto_times=local_command_buffer[4]+1;

/*
long auto_chronos_start=0;
bool auto_triggered=false;
*/

```

```

//19_2_08

bill_data->command_received=false;
#ifdef MUTEX_SR
//unlock mutex after data is read
pthread_mutex_unlock(&(bill_data->mutex_send));
#endif
} //TELOS ALLAGHS
/*
Char Commands:
If velocity > [1]m/sec then steer [2] degrees for [3]*10ms, repeat [4] times
Velocity:[1] Steer:[2] Holdfor:[3] Repeat:[4] :[5] :[6]
*/
//-----Auto steer-----//
if(auto_times>1){
    printf("Auto Steer Pending for:%d times",auto_times);
    if((auto_triggered==false)&&
(((unsigned)tp.tv_sec)*1000000+((unsigned)tp.tv_usec)-hold_for_3secs)>3000000)){ // Perimene na
perasoun 3 secs

        if(velocity>=((unsigned int)auto_speed)){
            printf("velocity:%f >=auto_speed:%u \n",velocity,auto_speed);
            auto_triggered=true;
            gettimeofday(&tp,0);
            auto_chronos_start = ((unsigned)tp.tv_sec)*1000000+((unsigned)tp.tv_usec);
            auto_times--;
            stab_commands[7]=auto_steer; // Stripse toses

            printf("Auto Triggered auto_chronos_start%d auto_steer:%d
auto_times:%d\n",auto_chronos_start/1000,auto_steer,auto_times);
        }
    } //if(auto_times>0)
    if(auto_triggered==true){
        if((((unsigned)tp.tv_sec)*1000000+((unsigned)tp.tv_usec)-
auto_chronos_start)<(auto_chronos*500000)){
            stab_commands[7]=auto_steer;
        }
        else{ // Time passed
            stab_commands[7]=254;
            printf("Times Up auto_chronos_start%d auto_steer:%d
auto_times:%d\n",(((unsigned)tp.tv_sec)*1000000+((unsigned)tp.tv_usec))/1000,auto_steer,auto_times);
            auto_triggered=false;
            hold_for_3secs=((unsigned)tp.tv_sec)*1000000+((unsigned)tp.tv_usec);
        }
    }
} //-----Auto steer-----//

// epeidh exoume ena servo frenaroume kai tous dyo mazi an frenarei o enas;

if((stab_commands[4]==back_left_servo_brake)||((stab_commands[5]==back_right_servo_brake)){
    stab_commands[4]=back_left_servo_brake; // Frenare to
aristero pishw troxo!

    stab_commands[5]=back_right_servo_brake; // Frenare to deji
pishw troxo!

}

consume_data=FALSE; //Data consumed
} // consume_data=TRUE

```

```

        if((bill_local_command!=0)&&(bill_local_command!=8)){ // bill_local_command=8 ==> Do
nothing, log data only
            // Ayto to tmhma grafei sthn seiriakh.
            ioctl(fd,TIOCOUTQ,&w_bytes);
            if(w_bytes==0){
                write_buffer=&stab_commands[0];
                write_buffer[0]=255; // Start 255, terminator

                n=write(fd,write_buffer,32);
                if (n < 0)
                    fputs("write() of 32 bytes failed!\n", stderr);
            }//(w_bytes==0)
        }//if(bill_local_command!=0)

    }//while (1)

/* restore old port settings */
tcsetattr(fd,TCSANOW,&oldtio);

//ALLAGH APO BILL
    bill_data->serial_running=false;
    bill_data->serial_stop=false;
//TELOS ALLAGHS APO BILL
}

```

headers.h

```

#include <termios.h>
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/signal.h>
#include <sys/types.h>
#include <stropts.h>
#include <termio.h>
#include <string.h>
#include <time.h>
#include <stdlib.h>
#include "rdtsc.h" //http://www-unix.mcs.anl.gov/~kazutomo/rdtsc.html
#include <cstdlib> // Gia malloc
#include <math.h>
#include <sys/time.h>
#include <linux/tty.h>

#define BAUDRATE B115200 //B230400
#define MODEMDEVICE "/dev/ttyS0"
#define _POSIX_SOURCE 1 /* POSIX compliant source */
#define FALSE 0
#define TRUE 1

```

init_serial.c

```
#include "headers.h"

/*****
 * signal handler. sets wait_flag to FALSE, to indicate above loop that
 * characters have been received.
 *****/

void signal_handler_IO (int status)
{
    //printf("signaled\n")    ;
    wait_flag = FALSE;
}

int open_serial(void)
{
    int fd;
    /* open the device to be non-blocking (read will return immediatly) */;
    fd = open(MODEMDEVICE, O_RDWR | O_NOCTTY | O_NONBLOCK);
    if (fd < 0) {perror(MODEMDEVICE); kill(getpid(),0); }
    return fd;
}

void init_serial(int &fd,struct sigaction &saio,struct termios &oldtio,struct termios &newtio)
{
    ///////////////////////////////////////////////////
    // install the signal handler before making the device asynchronous
    saio.sa_handler = signal_handler_IO;
    saio.sa_flags = 0;
    saio.sa_restorer = NULL;
    sigaction(SIGIO,&saio,NULL);

    // allow the process to receive SIGIO
    fcntl(fd, F_SETOWN, getpid());
    // Make the file descriptor asynchronous (the manual page says only
    //O_APPEND and O_NONBLOCK, will work with F_SETFL...)
    fcntl(fd, F_SETFL, FASYNC);

    tcgetattr(fd,&oldtio); // save current port settings

    //async1
    //
    // Set bps rate and hardware flow control and 8n1 (8bit,no parity,1 stopbit).
    // Also don't hangup automatically and ignore modem status.
    //Finally enable receiving characters.

    newtio.c_cflag = BAUDRATE | CRTSCTS | CS8 | CLOCAL | CREAD;

    newtio.c_iflag = IGNPAR; //Ignore bytes with parity errors and make terminal raw and dumb.

    newtio.c_oflag = 0;      //Raw output.

    //Don't echo characters because if you connect to a host it or your
    //modem will echo characters for you. Don't generate signals.

    newtio.c_lflag = 0;

    // blocking read until 1 char arrives
```

```

newtio.c_cc[VMIN]=1;
newtio.c_cc[VTIME]=0;

// now clean the modem line and activate the settings for modem
tcflush(fd, TCIFLUSH);
tcsetattr(fd,TCSANOW,&newtio);
}

```

time_etc.cpp

```

char * return_date(void)
{
    char buffer[30];
    struct timeval tv;
    char * bigger_buffer;
    bigger_buffer= new char[100];

    time_t curtime;

    gettimeofday(&tv, NULL);
    curtime=tv.tv_sec;

    strftime(buffer,30,"%m-%d-%Y %T.",localtime(&curtime));
    sprintf(bigger_buffer,"%s%ld\n",buffer,tv.tv_usec);

    return bigger_buffer;
}

```

```

int * return_time(void)
{
    int sec=0;
    int usec=0;
    int hour=0;
    int min=0;
    int *time_int;
    time_int=new int[4];

    char buffer[30];
    char buffer2[3];
    struct timeval tv;

    time_t curtime;
    gettimeofday(&tv, NULL);
    curtime=tv.tv_sec;

    strftime(buffer,30,"%T.",localtime(&curtime));

    strncpy(buffer2,&buffer[0],2*sizeof(char));
    buffer[2]='\0';
    hour=atoi(buffer2);

    strncpy(buffer2,&buffer[3],2*sizeof(char));
    buffer[2]='\0';
    min=atoi(buffer2);

    strncpy(buffer2,&buffer[6],2*sizeof(char));

```

```

buffer[2]='\0';
sec=atoi(buffer2);
usec=(int)tv.tv_usec;

//printf("hour:%d min:%d sec:%d usec:%d \n",hour,min,sec,tv.tv_usec);
time_int[0]=hour;
time_int[1]=min;
time_int[2]=sec;
time_int[3]=usec;

return time_int;
}

```

servos.h

// Definition gia tis 8eseis twn servo!

```

#define front_left_servo_brake 57
#define front_left_servo_release 47
#define front_right_servo_brake 37
#define front_right_servo_release 47
#define back_left_servo_brake 33
#define back_left_servo_release 47
#define back_right_servo_brake 47
#define back_right_servo_release 33

```

stabilization_routine1.c

/**-----Stabilization_method_1-----**/

```

char * stabilization_1(double front_left_speed,double front_right_speed,double back_left_speed,double back_right_speed,double
gyro_degs_ana_sec,int throttle_servo,int steer_servo,double steer,int* stab_int_parameters,double* stab_double_parameters,long
time_passed){

```

// Analoga me to pou an einai aristerh h dexia strofh. Dejia einai 8etikh. 8etikh einai epishs kai h epitaxynsh

```

static double sensitivity=0.9; // 0<sensitivity<1
static char *stab_commands;
stab_commands=new (nothrow) char [32];
if (stab_commands == 0){
    std::cout << "Error: memory could not be allocated";
    kill(getpid(),0);
}
stab_commands[0]=0;

```

```

static double time_passed_array[10];
static double time_passed_sum=0;

```

```

static double understeer_coeff=0;
double yaw_rate_desired=0,velocity=0;

```

```

if((stab_double_parameters[0]>0)&&(stab_double_parameters[0]<=1))
    sensitivity=stab_double_parameters[0]; // understeer_coefficient
else
    sensitivity=0.9;

```

```

understeer_coeff=stab_double_parameters[1];
// understeer_coefficient

```



```

velocity=(back_left_speed+back_right_speed+front_right_speed+front_left_speed)/4;
yaw_rate_desired=(velocity*steer/(wheelbase+(mass*velocity*velocity*understeer_coeff/(2*wheelbase))))/*(steer*pi/180);
// Einai se rad/sec
//yaw_rate_desired=yaw_rate_desired*(180/pi);

// Twra einai se moires/sec!!!

printf("steer:%2.1f gyro:%3.0f\t yaw_rate_desired:%3.3f\t velocity:%2.2f\t understeer_coeff:%3.7f\t
sensitivity%1.2f\n",steer,gyro_degs_ana_sec,yaw_rate_desired,velocity,understeer_coeff,sensitivity);

if (fabs(gyro_degs_ana_sec)<=10){ //If gyro_degs_ana_sec<=10 do nothing
    //-----Release all servos;
    stab_commands[2]=front_left_servo_release;//47; //stab_commands[2]=front_left_servo;
    stab_commands[3]=front_right_servo_release;//47; //stab_commands[3]=front_right_servo;
    stab_commands[4]=back_left_servo_release;//47; //stab_commands[4]=back_left_servo;
    stab_commands[5]=back_right_servo_release;//33; //stab_commands[5]=back_right_servo;
    //-----
}
else{
    // Oversteer

if((fabs(gyro_degs_ana_sec)>=(fabs(fabs(yaw_rate_desired)*(1/sensitivity))+3))&&(fabs(gyro_degs_ana_sec)>=10)){
    // Pros8etw 5 gia otan einai yaw_rate_desired==0
    if(gyro_degs_ana_sec>=0){ // Dejia
        troxh
        stab_commands[2]=front_left_servo_brake ; // Frenare to mprosta aristera troxo
        stab_commands[3]=front_right_servo_release;
        stab_commands[4]=back_left_servo_release;
        stab_commands[5]=back_right_servo_release;
    }
    else{
        // Aristero oversteer
        troxo!
        stab_commands[2]=front_left_servo_release ; // Frenare to deji mprosta
        stab_commands[3]=front_right_servo_brake;
        troxo!
        stab_commands[4]=back_left_servo_release;
        stab_commands[5]=back_right_servo_release;
    }
}
else{
    if((fabs(gyro_degs_ana_sec)<fabs(yaw_rate_desired*sensitivity*0.8))&&(velocity>2) ){
        // Understeer
        // Dejia strofh
        if(steer>0){
            troxo!
            stab_commands[2]=front_left_servo_release ; // Frenare to deji pisw
            stab_commands[3]=front_right_servo_release;
            stab_commands[4]=back_left_servo_release;
            stab_commands[5]=back_right_servo_brake;
            troxo!
            printf("\nUndersteer\n");
        }
        else{
            // Aristerh strofh
            troxo!
            stab_commands[2]=front_left_servo_release ; // Frenare to aristero pisw
            stab_commands[3]=front_right_servo_release;
            stab_commands[4]=back_left_servo_brake;
            troxo!
            stab_commands[5]=back_right_servo_release;
        }
    }
    /*
    // epeidh exoume ena servo frenaroume kai tous dyo mazi;
    stab_commands[4]=back_left_servo_brake; // Frenare to aristero pisw troxo!

```

```

        stab_commands[5]=back_right_servo_brake;           // Frenare to deji pisw troxo!
        */
    }
    else{
        //-----Release all servos;
        stab_commands[2]=front_left_servo_release ;//47;
        //stab_commands[2]=front_left_servo;
        stab_commands[3]=front_right_servo_release;//47;
        //stab_commands[3]=front_right_servo;
        stab_commands[4]=back_left_servo_release;//47;
        //stab_commands[4]=back_left_servo;
        stab_commands[5]=back_right_servo_release;//33;
        //stab_commands[5]=back_right_servo;
        //-----
    }
}

} // else

return stab_commands;
}

```

stabilization_routine2.c

```

/**-----Stabilization_method_2 ART-----**//
char * stabilization_2(double front_left_speed,double front_right_speed,double back_left_speed,double back_right_speed,double
gyro_degs_ana_sec,int throttle_servo,int steer_servo,double steer,int* stab_int_parameters,double* stab_double_parameters,long
time_passed){
    // Analoga me to pou an einai aristerh h dexia strofh. Dejia einai 8etikh. 8etikh einai epishs kai h epitaxyntsh

    static double sensitivity=0.9;           // 0<sensitivity<1
    static char *stab_commands;
    stab_commands=new (nothrow) char [32];
    if (stab_commands == 0){
        std::cout << "Error: memory could not be allocated";
        kill(getpid(),0);
    }
    stab_commands[0]=0;

    static double time_passed_array[10];
    static double time_passed_sum=0;

    static double understeer_coeff=0;
    double yaw_rate_desired=0,velocity=0;

    static int times_called=0;
    static double last_velocities [5][10];           // Aytos o pinakas periexei tis 10 prohgooumenes times gia tis taxythtes tw n
troxwn [1,:]:FL,[2,:]:FR,[3,:]:BL,[4,:]:BR. H [:,1] einai h pio prosfath h 8esh [5,:] einai dummy

    // Shift values by one;
    for(int pos=0;pos<9;pos++){
        last_velocities[1][pos+1]=last_velocities[1][pos];
        last_velocities[2][pos+1]=last_velocities[2][pos];
        last_velocities[3][pos+1]=last_velocities[3][pos];
        last_velocities[4][pos+1]=last_velocities[4][pos];
        time_passed_array[pos+1]=time_passed_array[pos];

        //printf("[1][0]:%2.2f,[2][0]:%2.2f,[3][0]:%2.2f,[4][0]:%2.2f\n[1][1]:%2.2f,[2][1]:%2.2f,[3][1]:%2.2f,[4][1]:%2.2f\n\n",last_
velocities[1][0],last_velocities[2][0],last_velocities[3][0],last_velocities[4][0],last_velocities[1][5],last_velocities[2][5],last_velociti
es[3][5],last_velocities[4][5]);
        time_passed_sum+=time_passed_array[pos];

```

```

}
//printf("Xronos se 10 deigmata time_passed=%f\n",time_passed_sum);
time_passed_sum=0;

last_velocities[1][0]=front_left_speed;
last_velocities[2][0]=front_right_speed;
last_velocities[3][0]=back_left_speed;
last_velocities[4][0]=back_right_speed;
time_passed_array[0]=(double)time_passed;
if((stab_double_parameters[0]>0)&&(stab_double_parameters[0]<=1))
    sensitivity=stab_double_parameters[0]; // understeer_coefficient
else
    sensitivity=0.9;

understeer_coeff=stab_double_parameters[1]/1000; // understeer_coefficient
velocity=(back_left_speed+back_right_speed+front_right_speed+front_left_speed)/4;
//velocity=(front_left_speed)*2;
yaw_rate_desired=(velocity/(wheelbase+(mass*velocity*velocity*understeer_coeff/(2*wheelbase))))*(steer*pi/180);
// Einai se rad/sec
yaw_rate_desired=yaw_rate_desired*(180/pi); // Twra einai se moires/sec!!!

printf("steer:%2.1f gyro:%3.0f\t yaw_rate_desired:%3.3f\t velocity:%2.2f\t understeer_coeff:%3.7f\t
sensitivity%1.2f\n",steer,gyro_degs_ana_sec,yaw_rate_desired,velocity,understeer_coeff,sensitivity);

if (fabs(gyro_degs_ana_sec)<=3){ //If gyro_degs_ana_sec<=10 do nothing
    //-----Release all servos;
    stab_commands[2]=front_left_servo_release;//47; //stab_commands[2]=front_left_servo;
    stab_commands[3]=front_right_servo_release;//47; //stab_commands[3]=front_right_servo;
    stab_commands[4]=back_left_servo_release;//47; //stab_commands[4]=back_left_servo;
    stab_commands[5]=back_right_servo_release;//33; //stab_commands[5]=back_right_servo;
    //-----
    //printf("\nDesired radius= apeirh\n");
}
else{
    if(fabs(gyro_degs_ana_sec)>=fabs(yaw_rate_desired*(1/sensitivity))){ // Oversteer
        if(gyro_degs_ana_sec>=0){ // Dejia
            stab_commands[2]=front_left_servo_release ;
            stab_commands[3]=front_right_servo_brake;
            stab_commands[4]=back_left_servo_release; // Frenare to deji mprosta
            stab_commands[5]=back_right_servo_release;
        }
        else{
            // Aristero oversteer
            stab_commands[2]=front_left_servo_brake ;
            stab_commands[3]=front_right_servo_release; // Frenare to mprosta
            stab_commands[4]=back_left_servo_release;
            stab_commands[5]=back_right_servo_release;
        }
    }
    else{
        if(fabs(gyro_degs_ana_sec)<fabs(yaw_rate_desired*sensitivity)){ //
            if(gyro_degs_ana_sec>=0){ // Dejia
                stab_commands[2]=front_left_servo_release ;
                stab_commands[3]=front_right_servo_release;
                stab_commands[4]=back_left_servo_release;
            }
        }
    }
}

```

```

troxo!
    stab_commands[5]=back_right_servo_brake; // Frenare to deji pisw

    printf("\nUndersteer\n");
}
else{
    // Aristerh strofh
    stab_commands[2]=front_left_servo_release ;
    stab_commands[3]=front_right_servo_release;
    stab_commands[4]=back_left_servo_brake; // Frenare to aristero pisw
troxo!
    stab_commands[5]=back_right_servo_release;
}
// epeidh exoume ena servo frenaroume kai tous dyo mazi;
stab_commands[4]=back_left_servo_brake; // Frenare to aristero pisw troxo!
stab_commands[5]=back_right_servo_brake; // Frenare to deji pisw troxo!
}
}

} // else

return stab_commands;
}

```

stabilization_routine3.c

```

/**-----Stabilization_method_3-----**/

char * stabilization_3(double acc_12_X_gs,double acc_12_Y_gs,double acc_2_front_X_gs,double acc_2_front_Y_gs,double
acc_2_back_X_gs,
double acc_2_back_Y_gs,double front_left_speed,double front_right_speed,double
back_left_speed,double back_right_speed,
double pont_volts,double gyro_degs_ana_sec,double batt_volts,int throttle_servo,int
steer_servo,double steer,int* stab_int_parameters,double* stab_double_parameters){
    // Analoga me to pou an einai aristerh h dexia strofh. Dejia einai 8etikh. 8etikh einai epishs kai h epitaxysh

    static char *stab_commands;
    stab_commands=new (nothrow) char [32];
    if (stab_commands == 0){
        std::cout << "Error: memory could not be allocated";
        kill(getpid(),0);
    }
    stab_commands[0]=0;

    static double sensitivity=0.9;
    static double non_linear_coef=0.1;
    double radius_des=0,radius_actual=0,slip_angle=0, velocity=0;
    sensitivity=stab_double_parameters[0]; // Sensitivity
    non_linear_coef=stab_double_parameters[1]; // non_linear_coefficient

    radius_des=(wheelbase*(velocity*non_linear_coef)/((pi/180)*(fabs(steer))));
    if (fabs(acc_12_Y_gs)>0.4)
        slip_angle=atan(acc_12_X_gs/acc_12_Y_gs);
    else
        slip_angle=0;
    //velocity=cos(slip_angle)*(back_left_speed+back_right_speed+front_right_speed+front_left_speed)/4;
    velocity=(back_left_speed+back_right_speed+front_right_speed+front_left_speed)/4;

```

```

radius_actual=(velocity*velocity)/(sqrt((acc_12_X_gs*acc_12_X_gs)+(acc_12_Y_gs*acc_12_Y_gs)));
printf("Desired radius=%4.1f m \t Actual_radiious=%4.1f m \t slip_angle=%4.4f \t sensitivity:%4.4f
non_linear_coef:%4.6f\n",radius_des,radius_actual,slip_angle,sensitivity,non_linear_coef);

if (fabs(steer)<=1){
    //-----Release all servos;
    //If steer_angle==0 do nothing
    stab_commands[2]=front_left_servo_release ;//47; //stab_commands[2]=front_left_servo;
    stab_commands[3]=front_right_servo_release;//47; //stab_commands[3]=front_right_servo;
    stab_commands[4]=back_left_servo_release;//47; //stab_commands[4]=back_left_servo;
    stab_commands[5]=back_right_servo_release;//33; //stab_commands[5]=back_right_servo;
    //-----
}
else{

    if(velocity>1){
        if(radius_actual>(radius_des*(1/sensitivity))){ // Understeer, apply rear brakes;
            stab_commands[2]=front_left_servo_release ;
            stab_commands[3]=front_right_servo_release;
            stab_commands[4]=back_left_servo_brake;
            stab_commands[5]=back_right_servo_brake;
        }
        else{
            if(radius_actual<(radius_des*sensitivity)){ // UnderSteer, apply front brakes
                stab_commands[2]=front_left_servo_brake ;
                stab_commands[3]=front_right_servo_brake;
                stab_commands[4]=back_left_servo_release;
                stab_commands[5]=back_right_servo_release;
            }
            else{ // Neutral Steer
                stab_commands[2]=front_left_servo_release ;//47;
                //stab_commands[2]=front_left_servo;
                stab_commands[3]=front_right_servo_release;//47;
                //stab_commands[3]=front_right_servo;
                stab_commands[4]=back_left_servo_release;//47;
                //stab_commands[4]=back_left_servo;
                stab_commands[5]=back_right_servo_release;//33;
                //stab_commands[5]=back_right_servo;
            }
        }
    }
    else{
        stab_commands[2]=front_left_servo_release ;//47; //stab_commands[2]=front_left_servo;
        stab_commands[3]=front_right_servo_release;//47; //stab_commands[3]=front_right_servo;
        stab_commands[4]=back_left_servo_release;//47; //stab_commands[4]=back_left_servo;
        stab_commands[5]=back_right_servo_release;//33; //stab_commands[5]=back_right_servo;
    }
}
return stab_commands;
}

```

9.1.2 Scripts

compile

```
#step 1
#set correct local address in defines.h ##define LADDR "192.168.86.2"

#step 2
#to compile run
echo "Compiling: g++ -c async.cpp socketserver.cpp"
g++ -c async.cpp socketserver.cpp
echo "Linking: g++ async.o socketserver.o -lpthread -o socketserver"
g++ async.o socketserver.o -lpthread -o socketserver
echo "Done!"
echo "Do you want me to execute y or n?"
read choice
if [ $choice = "y" ]
    then
        clear
        echo "Yes selected, Socketserver running!"
        ./socketserver
    else
        echo "Bye!"
fi

#step 3
#run
```

9.1.3 index.html

```
<html>
<body>
<p><b>
<applet code=demos.TestJApplet archive=index.jar width="800" height="700" align="left">
Your browser does not understand Java.
</applet></b></p>
<p><b>Top WebCam</b></p>
<p>
<APPLET CODE = "WebCamApplet.class" archive="applet.jar" WIDTH = "320" HEIGHT = "240">
<param name=URL value="http://192.168.1.101:8888">
<param name=FPS value="10">
<param name=width value="320">
<param name=height value="240">
</APPLET></p>
<p><b>Wheel Webcam</b></p>
<p><APPLET CODE = "WebCamApplet.class" archive="applet.jar" WIDTH = "320" HEIGHT = "240">
<param name=URL value="http://192.168.1.101:8887">
<param name=FPS value="10">
```

9.2 Microcontroller codes

9.2.1 ESC_new.c

```
#include <avr/io.h>
#include <math.h>
#include <avr/interrupt.h>
#include "header.h"
#include <avr/pgmspace.h>
#include <ctype.h>
#include <avr/sfr_defs.h>

// Routine definition
void InitPorts(void);
void USART_Init(unsigned int ubrr);
void update_values(void);
void Timers_Init(void);
void external_interrupts_init(void);
void adc_init(void);
void software_servo_int(void);

// Global variables

static u8 *TX_buffer;
static u8 TX_data_to_send[TX_buffer_length];

volatile static u8 TX_read;
static u8 RX_buffer[RX_buffer_length];
static u8 *Received_data;

volatile static u8 RX_write;
static u8 receive_full=0,fresh_data=0; //Receive Flag

static u8 send_data=1;//send_data=0;
static u8 number=0;
static u8 temp;
// Accelerometers
#include "files/adc_accelerometers.c" //ADC kai Accelerometers Variables

#include "files/interrupt_routines.c"
#include "files/servos.h" // Se poies 8eseis frenaroun ta servo
//#include "files/stability.c"
//-----main-----
int main(void)
{
    sei();
    adc_init();
    InitPorts();
    external_interrupts_init();

    USART_Init(MYUBRR); //115.2K
    Timers_Init();
    static u8 servo_propotion=0; // Poso na kinh8oun ta servo tou timoniou
    static u8 esp_on=0;

    while(1)
    {

        /*if (receive_full==1)
```

```

{
    TX_buffer=&RX_buffer[0];
    //TX_buffer=&IDN[0];
    //TX_buffer=pgm_read_byte(&IDNT[0]);

    UDR=TX_buffer[0]; //Trigger transmtion by loading UDR
    receive_full=0;
}*/
if((send_data==1)&&(fresh_data==1))
{
    //update_values();
    TX_buffer=0;
    //TX_buffer=Received_data;
    UDR=TX_buffer[0]; //Trigger transmtion by loading UDR

    send_data=0;
    fresh_data=0;
}

if(!(ADCSRA&0b01000000)) // An oloklhrw8hke to Conversion xekina ena neo, To ADSC
    adc_routine(); // Jekina neo Conversion

// 3_12_07

if(watchdog_timer<100)
    watchdog_timer++;
if(watchdog_timer>50){ // Mpei edw mesa, den exei labei poly wra dedomena.
    front_left_servo=47; // Kentrare ta servo
    front_right_servo=47;
    back_left_servo=47;
    back_right_servo=46; // Sto 47 frenarei kai mas xalaei ton akolou8o elegxo
    esp_on=0;
    //throttle_servo=47;
    //steer_servo=47;
}

if((front_left_servo==front_left_servo_brake)||
(front_right_servo==front_right_servo_brake)||
(back_left_servo==back_left_servo_brake)||
(back_right_servo==back_right_servo_brake))
    esp_on=1; // An hmaste edw shmainei pws douleyei to ESP;
else
    esp_on=0;
if((throttle_servo<41)&&(throttle_servo>20)&&(esp_on==0)){ // Boh8hse sto frenarisma // Sto 43
    servo_propotion=(43-throttle_servo);

    front_left_servo=front_left_servo_release+servo_propotion;
    front_right_servo=front_right_servo_release-servo_propotion;
    back_left_servo=back_left_servo_release-2*servo_propotion;
    back_right_servo=back_right_servo_release+2*servo_propotion;

    watchdog_timer=45; //
}
if((steer_servo_ser>23)&&(steer_servo_ser<63)&&(steer_servo>36)&&(steer_servo<50)) // An einai ola OK dwse
    steer_servo_out=steer_servo_ser;
else
    steer_servo_out=steer_servo;
// Alliws ayto pou methrses

// 3_12_07

```



```

        //stability();
        //software_servo_int();
        /*
#define front_left_servo_brake 57
#define front_left_servo_release 47
#define front_right_servo_brake 37
#define front_right_servo_release 47
#define back_left_servo_brake 33
#define back_left_servo_release 47
#define back_right_servo_brake 47
#define back_right_servo_release 33
*/

    }

}
//-----main-----

void InitPorts(void){

    // Disable Jtag in order to use PORTC5:2

    MCUCSR|=(1<<JTD); // ATmega 16 sel 59 kai sel 232
    MCUCSR|=(1<<JTD); // ATmega 16 sel 59 kai sel 232

    DDRB&=~((1<<PB7)|(1<<PB6)|(1<<PB5)|(1<<PB4)); // PB4:7 inputs (Hall Effect Sensors)
    DDRB&=~((1<<PB3)|(1<<PB2)); // PB2:3 inputs (SteerIn PB3, Trothle_in
PB2)
    DDRB|=0b00000011; // PB1:0 outputs (SteerOut PB1,
ThottleOut PB0)

    DDRA=0; // PortA input

    DDRD|=(0<<PD0)|(1<<PD1); // PD0 RXD in, PD1 TXDoutcv
    DDRD|=(0<<PD2); // PD2==>External Interrupt 0, in. Duty Cycle
Accelerometer
    DDRD|=(0<<PD3); // PD3==>External Interrupt 1, in. XOR from Hall
Effect

    DDRC&=~((1<<PC6)|(1<<PC7)); // PC6:in, PC7:in;
    DDRC|=(1<<PC5)|(1<<PC4)|(1<<PC3)|(1<<PC2); //PC2:5 out
}

void Timers_Init(void)
{
    // Timer Counter_0 8bit
    TCCR0|=(0<<CS02)|(0<<CS01)|(1<<CS00); //No prescaling Timer prescaler T_overflow=16us
    TIMSK|=(1<<TOIE0); //Timer/Counter 0, overflow enable

    // Timer Counter_2 8bit
    TCCR2|=(0<<CS22)|(1<<CS21)|(0<<CS20); //Timer prescaler=8 T_overflow=128 us
    TIMSK|=(1<<TOIE2); //Timer/Counter 2, overflow enable

}

void USART_Init(unsigned int ubrr)
{
    //Set Baud rate here

```

```

    UBRRH=(unsigned char)(ubrr>>8);
    UBRL=(unsigned char)ubrr;
    // Enable receiver and trasmitter and receiver and transmitter Interrupts
    UCSRB=(1<<RXEN)|(1<<TXEN)|(1<<RXCIE)|(1<<TXCIE);
    //Set frame format; 8 data, 1 stop bit sel 164
    UCSRC=(1<<URSEL)|(0<<USBS)|(1<<UCSZ1)|(1<<UCSZ0);
    //PORTB=~ubrr;

    TX_read=1;
    RX_write=0;
}

void update_values(void)
{
    TX_data_to_send[0]='F';
    TX_data_to_send[1]='L';
    TX_buffer=&TX_data_to_send[0];
    UDR=TX_buffer[0];           //Trigger transimtion by loading UDR
}

void external_interrupts_init(void)
{
    GICR|=(1<<INT0)           ;           //Enable external Interrupt 0
    MCUCR|=(0<<ISC01)|(1<<ISC00);       //Any logical change on Interrupt 0 generates
Interrupt sel 67

    GICR|=(1<<INT1)           ;           //Enable external Interrupt 1
    MCUCR|=(0<<ISC11)|(1<<ISC10);       //Any logical change on Interrupt 0 generates
Interrupt sel 67
}

void adc_init(void){
    ADMUX&=~((1<<REFS0)|(1<<REFS1));    // REFS0=0 kai REFS1=0 ARef,Internal Vref off(sel 215)
    //ADMUX|=((1<<REFS0)|(1<<REFS1));    // REFS0=1 kai REFS1=1 ARef,Internal Vref on(sel
215)
    ADMUX&=~(1<<ADLAR);               // Left Adjust result off

    ADMUX&=ADC0_and;
    ADMUX|=ADC0_or;                   // Start from ADC0 port

    ADCSRA|=(1<<ADIE)|(1<<ADEN);        // Interrupt Enable, ADC Enable
    //ADCSRA|=(0<<ADPS2)|(1<<ADPS1)|(0<<ADPS0);    // ADC prescaler=4 sel 218

    ADCSRA|=(1<<ADPS2)|(1<<ADPS1)|(0<<ADPS0);    // ADC prescaler=64 sel 218

    ADCSRA|=(1<<ADSC);                 // Start Conversion
}

```

9.2.2 adc_accelerometers.c

// Gia ton accelerometer 1.2 g

```
static u8      acc_12_flag=0;
unsigned int   acc_12_X_ticks=0;
unsigned int   acc_12_Y_ticks=0;
unsigned int   acc_12_X=0;
unsigned int   acc_12_Y=0;
```

// Gia na kanw prescaling 2, mia mpainei kai mia oxi
// Metraei to duty cycle

// Analog to Digital Conversion

```
static u8      ADC_flag=0;
// acc+-2g front
static u8 acc_2_front_X_low=0;
static u8 acc_2_front_X_high=0;
static u8 acc_2_front_Y_low=0;
static u8 acc_2_front_Y_high=0;
// acc+-2g back
static u8 acc_2_back_X_low=0;
static u8 acc_2_back_X_high=0;
static u8 acc_2_back_Y_low=0;
static u8 acc_2_back_Y_high=0;
// Pontentiometro (gia gwnia strofhs)
static u8 pont_low=0;
static u8 pont_high=0;
```

// Gia poio port eGINE to teleytaio conversion

// Gyroskopio

```
static u8 gyro_low=0;
static u8      gyro_high=0;
```

// Battery

```
static u8      batt_low=0;
static u8 batt_high=0;
```

```
static u8      temp_u8;
```

// Gia proswrinh xrhsh

// brake servos

```
//static u8      servo_clock;
20ms/128us=156,25 kyklous
static int      servo_clock;
opote 20ms/32us=625 kyklous
static u8      front_left_servo=47;
static u8      front_right_servo=47;
static u8      back_left_servo=47;
static u8      back_right_servo=47;
```

// Me prescaler 8, exoume overflow ka8e 128us, opote

// Me prescaler 0, exoume overflow ka8e 32us(mesw flag),

// Exw kentro me 1.5ms, opote 1.5ms/32us=47 palmous

// Other Servos

```
static u8 throttle_servo=47;
static u8 steer_servo=47;
```

//PB0 Ch2_out, PB2 Ch2_in

//PB1 Ch1_out, PB3 Ch1_in

```
static u8      steer_servo_out=47;
static u8      steer_servo_ser=0;
```

```
static u8 throttle_servo_ticks=47;
static u8 steer_servo_ticks=47;
```

//PB0 Ch2_out, PB2 Ch2_in

//PB1 Ch1_out, PB3 Ch1_in

```
static u8 watchdog_timer=0;
```

// Hall Effect Sensors

```

static int wheel_big_counter=0; // Metraei ta 1024*128us=131072us
static u8 front_left_ticks=0;
static u8 front_right_ticks=0;
static u8 back_left_ticks=0;
static u8 back_right_ticks=0;
static u8 previous_state=0; // Xrhsimopoieitai gia na ypodhlwsei thn prohgoumenh
katastash,high h low

static u8 front_left_speed=0;
static u8 front_right_speed=0;
static u8 back_left_speed=0;
static u8 back_right_speed=0;

void adc_routine(void){
    switch (ADC_flag){
        case 0:
            ADC_flag=1;
            if((TX_read<6)||((TX_read>7))){//if(TX_read!=7){
                acc_2_front_X_low=ADCL;
                acc_2_front_X_high=ADCH;
            }
            else{
                temp_u8=ADCL;
                temp_u8=ADCH;
            }
            ADMUX&=ADC1_and; // Sample ADC1
            ADMUX|=ADC1_or;
            break;
        case 1:
            ADC_flag=2;
            if((TX_read<8)||((TX_read>9))){//if(TX_read!=9){
                acc_2_front_Y_low=ADCL;
                acc_2_front_Y_high=ADCH;
            }
            else{
                temp_u8=ADCL;
                temp_u8=ADCH;
            }
            ADMUX&=ADC2_and;
            ADMUX|=ADC2_or; // Sample ADC2
            break;
        case 2:
            ADC_flag=3; // Sample next
            if((TX_read<10)||((TX_read>11))){//if(TX_read!=11){
                acc_2_back_X_low=ADCL;
                acc_2_back_X_high=ADCH;
            }
            else{
                temp_u8=ADCL;
                temp_u8=ADCH;
            }
            ADMUX&=ADC3_and;
            ADMUX|=ADC3_or; // Sample ADC3
            break;
        case 3:
            ADC_flag=4; // Sample next
            if((TX_read<12)||((TX_read>13))){//if(TX_read!=13){
                acc_2_back_Y_low=ADCL;
                acc_2_back_Y_high=ADCH;
            }
    }
}

```

```

        else{
            temp_u8=ADCL;
            temp_u8=ADCH;
        }
        ADMUX&=ADC4_and;
        ADMUX|=ADC4_or; // Sample ADC4

    case 4:
        ADC_flag=5;
        if((TX_read<18)||(TX_read>19)){//if(TX_read!=19){
            pont_low=ADCL;
            pont_high=ADCH;
        }
        else{
            temp_u8=ADCL;
            temp_u8=ADCH;
        }
        ADMUX&=ADC5_and;
        ADMUX|=ADC5_or; // Sample ADC5
        break;

    case 5:
        ADC_flag=6;
        //cli();
        if((TX_read<20)||(TX_read>21)){
            gyro_low=ADCL;
            gyro_high=ADCH;
            //sei();
        }
        else{
            //sei();
            temp_u8=ADCL;
            temp_u8=ADCH;
        }
        ADMUX&=ADC6_and;
        ADMUX|=ADC6_or; // Sample ADC6
        break;

    case 6:
        ADC_flag=0;
        if(TX_read!=23){
            batt_low=ADCL;
            batt_high=ADCH;
        }
        else{
            temp_u8=ADCL;
            temp_u8=ADCH;
        }
        ADMUX&=ADC0_and;
        ADMUX|=ADC0_or; // Sample ADC0
        break;

    default:
        ADC_flag=0;

        ADMUX&=ADC0_and;
        ADMUX|=ADC0_or;
        break;
}
ADCSRA|=(1<<ADSC); // Start New Conversion
}

```

9.2.3 interrupt_routines.c

```
// -----Interrupt routines----- start
/* USART, Rx Complete */
SIGNAL(SIG_UART_RECV)          /* USART, Rx Complete */
{
    if (RX_write==(RX_buffer_length-1)){
        RX_buffer[RX_write]=UDR;
        Received_data=&RX_buffer[0];
        fresh_data=1;
        RX_write=0;
        receive_full=1;
        //Mhdenise ton deikth
        //Received buffer full, start transmitting
    }
    else{
        RX_buffer[RX_write]=UDR;
        switch (RX_write){
            case 1:
                front_left_servo=UDR;
                break;
            case 2:
                front_right_servo=UDR;
                break;
            case 3:
                back_left_servo=UDR;
                break;
            case 4:
                back_right_servo=UDR;
                break;
            /*
            case 5:
                throttle_servo=UDR;
                break;
                */
            case 6:
                steer_servo_ser=UDR;
                break;
            default:
                break;
        }
        RX_write++;
        watchdog_timer=0;
        if(UDR==255){
            RX_write=0;
            //Increase pointer
            // Xrhsimopoeitai ws watchdog
            //ascii(255) Terminator
            //Mhdenise ton deikth
        }
    }
}

/* USART, Tx Complete */
SIGNAL(SIG_UART_TRANS)        /* USART, Tx Complete */
{
    if (TX_read<=(TX_buffer_length-1))
        //Check if more bytes are available
    {
        switch (TX_read)
        {
            case 1:
                UDR='S';
                break;
            case 2:
                // Higher Nibble First
                temp_u8=(acc_12_X>>8); // Right shift 8
                if (temp_u8==0)
                    temp_u8=254;
                // An einai mhden kanto 254
        }
    }
}
```

```

        UDR=temp_u8;
        break;
case 3:
    temp_u8=acc_12_X;           // Fortwnei to dexi tw n dyo char
    if (temp_u8==255)
        temp_u8--;           // An einai o terminator meiwse to
    UDR=temp_u8;
    break;
case 4:
    // Higher Nibble First
    temp_u8=(acc_12_Y>>8); // Right shift 8
    if (temp_u8==0)
        temp_u8=254;           // An einai mhden kanto 254
    UDR=temp_u8;
    break;
case 5:
    temp_u8=acc_12_Y;           // Fortwnei to dexi tw n dyo char
    if (temp_u8==255)
        temp_u8--;           // An einai o terminator meiwse to
    UDR=temp_u8;
    break;
/// Acc +-2g front Accelerometer
// Epeidh stelnei low kai high nibbles se jexwrista bytes kai endexetai na allaxoun xrhsimopoioume
// to TX_read ws flag gia na mhn allajei tis an exei fygei to ena apo ta dyo nibbles
case 6:
    if(acc_2_front_X_high==0)
        UDR=254;
    else
        UDR=acc_2_front_X_high;
    break;
case 7:
    if (acc_2_front_X_low==255)
        acc_2_front_X_low--;
    UDR=acc_2_front_X_low;
    break;
case 8:
    if(acc_2_front_Y_high==0)
        UDR=254;
    else
        UDR=acc_2_front_Y_high;
    break;
case 9:
    if (acc_2_front_Y_low==255)
        acc_2_front_Y_low--;
    UDR=acc_2_front_Y_low;
    break;
/// Acc +-2g back Accelerometer
case 10:
    if(acc_2_back_X_high==0)
        UDR=254;
    else
        UDR=acc_2_back_X_high;
    break;
case 11:
    if (acc_2_back_X_low==255)
        acc_2_back_X_low--;
    UDR=acc_2_back_X_low;
    break;
case 12:
    if(acc_2_back_Y_high==0)
        UDR=254;
    else
        UDR=acc_2_back_Y_high;
    break;

```

```

case 13:
    if (acc_2_back_Y_low==255)
        acc_2_back_Y_low--;
        UDR=acc_2_back_Y_low;
        break;
// An einai o terminator meiwse to

case 14:
    if (front_left_speed==0)
        front_left_speed=254;
        UDR=front_left_speed;
        break;
// An einai o terminator meiwse to

case 15:
    if (front_right_speed==0)
        front_right_speed=254;
        UDR=front_right_speed;
        break;
// An einai 0 diabazetai ws EOF
// Allaje to se 254 opou den tha ginei pote

case 16:
    if (back_left_speed==0)
        back_left_speed=254;
        UDR=back_left_speed;
        break;
// An einai 0 diabazetai ws EOF
// Allaje to se 254 opou den tha ginei pote

case 17:
    if (back_right_speed==0)
        back_right_speed=254;
        UDR=back_right_speed;
        break;
// An einai o terminator meiwse to

case 18:
    if(pont_high==0)
        UDR=254;
    else
        UDR=pont_high;
    break;

case 19:
    if (pont_low==255)
        pont_low--;
        UDR=pont_low;
        break;
// An einai o terminator meiwse to

case 20:
    if(gyro_high==0)
        gyro_high=254;
    else
        UDR=gyro_high;
    break;

case 21:
    if (gyro_low==255)
        gyro_low--;
        UDR=gyro_low;
        break;
// An einai o terminator meiwse to

case 22:
    if(batt_high==0)
        UDR=254;
    else
        UDR=batt_high;
    break;

case 23:
    if (batt_low==255)
        batt_low--;
        UDR=batt_low;
        break;
// An einai o terminator meiwse to

case 24:
    if (steer_servo==0)
        steer_servo=254;
        UDR=steer_servo;
// An einai o terminator meiwse to

```



```

        break;
    case 25:
        if (throttle_servo==0)
            throttle_servo=254;
        UDR=throttle_servo;
        break;
    default:
        if(TX_read!=(TX_buffer_length-1)){
            //UDR=TX_buffer[TX_read];
            if((number>='0')&& ( number<='9')){
                UDR=number ;
                number++;
            }
            else{
                UDR='0';
                number='0';
            }
        }
        else{
            UDR=255;
        }
        break; //Default
    } // End switch

    TX_read++;
}
else{
    TX_read=1;
    send_data=1;
}

}

/* Timer/Counter0 Overflow */
SIGNAL(TIMER0_OVF_vect){
    if(acc_12_X_ticks>0)
        acc_12_X_ticks++;
    if(acc_12_Y_ticks>0)
        acc_12_Y_ticks++;

    if(acc_12_flag==1){
        if(servo_clock==0){
            PORTC|=(1<<PC5)|(1<<PC4)|(1<<PC3)|(1<<PC2); // PC5:2 high
            PORTB|=(1<<PB1)|(1<<PB0); //PB1:0 high
            servo_clock++;
        }
        else{
            if(servo_clock==front_left_servo)
                PORTC&=~(1<<PC5); // PC5 low
            if(servo_clock==front_right_servo)
                PORTC&=~(1<<PC4); // PC4 low
            if(servo_clock==back_left_servo)
                PORTC&=~(1<<PC3); // PC3 low
            if(servo_clock==back_right_servo)
                PORTC&=~(1<<PC2); // PC2 low
            if(servo_clock==steer_servo_out)
                PORTB&=~(1<<PB0); // PB1 low
            if(servo_clock==throttle_servo)

```

```

PORTB&=~(1<<PB1); // PB0 low

servo_clock++;
if (servo_clock>625) //156 // An perasei to 156, tote oloklhrwse ton
{
    servo_clock=0;
}
} // if(acc_12_flag==0){

// Epomeno if
if (acc_12_flag==0){ //Tha mpainei edw kathe 32us

    acc_12_flag=1; //Mia mpainei kai mia oxi

    ////////////Pros8eto//////////
    ///-----
    if((PINB&0b00001000)) //PB2

Throttle_in High Trigger Counting
    throttle_servo_ticks++;
else{
    if(throttle_servo_ticks>0){ //PB2 Throttle_in

Low Stop Counting
        throttle_servo=throttle_servo_ticks;
        throttle_servo_ticks=0;
        if((throttle_servo<20)||((throttle_servo>90))){

            steer_servo=throttle_servo=47;
// Phgaine ta sto kentro // Xasame to shma, opote blepei mhden
        }
    }
    ///-----
    if((PINB&0b00000100)) //PB3

Steer_in High Trigger Counting
    steer_servo_ticks++;
else{
    if(steer_servo_ticks>0){ //PB3 Steer_in

Low Stop Counting
        steer_servo=steer_servo_ticks;
        steer_servo_ticks=0;
        if((steer_servo<20)||((steer_servo>90))){

            steer_servo=throttle_servo=47;
// Phgaine ta sto kentro // Xasame to shma, opote blepei mhden
        }
    }
    ///-----
    ////////////Pros8eto//////////

}
else
    acc_12_flag=0;
}

/* Timer/Counter2 Overflow */
SIGNAL(TIMER2_OVF_vect){ // Exoume Overflow Ka8e 128us

    wheel_big_counter++;
    if(wheel_big_counter>781){ // An mpei edw exoune perasei
781*128us=100000uSec=0.1sec

```

```

        front_left_speed=front_left_ticks;
1562*128us=200000usec)
        front_right_speed=front_right_ticks;
        back_left_speed=back_left_ticks;
        back_right_speed=back_right_ticks;
        wheel_big_counter=0;
        front_left_ticks=front_right_ticks=back_left_ticks=back_right_ticks=0;
    }

}

// External Interrupt Request 1, +1.2 Accelerometer
SIGNAL(INT1_vect)
{
    temp=PINC;
    if (temp&0b01000000){
        if(acc_12_X_ticks==0)
            acc_12_X_ticks++;
    }
    else{
        if(acc_12_X_ticks>0){
            if(TX_read!=3)
                acc_12_X=(acc_12_X_ticks-1);
            acc_12_X_ticks=0;
        }
    }
    if (temp&0b10000000){
        if(acc_12_Y_ticks==0)
            acc_12_Y_ticks++;
    }
    else{
        if(acc_12_Y_ticks>0){
            if(TX_read!=5)
                acc_12_Y=(acc_12_Y_ticks-1);
            acc_12_Y_ticks=0;
        }
    }
}

// External Interrupt Request 0 Wheel speeds
SIGNAL(INT0_vect)
{
    temp=PINB;
    if (temp&0b00010000){
        if ((previous_state&0b00010000)==0)
            front_right_ticks++;
    }
    if (temp&0b00100000){
        if ((previous_state&0b00100000)==0)
            front_left_ticks++;
    }
    if (temp&0b01000000){

```

// Pio prin (An mpei edw exoune perasei)

// Start Counting Again

//PC6 X axis (temp&0b01000000)

// Trigger Counting

// An exei steilei to high char mhn to allaxeis

//PC7 Y axis (temp&0b10000000)

// Trigger Counting

// An exei steilei to high nibble mhn to

allaxeis

// An einai high koita to previous state

// An isxyei exei allaxeis to state tou PB4

// An einai high koita to previous state

// An isxyei exei allaxeis to state tou PB5

// An einai high koita to previous state

```

        if((previous_state&0b01000000)==0)                // An isxyei exei allaxeï to state tou PB6
            back_right_ticks++;
    }

    if(temp&0b10000000){
        if((previous_state&0b10000000)==0)                // An einai high koita to previous state
            back_left_ticks++;                             // An isxyei exeï allaxeï to state tou
PB7
    }

    previous_state=temp;
}

// ADC Conversion Complete
SIGNAL(ADC_vect){
}

// Default interrupt vector to avoid reset
//http://hubbard.engr.scu.edu/embedded/avr/doc/avr-libc/avr-libc-user-manual/index.html
SIGNAL(__vector_default){
    // user code here
}

// -----Interrupt routines----- end

```

9.2.4 servos.c

```

// Definition gia tis 8seis twñ servo!

#define front_left_servo_brake 57
#define front_left_servo_release 47
#define front_right_servo_brake 37
#define front_right_servo_release 47
#define back_left_servo_brake 33      // Ayto einai to pisw servo pou douleyei
#define back_left_servo_release 47
#define back_right_servo_brake 47     // Ayto einai to asyndeto servo
#define back_right_servo_release 33

```

9.2.5 header.h

```

typedef unsigned char u8;
typedef unsigned int u16;
typedef unsigned long u32;

#define FOSC 16000000 //Clock speed
#define BAUD 230400 //38400 //57600 //115200
#define MYUBRR 8 //FOSC/16/BAUD-1 //8

#define TX_buffer_length 32
#define TX_buffer_mask 31 //Xrhsimopieitai gia na mhn ginetai out of bounds buffer
#define RX_buffer_length 32
#define RX_buffer_mask 31 //Xrhsimopieitai gia na mhn ginetai out of bounds buffer

// ADC Masks (sel 216) Ayta tha ginontai Bit -and me ton ADMUX gia thn epilogh tou channel

```

```
#define ADC0_and 0b11100000
#define ADC0_or 0
#define ADC1_and 0b11100001
#define ADC1_or 1
#define ADC2_and 0b11100010
#define ADC2_or 2
#define ADC3_and 0b11100011
#define ADC3_or 3
#define ADC4_and 0b11100100
#define ADC4_or 4
#define ADC5_and 0b11100101
#define ADC5_or 5
#define ADC6_and 0b11100110
#define ADC6_or 6
#define ADC7_and 0b11100111
#define ADC7_or 7
```

9.3.1 plotdata

```
FLB=data(:,20);
FRB=data(:,21);
BLB=data(:,22);
BRB=data(:,23);
Brake_command=data(:,24);
samples=data(:,25);
% to 23 to afhnw
time=data(:,27)/1000;
hour=data(:,28);
min=data(:,29);
sec=data(:,30);
usec=data(:,31);
velocity=(fl+fr+bl+br)/4;
wheelbase=0.54;
mass=11;

%range=1835:2200;
% range=934:1500;
```

```

% range=2:3000;%3000;
% range=300:1000;
%

% range=2021:3000; % kalo gia plotdata(logmath_23_2_cropped,0.004);

% range=2100:3000; % kalo gia plotdata(logmath_23_2_cropped,0.004);
% range=1:700; %plotdata(logmath_23_2_crop,0.004);
%range=1:900; %plotdata(log_25part_1,0.004);
% range=1050:1850; %plotdata(log_25_part2,0.004);
range=1050:1780; %plotdata(log_25_part2,0.004);
timeplot=time(range)-time(range(1));
% Normalize time
% 50954
% stoixeio 8357
for i=2:length(timeplot);
    timeplot(i)=timeplot(i-1)+8000;
end
timeplot=timeplot/1000000;
step=7;
fl(range)=exomalyne(fl(range),step);
fr(range)=exomalyne(fr(range),step);
bl(range)=exomalyne(bl(range),step);
br(range)=exomalyne(br(range),step);

velocity=(fl+fr+bl+br)/4;

for i=1:length(steer)
    yaw_rate_des(i)=(velocity(i)*steer(i))/(wheelbase+(mass*velocity(i)^2*k/(2*wheelbase)));
end

h1=subplot(3,1,1); plot(timeplot,gyro(range),'-r*',timeplot,yaw_rate_des(range),'-bx');
xlabel('Time (sec)')
title('ADXRS300 Gyroscope')
legend('Gyro (deg/sec)', 'Yaw Rate Des(deg/sec)')
h2=subplot(3,1,2); plot(timeplot,steer(range),'--m. ');
xlabel('Time (sec)')
title('Steering Angle')
legend('Steering Angle (degrees)')
h3=subplot(3,1,3); plot(timeplot,fl(range),'-r.',timeplot,fr(range),'-go',timeplot,bl(range),'-bx',timeplot,br(range),'-c+',timeplot,velocity(range),'-m. ');
title('Wheel Speed (m/sec)')
xlabel('Time (sec)')
legend('Front Left','Front Right','Back Left','BackRight','Velocity')

axis([h1 h2 h3] , 'tight')
set(h1,'nextplot','replacechildren');
set(h2,'nextplot','replacechildren');
set(h3,'nextplot','replacechildren');

brake_FLB=[];
brake_FRB=[];
brake_BLB=[];
brake_BRB=[];
brake_decision_m=[];
textaki=[];
for J=1:length(range);

    h1=subplot(3,1,1);plot(timeplot(1:J),gyro(range(1:J)),'-r*',timeplot(1:J),yaw_rate_des(range(1:J)),'-bx');
    if (Brake_command(range(J))==1)

```

```

temp=length(brake_decision_m);
brake_decision_m(temp+1)=J;
textaki='O';%'\uparrow O';
elseif (Brake_command(range(J))==2)
temp=length(brake_decision_m);
brake_decision_m(temp+1)=J;
textaki='U';%'\downarrow U';
end

text(timeplot(brake_decision_m),gyro(range(brake_decision_m)),textaki,'color','green','HorizontalAlignment','left','FontWeight','bold','FontSize',9);

h2=subplot(3,1,2); plot(timeplot(1:J),steer(range(1:J)),'--m. ');
h3=subplot(3,1,3); plot(timeplot(1:J),fl(range(1:J)),'-r.',timeplot(1:J),fr(range(1:J)),'-g.',timeplot(1:J),bl(range(1:J)),'-b.',timeplot(1:J),br(range(1:J)),'-c.',timeplot(1:J),velocity(range(1:J)),'-m. ')

if (FLB(range(J))==57)
temp=length(brake_FLB);
brake_FLB(temp+1)=J;
end
if (FRB(range(J))==37)
temp=length(brake_FRB);
brake_FRB(temp+1)=J;
end
if (BLB(range(J))==33)
temp=length(brake_BLB);
brake_BLB(temp+1)=J;
end
if (BRB(range(J))==47)
temp=length(brake_BRB);
brake_BRB(temp+1)=J;
end

text(timeplot(brake_FLB),fl(range(brake_FLB)),'\uparrow','HorizontalAlignment','left','color','red','FontWeight','bold','FontSize',12);
text(timeplot(brake_FRB),fr(range(brake_FRB)),'\uparrow','HorizontalAlignment','left','color','green','FontWeight','bold','FontSize',12);
;
text(timeplot(brake_BLB),bl(range(brake_BLB)),'\uparrow','HorizontalAlignment','left','color','black','FontWeight','bold','FontSize',12);
;
text(timeplot(brake_BRB),br(range(brake_BRB)),'\uparrow','HorizontalAlignment','left','color','cyan','FontWeight','bold','FontSize',12);
;

F(J)=getframe(gca,[-30; -30; 1100; 670]);

end

movie(F,1);
movie2avi(F,'test1');

function out=exomalyne(in,step)
out=in;
for i=1:(length(in)-step)
out(i)=sum(in(i:i+step))/step;
end

```


9.3.2 gwnies

```
function [p1 p2]=gwnies28_3_07

%
% xn=[1.924 1.943 1.953 1.987 2.04 2.051 2.090 2.104 2.104 2.139 2.163 2.183 2.212 2.246 2.266 2.3];
% yn=[21 19 18 14.5 10.5 8 4 0 0 -3 -5 -8 -10 -12.5 -14.5 -19];

format long
xn1=[1.924 1.943 1.953 1.987 2.04 2.051 2.090 2.104];
yn1=[21 19 18 14.5 10.5 8 4 0];
xn2=[ 2.104 2.139 2.163 2.183 2.212 2.246 2.266 2.3];
yn2=[0 -3 -5 -8 -10 -12.5 -14.5 -19 ];

p1 = polyfit(xn1,yn1,3);
p2 = polyfit(xn2,yn2,3);

y1 = polyval(p1,1.924:0.01:2.104);
y2 = polyval(p2,2.104:0.01:2.3);

plot(1.924:0.01:2.104,y1,2.104:0.01:2.3,y2,xn1,yn1,xn2,yn2);

xlabel('Voltage (vots)')
ylabel('Steering Angle(deg)')
title('Steering Angle Estimation')
legend('Estimated Angle(right)','Estimated Angle(left)','Real Angle(right)','Real Angle(left)')
grid

% est=polyval(p,[1.924 1.943 1.953 1.987 2.04 2.051 2.090 2.104 2.104 2.139 2.163 2.183 2.212 2.246 2.266 2.3]);
% polyval(p,2.104)
```

9.3.3 Bird's eye view

```
function birdeyview_final(data); % 13-7-2008

% Shmeiwsh, exei diafora, edw einai h taxythta apo tous 4 troxous

% sprintf(big_buffer_math,"%1.3f\t%1.3f\t%1.3f\t%1.3f\t%1.3f\t%1.3f\t%2.2f\t%2.2f\t%2.2f\t%2.2f\t%3.0f\t%2.2f\n");
    %d      %t%4.4f %t%4.4f %t%4.4f %t%d     %d       %t%d          %t%d   %t%d   %t%d   %t%d         %t%d        %t%d
    %t%d    %t%d %t%d %t%d %t%d
    \n",acc_12_X_gs,acc_12_Y_gs,acc_2_front_X_gs,acc_2_front_Y_gs,acc_2_back_X_gs,acc_2_back_Y_gs,front_left_speed,fr
ont_right_speed,back_left_speed,back_right_speed,gryo_degs_ana_sec,steer,bill_local_command,stab_double_parameters[0],s
tab_double_parameters[1],stab_double_parameters[2],auto_trig:auto_speed:auto_steel,stab_commands[2],stab_commands[3],s
tab_commands[4],stab_commands[5],bill_brake,deigmata,time_passed,(time_passed_sum/1000),hour,min,sec,usec);
% // MX tMY tFX tFY tBX tBY tFLS tFRS tBLS tBRS tGyro tSteer tMethod
\SDP[0] \SDP[1] \SDP[2]\auto_trig \auto_speed \auto_steel \FLB \FRB \RLB \BRB \tbill_brake \tSamples
\Time_passed: \TimeSum \tHour \min \tsec \tusec
\n",acc_12_X_gs,acc_12_Y_gs,acc_2_front_X_gs,acc_2_front_Y_gs,acc_2_back_X_gs,acc_2_back_Y_gs,front_left_speed,fr
ont_right_speed,back_left_speed,back_right_speed,gryo_degs_ana_sec,steer,bill_local_command,stab_double_parameters[0],s
tab_double_parameters[1],stab_double_parameters[2],auto_trig:auto_speed:auto_steel,stab_commands[2],stab_commands[3],s
tab_commands[4],stab_commands[5],bill_brake,deigmata,time_passed,(time_passed_sum/1000),hour,min,sec,usec);

compensate_range=470:530; % Apo edw exoume mhdenikes taxythtes opote mporoume na dior8wsoume tis times tw n
epitaxynsimetrwn
```

```

k=0.004;

enadyox=data(:,1);
enadyoy=data(:,2);
enadyoxcon=sum(enadyox(compensate_range))/length(compensate_range);
enadyoycon=sum(enadyoy(compensate_range))/length(compensate_range);
enadyox=enadyox-enadyoxcon;
enadyoy=enadyoy-enadyoycon;

dyofx=data(:,3);
dyofy=data(:,4);
dyofxcon=sum(dyofx(compensate_range))/length(compensate_range);
dyofycon=sum(dyofy(compensate_range))/length(compensate_range);
dyofx=dyofx-dyofxcon;
dyofy=dyofy-dyofycon;

dyobx=data(:,5);
dyoby=data(:,6);
dyobxcon=sum(dyobx(compensate_range))/length(compensate_range);
dyobycon=sum(dyoby(compensate_range))/length(compensate_range);
dyobx=dyobx-dyobxcon;
dyoby=dyoby-dyobycon;

fl=data(:,7);
fr=data(:,8);
bl=data(:,9);
br=data(:,10);

gyro=data(:,11);
steer=data(:,12);

method=data(1,13);    % 1: single accelerometer ESC

SDP0=data(:,14);    % Sensitivity
SDP1=data(:,15);    % Understeer Gradient
SDP2=data(:,16);

auto_trig=data(:,17); % An einai 1 exei klh8ei h auto steer
auto_speed=data(:,18); % Poso na ftasei gia na stripsei
auto_steer=data(:,19); % Poso na stripsei

FLB=data(:,20);
FRB=data(:,21);
BLB=data(:,22);
BRB=data(:,23);

Brake_command=data(:,24);
samples=data(:,25);
% to 23 to afhnw
time=data(:,27)/1000;
hour=data(:,28);
min=data(:,29);
sec=data(:,30);
usec=data(:,31);
velocity=(fl+fr+bl+br)/4;
wheelbase=0.54;
mass=11;

```

```
% range=1:650; %plotdata_auto(log25_2_08_auto_ESP_off,0.004);
% range=1:620; %plotdata_auto(log_25_2_esp_off_auto,0.004); kalo ayto
%plotdata_auto(log_25_2_auto_ESP_on,0.004); kai ayto kalo!!! gia esp on Apo to 470-530 deigma kai meta einai mhdenikes
oi taxythtes, opote mporoume na kanoume compensate ta accelerometers
```

```
range=1:620; %plotdata_auto(log_25part,0.004)
```

```
timeplot=time(range)-time(range(1));
```

```
dt=8000;
```

```
% Normalize time
```

```
for i=2:length(timeplot);
```

```
    timeplot(i)=timeplot(i-1)+dt; %dt=8000;
```

```
end
```

```
timeplot=timeplot/1000000;
```

```
step=7;
```

```
fl(range)=exomalyne(fl(range),step);
```

```
fr(range)=exomalyne(fr(range),step);
```

```
bl(range)=exomalyne(bl(range),step);
```

```
br(range)=exomalyne(br(range),step);
```

```
velocity=(fl+fr+bl+br)/4;
```

```
for i=1:length(steer)
```

```
    yaw_rate_des(i)=(velocity(i)*steer(i))/(wheelbase+(mass*velocity(i)^2*k/(2*wheelbase)));
```

```
end
```

```
steer_rad=steer(range)*pi/180; % Convert to radians
```

```
gyro_rad=gyro(range)*pi/180; % Convert to radians/sec
```

```
psi_rad(1)=0; % Arxikh Gwnia =0 rad
```

```
pos_x(1)=0;
```

```
pos_y(1)=0;
```

```
for i=2:length(gyro_rad)
```

```
    psi_rad(i)=psi_rad(i-1)+((gyro_rad(i)*dt)/1000000);
```

```
end
```

```
vel=bl(range)+br(range)+((fr(range)+fr(range)).*cos(steer_rad)); % Shmeio pros shmeio
```

```
vel=vel/4;
```

```
for i=2:length(gyro_rad)
```

```
    pos_x(i)=pos_x(i-1)+vel(i)*cos(psi_rad(i))*dt/1000000;
```

```
    pos_y(i)=pos_y(i-1)+vel(i)*sin(psi_rad(i))*dt/1000000;
```

```
end
```

```
% To akolou8o ta kanoume gia na kanoume hold ta plots
```

```
plot(pos_y,pos_x,'m-',0,0,'b-');
```

```
for i=1:length(gyro_rad)
```

```
    if(mod(i,20)==0)
```

```
        temp=sprintf('t=%1.1f',timeplot(i));
```

```
        text(pos_y(i),pos_x(i),temp,'FontSize',8);
```

```
        %text(pos_y(i),pos_x(i),'\uparrow','FontSize',14,'Rotation',-psi_rad(i)*180/pi,'color','red');
```

```
        text(pos_y(i),pos_x(i),'\rightarrow','FontSize',20,'Rotation',-psi_rad(i)*180/pi+90,'color','red');
```

```
    end
```

```
end
```

```
% 'linestyle','-','EdgeColor','red','LineWidth',1
```

```
% psi_rad*180/pi sthn abaparastash sto grafhma 8elei -psi_rad*180/pi
```

```

% Prwta 8a kanoume plot me to ESP of kai meta to me hold to auto
ylabel('Y (m)')
xlabel('X (m)')
title('Bird's Eye View')
sensitivity=data(1,14);
temp=sprintf('Vehicle's Trajectory\nSensitivity:%1.1f',sensitivity);
% legend('Vehicle's Trajectory')
temp1=sprintf('Vehicle's Trajectory\nSensitivity:0.9');
legend(temp,temp1);

brake_FLB=[];
brake_FRB=[];
brake_BLB=[];
brake_BRB=[];
brake_decision_m=[];
textaki=[];

brake_FLB=find(FLB(range)==57);
brake_FRB=find(FRB(range)==37);
brake_BLB=find(BLB(range)==33);
brake_BRB=find(BRB(range)==47);
temp=sprintf('Brake: \n FL:Front Right\nFR:Front Right\nBL:Back Left\nBR:Back Right\n');

% chi=-4.8; %on
chi=-2; % off
psi=0.5;
text(chi,psi+0.45,'Front Left:\clubsuit','color','red')
text(chi,psi+0.3,'Front Right:\heartsuit','color','green')
text(chi,psi+0.15,'Back Left:\diamondsuit','color','black')
text(chi,psi,'Back Right:\spadesuit','color','cyan')%,'linestyle','-','EdgeColor','red','LineWidth',2)

aktina=0.1;
text(pos_y(brake_FLB)-sin(-psi_rad(brake_FLB)+pi/4)*aktina,pos_x(brake_FLB)+cos(-
psi_rad(brake_FLB)+pi/4)*aktina,'\clubsuit','HorizontalAlignment','left','color','red','FontWeight','bold','FontSize',6);
text(pos_y(brake_FRB)-sin(-psi_rad(brake_FRB)-pi/4)*aktina,pos_x(brake_FRB)+cos(-psi_rad(brake_FRB)-
pi/4)*aktina,'\heartsuit','HorizontalAlignment','left','color','green','FontWeight','bold','FontSize',6); % Swsto gia front right
text(pos_y(brake_BLB)+sin(-psi_rad(brake_BLB)-pi/4)*aktina,pos_x(brake_BLB)-cos(-psi_rad(brake_BLB)-
pi/4)*aktina,'\diamondsuit','HorizontalAlignment','left','color','black','FontWeight','bold','FontSize',6);% Swsto gia back left
text(pos_y(brake_BRB)+sin(-psi_rad(brake_BRB)+pi/4)*aktina,pos_x(brake_BRB)-cos(-
psi_rad(brake_BRB)+pi/4)*aktina,'\spadesuit','HorizontalAlignment','left','color','cyan','FontWeight','bold','FontSize',6);

grid

function out=exomalyne(in,step)
    out=in;
    for i=1:(length(in)-step)
        out(i)=sum(in(i:i+step))/step;
    end

```

9.4 Typical Log file output

02-25-2008 09:54:35.767807													
Start writing Again													
0.098	-0.010	-0.044	-0.059	0.047	-0.028	0.00	0.00	0.00	0.00	0	-23.89	1	0.9000
	0.0040	0.0000	0		1				55			47	47
	47	33	0	1	46591		113322		9	54	35	778116	
0.095	-0.011	-0.051	-0.060	0.046	-0.029	0.00	0.00	0.00	0.00	-1	-23.86	1	0.9000
	0.0040	0.0000	0		1				55			47	47
	47	33	0	2	10546		113332		9	54	35	778455	
0.100	-0.009	-0.041	-0.054	0.046	-0.029	0.00	0.00	0.00	0.00	-1	-22.84	1	0.9000
	0.0040	0.0000	0		1				55			47	47
	47	33	0	3	206		113332		9	54	35	778639	
0.101	-0.006	-0.045	-0.059	0.046	-0.030	0.00	0.00	0.00	0.00	-1	-23.89	1	0.9000
	0.0040	0.0000	0		1				55			47	47
	47	33	0	4	172		113333		9	54	35	778807	
0.093	-0.014	-0.046	-0.060	0.047	-0.030	0.00	0.00	0.00	0.00	-2	-23.56	1	0.9000
	0.0040	0.0000	0		1				55			47	47
	47	33	0	5	166		113333		9	54	35	778971	
0.098	-0.009	-0.046	-0.054	0.053	-0.030	0.00	0.00	0.00	0.00	-1	-23.93	1	0.9000
	0.0040	0.0000	0		1				55			47	47
	47	33	0	6	162		113333		9	54	35	779164	
0.101	-0.006	-0.046	-0.046	0.049	-0.031	0.00	0.00	0.00	0.00	-2	-23.93	1	0.9000
	0.0040	0.0000	0		1				55			47	47
	47	33	0	7	192		113333		9	54	35	779329	
0.101	-0.005	-0.046	-0.050	0.055	-0.012	0.00	0.00	0.00	0.00	-2	-23.88	1	0.9000
	0.0040	0.0000	0		1				55			47	47
	47	33	0	8	1961		113335		9	54	35	781355	
0.097	-0.010	-0.046	-0.049	0.044	-0.025	0.00	0.00	0.00	0.00	0	-23.22	1	0.9000
	0.0040	0.0000	0		1				55			47	47
	47	33	0	9	7963		113343		9	54	35	789379	
0.102	-0.005	-0.047	-0.042	0.060	-0.027	0.00	0.00	0.00	0.00	-1	-23.93	1	0.9000
	0.0040	0.0000	0		1				55			47	47
	47	33	0	10	53053		113396		9	54	35	842461	
0.103	-0.006	-0.047	-0.052	0.052	-0.026	0.00	0.00	0.00	0.00	-2	-23.75	1	0.9000
	0.0040	0.0000	0		1				55			47	47
	47	33	0	11	437		113396		9	54	35	842775	
0.103	-0.007	-0.047	-0.054	0.056	-0.028	0.00	0.00	0.00	0.00	-2	-23.55	1	0.9000
	0.0040	0.0000	0		1				55			47	47
	47	33	0	12	213		113397		9	54	35	842974	
0.107	-0.004	-0.053	-0.049	0.046	-0.029	0.00	0.00	0.00	0.00	0	-23.93	1	0.9000
	0.0040	0.0000	0		1				55			47	47
	47	33	0	13	183		113397		9	54	35	843156	
0.108	-0.003	-0.049	-0.043	0.061	-0.029	0.00	0.00	0.00	0.00	-1	-23.36	1	0.9000
	0.0040	0.0000	0		1				55			47	47
	47	33	0	14	181		113397		9	54	35	843331	
0.098	-0.011	-0.042	-0.041	0.059	-0.030	0.00	0.00	0.00	0.00	-1	-23.87	1	
	0.9000	0.0040	0.0000										

The end

The end!