Πανεπιστήμιο Θεσσαλίας

Τμήμα Μηχανικών Η/Υ, Τηλεπικοινωνιών και Δικτύων

Διπλωματική εργασία με τίτλο :

Δυναμική συλλογή και ανάλυση δεδομένων διαγνωστικού ελέγχου αυτοκινήτου (OBD) | Dynamic Collection and Analysis of On-Board-Diagnostics (OBD) data

εκπόνηση :

Στέφανος Αϊβαλής

επιβλέπουσα καθηγήτρια :

Ασπασία Δασκαλοπούλου

Βόλος, καλοκαίρι 2012

Ευχαριστίες

Ένα πολύ μεγάλο ευχαριστώ στην οικογένειά μου για την υποστήριξή της όλα αυτά τα χρόνια. Οι λέξεις δεν μπορούν να περιγράψουν πόσο πολύτιμη βοήθεια και συμπαράσταση μου προσέφερε.

Επίσης, ευχαριστώ και τη κ. Δασκαλοπούλου για την εμπιστοσύνη που μου έδειξε και για τη λογικότητα που τη διακρίνει.

Στέφανος Αϊβαλής

καλοκαίρι 2012

ΓΕΝΙΚΗ ΠΕΡΙΓΡΑΦΗ

Ο τομέας της μηχανοκίνησης, όχι απαραίτητα του μηχανοκίνητου αθλητισμού, συνάρπαζε και συνεχίσει να συναρπάζει εκατομμύρια ανθρώπους ανά το κόσμο. Οι λειτουργίες του κινητήρα, των εμβόλων και των μηχανικών εξαρτημάτων, οι πρόσθετες βελτιώσεις που μπορεί να κάνει κάποιος σχεδόν σε οποιοδήποτε σημείο του οχήματός του και άλλα πολλά ελκύουν το ενδιαφέρον κυρίως των αντρών για να είμαστε ειλικρινείς.

Τις τελευταίες δεκαετίες χρησιμοποιείται όλο και πιο συχνά αλλά και πιο αποτελεσματικά ο Η/Υ στον τομέα της μηχανοκίνησης. Αφορμή για να γίνει αυτό ήταν η επιθυμία των μηχανικών αυτοκινήτων WRC (World Rally Championship) αλλά και Formula 1 να είναι ανταγωνιστικές οι ομάδες που εκπροσωπούσαν στα αντίστοιχα πρωταθλήματα.

Πλέον, σχεδόν κάθε εμπορικό αυτοκίνητο είναι εξοπλισμένο με συστήματα επικοινωνίας με Η/Υ. Ένα απλό παράδειγμα είναι η δυνατότητα να συνδεθεί κανείς μέσω Bluetooth με το ηχοσύστημα του αυτοκινήτου ώστε να απαντήσει σε μια κλήση στο κινητό του. Όμως, υπάρχουν και άλλα συστήματα επικοινωνίας με Η/Υ πολύ πιο πολύπλοκα και πολύ πιο ενδιαφέροντα.

Ένα τέτοιο σύστημα επικοινωνίας καθώς και η υλοποίησή του θα περιγραφτεί και θα αναλυθεί στο έγγραφο αυτό. Θα καταβληθεί προσπάθεια ώστε η όλη παρουσίαση να είναι απλή, όχι απλοϊκή μιας και απευθύνεται σε ανθρώπους που έχουν σχέση με τη Πληροφορική, σαφής και όσο γίνεται πιο ελκυστική. Καλή συνέχεια λοιπόν!

ΕΙΣΑΓΩΓΗ ΣΤΟ ΠΡΟΒΛΗΜΑ

Το πρόβλημα που επέλεξα και κλήθηκα να λύσω σε αυτή την εργασία είναι με απλά λόγια η συλλογή και επεξεργασία συγκεκριμένων δεδομένων από τον ηλεκτρονικό εγκέφαλο ενός αυτοκινήτου.

Αλλά ας πάρουμε τα πράγματα από την αρχή, μιας και δεν είμαστε όλοι εξοικειωμένοι με την τεχνολογία των αυτοκινήτων. Τα πρόσφατα χρόνια, και, ειδικά από το 1996 και έπειτα, όλοι οι κατασκευαστές αυτοκινήτων είναι υποχρεωμένοι να ενσωματώνουν στα αυτοκίνητα έναν υπολογιστή. Αυτός ο υπολογιστής δεν έχει καμία σχέση στην εμφάνιση με τους κοινούς υπολογιστές που χρησιμοποιούμε καθημερινά. Είναι καλά κρυμμένος από τον οδηγό και ονομάζεται OBD (On-Board-Diagnostics). Η εργασία που κάνει είναι να αντλεί συνεχώς πληροφορίες και δεδομένα από τους πολλούς διαφορετικούς αισθητήρες που έχει ένα αμάξι. Για παράδειγμα υπάρχουν αισθητήρες θερμοκρασίας εντός και εκτός μηχανής του αυτοκινήτου, αισθητήρες που αντιλαμβάνονται την ταχύτητα, τις στροφές του κινητήρα, το αν ο οδηγός ή/και οι επιβάτες φοράνε ζώνη, αισθητήρες που πληροφορούν για τον τύπο καυσίμου που χρησιμοποιεί το αυτοκίνητο, το κατά πόσο τις εκατό ο οδηγός πατάει τα πεντάλ ανα πάσα χρονική στιγμή και πολλά άλλα. Όλες αυτές οι πληροφορίες είναι μπορεί να αποβούν πολύ χρήσιμες στις παρακάτω ρεαλιστικές περιπτώσεις :

- Ένα αυτοκινητιστικό ατύχημα συμβαίνει. Δεν υπάρχουν αυτόπτες μάρτυρες. Η αστυνομία δε μπορεί να είναι σίγουρη για το τί συνέβη πραγματικά. Χρησιμοποιώντας τις πληροφορίες του OBD μπορεί να εξάγει χρήσιμα στοιχεία για το τί συνέβη. [Αυτή η μέθοδος έχει χρησιμοποιηθεί]
- Οι ασφαλιστικές εταιρίες μπορούν να επαληθεύσουν τα πορίσματα των αστυνομικών αρχών.
- Ένας κάτοχος αυτοκινήτου θέλει να γνωρίζει ανά πάσα στιγμή ή κατά διαστήματα όλες ή μερικές πληροφορίες του OBD που κατά τη γνώμη του είναι χρήσιμες.
- Ένας μηχανικός αυτοκινήτου θέλει να ελέγξει τυχόν βλάβες που παρουσιάζει ένα αυτοκίνητο.

Στη παρούσα φάση δε θα ασχοληθούμε αναλυτικά για το πού μπορούν να χρησιμοποιηθούν τα δεδομένα του OBD. Καλούμαστε όμως να ορίσουμε με σαφήνεια το πρόβλημα.

Η συλλογή και επεξεργασία συγκεκριμένων δεδομένων από τον ηλεκτρονικό εγκέφαλο ενός αυτοκινήτου είναι το κυρίως πρόβλημα όπως αναφέρθηκε παραπάνω. Πιο συγκεκριμένα, μέρη αυτού του προβλήματος είναι τα παρακάτω :

- Πώς να συνδεθώ με το OBD ενός αυτοκινήτου ;
- Θα χρειαστεί να φτιάξω κάποια εφαρμογή σε προγραμματιστική γλώσσα για την συλλογή και επεξεργασία των δεδομένων, και αν ναι, ποιά θα είναι αυτή ;
- Τί επεξεργασία θα κάνω πάνω στα δεδομένα ;
- Θα μπορώ να συνδεθώ σε κάθε αμάξι που έχει OBD ;

Αυτά είναι τα κυρίως υπο-προβλήματα που συνιστούν το γενικό πρόβλημα. Σίγουρα τα πράγματα είναι λιγάκι πιο περίπλοκα απ' οτι παρουσιάζονται, όμως ας έχουμε στο μυαλό μας απλά τα πράγματα για να κατάλαβουμε πιο εύκολα τη πορεία της επίλυσης και της υλοποίησης του λογισμικού. Αναφέροντας τη λέξη “λογισμικό” κάποιος μπορεί να καταλάβει οτι η απάντηση στη δεύτερη ερώτηση πιο πάνω είναι ναι!

Ανακεφαλαιώνοντας λοιπόν, καλούμαι σε αυτή την εργασία να υλοποιήσω ένα λογισμικό το οποίο θα συνδέεται μέσω κάποιου τρόπου με τον υπολογστή OBD ενός αυτοκινήτου, θα συλλέγω

τα δεδομένα σε συγκεκριμένες ερωτήσεις μου πάνω στις τιμές διάφορων αισθητήρων και στη συνέχεια θα κάνω ανάλυση αυτών για εξαγωγή πρακτικών συμπερασμάτων. Κάτι πολύ σημαντικό επίσης είναι, οτι όλη αυτή η διαδικασία θα πρέπει να γίνεται δυναμικά, δηλαδή ενώ το αυτοκίνητο είναι σε κίνηση!

ΘΕΩΡΗΤΙΚΟ ΥΠΟΒΑΘΡΟ

Παρακάτω ακολουθούν κάποιες χρήσιμες επεξηγήσεις ώστε να αντιληφθεί κάποιος την υλοποίηση πιο εύκολα.

- **Που βρίσκεται το OBD στο αυτοκίνητο**

  Το OBD συνήθως βρίσκεται πολύ κοντά στη θέση του οδηγού, για παράδειγμα κάτω από το τιμόνι ή δίπλα από αυτό ή κοντά στα πεντάλ. Συχνά προφυλάσσεται από κάποιο καπάκι.

- **RS232**

  Το RS232 είναι ένα πρωτόκολλο επικοινωνίας ενός υπολογιστή μέσω των σειριακών θυρών του. Αν συνδέσω δηλαδή ένα καλώδιο σε μια σειριακή θύρα του λάπτοπ μου, η μεταφορά των δεδομένων θα υπόκειται στο πρωτόκολλο RS232.

- **ELM327**

  Το ELM327 είναι ένα CMOS ολοκληρωμένο κύκλωμα που δρα ως πρωτόκολλο και ως γέφυρα ανάμεσα στο OBD και στο πρωτόκολλο RS232. Με απλά λόγια, αν θέλω να πάρω τα δεδομένα από το OBD, δε μπορώ να το κάνω απευθείας. Πρέπει να χρησιμοποιήσω το λάπτοπ μου για παράδειγμα, το οποίο θα έχει συνδεθεί μέσω μιας σειριακής θύρας με το ELM327. Το ELM327 θα είναι συνδεδεμένο απευθείας με το OBD.

- **Virtual Serial Port**

  Στη περίπτωση που δε θέλω να συνδέσω το λάπτοπ μου μέσω καλωδίου στη σειριακή θύρα μπορώ να συνδεθώ στη σειριακή θύρα και με άλλους τρόπους. Ένας από αυτούς είναι μέσω Bluetooth. Έτσι λοιπόν, μπορώ να συνδεθώ με το ELM327 μέσω μιας εικονικής σειριακής θύρας (virtual serial port) η οποία “ανοίγει” μέσω του Bluetooth.

- **Επικοινωνία του Η/Υ με το ELM327**

  Αφού συνδεθώ με το ELM327 το οποίο με τη σειρά του συνδέεται στο OBD, θα πρέπει να βρω ένα τρόπο ώστε να στέλνω και να λαμβάνω δεδομένα. Ο πιο απλός τρόπος είναι μέσω ενός τερματικού (terminal) όπως για παράδειγμα το HyperTerminal ή το TeraTerm. Βέβαια, για να χρησιμοποιήσει κάποιος ένα τερματικό για αυτή την επικοινωνία θα πρέπει πρώτα να κάνει κάποιες ρυθμίσεις στο τερματικό οι οποίες θα περιγραφούν αργότερα.

- **Επικοινωνία του ELM327 με το OBD**

  Η επικοινωνία αυτή χαρακτηρίζεται από μηνύματα send και receive. Ας υποθέσουμε οτι θέλω να μάθω τις στροφές του κινητήρα τη τρέχουσα χρονική στιγμή. Τότε, θα πρέπει μέσω του τερματικού να στείλω δεδομένα που αντιπροσωπεύουν την ερώτηση “Πόσες είναι οι στροφές του κινητήρα αυτή τη στιγμή;”. Αυτή η ερώτηση, σύμφωνα με το πρωτόκολλο επικοινωνίας OBD το οποίο είναι διεθνώς διαθέσιμο, θα γίνει αν στείλω την εξής εντολή μέσω του τερματικού “01 0c”. Το 01 αναφέρεται ως mode και το 05 είναι το PID. Δε χρειάζεται να τα μάθει κάποιος αυτά απ' έξω αφού υπάρχουν με ένα απλό search στο Διαδίκτυο. Υπάρχουν δεκάδες διαφορετικές ερωτήσεις που μπορεί να κάνει κάποιος στο OBD. Κάτι πολύ σημαντικό όμως : Οι απαντήσεις που μας στέλνει το OBD πίσω στο τερματικό μας, είναι σε δεκαεξαδική μορφή. Χρειάζεται λοιπόν να γίνει μετατροπή τους σε δεκαδική μορφή ώστε να είναι κατανοητές και, κάποιες από αυτές χρειάζονται και επιπλέον επεξεργασία. Για παράδειγμα, αν στείλω την εντολή “01 05”,

τότε θα ζητάω τη θερμοκρασία του κινητήρα. Μια πιθανή απάντηση είναι η “41 05 7B”. Το “41 05” επιβεβαιώνει οτι η ερώτηση που έκανα ήταν η “01 05”. Το “7B” είναι η απάντηση σε δεκεεξαδική μορφή. Αν τη μετατρέψω σε δεκαδική μορφή θα πάρω 7 x 16 + 11 = 123. Αν απο το 123 αφαιρέσω 40 θα πάρω τη πραγματική θερμοκρασία σε βαθμους Κελσίου -> 123 – 40 = 83 βαθμοί Κελσίου!

- **.NET Framework**

  Είναι ένα πολύ χρήσιμο σύνολο από κλάσεις και λειτουργίες που σκοπό έχουν να βοηθάνε ένα προγραμματιστή να γράφει ποιοτικά προγράμματα και εφαρμογές για περιβάλλον Microsoft Windows

- **Microsoft Visual C++**

  Είναι ένα IDE το οποίο βοηθάει ένα προγραμματιστή να γράφει και να μεταγλωττίζει προγράμματα σε C/C++ αλλά και σε περιβάλλον .NET χρησιμοποιώντας κλάσεις με στυλ συγγραφής κώδικα της C++. Για την εκπόνηση αυτής της εργασίας χρησιμοποιήθηκε το .NET framework 3.5 και η Visual C++ 2008.

ΣΧΕΔΙΑΣΜΟΣ ΤΟΥ ΣΥΣΤΗΜΑΤΟΣ

Σε αυτή την ενότητα, θα περιγραφτούν τα βήματα που ακολουθήθηκαν ώστε να υλοποιηθεί η λύση του προβλήματος. [Θα αναφέρομαι σε πρώτο πρόσωπο]

- Σύνδεση λάπτοπ με OBD

Κατ' αρχήν, έπρεπε να βεβαιωθώ οτι μπορούμε να συνδεθούμε δυναμικά μέσω υπολογιστή με το OBD ενός αυτοκινήτου. Γι' αυτό τον λόγο λοιπόν, εγκατέστησα ένα πρόγραμμα τερματικού στο λάπτοπ μου, το Tera Term. Το λάπτοπ μου έχει δυνατότητα ασύρματης σύνδεσης μέσω Bluetooth. Το ELM327 συνδέεται με το λάπτοπ μέσω Bluetooth πάνω από τη virtual port.

- Επιβεβαίωση επικοινωνίας και ανταλλαγής δεδομένων

Αφού λοιπόν συνδέθηκα με το OBD μέσω του λάπτοπ μου, έπρεπε να δω αν μπορώ να ανταλλάξω δεδομένα με το OBD και μάλιστα αν μπορώ να ανταλλάξω τα δεδομένα που θα με βοηθήσουν στο πρόβλημα.

Στέλνω την εντολή “AT RV” η οποία ζητά από το OBD τη τιμή σε Volt της μπαταρίας του αυτοκινήτου. Η απάντηση που πήρα ήταν κοντά στα 13.6 V και 14,3 όταν η μηχανή του αυτοκινήτου ήταν αναμμένη. Αυτό ήταν! Υπάρχει σωστή επικοινωνία.

- Προσομοίωση συμπεριφοράς τερματικού σε Visual C++

Για να λυθεί το πρόβλημα χωρίς να χρειάζεται ο χρήστης να χρησιμοποιεί ένα τερματικό, έπρεπε να προσομοιώσω τη διαδικασία σύνδεσης του λάπτοπ με το OBD μέσω μιας virtual serial port σε μια γλώσσα προγραμματισμού, ώστε ο χρήστης απλά να τρέχει μια εφαρμογή η οποία θα “μαζεύει” τις απαντήσεις του OBD σε ένα αρχείο .txt για παράδειγμα. Ξεκίνησα λοιπόν τη συγγραφή κώδικα σε C++. Το .NET framework βοήθησε πολύ στη προσομοίωση αυτή, καθώς παρέχει έτοιμες κλάσεις οι οποίες πραγματοποιούν μια σύνδεση σε σειριακή θύρα (serial port). Επίσης, παρέχει τη δυνατότητα να μπορώ να κάνω την εφαρμογή μου να αντιδρά όπως θέλω σε διάφορα γεγονότα (event driven programming). Παρακάτω, ακολουθούν κάποιες βασικές κλάσεις και λειτουργίες που χρησιμοποίησα στην υλοποίηση του συστήματος.

- SerialPort Class

  Αυτή είναι η πιο βασική κλάση για τη λύση του προβλήματος και για την υλοποίηση του συστήματος. Με μια αναζήτηση στο Διαδίκτυο μπορεί κάποιος να δει οτι αυτή η κλάση πρσφέρει πάρα πολλές χρήσιμες λειτουργίες για τη χρήση και την παραμετροποίηση μιας σειριακής θύρας (πχ baud rate, handshake protocol, encoding, partity bit, stop bits κλπ). Πιο πολλές λεπτομέρειες παρέχονται στον κώδικα της υλοποίησης.

- DataReceivedHandler

  Αυτή η συνάρτηση βοηθάει στον να γνωρίζει ο προγραμματιστής την ακριβή χρονική στιγμή που κάποια δεδομένα συγκεκριμένου μεγέθους, έρχονται προς τη σειριακή θύρα. Αυτή η γνώση με τη σειρά της επιτρέπει τη σωστή και χωρίς απώλεια δεδομένων σειρά ερωτήσεων στο OBD.

- StreamWriter Class

Αυτή η κλάση χρησιμοποιήθηκε πολύ συχνά ώστε να καταγραφούν τα δεδομένα επικοινωνίας από και προς το OBD. Η καταγραφή αρχικά έγινε σε αρχεία .txt.

Στη αρχή της συγγραφής του κώδικα, είχα υπόψη μου αυτά τα 3 βασικά εργαλεία. Βέβαια, για να γίνει σωστά η προσομοίωση στη Visual C++, έπρεπε να προσομοιώσω τα 2 αρχικά βήματα που αναφέρονται στην αρχή αυτής της ενότητας.

Ακολουθεί μια περίληψη της πιο σημαντικής κλάσης της υλοποίησης, της SerialPort Class [πηγή: msdn]

Constructors

SerialPort()	Initializes a new instance of the SerialPort class.
SerialPort(IContainer)	Initializes a new instance of the SerialPort class using the specified IContainer object.
SerialPort(String)	Initializes a new instance of the SerialPort class using the specified port name.
SerialPort(String, Int32)	Initializes a new instance of the SerialPort class using the specified port name and baud rate.
SerialPort(String, Int32, Parity)	Initializes a new instance of the SerialPort class using the specified port name, baud rate, and parity bit.
SerialPort(String, Int32, Parity, Int32)	Initializes a new instance of the SerialPort class using the specified port name, baud rate, parity bit, and data bits.
SerialPort(String, Int32, Parity, Int32, StopBits)	Initializes a new instance of the SerialPort class using the specified port name, baud rate, parity bit, data bits, and stop bit.

Properties

BaseStream	Gets the underlying Stream object for a SerialPort object.
BaudRate	Gets or sets the serial baud rate.
BreakState	Gets or sets the break signal state.
BytesToRead	Gets the number of bytes of data in the receive buffer.
BytesToWrite	Gets the number of bytes of data in the send buffer.
CanRaiseEvents	Gets a value indicating whether the component can raise an event. (Inherited from Component.)
CDHolding	Gets the state of the Carrier Detect line for the port.
Container	Gets the IContainer that contains the Component. (Inherited from Component.)
CtsHolding	Gets the state of the Clear-to-Send line.
DataBits	Gets or sets the standard length of data bits per byte.
DesignMode	Gets a value that indicates whether the Component is currently in design mode. (Inherited from Component.)
DiscardNull	Gets or sets a value indicating whether null bytes are ignored when transmitted between the port and the receive buffer.
DsrHolding	Gets the state of the Data Set Ready (DSR) signal.
DtrEnable	Gets or sets a value that enables the Data Terminal Ready (DTR) signal during serial communication.
Encoding	Gets or sets the byte encoding for pre- and post-transmission conversion of text.
Events	Gets the list of event handlers that are attached to this Component. (Inherited from Component.)
Handshake	Gets or sets the handshaking protocol for serial port transmission of data.
IsOpen	Gets a value indicating the open or closed status of the SerialPort object.
NewLine	Gets or sets the value used to interpret the end of a call to the ReadLine and WriteLine methods.
Parity	Gets or sets the parity-checking protocol.

ParityReplace	Gets or sets the byte that replaces invalid bytes in a data stream when a parity error occurs.
PortName	Gets or sets the port for communications, including but not limited to all available COM ports.
ReadBufferSize	Gets or sets the size of the SerialPort input buffer.
ReadTimeout	Gets or sets the number of milliseconds before a time-out occurs when a read operation does not finish.
ReceivedBytesThreshold	Gets or sets the number of bytes in the internal input buffer before a DataReceived event occurs.
RtsEnable	Gets or sets a value indicating whether the Request to Send (RTS) signal is enabled during serial communication.
Site	Gets or sets the ISite of the Component. (Inherited from Component.)
StopBits	Gets or sets the standard number of stopbits per byte.
WriteBufferSize	Gets or sets the size of the serial port output buffer.
WriteTimeout	Gets or sets the number of milliseconds before a time-out occurs when a write operation does not finish.

Methods

Close	Closes the port connection, sets the IsOpen property to false, and disposes of the internal Stream object.
CreateObjRef	Creates an object that contains all the relevant information required to generate a proxy used to communicate with a remote object. (Inherited from MarshalByRefObject.)
DiscardInBuffer	Discards data from the serial driver's receive buffer.
DiscardOutBuffer	Discards data from the serial driver's transmit buffer.
Dispose()	Releases all resources used by the Component. (Inherited from Component.)
Dispose(Boolean)	Releases the unmanaged resources used by the SerialPort and optionally releases the managed resources. (Overrides Component::Dispose(Boolean).)
Equals(Object)	Determines whether the specified Object is equal to the current Object. (Inherited from Object.)
Finalize	Releases unmanaged resources and performs other cleanup operations before the Component is reclaimed by garbage collection. (Inherited from Component.)
GetHashCode	Serves as a hash function for a particular type. (Inherited from Object.)
GetLifetimeService	Retrieves the current lifetime service object that controls the lifetime policy for this instance. (Inherited from MarshalByRefObject.)
GetPortNames	Gets an array of serial port names for the current computer.
GetService	Returns an object that represents a service provided by the Component or by its Container. (Inherited from Component.)
GetType	Gets the Type of the current instance. (Inherited from Object.)
InitializeLifetimeService	Obtains a lifetime service object to control the lifetime policy for this instance. (Inherited from MarshalByRefObject.)
MemberwiseClone()	Creates a shallow copy of the current Object. (Inherited from Object.)
MemberwiseClone(Boolean)	Creates a shallow copy of the current MarshalByRefObject object. (Inherited from MarshalByRefObject.)
Open	Opens a new serial port connection.
Read(array<Byte>, Int32, Int32)	Reads a number of bytes from the SerialPort input buffer and writes those bytes into a byte array at the specified offset.
Read(array<Char>, Int32, Int32)	Reads a number of characters from the SerialPort input buffer and writes them into an array of characters at a given offset.
ReadByte	Synchronously reads one byte from the SerialPort input buffer.
ReadChar	Synchronously reads one character from the SerialPort input buffer.
ReadExisting	Reads all immediately available bytes, based on the encoding, in both the stream and the input buffer of the SerialPort object.
ReadLine	Reads up to the NewLine value in the input buffer.

ReadTo	Reads a string up to the specified value in the input buffer.
ToString	Returns a String containing the name of the Component, if any. This method should not be overridden. (Inherited from Component.)
Write(String)	Writes the specified string to the serial port.
Write(array<Byte>, Int32, Int32)	Writes a specified number of bytes to the serial port using data from a buffer.
Write(array<Char>, Int32, Int32)	Writes a specified number of characters to the serial port using data from a buffer.
WriteLine	Writes the specified string and the NewLine value to the output buffer.

ΥΛΟΠΟΙΗΣΗ

Υπάρχουν κάποια πράγματα που αφορούν την υλοποίηση του συστήματος που πρέπει να προσέξει κάποιος.

- Επιλογή σωστής σειριακής θύρας (COM port)

  Είναι απαραίτητο να γνωρίζουμε το σωστό αριθμό της σειριακής θύρας που χρησιμοποιεί το Bluetooth του λάπτοπ. Πχ στο δικό μου λάπτοπ η σωστή θύρα είναι η COM 16. Αυτό τον αριθμό θα βάλουμε ως παράμετρο όταν δημιουργήσουμε ένα αντικείμενο της κλάσης SerialPort.

- Επιλογή σωστού data rate

  Για να βρει κάποιος το σωστό data rate χρειάζεται να ανατρέξει στο manual του ELM327 που έχει online η ELM Electronics. Συνήθως το σωστό data rate είναι 9600 baud.

- Επιλογή για επικοινωνία μέσω 8 data bits

- Επιλογή ενός stop bit

- Επιλογή κανενός parity bit

- Κατανόηση των ειδικών χαρακτήρων επιστροφής του ELM327

  Όλες οι απαντήσεις που επιστρέφει το ELM327 και κατά συνέπεια και το OBD, τερματίζονται από έναν χαρακτήρα carriage return (CR) και, ανάλογα τις ρυθμίσεις, μαζί με ένα χαρακτήρα αλλαγής γραμής – linefeed (LF). Έχοντας υπόψη αυτό το γεγονός, ο προγραμματιστής που θέλει να διαβάσει χαρακτήρα χαρακτήρα τις απαντήσεις που επιστρέφονται, ελέγχει πότε εμφανίζονται αυτοί οι ειδικοί χαρακτήρες και έτσι μορφοποιεί κατάλληλα το αρχείο που χρησιμοποιεί ως log των απαντήσεων (πχ ένα .txt)

- Επιλογή χρόνου αναμονής για επόμενη ερώτηση

  Λόγω του οτι η ανταλλαγή δεδομένων (ερώτηση-απάντηση και επόμενη ερώτηση-απάντηση) γίνεται ταυτόχρονα σε πραγματικό χρόνο ίσως υπάρξει πρόβλημα ταυτόχρονης ανάγνωσης και εγγραφής μιας μεταβλητής. Δεν έκρινα σκόπιμο να χρησιμοποιήσω ειδικά εργαλεία ταυτοχρονισμού όπως monitors ή σηματοφόρους. Όμως εκμεταλλεύτηκα το μερικώς thread-safe περιβάλλον που προσφέρει το .NET και επέλεξα να γίνονται οι ερωτήσεις προς το OBD κάθε X sec, με X τροποποιήσιμο.

- Επιλογή απόρριψης εισερχόμενων bytes με τιμή 00 (NULL)

  Υπάρχει μια πολύ μικρή πιθανότητα να εισχωρήσουν NULL χαρακτήρες μέσω των

δεδομένων που εισέρχονται στη σειριακή θύρα. Για να είμαστε σίγουροι οτι δεν θα υπάρξουν προβλήματα επιλέγουμε ως αληθή τη μεταβλητή DiscardNull της κλάσης SerialPort.

- Επιλογή διακόπτη μηχανής στο ON για επικοινωνία με OBD

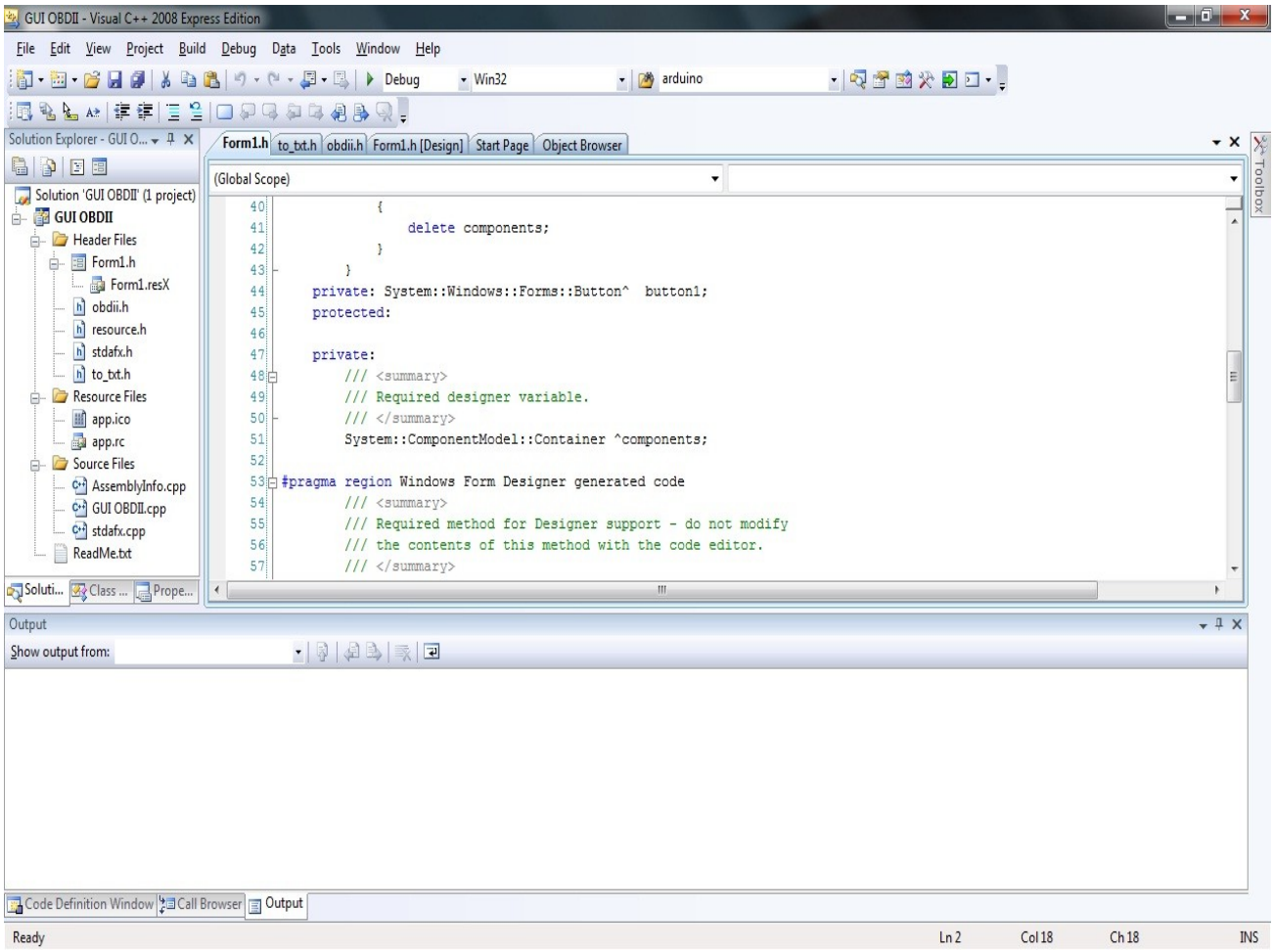
- Επιλογή εμφάνισης των ερωτήσεων ή όχι στη κονσόλα

  Αυτό γίνεται με τη χρήση των εντολών “AT e1” και “AT e0” αντίστοιχα

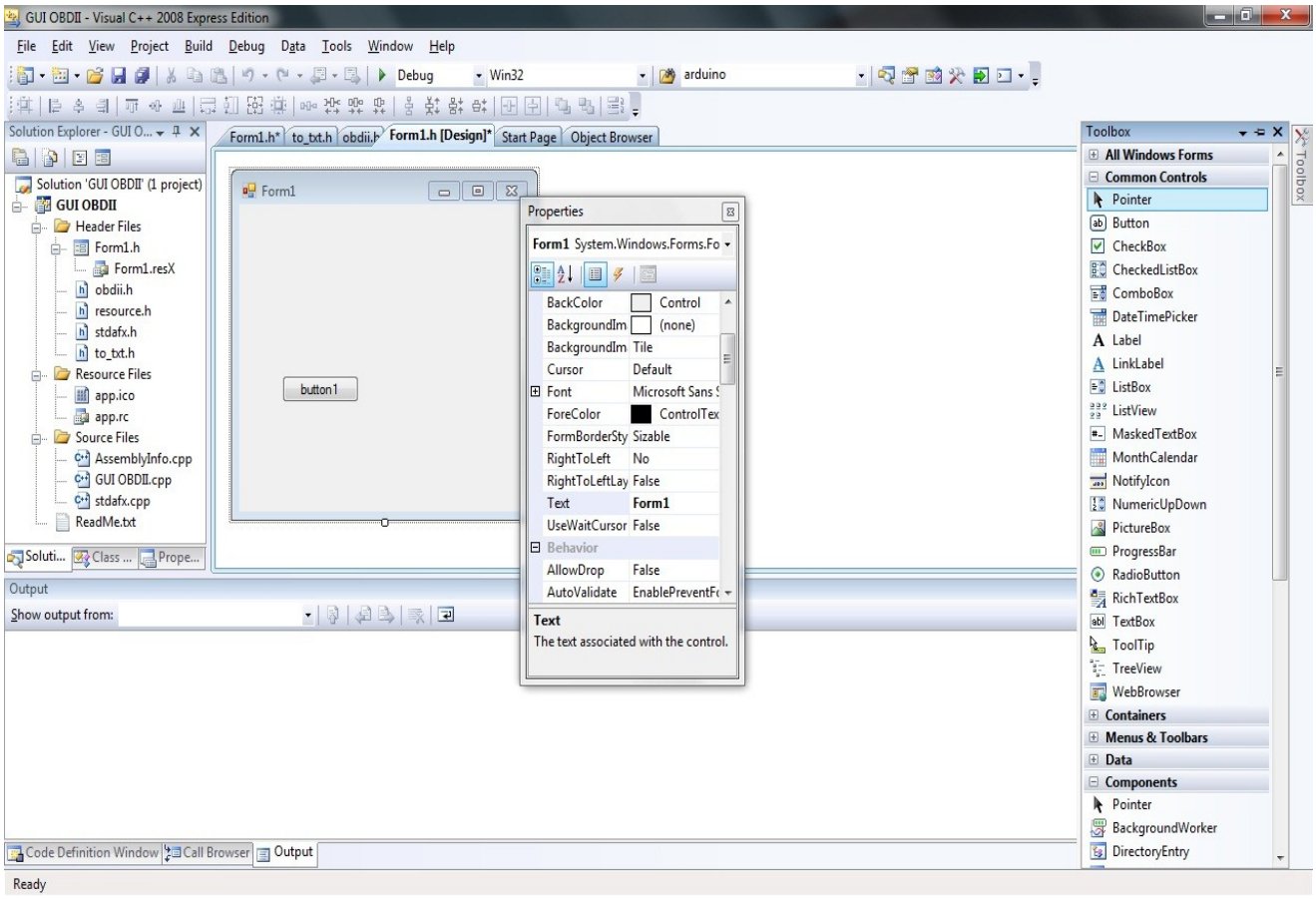
ΕΚΤΕΛΕΣΗ

Όσον αφορά την εκτέλεση του προγράμματος που υλοποιεί το σύστημα και επιλύει το πρόβλημα που τέθηκε στην αρχή, ακολουθούν κάποια screenshots από διάφορα μέρη του συστήματος:

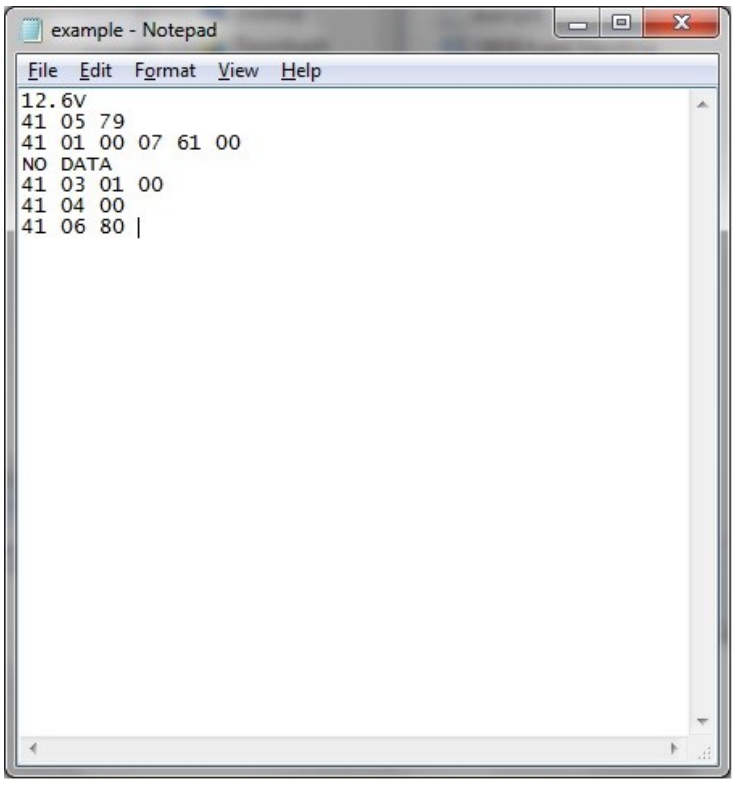
- περιβάλλον εργασίας Visual C++ 2008



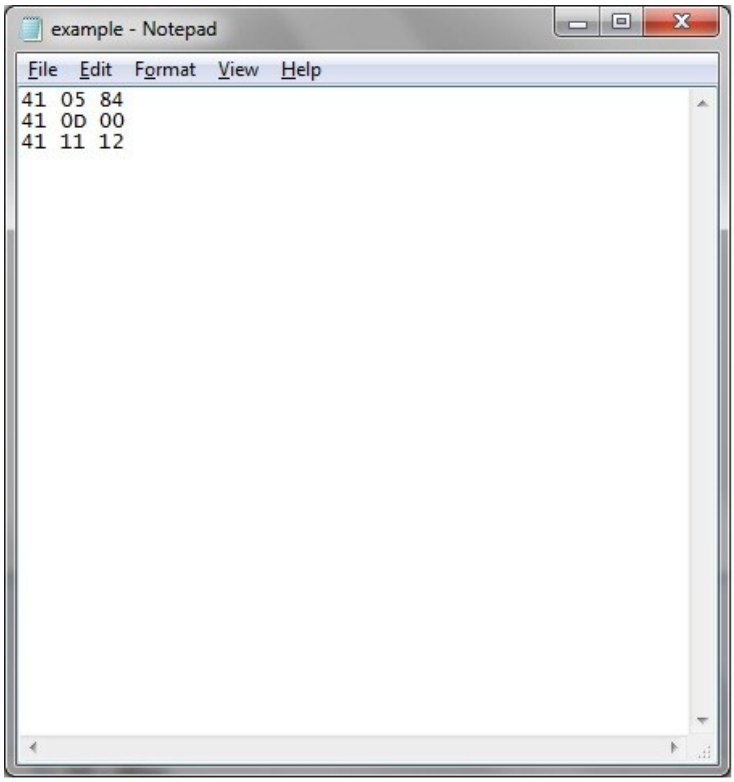
- σχεδίαση του παραθύρου επικοινωνίας με τον χρήστη



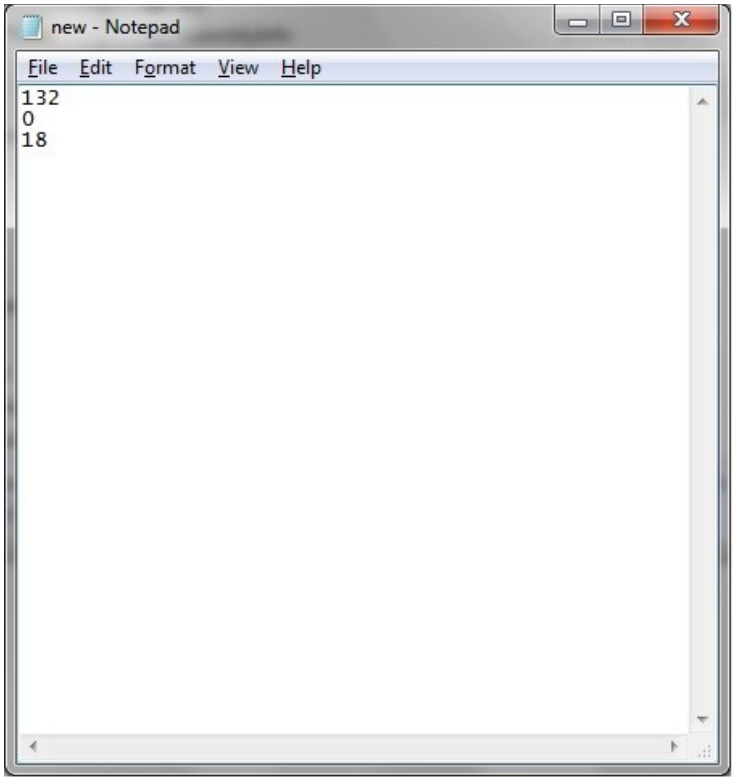
- παραγόμενο .txt αρχείο με απαντήσεις σε 7 διαφορετικές ερωτήσεις (παρατηρήστε τις απαντήσεις που είναι σε δεκαεξαδική μορφή)



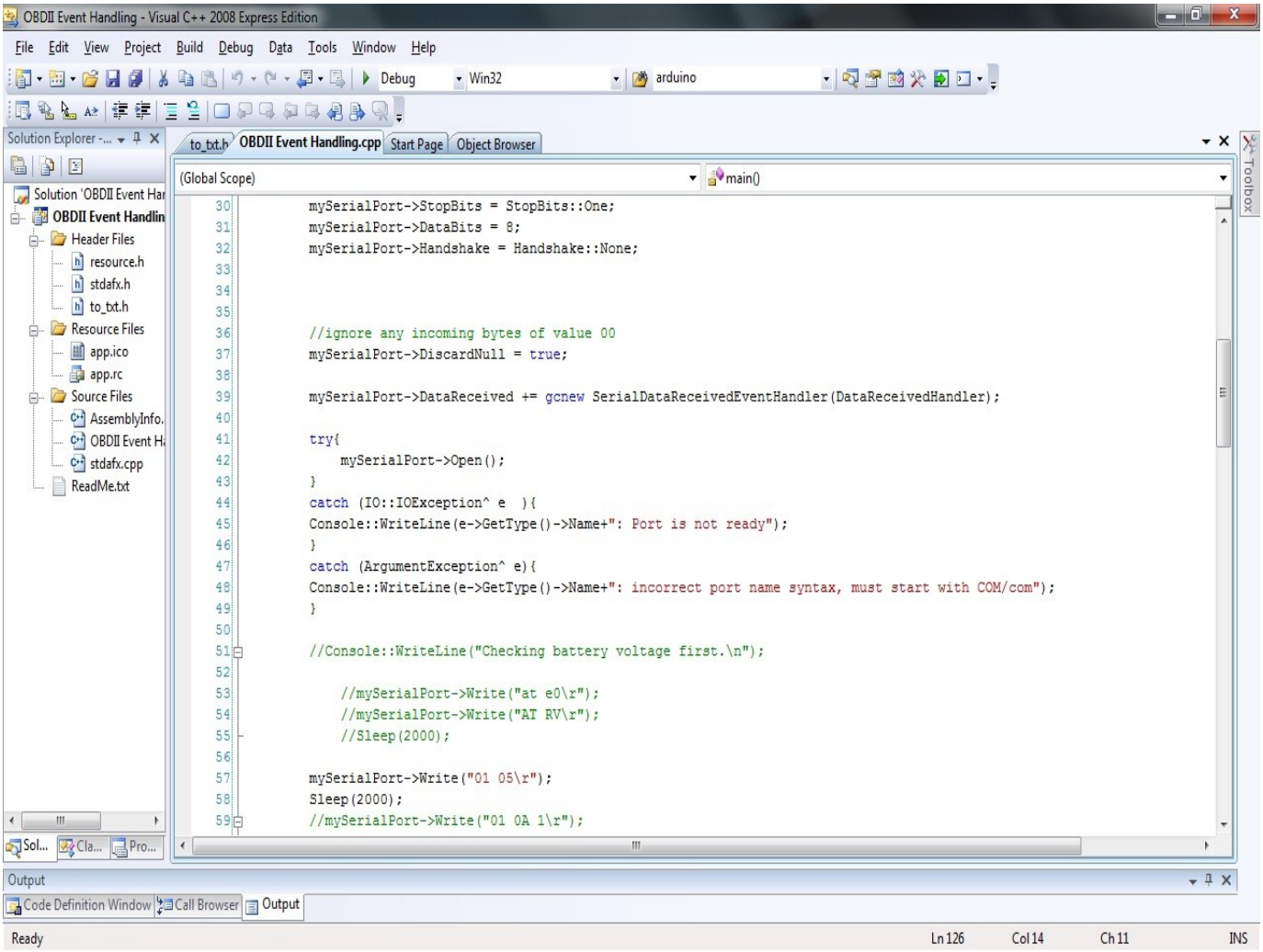
- παραγόμενο .txt αρχείο με απαντήσεις σε 3 διαφορετικές ερωτήσεις



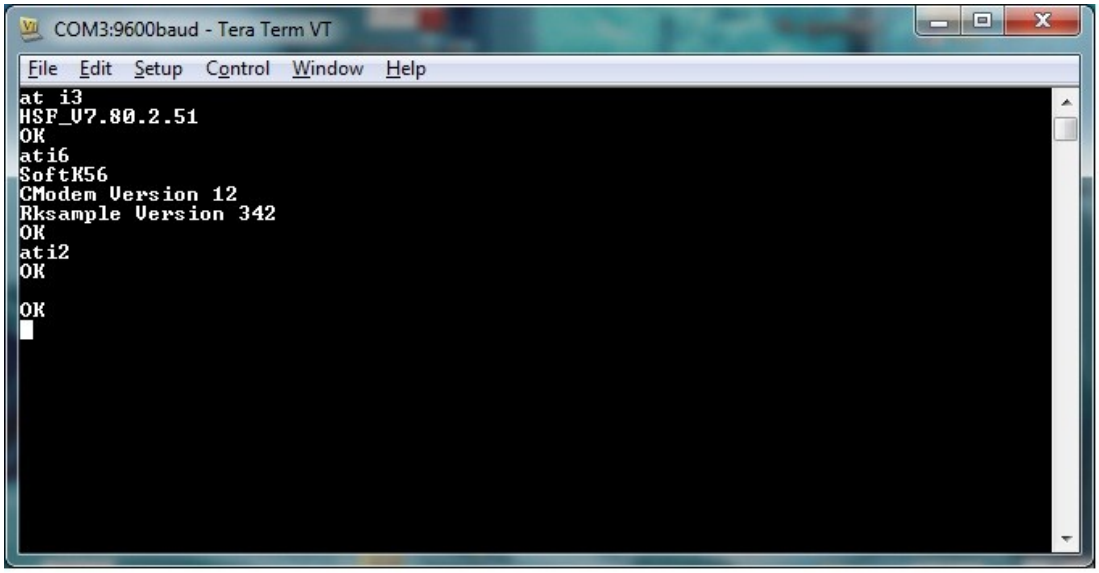
- αρχείο .txt με αποκωδικοποιημένες τις απαντήσεις του προηγούμενου αρχείο σε δεκαδική μορφή



- μια μικρή “γεύση” από τον κώδικα προσομοιώσης της λειτουργίας του τερματικού που υλοποίησα σε C++



- πειραματικό περιβάλλον τερματικού Tera Term τη στιγμή που επικοινωνεί με τη θύρα 3 του λάπτοπ μου (όπου βρίσκεται το modem)



ΣΥΜΠΕΡΑΣΜΑΤΑ ΚΑΙ ΜΕΛΛΟΝΤΙΚΗ ΕΡΓΑΣΙΑ

- Συμπεράσματα

Κατά τη διάρκεια εκπόνησης αυτής της εργασίας αλλά και μετά το τέλος της, υπήρξαν κάποια χρήσιμα συμπεράσματα σε πολλούς τομείς, θεωρητικούς αλλά και πρακτικούς!

Όσον αφορά τα συμπεράσματα που αφορούν το πρακτικό μέρος της εργασίας, αυτό που είναι σχεδόν σίγουρο είναι, οτι, όταν όταν κάποιος αποφασίσει να σχεδιάσει αλλά και να προγραμματίσει ένα σύστημα που αφορά επικοινωνία μεταξύ διαφορετικών πρωτοκόλλων, θα συναντήσει πολλές δυσκολίες. Χρειάζεται μέσω πολλών δοκιμών να κατανοήσει πώς δουλεύει το κάθε πρωτόκολλο και ποιές είναι οι ιδιαιτερότητές τους. Αφού κατανοήσει τα πρωτόκολλα, τότε χρειάζεται να σχεδιάσει πώς θα μεταφέρει τις γνώσεις αυτές σε προγραμματιστικό περιβάλλον. Αυτό που βοηθάει πολύ στο σχεδιασμό αλλά και στον προγραμματισμό ενός τέτοιου συστήματος επικοινωνίας είναι το να γίνει αρκετή προετοιμασία για κάθε μέρος του συστήματος με μολύβι και χαρτί.

Επίσης, επειδή η επικοινωνία που αφορά το OBD με το λάπτοπ γίνεται μέσω Bluetooth, ασύρματα δηλαδή, δεν είσαι ποτέ σίγουρος αν τα δεδομένα που περιμένεις θα έρθουν όλα, πόσο θα αργήσουν ή αν δε θα έρθουν καθόλου. Βέβαια, αυτή η αβεβαιότητα αντιμετωπίστηκε εν μέρει μέσω κατάλληλων μηχανισμών event driven προγραμματισμού.

Σχετικά με το .NET interface, έχω μόνο καλά λόγια να πω. Οι κλάσεις και οι λειτουργίες που προσφέρει λύνουν τα χέρια του προγραμματιστή σε πολλές περιπτώσεις.

Όσον αφορά τα όχι και τόσο τεχνικά συμπεράσματα, αυτό που μου έμεινε από αυτή την εργασία είναι οτι μπορείς να κάνεις οτι θέλεις αν έχεις θέληση αλλά και προσεκτική οργάνωση του χρόνου σου. Όταν κάποιος προγραμματίζει ένα σύστημα σε μια γλώσσα προγραμματισμού, χρειάζεται να σκέφτεται και να ξανασκέφτεται τα πράγματα. Αυτό γίνεται αποδοτικά μόνο αν δίνει στον εαυτό του χρόνο να ξεκουραστεί και να καθαρίσει το μυαλό του. Το να γίνει κάτι υπερβολικά βιαστικά, όσες ικανότητες και αν έχεις, είναι τουλάχιστον ανθυγιεινό! Θα ήταν παράλειψη αν δεν τόνιζα τη τεράστια βοήθεια που πρόσφερε το Διαδίκτυο στην εκπόνηση αυτής της διπλωματικής εργασίας., κάτι που θα γίνει αντιληπτό και από τις πηγές που παρατίθενται παρακάτω.

- Μελλοντική εργασία

Σίγουρα υπάρχουν πράγματα που μπορούν να γίνουν στην υλοποίηση αυτού του προγράμματος ώστε να βελτιωθεί η αποτελεσματικότητα αλλά και η διεπαφή με το χρήστη.

Θα μπορούσε για παράδειγμα, η παρουσίαση των αποτελεσμάτων(των απαντήσεων) του OBD, να γίνει με γραφικές παραστάσεις και γενικά με χρήση γραφημάτων που δείχνουν πιο έντονα κάποια τάση του οδηγού του αυτοκινήτου (πχ πατάει πολύ συχνά τέρμα το γκάζι!).

Ήδη έχει γίνει έρευνα από μέρους μου αλλά και υπάρχει demo με παρουσίαση αποτελεσμάτων με γραφήματα. Ένας τρόπος για να γίνει αυτό είναι το να γραφτούν συναρτήσεις στη C++ οι οποίες θα διαβάζουν τα δεδομένα των απαντήσεων από το .txt αρχείο και θα παρουσιάζουν, μέσω χρήσης OpenGL για παράδειγμα, τις απαντήσεις σε γραφικές παραστάσεις σε συνάρτηση με το χρόνο.

Κάτι λιγότερο χρονοβόρο και άκρως εντυπωσιακό, είναι το να χρησιμοποποιηθεί το API που προσφέρει η Google μέσω του προγράμματος Google chart. Αξίζει να ρίξετε μια ματιά σε αυτό το πολύ απλό εγχείρημα της Google. Να σημειωθεί οτι υπάρχει τρόπος να χρησιμοποιήσει κανείς αυτή τη διεπαφή και offline.

Τέλος, μια ερώτηση που τέθηκε στην αρχή ήταν “Θα μπορώ να συνδεθώ σε κάθε αμάξι που έχει

OBD ;”. Η απάντηση σε αυτή την ερώτηση είναι μάλλον όχι, καθώς κάποια αμάξια χρησιμοποιούν διαφορετικά πρωτόκολλα επικοινωνίας με το OBD. Πάντως, στα 2 αμάξια τελευταίας τεχνολογίας (< 5 έτη) που είχα στη διάθεση μου (Seat Ibiza και Nissan Note) για τις δοκιμές του συστήματος, δε συνάντησα κανένα πρόβλημα συμβατότητας κώδικα και επικοινωνίας. Όλα λειτούργησαν όπως έπρεπε!

ΒΑΣΙΚΕΣ ΠΗΓΕΣ

ΑΦΟΡΜΗ ΓΙΑ ΤΗΝ ΙΔΕΑ

- http://news.bbc.co.uk/2/hi/programmes/click_online/9631683.stm

OBD

- http://en.wikipedia.org/wiki/OBD-II_PIDs

CRASH DATA RETRIEVAL

- http://www.cdr-system.com/resources/multimedia.html

CRASH RECONSTRUCTION SOFTWARE

- http://www.accidentreconstruction.com/
- http://www.hvecsi.com/

Visual C++

- http://msdn.microsoft.com
- http://www.codeguru.com/cpp/i-n/network/serialcommunications/article.php/c2503

Datasheets

- 76 pages .pdf "ELM327 – OBD TO RS232 Interpreter"

+ πάρα πολλές άλλες διαδικτυακές πηγές που βοήθησαν στη συγγραφή του κώδικα