



School of
Electrical &
Computer
Engineering

Ενσωματωμένα Συστήματα Μικροεπεξεργαστών HPY 411

Σύστημα Αυτόματου Εντοπισμού και Παρακολούθησης Ουρανίων Σωμάτων

Πλήρης Παρουσίαση Συστήματος
Αναλυτικό Εγχειρίδιο Υλοποίησης

Γιακουμάκης Πάρις-Παύλος
Γκιώνης Νίκος
Πορτοκαλάκης Πέτρος

21 Δεκεμβρίου 2017

Περιεχόμενα

1 Εισαγωγή.....	1
2 Παρουσίαση του HORIZONS system του JPL της NASA.....	1
3 Μοντελοποίηση σε MATLAB.....	6
4 Υλοποίηση Συστήματος.....	8
4.1 Περιστροφική Βάση.....	8
4.2 AVR/STK500.....	8
4.3 Κύκλωμα Διεπαφής.....	9
4.3.1 Custom PCB-MAX232.....	9
4.3.2 RS232to485 Adaptor.....	11
4.4 RTC Κύκλωμα.....	12
4.5 Επισκόπηση Πλήρους Συστήματος.....	14
4.5.1 Εικόνα Συστήματος.....	14
4.5.2 Block Diagram Συστήματος.....	15
5 Κώδικας Χειρισμού Βάσης.....	16
6 Calibration Βάσης.....	17
7 Περιγραφή Κώδικα Συστήματος.....	18
8 Testing Συστήματος.....	20
9 Προβλήματα κατά την ανάπτυξη του συστήματος & Διαδικασία Αποσφαλμάτωσης.....	21
9.1 Αποσφαλμάτωση AVR.....	21
9.2 Αποσφαλμάτωση Περιστροφικής Βάσης AS20RS485.....	22
9.3 Ελαττωματικός Μετασχηματιστής.....	23
9.4 Λύση Προβλήματος.....	24
9.5 Σχόλια για το Debugging του κώδικα.....	24
10 Βελτιώσεις και Μελλοντικές Προσθήκες.....	25
11 Ευχαριστίες.....	25
11 Βιβλιογραφικές Αναφορές.....	26
12 Παράρτημα.....	28

1 Εισαγωγή

Το σύστημα που υλοποιήθηκε βασίζεται στους εξής βασικούς άξονες: Το σύστημα HORIZONS του JPL της NASA, τον μικροελεγκτή ATmega16L και την αναπτυξιακή πλακέτα STK500, την περιστροφική βάση AS20RS485 και ενδιάμεσα κυκλώματα για τη σύνδεση των υποσυστημάτων αλλά και για πρόσθετες λειτουργίες. Το σύστημα HORIZONS παράγει τα δεδομένα συντεταγμένων τα οποία, έπειτα από επεξεργασία, εισάγονται στον AVR. Με βάση αυτά ο μικροελεγκτής στέλνει εντολές για τον εντοπισμό του εκάστοτε στόχου, αλλά και για την παρακολούθησή του (tracking) στη συνέχεια, για όσο διάστημα έχουν εισαχθεί συντεταγμένες στον μικροελεγκτή. Η παρατήρηση του στόχου γίνεται μέσω υψηλής ισχύος Laser καθώς και εφαρμογής κινήτου, τα οποία προσαρμόζονται στη βάση.

2 Παρουσίαση του HORIZONS System του JPL της NASA

Το [HORIZONS System](#) είναι μία on-line υπηρεσία παροχής πληροφοριών του ηλιακού μας συστήματος και εξαγωγής εφημερίδων (ephemerides). Ephemerides είναι πινακοποιημένα δεδομένα που αφορούν σε ουράνια σώματα (π.χ. ταχύτητα, θέση κλπ).

Σημείωση: τα δεδομένα που εμφανίζονται σε διάφορα γραφήματα και screenshots στην συνέχεια δεν αφορούν στο ίδιο ephemeris.

Το HORIZONS παρέχει τρεις τρόπους εύρεσης των ζητούμενων δεδομένων.

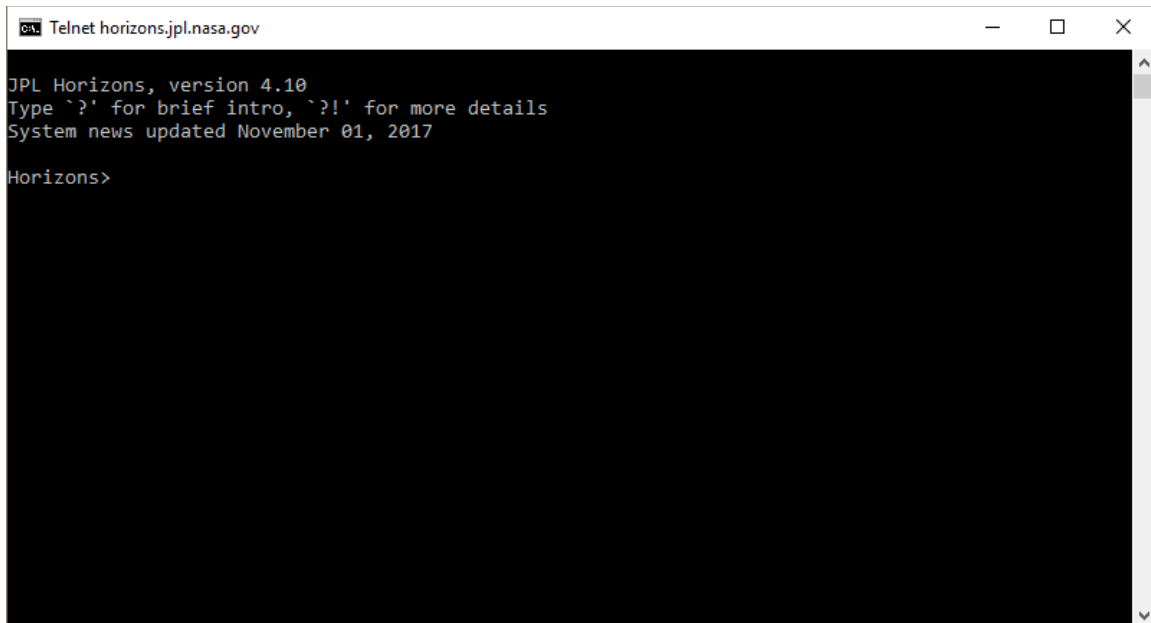
- telnet interface
- email
- web-interface

Το email δεν χρησιμοποιήθηκε στο παρόν project. Το web interface χρησιμοποιήθηκε στην αναζήτηση ουρανίων σωμάτων που έχουν επιθυμητές θέσεις, καθώς αρκεί μόνο η αλλαγή του ουράνιου σώματος για την εξαγωγή των νέων δεδομένων. Το telnet interface, το οποίο και χρησιμοποιήθηκε, ενώ έχει πρόσβαση σε περισσότερα δεδομένα από το email και το web interface, είναι χρονοβόρο ως προς την εισαγωγή των απαραίτητων δεδομένων (θέση παρατηρητή, ώρα παρατήρησης κλπ).

Πριν ξεκινήσει η σύνδεση στο telnet, ο χρήστης πρέπει να [ενεργοποιήσει το telnet client](#). Για την σύνδεση του υπολογιστή με το HORIZONS System μέσω telnet αρκεί να εκτελεστεί η παρακάτω εντολή στο Command Prompt:

```
telnet horizons.jpl.nasa.gov 6775
```

Αν η σύνδεση γίνει σωστά, εμφανίζεται το παρακάτω παράθυρο:

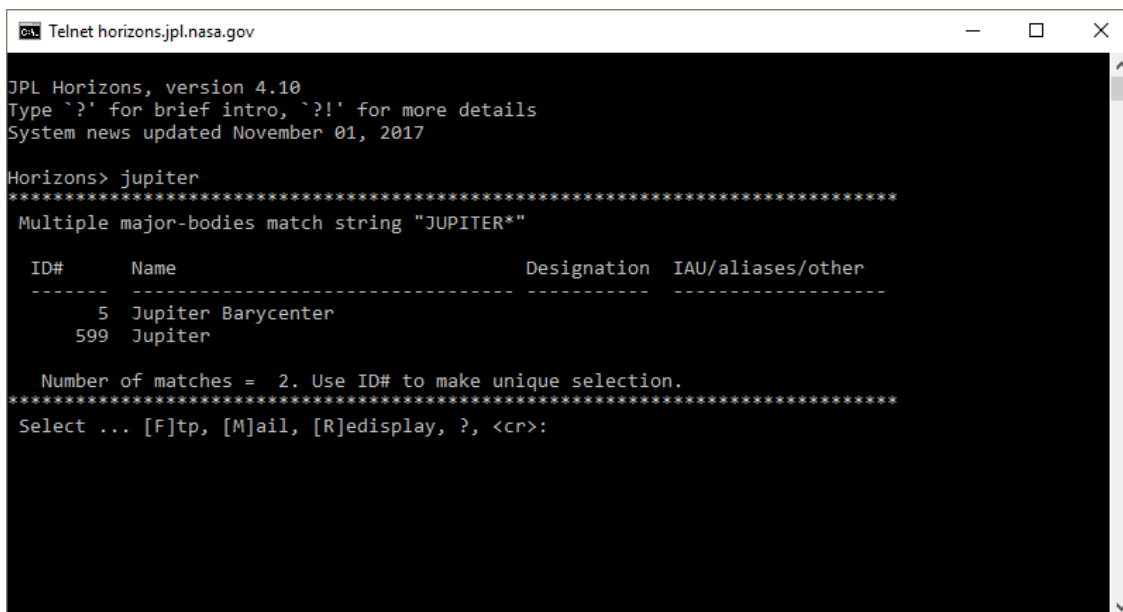


```
Telnet horizons.jpl.nasa.gov

JPL Horizons, version 4.10
Type `?' for brief intro, `?!' for more details
System news updated November 01, 2017

Horizons>
```

Καθώς δεν είναι ξεκάθαρο τι ζητάει το HORIZONS, πρέπει να χρησιμοποιηθεί η εντολή '?'. Να σημειωθεί ότι δεν θα γίνει επεξήγηση του telnet interface, και των δυνατοτήτων του, αλλά μόνο των σημείων που χρησιμοποιήθηκαν στο παρον project. Ακολουθεί μία σειρά εντολών που θα καταλήξει στα ζητούμενα δεδομένα, δηλαδή το ephemeris με στήλες ημερομηνία, ώρα, azimuth και elevation. Το ουράνιο σώμα που θα μελετηθεί είναι ο Jupiter (Δίας). Εισάγουμε Jupiter στο HORIZONS.



```
Telnet horizons.jpl.nasa.gov

JPL Horizons, version 4.10
Type `?' for brief intro, `?!' for more details
System news updated November 01, 2017

Horizons> jupiter
*****
Multiple major-bodies match string "JUPITER*"

  ID#      Name                      Designation  IAU/aliases/other
  -----
      5  Jupiter Barycenter
     599  Jupiter

Number of matches = 2. Use ID# to make unique selection.
*****
Select ... [F]tp, [M]ail, [R]edisplay, ?, <cr>:
```

Στο σημείο αυτό μπορούμε να επιλέξουμε μεταξύ του Δία και του βαρύκεντρου του Δία. Επιλέγουμε τον Δία, πληκτρολογώντας 599.

```
Telnet horizons.jpl.nasa.gov

Horizons> 599
*****
Revised: Sep 29, 2017          Jupiter          599

PHYSICAL DATA (updated 2009-Jan-28):
Mass (10^24 kg)      = 1898.13+-0.19   Density (g/cm^3)    = 1.326
Equat. radius (1 bar) = 71492+-4 km    Polar radius (km)   = 66854+-10
Volumetric mean radius = 69911+-6 km   Flattening           = 0.06487

Rotation period      = 9h 55m 29.685s   Rot. rate(10^-4 rad/s) = 1.75865
m = w^2a^3/GM        = 0.089195        Hydrostatic flat., fh = 0.06509
Inferred rot. period = 9.894+-0.02 hr   ks = 3*J2/m          = 0.494
Mom. of inert. I/MRo^2 = 0.254         I/MRo^2 (upper bound) = 0.267
Rocky core mass (Mc/M) = 0.0261        Y factor (He/H ratio) = 0.18+-0.04

GM (km^3/s^2)        = 126,686,511     GM 1-sigma (km^3/s^2) = +-100
Equ. grav. ge (m/s^2) = 24.79          Pol. grav. gp (m/s^2) = 28.34

Geometric albedo     = 0.52            Visual magnitude V(1,0) = -9.40
Vis. mag. (opposition) = -2.70         Obliquity to orbit     = 3.12 deg
Sidereal orbit period = 11.862615 yr    Sidereal orbit period  = 4332.820 d
Mean daily motion    = 0.0831294 deg/d  Mean orbit velocity    = 13.0697 km/s

Atmos. temp. (1 bar) = 165+-5 K        Heat flow/mass (x10^7) = 15 erg/gm*s
Planetary Solar Const = 50.5 W/m^2     Dipole tilt/offset     = 9.6deg/0.1Rp
Escape velocity (km/s) = 59.5           Mag.dip.mom(gauss-Rp^3) = 4.2
A_roche(ice)/Rp      = 2.76            Hill's sphere rad. Rp  = 740
*****
Select ... [E]phemeris, [F]tp, [M]ail, [R]edisplay, ?, <cr>: _
```

Έπειτα, εισάγουμε:

- ‘E’ για υπολογισμό ephemeris
- ‘O’ για παρατήρηση της τροχιάς του ουράνιου σώματος
- ‘Coord’ για εισαγωγή συντεταγμένων του παρατηρητή manually
- ‘g’ για εισαγωγή συντεταγμένων σε longitude, latitude και υψόμετρο (συντεταγμένες σε μοίρες και υψόμετρο σε χιλιόμετρα)
- ‘2017-Jul-19 20:00’ για αρχική ημερομηνία & ώρα παρατήρησης στο προκαθορισμένο format
- ‘2017-Jul-20 20:00’ για τελική ημερομηνία & ώρα παρατήρησης στο προκαθορισμένο format
- ‘5m’ για χρονικό διάστημα ανανέωσης συντεταγμένων

```
Telnet horizons.jpl.nasa.gov
Observe, Elements, Vectors [o,e,v,?] : 0
Coordinate center [ <id>,coord,geo ] : coord
Cylindrical or Geodetic input [ c,g ] : g
Input units must be DEGREES and KILOMETERS --
Specify geodetic {E. Long, lat, h } : 24.067700, 35.531560, 1.4
Starting UT [ >= 1600-Feb-05 23:59 ] : 2017-Jul-19 20:00
Ending UT [ <= 2599-Dec-08 23:58 ] : 2017-Jul-20 20:00
Output interval [ex: 10m, 1h, 1d, ? ] : 5m

Current output table defaults --
Reference frame = ICRF/J2000.0
Time zone correction = UT+00:00
Time format = CAL
Time digits output = MIN
R.A. format = HMS
RA/DEC extra precision= NO
Apparent coord. type = ATRLESS
Range units = AU
Suppress range-rate = NO
Minimum elevation = -90.0
Maximum airmass = 38.0000
Rise-Transit-Set only = NO
Skip daylight = NO
Solar elong. cut-off = 0.180
Hour angle cut-off = 0.000000000
RA/DEC rate cut-off = 0.0
CSV spreadsheet output= NO
Table quantities = A

Accept default output [ cr=(y), n, ? ] :
```

Στην συνέχεια επιλεγουμε τις default ρυθμίσεις με την εντολή ‘y’. Τέλος, ζητείται η μορφή της εξόδου, και εισάγεται η παράμετρος ‘4’, που υποδηλώνει συντεταγμένες σε azimuth & elevation (λεπτομέρειες για το σύστημα συντεταγμένων που χρησιμοποιήθηκε στο κεφάλαιο της μοντελοποίησης σε MATLAB):

```
Accept default output [ cr=(y), n, ? ] : y
Select table quantities [ <#,#..>, ? ] : 4
```

Ακολουθεί η μορφή των αποτελεσμάτων:

```
Telnet horizons.jpl.nasa.gov
2017-Jul-20 13:50 *m 145.0692 43.2849
2017-Jul-20 13:55 *m 146.6003 43.8575
2017-Jul-20 14:00 *m 148.1619 44.4070
2017-Jul-20 14:05 *m 149.7540 44.9328
2017-Jul-20 14:10 *m 151.3762 45.4338
2017-Jul-20 14:15 *m 153.0280 45.9092
2017-Jul-20 14:20 *m 154.7088 46.3582
2017-Jul-20 14:25 *m 156.4179 46.7799
2017-Jul-20 14:30 *m 158.1542 47.1736
2017-Jul-20 14:35 *m 159.9165 47.5383
2017-Jul-20 14:40 * 161.7034 47.8734
2017-Jul-20 14:45 * 163.5135 48.1780
2017-Jul-20 14:50 * 165.3450 48.4516
2017-Jul-20 14:55 * 167.1959 48.6935
2017-Jul-20 15:00 * 169.0641 48.9031
2017-Jul-20 15:05 * 170.9473 49.0799
2017-Jul-20 15:10 * 172.8433 49.2236
2017-Jul-20 15:15 * 174.7493 49.3336
2017-Jul-20 15:20 * 176.6629 49.4098
2017-Jul-20 15:25 * 178.5812 49.4520
2017-Jul-20 15:30 *t 180.5016 49.4600
2017-Jul-20 15:35 * 182.4212 49.4338
2017-Jul-20 15:40 * 184.3373 49.3735
2017-Jul-20 15:45 * 186.2472 49.2793
2017-Jul-20 15:50 * 188.1482 49.1514
2017-Jul-20 15:55 * 190.0377 48.9900
2017-Jul-20 16:00 * 191.9132 48.7957
2017-Jul-20 16:05 * 193.7725 48.5689
2017-Jul-20 16:10 * 195.6133 48.3100
< Scroll & Page: space, <cr>, <b>ack, OR arrow keys. <q> ends display. > 70%
```

Το HORIZONS παρέχει την επιλογή να στείλει τα δεδομένα σε συγκεκριμένο email, ιδιαίτερα βολική λειτουργία, καθώς αρκεί ένα απλό copy-paste για την εισαγωγή των

δεδομένων στο κομμάτι που αφορά την μοντελοποίηση σε MATLAB, και κυρίως την εξαγωγή των δεδομένων που θα εισαχθούν αργότερα στον AVR.

```
>>> Select... [A]gain, [N]ew-case, [F]tp, [K]ermit, [M]ail, [R]edisplay, ? : m
Enter your Internet e-mail address [?]: 'abcd@efg.com'
''abcd@efg.com''
Confirm e-mail address [yes(<cr>),no] :
Address stored this login only ... use "email" cmd to change
Standby ... Data sent to 'abcd@efg.com'
>>> Select... [A]gain, [N]ew-case, [F]tp, [K]ermit, [M]ail, [R]edisplay, ? :
```

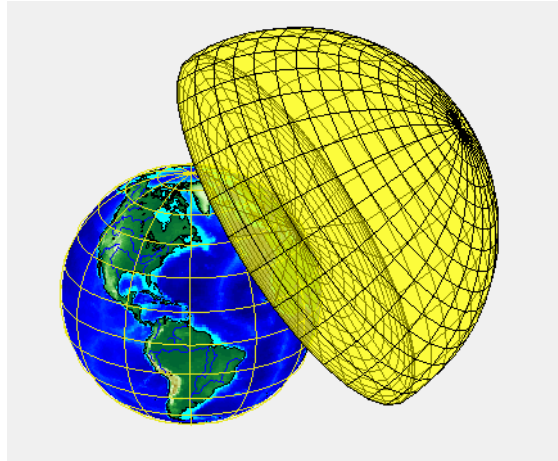
Έπειτα, αντιγράφονται τα δεδομένα από το email και εισάγονται σε αυτό το [site](#). Τα αποτελέσματα του site που προηγήθηκε, εισάγονται σε αυτό το [site](#). Αυτές οι ενέργειες γίνονται για να διευκολυνθεί το διάβασμα των αρχείων απο την MATLAB.

Επίσης, όπως φαίνεται στην εικόνα με το αποτέλεσμα του horizons, η 3^η στήλη είναι ένα σύμβολο το οποίο δεν μας είναι χρήσιμο, παρόλα αυτά η MATLAB το διαβάζει κανονικά και θεωρεί δεδομένο ότι βρίσκεται πάντα εκεί. Σε μερικές περιπτώσεις δεν υπάρχει όμως, και η MATLAB εμφανίζει σφάλματα. Δυστυχώς, αν υπάρχουν κενά ο χρήστης πρέπει να εισάγει στην στήλη αυτή σύμβολα με το χέρι.

Στο κομμάτι της μορφοποίησης των δεδομένων που στέλνονται από το HORIZONS υπάρχουν περιθώρια βελτίωσης, καθώς θα ήταν χρήσιμη η αυτοματοποίηση της παραπάνω διαδικασίας.

3 Μοντελοποίηση σε MATLAB

Η μοντελοποίηση του συστήματος που υλοποιήθηκε περιλαμβάνει δύο μέρη. Αρχικά, στο πρώτο σχήμα (δεξιά) φαίνεται η γη μαζί με ένα ημισφαίριο κεντραρισμένο σε μία περιοχή. Με αυτό το σχήμα μοντελοποιείται το σύστημα observer – celestial sphere, ή πιο συγκεκριμένα, το πως αντιλαμβάνεται ένας παρατηρητής τον ουρανό.



Χρησιμοποιώντας το HORIZONS System του Jet Propulsion Laboratory της NASA, και την δυνατότητά του αποστολής δεδομένων σε e-mail, μοντελοπήθηκε σε MATLAB η κίνηση ενός ουράνιου σώματος. Τα δεδομένα από το e-mail μεταφέρθηκαν σε ένα αρχείο .txt. Το αρχείο .txt έχει το ακόλουθο format:

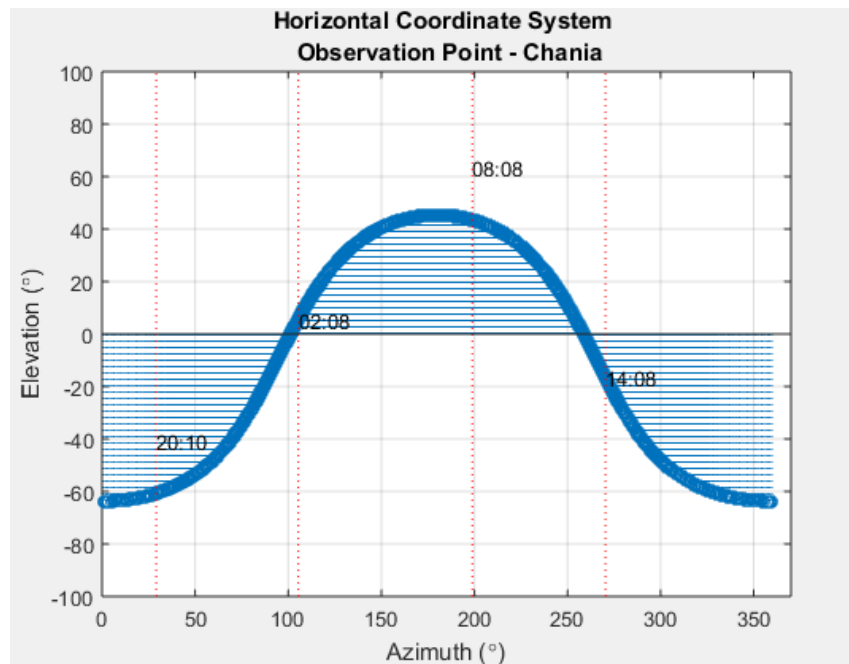
```
2017-Dec-02 20:14 m 30.9625 -60.1750
2017-Dec-02 20:16 m 31.8575 -59.9630
2017-Dec-02 20:18 m 32.7415 -59.7456
2017-Dec-02 20:20 m 33.6146 -59.5229
```

Να σημειωθεί ότι το αρχείο πρέπει να λέγεται data.txt. Σε κάθε άλλη περίπτωση, πρέπει να γίνει αλλαγή στον κώδικα της MATLAB.

Από αριστερά προς τα δεξιά, οι στήλες του αρχείου περιλαμβάνουν ημερομηνία, ώρα, ένα σύμβολο που δεν μας αφορά, την συντεταγμένη azimuth, και την συντεταγμένη elevation.

Έπειτα, τοποθετούνται σε ξεχωριστά arrays η ώρα, η συντεταγμένη azimuth και η συντεταγμένη elevation. Για την εφαρμογή που υλοποιήθηκε η ημερομηνία είναι παράμετρος που δεν λαμβάνεται υπ' όψιν, μιας και το σύστημα θα λειτουργήσει για λίγες ώρες, οπότε αρκεί η πληροφορία της ώρας.

Ακολουθεί παράδειγμα με δεδομένα από την 02/12/2017 20:08 έως την 3/12/2017 20:08, με ανανέωση συντεταγμένων κάθε 2 λεπτά.



Πιο συγκεκριμένα, το παραπάνω γράφημα περιγράφει την κίνηση ενός ουρανίου σώματος, κατα την διάρκεια ενός εικοσιτετραώρου. Έχουν τοποθετηθεί τέσσερις διαφορετικές στιγμές της ημέρας, για την καλύτερη αντίληψη της κίνησης του σώματος, αλλά και την δυνατότητα επαλήθευσης του συστήματος, σε περίπτωση ενός real time testing.

Το Horizontal Coordinate System, το οποίο χρησιμοποιείται στο σύστημα και στη μοντελοποίηση όπως αναφέρθηκε, αποτελείται από τις συντεταγμένες azimuth και elevation. Το azimuth πρακτικά, υποδηλώνει τις μοίρες που πρέπει να περιστραφεί η βάση γύρω από τον ορίζοντα του παρατηρητή, με σημείο 0° τον βορρά, και θετική φορά προς την ανατολή. Το elevation, υποδηλώνει τις μοίρες μεταξύ του ορίζοντα του παρατηρητή και του ουρανίου σώματος που μελετάται.

Συνοψίζοντας, εφόσον όλα τα χρήσιμα δεδομένα για την τροχιά του σώματος που θα μελετηθεί βρίσκονται σε πίνακες στην MATLAB, στο τέλος του κώδικα υπάρχει μία λειτουργία η οποία δημιουργεί ένα αρχείο με όνομα AVR_Input.txt, το οποίο έχει τα δεδομένα που θα φορτωθούν στον AVR, μέσω του Atmel Studio 7.

4 Υλοποίηση συστήματος

Το σύστημα αποτελείται συνολικά από τέσσερα βασικά υποσυστήμα: Το STK500 με τον ATmega16L, την περιστροφική βάση, το κύκλωμα που εξασφαλίζει ότι γίνεται σωστά η διεπαφή και η επικοινωνία των δύο προαναφερθέντων συστημάτων και τέλος το κύκλωμα RTC που εξασφαλίζει τον χρονισμό όλου του συστήματος.

4.1 Περιστροφική Βάση

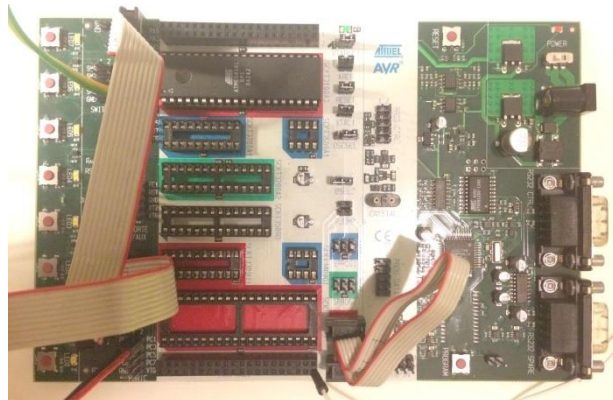
Η περιστροφική βάση AS20RS485 του εργαστηρίου που χρησιμοποιείται, έχει τα εξής χαρακτηριστικά: Η διεπαφή της γίνεται μέσω RS485, ενώ η τροφοδοσία της είναι AC 24V. Για την τροφοδοσία χρησιμοποιήθηκε κατάλληλος μετασχηματιστής που μας δόθηκε από το εργαστήριο μικροεπεξεργαστών και υλικού. Επιπλέον, η βάση επιτρέπει μέσω dip-switches τη ρύθμιση του πρωτοκόλλου επικοινωνίας καθώς και του baud rate της επικοινωνίας. Το πρωτόκολλο που επιλέχθηκε είναι το PELCO-P με baud rate 9600bps. Τα χαρακτηριστικά του πρωτοκόλλου, καθώς και ο κώδικας χειρισμού θα αναλυθούν παρακάτω.



Επίσης, στην εικόνα δεξιά, φαίνεται η θήκη που τοποθετήθηκε προκειμένου να προσαρμόζεται κινητό. Το κινητό αυτό λειτουργεί ως οπτικό μέσο στο σύστημά μας, με τρόπο που αναλύεται στη συνέχεια (παράγραφος Testing).

4.2 AVR/STK500

Ο μικροελεγκτής AVR έχει τη δυνατότητα να στέλνει και να δέχεται δεδομένα μέσω USART. Συγκεκριμένα, από το pin PD0 δέχεται δεδομένα ενώ από το PD1 στέλνει δεδομένα. Τα δεδομένα που στέλνει ο AVR είναι στάθμης TTL, το οποίο σημαίνει ότι το λογικό '0' αντιστοιχίζεται στα 0 Volts ενώ το λογικό '1' στα +5V.



Στην παραπάνω εικόνα παρουσιάζεται το STK500 και το AVR. Στη δεξιά πλευρά εφαρμόζονται η τροφοδοσία και η σύνδεση με τον υπολογιστή για τον προγραμματισμό του AVR. Πάνω αριστερά φαίνονται τα καλώδια κίτρινο-πράσινο, που προέρχονται από τα PD0 και PD1. Το PD1 σήμα είναι το transmit σήμα του UART του AVR και μέσω αυτού στέλνει ο AVR τις εντολές προς το υπόλοιπο σύστημα. Κάτω αριστερά στη φωτογραφία, φαίνεται η σύνδεση του AVR μέσω του PC0 και PC1 με το κύκλωμα του RTC (παρουσιάζεται παρακάτω) και συγκεκριμένα με τα σήματα SCL και SDA αντίστοιχα. Τέλος φαίνεται η γείωση στο κάτω μέρος της φωτογραφίας, μέσω του λευκού καλωδίου, που εξασφαλίζει κοινή τάση αναφοράς με όλα τα υπόλοιπα υποσυστήματα.

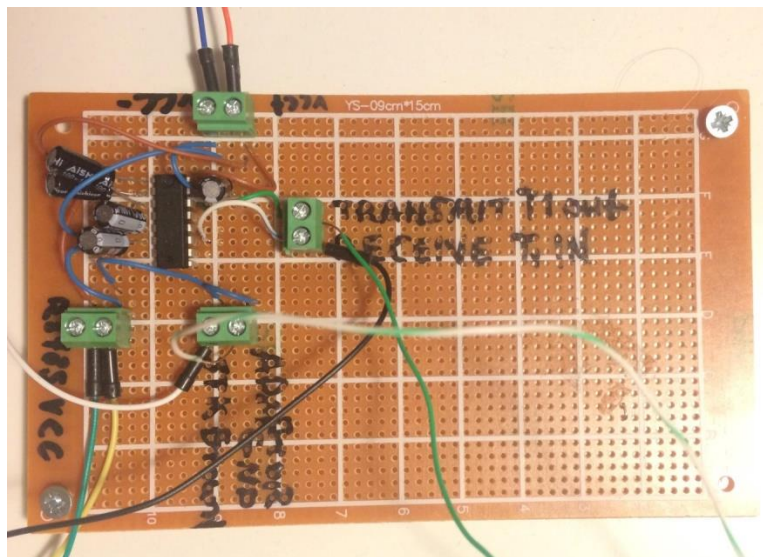
Όσον αφορά στα δεδομένα, ο AVR έχει αποθηκευμένες στη flash μνήμη του τις συντεταγμένες του στόχου, όπως αυτές αναλύονται στο πρώτο κεφάλαιο. Επιπλέον εισάγεται η ώρα έναρξης των μετρήσεων (η ώρα που αντιστοιχίζεται η πρώτη τιμή των συντεταγμένων) καθώς και το διάστημα μεταξύ διαδοχικών τιμών των συντεταγμένων. Με τα δεδομένα αυτά, και τη χρήση του κυκλώματος RTC που αναλύεται στη συνέχεια, επιτυγχάνεται η αποστολή των κατάλληλων εντολών, την κατάλληλη στιγμή, από τον AVR προς τη βάση.

4.3 Κύκλωμα διεπαφής

Όπως γίνεται κατανοητό από την περιγραφή των υποσυστημάτων του AVR και της βάσης, υπάρχει η ανάγκη μετατροπής του σήματος στάθμης TTL του AVR σε σήμα RS485, προκειμένου να ληφθεί σωστά από την βάση. Ο ρόλος του ενδιάμεσου κυκλώματος είναι να εξασφαλίσει την επικοινωνία αυτών των υποσυστημάτων. Αυτό επιτυγχάνεται μέσω δύο υποκυκλωμάτων. Το πρώτο custom PCB, το οποίο βασίζεται στο ολοκληρωμένο MAX232, μετατρέπει το σήμα από TTL σε RS232. Στη συνέχεια η έξοδος του PCB συνδέεται στο δεύτερο υποκύκλωμα, έναν RS232to485 adaptor. Μέσω της διαδικασίας αυτής, το σήμα TTL μεταφράστηκε σε σήμα RS485.

4.3.1 Custom PCB-MAX232

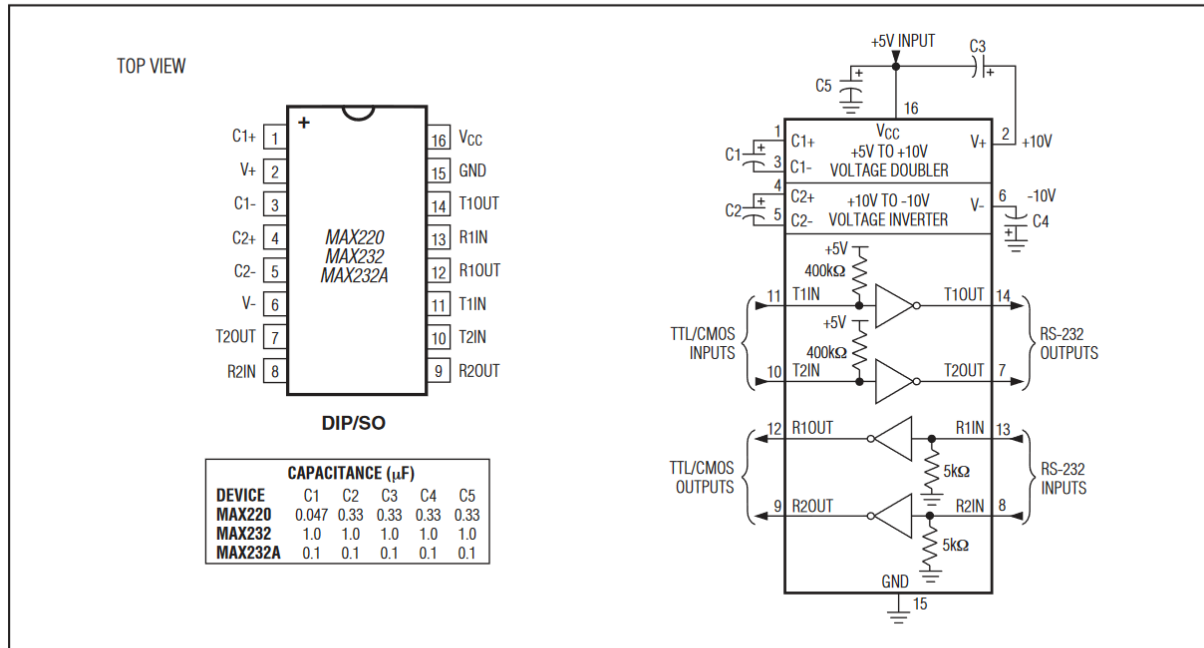
Στην παρακάτω εικόνα φαίνεται το custom PCB που υλοποιήθηκε. Διακρίνονται το ολοκληρωμένο MAX232 μαζί με τους απαραίτητους πυκνωτές για τη λειτουργία του, καθώς και τέσσερα screw connectors για τη σύνδεση των καλωδίων.



Στον πάνω screw connector συνδέεται η τάση τροφοδοσίας του MAX232, ένα τροφοδοτικό 5V DC. Στον δεξι connector συνδέεται το σήμα TTL που στέλνει ο AVR, και εξέρχεται το σήμα RS232 που οδηγείται στον adaptor. Στον κάτω δεξιά connector, συνδέονται οι δύο γειώσεις του STK500 και του RS232to485 adaptor, έτσι ώστε να έχουν όλα τα υποκυκλώματα κοινή γείωση. Τέλος στον κάτω αριστερά connector έχει μεταφερθεί η DC 5V τροφοδοσία, ώστε να συνδεθούν τα σήματα Vcc και Gnd του RS232to485 adaptor.

Η συνδεσμολογία έγινε με βάση το datasheet του MAXIM Max232:

MAX220/MAX232/MAX232A Pin Configuration and Typical Operating Circuit

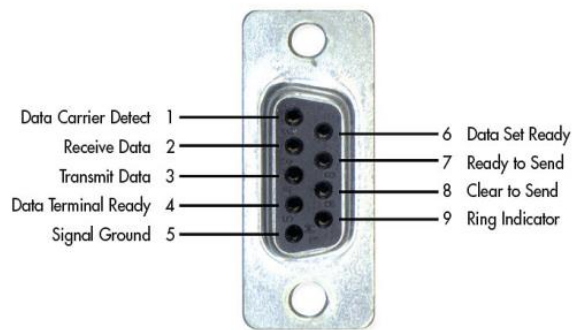


4.3.2 RS232to485 Adaptor

Στη συνέχεια παρουσιάζεται ο RS232to485 adaptor. Όπως φαίνεται και στην εικόνα, από τη μία μεριά (δεξιά) δέχεται είσοδο το σήμα RS232 που προέρχεται από το custom PCB, και από την άλλη μεριά έχει ως έξοδο τα δύο σήματα RXD (receive) και TXD (transmit) της RS485, καθώς και τα δύο pins, GND και V_{CC} που εξασφαλίζουν την απαραίτητη τροφοδοσία που απαιτεί η RS485.



Η συνδεσμολογία από τη μεριά της RS232 αποσαφηνίζεται από την παρακάτω εικόνα, σε συνδυασμό με τις εικόνες του Custom PCB του MAX232:



Τα pins που αξιοποιήθηκαν στην πράξη ήταν η γείωση (λευκό καλώδιο) και το Transmit data (πράσινο καλώδιο). Τα data προέρχονται από την έξοδο του MAX232. Στη μεριά του RS232, έχει συνδεθεί ένα νέο custom κύκλωμα, το οποίο επιτρέπει την εύκολη σύνδεση των δύο σημάτων στα αντίστοιχα pins του RS232.

Στην μεριά του RS485 χρησιμοποιούνται δύο σήματα, για την μετάδοση των δεδομένων. Για τη μετάδοση του λογικού '0' το ένα σήμα, T/R+, γίνεται +5Volts και το άλλο, T/R-, 0Volts, ενώ για τη μετάδοση του λογικού '1' γίνεται το αντίθετο. Έτσι επιτυγχάνεται half-duplex επικοινωνία, όπου μόνο στέλνονται δεδομένα προς τη βάση. Επομένως, τα τέσσερα σήματα που συνδέονται στη μεριά του RS485 του adaptor είναι: Δύο για την τροφοδοσία των 5Volts DC (V_{CC} και GND) και δύο για τη μεταφορά δεδομένων (T/R+, T/R-) που συνδέονται στη βάση.

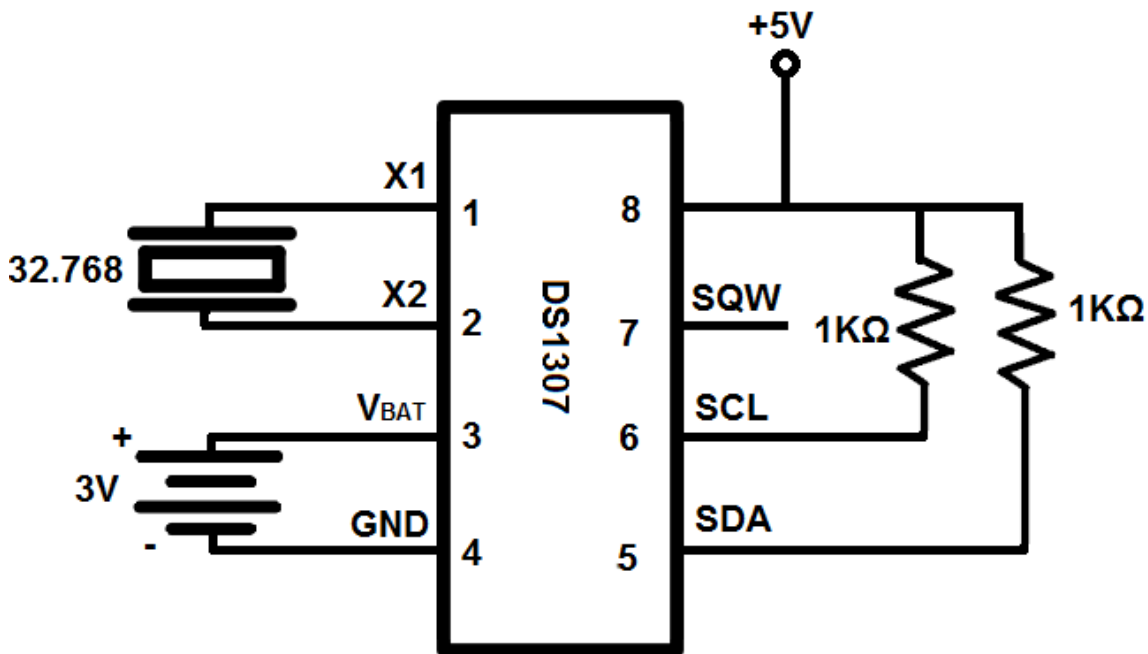
4.4 RTC κύκλωμα

Το κύκλωμα του RTC αποτελείται από δύο βασικές μονάδες: Το ολοκληρωμένο DS1307 της MAXIM και μία 16x2 LCD οθόνη.

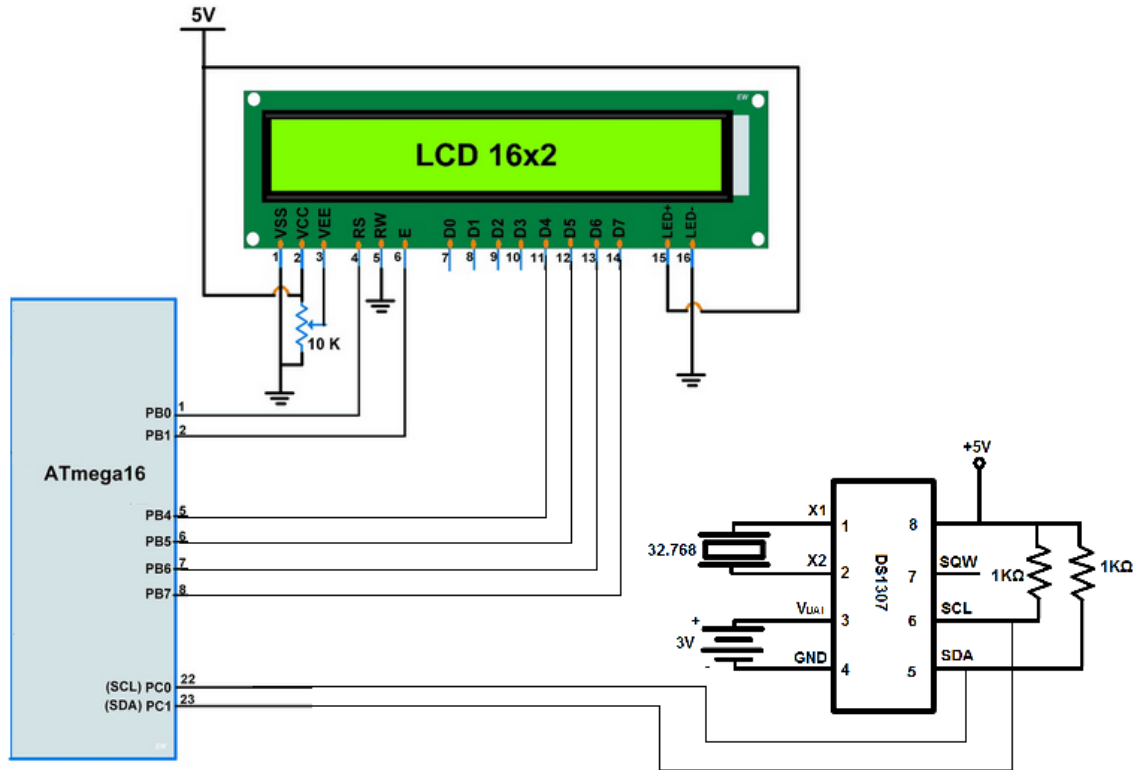
Το ολοκληρωμένο DS1307 είναι ένα real time clock chip το οποίο μετράει τόσο την ώρα, όσο και ημερολογιακά μεγέθη. Έχει συνολικά 8 pins:

Στα Pins 1 και 2 συνδέεται ένας κρύσταλλος συχνότητας 32.768kHz, ο οποίος είναι ο κατάλληλος για RTC. Στα Pins 8 και 4 συνδέεται η απαιτούμενη τροφοδοσία, 5V DC. Στα Pins 3 και 4 συνδέεται μία μπαταρία CR2035 η οποία επιτρέπει την λειτουργία του ρολογιού και στην περίπτωση όπου κλείνει η τροφοδοσία. Το ίδιο ακριβώς σύστημα επιτρέπει και στους H/Y να μην χάνουν την ώρα και την ημερομηνία κάθε φορά που αποσυνδέονται από την τροφοδοσία. Τέλος χρήσιμα είναι και τα pins 5 και 6. Μέσω αυτών αποστέλλεται η πληροφορία προς τον εκάστοτε μικροελεγκτή της ώρας και της ημερομηνίας που υπολογίζει το ολοκληρωμένο.

Το κύκλωμα του DS 1307, καθώς και τα ονόματα των pins, φαίνονται παρακάτω. Για την ορθή λειτουργία του κυκλώματος, χρειάζονται επιπλέον δύο αντιστάσεις του ενός kΩ:



Για την ευκολότερη χρήση του συστήματος, έχει τοποθετηθεί επίσης μία 16x2 LCD οθόνη. Στην οθόνη εκτυπώνονται τα δεδομένα που λαμβάνει ο AVR από το RTC. Η οθόνη χρειάζεται τροφοδοσία 5Volts, ενώ η μεταφορά των δεδομένων από τον AVR προς την οθόνη γίνεται μέσω των D4 έως D7 (αρκούν τα D4 έως D7, οπότε τα D0 έως D3 δεν χρησιμοποιούνται). Τα Pins RS, RW και E είναι pins ελέγχου, ενώ τέλος τα pins LED+ και LED- είναι για την λειτουργία του backlight της οθόνης. Συνολικά το κύκλωμα του Real time clock φαίνεται παρακάτω:

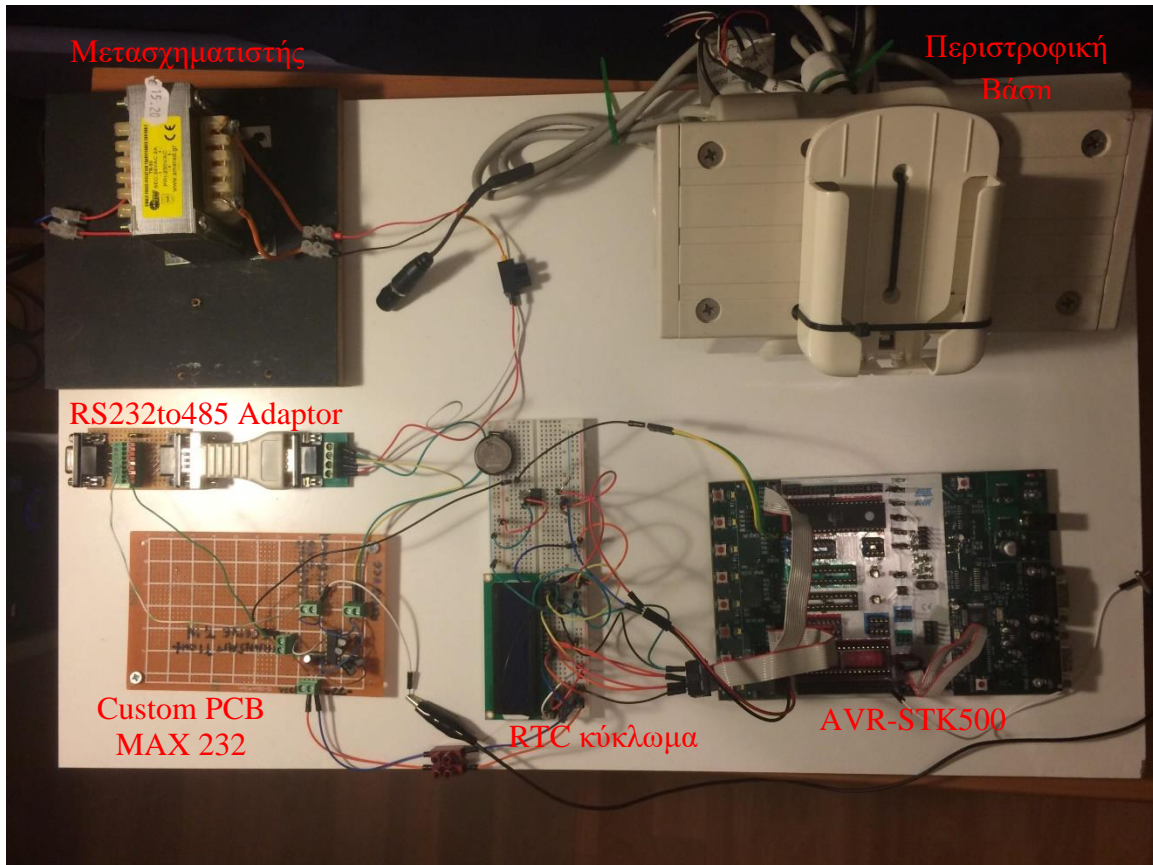


Να αναφερθεί επίσης ότι, για τη λειτουργία του RTC απαιτείται να γραφτεί σε αυτό μία φορά η τιμή της τρέχουσας ώρας. Από το σημείο εκείνο και έπειτα και αξιοποιώντας είτε την τροφοδοσία των 5V είτε την τροφοδοσία της μπαταρίας, το κύκλωμα κρατάει συνεχώς το σωστό χρόνο, τον οποίο μπορεί ο χρήστης να διαβάσει ανά πάσα στιγμή. Οι κώδικες εγγραφής και διαβάσματος του χρόνου του RTC δίνονται στην ενότητα των αναφορών.

4.5 Επισκόπηση Πλήρους Συστήματος

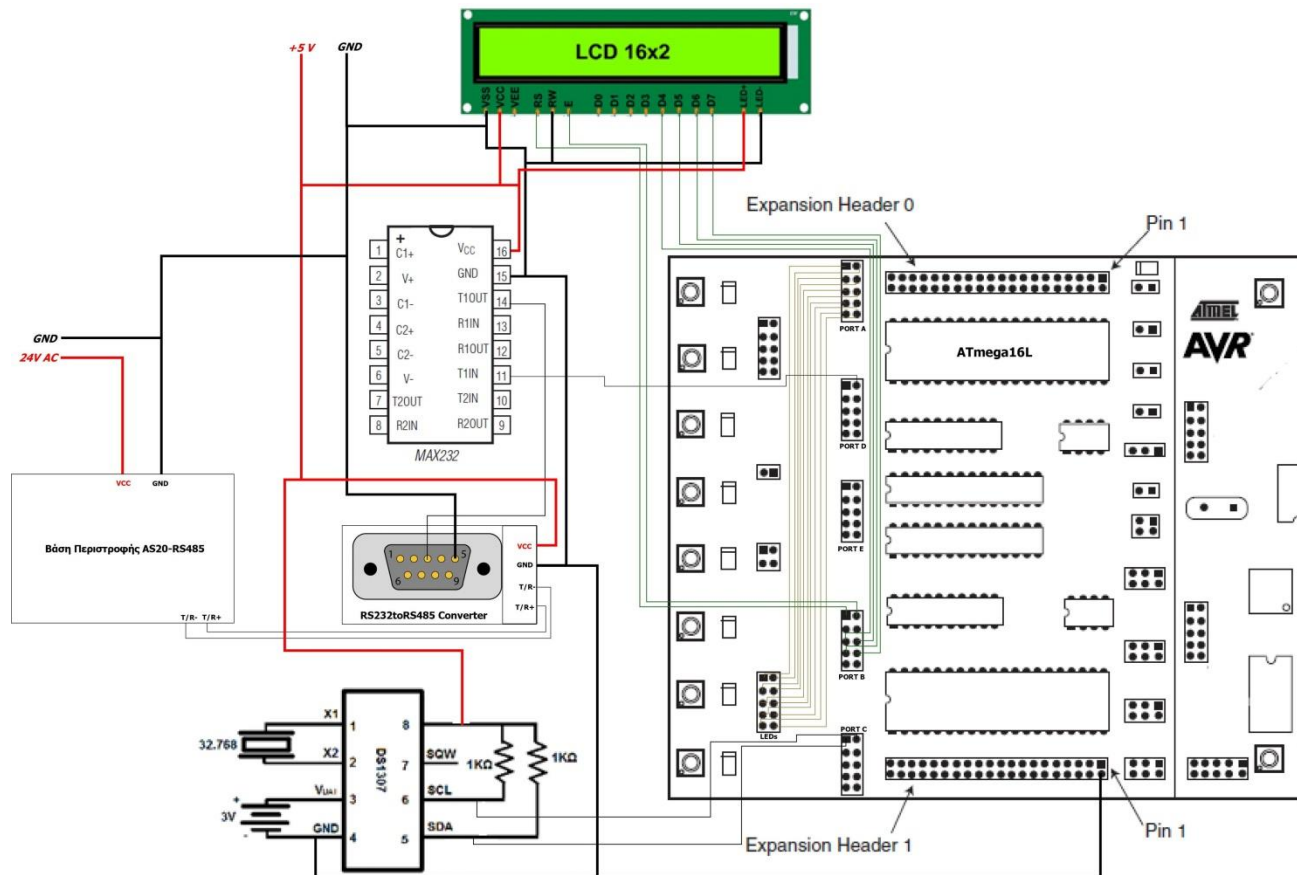
4.5.1 Εικόνα Συστήματος

Παρακάτω παρουσιάζεται η πλήρης συνδεσμολογία. Συνοψίζοντας, η ακολουθία των υποσυστημάτων έχει ως εξής: Ο AVR στέλνει, μέσω USART τις εντολές σε στάθμη TTL. Οι εντολές, και γενικότερα όλο το σύστημα συγχρονίζονται μέσω του κυκλώματος RTC. Στη συνέχεια οι εντολές λαμβάνονται από το Custom PCB και μετατρέπονται μέσω του MAX232 σε σήμα RS232. Το σήμα αυτό ακολούθως εισέρχεται στον RS232to485 adaptor, και από την έξοδο του adaptor προκύπτει η εντολή σε στάθμη RS485 πλέον, οπότε και συνδέεται με τη διεπαφή της βάσης. Τέλος αριστερά φαίνεται ο μετασχηματιστής που τροφοδοτεί την περιστροφική βάση.



4.5.2 Block Diagram Συστήματος

Το πλήρες σύστημα και οι συνδεσμολογίες μεταξύ των στοιχείων του φαίνονται αναλυτικά στο ακόλουθο διάγραμμα. Παρουσιάζονται όλα τα επιμέρους συστήματα που αναλύθηκαν προηγουμένως, καθώς και ο τρόπος που επιτυγχάνεται η σύνδεσή τους.



5 Κώδικας χειρισμού της βάσης

Για την κατανόηση του κώδικα και της λειτουργίας του είναι απαραίτητη η μελέτη του πρωτοκόλλου PELCO-P που χρησιμοποιεί η περιστροφική βάση. Η δομή του πρωτοκόλλου συνοψίζεται στον ακόλουθο πίνακα:

Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8
STX	Camera Address	Data 1	Data 2	Data 3	Data 4	ETX	Checksum

Όπως φαίνεται, μια εντολή PELCO-P αποτελείται από 8 bytes. Το πρώτο byte (πάντα 0xA0) λειτουργεί ως start byte (start of text). Το δεύτερο byte αφορά στη διεύθυνση της κάμερας, που έχει ρυθμιστεί μέσω dip switches. Τα data bytes παρουσιάζονται αναλυτικά παρακάτω. Το byte 7 (πάντα 0xAF) είναι το end byte (end of text). Το byte 8 χρησιμοποιείται για έλεγχο σφαλμάτων στη μετάδοση και υπολογίζεται ως το xOR των προηγούμενων bytes.

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Data 1	Fixed to 0	Camera On	Auto Scan On	Camera On/Off	Iris Close	Iris Open	Focus Near	Focus Far
Data 2	Fixed to 0	Zoom Wide	Zoom Tele	Tilt Down	Tilt Up	Pan Left	Pan Right	0 (for pan / tilt)
Data 3	Pan speed 00 (stop) to 3F (high speed) and 40 for Turbo							
Data 4	Tilt speed 00 (stop) to 3F (high speed)							

Από τα παραπάνω bits, αυτά που χρησιμοποιήθηκαν είναι τα bits 1 έως bit 4 του byte Data 2 που καθορίζουν τον τύπο της κίνησης που θα κάνει η βάση. Τα bytes Data 3 και Data 4 τέθηκαν στην προκαθορισμένη τιμή 0x20, καθώς η συγκεκριμένη βάση δεν υποστηρίζει διαφορετικές ταχύτητες περιστροφής. Το Data 1 byte τέθηκε σε 0x00 καθώς αφορά σε χειρισμό κάμερας, όπου δεν χρησιμοποιείται στην εφαρμογή που υλοποιείται.

Με βάση τις παραπάνω πληροφορίες, οι βασικές συναρτήσεις left(), right(), up(), down(), γίνονται ξεκάθαρες. Αναλυτικά οι εντολές φαίνονται στον κώδικα στην ενότητα «Παράρτημα».

Όσον αφορά στις συναρτήσεις που κάνουν τις αντίστοιχες κινήσεις αλλά παίρνουν ως όρισμα και τις μοίρες της εκάστοτε περιστροφικής κίνησης, να αναφερθεί ότι η λειτουργία τους βασίζεται στην εισαγωγή μιας καθυστέρησης μέσω της delay_ms(), όπου αφήνει την αντίστοιχη απλή εντολής περιστροφής ενεργή για κατάλληλο χρόνο, πρώτου κληθεί η συνάρτηση stop(). Για να καθοριστούν οι παραπάνω χρόνοι καθυστέρησης, έγινε μια διαδικασία calibration.

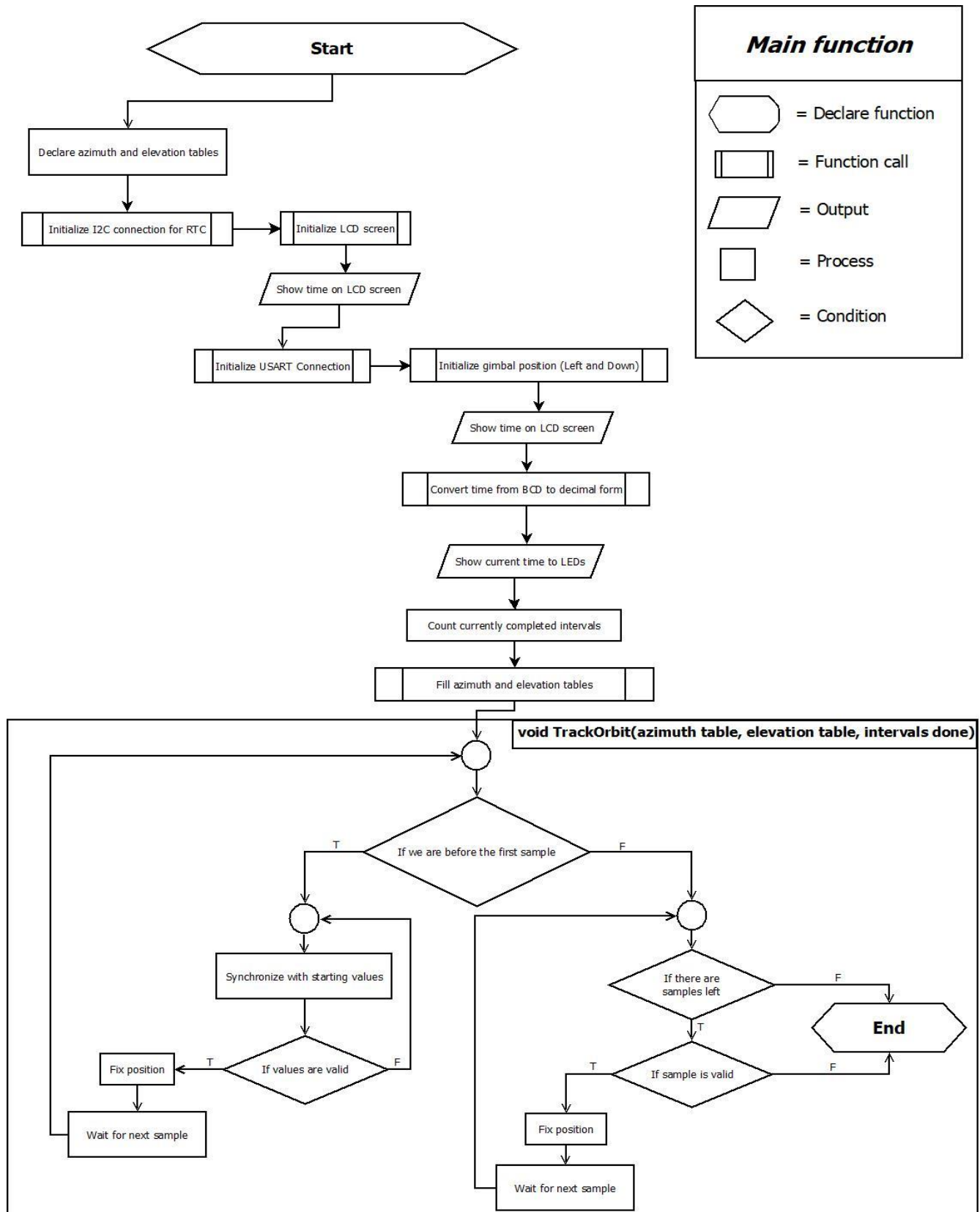
6 Calibration Βάσης

Αρχικά υλοποιήθηκε η συνάρτηση `gimbal_init()` η οποία φέρνει την περιστροφική βάση σε μια προκαθορισμένη θέση. Η θέση αυτή είναι η θέση τερματισμού κατά τη left κίνηση όσον αφορά στην περιστροφή pan, και η θέση τερματισμού κατά την down κίνηση όσον αφορά στην περιστροφή tilt. Η διαδικασία calibration έγινε ως εξής: Για την κίνηση pan, η βάση ξεκινώντας από την αρχική της θέση περιστρεφόταν με σκοπό να κάνει περιστροφή 270 μοιρών. Μέσω πολλών δοκιμών, καθορίστηκε ο χρόνος εκείνος που εξασφάλιζε την περιστροφή ακριβώς 270 μοιρών. Οι συναρτήσεις τύπου “_degree()” λαμβάνουν σαν όρισμα τις μοίρες της κίνησης, και εκτελούν ένα while-loop για την τιμή του ορίσματος. Ως εκ τούτου, μέσω της διαδικασίας που προαναφέρθηκε, στην πραγματικότητα προσδιορίστηκε ο χρόνος περιστροφής κατά μία μοίρα, ο οποίος εκτελείται 270 φορές. Ο λόγος που επιλέχθηκε η περιστροφή 270 μοιρών είναι αρχικά διότι μας παρείχε καλή οπτική επιβεβαίωση της γωνίας περιστροφής, αλλά κυρίως διότι, εφόσον εξασφαλίσουμε ότι το σφάλμα της μίας μοίρας ήταν το ελάχιστο δυνατό για μια τόσο μεγάλη γωνία, τότε σε όλες τις γωνίες μικρότερες των 270 μοιρών το σφάλμα θα είναι επίσης πολύ μικρό, ενώ για μεγαλύτερες γωνίες, το σφάλμα θα προσθεθεί το πολύ 90 φορές επιπλέον (μέχρι τη στροφή των 360 μοιρών). Ο λόγος που δεν επιλέχθηκε ως γωνία περιστροφής για το calibration οι 360 μοίρες είναι το γεγονός ότι η βάση δεν κάνει πλήρη περιστροφή 360 μοιρών, αλλά κάτι λιγότερο. Ως εκ τούτου, δεν θα μπορούσαμε να έχουμε καλή οπτική επιβεβαίωση της περιστροφής.

Με παρόμοιο τρόπο έγινε το calibration και για την κίνηση tilt, έχοντας ως αρχική θέση τις -49 μοίρες, και ελέγχοντας τον χρόνο που απαιτείται για να φτάσει η βάση σε κλίση 0 μοιρών. Η κίνηση σε αυτή την περίπτωση είναι σαφώς μικρότερη (το μέγιστο είναι 98 μοίρες αλλαγή) και συνεπώς η διαδικασία calibration σαφώς ευκολότερη.

Για την ορθή λειτουργία του συστήματος χρειάζεται επίσης η ευθυγράμμιση της βάσης περιστροφής με χρήση πυξίδας ή με χρήση της εφαρμογής που χρησιμοποιήθηκε για την αστροπαρατήρηση. Καθώς η βάση βρίσκεται στην αρχικοποιημένη θέση, αυτή τίθεται χειροκίνητα να κοιτάζει προς τον βορρά. Κατά αυτόν τον τρόπο, όλες οι περιστροφές γίνονται έκτοτε προς τα ανατολικά (δεξιά), όπως ορίζει και η συντεταγμένη azimuth.

7 Περιγραφή Κώδικα Συστήματος



Στο παραπάνω flowchart παρουσιάζεται συνοπτικά ο κύριος κώδικας λειτουργίας του συστήματος. Αρχικά, δηλώνονται οι απαραίτητοι πίνακες azimuth και elevation στους οποίους θα αποθηκευτούν οι τιμές που λαμβάνονται μέσω του HORIZONS. Στη συνέχεια αρχικοποιείται η σύνδεση I2C για την επικοινωνία του ATmega16L με το RTC, καθώς και η σύνδεση με την LCD οθόνη. Οι συνδέσεις αυτές γίνονται με κλήση συναρτήσεων που παρέχονται από έτοιμες βιβλιοθήκες. Έπειτα, και αφού εμφανιστεί η τρέχουσα ώρα στην LCD οθόνη, αρχικοποιούνται οι απαραίτητοι registers για την επίτευξη σύνδεσης με τη βάση περιστροφής χρησιμοποιώντας το πρωτόκολλο USART και περιστρέφεται η βάση στην αρχική της θέση μέσω της συνάρτησης gimbal_init(), που περιγράφεται παραπάνω. Με την επανεμφάνιση της τρέχουσας ώρας στην LCD οθόνη μετά την αρχικοποίηση θέσης της βάσης περιστροφής, θα είμαστε σε θέση να γνωρίζουμε σε πόσο χρόνο θα ξεκινήσει η παρακολούθηση βάση των δειγμάτων που λήφθηκαν απ' το HORIZONS. Αυτό θα υλοποιηθεί με κλήση συνάρτησης που υπολογίζει πόσα δείγματα έχουν περάσει από την ώρα έναρξης των μετρήσεων, χρησιμοποιώντας τις καθολικές μεταβλητές που υποδεικνύουν την ώρα έναρξης, καθώς και τα χρονικά διαστήματα που μεσολαβούν σε κάθε επόμενη μέτρηση. Με τον παραπάνω υπολογισμό και τη συμπλήρωση των πινάκων azimuth και elevation, ξεκινάει ουσιαστικά η διαδικασία εντοπισμού και παρακολούθησης του ουράνιου σώματος.

Κατά τη διαδικασία εντοπισμού της πρώτης θέσης του ουράνιου σώματος, γίνεται εισαγωγή μίας καθυστέρησης μέσω της delay_ms() για τον υπολειπόμενο χρόνο μέχρι το πρώτο δείγμα (επίτευξη συγχρονισμού συστήματος με τις καταχωρήσεις των τιμών του HORIZONS). Εάν η θέση αυτή είναι έγκυρη (το ουράνιο σώμα μπορεί να παρατηρηθεί βάση των προδιαγραφών της βάσης περιστροφής), γίνεται περιστροφή της βάσης στην τρέχουσα θέση και συνεχίζεται η παρακολούθηση. Σε αντίθετη περίπτωση, λαμβάνονται οι τιμές των επομένων δειγμάτων μέχρι να βρεθούν έγκυρες τιμές.

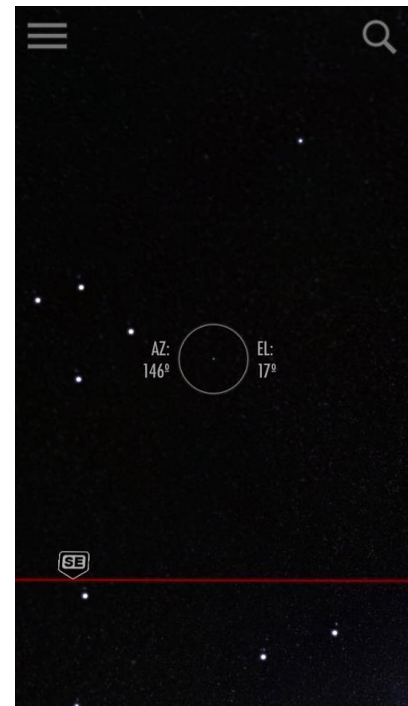
Κατά τη διαδικασία παρακολούθησης, γίνεται διόρθωση της θέσης της βάσης περιστροφής σε κάθε νέο δείγμα εφόσον οι τιμές του δείγματος είναι έγκυρες. Η διαδικασία αυτή γίνεται για όλα τα δείγματα που έχουν ληφθεί. Εάν κάποια στιγμή το δείγμα που θα ληφθεί δεν είναι έγκυρο, σημαίνει ότι το ουράνιο σώμα για το υπόλοιπο της ημέρας δε θα μπορεί να παρατηρηθεί και επομένως ολοκληρώνεται η διαδικασία με επιστροφή της βάσης στην αρχική της θέση.

8 Testing Συστήματος

Όπως αναφέρθηκε και στην εισαγωγή, η λειτουργικότητα του συστήματος επιτυγχάνεται προσθέτοντας ένα υψηλής ισχύος Laser και ένα smartphone με εφαρμογή αστροπαρατήρησης. Το Laser που χρησιμοποιήθηκε, δημιουργεί μία πράσινη δέσμη στον ουρανό, επιτρέποντας τον προσεγγιστικό προσδιορισμό του στόχου που έχει επιλεχθεί από το σύστημα.



Η εφαρμογή που χρησιμοποιήθηκε (δίνονται τα σχετικά links στην ενότητα της βιβλιογραφίας) επέτρεπε τόσο τον εντοπισμό και αναγνώριση του στόχου, όσο και την επιβεβαίωση των συντεταγμένων azimuth και elevation που είναι στραμμένο το σύστημα, χρησιμοποιώντας τους αισθητήρες του smartphone. Για τη χρήση όμως της εφαρμογής ήταν επίσης απαραίτητη η χρήση ενός προγράμματος (π.χ. TeamViewer), το οποίο μετέφερε την οθόνη του smartphone στον υπολογιστή. Ο λόγος είναι το γεγονός ότι, όπως τοποθετείται το smartphone στην ειδική θήκη πάνω στη βάση, δεν υπάρχει οπτική επαφή με την οθόνη αυτού. Δεξιά φαίνεται ένα χαρακτηριστικό στιγμιότυπο της εφαρμογής αυτής:



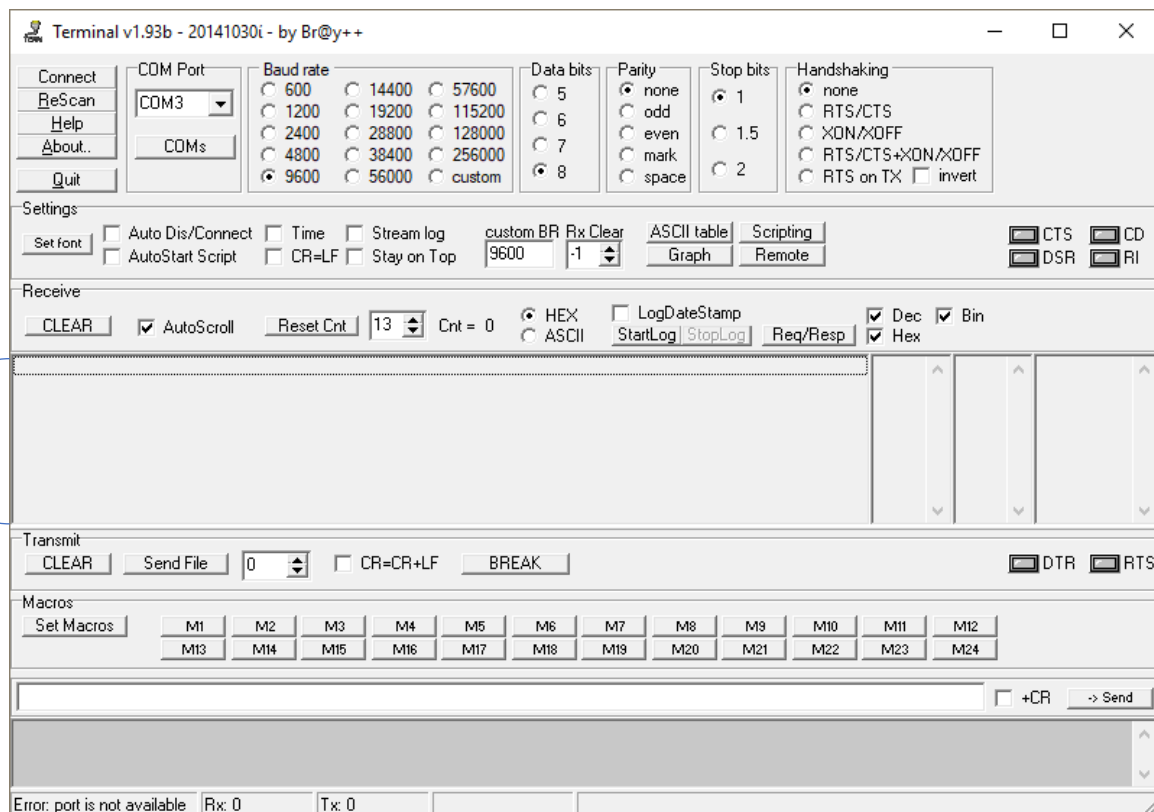
9 Προβλήματα κατά την ανάπτυξη του συστήματος & Διαδικασία Αποσφαλμάτωσης

Έχοντας μελετήσει τα πρωτόκολλα που χρησιμοποιούνται στο σύστημά, καθώς και τα datasheets ολοκληρωμένων κυκλωμάτων που ήταν απαραίτητα για την λειτουργία του συστήματος, υλοποιήθηκε η βασική μορφή των υποσυστημάτων. Έπειτα, τα υποσυστήματα συνδέθηκαν μεταξύ τους. Το σύστημα δεν λειτούργησε. Αρχικά, εξασφαλίστηκε ότι το πρόβλημα δεν οφείλεται σε κακές επαφές καλωδίων. Συνεπώς, φυσικό επακόλουθο ήταν να γίνει αποσφαλμάτωση του κάθε υποσυστήματος ξεχωριστά.

9.1 Αποσφαλμάτωση AVR

Αρχικά μελετήθηκε η αποστολή δεδομένων από τον AVR. Συμβουλευόμενοι το datasheet του AVR ακόμα μία φορά, επαληθεύθηκε ότι ο κώδικας ήταν σωστός. Έπρεπε συνεπώς να βρεθεί ένας τρόπος να οπτικοποιηθούν τα δεδομένα που στέλνονται μέσω του πρωτοκόλλου USART. Χρησιμοποιήθηκε το λογισμικό TERMINAL το οποίο είναι μία προσομοίωση για επικοινωνία μέσω σειριακής θύρας (serial port communication).

Παρακάτω φαίνεται ένα screenshot της εφαρμογής.



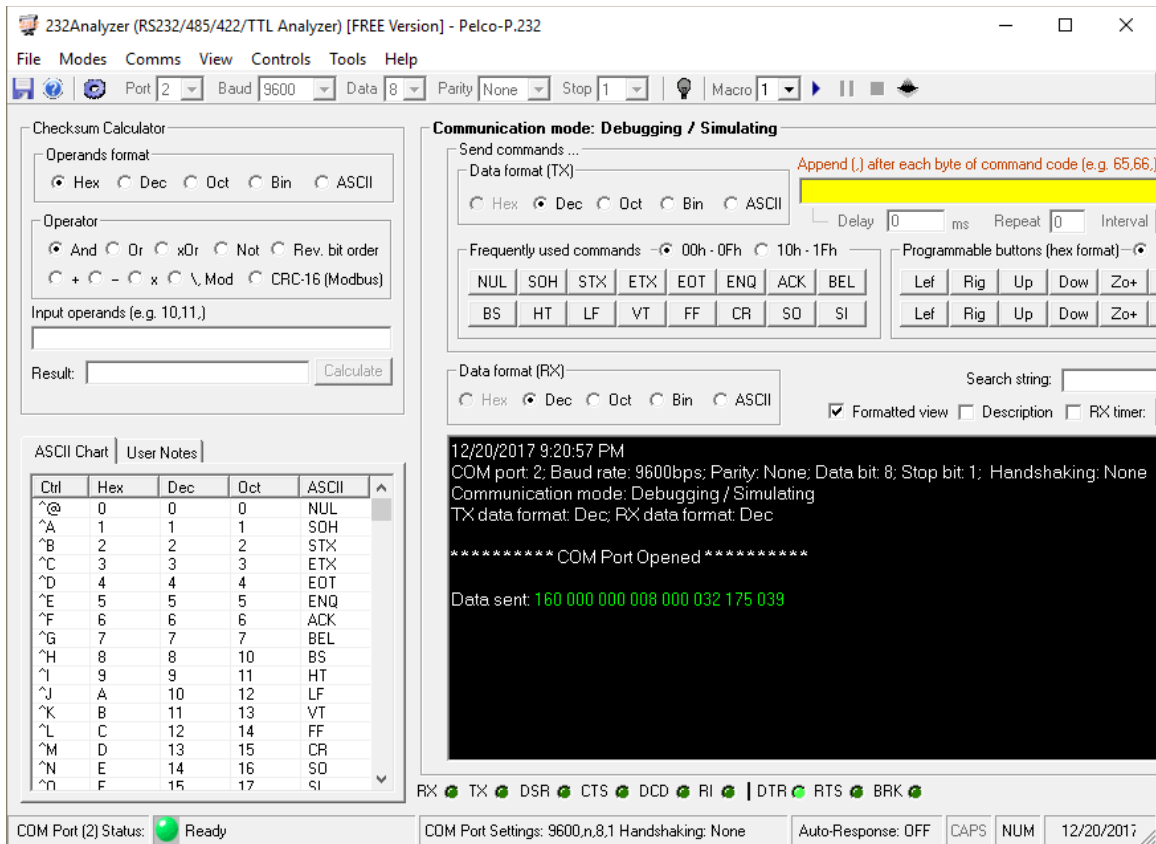
Εύκολα παρατηρεί κανείς ότι το πρόγραμμα έχει δυνατότητα σύνδεσης σε COM Ports του υπολογιστή, αλλά και ρύθμιση των βασικών στοιχείων της επικοινωνίας, όπως το Baud Rate, τα Data bits, το parity bit και τέλος το Stop Bit. Συνδέοντας λοιπόν τον υπολογιστή με την θύρα RS-232 Spare του STK500, με χρήση ενός καλωδίου USB-To-Serial και πατώντας το Connect, αναμένεται να εμφανιστούν τα ίδια δεδομένα με αυτά που στέλνει ο AVR. Τα δεδομένα που στέλνονται από τον AVR εμφανίζονται στο παράθυρο που έχει ονομαστεί «Output».

Ένα από τα προβλήματα που εμφανίστηκαν ήταν ότι κατά την αποστολή δεδομένων μέσω USART και ενώ κατά τις πρώτες δοκιμές τα δεδομένα στέλνονταν σωστά, σε μεταγενέστερες δοκιμές τα δεδομένα που στέλνονταν στο terminal ήταν λανθασμένα. Συγκεκριμένα, σε κάθε αποστολή byte, εμφανιζόταν λογικό 1 στο MSB. Αυτή η δυσλειτουργία, καθιστούσε αδύνατη την σωστή αποστολή των εντολών χειρισμού της βάσης. Ύστερα από εκτενή αποσφαλμάτωση, έγινε αντιληπτό ότι το λάθος αυτό οφείλεται στην τιμή του baud rate που είχε οριστεί στο καταχωρητή UBRR για την επικοινωνία με το πρωτόκολλο USART. Συγκεκριμένα, αυξάνοντας την τιμή του baud rate από την θεωρητικώς σωστή, το πρόβλημα διορθωνόταν, οπότε αυτή ήταν και η τελική λύση στο πρόβλημα. Μία ερμηνεία για το πρόβλημα αυτό είναι ότι το ρολόι του μικροεπεξεργαστή δεν ήταν ακριβώς 8 MHz, όπως είχε τεθεί, αλλά κάτι λίγο διαφορετικό, επηρεάζοντας έτσι και την τιμή που έπρεπε να καταχωρηθεί στον καταχωρητή UBRR, ώστε να επιτευχθεί το κατάλληλο baud rate και συνεπώς η επιτυχής επικοινωνία.

9.2 Αποσφαλμάτωση Περιστροφικής Βάσης AS20RS485

Επόμενο βήμα, ήταν η εξασφάλιση ότι η βάση είναι λειτουργική, άλλα και ότι τα δεδομένα που δέχεται από τον AVR, είναι στο σωστό format. Σε αυτό το σημείο χρησιμοποιήθηκε το λογισμικό 232Analyzer στο οποίο φορτώσαμε το απαραίτητο αρχείο για επικοινωνία με το πρωτόκολλο PELCO-P. Το πρόγραμμα 232Analyzer επιτρέπει τον έλεγχο CCTV από υπολογιστή. Δεδομένου ότι η περιστροφική βάση που χρησιμοποιήθηκε προέρχεται από CCTV σύστημα, το συγκεκριμένο λογισμικό ήταν το κατάλληλο για τον έλεγχό της.

Παρακάτω φαίνεται ένα screenshot της εφαρμογής:



Όπως και στο λογισμικό TERMINAL, ομοίως και στο 232Analyzer δίνεται η δυνατότητα ρύθμισης των βασικών παραμέτρων της επικοινωνίας (Baud Rate, Data bits, Parity bit, Stop bit). Συνδέοντας λοιπόν τον υπολογιστή με τον RS232toRS485 adaptor, χρησιμοποιώντας ένα καλώδιο USB-To-Serial, είναι δυνατή η αποστολή εντολών στις οποίες αποκρίνεται η βάση. Στο screenshot, έχει σταλεί μία εντολή left.

9.3 Ελαττωματικός Μετασχηματιστής

Από το documentation της περιστροφικής βάσης AS20RS485, φαίνεται ότι χρειάζεται τροφοδοσία 24V AC και 500 mA. Προμηθευτήκαμε λοιπόν έναν μετασχηματιστή 220V σε 24V AC, από το εργαστήριο Μικροεπεξεργαστών & Υλικού. Στις πρώτες προσπάθειες λειτουργίας της βάσης μέσω του λογισμικού 232Analyzer, ο μετασχηματιστής αρχισε να βγάζει καπνούς. Αμέσως βγήκε από την τροφοδοσία και δώθηκε πίσω στο εργαστήριο. Ο λόγος της καταστροφής του μετασχηματιστή αποδόθηκε σε αστοχία υλικού ή και στο ενδεχόμενο η περιστροφική βάση να απαιτούσε ρεύμα μεγαλύτερο των δυνατοτήτων του μετασχηματιστή, πράγμα που είχε ελεγχθεί, και συνεπώς συγκεντρώνει μικρότερη πιθανότητα. Εν τέλει, προμηθευτήκαμε έναν άλλο μετασχηματιστή, ο οποίος ήταν πλήρως λειτουργικός, και με τον οποίο πορευτήκαμε μέχρι το τέλος της υλοποίησης.

9.4 Λύση προβλήματος

Έπειτα, η μελέτη στράφηκε στο ενδιαμέσο κύκλωμα. Χρησιμοποιώντας ένα transistor tester (συσκευή που επιβεβαιώνει την ορθή λειτουργία διάφορων ηλεκτρονικών στοιχείων), βγήκε το συμπέρασμα ότι οι πυκνωτές που χρειάζονται για την οδήγηση του MAX232 λειτουργούν. Από το datasheet του MAX232 φαίνεται ότι υπάρχουν δύο μετατροπείς στάθμης σήματος. Επιβεβαιώθηκε μέσω ενός continuity test (βλ. «Χρήσιμα links για επίλυση προβλημάτων, στην ενότητα «Βιβλιογραφικές Αναφορές») ότι χρησιμοποιήθηκε χαλασμένος μετατροπέας. Ο δεύτερος ήταν λειτουργικός, οπότε έγιναν οι απαραίτητες αλλαγές στο κύκλωμα, και το σύστημα ήταν πλέον λειτουργικό.

9.5 Σχόλια για το Debugging του κώδικα

Το debugging του κώδικα του AVR έγινε αρχικά με χρήση του AVR Simulator του Atmel studio 7. Η φύση του κώδικα που αναπτύχθηκε είναι τέτοια όμως, που απαιτεί αρκετές κλήσεις των συναρτήσεων `_delay_ms()`, και `_delay_us()`. Παρατηρήθηκε ότι, όταν εμπλέκεται χρονισμός στον κώδικα, το AVR simulator δεν μπορεί να βοηθήσει. Συνεπώς, ένα μεγάλο μέρος του debugging έγινε χρησιμοποιώντας τα 8 LEDs που παρέχει το STK500.

10 Βελτιώσεις και Μελλοντικές Προσθήκες

Το σύστημα που αναλύθηκε σε αυτό το εγχειρίδιο, ενώ είναι σε πρώτη μορφή, υποστηρίζει πλήρως τον εντοπισμό και την παρακολούθηση της πορείας ουράνιων σωμάτων.

Βελτιώσεις που μπορούν να γίνουν στο άμεσο μέλλον είναι η ανάπτυξη ολοκληρωμένου κώδικα MATLAB, ο οποίος δεν χρειάζεται τα site για το text formatting, όπως αναφέρθηκε στην ενότητα της μοντελοποίησης.

Επίσης, έως τώρα η διαδικασία της εξαγωγής των δεδομένων από το HORIZONS System είναι χρονοβόρα, και απαιτεί πληροφορίες που ο χρήστης οφείλει να έχει στην διάθεσή του, όπως υψομετρο και γεωγραφικές συντεταγμένες της θέσης παρατήρησης. Επιπλέον, το φόρτωμα των συντεταγμένων στον κώδικα C του AVR γίνεται από τον χρήστη, όπως επίσης ο προσδιορισμός της χρονικής απόστασης μεταξύ 2 δειγμάτων συντεταγμένων, και η ώρα του πρώτου δείγματος. Χρήσιμο είναι να αναπτυχθούν scripts, τα οποία θα ανεξαρτητοποιήσουν πλήρως το σύστημα. Χρησιμοποιώντας ενδεχομένως ένα Raspberry Pi για να εξασφαλιστεί η φορητότητα και η ανεξαρτησία που απαιτεί ένα ενσωματωμένο σύστημα, και αυτόματο εντοπισμό συντεταγμένων μέσω του GPS του κινητού, θα φορτώνονται τα απαραίτητα δεδομένα εισόδου στο HORIZONS System, και έπειτα τα δεδομένα εξόδου θα φορτώνονται στον AVR.

Τέλος, για την περαιτέρω βελτίωση του συστήματος, υπάρχει η ιδέα προσθήκης εξωτερικής μνήμης, η οποία θα επιτρέπει την αποθήκευση μεγάλου όγκου δεδομένων – συντεταγμένων, παρέχοντας την δυνατότητα παρατήρησης στόχων για σημαντικά μεγαλύτερα χρονικά διαστήματα.

11 Ευχαριστίες

Στο σημείο αυτό οφείλουμε να ευχαριστήσουμε για την πολύτιμη βοήθειά τους αλλά και την καθοδήγησή τους στην υλοποίηση του συστήματος τον υπεύθυνο καθηγητή του μαθήματος κ. Δόλλα Απόστολο, τον υπεύθυνο του εργαστηριακού τμήματος του μαθήματος κ. Κιμιωνή Μάρκο, αλλά και τον απόφοιτο της σχολής HMMY του πολυτεχνείου κρήτης κ. Αποστολάκη Σταύρο.

12 Βιβλιογραφικές Αναφορές

Αστρικές συντεταγμένες

- Αρχική σελίδα HORIZONS System: <https://ssd.jpl.nasa.gov/?horizons>
- Ενεργοποίηση telnet client σε Windows 10: <https://www.rootusers.com/how-to-enable-the-telnet-client-in-windows-10/>
- Solar System Calculator: http://cosinekitty.com/solar_system.html

Μοντελοποίηση σε Matlab – Δημιουργία, Μορφοποίηση πινάκων

- Σβήσιμο επιπρόσθετων space απο κείμενο: <https://www.miniwebtool.com/remove-spaces/>
- Σβήσιμο space από αρχή ή το τέλος κάθε γραμμής ενός κειμένου: <https://www.miniwebtool.com/remove-leading-trailing-spaces/>
- Κώδικας MATLAB για σύστημα observer-celestial sphere: <https://www.mathworks.com/help/map/examples/plotting-a-3-d-dome-as-a-mesh-over-a-globe.html>
- Προσδιορισμός latitude & longitude οποιουδήποτε σημείου στο κόσμο: <https://www.latlong.net/>
- Προσδιορισμός υψόμετρου από την θάλασσα οποιουδήποτε σημείου στο κόσμο: <https://www.daftlogic.com/sandbox-google-maps-find-altitude.htm>

Testing Συστήματος

- Εφαρμογή για επιβεβαίωση θέσεων ουράνιων σωμάτων αλλά και των συντεταγμένων azimuth & elevation για IOS: <https://itunes.apple.com/us/app/skyview-explore-the-universe/id404990064?mt=8>
- 2η εφαρμογή σε Android: <https://play.google.com/store/apps/details?id=com.google.android.stardroid>

RTC κύκλωμα

- Οδηγίες για υλοποίηση RTC module συνοδευόμενο απο οθόνη LCD: <http://electronicwings.com/avr-atmega/real-time-clock-rtc-ds1307-interfacing-with-atmega16-32>
- 2ο link για RTC με ιδιαίτερη αναφορά στο DS1307: <http://www.learningaboutelectronics.com/Articles/DS1307-real-time-clock-RTC-circuit.php>

Ενδιάμεσο κύκλωμα

- Pinout RS-232 και RS-485: <https://opengear.zendesk.com/hc/en-us/articles/216371923-RS232-RS422-RS485-standard-DB-connector-pinout>
- Επιπλέον για RS485: <http://www.windmill.co.uk/rs485.html>
- Manual MAX232: <https://datasheets.maximintegrated.com/en/ds/DS1307.pdf>

Περιστροφική Βάση

- Manuals περιστροφικής βάσης: <http://www.copusa.com/ebproductdetail.asp?id=880>
- Δομή πρωτοκόλλου PELCO-P: <https://www.commfront.com/pages/pelco-p-protocol-tutorial>

Χρήσιμα Links για επίλυση προβλημάτων

- Forum με πληροφορίες για continuity test στον MAX232: <http://www.edaboard.com/thread223866.html>
- Συσκευή για testing διάφορων ηλεκτρονικών στοιχείων: https://www.banggood.com/DIY-M12864-Graphics-Version-Transistor-Tester-Kit-LCR-ESR-PWM-With-Case-p-997023.html?cur_warehouse=C
- Εξασφάλιση σωστής λειτουργίας MAX232: <http://www.edaboard.com/thread223866.html>
- Αρχική σελίδα λογισμικού TERMINAL: <https://hw-server.com/terminal-terminal-emulation-program-rs-232>
- Αρχική σελίδα λογισμικού 232Analyzer: <https://www.commfront.com/collections/advanced-serial-protocol-analyzer>
- Αδυναμία χρήσης STK500 ως debugger: <http://www.avrfreaks.net/forum/how-use-stk500-debug>

Χρήσιμα Datasheets-Manuals

- Manual STK500: <http://www.atmel.com/images/doc1925.pdf>
- Manual ATmega16L: <http://www.atmel.com/images/doc2466.pdf>
- Manual HORIZONS System: ftp://ssd.jpl.nasa.gov/pub/ssd/Horizons_doc.pdf

Γενικές Πηγές

Μεταπτυχιακή εργασία Αποστολάκη Στάυρου με τίτλο «Σχεδίαση και Υλοποίηση Ενσωματωμένου Συστήματος Πραγματικού Χρόνου για Ιχνηλάτηση Κατευθυντικών Κεραιών από Κινούμενο Όχημα» στη σχολή HMMY του πολυτεχνείου Κρήτης.

Παράρτημα

Παρατίθεται ο κώδικας AVR-C που αναπτύχθηκε για το σύστημα. Επίσης παρατίθεται ο κώδικας που χρησιμοποιήθηκε για το γράψιμο του RTC. Τέλος παρατίθεται ο κώδικας Matlab, που χρησιμεύει τόσο στη μοντελοποίηση όσο και στην απαιτούμενη μορφοποίηση των δεδομένων του HORIZONS για εισαγωγή στον AVR. Να σημειωθεί εκ νέου ότι οι κώδικες που αφορούν στο RTC αλλά και στην LCD οθόνη έχουν παρθεί από πηγές που εμφανίζονται στην ενότητα «Βιβλιογραφικές αναφορές». Παρόμοια, οι κώδικες Matlab, κυρίως του τμήματος της μοντελοποίησης, βασίζονται σε κώδικες που πάρθηκαν από το διαδίκτυο (υπάρχουν οι πηγές στη σχετική ενότητα) και έχουν γίνει προσθήκες και προσαρμογές για περαιτέρω λειτουργικότητα.

Οι κώδικες βρίσκονται διαθέσιμοι στο bitbucket. Για την ευκολότερη μελέτη τους, προτείνεται η λήψη τους από το παρακάτω link:

<https://bitbucket.org/embeddedsystems2017/embeddedsystems2017project/commits/all>

AVR-C

Main.c

```
/*
*****
University:      Technical University of Crete
School:         School of Electrical & Computer Engineering
Course:         Embedded Systems - HRY 411
Professor:      A. Dollas
Author:         S. Apostolakis
Modified by:    P. Giakoumakis, N. Ghionis, P.Portokalakis
Create Date:   24-11-2017
Project Name:   Automatic Detection of Celestial Bodies with Telescope
Target Device:  ATMEL AVR ATmega16L
Gimbal:        AS20-RS485
Development Platform:  AtmelStudio 7
Version:       0.2
*****
*/

// #define F_CPU 8000000L is defined in .h files included
// Define global variables
#define PAN_DEG_TIME      114000    // Pan time for one degree
#define TILT_DEG_TIME     190000    // Tilt time for one degree
#define NEXT_SAMPLE      120000    //Time interval(min)*60000ms
#define Device_Write_address 0xD0    /* Define RTC DS1307 slave address for write operation */
#define Device_Read_address 0xD1    // Make LSB bit high of slave address for read operation //
#define TimeFormat12      0x40    // Define 12 hour format //
#define AMPM              0x20
#define OBS_START_HOUR    10      //Time samples start hour
#define OBS_START_MINUTES 0       //Time samples start minute
#define TIME_INTERVAL     2       //Time between samples in minutes
#define uint8 unsigned char
#define BAUD              55      //set at 55 for a baud rate of 9600

//include needed libraries
#include <avr/io.h>
#include <avr/eeprom.h>
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
```

```

#include <string.h>
#include <stdbool.h>
#include "I2C_Master_H_file.h"
#include "LCD16x2_4Bit.h"
#include <util/delay.h>

//Declare functions
void stop(void);
void USART_Transmit(unsigned char data);
void gimbal_transmit(int byte1,int byte2,int byte3,int byte4,int byte5,int byte6,int
byte7,int byte8);
void down(void);
void up(void);
void left(void);
void left_deg(int deg);
void right(void);
void right_deg(int deg);
void down_deg(int deg);
void USART_Init(int baud);
void gimbal_init(void);
bool IsItPM(char hour_);
void RTC_Read_Clock(char read_clock_address);
void RTC_Read_Calendar(char read_calendar_address);
void ShowTime();
int HexToDec(int arg0);
void StartTime_ToLEDs();
void ConvertTimeToDecimal();
int IntervalCalculation();
void FillAz_El(int16_t* azimuth,int16_t* elevation);
void TrackOrbit(int16_t* azimuth,int16_t* elevation,int full_intervals);

/* Fill matrix with data from Horizons. The values are separated in pairs (azi-
muth,elevation) - Next pair=next value */

/* Data inserted - Orbit: Voyager I, Date: 15/12/2017, Time: 10:00 - 16:00, Time inter-
val: 2 minutes*/

int16_t eedata[] = {
    134,59,135,59,136,60,137,60,137,60,138,61,
    139,61,140,61,141,61,142,62,143,62,143,62,
    144,62,145,63,146,63,147,63,148,63,149,63,
    150,64,151,64,152,64,153,64,154,64,155,65,
    156,65,158,65,159,65,160,65,161,65,162,65,
    163,66,164,66,165,66,167,66,168,66,169,66,
    170,66,171,66,173,66,174,66,175,66,176,66,
    178,66,179,66,180,66,181,66,182,66,184,66,
    185,66,186,66,187,66,189,66,190,66,191,66,
    192,66,193,66,194,66,196,66,197,66,198,65,
    199,65,200,65,201,65,202,65,203,65,205,65,
    206,64,207,64,208,64,209,64,210,64,211,63,
    212,63,213,63,214,63,215,63,216,62,216,62,
    217,62,218,62,219,61,220,61,221,61,222,61,
    223,60,223,60,224,60,225,59,226,59,226,59,
    227,59,228,58,229,58,229,58,230,57,231,57,
    231,57,232,56,233,56,233,56,234,55,235,55,
    235,55,236,54,237,54,237,54,238,53,238,53,
    239,53,239,52,240,52,241,52,241,51,242,51,
    242,51,243,50,243,50,244,49,244,49,245,49,
    245,48,246,48,246,48,247,47,247,47,248,46,
    248,46
};

// Initialize USART connection
void USART_Init(int baud) {
    // Set baud register
    UBRRH=(baud>>8);
    UBRRL=baud;

    // Enable UART transmit
    UCSRB=(1<<TXEN) | (0<<UCSZ2);
    // Enable transmit: data 8 bytes, 1 stop bit and parity: none

```

```

        UCSRC=(1<<URSEL) | (0<<UMSEL) | (0<<UPM1) | (0<<UPM0) | (0<<USBS) |
(1<<UCSZ0) | (1<<UCSZ1);
    }

// Send data using the USART protocol
void USART_Transmit(unsigned char data)
{
    // Wait for empty transmit buffer
    while (!(UCSRA & (1<<UDRE)));

    // Put data into buffer, sends the data
    UDR = data;
}

// Initialize gimbal on starting position (pan left, tilt down)
void gimbal_init(void) {
    PORTA = 0xFB;           // Open LED2
    left();                 // Pan left
    _delay_ms(40000);       // Delay time for worst case scenario
    PORTA = 0xF7;          // Open LED3
    down();                 // Tilt down
    _delay_ms(19000);       // Delay time for worst case scenario
    PORTA = 0xFF;
}

// Transmits each byte separately
void gimbal_transmit(int byte1,int byte2,int byte3,int byte4,int byte5,int byte6,int
byte7,int byte8)
{
    USART_Transmit(byte1);
    USART_Transmit(byte2);
    USART_Transmit(byte3);
    USART_Transmit(byte4);
    USART_Transmit(byte5);
    USART_Transmit(byte6);
    USART_Transmit(byte7);
    USART_Transmit(byte8);
}

// Tilt down
void down(void)
{
    gimbal_transmit(0xA0,0x00,0x00,0x10,0x00,0x20,0xAF,0x3F);
}

// Tilt down on specific degrees
void down_deg(int deg)
{
    gimbal_transmit(0xA0,0x00,0x00,0x10,0x00,0x20,0xAF,0x3F);

    // Count down degrees
    while(deg != 0)
    {
        _delay_us(TILT_DEG_TIME);           // Tilt down one degree
        deg--;                               // One degree less
    }

    stop();
}

// Tilt up
void up(void)
{
    gimbal_transmit(0xA0,0x00,0x00,0x08,0x00,0x20,0xAF,0x27);
}

```



```

// Tilt up on specific degrees
void up_deg(int deg)
{
    gimbal_transmit(0xA0,0x00,0x00,0x08,0x00,0x20,0xAF,0x27);

    // Count down degrees
    while(deg != 0)
    {
        _delay_us(TILT_DEG_TIME);           // Tilt up one degree
        deg--;                               // One degree less
    }

    stop();
}

// Pan left
void left(void)
{
    gimbal_transmit(0xA0,0x00,0x00,0x04,0x20,0x00,0xAF,0x2B);
}

// Pan left on specific degrees
void left_deg(int deg)
{
    gimbal_transmit(0xA0,0x00,0x00,0x04,0x20,0x00,0xAF,0x2B);

    // Count down degrees
    while(deg != 0)
    {
        _delay_us(PAN_DEG_TIME);           // Pan left one degree
        deg--;                               // One degree less
    }
    stop();
}

// Pan right
void right(void)
{
    gimbal_transmit(0xA0,0x00,0x00,0x02,0x20,0x00,0xAF,0x2D);
}

// Pan right on specific degrees
void right_deg(int deg)
{
    gimbal_transmit(0xA0,0x00,0x00,0x02,0x20,0x00,0xAF,0x2D);

    // Count down degrees
    while(deg != 0)
    {
        _delay_us(PAN_DEG_TIME);           // Pan right one degree
        deg--;                               // One degree less
    }
    stop();
}

// Stop movement
void stop(void)
{
    gimbal_transmit(0xA0,0x00,0x00,0x00,0x00,0x00,0xAF,0x0F);
}

```

```

//*****RTC FUNCTIONS*****//
int second,minute,hour,day,date,month,year;

// Check if it's AM or PM
bool IsItPM(char hour_)
{
    if(hour_ & (AMPM))
        return 1;
    else
        return 0;
}

void RTC_Read_Clock(char read_clock_address)
{
    I2C_Start(Device_Write_address);    // Start I2C communication with RTC
    I2C_Write(read_clock_address);      // Write address to read
    I2C_Repeated_Start(Device_Read_address);    /* Repeated start with device
read address*/

    second = I2C_Read_Ack();            // Read second
    minute = I2C_Read_Ack();            // Read minute
    hour = I2C_Read_Nack();              // Read hour
    I2C_Stop();                          // Stop i2C communication
}

void RTC_Read_Calendar(char read_calendar_address)
{
    I2C_Start(Device_Write_address);
    I2C_Write(read_calendar_address);
    I2C_Repeated_Start(Device_Read_address);

    day = I2C_Read_Ack();                // Read day
    date = I2C_Read_Ack();               // Read date
    month = I2C_Read_Ack();              // Read month
    year = I2C_Read_Nack();              // Read the year with Nack
    I2C_Stop();                          // Stop i2C communication
}

// Read time from RTC and print it on LCD screen
void ShowTime()
{
    char buffer[20];
    char* days[7]= {"Sun","Mon","Tue","Wed","Thu","Fri","Sat"};

    RTC_Read_Clock(0); /* Read the clock with second address i.e location is 0 */
    if (hour & TimeFormat12)
    {
        sprintf(buffer, "%02x:%02x:%02x  ", (hour & 0b00011111), minute, sec-
ond);
        if(IsItPM(hour))
            strcat(buffer, "PM");
        else
            strcat(buffer, "AM");
        lcd_print_xy(0,0,buffer);
    }
    else
    {
        sprintf(buffer, "%02x:%02x:%02x  ", (hour & 0b00011111), minute, sec-
ond);
        lcd_print_xy(0,0,buffer);
    }
    RTC_Read_Calendar(3);/* Read the calender with day address i.e location is 3 */
    sprintf(buffer, "%02x/%02x/%02x %3s ", date, month, year,days[day-1]);
    lcd_print_xy(1,0,buffer);
}

//*****//

```

```

// Hours, Minutes, Seconds are read at BCD format. They are converted to decimals
int HexToDec(int arg0)
{
    int LB = (arg0 & 0b00001111);
    int HB = (arg0 & 0b11110000);
    HB = HB>>4;

    return (LB + HB*10);
}

// Used primarily for testing and debugging. Lights LEDs on appropriate values corresponding to starting time (binary format)
void StartTime_ToLEDs()
{
    PORTA = 0xFF;
    _delay_ms(1000);

    PORTA = ~(hour); // Light LEDs to show hours
    _delay_ms(2000);
    PORTA = 0xFF; // Turn off LEDs
    _delay_ms(2000);
    PORTA = ~(minute); // Light LEDs to show minutes
    _delay_ms(2000);
}

// Converts time from BCD to decimal
void ConvertTimeToDecimal()
{
    hour = hour & 0b00011111;
    hour = HexToDec(hour);
    minute = HexToDec(minute);
}

// Align and count intervals
int IntervalCalculation()
{
    int cur_hour = (hour - OBS_START_HOUR); // * Hours passed since observation started*/
    int cur_min; // * Minutes passed since observation started*/

    // Find the remainder between present minutes and observation's start
    if(minute - OBS_START_MINUTES>=0){
        cur_min = (minute - OBS_START_MINUTES);
    }
    else { // example: time now: 3.35 - time when observation started: 3.45
        cur_hour--;
        cur_min = 60-abs(minute-OBS_START_MINUTES);
    }

    // Total time difference between start time and current time, to minute format
    int cur_time_sample = (cur_hour*60 + cur_min);

    // Find full intervals completed (matrix starting point)
    int full_intervals = cur_time_sample/TIME_INTERVAL;

    // Find remaining minutes in current interval
    int interval_remainder = cur_time_sample%TIME_INTERVAL;

    // Align with intervals of HORIZONS
    for(int i=0; i<(TIME_INTERVAL - interval_remainder); i++) {
        _delay_ms(60000); // wait for one minute
    }

    return full_intervals; // return full intervals
}

```

```

// Fill azimuth and elevation tables
void FillAz_El(int16_t* azimuth,int16_t* elevation)
{
    int j=0;
    for(int k=0; k<254; k=k+2) {
        // Break pairs and put them in separate matrices
        azimuth[j] = eedata[k];
        elevation[j] = eedata[k+1];
        j++;
    }
}

// Main functionality - track and follow an orbit
void TrackOrbit(int16_t* azimuth, int16_t* elevation, int full_intervals)
{
    int flag=0;

    //The sample I want to find is at full_intervals+1!!!
    //First Sample
    while(flag==0){
        if(elevation[full_intervals+1]-41 >=0) {
            PORTA = 0xFF;
            // Pan right to the azimuth degrees of the first sample
            right_deg(azimuth[full_intervals+1]);
            // Tilt up to the elevation degrees of the first sample
            up_deg(elevation[full_intervals+1]-41);
            flag=1; // First sample - done
        }
        else {
            PORTA = 0xFE; // Turn on first LED for error checking
            full_intervals++; // Go to next coordinates
        }
        _delay_ms(NEXT_SAMPLE); // wait for the next sample
    }

    // Next samples - run until we run out of coordinates

    for(int l=full_intervals+1; l<126; l++) {
        if((int)elevation[l+1]-41 >= 0) {
            // Check if elevation to be done is valid (the gimbal tilts -41 to 41 degrees)
            if(azimuth[l+1] - azimuth[l] >= 0) {
                // find remaining degrees to match next value
                right_deg(azimuth[l+1] - azimuth[l]);
            }
            else {
                left_deg(azimuth[l] - azimuth[l+1]);
            }
            if((int)elevation[l+1] - (int)elevation[l] >= 0) {
                up_deg(elevation[l+1] - elevation[l]);
            }
            else {
                down_deg(elevation[l] - elevation[l+1]);
            }
        }
        else {
            PORTA = 0x00;
            _delay_ms(5000);
            gimbal_init(); // Initialize gimbal in starting position
            PORTA = 0x00;
            break; // The target is lost
        }
        _delay_ms(NEXT_SAMPLE); // Wait for next sample
    }
}

```

```

int main(void)
{
    DDRA = 0xFF;
    PORTA = 0xFF;

    int16_t azimuth[127];
    int16_t elevation[127];

    I2C_Init();           // Initialize I2C connection (RTC)
    lcdinit();            // Initialize LCD

    ShowTime();           // Show time at start

    USART_Init(BAUD);     // Initialize USART Connection
    gimbal_init();        // Initialize gimbal position (Left and Down)

    ShowTime();           // Show time for starting interval

    ConvertTimeToDecimal();
    // For Testing, Debugging
    StartTime_ToLEDs();

    // Count completed intervals
    int full_intervals = IntervalCalculation();

    // Fill azimuth and elevation tables
    FillAz_El(&azimuth[127], &elevation[127]);

    // Start tracking
    TrackOrbit(&azimuth[127], &elevation[127], full_intervals);

    while(1) { }
}

```

RTC_DS1307_Write.c

```
/*
 * ATmega 16/32 interfacing with RTC DS1307
 * http://www.electronicwings.com
 */

#include <avr/io.h>
#include <stdio.h>
#include "I2C_Master_H_file.h"

#define Device_Write_address 0xD0 /* Define RTC DS1307 slave address for write operation */
#define Device_Read_address 0xD1 /* Make LSB bit high of slave address for read operation */
#define hour_12_AM 0x40
#define hour_12_PM 0x60
#define hour_24 0x00

void RTC_Clock_Write(char _hour, char _minute, char _second, char AMPM)
{
    _hour |= AMPM;
    I2C_Start(Device_Write_address); /* Start I2C communication with RTC */
    I2C_Write(0); /* Write on 0 location
for second value */
    I2C_Write(_second); /* Write second value on 00
location */
    I2C_Write(_minute); /* Write minute value on 01
location */
    I2C_Write(_hour); /* Write hour value on 02 location */
    I2C_Stop(); /* Stop
I2C communication */
}

void RTC_Calendar_Write(char _day, char _date, char _month, char _year) /* function for calendar */
{
    I2C_Start(Device_Write_address); /* Start I2C communication with RTC */
    I2C_Write(3); /* Write on 3 location
for day value */
    I2C_Write(_day); /* Write day value on 03 location
*/
    I2C_Write(_date); /* Write date value on 04 location */
    I2C_Write(_month); /* Write month value on 05
location */
    I2C_Write(_year); /* Write year value on 06 location */
    I2C_Stop(); /* Stop
I2C communication */
}

int main(void)
{
    I2C_Init();
    RTC_Clock_Write(0x11, 0x59, 0x00, hour_12_PM); /* Write Hour Minute Second Format */
    RTC_Calendar_Write(0x07, 0x31, 0x12, 0x16); /* Write day date month and year */
    while(1);
}
```

I2C_Master_C_file.c

```
#include "I2C_Master_H_file.h"

void I2C_Init()
{
    TWBR = BITRATE(TWSR = 0x00);
}

uint8_t I2C_Start(char write_address)
/* I2C start function */
{
    uint8_t status;
    TWCR = (1<<TWSTA)|(1<<TWEN)|(1<<TWINT);
    while (!(TWCR & (1<<TWINT)));
    status = TWSR & 0xF8;
    if (status != 0x08)
        return 0;
    TWDR = write_address;
    TWCR = (1<<TWEN)|(1<<TWINT);
    while (!(TWCR & (1<<TWINT)));
    status = TWSR & 0xF8;
    if (status == 0x18)
        return 1;
    if (status == 0x20)
        return 2;
    else
        return 3;
}

uint8_t I2C_Repeated_Start(char read_address)
{
    uint8_t status;
    TWCR = (1<<TWSTA)|(1<<TWEN)|(1<<TWINT);
    while (!(TWCR & (1<<TWINT)));
    status = TWSR & 0xF8;
    if (status != 0x10)
        return 0;
    TWDR = read_address;
    TWCR = (1<<TWEN)|(1<<TWINT);
    while (!(TWCR & (1<<TWINT)));
    status = TWSR & 0xF8;
    if (status == 0x40)
        return 1;
    if (status == 0x20)
        return 2;
    else
        return 3;
}

void I2C_Stop()
{
    TWCR=(1<<TWSTO)|(1<<TWINT)|(1<<TWEN);
    while(TWCR & (1<<TWSTO));
}

void I2C_Start_Wait(char write_address)
{
    uint8_t status;
    while (1)
    {
        TWCR = (1<<TWSTA)|(1<<TWEN)|(1<<TWINT);
        while (!(TWCR & (1<<TWINT)));
        status = TWSR & 0xF8;
        if (status != 0x08)
            continue;
    }
}
```

```

        TWDR = write_address;
        TWCR = (1<<TWEN) | (1<<TWINT);
        while (!(TWCR & (1<<TWINT)));
        status = TWSR & 0xF8;
        if (status != 0x18 )
        {
            I2C_Stop();
            continue;
        }
        break;
    }
}

uint8_t I2C_Write(char data)
{
    uint8_t status;
    TWDR = data;
    TWCR = (1<<TWEN) | (1<<TWINT);
    while (!(TWCR & (1<<TWINT)));
    status = TWSR & 0xF8;
    if (status == 0x28)
        return 0;
    if (status == 0x30)
        return 1;
    else
        return 2;
}

int I2C_Read_Ack()
{
    TWCR=(1<<TWEN) | (1<<TWINT) | (1<<TWEA);
    while (!(TWCR & (1<<TWINT)));
    return TWDR;
}

int I2C_Read_Nack()
{
    TWCR=(1<<TWEN) | (1<<TWINT);
    while (!(TWCR & (1<<TWINT)));
    return TWDR;
}

```

I2C_Master_H_file.h

```

#ifndef I2C_MASTER_H_FILE_H_
#define I2C_MASTER_H_FILE_H_

#define F_CPU 8000000UL
#include <avr/io.h>
#include <util/delay.h>
#include <math.h>
#define SCL_CLK 100000L
#define BITRATE(TWSR) ((F_CPU/SCL_CLK)-
16)/(2*pow(4, (TWSR&((1<<TWPS0) | (1<<TWPS1))))) /* Define bit rate */

void I2C_Init();
uint8_t I2C_Start(char write_address);
uint8_t I2C_Repeated_Start(char read_address);
void I2C_Stop();
void I2C_Start_Wait(char write_address);
uint8_t I2C_Write(char data);
int I2C_Read_Ack();
int I2C_Read_Nack();

#endif

```


LCD16x2_4Bit.h

```
#ifndef LCD16X2_4BIT_H_
#define LCD16X2_4BIT_H_
#define F_CPU 8000000UL
#include <avr/io.h>
#include <util/delay.h>
#define LCD_DPRT PORTB
#define LCD_DDDR DDRB
#define LCD_RS 0
#define LCD_EN 1

void lcdcommand(unsigned char cmnd);
void lcddata(unsigned char data);
void lcdinit();
void lcd_print_xy(char row, char pos, char* str);
void lcd_print(char *str);
void lcd_clear();

#endif
```

LCD16x2_4Bit.c

```
#include "LCD16x2_4bit.h"

void lcdcommand(unsigned char cmnd)
{
    LCD_DPRT = (LCD_DPRT & 0x0f) | (cmnd & 0xf0);
    LCD_DPRT &= ~(1<<LCD_RS);
    LCD_DPRT |= (1<<LCD_EN);
    _delay_us(1);
    LCD_DPRT &= ~(1<<LCD_EN);
    _delay_us(100);

    LCD_DPRT = (LCD_DPRT & 0x0f) | (cmnd << 4);
    LCD_DPRT |= (1<<LCD_EN);
    _delay_us(1);
    LCD_DPRT &= ~(1<<LCD_EN);
    _delay_us(2000);
}

void lcddata(unsigned char data)
{
    LCD_DPRT = (LCD_DPRT & 0x0f) | (data & 0xf0);
    LCD_DPRT |= (1<<LCD_RS);
    LCD_DPRT |= (1<<LCD_EN);
    _delay_us(1);
    LCD_DPRT &= ~(1<<LCD_EN);
    _delay_us(100);

    LCD_DPRT = (LCD_DPRT & 0x0f) | (data << 4);
    LCD_DPRT |= (1<<LCD_EN);
    _delay_us(1);
    LCD_DPRT &= ~(1<<LCD_EN);
    _delay_us(2000);
}

void lcdinit()
{
    LCD_DDDR = 0xFF;
    _delay_ms(200);
    lcdcommand(0x33);
    lcdcommand(0x32);
    lcdcommand(0x28);
    lcdcommand(0x0C);
    lcdcommand(0x01);
    lcdcommand(0x82);
}
```

```

void lcd_print(char *str)
{
    unsigned char i=0;
    while (str[i] != 0)
    {
        lcddata(str[i]);
        i++;
    }
}

void lcd_print_xy(char row, char pos, char* str)
{
    if (row == 0 && pos<16)
        lcdcommand((pos & 0x0F)|0x80);
    else if (row == 1 && pos<16)
        lcdcommand((pos & 0x0F)|0xC0);
    lcd_print(str);
}

void lcd_clear()
{
    lcdcommand(0x01);
    _delay_ms(2);
}

```

MATLAB

```
% First Part
% Source: https://www.mathworks.com/help/map/examples/plotting-a-3-d-dome-as-a-mesh-over-a-globe.html

grs80 = referenceEllipsoid('grs80','km');
domeRadius = 10000; % km
domeLat = 35.52; % degrees
domeLon = 24.06; % degrees
domeAlt = 0; % km

[x,y,z] = sphere(30);
xEast = domeRadius * x;
yNorth = domeRadius * y;
zUp = domeRadius * z;
zUp(zUp < 0) = 0;
figure('Renderer','opengl')
surf(xEast, yNorth, zUp,'FaceColor','yellow','FaceAlpha',0.5)
axis equal

[xECEF, yECEF, zECEF] ...
    = enu2ecef(xEast, yNorth, zUp, domeLat, domeLon, domeAlt, grs80);
surf(xECEF, yECEF, zECEF,'FaceColor','yellow','FaceAlpha',0.5)
axis equal
figure('Renderer','opengl')
ax = axesm('globe','Geoid',grs80,'Grid','on', ...
    'GLineWidth',1,'LineStyle','-','...
    'Gcolor',[0.9 0.9 0.1],'Galtitude',100);
ax.Position = [0 0 1 1];
axis equal off
view(3)

load topo
geoshow(topo,topolegend,'DisplayType','texturemap')
demcmmap(topo)
land = shaperead('landareas','UseGeoCoords',true);
plotm([land.Lat],[land.Lon],'Color','black')
rivers = shaperead('worldrivers','UseGeoCoords',true);
plotm([rivers.Lat],[rivers.Lon],'Color','blue')

surf(xECEF, yECEF, zECEF,'FaceColor','yellow','FaceAlpha',0.5)

% Second Part: Visualization of the data to be used
% from the avr to track/find a celestial body
%
% Data taken from the JPL HORIZONS on-line solar system data
% and ephemeris computation service.

%Import data from a .txt file
data = readtable('data.txt','Delimiter',' ');

%Organize data into corresponding arrays
azimuth = table2array(data(1:end,4));
elevation = table2array(data(1:end,5));
times = table2array(data(1:end,2));

%Select key timestamps to draw to the plot
middle = times(floor(length(times)/2));
middle2 = middle{1};

azimuth_1st = azimuth(1);
elevation_1st = elevation(1);
```

```

azimuth_between_1_and_middle = azimuth(floor((1 + length(azimuth)/2)/2));
elevation_between_1_and_middle = elevation(floor((1 + length(azimuth)/2)/2));
time_between_1_and_middle = times(floor((1 + length(azimuth)/2)/2));

azimuth_middle = azimuth(floor(length(azimuth)/2));
elevation_middle = elevation(floor(length(elevation)/2));

azimuth_between_middle_and_end = azimuth(floor(length(azimuth)/2 + length(azimuth)/2/2));
elevation_between_middle_and_end = elevation(floor(length(azimuth)/2 + length(azimuth)/2/2));
time_between_middle_and_end = times(floor(length(azimuth)/2 + length(azimuth)/2/2));

azimuth_last = azimuth(end)

%Constructing the plot
figure;
stem(azimuth,elevation,':');
text(azimuth(floor(length(azimuth)/2)),elevation(floor(length(elevation)/2))+20,times(floor(length(times)/2))
text(azimuth_1st,elevation_1st+20,times(1))
text(azimuth_between_1_and_middle,elevation_between_1_and_middle,time_between_1_and_middle);
text(azimuth_between_middle_and_end,elevation_between_middle_and_end,time_between_middle_and_end);

azim_elev_text = sprintf('Horizontal Coordinate System \n Observation Point - Chania');
title(azim_elev_text);
xlabel('Azimuth (\circ)')
ylabel('Elevation (\circ)')
grid on;
axis([0 370 -100 100])

hold on;

%Plot vertical lines to clearly see selected hours of the day
vline(azimuth_1st)
vline(azimuth_middle)
vline(azimuth_last)
vline(azimuth_between_1_and_middle)
vline(azimuth_between_middle_and_end)
%figure;
%polar(azimuth,elevation);

elevation(elevation<41) = 0;
fileID = fopen('AVR_Input.txt','w');
for i=1:length(azimuth)
    if (mod(i,6) == 0 )
        fprintf(fileID,'%d,%d,\n',[round(azimuth(i)) round(elevation(i))]);
    else
        fprintf(fileID,'%d,%d,', [round(azimuth(i)) round(elevation(i))]);
    end

    if (i>126)
        break
    end
end
end

```