



ARDUINO COMPATIBLE PID BASED QUADCOPTER DEVELOPMENT



ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΑΤΡΩΝ ΤΜΗΜΑ ΦΥΣΙΚΗΣ

ΤΜΗΜΑ: ΗΕΠ

ΟΝΟΜΑΤΕΠΩΝΥΜΟ : ΜΟΝΑΧΟΠΟΥΛΟΣ ΚΩΝΣΤΑΝΤΙΝΟΣ

ΑΜ : 156

**ΜΑΘΗΜΑ : ΑΙΣΘΗΤΗΡΕΣ, ΜΙΚΡΟΕΛΕΓΚΤΕΣ ΚΑΙ ΣΥΣΤΗΜΑΤΑ
ΣΥΛΛΟΓΗΣ ΔΕΔΟΜΕΝΩΝ**

ΗΜΕΡΟΜΗΝΙΑ ΠΑΡΑΔΟΣΗΣ : -/11/2014

ΠΕΡΙΛΗΨΗ

Το *Quadcopter* είναι ένα αυτόνομο, μη επανδρωμένο ιπτάμενο ρομποτικό όχημα το οποίο ενσωματώνοντας τέσσερις περιστρεφόμενους έλικες αποτελεί πόλο ενδιαφέροντος τόσο στην έρευνα, όσο και στην ανάπτυξη πολλών εφαρμογών. Σε αυτό το *Project* το *Quadcopter* που αναπτύσσουμε είναι κατασκευασμένο από το μηδέν (*From Scratch*), ενώ οι αισθητήρες και ο εξοπλισμός έχουν επιλεχτεί προσεκτικά ώστε ο συνδυασμός τους να επιφέρει τα επιθυμητά αποτελέσματα. Η λειτουργία του στηρίζεται εξολοκλήρου στον μικροελεγκτή *Arduino Uno* ο οποίος αναλαμβάνει τον έλεγχο και την επικοινωνία των ανεξάρτητων εξαρτημάτων που συναποτελούν το *Quadcopter*.

Στόχος είναι να δημιουργηθεί μια πλήρως λειτουργική πλατφόρμα η οποία θα μπορεί να προγραμματίζεται εύκολα και θα έχει δυνατότητα αυτονομίας. Το *Quadcopter* ενσωματώνει έως και τέσσερις βαθμούς ελευθερίας, αφού διαθέτει τέσσερις κινητήρες, μέσω των οποίων χρησιμοποιώντας ανάλογο ρεύμα τροφοδότησης, ορίζεται και η ανάλογη κίνηση. Για να ολοκληρωθεί μία σταθερή και επιτυχημένη πτήση, απαιτείται η ενσωμάτωση αρκετών αισθητήρων υψηλής ακρίβειας σε συνδυασμό με ένα γρήγορο και αποτελεσματικό σύστημα ελέγχου. Τα δεδομένα των αισθητήρων πρέπει να εκτιμηθούν και ύστερα να συνδυαστούν προκειμένου να παραχθεί μία ορθή λειτουργικά πτήση. Σε αυτήν την έκθεση πραγματοποιούμε εκτενέστατη περιγραφή όλων των στοιχείων που συναποτελούν το *Quadcopter*, καθώς επίσης αναλύεται και η αλγορίθμική προσέγγιση που το κάθε στοιχείο χρησιμοποιεί. Επιπροσθέτως, για κάθε στοιχείο αναφέρεται το θεωρητικό υπόβαθρο που το περιγράφει, μέσω του οποίου στηρίζεται και η ανάπτυξη του αντίστοιχου κώδικα.

Τέλος θα αναφέρουμε τη μέθοδο που περικλείει το θεωρητικό και το προγραμματιστικό μέρος του ελεγκτή πτήσης, επισυνάπτοντας παράλληλα και το συνολικό αλγόριθμο. Ο έλεγχος ισορροπίας γίνεται με χρήση ενός *PID Controller* μέσω του οποίου εξαλείφουμε το σφάλμα κίνησης σε κάθε άξονα. Υστερα από την ολοκλήρωση των επιμέρους απαραίτητων βημάτων, συνεχίζουμε με τη διαδικασία δοκιμαστικών πτήσεων, η οποία περιλαμβάνει πτήσεις τόσο εσωτερικού, όσο και εξωτερικού χώρου. Ο συνολικός κώδικας αποτελείται από περίπου 1500 γραμμές και έχει αναπτυχθεί σε γλώσσα προγραμματισμού *C ++* μέσω του αναπτυξιακού περιβάλλοντος που μας παρέχει το *IDE* του *Arduino*.

ABSTRACT

Quadcopter is an autonomous, unmanned flying robotic system which includes four onboard rotating propellers. It consists a field of interest both in research and in development of many applications. In this project, the Quadcopter that we develop is manufactured from scratch while the sensors and the components that it is constructed from are carefully selected to work in harmony, providing the desired results. The whole process is based on Arduino Uno that is responsible for undertaking both control and communication procedure of the Quadcopter, combining the implementation of independent components.

The target of the main project is to create a full controllable and functional platform that will be appropriate to be easily programmed and have autonomous ability. The Quadcopter implements up to four degrees of freedom, as it consists of four motors, that can be controlled using the appropriate amount of current. To complete a stable and successful flight, several high accuracy sensors are coupled with a fast and efficient control system. The sensor data must be estimated and then combined to produce a functionally correct flight. In this report we perform extensive description of all the components that we used for making up the Quadcopter, and also analyze the algorithmic approach of each item. In addition, for each element the theoretical background is reported, through which the development of the corresponding code is achieved.

Finally we explicate both the theoretical and the programming approach of the flight controller, enclosing the corresponding algorithm. The balance control is done using a PID Controller through which we eliminate the error motion in each axis. After the completion of the individual necessary steps, we implement the necessary test flights, including both in interior and in exterior area. The total code consists of about 1500 lines of code and has been developed in C ++ programming language through the development environment that Arduino provides.

ΠΕΡΙΕΧΟΜΕΝΑ

| | |
|---|----|
| ΠΕΡΙΛΗΨΗ | 1 |
| ΚΑΤΑΛΟΓΟΣ ΕΙΚΟΝΩΝ | 7 |
| ΚΑΤΑΛΟΓΟΣ ΠΙΝΑΚΩΝ | 11 |
| ΛΙΣΤΑ ΤΕΧΝΙΚΩΝ ΛΕΠΤΟΜΕΡΕΙΩΝ | 12 |
| ΚΕΦΑΛΑΙΟ 1: ΕΙΣΑΓΩΓΗ | 13 |
| ΚΕΦΑΛΑΙΟ 2: ΓΕΩΜΕΤΡΙΚΗ ΑΝΑΛΥΣΗ | 14 |
| 2.1 ΣΥΣΤΗΜΑ ΣΥΝΤΕΤΑΓΜΕΝΩΝ | 14 |
| 2.2 ΚΙΝΗΜΑΤΙΚΗ & ΔΙΑΜΟΡΦΩΣΗ ΚΙΝΗΣΗΣ | 15 |
| 2.2.1 THROTTLE | 15 |
| 2.2.2 ROLL | 16 |
| 2.2.3 PITCH | 16 |
| 2.2.4 YAW | 17 |
| 2.3 ΔΙΑΜΟΡΦΩΣΗ TOY FRAME | 18 |
| ΚΕΦΑΛΑΙΟ 3 : ΥΛΙΚΑ ΜΕΡΗ ΤΟΥ QUADCOPTER | 20 |
| 3.1 Ο ΜΙΚΡΟΕΛΕΓΚΤΗΣ ARDUINO UNO | 20 |
| 3.2 ΤΟ ΠΛΑΙΣΟ ΤΟΥ QUADCOPTER | 22 |
| 3.3 ΧΑΡΑΚΤΗΡΗΣΤΙΚΑ ΠΡΟΠΕΛΩΝ | 24 |
| 3.3.1 ΙΣΟΡΡΟΠΙΣΤΗΣ ΠΡΟΠΕΛΩΝ | 25 |
| 3.4 ΧΑΡΑΚΤΗΡΗΣΤΙΚΑ ΜΟΤΕΡ | 27 |
| 3.5 ESC - ELECTRONIC SPEED CONTROLLERS | 33 |
| 3.5.1 ΑΣΦΑΛΕΙΑ | 36 |
| 3.5.2 ΣΥΝΔΕΣΜΟΛΟΓΙΑ | 36 |
| 3.5.3 ΒΑΘΜΟΝΟΜΗΣΗ | 38 |
| 3.5.4 ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ | 41 |
| 3.5.5 ΔΟΚΙΜΑΣΤΙΚΗ ΛΕΙΤΟΥΡΓΙΑ | 44 |

| | |
|---|-----|
| <i>3.6 ΜΠΑΤΑΡΙΑ ΠΟΛΥΜΕΡΟΥΣ ΛΙΘΙΟΥ</i> | 46 |
| 3.6.1 <i>ΤΑΣΗ ΜΠΑΤΑΡΙΑΣ</i> | 49 |
| 3.6.2 <i>ΧΩΡΗΤΙΚΟΤΗΤΑ ΜΠΑΤΑΡΙΑΣ</i> | 49 |
| 3.6.3 <i>ΡΥΘΜΟΣ ΕΚΦΟΡΤΙΣΗΣ</i> | 51 |
| 3.6.4 <i>ΟΡΙΑ ΜΠΑΤΑΡΙΑΣ ΚΑΙ ΕΛΕΓΧΟΣ ΑΠΟΦΟΡΤΙΣΗΣ</i> | 52 |
| 3.6.5 <i>ΦΟΡΤΙΣΗ ΜΠΑΤΑΡΙΑΣ</i> | 53 |
| <i>3.7 Ο ΔΙΑΥΛΟΣ I2C</i> | 54 |
| 3.7.1 <i>ΛΟΓΙΚΕΣ ΣΤΑΘΜΕΣ ΤΩΝ ΓΡΑΜΜΩΝ SCL ΚΑΙ SDA</i> | 56 |
| 3.7.2 <i>KYPIOI KAI YΠΟΤΕΛΕΙΣ (MASTERS AND SLAVES)</i> | 56 |
| 3.7.3 <i>ΤΟ ΦΥΣΙΚΟ ΠΡΩΤΟΚΟΛΛΟ ΤΟΥ ΔΙΑΥΛΟΥ I²C</i> | 57 |
| 3.7.4 <i>ΔΙΕΥΘΥΝΣΙΟΔΟΤΗΣΗ ΣΥΣΚΕΥΩΝ ΤΟΥ I²C ΔΙΑΥΛΟΥ</i> | 58 |
| 3.7.5 <i>THE I²C SOFTWARE PROTOCOL</i> | 59 |
| 3.7.6 <i>ΑΝΑΓΝΩΣΗ ΑΠΟ ΤΗΝ SLAVE ΣΥΣΚΕΥΗ</i> | 60 |
| <i>3.8 10DOF IMU</i> | 61 |
| 3.8.1 <i>I2C SCANNER</i> | 61 |
| 3.8.2 <i>IMU-6050</i> | 65 |
| 3.8.2.1 <i>ΑΝΑΓΝΩΣΗ ΤΙΜΩΝ</i> | 66 |
| 3.8.2.2 <i>ΘΕΩΡΙΑ ΕΠΙΤΑΝΧΥΣΙΟΜΕΤΡΟΥ</i> | 67 |
| 3.8.2.3 <i>ΘΕΩΡΙΑ ΓΥΡΟΣΚΟΠΙΟΥ</i> | 69 |
| 3.8.2.4 <i>ΣΥΝΔΙΑΣΜΟΣ ΤΙΜΩΝ ΤΟΥ MPU - 6050</i> | 70 |
| 3.8.2.5 <i>ΥΠΟΛΟΓΙΣΜΟΣ ΓΩΝΙΑΣ ΜΕ ΧΡΗΣΗ ΤΟΥ ARDUINO UNO</i> | 71 |
| 3.8.2.6 <i>DEBUGGING</i> | 84 |
| 3.8.2.7 <i>ΠΡΟ – ΒΑΘΜΟΝΟΜΗΣΗ ΤΟΥ MPU-6050</i> | 90 |
| 3.8.3 <i>ΜΑΓΝΗΤΟΜΕΤΡΟ (HMC5883L)</i> | 98 |
| 3.8.3.1 <i>ΥΠΟΛΟΓΙΣΜΟΣ ΓΩΝΙΑΣ ΜΕ ΧΡΗΣΗ ΤΟΥ ARDUINO UNO</i> | 99 |
| 3.8.3.2 <i>ΠΡΟ – ΒΑΘΜΟΝΟΜΗΣΗ ΤΟΥ HMC5883L</i> | 108 |

| | |
|---|------------|
| <i>3.8.4 ΒΑΡΟΜΕΤΡΟ (MS5611)</i> | 118 |
| <i> 3.8.4.1 ΥΠΟΛΟΓΙΣΜΟΣ ΥΨΟΥΣ ΜΕ ΧΡΗΣΗ ΤΟΥ ARDUINO UNO</i> | 120. |
| ΚΕΦΑΛΑΙΟ 4 : ΑΝΑΠΤΥΞΗ ΤΟΥ ΕΛΕΓΚΤΗ PID | 131 |
| <i> 4.1 ΘΕΩΡΙΑ ΕΛΕΓΚΤΗ PID</i> | <i>131</i> |
| <i> 4.1.1 ΑΝΑΛΟΓΙΚΟΣ ΟΡΟΣ (PROPORTIONAL)</i> | <i>132</i> |
| <i> 4.1.1.1 DROOP.....</i> | <i>133</i> |
| <i> 4.1.2 ΟΛΟΚΛΗΡΩΤΙΚΟΣ ΟΡΟΣ (INTEGRAL)</i> | <i>133</i> |
| <i> 4.1.3 ΠΑΡΑΓΩΓΙΚΟΣ ΟΡΟΣ (DERIVATIVE)</i> | <i>134</i> |
| <i> 4.2 ΕΠΙΡΡΟΗ ΤΟΥ PID ΣΤΟ QUADCOPTER</i> | <i>135</i> |
| <i> 4.3 ΥΛΟΠΟΙΗΣΗ PID ΜΕ ΧΡΗΣΗ ΤΟΥ ARDUINO UNO</i> | <i>137</i> |
| <i> 4.4 ΑΛΛΗΛΕΠΙΔΡΑΣΗ ΜΟΤΕΡ - ΕΛΕΓΚΤΗ</i> | <i>146</i> |
| ΚΕΦΑΛΑΙΟ 5 : ΤΗΛΕΧΕΙΡΙΣΜΟΣ | 151 |
| <i> 5.1 ΧΕΙΡΗΣΜΟΣ ΣΤΟ ΔΕΚΤΗ (QUADCOPTER)</i> | <i>151</i> |
| <i> 5.2 ΧΕΙΡΗΣΜΟΣ ΣΤΟ ΠΟΜΠΟ (PS2).....</i> | <i>166</i> |
| <i> 5.3 ΠΡΟΤΥΠΟ IEEE 802.15.4</i> | <i>184</i> |
| <i> 5.3.1 ΤΟ ΠΡΟΤΟΚΟΛΛΟ ZIGBEE</i> | <i>186</i> |
| <i> 5.3.1.1 NETWORK LAYER.....</i> | <i>188</i> |
| <i> 5.3.1.2 APPLICATION LAYER.....</i> | <i>188</i> |
| <i> 5.3.2 ΠΑΡΑΜΕΤΡΟΠΟΙΗΣΗ ΤΩΝ XBEE</i> | <i>189</i> |
| <i> 5.3.2.1 ΑΡΧΙΚΟΠΟΙΗΣΗ ΤΩΝ XBEE.....</i> | <i>191</i> |
| ΚΕΦΑΛΑΙΟ 6 : ΣΥΜΠΕΡΑΣΜΑΤΑ ΚΑΙ ΜΕΛΛΟΝΤΙΚΗ ΕΡΓΑΣΙΑ | 195 |
| <i> 6.1 ΣΥΜΠΕΡΑΣΜΑΤΑ</i> | <i>195</i> |
| <i> 6.2 ΜΕΛΛΟΝΤΙΚΗ ΕΡΓΑΣΙΑ</i> | <i>197</i> |
| ΑΝΑΦΟΡΕΣ..... | 198 |

ΚΑΤΑΛΟΓΟΣ ΕΙΚΟΝΩΝ

| | |
|---|----|
| Εικόνα 1. Συντεταγμένες του Quadcopter στο χώρο | 14 |
| Εικόνα 2. Γωνίες Euler | 15 |
| Εικόνα 3. Κίνηση Throttle.. | 16 |
| Εικόνα 4. Κίνηση Roll..... | 16 |
| Εικόνα 5. Κίνηση Pitch..... | 17 |
| Εικόνα 6. Κίνηση Yaw..... | 17 |
| Εικόνα 7. Διαμόρφωση frame «+»..... | 18 |
| Εικόνα 8. Διαμόρφωση frame «X»..... | 19 |
| Εικόνα 9. Κυκλική κατεύθυνση των προπελάρων | 19 |
| Εικόνα 10. Εγκατάσταση Arduino στο PS2 ελεγκτή | 21 |
| Εικόνα 11. Εγκατάσταση Arduino στο Quadcopter. | 21 |
| Εικόνα 12. Διάμετρος frame | 22 |
| Εικόνα 13. X -580 frame..... | 23 |
| Εικόνα 14. Τα είδη των προπελάρων | 24 |
| Εικόνα 15. Στρέψη και ροπή των προπελάρων | 24 |
| Εικόνα 16. Κάθετος ισποροποιημός προπελάρων..... | 26 |
| Εικόνα 17. Οριζόντιος ισποροποιημός προπελάρων..... | 26 |
| Εικόνα 18. Εσωτερική διάταξη των Brushless motors..... | 27 |
| Εικόνα 19. Εσωτερική διάταξη των Brushled motors..... | 28 |
| Εικόνα 20. NTM 28-30s Brushless motor..... | 30 |
| Εικόνα 21. Εγκατάσταση NTM 28-30s Brushless motor στο frame..... | 32 |
| Εικόνα 22. Κυκλική κατεύθυνση των Brushless motor..... | 33 |
| Εικόνα 23. Afro ESC 30 Amp Multi - rotor Motor Speed Controller (SimonK Firmware). | 34 |
| Εικόνα 24. Εγκατάσταση Afro ESC 30 Amp Multi - rotor Motor Speed Controller (SimonK Firmware)..... | 34 |

| | |
|--|-----|
| Εικόνα 25. Σύνδεση ESC με Brushless Motor 1/2 | 36 |
| Εικόνα 26. Σύνδεση ESC με Brushless Motor 2/2. | 37 |
| Εικόνα 27. Σύνδεση ESC με Arduino 1/2 | 37 |
| Εικόνα 28. Σύνδεση ESC με Arduino 2/2 | 37 |
| Εικόνα 29. Duty Cycle PWM σημάτων. | 37 |
| Εικόνα 30. Μορφές μπαταρίας τύπου Lipo. | 47 |
| Εικόνα 31. 3S - 35C – 3000mah DragonRed μπαταρία τύπου Lipo. | 48 |
| Εικόνα 32. Ψηφιακός μετρητής τάσης πραγματικού χρόνου. | 52 |
| Εικόνα 33. Mystery 2-3 Cells φορτιστής lipo μπαταριών | 53 |
| Εικόνα 34. Ακολουθίες Έναρξης – Λήξης. | 57 |
| Εικόνα 35. Αποστολή Byte | 57 |
| Εικόνα 36. Αποστολή Διεύθυνσης 7 bit | 58 |
| Εικόνα 37. Εγγραφή σε Slave. | 59 |
| Εικόνα 38. Ανάγνωση από Slave. | 60 |
| Εικόνα 39. Δομικά στοιχεία του MPU – 6050 6DOF sensor | 68 |
| Εικόνα 40. Στατική μορφή επιταχυνσιομέτρου | 67 |
| Εικόνα 41. Μη στατική μορφή επιταχυνσιομέτρου X άξονα | 67 |
| Εικόνα 42. Ι Μη στατική μορφή επιταχυνσιομέτρου Z άξονα | 68 |
| Εικόνα 43. Μη στατική μορφή επιταχυνσιομέτρου X,Y άξονα | 69 |
| Εικόνα 44. Μορφή γυροσκοπίου αρχικής θέσης | 70 |
| Εικόνα 45. Εκτύπωση αποτελεσμάτων debugging του MPU – 6050 στη σειριακή οθόνη | 84 |
| Εικόνα 46. Εκτύπωση αποτελεσμάτων debugging του MPU – 6050 στο Matlab | 90 |
| Εικόνα 47. Ο αισθητήρας Honeywell magnetoresistive (AMR). | 99 |
| Εικόνα 48. Μαγνητομετρο (HMC5883L) | 100 |
| Εικόνα 49. Εκτύπωση αποτελεσμάτων debugging του HMC5883L στη σειριακή οθόνη | 103 |
| Εικόνα 50. Ο αισθητήρας βαρομετρικής πίεσης MS5611-01BA | 119 |

| | |
|---|-----|
| Εικόνα 51. Σχέση βαρομετρικής πίεσης - υψομέτρου | 122 |
| Εικόνα 52. Εκτύπωση αποτελεσμάτων debugging του MS5611 στη σειριακή οθόνη | 130 |
| Εικόνα 53. Απόκριση σήματος PID ελεγκτή με χρήση μόνο του αναλογικού όρου | 132 |
| Εικόνα 54. Απόκριση σήματος PID ελεγκτή με χρήση αναλογικού - ολοκληρωτικού όρου | 133 |
| Εικόνα 55. Απόκριση σήματος PID ελεγκτή με χρήση αναλογικού - ολοκληρωτικού – παραγωγικού όρου | 134 |
| Εικόνα 56. Περιγραφή πλήρες συστήματος PID ελεγκτή | 136 |
| Εικόνα 57. PS2 ελεγκτής χειρισμού και Arduino Uno | 167 |
| Εικόνα 58. Διαμόρφωση σημάτων εξόδου PS2 ελεγκτή | 167 |
| Εικόνα 59. Περιγραφή σύνδεσης Pull – Up αντίστασης στο PS2 ελεγκτή | 168 |
| Εικόνα 60. Εκτύπωση αποτελεσμάτων debugging του PS2 ελεγκτή στη σειριακή οθόνη | 183 |
| Εικόνα 61. Διαμόρφωση λειτουργίας των ενσωματωμένων κουμπιών του PS2 ελεγκτή | 183 |
| Εικόνα 62. Κανάλια συχνοτήτων μετάδοσης της οικογενείας IEEE 802.15.4 | 185 |
| Εικόνα 63. Δομή βασικών επιπέδων του πρωτοκόλλου 802.15.4 | 186 |
| Εικόνα 64. Λογότυπο του ομίλου Zigbee | 187 |
| Εικόνα 65. Η στοίβα πρωτοκόλλων του ZigBee (NWK-APS-APF)..... | 187 |
| Εικόνα 66. Η μονάδα Xbee..... | 189 |
| Εικόνα 67. Xbee Shield..... | 190 |
| Εικόνα 68. Συνδεσμολογία Xbee σε Breadboard..... | 190 |
| Εικόνα 69. Σχηματικό συνδεσμολογίας Xbee | 191 |
| Εικόνα 70. Αναγνώριση αναπτυξιακού μέσω X-CTU | 192 |
| Εικόνα 71. Επιβεβαίωση αναγνώρισης αναπτυξιακού μέσω X-CTU | 192 |
| Εικόνα 72. Διαμόρφωση Xbee μέσω X-CTU | 193 |
| Εικόνα 73. Αναγνώριση διεύθυνσης Xbee μέσω X-CTU | 193 |
| Εικόνα 74. Τοποθέτηση διεύθυνσης Xbee μέσω X-CTU | 194 |
| Εικόνα 75. Διαδικασία δοκιμών εύρεσης ισσοροπίας του Quadcopter (1/2) | 196 |

Εικόνα 76. Διαδικασία δοκιμών εύρεσης ισσοροπίας του Quadcopter (2/2) 196

ΚΑΤΑΛΟΓΟΣ ΠΙΝΑΚΩΝ

| | |
|--|-----|
| Πίνακας 1. Χαρακτηριστικά του NTM 28-30s..... | 30 |
| Πίνακας 2. Χαρακτηριστικά της Hobbyking για το NTM 28-30s..... | 31 |
| Πίνακας 3. Χαρακτηριστικά της Hobbyking για αποδόσεις ώθησης προπελάν. | 31 |
| Πίνακας 4. Χαρακτηριστικά του Afro ESC 30 Amps..... | 35 |
| Πίνακας 5. Ηχητικές επιλογές Programming Mode των ESC | 42 |
| Πίνακας 6. Πληροφορίες I2C επικοινωνίας | 55 |
| Πίνακας 7. Χαρακτηριστικά του ενσωματομένου βαθυπερατού φίλτρου του MPU - 6050..... | 73 |
| Πίνακας 8. Κλίμακα μέγιστης Τιμής μέτρησης του MPU – 6050 | 73 |
| Πίνακας 9. Χαρακτηριστικά κόμβων διαφόρων σκοπών..... | 176 |
| Πίνακας 10. Τεχνικά χαρακτηριστικά των Xbee..... | 181 |

ΛΙΣΤΑ ΤΕΧΝΙΚΩΝ ΛΕΠΤΟΜΕΡΕΙΩΝ

| | |
|---|-----|
| <i>Listing 1.</i> ESC calibration c++ language technical details | 40 |
| <i>Listing 2.</i> ESC programming c++ language technical details | 42 |
| <i>Listing 3.</i> ESC testing c++ language technical details | 44 |
| <i>Listing 4.</i> ESC Scanner Port Report c++ language technical details | 62 |
| <i>Listing 5.</i> Gyro_accel.h Report c++ language technical details | 77 |
| <i>Listing 6.</i> Gyro_accel.cpp Report c language technical details | 78 |
| <i>Listing 7.</i> Debug.h Report c language technical details | 85 |
| <i>Listing 8.</i> Debug.cpp Report c language technical details | 86 |
| <i>Listing 9.</i> SensorDataLogging.m Report technical details | 87 |
| <i>Listing 10.</i> MPU_6050_Precalibration.ino Report c language technical details | 90 |
| <i>Listing 11.</i> MPU_6050_Control.ino Report c language technical details | 93 |
| <i>Listing 12.</i> Compass.h Report c language technical details | 103 |
| <i>Listing 13.</i> Compass.cpp Report c language technical details | 104 |
| <i>Listing 14.</i> Compass_HMC5883L.ino Report c language technical details | 107 |
| <i>Listing 15.</i> Compass_calibration.h Report c language technical details | 110 |
| <i>Listing 16.</i> Compass_calibration.cpp Report c language technical details | 111 |
| <i>Listing 17.</i> Compass_HMC5883L_Calibration.ino Report c language technical details..... | 117 |
| <i>Listing 18.</i> MS5611.h Report c language technical details | 122 |
| <i>Listing 19.</i> MS5611.cpp Report c language technical details | 124 |
| <i>Listing 20.</i> MS5611_Control.ino Report c language technical details..... | 128 |
| <i>Listing 21.</i> pid.h Report c language technical details | 139 |
| <i>Listing 22.</i> pid.cpp Report c language technical details | 140 |
| <i>Listing 23.</i> Motors.h Report c language technical details | 147 |
| <i>Listing 24.</i> Motors.cpp Report c language technical details | 143 |

| | |
|---|------------|
| <i>Listing 25. Main_Quadcopter_Code.cpp Report c language technical details</i> | 147 |
| <i>Listing 26. PS2X_lib.h Report c language technical details</i> | 161 |
| <i>Listing 27. PS2X_lib.cpp Report c language technical details</i> | 164 |
| <i>Listing 28. PS2X_QadCpterController.ino Report c language technical details</i> | 171 |

ΚΕΦΑΛΑΙΟ 1: ΕΙΣΑΓΩΓΗ

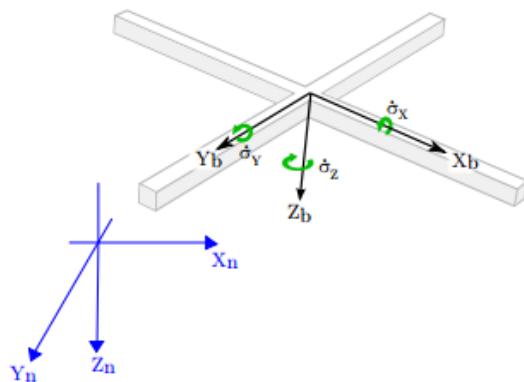
Ένα *Quadcopter* είναι ένα *RC* ιπτάμενο όχημα που εμπίπτει στην κατηγορία των πολυκόπτερων αφού διαθέτει πολλαπλούς κινητήρες. Είναι ένα αεροσκάφος με τέσσερις κύριους έλικες που μαζί παρέχουν την απαραίτητη ανέλκυση. Οι τέσσερις αυτοί κινητήρες τοποθετούνται επάνω σε ένα πλαίσιο που συχνά έχει το σχήμα σταυρού. Το *Quadcopter* μπορεί να χρησιμοποιηθεί σχεδόν αποκλειστικά ως ένα αυτόνομο μικρό αεροσκάφος ή ως ένα ελεγχόμενο από απόσταση. Το 1922 ο Γάλλος μηχανικός, *Etienne Edmond Oemichen*, κατασκεύασε και πέταξε το δεύτερό του ελικόπτερο. Αυτό το ελικόπτερο, "*Oehmichen No.2*", είχε ένα πλαίσιο σε σχήμα «X» με ένα κινητήρα σε κάθε βραχίονα. Το 1924 το μοντέλο αυτό ήταν η πρώτη επιτυχημένη πτήση για ένα χιλιόμετρο. Το *Quadcopter* είναι μια δημοφιλής ιδέα μη επανδρωμένου αεροσκάφους, λόγω των ιδιοτήτων του. Τα σημαντικότερα πλεονεκτήματα του είναι η ικανότητά να αιωρείται, και η απογείωση κάθετα. Αυτό καθιστά το *Quadcopter* χρήσιμο για πολλούς σκοπούς και επιτρέπει να λειτουργεί σε σχεδόν σε οποιοδήποτε περιβάλλον.

Ο τυπικός σχεδιασμός για ένα *Quadcopter* δεν περιέχει κινούμενα μέρη, εκτός από τους κινητήρες που τοποθετούνται στο *frame*. Δεδομένου ότι οι κινητήρες είναι στερεωμένοι στο σκελετό, ο μόνος τρόπος για να προκληθεί μια πλευρική κίνηση είναι να γείρει το σύνολο του *frame*. Για να αλλάξει η κλίση, πρέπει ένα ή περισσότερα μοτέρ να παρέχουν διαφορετική ώθηση. Για να μείνει το *Quadcopter* αμετάβλητο θα πρέπει να αυξηθεί ή να μειωθεί η ώθηση κατά το ίδιο ποσοστό στα υπόλοιπα μοτέρ. Αυτός είναι επίσης ο λόγος για τον οποίο οι κινητήρες που στρέφονται προς την ίδια κατεύθυνση είναι τοποθετημένοι ο ένας απέναντι από την άλλον σε σχήμα «X». Αντίθετα με ένα συμβατικό ελικόπτερο, το *Quadcopter* δεν έχει ουραίο στροφείο για να βοηθήσει τον έλεγχο της κίνησης περί του άξονα Z. Για τον έλεγχο αυτής της κίνησης το *Quadcopter* στηρίζεται στον έλεγχο της ροπής των κινητήρων. Το *Quadcopter* είναι κατασκευασμένο με τέσσερις κινητήρες, όπου δύο περιστρέφονται δεξιόστροφα (*CW*) και οι άλλοι δύο αριστερόστροφα (*CCW*), ώστε η ροπή να αλληλοεξουδετερώνεται. Αυτό ισχύει μόνο για το διάστημα που κάθε ζεύγος κινητήρων έχει την ίδια ροπή. Εάν τουλάχιστον ένας από τους κινητήρες έχει μειωμένη ροπή το *Quadcopter* θα αρχίσει να περιστρέφεται στον αέρα [Ref 1].

ΚΕΦΑΛΑΙΟ 2: ΓΕΩΜΕΤΡΙΚΗ ΑΝΑΛΥΣΗ

2.1 ΣΥΣΤΗΜΑ ΣΥΝΤΕΤΑΓΜΕΝΩΝ

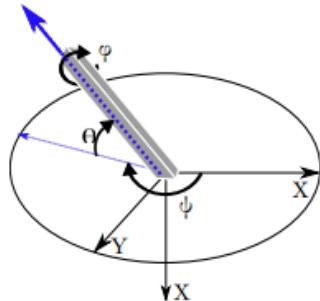
Όταν το *Quadcopter* κινείται στον τρισδιάστατο χώρο υπάρχουν δύο διαφορετικά συστήματα συντεταγμένων. Ένα είναι το σύστημα συντεταγμένων του σώματος το οποίο επηρεάζεται από τους κινητήρες, ενώ το άλλο είναι το πλαίσιο πλοήγησης, όπου επιδρούν δυνάμεις όπως η βαρύτητα. Το σύστημα συντεταγμένων του σώματος θα κινηθεί μαζί με το *Quadcopter*, ενώ το σύστημα συντεταγμένων πλοήγησης είναι το σημείο αναφοράς για το *Quadcopter*. Το σύστημα αναφοράς συντεταγμένων μπορεί να τοποθετηθεί οπουδήποτε, αλλά θα πρέπει να καθοριστεί μία φορά όταν το *Quadcopter* αρχίζει να κινείται. Μία υπόθεση που γίνεται για τα συστήματα συντεταγμένων για τον πιο εύκολο υπολογισμό, είναι να παραμελήσει τη καμπυλότητα της γης. Το *Quadcopter* έχει μια περιορισμένη περιοχή για να κινηθεί ενώ η υπόθεση που κάνουμε δεν θα επηρεάσει το αποτέλεσμα. Στα ηλεκτρονικά συστήματα πλοήγησης ο Z-άξονας είναι συνήθως στραμμένος προς τη γη. Το *Quadcopter* είναι σε θέση να περιστρέφεται γύρω από κάποιον άξονά του με μία γωνιακή ταχύτητα. Αυτή η γωνιακή ταχύτητα συμβολίζεται ως σ με ένα δείκτη για τον αντίστοιχο άξονα.



Εικόνα 1. Συντεταγμένες του *Quadcopter* στο χώρο.

Προκειμένου να πάρει τον προσανατολισμό του συναρτήσει των γωνιών, το σύστημα συντεταγμένων του *Quadcopter* πρέπει να δέχεται ως εισόδους τις συντεταγμένες που ορίζονται κατά την πλοήγηση. Ο τρόπος που γίνεται αυτό είναι με την εφαρμογή του

γωνιών του *Euler*. Υπάρχουν τρεις γωνίες *Euler*, ϕ , θ και ψ , γνωστές ως *roll*, *pitch* και *yaw*. Ο συμβολισμός αυτός χρησιμοποιείται συχνά στα ηλεκτρονικά συστήματα πτήσης [Ref 1].



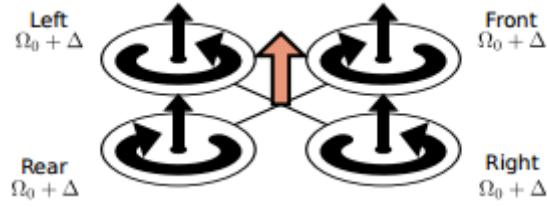
Εικόνα 2. Γωνίες Euler.

2.2 ΚΙΝΗΜΑΤΙΚΗ & ΔΙΑΜΟΡΦΩΣΗ ΚΙΝΗΣΗΣ

Για να διατηρηθεί η ισορροπία του *Quadcopter* πρέπει να λαμβάνει συνεχώς μετρήσεις από τους αισθητήρες και να κάνει προσαρμογές στην ταχύτητα του κάθε κινητήρα ώστε να διατηρήσει το επίπεδο του *frame*. Συνήθως αυτές οι προσαρμογές γίνονται αυτόνομα από ένα εξελιγμένο σύστημα ελέγχου προκειμένου να μείνει το *Quadcopter* απόλυτα εξισορροπημένο. Ένα *Quadcopter* έχει τέσσερις ελεγχόμενους βαθμούς ελευθερίας. Έτσι είναι σχετικά εύκολο να οριστούν τέσσερις διαφορετικές κινήσεις που το *Quadcopter* είναι σε θέση να κάνει προκειμένου να επιτευχθούν τα επιθυμητά αποτελέσματα. Παρακάτω θα εξεταστούν οι τέσσερις βασικές κινήσεις.

2.2.1 THROTTLE

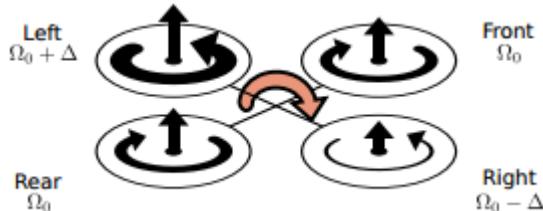
Η κίνηση αυτή είναι άμεσα συνδεδεμένη με μια γραμμική επιτάχυνση στον *Z*-άξονα. Με την αύξηση (ή μείωση) της γωνιακής ταχύτητας όλων των κινητήρων κατά το ίδιο ποσοστό, δημιουργείται μια κάθετη δύναμη που ωθεί προς τα πάνω (ή προς τα κάτω) το σώμα του *Quadcopter*. Αν αυτό αιωρείται τέλεια, αυτή η δύναμη θα προκαλέσει το *Quadcopter* να κινείται με σταθερή θέση για τους *X* και *Y* άξονες.



Εικόνα 3. Κίνηση Throttle.

2.2.2 ROLL

Αυτή η κίνηση σχετίζεται άμεσα με μια γωνιακή επιτάχυνση στον άξονα X. Με τη διατήρηση της ταχύτητας των εμπρός και πίσω μοτέρ ενώ αυξάνοντας την ταχύτητα του αριστερού μοτέρ και μειώνοντας παράλληλα την ταχύτητα του δεξιού μοτέρ κατά το ίδιο ποσοστό, το σώμα θα περιστρέφεται θετικά στον X-άξονα. Το αντίθετο αποτέλεσμα θα επιτευχθεί με τη μείωση της ταχύτητας του αριστερού μοτέρ ενώ αυξάνεται η ταχύτητα του δεξιού μοτέρ κατά την ίδια τιμή. Με μία πρώτη προσέγγιση αυτή η κίνηση θα δημιουργήσει μόνο μία γωνιακή επιτάχυνση αλλά ή ώθηση θα έχει και μια οριζόντια συνιστώσα που θα κάνει το Quadcopter να κινηθεί δεξιά ή αριστερά ανάλογα με τη φορά την γωνιακής επιτάχυνσης.

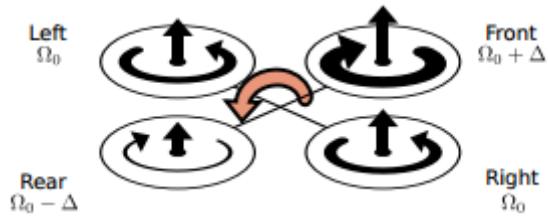


Εικόνα 4. Κίνηση Roll.

2.2.3 PITCH

Αυτή η κίνηση σχετίζεται άμεσα με μια γωνιακή επιτάχυνση στον άξονα-Y. Παρόμοια με την προηγούμενη κίνηση, η κίνηση *pitch* επιτυγχάνεται διατηρώντας την ταχύτητα του αριστερού και δεξιού κινητήρα ενώ παράλληλα γίνεται αύξηση της ταχύτητας του μπροστινού και μείωση της ταχύτητας του πίσω κατά το ίδιο ποσοστό. Αυτά θα επιβάλουν στο σώμα μια θετική περιστροφή στον Y-άξονα. Το αρνητικό *pitch* θα επιτευχθεί με τη μείωση της ταχύτητας του μπροστινού κινητήρα ενώ παράλληλα

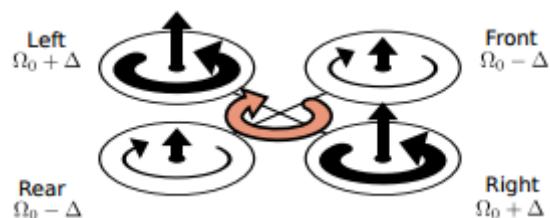
αυξάνεται η ταχύτητα του πίσω κινητήρα κατά κατά το ίδιο ποσοστό. Με μια πρώτη προσέγγιση, η κίνηση αυτή θα προκαλέσει μόνο μια γωνιακή επιτάχυνση αλλά όπως και η κίνηση roll η ώθηση θα περιέχει μια οριζόντια συνιστώσα που θα κάνει το *Quadcopter* να μετακινηθεί προς τα πίσω ή προς τα εμπρός.



Eikόνα 5. Κίνηση Pitch.

2.2.4 YAW

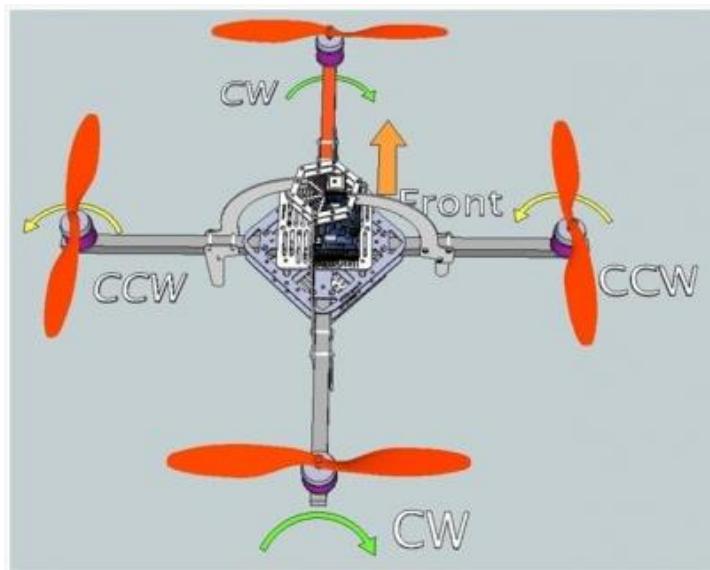
Αυτή η κίνηση είναι άμεσα συνδεδεμένη με μια γωνιακή επιτάχυνση κατά τη διεύθυνση του Z-άξονα. Εκμεταλλεύεται το γεγονός ότι οι εμπρός και πίσω έλικες περιστρέφονται δεξιόστροφα, ενώ οι αριστερά και δεξιά έλικες περιστρέφονται αριστερόστροφα. Μειώνοντας την ταχύτητα του μπροστινού και του πίσω κινητήρα ενώ παράλληλα αυξάνουμε την ταχύτητα του αριστερού και του δεξιού κατά το ίδιο ποσό, δημιουργείται μια ανισορροπία της ροπής περί τον άξονα Z.



Eikόνα 6. Κίνηση Yaw.

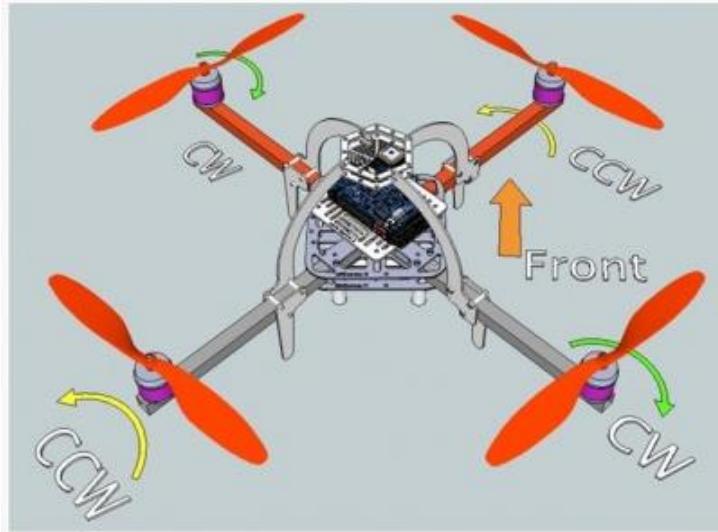
2.3 ΔΙΑΜΟΡΦΩΣΗ ΤΟΥ FRAME

Η διαφορά μεταξύ των διαμορφώσεων είναι ο τρόπος με τον οποίο οι κινητήρες πρέπει να ελέγχονται. Για τη διαμόρφωση «+» ο έλεγχος του κινητήρα είναι αρκετά απλός. Υπάρχει ένα μοτέρ σε κάθε κατεύθυνση. Η αρνητική πτυχή της διαμόρφωσης «+» όμως είναι ότι υπάρχει μόνο ένα μοτέρ για να παρέχεται επιπλέον ώθηση (επιτάχυνση) προς την αντίστοιχη κατεύθυνση.



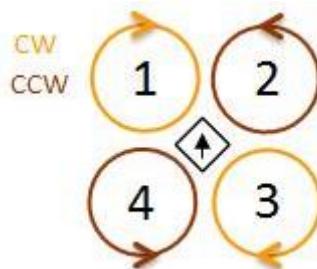
Εικόνα 7. Διαμόρφωση frame «+».

Οι παραπάνω περιγραφές όσον αφορά τις κινήσεις του *Quadcopter* σε σχέση πάντα με τους βαθμούς ελευθερίας, αναφέρονται σαν θεωρητικό υπόβαθρο για να αντιληφθούμε τις δυνατότητες μανουνβρών που μπορούν να πραγματοποιηθούν. Στη δική μας περίπτωση ο προσανατολισμός των κινήσεων αντιπροσωπεύεται από την παρακάτω εικόνα, αφού επιθυμούμε οι πτήσεις να γίνονται σε «X» διαμόρφωση. Για το λόγο αυτό η διαφοροποίηση σε σχέση με τη διαμόρφωση «+» που περιγράφεται παραπάνω θεωρητικά είναι ότι για να πραγματοποιηθούν οι κινήσεις πρέπει οι αυξομειώσεις των μοτέρ να γίνονται ταυτόχρονα σε δύο κινητήρες μαζί [Ref 2].



Εικόνα 8. Διαμόρφωση frame «X».

Για να γίνει πιο κατανοητά τόσο ο τρόπος τοποθέτησης των προπελών, όσο και η εκάστοτε κατεύθυνση της στροφικής κίνηση των κινητήρων παρακάτω επισυνάπτουμε μια εικόνα με σκοπό τη περιγραφή της σαν παράδειγμα. Σε αυτή την εικόνα οι κινητήρες αριθμούνται διαδοχικά σε μια δεξιόστροφη κατεύθυνση ξεκινώντας από το μπροστινό αριστερό κινητήρα. Για να πραγματοποιηθεί ο έλεγχος της εκτροπής του *Quadcopter* (περιστροφή γύρω από τον άξονα z), οι κινητήρες 1 και 3 πραγματοποιούν περιστροφή αριστερόστροφα, ενώ οι κινητήρες 2 και 4 περιστροφή δεξιόστροφα. Ατομικά, οι κινητήρες 1 και 3 πρέπει να περιστρέφονται προς τα δεξιά με μια δεξιόστροφη προπέλα, και κινητήρες 2 και 4 θα πρέπει να περιστρέφονται αριστερόστροφα με μία αριστερόστροφη προπέλα. Για να σημαδέψουμε τον λογικό προσανατολισμό του *Quadcopter* τοποθετούμε στο κέντρο της εικόνας το ανάλογο βελάκι που δείχνει την κατεύθυνση εμπρόσθιας κίνησης [Ref 3].



Εικόνα 9. Κυκλική κατεύθυνση των προπελών.

ΚΕΦΑΛΑΙΟ 3: ΥΛΙΚΑ ΜΕΡΗ ΤΟΥ QUADCOPTER

3.1 Ο ΜΙΚΡΟΕΛΕΓΚΤΗΣ ARDUINO UNO

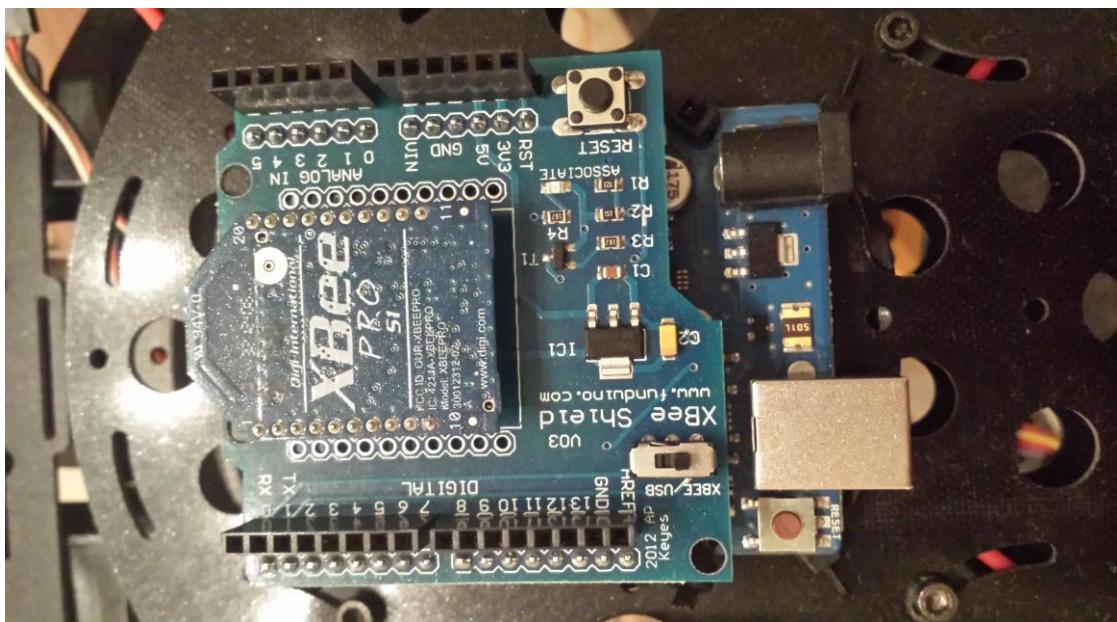
To Arduino είναι μια ένας *single-board* μικροελεγκτής, δηλαδή μια απλή μητρική πλακέτα ανοικτού κώδικα, με ενσωματωμένο το programmer και εισόδους/εξόδους, η οποία μπορεί να προγραμματιστεί με τη γλώσσα Wiring (ουσιαστικά πρόκειται για τη γλώσσα προγραμματισμού C++ και ένα σύνολο από βιβλιοθήκες, υλοποιημένες επίσης στην C++). Το Arduino μπορεί να χρησιμοποιηθεί για την ανάπτυξη ανεξάρτητων δια δραστικών αντικειμένων αλλά και να συνδεθεί με υπολογιστή μέσω προγραμμάτων όπως *Processing*, *Max/MSP*, *Pure Data*, *Super Collider*.

Μία πλακέτα *Arduino* αποτελείται από ένα μικροελεγκτή της *Atmel AVR* (*ATmega328* και *ATmega168* στις νεότερες εκδόσεις και *ATmega8* στις παλαιότερες) και συμπληρωματικά εξαρτήματα για την διευκόλυνση του χρήστη στον προγραμματισμό και την ενσωμάτωση του σε άλλα κυκλώματα. Όλες οι πλακέτες περιλαμβάνουν ένα γραμμικό ρυθμιστή τάσης 5V και έναν κρυσταλλικό ταλαντωτή 16MHz. Ο μικροελεγκτής είναι από κατασκευής προγραμματισμένος με ένα *bootloader*, έτσι ώστε να μην χρειάζεται εξωτερικός προγραμματιστής [Ref 4] .

Για την υλοποίηση του *fly controller* αλλά και για το χειρισμό του *Quadcopter* αποφασίσαμε να χρησιμοποιήσουμε δύο μικροελεγκτές *Arduino Uno* τους οποίους πραγματοποιήσαμε κατάλληλα και στις δύο περιπτώσεις. Το ένα *Arduino* είναι αφοσιωμένο στον έλεγχο ισορροπίας και στη πραγματοποίηση των κινήσεων του *Quadcopter* ενώ το δεύτερο είναι συνδεδεμένο με ένα χειριστήριο PS2 το οποίο μέσω της κατάλληλης διασύνδεσης και επικοινωνίας τροφοδοτεί το πρώτο με τις εντολές κίνησης που επιθυμούμε.



Εικόνα 10. Εγκατάσταση Arduino στο PS2 ελεγκτή.



Εικόνα 11. Εγκατάσταση Arduino στο Quadcopter.

3.2 ΤΟ ΠΛΑΙΣΟ ΤΟΥ QUADCOPTER

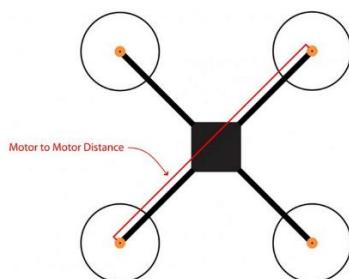
Το πλαίσιο είναι η κατασκευή που φέρει πάνω του όλα τα εξαρτήματα. Το πλαίσιο πρέπει να είναι στιβαρό και να μπορεί να απορροφά τους κραδασμούς που δημιουργούν οι κινητήρες ενώ αποτελείται από 2 με 3 τμήματα τα οποία δεν είναι απαραίτητο να είναι κατασκευασμένα από το ίδιο υλικό:

- Την κεντρική πλάκα πάνω στην οποία τοποθετούνται τα ηλεκτρονικά
- Τέσσερεις βραχίονες που προσαρμόζονται στην κεντρική πλάκα
- Τέσσερεις θέσεις για τα μοτέρ που βρίσκονται στο άκρο του κάθε βραχίονα

Τα πιο συνηθισμένα υλικά για την κατασκευή του πλαισίου είναι :

- Ανθρακονήματα
- Αλουμίνιο
- Ξύλο όπως κόντρα πλακέ ή MDF

Το ανθακόνημα είναι το πιο σταθερό υλικό που απορροφά καλύτερα τους κραδασμούς, είναι όμως και το πιο ακριβό. Το πιο δημοφιλές υλικό για την κατασκευή των βραχιόνων αποτελεί το αλουμίνιο σε σχήμα διάτρητων κατά μήκος ράβδων σε τετράγωνο σχήμα. Χρησιμοποιείται κατά κόρον λόγω του χαμηλού του βάρους, της μεγάλης στιβαρότητας και της άμεσης διαθεσιμότητας που τα διακρίνουν. Παρόλα αυτά το αλουμίνιο μειονεκτεί στην απορρόφηση των κραδασμών διότι μπορούν να δημιουργηθούν λάθη στη λήψη μετρήσεων από τους αισθητήρες. Το ξύλο μπορεί να χρησιμοποιηθεί αντί του αλουμινίου στους βραχίονες καθώς απορροφά πολύ καλύτερα τους κραδασμούς χωρίς όμως να είναι πολύ στιβαρό υλικό, ενώ μπορεί εύκολα να σπάσει με ενδεχόμενα χτυπήματα. Όσον αφορά το μήκος των βραχιόνων ο όρος «απόσταση μεταξύ κινητήρων» χρησιμοποιείται συχνά και αποτελεί την απόσταση μεταξύ των κέντρων αντιδιαμετρικών κινητήρων.



Εικόνα 12. Διάμετρος frame .

Η απόσταση μεταξύ κινητήρων συνήθως εξαρτάται από τη διάμετρο των ελίκων διότι χρειάζεται να υπάρχει αρκετός χώρος μεταξύ τους ώστε να μη συμπλέκονται. Ο στόχος του *Quadcopter* ήταν εξαρχής η λήψη εναέριων βίντεο (*Aerial Video*). Για το σκοπό αυτό απαιτείται η επιλογή ενός frame αρκετά μεγάλου έτσι ώστε να εξασφαλίσουμε τόσο την ισορροπία και σταθερότητα κατά τις λήψεις, όσο και την ομαλότητα στις κινήσεις που πραγματοποιεί. Για να επιτευχθούν όλα αυτά επιλέξαμε το *frame X - 580* της *Hobbyking*, το οποίο είναι αρκετά ελαφρύ αφού αποτελείται από ανθρακόνημα σε συνδυασμό με αλουμίνιο ενώ με τις κατάλληλες προπέλες καταφέρνει να ανταπεξέλθει στις παραπάνω απαιτήσεις. Από το όνομα του frame μπορεί κανείς να καταλάβει την διάμετρο μεταξύ των κέντρων δύο κινητήρων η οποία είναι 580 cm. Επίσης, το frame συνοδεύεται από βάση κάμερας (*camera mount*) και κατάλληλη βάση στήριξης για να τη κρατά σε απόσταση από το έδαφος. Τέλος το *frame X - 580* είναι αναδιπλούμενο και ρυθμιζόμενο για να αποθηκεύεται και να μεταφέρεται εύκολα.



Εικόνα 13. X -580 frame .

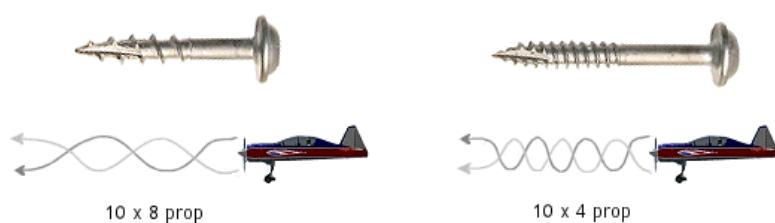
3.3 ΧΑΡΑΚΤΗΡΗΣΤΙΚΑ ΠΡΟΠΕΛΩΝ

Κάθε ένας από τους κινητήρες φέρει μια έλικα ενώ από το σύνολο των τεσσάρων ελίκων ανά δύο ζευγάρια είναι όμοιες . Η επάνω δεξιά και η κάτω αριστερά είναι δεξιόστροφη ενώ οι υπόλοιπες δύο αριστερόστροφες. Όπως αναφέρθηκε πριν δύο κινητήρες περιστρέφονται προς την αντίθετη κατεύθυνση για να αποφευχθεί η ελεύθερη περιστροφή του *Quadcopter* στον αέρα. Αυτή η διαμόρφωση κάνει εφικτή την σταθεροποίηση και τον έλεγχο της γωνιακής ταχύτητας περί από του άξονα Z, δηλαδή της περιστροφής του *Quadcopter* γύρω από τον εαυτό του.



Εικόνα 14. Τα είδη των προπελών.

Οι προπέλες κατηγοριοποιούνται με βάση το μήκος και τη στρέψη (*pitch*). Στο *Quadcopter* χρησιμοποιούμε προπέλες 10×4.5 , δηλαδή μήκους 10 ιντσών και τιμή στρέψης (*pitch*) 4,5. Γενικά ισχύει ότι προπέλες μεγαλύτερου μήκους και pitch απαιτούν περισσότερο ρεύμα για τη κίνηση τους. Το *pitch* μπορεί να οριστεί ως η απόσταση που διανύεται σε μία πλήρης περιστροφή της προπέλας. Εν συντομίᾳ υψηλότερο *pitch* σημαίνει πιο αργή περιστροφή αυξάνοντας μεν την ταχύτητα του οχήματος, απαιτώντας δε μεγαλύτερη ισχύ.



Εικόνα 15. Στρέψη και ροπή των προπελλών.

Γενικά μία έλικα με μικρό *pitch* δημιουργεί μικρότερη ροπή όπως επίσης βελτιώνει τη σταθερότητα στη πτήση. Κατά τη λήψη απόφασης τόσο για το μήκος όσο και το *pitch* χρειάζεται να βρεθεί ένα σημείο ισορροπίας. Οι κινητήρες δεν χρειάζεται να λειτουργούν τόσο έντονα σε σύγκριση εφαρμογής μεγαλύτερου *pitch* αφού απαιτείται λιγότερο ρεύμα. Αν κάποιος θέλει να κάνει ακροβατικές μανούβρες κατά τη πτήση του *Quadcopter* απαιτείται μεγάλη ροπή, ώστε να παρέχεται μεγάλη επιτάχυνση. Προπέλες χαμηλού *pitch* επίσης βελτιώνουν τη σταθερότητα κατά τη πτήση.

Μία προπέλα υψηλού *pitch* μετακινεί μεγαλύτερο όγκο αέρα κάτι που θα μπορούσε να δημιουργήσει κραδασμούς και να κάνει την κατασκευή να πάλλεται ενώ αιωρείται. Όσον αφορά το μήκος, σχετίζεται στενά με τον όγκο αέρα που μετακινεί η προπέλα αφού μικρή αύξηση του μήκους αυξάνει την αποτελεσματικότητα της. Η μικρότερη προπέλα δε, επιταχύνει και επιβραδύνει ευκολότερα αλλάζοντας παράλληλα τη ταχύτητα με αργούς ρυθμούς καταναλώνοντας μικρότερο ρεύμα. Για μεγαλύτερα *Quadcopter* που μεταφέρουν φορτία, όπως αυτό που επιλέξαμε, χρησιμοποιώντας μεγαλύτερες προπέλες σε συνδυασμό με κινητήρες χαμηλών στροφών μπορούν να λειτουργήσουν καλύτερα [Ref 5].

3.3.1 ΙΣΟΡΡΟΠΙΣΤΗΣ ΠΡΟΠΕΛΩΝ

Λόγω του γεγονότος ότι τα πολυκόπτερα διαθέτουν 3, 4, 6 ακόμα και 8 προπέλες, οι δονήσεις που παράγονται από αυτές αποτελούν μείζον πρόβλημα. Τα γυροσκόπια και τα επιταχυνσιόμετρα από τα οποία, μέσω των μετρήσεων αποφαινόμαστε για την ισορροπία του *Quadcopter*, είναι πολύ ευαίσθητα στις δονήσεις. Το *Project* που αναπτύσσουμε ως τελικό στόχο έχει τη λήψη εναέριων βίντεο, συνεπώς οι δονήσεις θα έχουν ως αποτέλεσμα την κακή λήψη του. Οι δονήσεις μπορούν να αποφέρουν μέχρι και δομικές αποτυχίες, που σαν αποτέλεσμα θα έχουν την αποτροπή σωστής πτήσης του *Quadcopter*. Για να αποφύγουμε τα προαναφερθέντα προβλημάτα πρέπει να πραγματοποιήσουμε το ζύγισμα των προπελών πρέπει να συνδέσουμε την κάθε προπέλα σε έναν *propeller balancer* και να εντοπίσουμε την βαριά πλευρά της κάθε προπέλας. Η διαδικασία του εντοπισμού γίνεται εύκολα, αρκεί να παρατηρήσουμε ποια μεριά θα μείνει σταθερά στο κάτω μέρος. Εφόσον αυτή εντοπιστεί τότε με λεπτό ναλόχαρτο (800 – 1000 mm) τρίβουμε ελαφρά στο κάτω μέρος της βαριάς μεριάς της

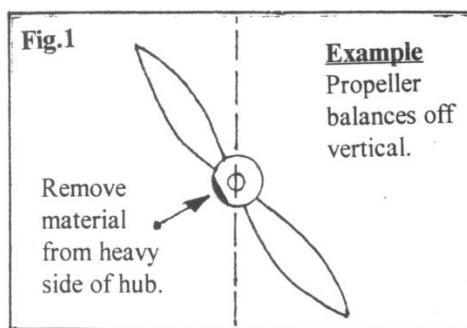
Για να πραγματοποιήσουμε το ζύγισμα των προπελών πρέπει να συνδέσουμε την κάθε προπέλα σε έναν *propeller balancer* και να εντοπίσουμε την βαριά πλευρά της κάθε προπέλας. Η διαδικασία του εντοπισμού γίνεται εύκολα, αρκεί να παρατηρήσουμε ποια μεριά θα μείνει σταθερά στο κάτω μέρος. Εφόσον αυτή εντοπιστεί τότε με λεπτό ναλόχαρτο (800 – 1000 mm) τρίβουμε ελαφρά στο κάτω μέρος της βαριάς μεριάς της

προπέλας και επαναλαμβάνουμε μέχρι η προπέλα να μείνει παράλληλα ως προς το έδαφος, χωρίς να υπάρχει κάποια πλευρά να χαλάει την ισορροπία, όπως ακριβώς φαίνεται στη παρακάτω εικόνα.



Εικόνα 16. Κάθετος ισσοροποσμός προπελών.

Όταν ολοκληρωθεί η διαδικασία της οριζόντιας ευθυγράμμισης επαναλαμβάνουμε τη διαδικασία για την κάθετη ευθυγράμμιση. Για να σταθεί η προπέλα κάθετα θα πρέπει να αφαιρούμε υλικό από το κέντρο της πλευράς που θεωρείται πιο βαριά. Όταν ολοκληρώσουμε και τις δύο διαδικασίες για κάθε προπέλα, τότε απαλλασσόμαστε από τις ανεπιθύμητες δονήσεις κατά την πτήση. Επιπλέον οι ευθυγραμμισμένες προπέλες βιοηθούν καταλυτικά στη μακροζωία των μοτέρ. Παρακάτω παρουσιάζουμε μία φωτογραφία αρκετά παραστατική για την καλύτερη κατανόηση της διαδικασίας [Ref 6] .

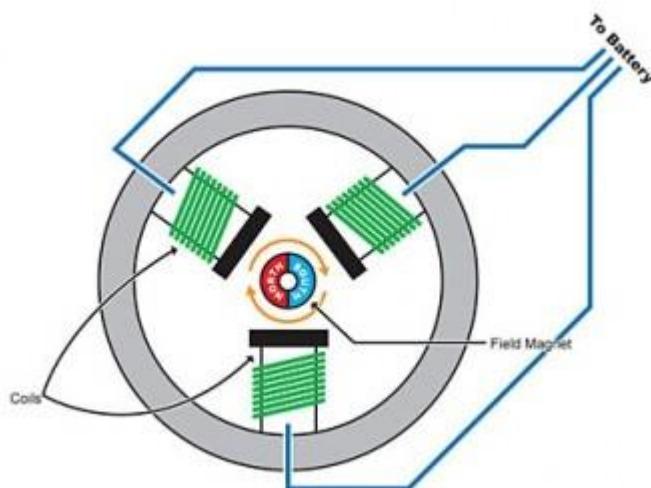


Εικόνα 17. Οριζόντις ισσοροποσμός προπελών.

3.4 ΧΑΡΑΚΤΗΡΗΣΤΙΚΑ ΜΟΤΕΡ

Όπως έχουμε ήδη προαναφέρει τα *Quadcopters* έχουν τέσσερα μοτέρ που το κάθε ένα οδηγεί μία προπέλα. Τις περισσότερες φορές τα μοτέρ αυτά αναφέρονται σαν *Brushless Motors*. Έτσι σε αυτό το σημείο θα εξηγήσουμε τη διαφοροποίηση τους από τα *brushed motor* και γιατί επιλέξαμε να τα χρησιμοποιήσουμε. Τα *Brushless motors* είναι κάπως όμοια με τα γνωστά *DC motors* όσον αφορά τα υλικά τα οποία εμπεριέχονται. Τα υλικά αυτά είναι τα πηνία και οι μαγνήτες που είναι υπεύθυνα για την οδήγηση των αξόνων των μοτέρ. Παρόλα αυτά ο τρόπος που είναι διατεταγμένα εσωτερικά διαφέρει.

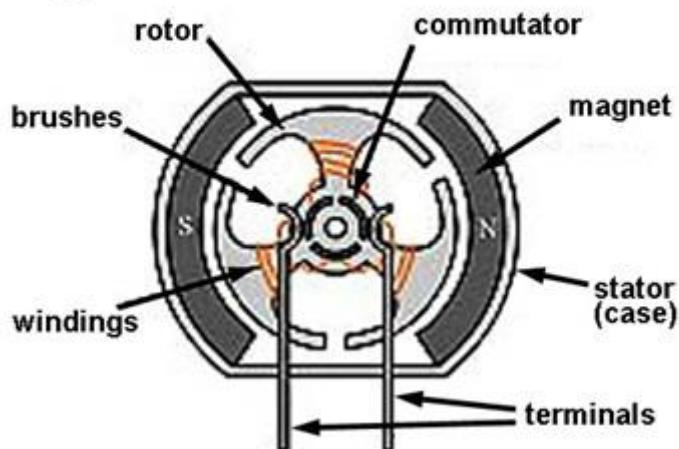
Στα *Brushless motors* τα πηνία είναι προσκολλημένα στη περίμετρο του εσωτερικού του μοτέρ και δεν αποτελούν μέρος του κινούμενου άξονα. Αυτός είναι ο λόγος που αυτού του είδους τα μοτέρ ονομάζονται *brush – less*. Πιο συγκεκριμένα εμπεριέχουν τρία σταθερά πηνία περιμετρικά και έναν άξονα που ενσωματώνει τρείς μαγνήτες οι οποίοι αποτελούν το κινούμενο μέρος. Με τον τρόπο αυτό τα καλώδια ρεύματος τοποθετούνται κατευθείαν στα πηνία δημιουργώντας το απαραίτητο ηλεκτρομαγνητικό πεδίο που με τη κατάλληλη πόλωση περιστρέφει τους εσωτερικούς μαγνήτες και κατά συνέπεια όλο τον άξονα.



Eikόνα 18. Εσωτερική διάταξη των Brushless motors.

Ο λόγος που στα *Quadcopter* χρησιμοποιούμε brushless motors έναντι των *brushed* motor είναι η πολύ μεγαλύτερη ταχύτητα που μπορούμε να πετύχουμε σε συνδυασμό με την μικρότερη κατανάλωση ενέργειας. Αυτό είναι λογικό αφού το κινούμενο μέρος, μη περιέχοντας τα πηνία είναι πολύ ελαφρύτερο παρέχοντάς μας μεγαλύτερη γωνιακή ταχύτητα κατά την περιστροφή κάνοντάς τα πιο αποδοτικά. Στη παρακάτω φωτογραφία παρουσιάζεται ένα *brushed motor* στο οποίο παρατηρούμε ότι τα πηνία είναι αυτά που περιστρέφονται με σταθερούς τους μαγνήτες στη περίμετρο, εξακριβώνοντας τα προ ανεπτυγμένα συμπεράσματα.

Typical brushed motor in cross-section



Eικόνα 19. Εσωτερική διάταξη των Brushed motors.

Tα *Brushless* motors εντοπίζονται σε μεγάλη ποικιλία με διάφορα χαρακτηριστικά των οποίων το μέγεθος της κατανάλωσης ρεύματος διαφέρει. Κατά την επιλογή ενός *Brushless* motor πρέπει να λάβουμε υπόψιν το βάρος, το μέγεθος του μοτέρ καθώς επίσης και το μέγεθος της προπέλας που θα χρησιμοποιήσουμε. Με αυτό τον τρόπο όλα τα χαρακτηριστικά και τα παρελκόμενα του μοτέρ θα μας παρέχουν τη κατάλληλη κατανάλωση ρεύματος που επιθυμούμε λειτουργώντας καταλυτικά ως προς την αποτελεσματικότητά τους. Επίσης όταν αναφερόμαστε σε *brushless motors* σημαντικός παράγοντας που πρέπει προσέξει κάποιος είναι ειδικά ο παράγοντας (*Kv - rating*). Ο όρος αυτός είναι ένα μετρούμενο μέγεθος που σαν ένδειξη αντιπροσωπεύει το ποσό των στροφών (*RPMs*) που θα περιστρέψει το μοτέρ εάν τροφοδοτηθεί με πεπερασμένες τιμές τάσης. Οι στροφές μπορούν να υπολογιστούν ως **RPM = Kv * U.**

Για να επιλέξουμε ένα μοτέρ πρέπει πρώτα να γνωρίζουμε ή να εκτιμήσουμε το βάρος το οποίο θα είναι ικανό το κάθε μοτέρ να σηκώσει και στη συνέχεια να ορίσουμε το συνολικό βάρος του *Quadcopter*, για να διαπιστώσουμε ότι ο συνδυασμός των μοτέρ με το *Quadcopter* είναι αρεστός. Άμεσος παράγοντας για την εύρεση αυτή είναι η ώθηση (*thrust*) που το κάθε μοτέρ παράγει στη μέγιστη δυνατή λειτουργία του. Πρέπει να σημειωθεί ότι η τιμή αυτή δεν εξαρτάται πλήρως από τα μοτέρ αλλά από τον συνδυασμό τους με τις προπέλες που θα χρησιμοποιήσουμε. Ένας ιδανικός, εμπειρικός κανόνας λέει ότι η συνολική, μέγιστη ώθηση πρέπει να εξυπηρετεί το διπλάσιο συνολικό βάρος που αποτελεί το *Quadcopter* έτσι ώστε να παραμένει σε σταθερό ύψος κατά τη πτήση του (*hover*) στο 50% της ισχύς. Παράγοντας λιγότερη ώθηση θα έχει σαν συνέπεια το *Quadcopter* να δυσκολεύεται τόσο κατά την απογείωση όσο και για την πραγματοποίηση των μανουβρών, ενώ παράγοντας μεγαλύτερης ώθηση από την επιθυμητή έχει σαν αποτέλεσμα το *Quadcopter* θα γίνει απότομο και δύσκολο στο χειρισμό. Στην περίπτωση μας εφόσον το ιδανικό δεν είναι και εύκολα επιτεύσημο θα προτιμούσαμε να έχουμε λιγότερη ώθηση για να αποφεύγουμε τις απότομες κινήσεις στην βιντεοσκόπηση του εναέριου βίντεο. Ο μαθηματικός τύπος που χρησιμοποιείται για τον υπολογισμό της ώθησης σε κάθε μοτέρ είναι :

$$\text{Required Thrust per motor} = (\text{QuadWeight} \times 2) / 4$$

Παρακάτω θα αναφέρουμε το τύπο των *brushless motors* που χρησιμοποιήσαμε, όπως επίσης θα υπολογίσουμε μέσω του παραπάνω τύπου το μέγιστο βάρος που μπορεί να σηκώσει το *Quadcopter* συμπεριλαμβανομένου του *frame*.

Για το *Quadcopter* επιλέξαμε *brushless motors* με χαμηλή τιμή *Kv* εφόσον χρησιμοποιήσουμε μεγαλύτερες προπέλες τις οποίες απαιτεί το *frame*. Έτσι εγκαταστήσαμε τέσσερα *NTM 28-30S 800 kv brushless outrunner motor* τα οποία αποκτήσαμε από την εταιρία *hobbyking*.



Εικόνα 20. NTM 28-30s Brushless motor.

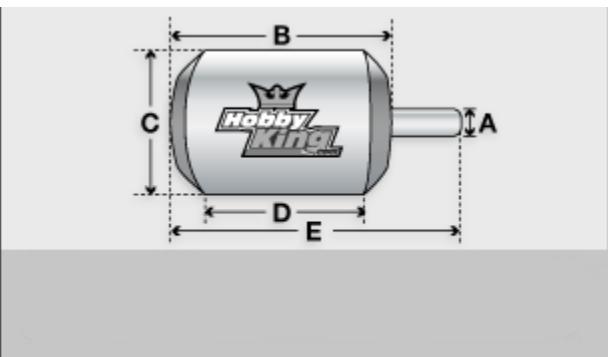
Παρακάτω ακολουθεί ένας πίνακας με τα χαρακτηριστικά του μοτέρ.

ΠΙΝΑΚΑΣ 1. Χαρακτηριστικά του NTM 28-30s .

| MODEL | NTM Prop Drive Series 28-30A 800kv (short shaft version) |
|-------------|--|
| KV | 800rpm/v |
| MAX CURRENT | 20A |
| MAX POWER | 300W |
| SHAFT | 3mm |
| WEIGHT | 65g |
| ESC | 20~30A |
| CELL COUNT | 3s~6s Lipoly |
| BOLT HOLES | 16mm & 19mm |
| BOLT THREAD | M3 |
| CONNECTION | 3.5mm Bullet-connector |

ΠΙΝΑΚΑΣ 2. Χαρακτηριστικά της Hobbyking για το NTM 28-30s.

| | |
|---------------------|-----|
| Kv(rpm/v) | 800 |
| Weight (g) | 65 |
| Max Current(A) | 20 |
| Resistance(mh) | 0 |
| Max Voltage(V) | 23 |
| Power(W) | 300 |
| Shaft A (mm) | - |
| Length B (mm) | 30 |
| Diameter C (mm) | 28 |
| Can Length (mm) | 19 |
| Total Length E (mm) | 33 |



Στα μοτέρ τα οποία επιλέξαμε, όπως αναφέραμε και σε προηγούμενο κεφαλαίο χρησιμοποιούμε προπέλες 10 x 4.5. Συνεπώς μέσω του πίνακα δοκιμών (*Prop Test*) που παρέχεται από τη hobbyking παρατηρούμε ότι το κάθε μοτέρ παρέχει ώθηση (*thrust*) ικανή να σηκώσει περίπου 1Kg. Άρα μέσω του παραπάνω τύπου που παρουσιάσαμε, σε πλήρη ισχύ μπορούμε να σηκώσουμε :

$$ThrustPerMotor = \frac{QuadWeight * 2}{4} \Rightarrow 2 * ThrustPerMotor = QuadWeight \Rightarrow QuadWeight = 2 * 1Kg = 2Kg$$

Μέσω των αποτελεσμάτων αντιλαμβανόμαστε ότι το *Quadcopter* θα επιθυμούσαμε να ζυγίζει 2 με 2,3 Kg το πολύ. Το *Quadcopter* που κατασκευάσαμε ζυγίζει 1,4 Kg.

ΠΙΝΑΚΑΣ 3. Χαρακτηριστικά της Hobbyking για αποδόσεις ώθησης προπελών.

8x4E - 22.2V / 310W / 13.9A / 1.11kg thrust

10x4E - 18.5V / 315W / 17.3A / 1.27kg thrust

11x7E - 14.8V / 260W / 17.8A / 1.05kg thrust

12x6E - 14.8V / 276W / 18.7A / 1.20kg thrust

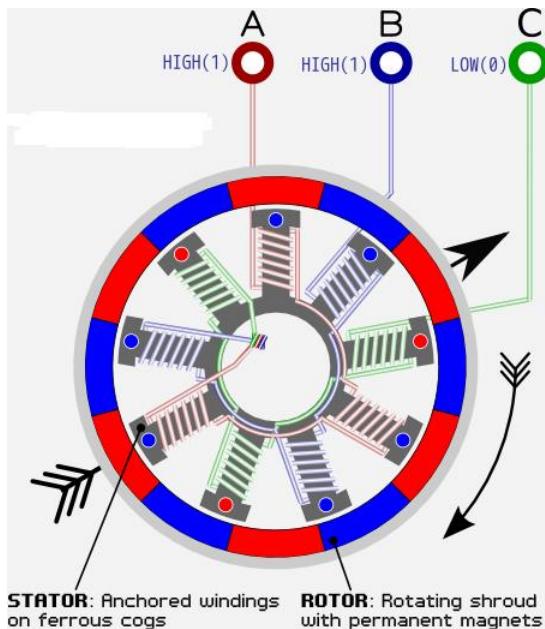
Οι παράγοντες KV και thrust δεν είναι οι μοναδικοί παράγοντες που πρέπει να διερευνηθούν. Ένας άλλος παράγοντας είναι η αποτελεσματικότητα των μοτέρ. Αυτό εξαρτάται από την ποιότητά τους και συχνά ένα μέτρο που χρησιμοποιούμε για να μπορούμε να την αντιληφθούμε, εφόσον αυτή αναφέρεται από τον κατασκευαστή, είναι το ποσοστό της ενέργειας που μπορεί να εκμεταλλευτεί το μοτέρ. Αυτό είναι επιτεύσημο μετρώντας το ποσοστό της ενέργειας που μετατρέπεται σε κινητική και το

ποσοστό της ενέργειας που μετατρέπεται σε θερμική. Μέσω αυτού του ποσοστού παρατηρούμε τη σπατάλη της ισχύος που πραγματοποιείται σε κάθε μοτέρ έναντι αυτής που θεωρείται ωφέλιμη. Τα μοτέρ που χρησιμοποιούμε έχουν ποσοστό αποδοτικότητας $Efficiency \approx 83\%$ [Ref 7].



Εικόνα 21. Εγκατάσταση NTM 28-30s Brushless motor στο frame.

Τέλος πρέπει να αναφέρουμε ότι στα *brushless* motors υπάρχει η δυνατότητα επιλογής της φοράς περιστροφής η οποία θα πραγματοποιείται. Όπως είδαμε στο προηγούμενο κεφάλαιο υπάρχουν δύο τύποι προπελών, που τοποθετούνται στις κατάλληλες θέσεις του *Quadcopter* αναλόγως τη φορά περιστροφής των μοτέρ. Η επιλογή της φοράς των μοτέρ γίνεται απλά με την επιλογή της σειράς που έχουμε συνδέσει τα καλώδια από το *ESC* στο *brushless motor*. Η σειρά που θα τα συνδέσεις δεν παίζει κανέναν ρόλο, εάν η διάταξη σου περιστρέφει το μοτέρ κατά την αριστερόστροφη φορά και επιθυμείς να αλλάξεις φορά (προς τα δεξιόστροφα) απλά αλλάζεις σε δύο οποιαδήποτε καλώδια τη θέση μεταξύ τους. Σαν παράδειγμα, στη παρακάτω εικόνα έχουμε συνδέσει το *ESC* με τη σειρά *ABC*, η οποία γυρνά το μοτέρ αριστερόστροφα. Για να αλλάξει η φορά αλλάζουμε τις δύο θέσεις των καλωδίων (*CBA*, *ACB*, *BAC* .. κ.λπ.). Έτσι η κατεύθυνση επιλέγεται πολύ εύκολα χωρίς να ζημιώνουμε τα μοτέρ. Αναφορικά να πούμε ότι εκτός από αυτόν τον εύκολο και πολυχρησιμοποιούμενο τρόπο, υπάρχει η δυνατότητα αλλαγής της κατεύθυνσης μέσω των *ESC* προγραμματίζοντάς τα κατάλληλα μέσω ενός firmware. Ο τρόπος αυτός όμως είναι πιο πολύπλοκος ενώ επιφέρει τα ίδια αποτελέσματα.



Eικόνα 22. Κυκλική κατεύθυνση του Brushless motor.

3.5 ESC - ELECTRONIC SPEED CONTROLLERS

Λόγω του ότι τα *brushless motors* εμπεριέχουν τρία πηνία και συνεπώς τρείς φάσεις που πρέπει να τροφοδοτηθούν για να λειτουργήσουν, δεν είναι εφικτό παρέχοντας απλά συνεχές ρεύμα σε αυτά αρχίσουν την περιστροφή. Τα μοτέρ απαιτούν ηλεκτρονικά εξαρτήματα ελέγχου φάσης τα οποία με τη σειρά τους παράγουν σήματα υψηλών συχνοτήτων διαφορετικών και ελεγχόμενων φάσεων. Έτσι θα πρέπει να είναι σε θέση να παρέχουν την απαραίτητη ποσότητα ρεύματος καθώς οι απαιτήσεις των μοτέρ μπορεί να αυξηθούν αρκετά, αναλόγως των συνθηκών λειτουργίας.

Για τη σωστή λειτουργία των μοτέρ χρησιμοποιούμε κάποιες ηλεκτρονικές διατάξεις, γνωστές και ως *ESC* (*Electronic Speed Controllers*). Τα *ESC* αποτελούν διαχειριστές των *brushless motor* τα οποία έχουν σταθερή τάση ως είσοδο και τρία διαφορετικές φάσης καλώδια ρεύματος ως έξοδο. Για το έλεγχο, το σήμα που παράγεται στην έξοδο είναι ένα *PWM* (*Pulse Width Modulation*) σήμα του οποίου τα συχνότητά όρια λειτουργίας διαφέρουν αναλόγως το μοντέλο που θα επιλέξουμε, αφού κάθε μοντέλο υποστηρίζει διαφορικά χαρακτηριστικά. Ωστόσο σε κάθε περίπτωση επιθυμούμε να αυξήσουμε το εύρος λειτουργίας των *ESC* γεγονός που επιτρέπει ο *microcontroller* που

χρησιμοποιούμε. Η διαδικασία αυτή λέγεται διαδικασία calibration των *ESC* και την οποία θα περιγράψουμε παρακάτω.

Πρωτίστως για την επιλογή ενός *ESC* ένας σημαντικός παράγοντας που πρέπει να λάβουμε υπόψιν είναι τα watt που υποστηρίζονται από τα μοτέρ. Παρατηρώντας το πίνακα των χαρακτηριστικών βλέπουμε ότι τα μοτέρ παρέχουν ισχύ ίση με 300 watts. Το μέγιστο ποσό ρεύματος που αυτά τα μοτέρ θα χρειαστούν είναι 20 A/ *Motor*. Για το λόγο αυτό πρέπει να επιλέξουμε κατάλληλα τα *ESC* τα οποία θα πληρούν τις προϋποθέσεις και θα επιτρέπουν αυτό το ποσό του ρεύματος. Τα 4 x *ESC* που επιλέξαμε είναι της εταιρίας *hobbyking* (*Afro ESC 30 Amp Multi - rotor Motor Speed Controller (SimonK Firmware)*) τα οποία επιτρέπουν διέλευση ρεύματος ίση με 30A, όπως προαπαιτεί ο πίνακας χαρακτηριστικών των μοτέρ.



Εικόνα 23. *Afro ESC 30 Amp Multi - rotor Motor Speed Controller (SimonK Firmware).*



Εικόνα 24. Εγκατάσταση *Afro ESC 30 Amp Multi - rotor Motor Speed Controller (SimonK Firmware)*.

ΠΙΝΑΚΑΣ 4. Χαρακτηριστικά του Afro ESC 30 Amps.

| CURRENT DRAW | 30A Continuous |
|----------------------|--|
| VOLTAGE RANGE | 2-4s Lipoly |
| BEC | 0.5A Linear |
| INPUT FREQ | 1KHz |
| FIRMWARE | afro_nfet.hex |
| DISCHARGE WIRE/PLUGS | 15AWG/Male 3.5mm |
| MOTOR WIRE/PLUGS | 16AWG/Female 3.5mm |
| WEIGHT | 26.5g (Included wire, plug, heat shrink) |
| SIZE | 50 x 25 x 11mm |

Ένας διαφορετικός αλλά σημαντικός παράγοντας είναι οι προγραμματιστικές δυνατότητες που διαθέτει το κάθε *ESC*. Ορισμένα από αυτά υποστηρίζουν πλήρως ρυθμιζόμενες λειτουργικές διαδικασίες μέσω ενός *firmware*, ενώ άλλα παρέχουν μόνο ρυθμίσεις που πραγματοποιούνται κατά επιλογήν μέσω κατάλληλων σημάτων εισόδου, στέλνοντας με τη σειρά τους τα *ESC* μηνύματα επιβεβαίωσης μέσω ακουστικών συχνοτήτων.

Τα εν λόγῳ *esc* είναι διαθέσιμα με εγκατεστημένο ήδη το *firmware* που τα οδηγούν, καθώς το καθένα εμπεριέχει έναν μικροελεγκτή ο οποίος είναι υπεύθυνος για τη λειτουργία και την παραμετροποίηση τους. Το καλύτερο από αναφορές *firmware* που μπορεί να εγκατασταθεί θεωρείται το *SimonK* το οποίο είναι και αυτό που χρησιμοποιούμε.

Το *SimonK* είναι ένα *firmware* σχεδιασμένο ειδικά για τα *esc* με στόχο τη χρήση τους σε *Quadcopter*, το οποίο αναπτύχθηκε από τον *Simon Kirby*. Αυτό το *firmware* προσφέρει πολύ πιο γρήγορες αποκρίσεις σε σύγκρισή με τα απλά *ESC*, εύκολο επαναπρογραμματισμό και καλή συμβατότητα, με τον συνδυασμό όλων αυτών να βελτιώνει την γενική απόδοση.

3.5.1 ΑΣΦΑΛΕΙΑ

Πριν αναλύσουμε τη διαδικασία *calibration*, *programming* και *testing* των *ESC* θα πρέπει να αναφέρουμε κάποια προληπτικά μέτρα ασφαλείας έτσι ώστε να προστατεύσουμε πρώτα εμάς και ύστερα τα ηλεκτρονικά εξαρτήματα που χρησιμοποιούμε.

- Δεν συνδέουμε τις προπέλες στα μοτέρ κατά τη διαδικασία του *calibration* ή του *testing*. Η σύνδεση των προπελών είναι πραγματικά το τελευταίο βήμα που πρέπει να κάνουμε πριν τη δοκιμαστική πτήση. Εάν την συνδέσουμε νωρίτερα κατά πάσα πιθανότητα αυτή θα σπάσει μέσω ανεξέλεγκτης κίνησης του *Quadcopter*, ενώ υπάρχει και ο κίνδυνος τραυματισμού
- Πρέπει να προστατεύουμε τη θύρα *USB* και το *Arduino*. Για το λόγο αυτό ποτέ δε συνδέουμε τη γραμμή τάσης *BEC* (το μεσαίο από τα τρία καλώδια εισόδου *PWM* του *ESC*) στο *Arduino* και πιο συγκεκριμένα στο *pin 5v*. Εάν πραγματοποιηθεί η συνδεσμολογία κατά πάσα πιθανότητα θα καεί το *ESC*, το *Arduino*, ενώ μπορεί να προκληθεί και ζημιά στη θύρα *USB* του υπολογιστή μας. Για το λόγο αυτό το καλώδιο δεν πρέπει να συνδεθεί πουθενά, είναι έξοδος τάσης, όχι είσοδος.

3.5.2 ΣΥΝΔΕΣΜΟΛΟΓΙΑ

Σαν πρώτο βήμα κατά τη συνδεσμολογία είναι απαραίτητο να συνδέσουμε το *ESC* στο μοτέρ. Στο παράδειγμα αυτό το κίτρινο καλώδιο είναι το μεσαίο καλώδιο. Όπως προαναφέραμε ο συνδυασμός της συνδεσμολογίας δεν παίζει κανένα ρόλο. Απλά συνδέουμε το *ESC* με το μοτέρ και εάν η συνδεσμολογία δεν γίνει όπως την επιθυμούμε, απλά το μοτέρ θα περιστρέψει στη λάθος κατεύθυνση. Η σύνδεση πραγματοποιείται μέσω συνδετήρων (αρσενικού – θηλυκού) των 3,5 mm.



*Εικόνα 25. Σύνδεση *ESC* με *Brushless Motor* 1/2 .*



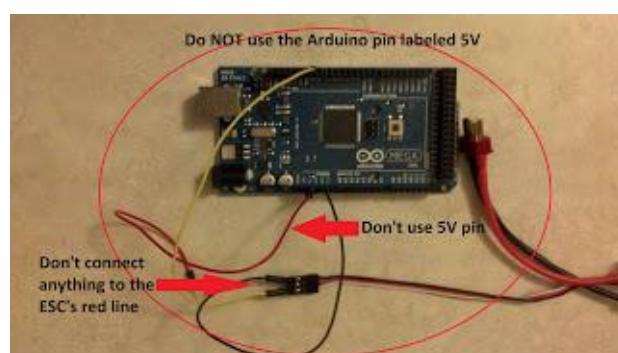
Εικόνα 26. Σύνδεση ESC με Brushless Motor 2/2.

Το δεύτερο βήμα είναι να συνδέσουμε το *ESC* με το *Arduino*. Στο παράδειγμα, χρησιμοποιείται ένας μικροελεγκτής *Arduino Mega*. Παρόλο που εμείς χρησιμοποιούμε τον μικροελεγκτή *Arduino Uno*, αυτό δεν μας επηρεάζει στο να εξηγήσουμε τη διαδικασία. Για τη συνδεσμολογία συνδέσαμε το καλώδιο σήματος ελέγχου *PWM* στην έξοδο του *Arduino* μέσω του *Pin A6* και το αντίστοιχο καλώδιο της γείωσης, στη γείωση του *Arduino*. Στη συνέχεια συνδέσαμε την τάση εισόδου του *ESC* με τη *Lipo* μπαταρία την οποία θα περιγράψουμε αργότερα σε επόμενο κεφάλαιο.



Εικόνα 27. Σύνδεση ESC με Arduino 1/2 .

Προσοχή όμως, όπως αναφέραμε δεν πρέπει να συνδέσουμε τίποτα στη γραμμή τάσης (*BEC*) του *ESC*.

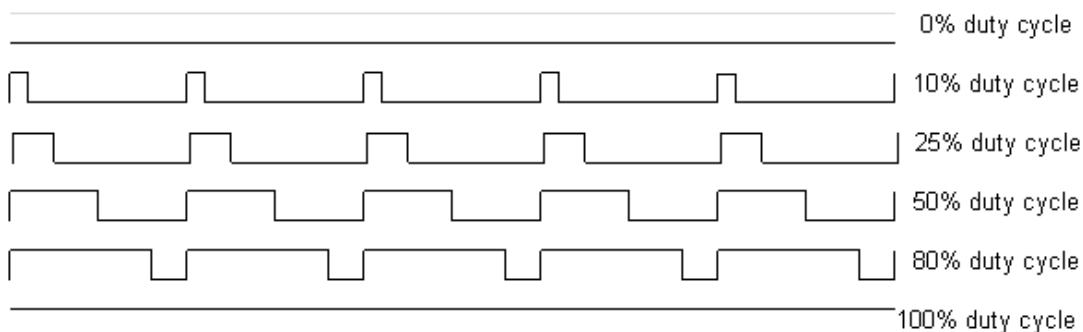


Εικόνα 28. Σύνδεση ESC με Arduino 2/2 .

Τέλος συνδέουμε το *Arduino* με τον υπολογιστή μέσω της θύρας *USB* ολοκληρώνοντας τη διαδικασία. Μόλις πραγματοποιηθεί ορθά η σύνδεση θα ακουστεί ένας χαρακτηριστικός ήχος κατά τη τοποθέτηση τάσης στο *ESC* για πρώτη φορά, ενώ κάθε περίπου 2 δευτερόλεπτα θα ακούγεται ένα σύντομο ακουστικό σήμα (*beep*). Παρακάτω θα διαπιστώσουμε ότι ακούγοντας επανειλημμένα αυτό το σήμα, σημαίνει ότι τα *ESC* έχουν την απαραίτητη τροφοδοσία αλλά δεν έχουν ακόμα οπλίσει, έτσι ώστε να είναι έτοιμα για λειτουργία [Ref 8].

3.5.3 ΒΑΘΜΟΝΟΜΗΣΗ

Η διαδικασία της βαθμονόμησης (*Calibration*) των *ESC* ουσιαστικά σημαίνει η οριοθέτηση της μέγιστης και της ελάχιστης ταχύτητας στα μοτέρ, η οποία ορίζεται μέσω της εισόδου του τετραγωνικού παλμού (με μέγιστο τα 5v και το ελάχιστο τα 0v). Πιο αναλυτικά θέτοντας το *Duty Cycle*, επηρεάζουμε το ποσοστό του θετικού μετώπου του παλμού ως προς τη περίοδο του. Παρακάτω παρουσιάζουμε διάφορα δείγματα *PWM* με διαφορετικό *Duty Cycle* και συνεπώς διαφορετικής διάρκειας.



Eikόνα 29. Duty Cycle PWM σημάτων .

Το *PWM* σήμα διαβάζεται από το *ESC* ακριβώς όπως διαβάζεται και σε έναν *Servo Motor*, έτσι για να το χρησιμοποιήσουμε από το *Arduino*, χρησιμοποιούμε τη βιβλιοθήκη *Servo.h* που είναι εγκατεστημένη στις βιβλιοθήκες του *IDE*. Την ίδια βιβλιοθήκη χρησιμοποιούμε για να πραγματοποιήσουμε τη διαδικασία του *Calibration*. Το εργοστασιακό εύρος σήματος εισόδου *PWM* των *ESC* είναι μεταξύ 1200 και 1800 *microseconds* που επαναλαμβάνεται σε μία περίοδο εκτέλεσης, μέσω του *Arduino*, κάθε 20 *milliseconds* (εάν υποθέσουμε ότι η επανάληψη – *loop* στο *sketch* έχει συχνότητα 50hz). Για το *Quadcopter* όμως, επιθυμούμε το εύρος να είναι όσον το

δυνατό μεγαλύτερο, έτσι ώστε οι κβαντισμένες τιμές ταχύτητας των μοτέρ να είναι περισσότερες. Τα *Afro ESC* τα οποία έχουμε στη κατοχή μας αποφασίσαμε να τα “καλιμπράρουμε” μεταξύ των τιμών 1000 *microseconds* ως ελάχιστο και 2000 *microseconds* ως μέγιστο. Αυτές οι ρυθμίσεις θεωρούνται ιδανικές διότι ούτως η αλλιώς, τα *ESC* δύσκολα αντιλαμβάνονται σήμα εισόδου μικρότερης περιόδου από 900 *microseconds* και περιόδων μεγαλύτερης διάρκειας των 2100 *microseconds*.

Για να θέσουμε αυτές τις τιμές στο κάθε *ESC* θα πρέπει πρώτα να το θέσουμε σε *programming mode*. Αρχικά θέτουμε το μέγιστο σήμα εξόδου στο *ESC* (2000 *microseconds*) όσο το *ESC* είναι εκτός τροφοδοσίας. Αργότερα συνδέουμε το *ESC* στη τροφοδοσία όσο στέλνουμε το προαναφερθέν σήμα και περιμένουμε δύο δευτερόλεπτα (Θα ακουστεί ένας χαρακτηριστικός ήχος ότι μπήκαμε στο *programming mode* ενώ ταυτόχρονα θέσαμε τη μέγιστη τιμή *PWM*). Ύστερα από το πέρας των 2 δευτερολέπτων θέτουμε την ελάχιστη τιμή στο *ESC* θέτοντας παράλληλα την ελάχιστη τιμή *PWM* στο *ESC*. Τέλος, ύστερα από μία σειρά ήχων επιβεβαίωσης αποσυνδέουμε το *ESC* από την τροφοδοσία για 5-6 δευτερόλεπτα και είναι έτοιμο για χρήση. Επαναλαμβάνουμε για όλα τα *ESC* για να περάσουν όλα τη διαδικασία *Calibration*.

Παρακάτω παρουσιάζουμε τον κώδικα που χρησιμοποιήσαμε έτσι ώστε να πραγματοποιηθεί η διαδικασία. Πρέπει να παρατηρήσουμε ότι δεν χρησιμοποιούμε την εντολή *Servo.write(int)* ούτε για το *calibration* ούτε για το *testing*. Αντί αυτού χρησιμοποιούμε την εντολή που μας παρέχει η *Servo.h* *writeMicroseconds(int)* η οποία είναι πολύ πιο συγκεκριμένη στη τιμή που θέτει ως σήμα *PWM* αφού η διακριτική ικανότητα για να θέσει μια τιμή είναι πολύ μεγαλύτερη [Ref 10] .

Listing 1. ESC calibration c++ language technical details

```
/*
-----
MSc HEP 2013-2014
-----

*/
/*
-----
Project      : Quadcopter
File         : Calibrate_esc_min_max.ino
Description   : This code calibrates a single ESC through PWM output
Author       : Monahopoulos Konstantinos
-----

*/
/*
-----
Includes
-----

*/
#include <Servo.h>
/*


-----
Definitions
-----

*/
#define MAX_SIGNAL 2000
#define MIN_SIGNAL 1000
#define MOTOR_PIN 6

/*
-----
Assign Classes
-----

*/
Servo motor; // Instance of Servo Class

/*
-----
Main Code
-----

*/
void setup() {
    Serial.begin(9600);
    Serial.println("Program begin...");
    Serial.println("This program will calibrate the ESC.");

    motor.attach(MOTOR_PIN);

    Serial.println("Now writing maximum output.");
    Serial.println("Turn on power source, then wait 2 seconds and press any key.");
    motor.writeMicroseconds(MAX_SIGNAL);
```

```

// Wait for input
while (!Serial.available());
Serial.read();

// Send min output
Serial.println("Sending minimum output");
motor.writeMicroseconds(MIN_SIGNAL);

}

void loop() {
// Nothing to do here
}

```

Με χρήση των παραπάνω διαδικασιών καταφέραμε και πραγματοποιήσαμε τόσο την ορθή συνδεσμολογία, όσο και το *Calibration* σε όλα τα μοτέρ. Στη συνέχεια θα γίνει περιγραφή για τον τρόπο που μπορούμε να εισέλθουμε στο *Programming Menu* των *ESC* καθώς επίσης και το πώς γίνεται η παραμετροποίηση τους.

3.5.4 ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ

Αν και αυτό το βήμα δεν θεωρείται απολύτως απαραίτητο, πληροφοριακά θα παρουσιάσουμε το κώδικα που μπορεί να χρησιμοποιήσει κάποιος για την παραμετροποίηση των *ESC* μέσω του *Programing mode*, εφόσον δεν κατέχει κάποιο εξειδικευμένο *firmware*. Παρόλη τη σημασία του όμως αποτελεί μια πολύ χρήσιμη πληροφορία σε περίπτωση που επιθυμούμε να αλλάξουμε τον τρόπο λειτουργίας του *ESC* ή σε περίπτωση που από λάθος το *ESC* χάσει τις ρυθμίσεις του και επιθυμούμε να κάνουμε *factory reset*. Όπως και στη διαδικασία του *Calibration* θέτοντας την υψηλότερη τιμή *PWM*, το *ESC* μπαίνει σε *Programming Mode*. Περιμένοντας ένα μικρό χρονικό διάστημα θα ακούσουμε μια σειρά από ηχητικά ακούσματα τα οποία σηματοδοτούν το καθένα και από μία επιλογή. Έτσι κάθε επιλογή που μας δίνεται έχει το δικό του χαρακτηριστικό άκουσμα.

Πιο συγκεκριμένα επισυνάπτουμε το πίνακα χαρακτηριστικών ακουστικών επιλογών (Όπου *Low* = ήχος χαμηλής συχνότητας, όπου *High* = ήχος υψηλής συχνότητας). Σε αυτό το σημείο θα αναφέρουμε μόνο τις επιλογές του μενού, ενώ δεν εμβαθύναμε σε υπό -επιλογές αυτού.

ΠΙΝΑΚΑΣ 5. Ηχητικές επιλογές Programming Mode των ESC .

| PROGRAMMING MENU | SOUND BEEP |
|-----------------------|-----------------|
| BRAKE | LOW |
| BATTERY TYPE | LOW-LOW |
| CUT-OFF MODE | LOW-LOW-LOW |
| CUT-OFF THRESHOLD | LOW-LOW-LOW-LOW |
| START-UP MODE | HIGH |
| TIMING | HIGH- LOW |
| RESET TO DEFAULT | HIGH- LOW- LOW |
| EXIT PROGRAMMING MODE | HIGH- HIGH |

Στο τέλος κάθε κύκλου σάρωσης του μενού, οι επιλογές επαναλαμβάνονται από την αρχή. Για να μπει κάποιος στο *Sub - Menu* κάποιας από τις επιλογές δεν έχει παρά να θέσει το *PWM* στην ελάχιστη τιμή όταν ακούσει την επιθυμητή επιλογή. Για να ορίσει μία από τις επιλογές του *Sub - Menu* θα πρέπει να ξαναστείλει τη μέγιστη τιμή *PWM* στο *ESC*. Κάπως έτσι με εναλλαγές μεγίστου – ελαχίστου πλοηγούμαστε στο *Programming Mode* των *ESC*. Όταν θέσουμε μία επιλογή το *ESC* απαντά με κάποιους ήχους επιβεβαίωσης. Όπως και πριν ύστερα από κάποια ρύθμιση πρέπει να αποσυνδέσουμε το *ESC* από την τροφοδοσία για 5-6 δευτερόλεπτα για να πραγματοποιηθεί η αλλαγή. Παρακάτω παρουσιάζουμε τον κώδικα που χρησιμοποιήσαμε έτσι ώστε να πραγματοποιηθεί η διαδικασία που περιγράψαμε.

Listing 2. ESC programming c++ language technical details

```
/*
-----
MSC HEP 2013-2014
-----
*/
/*
-----
Project      : Quadcopter
File        : Programming_esc_enter_menu.ino
Description  : This code enters Programming Menu of ESC through PWM
-----
```

```

Author      : Monahopoulos Konstantinos
-----
*/
/*
-----
Includes
-----
*/
#include <Servo.h>
/*
-----
Definitions
-----
*/
#define MAX_SIGNAL 2000
#define MIN_SIGNAL 1000
#define MOTOR_PIN 6
/*
-----
Assign Classes
-----
*/
Servo motor;
/*
-----
Main Code
-----
*/
void setup() {
    Serial.begin(9600);
    Serial.println("Program begin...");
    Serial.println("This program will enter Programming Menu of the ESC.");
    motor.attach(MOTOR_PIN);

    Serial.println("Now writing maximum output for entering Programming Mode.");
    motor.writeMicroseconds(MAX_SIGNAL);
    Serial.println("Power the esc, then wait 2 seconds ");

    /*Wait for input*/
    while (!Serial.available());
    Serial.read();

    /*Send min output*/
    Serial.println("Sending minimum output to choose programming mode");
    motor.writeMicroseconds(MIN_SIGNAL);

    /*Wait for input*/
    Serial.println("Press again to send maximum output for sub-menu or option");
    while (!Serial.available());
    Serial.read();

    /*Send max output*/
    motor.writeMicroseconds(MAX_SIGNAL);
}

```

```

void loop() {
// Nothing to do here
}

```

3.5.5 ΔΟΚΙΜΗ ΛΕΙΤΟΥΡΓΙΑΣ

Έχοντας ολοκληρώσει όλες τις διαδικασίες, δεν μένει τίποτα άλλο από την διαδικασία του *testing*. Για το λόγο αυτό έχουμε αναπτύξει έναν αλγόριθμο στον οποίο χρησιμοποιώντας τη σειριακή θύρα μέσω του *Command Prompt* μπορούμε να θέσουμε το ποσό περιστροφής του εκάστοτε κινητήρα. Για να πραγματοποιηθεί το test αυτό συνδέσαμε και τα 4 esc στους 4 κινητήρες και αντίστοιχα στο *Arduino Board* μέσω των *Pins* (3, 5, 6, 9).

Listing 3. ESC testing c++ language technical details

```

/*
-----
MSC HEP 2013-2014
-----
*/
/*
-----
Project      : Quadcopter
File         : Testing_esc.ino
Description   : This code tests the correct behavior of the motors
Author       : Monahopoulos Konstantinos
-----
*/
/*
-----
Includes
-----
*/
#define <Servo.h>
/*
-----
Definitions
-----
*/
#define MAX_SIGNAL 2000
#define MIN_SIGNAL 900 //MIN_SIGNAL must be smaller than min of calibration
(1000)
/*
-----
Assign Classes
-----

```

```

*/
Servo myescFL;
Servo myescFR;
Servo myescRL;
Servo myescRR;

/*
-----
----- Function Declaration -----
-----

*/
void arm(void);
int incomingByte;

/*
-----
----- Main Code -----
-----

*/
void setup()
{
    Serial.begin(9600);

    /*Attach the pins to escs*/
    myescFL.attach(3);
    myescFR.attach(5);
    myescRL.attach(6);
    myescRR.attach(9);

    /*arming the escs..*/
    arm();

    Serial.println("Please enter value");
}

void loop()
{
    if (Serial.available() > 0) {

        delay(100);

        /*read incoming byte*/
        incomingByte=Serial.parseInt();
        Serial.print("Write to ESC: ");Serial.print(" ");
        Serial.println(incomingByte);
        Serial.println("Please enter value");
    }

    if(incomingByte!=0){
        myescFL.writeMicroseconds(incomingByte);
        myescFR.writeMicroseconds(incomingByte);
        myescRL.writeMicroseconds(incomingByte);
        myescRR.writeMicroseconds(incomingByte);
        delay(100);
    }
}

/*
-----
----- Function Prototyping -----
-----

```

```

*/
void arm(void) {
    Serial.print("Arming the esc ..");
    myescFL.writeMicroseconds(MIN_SIGNAL);
    myescFR.writeMicroseconds(MIN_SIGNAL);
    myescRL.writeMicroseconds(MIN_SIGNAL);
    myescRR.writeMicroseconds(MIN_SIGNAL);
    delay(2000);
}

```

Στην αρχή του κώδικα ορίζουμε τα pins που έχουμε συνδέσει τα *ESC* και δημιουργούμε τα απαραίτητα *assignments* της κλάσης *Servo* για τη χρήση των τεσσάρων μοτέρ. Με τη συντομογραφία (*FL, FR, RL, RR*) εννοούμε αντίστοιχα (*Front Left, Front Right, Rear Left, Rear Right*), τα οποία περιγράφουν τη θέση του κάθε μοτέρ στο *Quadcopter*. Εκτός από τη διαδικασία αρχικοποίησης της σειριακής μέσω της εντολής *Serial.Begin(BaudRate)* έχουμε δημιουργήσει μία συνάρτηση που είναι αρμόδια για το όπλισα των *ESC*. Κάθε *ESC* πριν ξεκινήσει τη λειτουργία του, κάθε φορά που τροφοδοτείται, πρέπει να οπλίζεται. Αυτό πραγματοποιείται θέτοντας σε όλα τα *ESC* τιμή μικρότερη της ελάχιστης τιμής που θέσαμε στη διαδικασία *calibration*. Με αυτό τον τρόπο οπλίζουμε τα *ESC* μέσω της συνάρτησης *arm()* και στη συνέχεια σε κάθε επανάληψη εντός της *loop*, ζητάμε από το χρήστη να εισάγει μία τιμή περιστροφής του κινητήρα (προφανώς ενδιάμεσα στα 1000 - 2000), εκτός και εάν θέλει να σταματήσει τη περιστροφή των μοτέρ θέτοντας τιμή μικρότερη από 1000.

3.6 ΜΠΑΤΑΡΙΑ ΠΟΛΥΜΕΡΟΥΣ ΛΙΘΙΟΥ

Μία μπαταρία πολυμερούς λιθίου ή όπως αποκαλείται πολυμερούς ιόντων - λιθίου (*LiPo*) είναι μία επαναφορτιζόμενη μπαταρία σε μαλακή εύκαμπτη μορφή. Σε αντίθεση με τις κυλινδρικές και τις πρισματικές, οι μπαταρίες τύπου *LiPo* είναι ελαφρύτερες, χωρίς να παρέχουν αξιοσημείωτη στιβαρότητα.

Η ονομαστική συντόμευση «πολυμερές λιθίου» μπορεί να ερμηνευτεί με δύο τρόπους. Το πολυμερές λίθιο αφορά μία αναπτυσσόμενη τεχνολογία που χρησιμοποιούσε πολυμερές ηλεκτρολύτη αντί για τον κοινό υγρό ηλεκτρολύτη. Το αποτέλεσμα είναι ένα πλαστικό «cell», το οποίο αποτελεί ένα λεπτό και εύκαμπτο κατασκεύασμα ενώ μπορεί να παραχθεί σε διάφορα σχήματα. Η δεύτερη ερμηνεία διαδόθηκε όταν κατασκευαστές χρησιμοποιούσαν πολυμερές για την κατασκευή των cells ιόντων λιθίου σε μορφή μικρής θήκης. Σήμερα αυτή η μορφή συναντάται συχνότερα

αντικαταστόντας τον πολυμερή ηλεκτρολύτη με πολυμερές περίβλημα. Αυτά τα *cells* είναι γνωστά σαν *LiPo* ενώ κατασκευάζονται σε σειρά ή παράλληλα [Ref 10] .



Εικόνα 30. Μορφές μπαταρίας τύπου *Lipo*.

Οι μπαταρίες τύπου *LiPo* είναι ο κύριος λόγος που η ηλεκτρική πτήση αποτελεί πλέον μία πολύ βιώσιμη επιλογή σε σχέση με μοντέλα τροφοδοτούμενα από καύσιμο. Οι μπαταρίες τύπου *RC LiPo* έχουν τρία κύρια σημεία που τις κάνουν την καλύτερη λύση για *Quadcopter* σε σχέση με επαναφορτιζόμενες μπαταρίες τύπου *NiCad* ή *NiMH*

- Οι μπαταρίες *RC LiPo* είναι ελαφριές και μπορούν να κατασκευαστούν σε κάθε σχήμα και μέγεθος.
- Οι μπαταρίες *RC LiPo* έχουν μεγάλη χωρητικότητα.
- Οι μπαταρίες *RC LiPo* έχουν υψηλό ρυθμό αποφόρτισης ώστε να τροφοδοτήσουν μοτέρ υψηλών απαιτήσεων.

Εν συντομία οι *LiPo* παρέχουν υψηλή χωρητικότητα σε σχέση με το βάρος τους σε κάθε σχήμα και μέγεθος. Υπάρχουν βέβαια και κάποια αρνητικά σημεία, όπως αυτά που αναφέρονται παρακάτω.

- Οι μπαταρίες *RC LiPo* είναι ακόμη ακριβές σε σχέση με τις μπαταρίες τύπου *NiCad* and *NiMH* αλλά η τιμή τους ελαττώνεται διαρκώς.
- Παρόλο που βελτιώνονται διαρκώς δεν κρατούν πολύ . Μόνο 300-400 κύκλους φόρτισης αποφόρτισης.

- Ζητήματα ασφαλείας - λόγω του πτητικού ηλεκτρολύτη που χρησιμοποιείται στις *LiPo*, μπορούν να εκραγούν και να πιάσουν φωτιά όταν δεν γίνεται σωστή χρήση.
- Οι μπαταρίες *RC LiPo* απαιτούν ειδική φροντίδα.

Μια πραγματική *LiPo* μπαταρία δεν χρησιμοποιεί υγρό ηλεκτρολύτη αλλά αντιθέτως στεγνό πολυμερές διαχωριστικό που μοιάζει με λεπτό φιλμ. Αυτός ο διαχωριστής διατηρεί τη μορφή σάντουντς μεταξύ ανόδου και καθόδου επιτρέποντας την ανταλλαγή ιόντων λιθίου. Έτσι προέκυψε το όνομα πολυμερές λιθίου. Αυτή η μέθοδος επιτρέπει πολύ λεπτό *design* και μία μεγάλη γκάμα από σχήματα και μεγέθη. Το πρόβλημα με την κατασκευή μίας *LiPo* μπαταρίας είναι η ανταλλαγή ιόντων μέσα από το στεγνό πολυμερές αφού είναι αργή και έτσι ελαττώνει το ρυθμό φόρτισης και αποφόρτισης. Αυτό το πρόβλημα μπορεί να ξεπεραστεί με την αύξηση της θερμοκρασίας της μπαταρίας ώστε να επιτρέπει ταχύτερη ανταλλαγή ιόντων μέσα από το πολυμερές μεταξύ ανόδου και καθόδου [Ref 11].

Για το *Quadcopter* χρησιμοποιούμε μία 3s (*DragonRed*) μπαταρία τύπου *LiPo*, 35C, με συνολική τάση 11.1 volt. Σύμφωνα με τα χαρακτηριστικά υπάρχουν τρείς κύριες μεταβλητές που πρέπει να ελέγχουμε κατά την επιλογή μπαταρίας. Αυτές είναι : η τάση ,η χωρητικότητα και ρυθμός εκφόρτωσης.



Εικόνα 31. 3S - 35C – 3000mah *DragonRed* μπαταρία τύπου *Lipo*.

3.6.1 ΤΑΣΗ ΜΠΑΤΑΡΙΑΣ

Σε αντίθεση με τα συμβατικά *NiCad* ή *NiMH* cells μπαταριών που έχουν τάση 1.2 *volt*s το καθένα , οι μπαταρίες τύπου *LiPo* παρέχουν έως και 3.7 *volt*s /cell. Το όφελος είναι ότι λιγότερα *cells* μπορούν να χρησιμοποιηθούν για ένα *battery pack* αφού για περιπτώσεις μικρότερων μικρό - αεροσκαφών μία μπαταρία 3.7 volt επαρκεί. Εκτός από τα πιο μικρά *RC* μοντέλα, σε όλες τις άλλες περιπτώσεις οι μπαταρίες *RC LiPo* θα έχουν τουλάχιστον 2 *cells* σε σειρά για την παροχή υψηλότερων τιμών τάσης. Για μεγαλύτερα μοντέλα, ο αριθμός των *cells* σε σειρά μπορεί να φτάσει τα 6 ή και περισσότερα.

- $3.7 \text{ volt battery} = 1 \text{ cell} \times 3.7 \text{ volts (1S)}$
- $7.4 \text{ volt battery} = 2 \text{ cells} \times 3.7 \text{ volts (2S)}$
- $11.1 \text{ volt battery} = 3 \text{ cells} \times 3.7 \text{ volts (3S)}$
- $14.8 \text{ volt battery} = 4 \text{ cells} \times 3.7 \text{ volts (4S)}$
- $18.5 \text{ volt battery} = 5 \text{ cells} \times 3.7 \text{ volts (5S)}$
- $22.2 \text{ volt battery} = 6 \text{ cells} \times 3.7 \text{ volts (6S)}$
- $29.6 \text{ volt battery} = 8 \text{ cells} \times 3.7 \text{ volts (8S)}$
- $37.0 \text{ volt battery} = 10 \text{ cells} \times 3.7 \text{ volts (10S)}$
- $44.4 \text{ volt battery} = 12 \text{ cells} \times 3.7 \text{ volts (12S)}$

Πρέπει να τονίσουμε ότι υπάρχουν περιπτώσεις που μπαταρίες συνδέονται παράλληλα για να αυξηθεί η χωρητικότητα. Αυτό συμβολίζεται με τον ακολουθούμενο αριθμό - δείκτη “P”. Για παράδειγμα 2S2P συμβολίζει 2 *cells* σε σειρά παράλληλα με άλλα 2 *cells* επίσης σε σειρά [Ref 11] .

3.6.2 ΧΩΡΗΤΙΚΟΤΗΤΑ ΜΠΑΤΑΡΙΑΣ

Η χωρητικότητα δείχνει πόση ενέργεια μπορεί η μπαταρία να περιέχει και μετριέται σε μιλιαμπερώρες (*mAh*). Δηλαδή αντιπροσωπεύει το ποσό του φορτίου που μπορεί να απορριφθεί από τη μπαταρία σε χρόνο μίας ώρας. Γνωρίζουμε ότι το μέγιστο ρεύμα που μπορεί να καταναλωθεί σε κάθε μοτέρ είναι 20 *Amp*. Μέσω αυτού, καθώς και μέσω της τιμής των χαρακτηριστικών της μπαταρίας, υπολογίζουμε το μέγιστο χρόνο πτήσης στη μέγιστη ισχύ. Το μέγιστο ρεύμα κατανάλωσης ταυτόχρονης λειτουργίας των τεσσάρων μοτέρ, θα είναι :

$$20(\text{MaxAh}) * 4(\text{esc}) = 80A$$

Εφόσον η μπαταρία μας είναι $3000mAh$, τότε έως την αποφόρτιση ο χρόνος που θα περάσει θα είναι:

$$\frac{3(Ah)}{80(A)} = 0.0375(\text{hours})$$

Το οποίο σε δευτερόλεπτα υπολογίζεται ως:

$$0.0375(\text{hours}) * 3600 = 135(\text{sec})$$

Και σε λεπτά :

$$135(\text{sec}) / 60(\text{sec/ min ute}) = 2.25(\text{min utes})$$

Αν υποθέσουμε ότι το *Quadcopter* στη λειτουργία του *hovering*, πρέπει να καταναλώνει το 30% της ισχύος, τότε το κάθε μοτέρ θα καταναλώνει $6Ah$. Έτσι, υπολογίζουμε το χρόνο πτήσης με χρήση των παραπάνω συνιστωσών μέσω των τύπων που ακολουθούν. Το ρεύμα, καταναλώνοντας τη απαραίτητη ισχύ για hovering και στα τέσσερα μοτέρ, θα είναι :

$$6(\text{MaxAh}) * 4(\text{esc}) = 24A$$

Εφόσον η μπαταρία που χρησιμοποιούμε είναι $3000mAh$, τότε έως την αποφόρτιση της ο χρόνος που θα περάσει είναι:

$$\frac{3(Ah)}{24(A)} = 0.125(\text{hours})$$

Το οποίο σε δευτερόλεπτα υπολογίζεται ως:

$$0.125(\text{hours}) * 3600 = 450(\text{sec})$$

Και σε λεπτά :

$$450(\text{sec}) / 60(\text{sec/ min ute}) = 7.5(\text{min utes})$$

Το κύριο ζήτημα γύρω από αυτό το θέμα είναι ότι για την αύξηση του χρόνου πτήσης απαιτείται μπαταρία μεγαλύτερης χωρητικότητας. Βέβαια λόγω περιορισμών μεγέθους και βάρους υπάρχει και περιορισμός στη χωρητικότητα [Ref 11].

3.6.3 ΡΥΘΜΟΣ ΕΚΦΟΡΤΙΣΗΣ

Η τρίτη παράμετρος είναι ο ρυθμός εκφόρτωσης. Δηλαδή το πόσο γρήγορα μπορεί να εκφορτιστεί η μπαταρία με ασφάλεια. Όσο πιο γρήγορα τα ιόντα πηγαίνουν από την άνοδο στην κάθοδο, τόσο μεγαλύτερος είναι ο ρυθμός εκφόρτωσης ή όπως αποκαλείται η παράμετρος «C». Μία μπαταρία με βαθμολογία $10C$ θα σήμαινε ότι είναι εφικτή η εκφόρτωση με ρυθμό 10 φορές μεγαλύτερο από τη χωρητικότητα της μπαταρίας (*10 times more than the capacity of the pack*). Χρησιμοποιώντας τη μπαταρία που έχουμε εφοδιαστεί, με $3000mAh$ και $35C$ ως ρυθμό αποφόρτισης, αυτομάτως σημαίνει ότι η μπαταρία μπορεί να υποστεί φορτίο ίσο με :

$$3000(mAh) * 35(C) = 105.000(mAh) = 105Ah$$

Από καθαρά θεωρητικής άποψης, η παραπάνω τιμή ισούται με

$$105.000(mAh) / 60 = 1750(mAh)$$

κατανάλωση ρεύματος το λεπτό, άρα η μπαταρία θα είχε ολοκληρωτικά αποφορτιστεί σε

$$3000(mAh) / 1750(mAh) = 1.7(\text{min utes})$$

Από ότι παρατηρούμε η τιμή των 1,7 λεπτών ισχύει για το μέγιστο ρεύμα των $105Ah$. Άρα μέσω των παραπάνω υπολογισμών πρέπει η μέγιστη τιμή κατανάλωσης ρεύματος να μην υπερβαίνει τη τιμή των $105Ah$. Στη περίπτωση μας η μέγιστη τιμή κατανάλωσης ρεύματος που υπολογίζεται από την κατανάλωση των μοτέρ είναι $80Ah$, όπως υπολογίσαμε και παραπάνω. Με αυτό το κριτήριο η μπαταρία που διαλέξαμε είναι εντός ορίων και ανταποκρίνεται ικανοποιητικά στις απαιτήσεις. Ο ρυθμός ταχείας ή εκρηκτικής αποφόρτισης αφορά το ρυθμό αποφόρτισης για μικρά διαστήματα διάρκειας λίγων δευτερολέπτων στα οποία η παροχή ισχύος είναι αυξημένη [Ref 11].

3.6.4 ΟΡΙΑ ΜΠΑΤΑΡΙΑΣ ΚΑΙ ΕΛΕΓΧΟΣ ΑΠΟΦΟΡΤΙΣΗΣ

Ένας πολύ καλός κανόνας προσέγγισης, είναι ο κανόνας του ποσοστού "80% rule". Αυτό σημαίνει ότι δεν πρέπει ποτέ να αποφορτίσουμε τη LiPo μπαταρία κάτω από το 80% της χωρητικότητας, για να είμαστε πάντα ασφαλείς. Στην μπαταρία που κατέχουμε η χωρητικότητα είναι 3000 mAh, άρα δεν πρέπει να υπερβούμε κατανάλωση, μεγαλύτερη των $80\% \times 3000mAh = 2400mAh$.

Για να εξακριβώσουμε το ποσό της χωρητικότητας, μια καλή προσέγγιση είναι να μετρήσουμε συνολική τάση ανοιχτού κυκλώματος μέσω ενός ψηφιακού μετρητή τάσης ή να μετρήσουμε ξεχωριστά τη τάση στα cells. Μια αποφόρτιση των 80% θα δώσει κατά προσέγγιση τάση ανοιχτού κυκλώματος ίση με 3.72 έως 3.74 volts/cell. Σε μία 3S LiPo μπαταρία, όπως αυτή που χρησιμοποιούμε, ύστερα από μία ασφαλή πτήση, η συνολική τάση ανοιχτού κυκλώματος σε όλα τα cells θα είναι περίπου 11.2 volts στο ποσοστό του 80%. Για λόγους ασφαλείας έχουμε τοποθετήσει έναν ψηφιακό μετρητή τάσης, συνδεδεμένο μόνιμα στη μπαταρία κατά τη διάρκεια της πτήσης. Όταν η συνολική τάση αποφόρτισης υπερβεί το 80% της συνολικής αρχικής τάσης, τότε μία ηχητική ειδοποίηση μας ενημερώνει για το τέλος της πτήσης έως ότου επαναφορτίσουμε τη μπαταρία.



Εικόνα 32. Ψηφιακός μετρητής τάσης πραγματικού χρόνου.

Οι ψηφιακοί μετρητές τάσης είναι πολύ χρήσιμοι για να μπορούμε γρήγορα να αντιληφθούμε το επίπεδο φόρτισης της μπαταρίας. Η σύνδεση είναι απλή αφού μέσω ενός καλωδίου (*JST*) που εντοπίζεται στη μπαταρία, μπορούμε να συνδέσουμε το ψηφιακό μετρητή [Ref 11].

3.6.5 ΦΟΡΤΙΣΗ ΜΠΑΤΑΡΙΑΣ

Οι μπαταρίες τύπου *LiPo*, *LiIon*, και *LiFe* έχουν προφανώς πολύ διαφορετικά χαρακτηριστικά από τις συμβατικές επαναφορτιζόμενες μπαταρίες. Η σωστή φόρτιση των μπαταριών λιθίου είναι κρίσιμη για τη διάρκεια ζωής της μπαταρίας αλλά και για την ασφάλεια της. Μία μπαταρία τύπου *LiPo* 3.7 volt φορτίζεται στο 100% όταν φτάσει τα 4.2 volts, ενώ φόρτιση πέρα από αυτό το όριο θα καταστρέψει την μπαταρία ενώ μπορεί να προκαλέσει μέχρι και φωτιά. Είναι απαραίτητο ο φορτιστής να είναι κατάλληλος για *LiPo* μπαταρίες και να έχει επιλεγεί η σωστή τάση φόρτισης ανάλογα πάντα με τον αριθμό των *cells*. Για την φόρτιση της μπαταρίας εφοδιαστήκαμε με τον φορτιστή 105 Simple Li-Poly battery Balance Charger for 2-3 Cells lipo της *Mystery*, κατάλληλο για ασφαλείς φορτίσεις μπαταριών τύπου *LiPo*. Ο φορτιστής αυτός υποστηρίζει 2S και 3S μπαταρίες, συνδέοντάς τες σε ξεχωριστές θύρες για την ασφάλεια τόσο της μπαταρίας, όσο και τη δικιά μας.



Eικόνα 33. Mystery 2-3 Cells φορτιστής lipo μπαταριών.

Όλες οι μπαταρίες τύπου *LiPo* χρησιμοποιούν μέθοδο φόρτισης σταθερής τάσης και σταθερού ρεύματος. Αυτό σημαίνει ότι ένα σταθερό ρεύμα εφαρμόζεται στην μπαταρία κατά τη διάρκεια του πρώτου μέρους του κύκλου φόρτισης. Καθώς η τάση της μπαταρίας πλησιάζει το 100% της τάσης φόρτισης, ο φορτιστής αυτόματα ξεκινά να ελαττώνει το ρεύμα φόρτισης, εφαρμόζοντας ένα σταθερό επίπεδο τάσης για το υπόλοιπο του κύκλου φόρτισης. Ο φορτιστής θα σταματήσει τη διαδικασία της φόρτισης όταν η τάση της μπαταρίας γίνει ίση με την μέγιστη τάση του φορτιστή (4,2 volt/cell) ολοκληρώνοντας τους κύκλους φόρτισης [Ref 11].

3.7 Ο ΔΙΑΥΛΟΣ I²C

Ο δίαυλος I²C είναι ένας σειριακός δίαυλος που δημιουργήθηκε από τη *Philips* και χρησιμοποιείται για την σύνδεση περιφερειακών μικρής ταχύτητας σε *motherboard*, *embedded systems*, κινητά τηλέφωνα ή άλλες ηλεκτρονικές συσκευές. Ο δίαυλος I²C δεν χρησιμοποιείται μόνο για την επικοινωνία συσκευών που βρίσκονται πάνω σε ένα τυπωμένο κύκλωμα, αλλά και για την επικοινωνία συσκευών που συνδέονται με καλώδια.

Χρησιμοποιεί μόνο δύο καλώδια, τα οποία είναι αμφίδρομης κατεύθυνσης: Τα *SCL* και *SDA*. Η γραμμή *SCL* είναι η γραμμή ρολογιού, ενώ η *SDA* είναι η γραμμή δεδομένων. Οι γραμμές αυτές συνδέονται σε όλες τις συσκευές στον δίαυλο I²C. Προφανώς εκτός από τα παραπάνω καλώδια, απαιτείται και ένα τρίτο καλώδιο, το οποίο είναι η γείωση (*GND*) ή 0 V. Επίσης μπορεί να υπάρχει και ένα τέταρτο καλώδιο το οποίο είναι η γραμμή τροφοδοσίας, με την οποία τροφοδοτούνται με ισχύ οι διάφορες συσκευές που υπάρχουν στο δίκτυο. Τυπικές τάσεις που χρησιμοποιούνται στο δίαυλο είναι τα +5V ή 3,3V, αν και επιτρέπονται συστήματα με διαφορετικές τάσεις (συνήθως στην περιοχή από 1,2V-5,5V). Ο μέγιστος αριθμός κόμβων, που μπορούν να συνδεθούν στον δίαυλο, περιορίζεται από τον αριθμό των διαθέσιμων διευθύνσεων (θα επεξηγηθεί παρακάτω), αλλά και από τη συνολική χωρητικότητα του διαύλου, η οποία π.χ. για την *Standard mode* δεν πρέπει να υπερβαίνει τα 400pF, το οποίο και περιορίζει τις πρακτικές αποστάσεις επικοινωνίας. Παρακάτω γίνεται η σύγκριση διαφόρων πραγματοποιήσεων του διαύλου (πηγή). Στην πράξη μπορούμε π.χ. να πετύχουμε μεγαλύτερα μήκη διαύλου μειώνοντας την ταχύτητα.

ΠΙΝΑΚΑΣ 6. Πληροφορίες I2C επικοινωνίας .

| Δίαυλος | Ρυθμός Δεδομένων (bit/sec) | Μήκος Διαύλου (μέτρα) | Παράγοντας περιορισμού μήκους διαύλου | Μέγιστος Αριθμός κόμβων | Παράγοντας Περιορ. Αριθμ. κόμβων |
|-------------------------|----------------------------|-----------------------|---------------------------------------|-------------------------|----------------------------------|
| I2C | 400k | 2 | Χωρητικότητα καλωδίωσης | 20 | 400pF max |
| I2C με Οδηγούς (Buffer) | 400k | 100 | Καθυστέρηση διάδοσης | Οποιοσδήποτε | Κανένας περιορισμός |
| I2C | 3,4M | 0,5 | Χωρητικότητα καλωδίωσης | 5 | 100pF max |

Αμφότερες, οι γραμμές *SCL* και *SDA* είναι τύπου ανοικτού απαγωγού (*open drain*) ή ανοικτού συλλέκτη (*open collector* στον κόσμο των *TTL*). Αυτό σημαίνει ότι και οι δύο αυτές γραμμές πρέπει να συνδέονται η κάθε μία, μόνο οι γραμμές και όχι κάθε συσκευή που συνδέεται στο δίαυλο ξεχωριστά, με μία αντίσταση στην γραμμή τροφοδοσίας. Η αντίσταση αυτή ονομάζεται αντίσταση τερματισμού. Η τιμή των αντιστάσεων δεν είναι κρίσιμη, αλλά μαζί με την χωρητικότητα του διαύλου, επηρεάζει την μέγιστη ταχύτητα λειτουργίας του διαύλου. Μεγάλες χωρητικότητες του διαύλου, μπορούν να αντισταθμιστούν με μικρές αντιστάσεις τερματισμού. Συνηθισμένες τιμές αντιστάσεων είναι από $1K\Omega$ έως $10K\Omega$. Οι αντιστάσεις αυτές δεν μπορούν να απουσιάζουν, διότι τότε οι γραμμές θα είναι μονίμως σε κατάσταση λογικού 0 και δίαυλος δεν θα δουλεύει.

3.7.1 ΛΟΓΙΚΕΣ ΣΤΑΘΜΕΣ ΤΩΝ ΓΡΑΜΜΩΝ SCL KAI SDA

Επειδή μέσω του δίαυλου *I²C* συνδέονται ποικιλά συσκευών, διαφόρων τεχνολογιών (*CMOS*, *NMOS*, Διπολικής τεχνολογίας), οι οποίες μπορεί να έχουν διαφορετικές τάσεις λειτουργίας, οι στάθμες του λογικού 0 (*Low*) και λογικού 1 (*High*) σε όλες τις νέες συσκευές δεν είναι σταθερές, αλλά εξαρτώνται από την τάση τροφοδοσίας. Έτσι τα κατώφλια του λογικού 0 και του λογικού 1 τοποθετούνται στο 30% και το 70% της τάσης τροφοδοσίας αντίστοιχα. Δηλ. μία τάση στην περιοχή 0-0,3VDD θεωρείται λογικό 0 (*Low*), ενώ μία τάση στην περιοχή 0,7 VDD, θεωρείται λογικό 1 (*High*). Παλαιότερα τα κατώφλια του λογικού 0 και 1 είχαν τοποθετηθεί στα 1,5V και 3,0V αντίστοιχα.

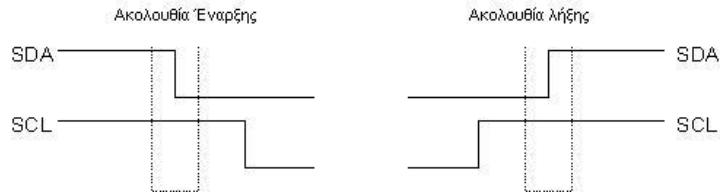
3.7.2 ΚΥΡΙΟΙ ΚΑΙ ΥΠΟΤΕΛΕΙΣ (MASTERS AND SLAVES)

Οι συσκευές στον δίαυλο *I²C* είναι είτε Κύριοι (*Masters*) είτε Υποτελείς (*Slave*). Η *Master* συσκευή είναι αυτή που ελέγχει και οδηγεί τη γραμμή ρολογιού *SCL* (παράγει τους παλμούς ρολογιού). Οι *Slave* συσκευές είναι αυτές που ανταποκρίνονται στις συσκευές *Master*. Μία συσκευή *Slave* δεν μπορεί να ξεκινήσει μία μεταφορά πάνω στο δίαυλο, μόνο μία συσκευή *Master* μπορεί. Σε έναν δίαυλο μπορεί να είναι συνδεμένες πολλές *Master* και πολλές *Slaves* συσκευές. Και οι *Master* και οι *Slave* συσκευές μπορούν να μεταφέρουν δεδομένα στον δίαυλο, αλλά μόνο οι *Master* συσκευές ελέγχουν την μεταφορά.

3.7.3 ΤΟ ΦΥΣΙΚΟ ΠΡΩΤΟΚΟΛΛΟ ΤΟΥ ΔΙΑΥΛΟΥ *I²C*

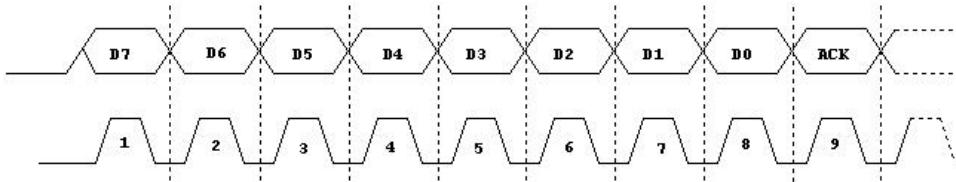
Όταν μία *Master* συσκευή επιθυμεί να επικοινωνήσει με μία *Slave* συσκευή, ξεκινά στέλνοντας στον δίαυλο μία ακολουθία έναρξης (*start sequence*). Η ακολουθία έναρξης, είναι μία από τις δύο ειδικές ακολουθίες που ορίζονται στο δίαυλο *I²C*, ή άλλη είναι η ακολουθία λήξης (*stop sequence*). Οι ακολουθίες έναρξης και λήξης διαφέρουν στο ότι είναι οι μοναδικές θέσεις στις οποίες επιτρέπεται να αλλάζει η γραμμή δεδομένων (*SDA*), ενόσω η γραμμή ρολογιού είναι σε κατάσταση λογικού 1 (*high*). Όταν μεταφέρονται δεδομένα, η γραμμή *SDA* πρέπει να παραμένει σταθερή και να μην αλλάζει όσο η γραμμή ρολογιού είναι *high*. Οι ακολουθίες έναρξης και λήξης σημαδεύουν την έναρξη και τη λήξη μιας μεταφοράς με μία *slave* συσκευή. Δηλαδή ο

δίαυλος θεωρείται ότι είναι αποσχολημένος, μετά από μία ακολουθία έναρξης και ελεύθερος λίγο χρόνο μετά την ακολουθία λήξης.



Εικόνα 34. Ακολουθίες Έναρξης – Λήξης.

Τα δεδομένα μεταφέρονται σε ακολουθίες των 8 bit. Τα bit τοποθετούνται στη γραμμή *SDA*, ξεκινώντας από το περισσότερο σημαντικό bit (*MSB*). Η γραμμή *SCL* πάλλεται *high* και μετά *low*. Για κάθε 8 bit δεδομένων που μεταφέρονται, η συσκευή που λαμβάνει στέλνει πίσω ένα bit επιβεβαίωσης (*ACK*). Έτσι στην πραγματικότητα απαιτούνται 9 παλμοί ρολογιού, για την μεταφορά των 8 bit κάθε byte δεδομένων. Εάν η συσκευή που λαμβάνει, στείλει πίσω ένα *low* bit επιβεβαίωσης (*ACK*), τότε έχει λάβει τα δεδομένα και είναι έτοιμη να λάβει το επόμενο byte δεδομένων. Εάν στείλει πίσω ένα *high* bit επιβεβαίωσης (που συμβολίζεται και με *NACK*, από τη φράση *Not Acknowledged*), αυτό δείχνει ότι η συσκευή που λαμβάνει, δεν μπορεί να λάβει περαιτέρω δεδομένα και η *master* συσκευή πρέπει να τερματίσει την αποστολή δεδομένων, εκπέμποντας μία ακολουθία λήξης.

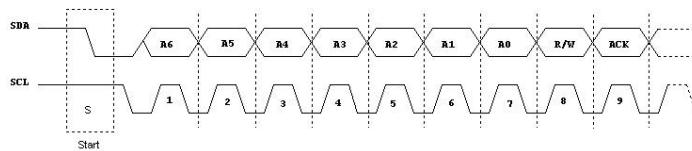


Εικόνα 35. Αποστολή Byte.

3.7.4 ΔΙΕΥΘΥΝΣΙΟΔΟΤΗΣΗ ΣΥΣΚΕΥΩΝ ΤΟΥ I^C ΔΙΑΥΛΟΥ

Σε όλες τις *slave* συσκευές που συνδέονται στον δίαυλο, έχει αποδοθεί ένας αριθμός σαν διεύθυνση. Οι *master* συσκευές δεν είναι απαραίτητο να έχουν διεύθυνση, εκτός εάν υπάρχουν πολλές *master* συσκευές στον δίαυλο (περιβάλλον *Multi-master*). Οι *master* συσκευές μπορούν να διαλέξουν αυθαίρετα μία από τις συνδεμένες *slave*, για επικοινωνία, χρησιμοποιώντας τη διεύθυνσή της. Οι διευθύνσεις των συσκευών του I^C διαύλου είναι είτε 7 bit (θεωρητικά έως 128 συσκευές στο δίαυλο), είτε 10 bit (θεωρητικά έως 1204 συσκευές στο δίαυλο).

Εδώ θα μιλήσουμε για την περίπτωση που η διευθυνσιοδότηση είναι 7-bit. Θεωρητικά με 7 bit μπορούμε να έχουμε έως και 128 συσκευές στο δίαυλο. Επειδή όμως ορισμένες διευθύνσεις χρησιμοποιούνται για ειδικούς σκοπούς, μόνο 112 διευθύνσεις είναι διαθέσιμες στην 7-μπιτη διευθυνσιοδότηση. Π.χ η I^C διεύθυνση 0, είναι γενική κλήση προς όλες τις συσκευές. Στον παρακάτω πίνακα φαίνονται οι διευθύνσεις που είναι δεσμευμένες για ειδικές χρήσεις.



Εικόνα 36. Αποστολή Διεύθυνσης 7 bit

Όταν στέλνουμε την διεύθυνση A₆, A₅, A₄, A₃, A₂, A₁, A₀ των 7-bit της συσκευής με την οποία θέλουμε να επικοινωνήσουμε, ακόμη και τότε στέλνουμε 8 bit. Το επιπλέον (R/W) bit χρησιμεύει να πληροφορήσει την slave συσκευή, εάν η *master* συσκευή πρόκειται να γράψει ή να διαβάσει από αυτήν. Εάν το bit είναι μηδέν η *master* συσκευή πρόκειται να γράψει. Εάν είναι 1 (ένα) πρόκειται να διαβάσει από την *slave*. Τα 7 bit της διεύθυνσης τοποθετούνται στα 7 πάνω bit του byte, ενώ το bit ανάγνωσης/εγγραφής (R/W) στο λιγότερο σημαντικό bit (LSB).

3.7.5 THE I^C SOFTWARE PROTOCOL

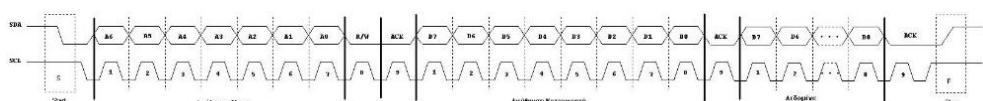
Για να ξεκινήσει μία επικοινωνία, το πρώτο πράγμα που θα κάνει η *master* συσκευή είναι να εκπέμψει την ακολουθία έναρξης. Αυτό ειδοποιεί όλες τις *slave* συσκευές στο δίαυλο, ότι πρόκειται να ξεκινήσει μία εκπομπή και να ακούσουν για την περίπτωση

που είναι γι' αυτές. Μετά η *master* συσκευή θα εκπέμψει την διεύθυνση της συσκευής. Η *slave* συσκευή που η διεύθυνση της ταιριάζει με αυτήν θα συνεχίσει, ενώ όλες οι άλλες θα περάσουν σε αναμονή περιμένοντας την επόμενη επικοινωνία.

Μετά την αποστολή της δ/νσης της *slave* συσκευής, η *master* στέλνει τη διεύθυνση του καταχωρητή της *slave* στον οποίο θέλει να γράψει. Τώρα η *master* μπορεί να στείλει το *Byte* ή τα *Bytes* δεδομένων. Η *master* μπορεί να συνεχίσει να στέλνει δεδομένα, τα οποία θα τοποθετηθούν στις επόμενες θέσεις, επειδή η *slave* συσκευή θα αυξάνει αυτόματα την διεύθυνση του εσωτερικού καταχωρητή μετά την λήψη κάθε *byte*. Όταν η *master* συσκευή στείλει όλα τα δεδομένα, σταματάει την εκπομπή εκπέμποντας μια ακολουθία λήξης. Συγκεκριμένα τα βήματα που ακολουθούνται έχουν ως εξής :

Η *master* συσκευή ..

1. Στέλνει την ακολουθία έναρξης
2. Στέλνει την διεύθυνση της *slave* συσκευής με το *R/W bit low* (άρτια διεύθυνση), δηλώνοντας έτσι ότι θέλει να κάνει εγγραφή δηλ. να στείλει δεδομένα.
3. Στέλνει την διεύθυνση του εσωτερικού καταχωρητή στον οποίο θέλει να γράψει
4. Στέλνει το *byte* δεδομένων
5. Στέλνει (προαιρετικά) οποιονδήποτε αριθμό επιπλέον *byte*
6. Στέλνει την ακολουθία λήξης



Εικόνα 37. Εγγραφή σε *Slave*.

3.7.6 ΑΝΑΓΝΩΣΗ ΑΠΟ ΤΗΝ SLAVE ΣΥΣΚΕΥΗ

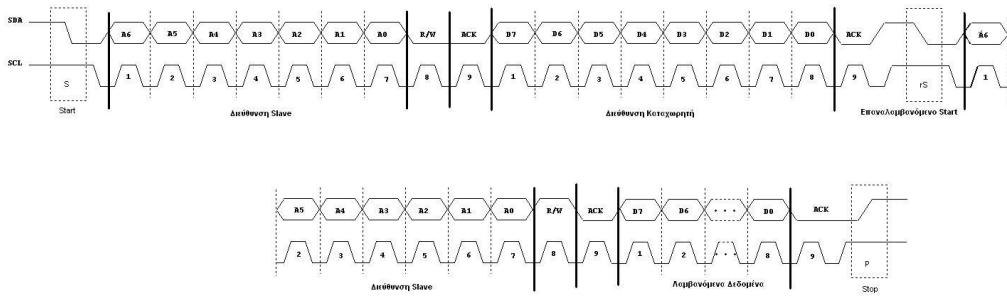
Πριν να διαβάσουμε δεδομένα από μία slave συσκευή, πρέπει να την πληροφορήσουμε ποιον εσωτερικό καταχωρητή της θέλουμε να διαβάσουμε. Έτσι μία ανάγνωση από την slave συσκευή στην πραγματικότητα ξεκινά με μία εγγραφή σ' αυτήν. Έτσι η διαδικασία που ακολουθείται για να διαβάσει μία master από μία slave έχει ως εξής:

H master συσκευή ..

7. Στέλνει την ακολουθία έναρξης
8. Στέλνει την διεύθυνση της slave συσκευής με το R/W bit low (εγγραφή, άρτια διεύθυνση).
9. Στέλνει την διεύθυνση του εσωτερικού καταχωρητή από τον οποίο θέλει να διαβάσει.

Με τα δύο προηγούμενα βήματα, γράφεται ο καταχωρητής - δείκτης της slave συσκευής, σύμφωνα με αυτά που αναφέραμε νωρίτερα για γρήγορη πρόσβαση στα δεδομένα

10. Στέλνει πάλι την ακολουθία έναρξης (επαναλαμβανόμενη έναρξη rS)
11. Στέλνει πάλι την διεύθυνση της slave συσκευής με το R/W bit high (περιττή διεύθυνση), για να δηλώσει ότι επιθυμεί ανάγνωση
12. Διαβάζει τα δεδομένα (ένα ή περισσότερα byte). Μετά από την λήψη κάθε byte, η συσκευή που λαμβάνει επιβεβαιώνει (ACK) τη λήψη.
13. Στέλνει την ακολουθία λήξης



Εικόνα 38. Ανάγνωση από Slave.

3.8 10DOF IMU

Ένα σύστημα *IMU* (*Inertial Measurement Unit*), αποτελείται από ένα επιταχυνσιόμετρο, ένα γυροσκόπιο και διάφορους επιπλέον αισθητήρες (ανάλογα τους βαθμούς ελευθερίας) από τους οποίους λαμβάνοντας τιμές και συνδυάζοντας τες κατάλληλα, μπορούμε να υπολογίσουμε το διάνυσμα κατεύθυνσης του αντικειμένου. Πιο συγκεκριμένα οι τιμές αυτές αποτελούν τόσο δυνάμεις επιτάχυνσης, όσο και δυνάμεις βαρυτικής έλξης, στις οποίες εφαρμόζουμε κατάλληλους αλγόριθμους φιλτραρίσματος για να αποκτήσουμε τα επιθυμητά αποτελέσματα. Τα παραπάνω χαρακτηριστικά κάνουν το *IMU* να αποτελεί έναν πολύ δυνατό μηχανισμό ο οποίος χρησιμοποιείται σε ρομπότ, *UAVs*, αυτόνομα οχήματα και συστήματα σταθεροποίησης εικόνας.

Ένα *10DOF IMU* (με δέκα βαθμούς ελευθερίας), σαν αυτό που χρησιμοποιούμε, εμπεριέχει ουσιαστικά τέσσερις αισθητήρες ενσωματωμένους σε ένα ολοκληρωμένο chip. Πιο συγκεκριμένα εμπεριέχει ένα *ADXL345* (*triple - axis accelerometer*), ένα *L3G4200D* (*triple - axis gyro*), ένα *HMC5883L* (*triple-axis magnetometer*) και ένα *MS5611* (*Barometric Pressure sensor*), δίνοντας όλα μαζί τους δέκα βαθμούς ελευθερίας του αισθητήρα .

Οι βαθμοί ελευθερίας περιγράφουν τις πιθανές κινήσεις ενός αντικειμένου. Στην πραγματικότητα όμως υπάρχουν συνολικά μόνο έξι βαθμοί ελευθερίας, οι τρείς πρώτοι βαθμοί προέρχονται κατά την περιστροφή του αντικειμένου (*roll, pitch, yaw*) και οι τρείς επόμενοι από την μεταφορά του αντικειμένου το χώρο (*forward/backward, left/right, up/down*). Όλοι οι παραπάνω βαθμοί ελευθερίας μπορούν να αποφέρουν πολύ ακριβή αποτελέσματα συνδυάζοντας κατάλληλα τα σήματα των αισθητήρων [Ref 12] .

3.8.1 I2C SCANNER

Συνδέοντας το *10DOF IMU*, πρέπει πρώτα να ερευνήσουμε τις διευθύνσεις του κάθε αισθητήρα, μέσω του *I2C* διαδρόμου. Εάν μια συσκευή (ένας αισθητήρας) βρεθεί, τότε αυτός αναφέρεται στη σειριακή θύρα του *Arduino*. Αυτό το πρόγραμμα που αναπτύξαμε αποτελεί το πρώτο βήμα για την ορθή επικοινωνία του μικροελεγκτή και των συσκευών.

Το πρόγραμμα είναι σε θέση να σαρώνει όλες τις πιθανές 7-bit διευθύνσεις που υποστηρίζονται μέσω της *I2C* επικοινωνίας. Όπως και θεωρητικά εξηγήσαμε παραπάνω στέλνοντας την κατάλληλη παλμοσειρά στη συσκευή, όταν εντοπιστεί κάποιος αισθητήρας με αυτή τη διεύθυνση τότε αυτή μας απαντά με τη τιμή "0". Υστερα, η τιμή αυτή μπορεί να χρησιμοποιηθεί μέσω της συνάρτησης "Wire.begin" που εμπεριέχεται στη βιβλιοθήκη "Wire.h", για την έναυση της επικοινωνίας με τη συσκευή. Όπως αναμένουμε οι συσκευές που πρέπει να εντοπιστούν είναι τρείς, οι οποίες φυσικά γνωρίζουμε εκ των προτέρων ότι είναι με αύξουσα σειρά, για το μαγνητόμετρο (*HMC5883L* = 0x1E), για το επιταχυνσιόμετρο και το γυροσκόπιο (*MPU-6050* = 0x69) και τέλος για το βαρόμετρο (*MS5611* = 0x77). Παρακάτω παρουσιάζουμε τον κώδικα "I2c scanner.ino" καθώς επίσης και τα αποτελέσματα που μας παράγει μέσω της σειριακής.

Listing 4. ESC Scanner Port Report c++ language technical details

```
/*
-----
MSC HEP 2013-2014
-----
*/
/*
-----
Project      : Quadcopter
File         : Scanner_port_report.ino
Description   : This code tests the addresses of the devices connected to
I2C bus.
Author       : Monahopoulos Konstantinos
-----
*/
/*
-----
Includes
-----
*/
#include <Wire.h>
/*
-----
Global Variables
-----
*/
byte start_address = 0;
byte end_address = 128 ;
```

```

/*
-----
Main Code
-----
*/
void setup()
{
    byte count = 0;
    byte rc;
    Wire.begin();

    Serial.begin(9600);
    Serial.println("\nI2C Scanner");
    Serial.print("Scanning I2C bus from ");
    Serial.print(start_address,DEC);  Serial.print(" to ");
    Serial.print(end_address,DEC);
    Serial.println("...");

/*Scan all the addresses*/
for( byte addr = start_address;
     addr <= end_address;
     addr++ ) {

    Wire.beginTransmission(addr); /*Begin transmission on address */
    rc = Wire.endTransmission(); /*Address return*/

    if (addr<16) Serial.print("0"); /*For printing reasons*/
    Serial.print(addr,HEX);

    if (rc==0) { /*Check return value of address */
        Serial.print(" found!"); /*Print Found if address responded
properly*/
    } else {
        Serial.print(" "); Serial.print(rc-2); Serial.print("      ");
        /*Else print "0" */
    }
    Serial.print( (addr%8)==7 ? "\n ":" "); /*Change line per 8 addresses*/
}

Serial.println("\n-----\nPossible devices:");

for( byte addr = start_address;
     addr <= end_address;
     addr++ ) {
    Wire.beginTransmission(addr);
    rc = Wire.endTransmission();

    if (rc == 0) {
        switch (addr) {
            case 0x50: Serial.println("AT24C32/AT24C64 - EEPROM"); break;
            case 0x68: Serial.println("DS1307"); break;
            default:   Serial.print ("Found address: ");
                       Serial.print (addr, DEC);
                       Serial.print (" (0x");
                       Serial.print (addr, HEX);
                       Serial.println (")");
                       count++;
                       break;
        }
    }
}

Serial.print ("Total ");
Serial.print (" device(s) = ");
Serial.println (count, DEC);
Serial.println("\nDone !!");

```

```

    }

void loop()
{
    // Nothing to do here
}

```

Ο κώδικα μας παράγει τα παρακάτω αποτελέσματα, σηματοδοτώντας με τις διευθύνσεις των συσκευών που βρέθηκαν με τη λέξη found δίπλα από τη δεκαεξαδική τιμή της [Ref 13] .

Scanning I2C bus from 0 to 128...

| | | | | | | | |
|------|-----------|------|------|------|------|-----------|-----------|
| 00 0 | 01 0 | 02 0 | 03 0 | 04 0 | 05 0 | 06 0 | 07 0 |
| 08 0 | 09 0 | 0A 0 | 0B 0 | 0C 0 | 0D 0 | 0E 0 | 0F 0 |
| 10 0 | 11 0 | 12 0 | 13 0 | 14 0 | 15 0 | 16 0 | 17 0 |
| 18 0 | 19 0 | 1A 0 | 1B 0 | 1C 0 | 1D 0 | 1E found! | 1F 0 |
| 20 0 | 21 0 | 22 0 | 23 0 | 24 0 | 25 0 | 26 0 | 27 0 |
| 28 0 | 29 0 | 2A 0 | 2B 0 | 2C 0 | 2D 0 | 2E 0 | 2F 0 |
| 30 0 | 31 0 | 32 0 | 33 0 | 34 0 | 35 0 | 36 0 | 37 0 |
| 38 0 | 39 0 | 3A 0 | 3B 0 | 3C 0 | 3D 0 | 3E 0 | 3F 0 |
| 40 0 | 41 0 | 42 0 | 43 0 | 44 0 | 45 0 | 46 0 | 47 0 |
| 48 0 | 49 0 | 4A 0 | 4B 0 | 4C 0 | 4D 0 | 4E 0 | 4F 0 |
| 50 0 | 51 0 | 52 0 | 53 0 | 54 0 | 55 0 | 56 0 | 57 0 |
| 58 0 | 59 0 | 5A 0 | 5B 0 | 5C 0 | 5D 0 | 5E 0 | 5F 0 |
| 60 0 | 61 0 | 62 0 | 63 0 | 64 0 | 65 0 | 66 0 | 67 0 |
| 68 0 | 69 found! | 6A 0 | 6B 0 | 6C 0 | 6D 0 | 6E 0 | 6F 0 |
| 70 0 | 71 0 | 72 0 | 73 0 | 74 0 | 75 0 | 76 0 | 77 found! |
| 78 0 | 79 0 | 7A 0 | 7B 0 | 7C 0 | 7D 0 | 7E 0 | 7F 0 |
| 80 0 | | | | | | | |

Possible devices:

Found address: 30 (0x1E)

Found address: 105 (0x69)

Found address: 119 (0x77)

```
Total device(s) = 3
```

```
Done !!
```

3.8.2 IMU-6050

Ο βασικός αισθητήρας για την ανίχνευση της κίνησης του αντικειμένου είναι ο *MPU-6050*. Η οικογένεια *MPU-6000™* ενσωματώνει το στοιχείο *6-axis Motion Processing™* που εξαλείφει τα προβλήματα που προκύπτουν από κακή ευθυγράμμιση. Η συσκευή συνδυάζει ένα γυροσκόπιο 3 αξόνων και ένα επιταχυνσιόμετρο 3 αξόνων στο ίδιο *chip*. Μαζί με αυτά υπάρχει και ένας *onboard Digital Motion Processor™ (DMP™)* ικανός να εκτελεί σύνθετους αλγορίθμους τύπου *9-axis Motion Fusion*. Αυτοί οι ενσωματωμένοι αλγόριθμοι έχουν πρόσβαση στα δεδομένα εξωτερικών μαγνητόμετρων ή άλλων αισθητήρων χωρίς τη διαμεσολάβηση του κεντρικού επεξεργαστή. Η οικογένεια *MPU-6000™* κατασκευάζεται με το ίδιο 4x4x0.9 mm *QFN* footprint και τον ίδιο αριθμό pin με την οικογένεια *MPU-3000™*. Η συσκευή ενσωματώνει επίσης το στοιχείο *InvenSenseMotionApps™ Platform* το οποίο απλοποιεί την πολυπλοκότητα που σχετίζεται με την επεξεργασία της κίνησης, διαχειρίζεται τους αισθητήρες ώστε να μην επιβαρύνεται το λειτουργικό σύστημα και παρέχει το *API* για την ανάπτυξη εφαρμογών. Για τον εντοπισμό αργής και γρήγορης κίνησης με ακρίβεια χρησιμοποιείται ένα προγραμματιζόμενο γυροσκόπιο με εύρος πλήρους κλίμακας ± 250 , ± 500 , ± 1000 , and $\pm 2000^{\circ}/sec$ (*dps*) και ένα προγραμματιζόμενο επιταχυνσιόμετρο με εύρος πλήρους κλίμακας $\pm 2g$, $\pm 4g$, $\pm 8g$, and $\pm 16g$.

Ο *MPU-6050* υποστηρίζει το δίαυλο επικοινωνίας *I2C* μέχρι τη συχνότητα των $400kHz$ και έχει ένα *VLOGIC* pin που καθορίζει τα επίπεδα τάσης. Ο *MPU-6000* υποστηρίζει *SPI* μέχρι τη συχνότητα των $20MHz$. και έχει ένα μονό pin τροφοδοσίας *VDD*. Το μέγεθος του *package* των *MPU* συσκευών έχει συρρικνωθεί στις διαστάσεις $4x4x0.9mm$ (*QFN*). Επιπλέον συμπεριλαμβάνεται ένας ενσωματωμένος δείκτης θερμοκρασίας και ένας on-chip ταλαντωτής [Ref 14].



Εικόνα 39. Δομικά στοιχεία του *MPU-6050 6DOF sensor*.

3.8.2.1 ΑΝΑΓΝΩΣΗ ΤΙΜΩΝ

Για την ανάγνωση των τιμών από το επιταχυνσιόμετρο και το γυροσκόπιο πρέπει πρώτα να απενεργοποιηθεί η κατάσταση αδράνειας του αισθητήρα. Επίσης πρέπει να αναφέρουμε ότι ο αισθητήρας περιέχει έναν *FIFO* καταχωρητή 1024 byte. Ο αισθητήρας μπορεί να προγραμματιστεί ώστε να καταχωρεί τις τιμές στον *FIFO* καταχωρητής και μετά από εκεί γίνεται ανάγνωση από το *Arduino*.

Ο *FIFO* καταχωρητής δεδομένων χρησιμοποιείται μαζί με το σήμα διακοπής. Αν ο *MPU-6050* τοποθετήσει δεδομένα στον *buffer*, στέλνεται σήμα στο *Arduino* ώστε να ξεκινήσει η διαδικασία ανάγνωσης. Ο *MPU-6050* λειτουργεί σαν slave σε σχέση με τον *Arduino* έχοντας τα pin *SDA* και *SCL* συνδεδεμένα στο *I2C-bus*.

Πέρα από το *I2C-bus* έχει και έναν δικό του ελεγκτή *I2C* στο ρόλο του *master* για ένα δεύτερο (*sub*)-*I2C-bus* που χρησιμοποιεί τα pins *AUX_DA* και *AUX_CL*. Μπορεί να ελέγχει για παράδειγμα ένα μαγνητόμετρο. Τα δεδομένα από το μαγνητόμετρο μπορούν μετά να περάσουν στο *Arduino*. Στην περίπτωσή μας το μαγνητόμετρο είναι ενσωματωμένο στο *chip*.

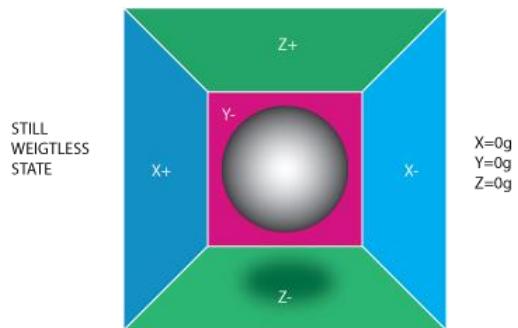
Επίσης ο αισθητήρας περιέχει έναν *Digital Motion Processor*" (*DMP*) ή αλλιώς "*Digital Motion Processing Unit*". Ο *DMP* μπορεί να προγραμματιστεί με *firmware* και μπορεί να κάνει σύνθετους υπολογισμούς με τα δεδομένα που παρέχουν οι αισθητήρες.

Ο *DMP* ("Digital Motion Processor") μπορεί να κάνει γρήγορους υπολογισμούς απευθείας στο ενσωματωμένο σύστημα. Αυτή η στρατηγική ελαττώνει το φορτίο για τον μικροελεγκτή. Ο *DMP* είναι επίσης ικανός να κάνει υπολογισμούς με τις τιμές εξωτερικών αισθητήρων. Για παράδειγμα ένα μαγνητόμετρο συνδεδεμένο στο δεύτερο (*sub*) - *I2C – bus* [Ref 14] .

3.8.2.2 ΘΕΩΡΙΑ ΕΠΙΤΑΧΥΣΙΟΜΕΤΡΟΥ

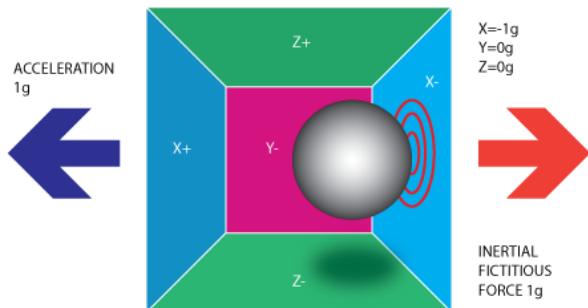
Ένα επιταχυνσιόμετρο είναι μια συμπαγής συσκευή που έχει σχεδιαστεί για τη μέτρηση της μη - βαρυτικής επιτάχυνσης. Όταν το αντικείμενο αλλάζει τη κατάστασή του από αδρανειακή σε κάποια μη στατική, τότε το επιταχυνσιόμετρο είναι σχεδιασμένο να ανταποκρίνεται στις αλλαγές που αφορούν την κίνηση αυτή. Πιο συγκεκριμένα, χρησιμοποιεί μικροσκοπικούς κρυστάλλους οι οποίοι πιέζονται καθώς συμβαίνουν οι αλλαγές δημιουργώντας ένα ποσό τάσης το οποίο αργότερα εισέρχεται στο μικροελεγκτή [Ref 16] .

Για τη καλύτερη κατανόηση θα αναπτύξουμε ένα παράδειγμα λειτουργίας του. Ας φανταστούμε ένα κουτί σε σχήμα κύβου με ένα φανταστικό αντικείμενο, σφαιρικού σχήματος στο κέντρο, όπως φαίνεται στο παρακάτω σχήμα.



Εικόνα 40. Στατική μορφή επιταχυνσιομέτρου .

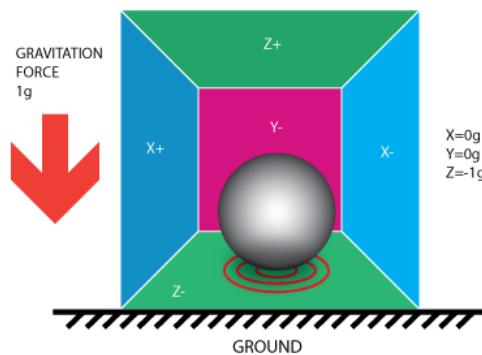
Εάν τοποθετήσουμε το κουτί αυτό σε ένα χώρο δίχως βαρυτικές έλξεις, η μπάλα θα αιωρείται ακριβώς στη θέση ($X=0$, $Y=0$, $Z=0$). Στο παραπάνω σχήμα μπορεί εύκολα κάποιος να παρατηρήσει ότι σε κάθε τοίχωμα έχουμε αναθέσει και έναν άξονα κατεύθυνσης, ενώ το εκάστοτε τοίχωμα αποτελείται από ένα πλήθος αισθητήρων πίεσης. Εάν μετακινήσουμε απότομα τον κύβο προς τα αριστερά, η σφαίρα θα επιταχύνει με επιτάχυνση $1g = 9.8m/s^2$ και θα συγκρουστεί στο τοίχωμα $X-$. Ουσιαστικά αυτό σημαίνει ότι στον X άξονα μετρήθηκε επιτάχυνση $-1g$.



Εικόνα 41. Μη στατική μορφή επιταχυνσιομέτρου X άξονα .

Θα πρέπει να προσέξουμε ότι το επιταχυνσιόμετρο εντοπίζει μία αδρανειακή ή πλασματική δύναμη, όπως την αναφέρουν, της οποίας η κατεύθυνση θα είναι αντίθετη

του διανύσματος επιτάχυνσης. Με αυτό τον τρόπο ένα επιταχυνσιόμετρο εντοπίζει τη δύναμη που ασκείται στην αντίθετη κατεύθυνση από την πραγματική. Στο παράδειγμά μας, η κίνηση έγινε προς τα αριστερά, ενώ η επιτάχυνση εντοπίστηκε στα δεξιά. Τέτοιου είδους δυνάμεις συνήθως δημιουργούνται από κατεύθυντικές επιταχύνσεις, όπως θα δούμε και παρακάτω το γεγονός αυτό όμως δεν αποτελεί τον κανόνα. Εάν στο κύβο αυτό εφαρμόσουμε όλες τις δυνάμεις που διέπουν το περιβάλλον γύρω μας, τότε λόγω της έλξης από τη Γη η σφαίρα θα μείνει σταθερά στο κάτω μέρος του κύβου (Z^-) δημιουργώντας δύναμη $1g$, στη κατεύθυνση αυτή, όπως φαίνεται στη παρακάτω εικόνα.

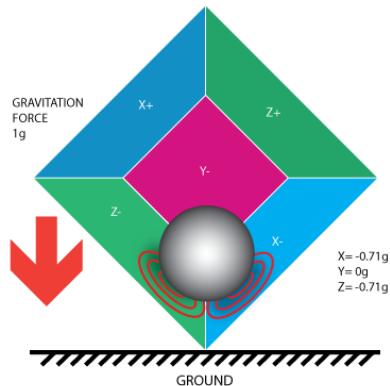


Εικόνα 42. Μη στατική μορφή επιταχυνσιομέτρου Z άξονα .

Στη περίπτωση αυτή ο κύβος παραμένει ακίνητος, παρόλα αυτά λαμβάνουμε τη δύναμη που προ-είπαμε $-1g$ στον Z άξονα. Θεωρητικά, εάν η μπάλα ήταν μεταλλική και εφαρμόζαμε ένα μαγνήτη στο πάνω μέρος του κύβου τότε θα λαμβάναμε, λόγω έλξης, σταθερή τιμή, δύναμης $1g$ στον Z άξονα. Έτσι παρατηρούμε ότι ουσιαστικά το επιταχυνσιόμετρο δεν μετρά την επιτάχυνση αλλά την ελεκτική δύναμη σε κάθε κατεύθυνση, η οποία και αποτελεί τη κύρια μετρούμενη τιμή μέτρησης του επιτανχυσιομέτρου.

Προφανώς όλες οι παραπάνω συνθήκες αναπτύχθηκαν για χάριν παραδείγματος, έτσι ώστε να κατανοήσουμε τη λειτουργία του επιτανχυσιομέτρου. Οι MEMS αισθητήρες όπως ο MPU-6050 είναι κατασκευασμένοι διαφορετικά ακολουθώντας όμως την ίδια λειτουργία. Μέχρι τώρα έχουμε αναπτύξει παραδείγματα σε έναν άξονα, στη πραγματικότητα όμως οι δυνάμεις ασκούνται και στους τρείς άξονες. Επιστρέφοντας πάλι στο παράδειγμα του κύβου περιστρέφοντας τον κατά 45 μοίρες σε σχέση με τη

τελευταία του θέση, η σφαίρα θα ασκήσει δύναμη τόσο στον άξονα (Z -) όσο και στον άξονα (X -) όπως φαίνεται και στη παρακάτω εικόνα.

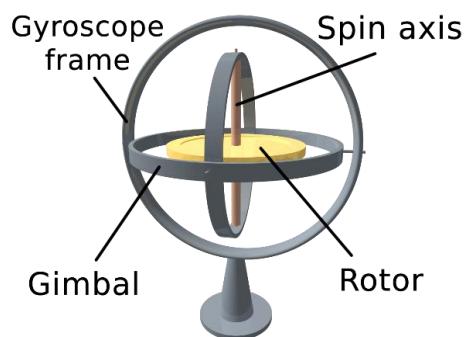


Εικόνα 43. Μη στατική μορφή επιταχυνσιομέτρου X, Y άξονα .

Η τιμή 0.71 που εντοπίζεται στους δύο άξονες δεν είναι τυχαία αλλά αποτελεί τη προσέγγιση της τιμής $SQRT(1/2)$ που χαρακτηρίζει τις επιμέρους δυνάμεις [Ref 17] .

3.8.2.3 ΘΕΩΡΙΑ ΓΥΡΟΣΚΟΠΙΟΥ

Το γυροσκόπιο, όπως και το επιταχυνσιόμετρο είναι μία συσκευή που χρησιμοποιείται για να υπολογίσει την κατεύθυνση ενός αντικειμένου. Η σχεδίαση του περιλαμβάνει ένα πλήρως περιστρεφόμενο ρότορας (*rotor*), τοποθετημένο στο κέντρο ενός δακτυλίου μεγαλύτερης ακτίνας, σταθερής κατάστασης (*Gyroscope frame*) [Ref 18] .



Εικόνα 44. Μορφή γυροσκοπίου αρχικής θέσης .

Τα γυροσκόπια μετρούν τη γωνιακή ταχύτητα, δηλαδή πόσο γρήγορα ο ρότορας περιστρέφεται σε σχέση με έναν άξονα. Όμως για τον εντοπισμό της ακριβής κατεύθυνσης ενός αντικειμένου, χρειαζόμαστε τόσο τις τιμές του γυροσκοπίου όσο και τις τιμές του επιταχυνσιόμετρου. Σε αντίθεση με τα επιταχυνσιόμετρα το γυροσκόπιο δεν επηρεάζεται από τις βαρυτικές δυνάμεις, έτσι ο ένας αισθητήρας συμπληρώνει τέλεια τον άλλον. Συνήθως οι μονάδες μέτρησης που παρατηρούνται στο γυροσκόπιο είναι μονάδες γωνιακής ταχύτητας περιστροφής ανά λεπτό (*RPM*) ή μοιρών ανά λεπτό ($^{\circ}/s$). Οι τρείς άξονες περιστροφής αντιπροσωπεύονται σαν *x*, *y*, και *z*, ή σαν *roll*, *pitch*, και *yaw*. Στο παρελθόν, τα γυροσκόπια είχαν χρησιμοποιηθεί σαν πλοηγοί διαστήματος, μέσω υπολογισμού πορείας εκτόξευσης πυραύλων ή ακόμα και καθοδηγητές υποθαλάσσιων μονάδων. Σήμερα, τα γυροσκόπια βρίσκουν χρήση σε πληθώρα εφαρμογών και ειδικότερα σε συνδυασμό με τα επιταχυνσιόμετρα καταφέρνουν και επιφέρουν πολύ ικανοποιητικά αποτελέσματα. Κάποιες από τις εφαρμογές που εντοπίζονται είναι ανιχνευτές κινήσεων ή πλοϊγηση αυτοκινούμενων οχημάτων [Ref 21].

3.8.2.4 ΣΥΝΔΙΑΣΜΟΣ ΤΙΜΩΝ ΤΟΥ MPU - 6050

Για να υπολογίσει κανείς τον προσανατολισμό ενός μη – κινούμενου αντικειμένου (*pitch* και *roll*), μπορεί απλά να χρησιμοποιηθεί ένα επιταχυνσιόμετρο τριών αξόνων. Για ένα στατικό αντικείμενο αυτό μας παρέχει τις τιμές των βαρυτικών δυνάμεων και στους τρεις άξονες και συνεπώς το προσανατολισμό του. Αφού πάντα οι τιμές που εξάγονται έχουν σαν αναφορά το κέντρο της Γης, τότε μέσω τριγωνομετρικών πράξεων υπολογίζεται ο τρόπος που το επιταχυνσιόμετρο είναι κεκλισμένο. Η μέθοδος αυτή χρησιμοποιείται κατά κόρον σε έξυπνες κινητές συσκευές και δίνει αρκετά ικανοποιητικά και ακριβή αποτελέσματα, με τη προϋπόθεση όμως ότι το αντικείμενο δεν κινείται. Θεωρητικά, εάν περιστρέψουμε μια έξυπνη κινητή συσκευή στο χώρο, εφαρμόζοντας μία δύναμη σε αυτή, μπορούμε μέσω του επιταχυνσιόμετρου να υπολογίσουμε το προσανατολισμό του αντικειμένου. Όμως οι μετρήσεις που θα πάρουμε εφόσον ασκούμε εμείς εξωτερική δύναμη δεν θα είναι ακριβής.

Ένα άλλο πρόβλημα που μας ώθησε στη συνεργασία των δύο αισθητήρων είναι η εναισθησία του αισθητήρα στις δονήσεις. Αυτό δεν αποτελεί ακριβώς πρόβλημα, αλλά απαρτίζει μέρος των αποτελεσμάτων ως συνέπεια της κίνησης του αντικειμένου. Στην εφαρμογή που αναπτύσσουμε, εάν επιθυμούμε να σταθεροποιήσουμε το *Quadcopter*,

πρέπει οπωσδήποτε να εξομαλύνουμε τη μετρούμενη γωνία, αφού το γεγονός αυτό αποτελεί το πιο σημαντικό κομμάτι. Συνήθως, αναπτύσσοντας τον αλγόριθμο σταθεροποίησης *PID*, εάν δεν επιφέρονται τα επιθυμητά αποτελέσματα στη διαδικασία του fine tuning, τότε πολύ πιθανό να οφείλεται στη μη εξομάλυνση της γωνίας. Έτσι οι τιμές που επεξεργαζόμαστε δεν θα είναι αρκετά ακριβής.

Όπως προ-είπαμε χρησιμοποιώντας μόνο το επιταχυνσιόμετρο δεν αποτελεί μια ασφαλής και έγκυρη επιλογή. Ενδεχόμενα, υπάρχει η λύση του φιλτραρίσματος του σήματος, για να μειωθεί η εξάρτηση του από τις δονήσεις που επιφέρουν τα μοτέρ, αλλά δεν θα είμαστε σε θέση να επιφέρουμε τα επιθυμητά αποτελέσματα, ιδιαίτερα σε τέτοιου είδους εφαρμογές. Έτσι η λύση όλων αυτών των «προβλημάτων» που περιγράψαμε είναι ο συνδυασμός των δύο αισθητήρων μέσω τριγωνομετρικών πράξεων [Ref 22].

Παρακάτω θα παρουσιάσουμε τον κώδικα που περιγράφει τη λειτουργία των επιμέρους αισθητήρων του *IMU-6050* (επιταχυνσιόμετρο - γυροσκόπιο) αλλά και τη λογική που ακολουθήσαμε για να εξάγουμε τα αποτελέσματα, συνδυάζοντας τα σήματα αυτά. Επιπλέον απαραίτητη διαδικασία αποτελεί το *Calibration* των αισθητήρων, έτσι ώστε να αρχικοποιηθεί η κατεύθυνση τους πριν τη χρήση τους. Υστερα, θα περάσουμε στη διαδικασία του *testing* εξάγοντας τις τελικές μετρούμενες τιμές μέσω της σειριακής θύρας του *Arduino*, τυπώνοντας τες και επισυνάπτοντας στο έγγραφο αυτό. Ως επιπλέον βήμα για την οπτικοποίηση των αποτελεσμάτων αναπτύχθηκε ένας αλγόριθμος στο πρόγραμμα *matlab* που σε συνεργασία με το *Arduino*, καταφέρνει σε *Real – Time* να εξάγει τις γραφικές παραστάσεις των τιμών για κάθε άξονα ξεχωριστά.

3.8.2.5 ΥΠΟΛΟΓΙΣΜΟΣ ΓΩΝΙΑΣ ΜΕ ΧΡΗΣΗ ΤΟΥ ARDUINO UNO

Αρχικά η συνδεσμολογία που πρέπει να πραγματοποιήσουμε για την σωστή εγκαθίδρυση των αισθητήρων στο *Arduino* ακολουθεί την εξής λογική. Θα πρέπει να συνδέσουμε την αναλογική είσοδο του *Arduino* (*pin Analog input 4*) που χρησιμοποιεί το *I2C Bus* στο *SDA* της συσκευής. Ανάλογα συνδέσουμε την αναλογική είσοδο του *Arduino* (*pin Analog input 5*) στο *SCL* της συσκευής. Τέλος συνδέουμε ως τάση εισόδου στον αισθητήρα την τάση εξόδου 3.3 volt του *Arduino* και την γείωση του μικροελεγκτή στο *GND* του *10DOF*. Πρέπει να σημειωθεί ότι είναι προτιμότερο να

τροφοδοτούμε τον αισθητήρα με χαμηλή τάση (3.3v) παρά με 5v, αλλά για να πραγματοποιηθεί αυτό θα πρέπει να απενεργοποιήσουμε τις ενσωματωμένες *pull – up* αντιστάσεις του *Arduino*. Αυτό γίνεται αρχικά πηγαίνοντας στις βιβλιοθήκες του *Arduino* στο φάκελο *Arduino\libraries\Wire\utility* και εντοπίζοντας το αρχείο *twi.c* ανοίγοντάς το με κάποιον *editor*. Και στη συνέχεια εντοπίζουμε τις γραμμές *digitalWrite(SDA, 1);* και *digitalWrite(SCL, 1);* και τις σχολιάζουμε. Με αυτό τον τρόπο έχουμε απενεργοποιήσει σε κάθε sketch που αναπτύσσουμε τις *pull – up* αντιστάσεις στα pin A4 και A5 του μικροελεγκτή.

Για την σωστή λειτουργία του προγράμματος πρέπει να συμπεριλάβουμε στο *project* το αρχείο *<Wire.h>* το οποίο και κάνει τη διαχείριση της συσκευής που έχει τοποθετηθεί στο διάδρομο *I2C* του *Arduino*. Επίσης το *MPU-6050* θα πρέπει να έχει ελεγχθεί ότι εντοπίζεται στη διεύθυνση *0x69* όπως ορίζεται στον εντοπισμό των συσκευών μέσω του *Scanner_port_report* που δείξαμε παραπάνω. Τα αρχεία που έχουν δημιουργηθεί για τον έλεγχο του επιτανχυσιομέτρου και του γυροσκοπίου είναι τα *gyro_accel.cpp* και *gyro_accel.h*. Σε αυτό το σημείο θα περιγράψουμε κομμάτια του κώδικα που αφορούν την εκτέλεση του *MPU-6050* και μόνο. Το *project* που αναλαμβάνει την εκτέλεση και εμφάνιση των αποτελεσμάτων είναι το *MPU-6050_Control.ino*.

Θα ξεκινήσουμε τη περιγραφή από το αρχείο *gyro_accel.h*. Ανοίγοντας το αρχείο αυτό παρατηρούμε ότι έχουμε δημιουργήσει μία κλάση που την έχουμε ονομάσει *MPU6050_CLASS* η οποία εμπεριέχει τόσο τις μεταβλητές (*Public* και *Private*) όσο και τις συναρτήσεις που θα χρειαστούμε. Αρχικά μέσα στη κλάση ορίζουμε μια *class constructor* συνάρτηση *MPU6050_CLASS(void)*; με *void* όρισμα η οποία πρέπει να έχει το ίδιο όνομα με το όνομα της κλάσης.

Η συνάρτηση αυτή είναι υπεύθυνη για την αρχικοποίηση της κλάσης ενώ κατά την κλήση της αναλαμβάνει την αρχικοποίηση των μεταβλητών στη τιμή μηδέν. Στη συνέχεια παρατηρούμε τις *public* μεταβλητές τύπου *float accel_x_scaled,* *accel_y_scaled,* *accel_z_scaled,* *gyro_x_scaled,* *gyro_y_scaled,* *gyro_z_scaled*. Καθώς γνωρίσουμε οι τιμές που διαβάζονται από τους αισθητήρες δεν είναι στη μορφή που θα θέλαμε, έτσι μέσω ενός παράγοντα μπορούμε και κάνουμε *mapping* τις τιμές αυτές στις επιθυμητές μοίρες που αντιπροσωπεύουν οι αισθητήρες.

Σε επόμενη φάση συναντάμε τις μεταβλητές float angle_x_gyro, angle_y_gyro, angle_z_gyro, angle_x_accel, angle_y_accel, angle_z_accel, angle_x, angle_y, angle_z οι οποίες αντιπροσωπεύουν την πραγματική γωνία που προφανώς υπολογίζεται από τις πραγματικές μοίρες που εξηγήσαμε παραπάνω.

Ως public συναρτήσεις παρατηρούμε τις void MPU6050_ReadData() η οποία σαν λειτουργία αναλαμβάνει την ανάγνωση των τιμών του επιτανχυσιομέτρου και του γυροσκοπίου αφού βέβαια τις κανονικοποιήσει. Ο παράγοντας κανονικοποίησης εντοπίζεται στις μεταβλητές float accel_scale_fact, gyro_scale_fact οι οποίες είναι private. Εάν επιθυμούμε να λάβουμε τα δεδομένα χωρίς κανονικοποίηση τότε θα πρέπει αυτές τις τιμές να τις θέσουμε στο μηδέν. Ύστερα από την κλήση αυτής της συνάρτησης οι τιμές (temp_scaled, accel_x_scaled, accel_y_scaled, accel_z_scaled, gyro_x_scaled, gyro_y_scaled και gyro_z_scaled) ανανεώνονται με τις καινούριες τιμές ανάγνωσης.

Η συνάρτηση MPU6050_ResetWake() είναι αυτή που πραγματοποιεί επανεκκίνηση του chip στις default τιμές. Πριν από την πρώτη χρήση του chip είναι αναγκαστικό να πραγματοποιείται η κλήση της συνάρτησης για την απαραίτητη αρχικοποίηση. Η συνάρτηση MPU6050_SetDLPF(int BW); ρυθμίζει το ενσωματωμένο Low Pass φίλτρο που συνοδεύει το MPU-6050. Η τιμή int BW θα πρέπει να εμπεριέχει τιμές από 0 έως 6. Με αυτή τη ρύθμιση το BW του φίλτρου μπορεί να αλλάξει σύμφωνα με το πίνακα που επισυνάπτουμε παρακάτω.

ΠΙΝΑΚΑΣ 7. Χαρακτηριστικά του ενσωματωμένου βαθυπερατού φίλτρου του MPU - 6050

| int BW | DLPF Bandwidth |
|-----------------|----------------|
| 0 or Any | Inf |
| 1 | 184 |
| 2 | 94 |
| 3 | 44 |
| 4 | 21 |
| 5 | 10 |
| 6 | 5 |
| | |

Ένα τοποθετήσουμε τιμή ως όρισμα της συνάρτησης που δεν είναι μεταξύ (0-6) τότε το ενσωματωμένο φίλτρο απενεργοποιείται ορίζοντας το BW ως άπειρο. Η συνάρτηση

`MPU6050_SetGains(int gyro, int accel)` θέτει τη μέγιστη κλίμακα της μέτρησης και ακολουθεί τη λογική σύμφωνα με το παρακάτω πίνακα :

ΠΙΝΑΚΑΣ 8. Κλίμακα μέγιστης Τιμής μέτρησης του MPU – 6050 .

| int gyro | Max scale [degree/s] | int accel | Max scale [m/s2] |
|----------|-------------------------|-----------|---------------------|
| 0 | 250 | 0 | 2g |
| 1 | 500 | 1 | 4g |
| 2 | 1000 | 2 | 8g |
| 3 | 2000 | 3 | 16g |
| any | Raw value | any | Raw value |

Αργότερα θα δούμε με ποιο όρισμα καλούμε τη συνάρτηση αυτή και κατά πόσο μας επηρεάζει τις τιμές μέτρησης. Η συνάρτηση `MPU6050_OffsetCal()` θα υπολογίσει το offset, δηλαδή τη διαφορά της αρχικής μέτρησης από το μηδέν για να υπολογίσει το ποσοστό λάθους σε κάθε μέτρηση και να την αφαιρέσει από αυτή. Το *offset* υπολογίζεται και στις τιμές του επιτανχυσιομέτρου και στις τιμές του γυροσκοπίου ενώ αποθηκεύονται στις μεταβλητές `accel_x_OC, accel_y_OC, accel_z_OC, gyro_x_OC, gyro_y_OC` και `gyro_z_OC`. Για να υπολογιστούν οι τιμές έχουμε πραγματοποιήσει δύο επιλογές στο χρήστη. Είτε μπορεί να υπολογιστεί το *offset* κατά την αρχικοποίηση του *Quadcopter*, είτε μπορεί να έχει προ - υπολογιστεί από εμάς σε προηγούμενο στάδιο και να χρησιμοποιήσουμε τις *precalibrated* τιμές. Αργότερα θα εξηγήσουμε τη διαδικασία του *calibration* που εξάγουν *pre-calibrated* τιμές.

Στο κύριο κώδικα υπάρχουν κάποια *definitions* που πρέπει να σχολιάσουμε ή να από - σχολιάσουμε για να διαλέξουμε το *mode* του *calibration*. Για να πραγματοποιήσουμε *calibration* κατά την αρχικοποίηση πρέπει να είναι σχολιασμένο το σημείο που αναγράφει `#define QUAD_CALIBRATION` και από - σχολιασμένο το σημείο `#define PRECALIBRATION`. Ανάλογα, για την αντίθετη διαδικασία σχολιάζουμε και αποσχολιάζουμε την αντίθετη επιλογή.

Για να γίνει το *calibration* όμως κατά την αρχικοποίηση του *Quadcopter* πρέπει να έχουμε τοποθετήσει τον αισθητήρα σε πλήρως επίπεδο έδαφος έτσι ώστε ο x και ο y άξονας να είναι οριζόντια στο επίπεδο της γης και ο z άξονας προφανώς κάθετα. Αυτό

δεν είναι τόσο εύκολο κάθε φορά αφού οι συνιστώσες που μπορεί να στρεβλώσουν την μέτρηση είναι πάρα πολλές (το επίπεδο του αισθητήρα ως προς το *Quadcopter* και το επίπεδο του *Quadcopter* ως προς το έδαφος). Παρόλα αυτά, το να πραγματοποιήσεις calibration στο επιταχυνσιόμετρο και το γυροσκόπιο είναι μια απαραίτητη διαδικασία, αφού σε διαφορετική περίπτωση δεν θα μπορούμε να υπολογίσουμε το μηδέν για να πετά οριζόντια το *Quadcopter* ως προς το έδαφος.

Για το γυροσκόπιο οι κλιμακωμένες τιμές δίνονται μέσω της εξίσωσης “ $\text{gyro_x_scaled} = \frac{d}{dt}\theta_x$ ”, ως εκ τούτου, απαιτούν την ολοκλήρωση τους για να ληφθεί η γωνία γύρω από τον αντίστοιχο άξονα. Εάν παράδειγμα η τιμή “ gyro_x_scaled ” περιλαμβάνει τυχόν σφάλμα στο offset, τότε αυτό το σφάλμα θα ολοκληρωθεί (*drift away*) και θα παράγει μία πολύ μεγάλη, μη ρεαλιστική τιμή. Οπότε σε ιδανικές συνθήκες οι μετρούμενες τιμές θα πρέπει να είναι μηδέν, κάτι που δεν συμβαίνει σχεδόν ποτέ, είτε λόγο θορύβου από τον αισθητήρα, είτε λόγω δονήσεων που παράγουν τα μοτέρ.

Παρόλα αυτά η τιμή αυτή θα είναι πολύ κοντά στο μηδέν ενώ εξαρτάται από εμάς πόσο θα μας επηρεάσει, τόσο από το ποσοστό εμπιστευτικότητας που θα δείξουμε σε κάθε αισθητήρα όταν συνδυάσουμε τις τιμές τους, όσο και κατά την κατασκευή του *PID* που ελέγχουν το ποσοστό ισχύος στα μοτέρ.

Για τον υπολογισμό της γωνίας οι έξοδοι για τις scaled τιμές είναι οι $\text{gyro_x_scaled} = \frac{d}{dt}\theta_x^{\text{gyro}}$, $\text{gyro_y_scaled} = \frac{d}{dt}\theta_y^{\text{gyro}}$ και $\text{gyro_z_scaled} = \frac{d}{dt}\theta_z^{\text{gyro}}$. Παρακάτω θα εξηγήσουμε την περίπτωση μόνο του x άξονα για την περαιτέρω κατανόηση του αλγορίθμου. Εφόσον το γυροσκόπιο μετράει μόνο τις αλλαγές στη κλίση και όχι την πραγματική γωνία, τότε για να χρησιμοποιήσουμε τις παραπάνω τιμές θα πρέπει να ολοκληρώσουμε τις μετρήσεις. Αφού το *Arduino* αποτελεί ένα διακριτό σύστημα σύμφωνα και με τη θεωρία της ψηφιακής επεξεργασίας σήματος θα πρέπει να δειγματοληπτίσουμε τις τιμές, με σταθερό χρόνο.

Η διαδικασία αυτή γίνεται μέσα στην συνάρτηση void loop() που ο διακριτός χρόνος σηματοδοτείται ως T και ορίζεται ως $T = t_n - t_{n-1}$, όπου $n = \{1,2,3,\dots\}$ είναι οι αριθμοί δειγματοληψίας. Επιπροσθέτως υποθέτουμε ότι σε κάθε κύκλο η τιμή “ gyro_x_scaled ” παραμένει σταθερή. Κάτι το οποίο θα μας παράξει ένα μικρό σφάλμα το οποίο δεν είναι μείζον σημασίας. Για να εφαρμόσουμε διακριτό ολοκλήρωμα (άθροισμα) χρησιμοποιήσαμε την μέθοδο *forward - Euler's integration*

διότι αποτελεί την πιο συνηθισμένη μέθοδο με τις πιο απλές μαθηματικές πράξεις για το *Arduino*.

$$\theta_x^{gyro}(t_n) = \text{gyro_x_scaled} * T + \theta_x^{gyro}(t_{n-1})$$

Ας υποθέσουμε ότι κατά την αρχικοποίηση η διαδικασία του *calibration* έχει πραγματοποιηθεί αλάθευτα και οι γωνίες που μετράμε στη πρώτη μέτρηση για $n=0$, θα είναι $\theta_x^{gyro}(t_0) = 0^\circ$, $\theta_y^{gyro}(t_0) = 0^\circ$, $\theta_z^{gyro}(t_0) = 90^\circ$. Η τιμή T (η διαφορά χρόνου μεταξύ των δειγμάτων) και η δυναμική των αισθητήρων επηρεάζουν πολύ τα αποτελέσματα των γωνιών. Όσο πιο αργά αλλάζει η μετρούμενη γωνία και όσο πιο γρήγορα επαναλαμβάνεται η μέτρηση, τόσο πιο ακριβή θα είναι τα αποτελέσματα.

Ωστόσο, το *Arduino* είναι αρκετά αργό (χρησιμοποιώντας έναν 16 *Mhz* κρύσταλλο) και κρατώντας σταθερό το T σε τάξεις των 10-20 ms αποτελεί μία δύσκολη διαδικασία, ειδικά αφού το *Arduino* πρέπει να πραγματοποιήσει και πολλές ακόμα πράξεις στο ενδιάμεσο μεταξύ ανάγνωσης των δειγμάτων. Σε αυτή τη μορφή του κώδικα θα χρησιμοποιήσουμε την τιμή των 20 ms για κάθε κύκλο ανάγνωσης, εφόσον δεν έχουμε ακόμα να πραγματοποιήσουμε περαιτέρω πράξεις μεγαλύτερης πολυπλοκότητας.

Όπως προαναφέραμε το πρόβλημα του *drift* στο γυροσκόπιο αποτελεί από μόνο του ένα σφάλμα που δεν πρέπει να συνυπολογίσουμε στον αλγόριθμο. Εφόσον η τιμή “gyro_x_scaled” δεν είναι ποτέ μηδέν, για να διορθώσουμε το πρόβλημα η γωνία συνυπολογίζεται και από το επιταχυνσιόμετρο και συνδυάζεται κατάλληλα. Υποθέτοντας ότι η γωνία z είναι κάθετη στο έδαφος, και μετρά 1g (9.81), μπορούμε να χρησιμοποιήσουμε το διάνυσμα επιτάχυνσης και την προβολή αυτού στο x και στο y άξονα για να υπολογίσουμε τη γωνία x και y. Αυτό γίνεται για την γωνία x και μέσω του τύπου:

$$\theta_x^{accel} = \tan^{-1} \frac{\text{accel } x \text{ scaled}}{\sqrt{\text{accel } y \text{ scaled}^2 + \text{accel } z \text{ scaled}^2}}$$

και για την γωνία y μέσω του τύπου :

$$\theta_y^{accel} = \tan^{-1} \frac{\text{accel } y \text{ scaled}}{\sqrt{\text{accel } x \text{ scaled}^2 + \text{accel } z \text{ scaled}^2}}$$

Το πρόβλημα των επιτανχυσιομέτρων είναι ότι είναι αρκετά θορυβώδες και ευαίσθητα σε δονήσεις. Όπως εξηγήσαμε και παραπάνω θα πρέπει να θέσουμε ένα βαθμό εμπιστευτικότητας σε κάθε αισθητήρα. Αυτό πραγματοποιείται μέσω ενός φίλτρου για τον συνδυασμό των τιμών, με χρήση του τύπου :

$$\theta_x = \text{Filter_gain}^{\theta_x^{gyro}} + (1-\text{Filter_gain})^{\theta_x^{accel}}$$

Χρησιμοποιώντας την φίλτραρισμένη γωνία “ θ_x ” στην ολοκλήρωση των τιμών του γυροσκοπίου αντί να χρησιμοποιήσουμε απευθείας τη τιμή “ θ_x^{gyro} ”, για να αποφύγουμε το πρόβλημα του *drift*. Άρα ο υπολογισμός της γωνίας του γυροσκοπίου πλέον πραγματοποιείται μέσω του τύπου :

$$\theta_x^{gyro}(t_n) = \text{gyro_x_scaled} * T + \theta_x(t_{n-1})$$

Μία καλή προσέγγιση της τιμής του φίλτρου, ύστερα από πειραματικές ενδείξεις είναι η τιμή Filter_gain να είναι ίση με 0.95 , δίνοντας το μεγαλύτερο βάρος στο γυροσκόπιο. Παρακάτω επισυνάπτουμε τους κώδικες που εμπεριέχονται στα αρχεία gyro_accel.h και gyro_accel.cpp [Ref 23] .

Listing 5. Gyro_accel.h Report c++ language technical details

```
/*
-----
----- MSc HEP 2013-2014 -----
-----
*/
/*
-----
----- Project      : Quadcopter
File          : MPU_6050_Control.ino
Description   : This code calibrates, test and debug MPU-6050 sensor
Author        : Monahopoulos Konstantinos
-----
*/
#include <Arduino.h>
#ifndef GYRO_ACCEL_H
#define GYRO_ACCEL_H

/*
-----
----- GYRO AND ACCELEROMETER CLASS DECLARATION
-----
*/

```

```

class MPU6050_CLASS
{
public:
    MPU6050_CLASS(void); //class constructor

    float
    accel_x_scaled,accel_y_scaled,accel_z_scaled,gyro_x_scaled,gyro_y_scaled,
    gyro_z_scaled; //Scalled Data variables
    float
    angle_x_gyro,angle_y_gyro,angle_z_gyro,angle_x_accel,angle_y_accel,angle_z_accel,
    angle_x,angle_y,angle_z;// angle variables
    void MPU6050_ReadData();
    void MPU6050_ResetWake();
    void MPU6050_SetDLPF(int BW);
    void MPU6050_SetGains(int gyro,int accel);
    void MPU6050_OffsetCal();
    void MPU6050_Set_Precalibration_offset(void);

private:
    float accel_scale_fact, gyro_scale_fact; // Scale factor variables
    int accel_x, accel_y, accel_z, gyro_x, gyro_y, gyro_z; // Raw values
    variables
    int accel_x_OC, accel_y_OC, accel_z_OC, gyro_x_OC ,gyro_y_OC, gyro_z_OC;
    // offset variables
    float temp_scaled;
    int read_bytes;
    byte gyro_byte,accel_byte;
    int x,y,z,i,temp;
};

#endif

```

Listing 6. Gyro_accel.cpp Report c++ language technical details

```

/*
-----
----- MSc HEP 2013-2014 -----
-----

*/
/*
-----
----- Project      : Quadcopter
File          : MPU_6050_Control.ino
Description   : This code calibrates, test and debug MPU-6050 sensor
Author        : Monahopoulos Konstantinos
-----
----- */

#include <Arduino.h>
#include <Wire.h>
#include "gyro_accel.h"

/*
-----
-----
```

```

DEFINITIONS
-----
*/
#define MPU6050_address 0x69

// ----- SELF Test Trim Factors -----
#define MPU6050_self_test_x 13      // R/W
#define MPU6050_self_test_y 14      // R/W
#define MPU6050_self_test_z 15      // R/W
#define MPU6050_self_test_A 16      // R/W

// ----- Sample Divider -----
#define MPU6050_sample_div 25      // R/W
/*      Sample Divider Description
    Sample rate = 8/(1 + Sample rate divider) [kHz] if DLPF is disabled
    Sample rate = 1/(1 + Sample rate divider) [kHz] if DLPF is enabled
*/
// ----- Configuration -----
#define MPU6050_config 26          // R/W
#define MPU6050_gyro_config 27      // R/W
#define MPU6050_accel_config 28      // R/W

// ----- Data -----
#define MPU6050_data_start 59

// ----- Power Management -----
#define MPU6050_PWR1 107
#define MPU6050_PWR2 108

// ----- Defining Constant -----
#define g 9.81                      // Gravitational acceleration

/*
-----
Function Prototyping
-----
*/
// ----- Constructor function
// -----

MPU6050_CLASS::MPU6050_CLASS(void){

accel_x_scaled,accel_y_scaled,accel_z_scaled,gyro_x_scaled,gyro_y_scaled,
gyro_z_scaled=0;
angle_x_gyro,angle_y_gyro,angle_z_gyro,angle_x_accel,angle_y_accel,angle_z_accel,
angle_x,angle_y,angle_z=0;
accel_x_OC, accel_y_OC, accel_z_OC, gyro_x_OC ,gyro_y_OC ,
gyro_z_OC,temp_scaled,read_bytes=0;
gyro_byte,accel_byte=0;
x,y,z,i,temp=0;
}

// -----
//      Read Data From MPU6050
// -----

void MPU6050_CLASS::MPU6050_ReadData(void){

Wire.beginTransmission(MPU6050_address);
Wire.write(MPU6050_data_start);
Wire.endTransmission();
}

```

```

read_bytes = 14;

Wire.requestFrom(MPU6050_address, read_bytes);

if(Wire.available() == read_bytes){

    accel_x = Wire.read()<<8 | Wire.read();
    accel_y = Wire.read()<<8 | Wire.read();
    accel_z = Wire.read()<<8 | Wire.read();

    temp = Wire.read()<<8 | Wire.read();

    gyro_x = Wire.read()<<8 | Wire.read();
    gyro_y = Wire.read()<<8 | Wire.read();
    gyro_z = Wire.read()<<8 | Wire.read();
}

accel_x_scaled = (float)(accel_x-accel_x_OC)*accel_scale_fact/1000; // divided by 1000 as the Scale factor is in milli units
accel_y_scaled = (float)(accel_y-accel_y_OC)*accel_scale_fact/1000;
accel_z_scaled = (float)(accel_z-accel_z_OC)*accel_scale_fact/1000;

gyro_x_scaled = (float)(gyro_x-gyro_x_OC)*gyro_scale_fact/1000;
gyro_y_scaled = (float)(gyro_y-gyro_y_OC)*gyro_scale_fact/1000;
gyro_z_scaled = ((float)(gyro_z-gyro_z_OC)*gyro_scale_fact/1000);

temp_scaled = (float)temp/340+36.53;
}

// -----
//      Reset and wake up the MPU6050
// -----


void MPU6050_CLASS::MPU6050_ResetWake(){

    Serial.println("Resetting MPU6050 and waking it up.....");
    Serial.println(" ");
    Wire.beginTransmission(MPU6050_address);
    Wire.write(MPU6050_PWR1);
    Wire.write(0b10000000);
    Wire.endTransmission();

    delay(100); // Waiting for the reset to complete

    Wire.beginTransmission(MPU6050_address);
    Wire.write(MPU6050_PWR1);

    Wire.write(0b00000000);
    Wire.endTransmission();
}

// -----
//      Setting up the DLFP (LOW PASS FILTER)
// -----


void MPU6050_CLASS::MPU6050_SetDLPF(int BW){

    if (BW < 0 || BW > 6){
        BW = 0;
    }
    Wire.beginTransmission(MPU6050_address);
    Wire.write(MPU6050_config); // Address to the configuration register
/*          config Description ---- x x 0 0 0 F2 F1 F0
I am only interested in the Digital Low Pass Filter (DLPF)
F2 F1 F0      Bandwidth [Hz]
0 0 0
0 0 1          184

```

```

0 1 0      94
0 1 1      44
1 0 0      21
1 0 1      10
1 1 0      5
*/
Wire.write(BW);
Wire.endTransmission();

}

// -----
//      Setting up the Accelerometer and Gyro Gains
// -----
void MPU6050_CLASS::MPU6050_SetGains(int gyro,int accel){

// Setting up Gyro
Wire.beginTransmission(MPU6050_address);
Wire.write(MPU6050_gyro_config); // Address to the configuration
register
if (gyro==0)
{
    gyro_scale_fact =(float)250*0.0305; // each data is of 16 bits that
means, 250 is divided along 2^(15)-1 = 32767 so for milli degree/s 0.0305 =
1000/32767
    gyro_byte = 0b00000000;
}else if (gyro == 1)
{
    gyro_scale_fact = 500*0.0305; // each data is of 16 bits that means,
500 is divided along 2^(15)-1 = 32767 so for milli degree/s 0.0305 =
1000/32767
    gyro_byte = 0b00001000;
}else if (gyro == 2)
{
    gyro_scale_fact = 1000*0.0305;// each data is of 16 bits that means,
1000 is divided along 2^(15)-1 = 32767 so for milli degree/s 0.0305 =
1000/32767
    gyro_byte = 0b00010000;
}else if (gyro == 3)
{
    gyro_scale_fact = 2000*0.0305; // each data is of 16 bits that means,
2000 is divided along 2^(15)-1 = 32767 so for milli degree/s 0.0305 =
1000/32767
    gyro_byte = 0b00011000;
}else
{
    gyro_scale_fact = 1;
}

Wire.write(gyro_byte);
Wire.endTransmission();

Serial.print("The gyro scale is set to ");
Serial.print(gyro_scale_fact);
Serial.println(" milli Degree/s");

// Setting up Accel
Wire.beginTransmission(MPU6050_address);
Wire.write(MPU6050_accel_config); // Address to the configuration
register
if (accel==0)
{
    accel_scale_fact =(float)2*g*0.0305; // each data is of 16 bits that
means, 2g is divided along 2^(15)-1 = 32767 so for milli m/s^2 0.0305 =
1000/32767
}

```

```

        accel_byte = 0b00000000;
    }else if (accel == 1)
    {
        accel_scale_fact = 4*g*0.0305; // each data is of 16 bits that means,
4g is divided along 2^(15)-1 = 32767 so for milli m/s^2 0.0305 = 1000/32767
        accel_byte = 0b00001000;
    }else if (accel == 2)
    {
        accel_scale_fact = 8*g*0.0305;// each data is of 16 bits that means,
8g is divided along 2^(15)-1 = 32767 so for milli m/s^2 0.0305 = 1000/32767
        accel_byte = 0b00010000;
    }else if (accel == 3)
    {
        accel_scale_fact = 16*g*0.0305; // each data is of 16 bits that means,
16g is divided along 2^(15)-1 = 32767 so for milli m/s^2 0.0305 = 1000/32767
        accel_byte = 0b00011000;
    }else
    {
        accel_scale_fact = 1;
    }

    Wire.write(accel_byte);
    Wire.endTransmission();

    Serial.print("The accel scale is set to ");
    Serial.print(accel_scale_fact);
    Serial.println(" milli m/s^2");
    Serial.println(" ");

}

// -----
//      offset calibration
// -----


void MPU6050_CLASS::MPU6050_OffsetCal(){

    Serial.println("Calibrating gyroscope .... dont move the hardware
.....");

    x=0,y=0,z=0,i;

    MPU6050_ReadData(); //First time Reads Zeros
    MPU6050_ReadData(); //Second time Reads Trash

    // Gyro Offset Calculation
    x=gyro_x;
    y=gyro_y;
    z=gyro_z;

    // Calculate mean value from 1000 gyro values
    for (i=1;i<=1000;i++){
        MPU6050_ReadData();
        x=(x+gyro_x)/2;
        y=(y+gyro_y)/2;
        z=(z+gyro_z)/2;
    }
    Serial.println(" ");
    gyro_x_OC=x;
    gyro_y_OC=y;
    gyro_z_OC=z;

    Serial.print("gyro_x register offset = ");
    Serial.println(x);

    Serial.print("gyro_y register offset = ");
    Serial.println(y);
}

```

```

Serial.print("gyro_z register offset = ");
Serial.println(z);

// Accel Offset Calculation
Serial.println("Calibrating accelerometer .... dont move the hardware
.....");
x=accel_x;
y=accel_y;
z=accel_z;

// Calculate mean accel from 1000 accel values
for (i=1;i<=1000;i++){
    MPU6050_ReadData();
    x=(x+accel_x)/2;
    y=(y+accel_y)/2;
    z=(z+accel_z)/2;
}
Serial.println(" ");
accel_x_OC=x;
accel_y_OC=y;
accel_z_OC=z-(float)g*1000/accel_scale_fact;

Serial.print("Accel_x register offset = ");
Serial.println(x);

Serial.print("Accel_y register offect = ");
Serial.println(y);

Serial.print("Accel_z register offset = ");
Serial.println(z);

Serial.println(" ");
Serial.println(" ");

}

// -----
//      offset Pre-Calibrated Values
// -----


void MPU6050_CLASS::MPU6050_Set_PreCalibration_offset(void){
    gyro_x_OC=146;
    gyro_y_OC=63;
    gyro_z_OC=-1;

    accel_x_OC=138;
    accel_y_OC=50;
    accel_z_OC=8648-(float)g*1000/accel_scale_fact;
}

```

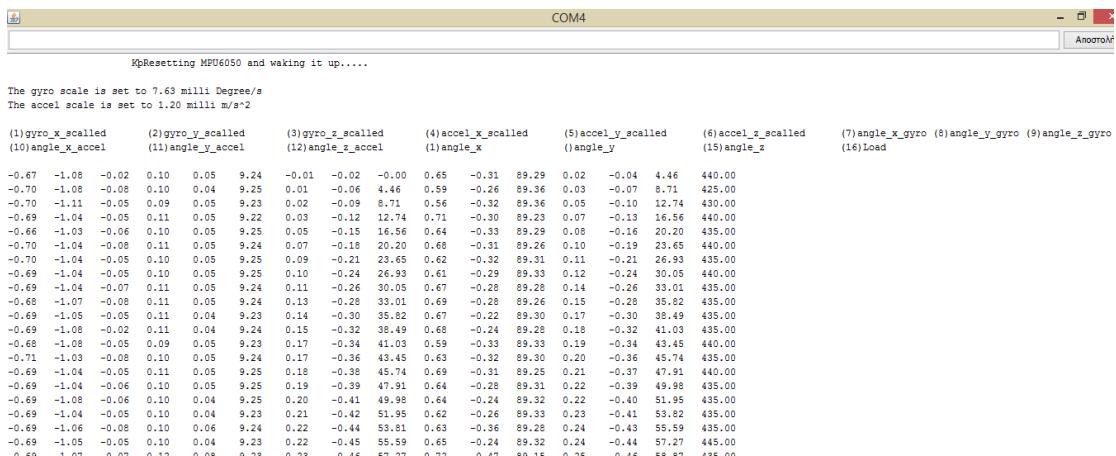
3.8.2.6 DEBUGGING

Σε δεύτερη φάση θα πρέπει να εξηγήσουμε τη διαδικασία του *debugging*. Υπάρχουν δύο τρόποι να απεικονίσουμε τις τιμές των δεδομένων που εξαγοννται από το *IMU*. Ο

πρώτος τρόπος είναι μέσω της σειριακής θύρας του *Arduino* στη οποία πραγματοποιούμε πλήρως απεικόνιση όλων των σταδίων των δεδομένων μέχρι τον υπολογισμό της τελικής γωνίας σε κάθε άξονα. Ενώ ο δεύτερο πραγματοποιεί γραφική απεικόνιση των αποτελεσμάτων στο πρόγραμμα *Matlab* αναλόγως τα δεδομένα που θα εισάγουμε.

Στο κύριο κώδικα υπάρχουν κάποια *definitions* που πρέπει να σχολιάσουμε ή να από – σχολιάσουμε για να διαλέξουμε το mode του *debug*. Για να απεικονίσουμε τα δεδομένα μέσω του *Arduino* πρέπει να είναι σχολιασμένο το σημείο που αναγράφει `#define MATLAB_DATA_LOGGING` και από – σχολιασμένο το σημείο `#define DEBUG`. Ανάλογα για την αντίθετη διαδικασία σχολιάζουμε και αποσχολιάζουμε την αντίθετη επιλογή.

Τα αρχεία *debug.h* και *debug.cpp* εμπεριέχονται τα κατάλληλα δεδομένα για την απεικόνιση των αποτελεσμάτων στο *Matlab*. Μέσω του *Arduino* στη `void setup()` συνάρτηση ορίζουμε μία σειρά από δεδομένα απεικόνισης εξηγόντας στο χρήστη την έννοια της κάθε τιμής, Ενώ ύστερα μέσα στην `void loop()` εκτυπώνουμε τα αποτελέσματα. Παρακάτω δείχνουμε ένα δείγμα μετρήσεων μέσω της σειριακής θύρας, χρησιμοποιώντας το *IDE* του *Arduino*.



```

COM4
ApodoM

KpResetting MPU6050 and waking it up.....
The gyro scale is set to 7.63 milli Degree/s
The accel scale is set to 1.20 milli m/s^2

(1)gyro_x_scaled   (2)gyro_y_scaled   (3)gyro_z_scaled   (4)accel_x_scaled   (5)accel_y_scaled   (6)accel_z_scaled   (7)angle_x_gyro (8)angle_y_gyro (9)angle_z_gyro
(10)angle_x_accel  (11)angle_y_accel  (12)angle_z_accel  (13)angle_x           (14)angle_y           (15)angle_z           (16)Load

-0.67 -1.08 -0.02 0.10 0.05 9.24 -0.01 -0.02 -0.00 0.65 -0.31 89.29 0.02 -0.04 4.46 440.00
-0.70 -1.08 -0.08 0.10 0.04 9.25 0.01 -0.06 4.46 0.59 -0.26 89.36 0.03 -0.07 8.71 425.00
-0.70 -1.11 -0.05 0.09 0.05 9.23 0.02 -0.09 8.71 0.56 -0.32 89.36 0.05 -0.10 12.74 430.00
-0.69 -1.04 -0.05 0.11 0.05 9.22 0.03 -0.12 12.74 0.71 -0.30 89.23 0.07 -0.13 16.56 440.00
-0.66 -1.03 -0.06 0.10 0.05 9.25 0.05 -0.15 16.56 0.64 -0.33 89.29 0.08 -0.16 20.20 435.00
-0.70 -1.04 -0.08 0.11 0.05 9.24 0.07 -0.18 20.20 0.68 -0.31 89.26 0.10 -0.19 23.65 440.00
-0.70 -1.04 -0.05 0.10 0.05 9.25 0.09 -0.21 23.65 0.62 -0.32 89.31 0.11 -0.21 26.93 435.00
-0.69 -1.04 -0.05 0.10 0.05 9.25 0.10 -0.24 26.93 0.61 -0.29 89.33 0.12 -0.24 30.05 440.00
-0.69 -1.04 -0.07 0.11 0.05 9.24 0.11 -0.26 30.05 0.67 -0.28 89.28 0.14 -0.26 33.01 435.00
-0.68 -1.07 -0.08 0.11 0.05 9.24 0.13 -0.28 33.01 0.69 -0.28 89.26 0.15 -0.28 35.82 435.00
-0.68 -1.05 -0.05 0.11 0.04 9.23 0.14 -0.30 35.82 0.67 -0.22 89.30 0.17 -0.30 38.49 435.00
-0.68 -1.08 -0.02 0.11 0.04 9.24 0.15 -0.32 38.49 0.68 -0.24 89.28 0.18 -0.32 41.03 435.00
-0.68 -1.08 -0.05 0.09 0.05 9.23 0.17 -0.34 41.03 0.59 -0.33 89.33 0.19 -0.34 43.45 440.00
-0.71 -1.03 -0.08 0.10 0.05 9.24 0.17 -0.36 43.45 0.63 -0.32 89.30 0.20 -0.36 45.74 435.00
-0.69 -1.04 -0.05 0.11 0.05 9.25 0.18 -0.38 45.74 0.69 -0.31 89.25 0.21 -0.37 47.91 440.00
-0.69 -1.04 -0.06 0.10 0.05 9.25 0.19 -0.39 47.91 0.64 -0.28 89.31 0.22 -0.39 49.98 435.00
-0.69 -1.08 -0.06 0.10 0.04 9.25 0.20 -0.41 49.98 0.64 -0.24 89.32 0.22 -0.40 51.95 435.00
-0.69 -1.04 -0.05 0.10 0.04 9.23 0.21 -0.42 51.95 0.62 -0.26 89.33 0.23 -0.41 53.82 435.00
-0.69 -1.06 -0.08 0.10 0.06 9.24 0.22 -0.44 53.81 0.63 -0.36 89.28 0.24 -0.43 55.59 435.00
-0.69 -1.05 -0.05 0.10 0.04 9.23 0.22 -0.45 55.59 0.65 -0.24 89.32 0.24 -0.44 57.27 445.00
^n nn ^n nn

```

Εικόνα 45. Εκτύπωση αποτελεσμάτων debugging του *MPU – 6050* στη σειριακή οθόνη .

Όπως παρατηρούμε οι τιμές των δεδομένων απεικόνισης αντιπροσωπεύουν τις γωνίες που έχουν υπολογιστεί στον εκάστοτε αισθητήρα σε όλες τις φάσεις υπολογισμού αλλά και την τελική γωνία. Στο *debug.h* αρχείο έχουμε ορίσει μία κλάση με το όνομα *DEBUG_CLASS* στην οποία ως *public* μεταβλητή τοποθετούμε την *float SensorArray[3]*. Η μεταβλητή αυτή κατέχει τις τιμές ανά τριάδες των δεδομένων που

εισάγονται στο matlab μέσω της σειριακής. Ύστερα συναντάμε την αρχικοποίηση της κλάσης μέσω της συνάρτησης του class constructor DEBUG_CLASS() : SensorArray(), frac(0), frac1(0) { } αρχικοποιόντας τις private μεταβλητές κατευθείαν, χωρίς να ορίσουμε την συνάρτηση αρχικοποίησης σε άλλο αρχείο.

Η συνάρτηση void Debug_XYZ_matlab_Print(float XYZ_matlab_data_logging[]) είναι υπεύθυνη για την εισαγωγή των δεδομένων στο *Matlab* και σαν όρισμα έχει ένα διάνυσμα τριών θέσεων τύπου float. Τέλος συναντάμε τη συνάρτηση void Debug_printDouble(double val, unsigned int precision) η οποία εκτυπώνει δεκαδικούς αριθμούς και την οποία χρησιμοποιούμε με το κατάλληλο όρισμα ακριβείας για την επιθυμητή απεικόνιση. Για να εκτελέσουμε την απεικόνιση μέσω του *Matlab*, αποθηκεύουμε πρώτα τις τιμές που θέλουμε να απεικονίσουμε στο διάνυσμα SensorArray[] το οποίο μετά χρησιμοποιούμε σαν όρισμα εισόδου στη συνάρτηση DEBUGObj. Debug_XYZ_matlab_Print(DEBUGObj.SensorArray) που εμπεριέχεται στη κλάση DEBUG_CLASS. Παρακάτω απεικονίζουμε τον αλγόριθμο debug.h και debug.cpp.

Listing 7. Debug.h Report c++ language technical details

```
/*
-----
-----
MSc HEP 2013-2014
-----
*/
/*
-----
Project      : Quadcopter
File         : MPU_6050_Control.ino
Description   : This code calibrates, test and debug MPU-6050 sensor
Author       : Monahopoulos Konstantinos
-----
*/
/*
-----
DEBUG CLASS DECLARATION
-----
*/
#ifndef DEBUG_H
#define DEBUG_H

class DEBUG_CLASS
{
public:
    float SensorArray[3];
}
```

```

DEBUG_CLASS() : SensorArray(),frac(0),frac1(0) { } //class constructor
void Debug_XYZ_matlab_Print(float XYZ_matlab_data_logging[]);
void Debug_printDouble( double val, unsigned int precision);

private:
unsigned int frac;
int frac1;
};

#endif

```

Listing 8. Debug.cpp Report c++ language technical details

```

/*
-----
MSC HEP 2013-2014
-----

*/
/*
-----


Project      : Quadcopter
File         : MPU_6050_Control.ino
Description   : This code calibrates, test and debug MPU-6050 sensor
Author       : Monahopoulos Konstantinos
-----


*/



#include <Arduino.h>
#include "debug.h"

/*
-----
Function Prototyping
-----
*/
// -----
// Print Data To Serial for Matlab Data acquisition
// -----
void DEBUG_CLASS::Debug_XYZ_matlab_Print(float XYZ_matlab_data_logging[]){

    Debug_printDouble(XYZ_matlab_data_logging[0],100);
    Serial.print(" ");

    Debug_printDouble(XYZ_matlab_data_logging[1],100);
    Serial.print(" ");

    Debug_printDouble(XYZ_matlab_data_logging[2],100);
    Serial.println(" ");
}

// -----
// Prints into Serial Double Values
// -----
void DEBUG_CLASS::Debug_printDouble( double val, unsigned int precision){

    /* prints val with number of decimal places determine by precision

```

```

NOTE: precision is 1 followed by the number of zeros for the desired
number of decimal places
example: printDouble( 3.1415, 100); // prints 3.14 (two decimal places) */

Serial.print (int(val)); //prints the int part
Serial.print(".");
if(val >= 0)
    frac = (val - int(val)) * precision;
else
    frac = (int(val)- val ) * precision;
    frac1 = frac;
while( frac1 /= 10 )
    precision /= 10;
precision /= 10;
while( precision /= 10 )
    Serial.print("0");

Serial.print(frac,DEC) ;
}

```

Ο κώδικας *matlab* που έχουμε αναπτύξει έχει την ικανότητα να ολισθαίνει το παράθυρο απεικόνισης στο χρόνο ενώ είναι σε θέση επιλεκτικά να ανοίγει και να κλείνει τη σειριακή θύρα με την έναρξη ή με το κλείσιμο του παραθύρου. Αυτό μας επιτρέπει την μη δέσμευση της σειριακής θύρας έτσι ώστε να μπορούμε να τον επανα - εκτελέσουμε το κώδικα αλλάζοντας απλώς τον άξονα απεικόνισης.

Listing 9. SensorDataLogging.m Report technical details

```

%-----%
% Project      : Quadcopter
% File         : SensorDataLogging.m
% Description   : Plots graphical data acquisition from Arduino
% Author        : Monahopoulos Konstantinos
%-----%

clear
clc
close all;

%User Defined Properties
xLabel = 'Time --> (s)';           % x-axis label
yLabel = 'Data';                   % y-axis label
min = -100;                      % set y-min
max = 100;                       % set y-max
scrollWidth = 10;                 % display period in plot
delay = .001;                     % make sure sample faster than resolution
Trash = 0;                        % Initialize Trash
CoordinatePlot='X';               % Select Coordinates

%Open Serial COM Port
s = serial('COM4'); % define Arduino COM port #
disp('Close Plot to End Session');
fopen(s);

%Define Function Variables
time = 0;
SensorData = 0;
count = 0;

```

```

switch(CoordinatePlot)
    case 'X'
        %%%%%% Setup First Plot %%%%%%
        plotGraph = plot(time,SensorData,'-mo',...
            'LineWidth',1,...
            'MarkerEdgeColor','b',...
            'MarkerFaceColor',[0 0 0],...
            'MarkerSize',2);

        title('X coordinates','FontSize',15);
        xlabel(xLabel,'FontSize',5);
        ylabel(yLabel,'FontSize',5);
        axis([0 10 min max]);
        grid('on');

    case 'Y'
        %%%%%% Setup Second Plot %%%%%%
        figure;
        plotGraph2 = plot(time,SensorData,'-mo',...
            'LineWidth',1,...
            'MarkerEdgeColor','b',...
            'MarkerFaceColor',[0 0 0],...
            'MarkerSize',2);

        title('Y coordinates','FontSize',15);
        xlabel(xLabel,'FontSize',5);
        ylabel(yLabel,'FontSize',5);
        axis([0 10 min max]);
        grid('on');

    case 'Z'
        %%%%%% Setup Third Plot %%%%%%
        figure;
        plotGraph3 = plot(time,SensorData,'-mo',...
            'LineWidth',1,...
            'MarkerEdgeColor','b',...
            'MarkerFaceColor',[0 0 0],...
            'MarkerSize',2);

        title('Z coordinates','FontSize',15);
        xlabel(xLabel,'FontSize',5);
        ylabel(yLabel,'FontSize',5);
        axis([0 10 min max]);
        grid('on');
    otherwise
        disp('Select Coordinates to Display');
end

tic
while(1)
    %Loop while Plot is Active

    if (s.bytesavailable>0 && Trash>100)

        Stringdata = fscanf(s);
        SensorData=str2num(Stringdata);
        while (isempty(SensorData))
            Stringdata = fscanf(s);
            SensorData=str2num(Stringdata);
        end
        count = count + 1;
        time(count) = toc;      %Extract Elapsed Time
        X_coor_Data(count) = SensorData(1); %Extract 1st Data Element
        Y_coor_Data(count) = SensorData(2); %Extract 2st Data Element
        Z_coor_Data(count) = SensorData(3); %Extract 3st Data Element

        %Set Axis according to Scroll Width

```

```

if(scrollWidth > 0)

    switch(CoordinatePlot)
        case 'X'
            if (ishandle(plotGraph)==0)
                break;
            end
            set(plotGraph, 'XData',time(time > time(count)-
scrollWidth)...
            , 'YData',X_coor_Data(time > time(count)-
scrollWidth));
            axis([time(count)-scrollWidth time(count) min max]);

        case 'Y'
            if (ishandle(plotGraph2)==0)
                break;
            end
            set(plotGraph2, 'XData',time(time > time(count)-
scrollWidth)...
            , 'YData',Y_coor_Data(time > time(count)-
scrollWidth));
            axis([time(count)-scrollWidth time(count) min max]);

        case 'Z'
            if (ishandle(plotGraph3)==0)
                break;
            end
            set(plotGraph3, 'XData',time(time > time(count)-
scrollWidth)...
            , 'YData',Z_coor_Data(time > time(count)-
scrollWidth));
            axis([time(count)-scrollWidth time(count) min max]);
        otherwise
            disp('Select Coordinates to Display');
        end
    end

    %Allow MATLAB to Update Plot
    pause(delay);
end
Trash=Trash+1; % Count 100 Trashes from Serial and after collect data
end

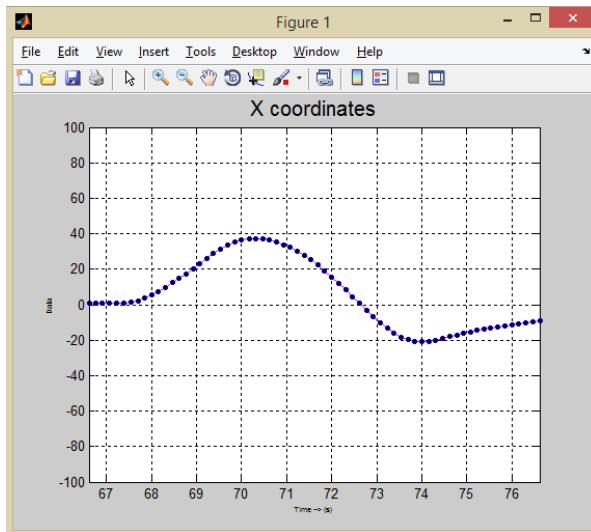
%Close Serial COM Port and Delete useless Variables
priorPorts=instrfind;
fclose(priorPorts);
fclose(s);

clear count dat delay max min plotGraph plotGrid plotTitle s ...
scrollWidth serialPort xLabel yLabel;

disp('Session Terminated...');


```

Έχοντας σταθεροποιήσει το *Quadcopter*, κουνώντας μόνο τη μεριά που αναπαριστά τη *Roll* κίνηση παρατηρούμε τα παρακάτω αποτελέσματα.



Εικόνα 46. Εκτύπωση αποτελεσμάτων debugging του MPU – 6050 στο Matlab .

Ανάλογα αποτελέσματα θα απεικονιστούν και με τους άλλους άξονες τοποθετώντας ανάλογα το *Quadcopter* στο χώρο αλλάζοντας μόνο τον άξονα απεικόνισης της μεταβλητής *CoordinatePlot* .

3.8.2.7 ΠΡΟ – ΒΑΘΜΟΝΟΜΗΣΗ ΤΟΥ MPU-6050

Έχοντας εξηγήσει τη διαδικασία του *calibration* κατά την αρχικοποίηση, σε αυτό το σημείο θα απεικονίσουμε τον αλγόριθμο που μας εξάγει τις τιμές *precalibration* τις οποίες χρησιμοποιούμε στον υπολογισμό της γωνίας αφαιρώντας το offset. Εφόσον έχουμε τοποθετήσει τον αισθητήρα σε πλήρως επίπεδο δάπεδο και έχουμε σιγουρευτεί για τη κλίση του (να είναι μηδέν) εκτελούμε τον αλγόριθμο και τον αφήνουμε να τρέχει για περίπου 4-5 λεπτά. Τότε εφόσον οι τιμές έχουν κατασταλάξει σε μία σταθερή κατάσταση τις καταγράφουμε κα τις τοποθετούμε στη συνάρτηση `Set_MP6050_Precalibration_Offset(void)` .

Listing 10. MPU_6050_Precalibration.ino Report c++ language technical details

```
/*
-----
-----  
MSC HEP 2013-2014  
-----
*/
/*
```

```

-----
Project      : Quadcopter
File         : MPU_6050_Precalibration.ino
Description   : This code calculates the pre - calibratied values of MPU-
6050
Author       : Monahopoulos Konstantinos
-----

*/
// Structure:
// -----
// Arduino | -----
// UNO    |- 3.3 V -----| MPU 6050 |
//          |- GND -----| HMC5883, |
//          |- SDA -----| MS5611  |
//          |- SCL -----|           |
//          |-----|
//          |-----|_____|
//          |-----| V
//          |-----| Integrated IMU sensor
/*
-----
Includes
-----
*/
#include <Wire.h>
#include <Arduino.h>

/*
First Place the IMU-6050 in flat surface.

Run this Algorithm for 4-5 minutes and write down the offset's values,
these offsets has to be registered in main Gyro_header.ino project in
function :
void Set_MP6050_Precalibration_Offset(void).

DO not let the programm run inf because it will overflow.

*/
/*
-----
DEFINITIONS
-----
*/
#define dt 20
#define rad2degree 57.3
#define Filter_gain 0.95

/*
-----
GLOBAL VARIABLE DECLARATION
-----
*/
*/
long acc_counterx=0;
long acc_countery=0;
long acc_counterz=0;
long gyro_counterx=0;
long gyro_countery=0;

```

```

long gyro_counterz=0;
long loop_counter;
int temp=0, accel_x=0, accel_y=0, accel_z=0, gyro_x=0, gyro_y=0, gyro_z=0;
unsigned long t=0;

void setup(){
    Serial.begin(9600);
    Wire.begin();

    // -----
    //      WAKE UP MPU
    // -----
    Wire.beginTransmission(0x69);
    Wire.write(107);
    Wire.write(0b10000000);
    Wire.endTransmission();
    delay(100);
    Wire.beginTransmission(0x69);
    Wire.write(107);
    Wire.write(0b00000000);
    Wire.endTransmission();

    // -----
    //      SCALE FACTOR
    // -----
    // Setting up Gyro
    Wire.beginTransmission(0x69);
    Wire.write(27);
    Wire.write(0b00000000);
    Wire.endTransmission();

    // Setting up Accel
    Wire.beginTransmission(0x69);
    Wire.write(28); // Address to the configuration register
    Wire.write(0b00001000);
    Wire.endTransmission();

    // -----
    //      FILTER BANDWIDTH IMU DEFINITION
    // -----
    Wire.beginTransmission(0x69);
    Wire.write(26); // Address to the configuration register
    Wire.write(0);
    Wire.endTransmission();

    Serial.print("\t(1)Offset Value Gyro X");
    Serial.print("\t(2)Offset Value Gyro Y");
    Serial.print("\t(3)Offset Value Gyro Z");
    Serial.print("\t(4)Offset Value ACC X");
    Serial.print("\t(5)Offset Value ACC Y");
    Serial.print("\t(6)Offset Value ACC Z");
    Serial.println("");
    Serial.println("");
}

void loop(){

    t=millis();
    loop_counter+=1;
    Wire.beginTransmission(0x69);
    Wire.write(59);
    Wire.endTransmission();
    Wire.requestFrom(0x69,14);

    if(Wire.available() == 14){

```

```

accel_x = Wire.read()<<8 | Wire.read();
accel_y = Wire.read()<<8 | Wire.read();
accel_z = Wire.read()<<8 | Wire.read();
temp = Wire.read()<<8 | Wire.read();
gyro_x = Wire.read()<<8 | Wire.read();
gyro_y = Wire.read()<<8 | Wire.read();
gyro_z = Wire.read()<<8 | Wire.read();
}

gyro_counterx=gyro_counterx+gyro_x;
gyro_countery=gyro_countery+gyro_y;
gyro_counterz=gyro_counterz+gyro_z;

acc_counterx=acc_counterx+accel_x;
acc_countery=acc_countery+accel_y;
acc_counterz=acc_counterz+accel_z;

Serial.print(round(gyro_counterx/loop_counter));
Serial.print("\t");
Serial.print(round(gyro_countery/loop_counter));
Serial.print("\t");
Serial.print(round(gyro_counterz/loop_counter));
Serial.print("\t");
Serial.print(round(acc_counterx/loop_counter));
Serial.print("\t");
Serial.print(round(acc_countery/loop_counter));
Serial.print("\t");
Serial.println(round(acc_counterz/loop_counter));

while((millis()-t) < dt){ // Making sure the cycle time is equat to dt
// Do nothing
}
delay(250);
}

```

Παρακάτω επισυνάπτουμε τον κύριο κώδικα που σε συνεργασία με τα υπόλοιπα αρχεία εκτελεί τη προαναφερθείσα διαδικασία.

Listing 11. MPU_6050_Control.ino Report c++ language technical details

```

/*
-----
-----MSC HEP 2013-2014-----
-----

*/
/*
-----
-----Project      : Quadcopter
File          : MPU_6050_Control.ino
Description   : This code calibrates, test and debug MPU-6050 sensor
Author        : Monahopoulos Konstantinos
-----


*/
// Structure:
// -----
// Arduino  |
-----
```

```

//  UNO      | - 3.3 V ----- | MPU 6050      |
//           | - GND -----| HMC5883,       |
//           | - SDA -----| MS5611        |
//           | - SCL -----|                  |
//           |                  |
//-----          | _____ | 
//                   V
//                   Integrated IMU sensor

/*
-----
Includes
-----
*/
#include <Wire.h>
#include <Servo.h>
#include "gyro_accel.h"
#include "debug.h"
/*
-----
DEFINITIONS
-----
*/
/* DEFINE QUAD_CALIBRATION TO CALIBRATE QUAD ON INITIALIZATION */
//#define QUAD_CALIBRATION

/* DEFINE QUAD_CALIBRATION TO USE PRECALIBRATION OFFSET QUAD ON
INITIALIZATION */
#define PRECALIBRATION

/* DEFINE MATLAB_DATA_LOGGING TO DEBUG SENSOR VALUES TO MATLAB */
//#define MATLAB_DATA_LOGGING

/* DEFINE DEBUG TO DEBUG SENSOR VALUES TO ARDUINO */
#define DEBUG

#define rad2degree 57.3          // Radian to degree conversion

/* default Filter_gain = 0.95 . Lowering the Filter Gain means using more
the accelerometer to calculate so it reach to the angle value quicker.
But the more the Filter_gain reduced the more noise you get from
accelerometer. If you set Filter_gain=1 means using only Gyroscope
measurement.
*/
#define Filter_gain 0.95
#define MAX_SIGNAL 2000 // maximum us at esc
#define MIN_SIGNAL 900  // minimum us at esc
#define STATUS_LED 13   // Led in pin 13
#define dt 20 // time diffrence in millis seconds --> loop constat 100hz

/*
-----
Global Variables
-----
*/
unsigned long t; // Time Variables
/*
-----

```

```

Assign Classes
-----
*/
MPU6050_CLASS MPU6050obj;

DEBUG_CLASS DEBUGobj;

/*
-----
Main Code
-----
*/
void setup(){
    Serial.begin(9600);

/*
-----
Led Configuration
-----
*/
    pinMode(STATUS_LED, OUTPUT);
/*
-----
MPU6050 Configuration
-----
*/
    Wire.begin();

    MPU6050obj.MPU6050_ResetWake();
    MPU6050obj.MPU6050_SetGains(0,1); // Setting the low scale
    MPU6050obj.MPU6050_SetDLPF(0); // Setting the DLPF to inf Bandwidth for
calibration
/*
-----
Definitions Configuration
-----
*/
#endif QUAD_CALIBRATION
    MPU6050obj.MPU6050_OffsetCal();
#endif PRECALIBRATION
    MPU6050obj.MPU6050_Set_PreCalibration_offset();
MPU6050obj.MPU6050_SetDLPF(6); // Setting the DLPF to lowest Bandwidth
/*
-----
DEBUG Printing Messages
-----
*/
}

```

```

#define DEBUG
Serial.print("(1)gyro_x_scalled");
Serial.print("\t(2)gyro_y_scalled");
Serial.print("\t(3)gyro_z_scalled");

Serial.print("\t(4)accel_x_scalled");
Serial.print("\t(5)accel_y_scalled");
Serial.print("\t(6)accel_z_scalled");

Serial.print("\t(7)angle_x_gyro");
Serial.print("\t(8)angle_y_gyro");
Serial.println("\t(9)angle_z_gyro");

Serial.print("(10)angle_x_accel");
Serial.print("\t(11)angle_y_accel");
Serial.print("\t(12)angle_z_accel");

Serial.print("\t(1)angle_x");
Serial.print("\t\t(1)angle_y");
Serial.print("\t\t(15)angle_z");

Serial.println("\t\t(16)Load");
Serial.println("");
#endif
}

void loop() {
    /* AT THE BEGINNING OF EVERY LOOP I MUST CHECK THE STRENGTH OF THE XBEE
    RANGE. IF STRENGTH IS LOWER THAN A VALUE ...
    LOWER THE MOTOR SPEED UNTIL HIGH EQUALS TO ...
    */
    t=millis();
    /*
    -----
    -----
    MAIN PROCESS
    -----
    */
    MPU6050obj.MPU6050_ReadData();

    // -----
    //      Read Gyro Values From MPU6050
    // -----
    MPU6050obj.angle_x_gyro =
    (MPU6050obj.gyro_x_scalled*((float)dt/1000)+MPU6050obj.angle_x);
    MPU6050obj.angle_y_gyro =
    (MPU6050obj.gyro_y_scalled*((float)dt/1000)+MPU6050obj.angle_y);
    MPU6050obj.angle_z_gyro =
    (MPU6050obj.gyro_z_scalled*((float)dt/1000)+MPU6050obj.angle_z);

    // -----
    //      Read Accelerometer Values From MPU6050
    // -----
    MPU6050obj.angle_x_accel = atan(MPU6050obj.accel_y_scalled
    / (sqrt(MPU6050obj.accel_x_scalled*MPU6050obj.accel_x_scalled+MPU6050obj.acce
    l_z_scalled*MPU6050obj.accel_z_scalled)))*(float)rad2degree;
    MPU6050obj.angle_y_accel = -
    atan(MPU6050obj.accel_x_scalled/(sqrt(MPU6050obj.accel_y_scalled*MPU6050obj.
    accel_y_scalled+MPU6050obj.accel_z_scalled*MPU6050obj.accel_z_scalled)))*(fl
    oat)rad2degree;
}

```

```

    MPU6050obj.angle_z_accel =
atan(MPU6050obj.accel_z_scaled/(sqrt(MPU6050obj.accel_y_scaled*MPU6050obj.
accel_y_scaled+MPU6050obj.accel_x_scaled*MPU6050obj.accel_x_scaled)))*(f1
oat)rad2degree;

// -----
//      Compute Overall Angle
// -----
MPU6050obj.angle_x = Filter_gain*MPU6050obj.angle_x_gyro+(1-
Filter_gain)*MPU6050obj.angle_x_accel;
MPU6050obj.angle_y = Filter_gain*MPU6050obj.angle_y_gyro+(1-
Filter_gain)*MPU6050obj.angle_y_accel;
MPU6050obj.angle_z = Filter_gain*MPU6050obj.angle_z_gyro+(1-
Filter_gain)*MPU6050obj.angle_z_accel;
/*
-----
Matlab Data Acquisition for Debug
-----
*/
/*You can Send to Matlab for plotting Only three variables at a time, you
have a choice Between : */

/* gyro_x_scaled - gyro_y_scaled - gyro_z_scaled */
/* accel_x_scaled - accel_y_scaled - accel_z_scaled */
/* angle_x_gyro - angle_y_gyro - angle_z_gyro */
/* angle_x_accel - angle_y_accel - accel_z_scaled */
/* angle_x - angle_y - angle_z */

#endif MATLAB_DATA_LOGGING

DEBUGObj.SensorArray[0]=MPU6050obj.angle_x;
DEBUGObj.SensorArray[1]=MPU6050obj.angle_y;
DEBUGObj.SensorArray[2]=MPU6050obj.angle_z;

DEBUGObj.Debug_XYZ_matlab_Print(DEBUGObj.SensorArray);
delay(150); // Delay 150 so matlab can catch the values ..

#endif

// -----
//      Arduino Data Acquisition for Debug
// -----
#endif DEBUG

Serial.print(MPU6050obj.gyro_x_scaled);
Serial.print("\t");
Serial.print(MPU6050obj.gyro_y_scaled);
Serial.print("\t");
Serial.print(MPU6050obj.gyro_z_scaled);
Serial.print("\t");

Serial.print(MPU6050obj.accel_x_scaled);
Serial.print("\t");
Serial.print(MPU6050obj.accel_y_scaled);
Serial.print("\t");
Serial.print(MPU6050obj.accel_z_scaled);
Serial.print("\t");

Serial.print(MPU6050obj.angle_x_gyro);
Serial.print("\t");
Serial.print(MPU6050obj.angle_y_gyro);
Serial.print("\t");
Serial.print(MPU6050obj.angle_z_gyro);
Serial.print("\t");

```

```

Serial.print(MPU6050obj.angle_x_accel);
Serial.print("\t");
Serial.print(MPU6050obj.angle_y_accel);
Serial.print("\t");
Serial.print(MPU6050obj.angle_z_accel);
Serial.print("\t");

Serial.print(MPU6050obj.angle_x);
Serial.print("\t");
Serial.print(MPU6050obj.angle_y);
Serial.print("\t");
Serial.print(MPU6050obj.angle_z);
Serial.print("\t");

Serial.println((float)(millis()-t)/(float)dt)*100);

#endif

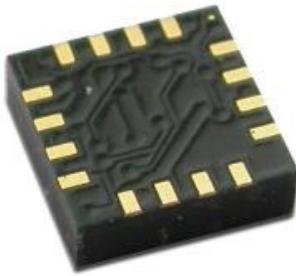
while((millis()-t) < dt){ } // Just making sure the cycle time is constant
}

```

3.8.3 ΜΑΓΝΗΤΟΜΕΤΡΟ (HMC5883L)

Ο αισθητήρας *Honeywell HMC5883L* αποτελεί μια επιφανειακή μονάδα μιας *multi-chip* διάταξης, ο οποίος είναι σχεδιασμένος να ανιχνεύει μαγνητικά πεδία χαμηλής ισχύος. Ο *HMC5883L* προορίζεται για χαμηλού κόστους εφαρμογές, πετυχαίνοντας όμως υψηλή ποιότητα και χαμηλό θόρυβο μέσω των μαγνητο – αντιστάσεων της σειράς *HMC118X* που ενσωματώνει. Επιπροσθέτως, ικανοποιεί διαδικασίες αυτό-ακύρωσης του offset όπως επίσης και έναν *ADC* 12-bit που επιτρέπει ακρίβεια πυξίδας 1° και 2° . Η συσκευή αυτή λειτουργεί μέσω του σειριακού δίαυλου *I2C* ο οποίος επιτρέπει και την εύκολη διασύνδεση. Πιο συγκεκριμένα, ο *HMC5883L* είναι ένα *SoC* με διαστάσεις 3.0x3.0x0.9mm, ενώ χρησιμοποιείται σε μια πληθώρα από εφαρμογών, όπως κινητά τηλέφωνα, *Netbooks*, *Consumer Electronics*, συστήματα πλοήγησης αυτοκινήτου, και Προσωπικές Συσκευές Πλοήγησης.

Επίσης ο αισθητήρας *HMC5883L* χρησιμοποιεί την τεχνολογία της *Honeywell magnetoresistive (AMR)* η οποία παρέχει πλεονεκτήματα σε σχέση με άλλους παρόμοιους αισθητήρες. Ο υπολογισμός της κατεύθυνσης μέσω των αισθητήρων διαθέτει μεγάλη ακρίβεια ενώ ανταποκρίνεται με μεγάλη ευαισθησία και γραμμικότητα. Επιπροσθέτως διαθέτει την ιδιότητα να μετρά εκτός από την κατεύθυνση στο χώρο, και το μαγνητικό πεδίο ως προς τη γη σε βαθμίδες της τάξεως του *milli - gauss* έως και 8 *gauss* [Ref 24].



Εικόνα 47. Ο αισθητήρας Honeywell magnetoresistive (AMR).

Επιπλέον, εφόσον οι μετρούμενες τιμές πραγματοποιούνται σε αναφορά με το διάνυσμα αναφοράς το οποίο στοχεύει στο βορρά, παρέχονται τρία νέα διανύσματα μέσω των οποίων είμαστε σε θέση να υπολογίσουμε το *Yaw*, *Pitch*, και *Roll*, με μεγάλη σταθερότητα στο χρόνο και με ολίσθηση μικρότερη της μιας μοίρας. Τέλος αυτός ο αισθητήρας είναι σε θέση να συνεργαστεί τέλεια με τον *MPU-6050* αισθητήρα, παράγοντας επιτυχημένα τη κατεύθυνση κίνησης του αντικειμένου [Ref 25].

3.8.3.1 ΥΠΟΛΟΓΙΣΜΟΣ ΓΩΝΙΑΣ ΜΕ ΧΡΗΣΗ ΤΟΥ ARDUINO UNO

Για τη λειτουργία του μαγνητόμετρου η συνδεσμολογία που πρέπει να πραγματοποιήσουμε για την σωστή εγκαθίδρυση στο *Arduino* ακολουθεί την εξής λογική. Θα πρέπει να συνδέσουμε την αναλογική είσοδο του *Arduino* (pin *Analog input 4*) που χρησιμοποιεί το *I2C Bus* στο *SDA* της συσκευής. Ανάλογα συνδέσουμε την αναλογική είσοδο του *Arduino* (pin *Analog input 5*) στο *SCL* της συσκευής. Τέλος συνδέουμε ως τάση εισόδου στον αισθητήρα την τάση εξόδου 3.3 volt του *Arduino* και την γείωση του μικροελεγκτή στο *GND* του *10DOF*. Πρέπει να σημειωθεί ότι είναι προτιμότερο να τροφοδοτούμε τον αισθητήρα με χαμηλή τάση (3.3v) παρά με 5v, αλλά για να πραγματοποιηθεί αυτό θα πρέπει να απενεργοποιήσουμε τις ενσωματωμένες *pull – up* αντιστάσεις του *Arduino*. Αυτό γίνεται αρχικά πηγαίνοντας στις βιβλιοθήκες του *Arduino* στο φάκελο *Arduino\libraries\Wire\utility* και εντοπίζοντας το αρχείο *twi.c* ανοίγοντάς το με κάποιον editor. Και στη συνέχεια εντοπίζουμε τις γραμμές *digitalWrite(SDA, 1);* και *digitalWrite(SCL, 1);* και τις σχολιάζουμε. Με αυτό τον τρόπο έχουμε απενεργοποιήσει σε κάθε sketch που αναπτύσσουμε τις *pull – up* αντιστάσεις στα pin A4 και A5 του μικροελεγκτή.



Εικόνα 48. Μαγνητομετρο (HMC5883L) .

Για την σωστή λειτουργία του προγράμματος πρέπει να συμπεριλάβουμε στο *project* το αρχείο <Wire.h> το οποίο και κάνει τη διαχείριση της συσκευής που έχει τοποθετηθεί στο διάδρομο I2C του *Arduino*. Επίσης το HMC5883L θα πρέπει να έχει ελεγχθεί ότι εντοπίζεται στη διεύθυνση 0x1E όπως ορίζεται στον εντοπισμό των συσκευών μέσω του Scanner_port_report που δείξαμε παραπάνω. Τα αρχεία που έχουν δημιουργηθεί για τον έλεγχο του μαγνητόμετρου είναι τα compass.cpp και compass.h. Σε αυτό το σημείο θα περιγράψουμε κομμάτια του κώδικα που αφορούν την εκτέλεση του HMC5883L και μόνο. Το project που αναλαμβάνει την εκτέλεση και εμφάνιση των αποτελεσμάτων είναι το Compass_HMC5883L.ino.

Για την ορθή χρήση του HMC5883L, έχουμε αναπτύξει ένα πρόγραμμα το οποίο χρησιμοποιεί το μαγνητόμετρο ως πυξίδα στο δύο διαστάσεων χώρο (x και y) που εντοπίζονται παράλληλα στο έδαφος. Επιπροσθέτως, πρέπει να αναφέρουμε ότι η πυξίδα δεν έχει σαν διάνυσμα αναφοράς το πραγματικό βορρά, αλλά αυτόν που υπολογίζουμε μέσω της διαδικασίας της ρύθμισης. Ούτως ή αλλιώς δεν μας απασχολεί το πού εντοπίζονται τα ημισφαίρια αλλά μας απασχολεί να γνωρίζουμε τον προσανατολισμό του *Quadcopter* με κάποιο σημείο αναφοράς στο δύο διαστάσεων χώρο. Το heading του *Quadcopter* θα συμπεριληφθεί στον υπολογισμό του PID και θα επηρεάσει σε κάποιο βαθμό την ταχύτητα των μοτέρ, για να καταφέρουμε να ευθυγραμμίσουμε το *Quadcopter* στο χώρο.

Πρώτη εργασία που πρέπει να πραγματοποιηθεί είναι να αρχικοποιηθεί η επικοινωνία του μαγνητόμετρου με το *Arduino*, το οποίο πραγματοποιείται μέσω της I2C συνάρτησης “Wire.begin();”. Το μαγνητόμετρο ύστερα από κάθε κύκλο ανάγνωσης

τίθεται σε sleep mode και μόνο η συνάρτηση `compass_scalled_reading()` «ξυπνά» τον αισθητήρα για την ανανέωση των τιμών.

Στο αρχείο `compass.h` έχουμε δημιουργήσει μία κλάση `COMPASS_CLASS` η οποία εμπεριέχει όλες τις απαραίτητες πληροφορίες που θα χρειαστούμε. Εκτός από την συνάρτηση αρχικοποίησης της κλάσης `COMPASS_CLASS(void)` που λειτουργεί σαν `class constructor` εντοπίζονται και άλλες χρήσιμες συναρτήσεις τις οποίες θα περιγράψουμε εκτενέστερα παρακάτω. Σε πρώτη φάση συναντάμε τη συνάρτηση `compass_init(int gain)` η οποία σκοπό έχει την αρχικοποίηση του μαγνητόμετρου. Πιο αναλυτικά μέσω της μεταβλητής `compass_gain_fact` μπορούμε να θέσουμε τα όρια μέτρησης του μαγνητόμετρου, γράφοντας παράλληλα στον εκάστοτε `buffer` την δυαδική τιμή που αναλογεί μέσω της τύπου `byte` μεταβλητής `gain_reg`. Τα μαγνητικά πεδία αποτελούν διακριτά διανύσματα που χαρακτηρίζονται τόσο από την ισχύ του πεδίου όσο και από την κατεύθυνση. Η ισχύς του πεδίου μετράται σε μονάδες *Tesla* στο SI και σε μονάδες *Gauss* στο cgs σύστημα.

Το όρισμα της συνάρτησης αρχικοποίησης θέτει την ισχύ του μαγνητικού πεδίου σε μονάδα μέτρησης *gauss* το οποίο στη δικιά μας περίπτωση επιλέγεται να είναι 1.22 *gauss*. Αργότερα συναντάμε τη συνάρτηση `compass_scalled_reading()` η οποία συνάρτηση μέσω του `compass_gain_error` στον ανάλογο άξονα υπολογίζει τις σωστές τιμές μέτρησης του μαγνητόμετρου. Όπως και στον *MPU-6050* αισθητήρα, έτσι και εδώ οι τιμές αυτές είτε προ – υπολογίζονται μέσω του *pre-calibration* και τις θέτουμε απλά σαν *constant* παραμέτρους, είτε μπορούμε να τις υπολογίσουμε κατά την εκκίνηση του προγράμματος του *Quadcopter*. Συνεπώς, για τον υπολογισμό των τιμών `compass_x_scalled`, `compass_y_scalled` και `compass_z_scalled` πραγματοποιούμε ανάγνωση των *raw data* του μαγνητόμετρου, μέσω της συνάρτησης `compass_read_XYZdata()` και πολλαπλασιάζουμε με το `compass_gain_fact` και το `compass_gainError` του ανάλογου άξονα. Τέλος προσθέτουμε το `offset` που έχει υπολογιστεί από τη διαδικασία του *calibration*. Άρα πλέον οι τιμές έχουν αναβαθμιστεί και δεν μένει τίποτε άλλο από τη μετατροπή τους σε μοίρες. Η διαδικασία αυτή συντελείται μέσω της συνάρτησης `compass_heading()` η οποία αφού έχει διαβάσει τις `scalled` τιμές τότε αναλόγως του τεταρτημόριου που βρίσκεται το `y-heading` του μαγνητόμετρου πραγματοποιεί την ανάλογη τριγωνομετρική πράξη. Πιο αναλυτικά η σχέση (*bearing*) που διέπει τον υπολογισμό αυτό θα είναι :

- Για μέτρηση του γάξονα σε τιμή μεγαλύτερη του μηδενός
bearing = 90-
atan(compass_x_scaled/compass_y_scaled)*compass_rad2degree;
- Για μέτρηση του γάξονα σε τιμή μικρότερη του μηδενός :
bearing = 270-
atan(compass_x_scaled/compass_y_scaled)*compass_rad2degree;
- συννυπολογισμένου του x γάξονα για y = 0 και x < 0 θα είναι
bearing = 180;
- Σε διαφορετική περίπτωση
bearing = 0

Τέλος εκτυπώνουμε τις μοίρες που έχουν υπολογιστεί μέσω της σειριακής θύρας του *Arduino*. Τα αποτελέσματα που εξάγονται μέσω της σειριακής χρησιμοποιόντας *pre – calibration offset* τιμές για περιστροφή 360 μοιρών του αισθητήρα είναι τα παρακάτω.

```

Gain updated to = 1.22 mG/bit
Heading angle = 51.45 Degree
Heading angle = 51.45 Degree
Heading angle = 51.58 Degree
Heading angle = 51.55 Degree
Heading angle = 51.56 Degree
Heading angle = 51.39 Degree
Heading angle = 51.58 Degree
Heading angle = 51.55 Degree
Heading angle = 51.56 Degree
Heading angle = 51.41 Degree
Heading angle = 51.64 Degree
Heading angle = 51.70 Degree
Heading angle = 51.62 Degree
Heading angle = 51.62 Degree
Heading angle = 51.50 Degree
Heading angle = 51.61 Degree
Heading angle = 51.56 Degree
Heading angle = 51.67 Degree
Heading angle = 51.52 Degree
Heading angle = 51.75 Degree
Heading angle = 51.42 Degree
Heading angle = 51.53 Degree
Heading angle = 51.50 Degree
Heading angle = 51.47 Degree
Heading angle = 51.42 Degree
Heading angle = 51.55 Degree
Heading angle = 51.58 Degree

```

Εικόνα 49. Εκτύπωση αποτελεσμάτων debugging του HMC5883L στη σειριακή οθόνη .

Ακολούθως θα αναπτύξουμε τα απαραίτητα αρχεία που συναποτελούν την συνολική διαδικασία του προγράμματος, τα οποία ονομαστικά αναφέραμε στην αρχή του άρθρου [Ref 26].

Listing 12. Compass.h Report c++ language technical details

```
/*
-----
MSC HEP 2013-2014
-----
*/
/*
-----
Project      : Quadcopter
File         : Compass_HMC5883L.ino
Description   : This code test HMC5883L magnetometer sensor
Author       : Monahopoulos Konstantinos
-----
*/
/*
-----
COMPASS CLASS DECLARATION
-----
*/
#ifndef compass.h
#define compass.h

class COMPASS_CLASS{

public:

    COMPASS_CLASS(void); //class constructor
    void compass_init(int gain);
    void compass_scaled_reading();
    void compass_heading();

    float bearing;
    float compass_x_scaled;
    float compass_y_scaled;
    float compass_z_scaled;

    float compass_x_offset, compass_y_offset, compass_z_offset;
    float compass_x_gainError,compass_y_gainError,compass_z_gainError;

private:

    float compass_gain_fact;
    int compass_x,compass_y,compass_z;
    void compass_read_XYZdata();

};

#endif
```

Listing 13. Compass.cpp Report c++ language technical details

```
/*
-----
MSC HEP 2013-2014
-----

*/
/*
-----
Project      : Quadcopter
File        : Compass_HMC5883L.ino
Description   : This code test HMC5883L magnetometer sensor
Author       : Monahopoulos Konstantinos
-----


*/
/*
-----
Includes
-----
*/
#include <Arduino.h>
#include <Wire.h>
#include "compass.h"

/*
-----
DEFINITIONS
-----
*/
#define compass_address 0x1E      // The I2C address of the Magnetometer
#define compass_XY_excitation 1160 // The magnetic field excitation in X and
Y direction during Self Test (Calibration)
#define compass_Z_excitation 1080 // The magnetic field excitation in Z
direction during Self Test (Calibration)
#define compass_rad2degree 57.3
#define compass_cal_x_offset 116    // Manually calculated offset in X
direction
#define compass_cal_y_offset 225    // Manually calculated offset in Y
direction
#define compass_cal_x_gain 1.1     // Stored Gain offset at room temperature
#define compass_cal_y_gain 1.12    // Stored Gain offset at room temperature

/*
-----
Function Prototyping
-----
*/
// -----
//     Constructor function
// -----
COMPASS_CLASS::COMPASS_CLASS(void){

compass_x_offset, compass_y_offset, compass_z_offset=0;
compass_x_scaled,compass_y_scaled,compass_z_scaled=0;
compass_x_gainError,compass_y_gainError,compass_z_gainError=1;
compass_x,compass_y,compass_z=0;
```

```

compass_gain_fact=1;
bearing=0;
}

// -----
//      HMC5883L initialization
// -----
// This Function updates the gain_fact variable

void COMPASS_CLASS::compass_init(int gain){

    byte gain_reg,mode_reg;
    Wire.beginTransmission(compass_address);
    Wire.write(0x01);

    if (gain == 0){
        gain_reg = 0b00000000;
        compass_gain_fact = 0.73;
    }
    else if (gain == 1){
        gain_reg = 0b00100000;
        compass_gain_fact= 0.92;
    }
    else if (gain == 2){
        gain_reg = 0b01000000;
        compass_gain_fact= 1.22;
    }
    else if (gain == 3){
        gain_reg = 0b01100000;
        compass_gain_fact= 1.52;
    }
    else if (gain == 4){
        gain_reg = 0b10000000;
        compass_gain_fact= 2.27;
    }
    else if (gain == 5){
        gain_reg = 0b10100000;
        compass_gain_fact= 2.56;
    }
    else if (gain == 6){
        gain_reg = 0b11000000;
        compass_gain_fact= 3.03;
    }
    else if (gain == 7){
        gain_reg = 0b11100000;
        compass_gain_fact= 4.35;
    }

    Wire.write(gain_reg); // bit configuration = g2 g1 g0 0 0 0 0 0, g2 g1 g0
= 0 0 1 for 1.3 guass and 0 1 0 for 1.9 Guass
    Wire.write(0b00000011); // Putting the Magnetometer in idle
    // Writing the register value 0000 0000 for continous mode
    // Writing the register value 0000 0001 for single
    // Writing the register value 0000 0011 for Idle
    Wire.endTransmission();

}

// -----
//      HMC5883L update values
// -----
// This function updates the Global X Y Z Variables

void COMPASS_CLASS::compass_read_XYZdata(){

    Wire.beginTransmission(compass_address);
    Wire.write(0x02);
}

```

```

Wire.write(0b10000001);
// Writing the register value 0000 0000 for continuous mode
// Writing the register value 0000 0001 for single
Wire.endTransmission();
Wire.requestFrom(compass_address, 6);

if (6 <= Wire.available()){

    compass_x = Wire.read()<<8 | Wire.read();

    compass_z = Wire.read()<<8 | Wire.read();

    compass_y = Wire.read()<<8 | Wire.read();

}

void COMPASS_CLASS::compass_scaled_reading(){

    compass_read_XYZdata();

    compass_x_scaled=compass_x*compass_gain_fact*compass_x_gainError+compass_x_offset;

    compass_y_scaled=compass_y*compass_gain_fact*compass_y_gainError+compass_y_offset;

    compass_z_scaled=compass_z*compass_gain_fact*compass_z_gainError+compass_z_offset;

}

// -----
//      HMC5883L compute degrees
// -----

void COMPASS_CLASS::compass_heading(){
    compass_scaled_reading();

    if (compass_y_scaled>0){
        bearing = 90-
atan(compass_x_scaled/compass_y_scaled)*compass_rad2degree;
    }else if (compass_y_scaled<0){
        bearing = 270-
atan(compass_x_scaled/compass_y_scaled)*compass_rad2degree;
    }else if (compass_y_scaled==0 & compass_x_scaled<0){
        bearing = 180;
    }else{
        bearing = 0;
    }
}

```

Listing 14. Compass HMC5883L.ino Report c++ language technical details

```
/*
-----
MSC HEP 2013-2014
-----

*/
/*
-----
Project      : Quadcopter
File        : Compass_HMC5883L.ino
Description   : This code test HMC5883L magnetometer sensor
Author       : Monahopoulos Konstantinos
-----


*/
// Structure:
// -----
// Arduino    | -----
// UNO         |- 3.3 V -----| MPU 6050
//             |- GND -----| HMC5883,
//             |- SDA -----| MS5611
//             |- SCL -----|
//             |
//-----|-----|-----|-----|-----|-----|-----|
//             |-----|-----|-----|-----|-----|-----|
//             V
//             Integrated IMU sensor

/*
-----
Includes
-----


*/
#include <Wire.h>
#include "compass.h"

/*
-----
DEFINITIONS
-----


*/
#define Task_t 10           // Task Time in milli seconds

/*
-----
Assign Classes
-----


*/
COMPASS_CLASS COMPASSobj;

/*
-----
Main Code
-----


*/

```

```

void setup() {
    Serial.begin(9600);

    /*
    -----
    HMC5883L Configuration
    -----
    */

    Wire.begin();

    // -----
    // Precalibration Values
    // -----
    COMPASSObj.compass_x_offset = 122.17;
    COMPASSObj.compass_y_offset = 230.08;
    COMPASSObj.compass_z_offset = 389.85;

    COMPASSObj.compass_x_gainError = 1.12;
    COMPASSObj.compass_y_gainError = 1.13;
    COMPASSObj.compass_z_gainError = 1.03;

    // -----
    // HMC5883L Initialization
    // -----
    COMPASSObj.compass_init(2);
}

void loop() {

    COMPASSObj.compass_scaled_reading(); /* Read Scalled Vaues */
    COMPASSObj.compass_heading(); /* Compute Angles */

    /*Print the results*/
    Serial.print ("Heading angle = ");
    Serial.print (COMPASSObj.bearing);
    Serial.println(" Degree");
}

```

3.8.3.2 ΠΡΟ – ΒΑΘΜΟΝΟΜΗΣΗ ΤΟΥ HMC5883L

Για να υπολογιστεί σωστά το μαγνητικό διάνυσμα αναφοράς θα πρέπει να πραγματοποιηθεί πρώτα η διαδικασία του *calibration*. Ο σημαντικότερος λόγος για την εξαγωγή των *offset* παραμέτρων είναι ο επηρεασμός που γίνεται στο μαγνητόμετρο από τα ηλεκτρονικά εξαρτήματα που το περικλείουν. Ο αισθητήρας είναι πολύ ευαίσθητος στις μετρήσεις του καθώς τα ηλεκτρομαγνητικά πεδία που μας επηρεάζουν είναι πολλά, συμπεριλαμβανομένου του πεδίου που δημιουργεί το *Sensor Board*. Είναι απαραίτητο μετά από οποιαδήποτε αλλαγή του υλικού, ακόμα αν αλλάξουμε θέση του αισθητήρα να επανα - πραγματοποιήσουμε τη διαδικασία του *calibration* και να ανανεώσουμε τις *offset* τιμές που εξάγονται. Τον αλγόριθμο του *calibration* τον έχουμε ενσωματώσει στο αρχείο *Compass_HMC5883L_Calibration.ino* το οποίο και θα

εξηγήσουμε παρακάτω. Επιπλέον στον αλγόριθμο έχουμε συμπεριλάβει και λειτουργία τύπου *debug* η οποία χρησιμοποιείται για την εξαγωγή των απαραίτητων πληροφοριών στο χρήστη μέσω της σειριακής θύρας του μικροελεγκτή.

Η αρχικοποίηση του αισθητήρα γίνεται με τον ίδιο τρόπο που εξηγήσαμε και παραπάνω μέσω της συνάρτησης `compass_init(int gain)`. Το μεγαλύτερο ενδιαφέρον σε αυτή τη φάση εντοπίζεται στη συνάρτηση `compass_offset_calibration(int select)` η οποία αποτελεί και τη κύρια συνάρτηση του αλγορίθμου. Η συνάρτηση `compass_offset_calibration(int select)` εκτελεί δύο κύριες λειτουργίες. Υπολογίζει τη διαφορά στην ενίσχυση του κάθε άξονα του μαγνητικού πεδίου, χρησιμοποιώντας μία συνάρτηση αυτοδιέγερσης του *HMC5883L* και δεύτερον υπολογίζει τη μέση τιμή των μετρήσεων στο κάθε άξονα, όταν το μαγνητόμετρο περιστρέφεται 360 μοίρες. Φυσικά μπορούμε να εκτελέσουμε τις λειτουργίες είτε ξεχωριστά, είτε και τις δύο μαζί στο ίδιο *sketch*. Η διαδικασία απαιτεί περιστροφή του μαγνητόμετρου γύρο από τον άξονά του. Και εκτελώντας την εξάγονται τα παρακάτω αποτελέσματα. Να σημειωθεί ότι τα αποτελέσματα αυτά δεν είναι τα τελικά καθώς δεν έχουμε φτάσει ακόμα στη τελική μορφή του *Quadcopter*. Οι *offset* τιμές που ενσωματώνουμε στο κύριο κώδικα μπορεί να αλλάξουν αναλόγως τις εξαγόμενες τιμές του *calibration*. Έχοντας ενεργοποιήσει τη *debug* λειτουργία, θέτοντας τη μεταβλητή `compass_debug = 1` εξάγονται τα παρακάτω αποτελέσματα έτσι ώστε να αντιληφθούμε τη λειτουργία του προγράμματος.

```
Gain updated to = 1.22 mG/bit
Calibrating the Magnetometer ..... Gain
x_gain_offset = 0.99
y_gain_offset = 1.01
z_gain_offset = 0.99
Calibrating the Magnetometer ..... Offset
Please rotate the magnetometer 2 or 3 times in complete circules with in one
minute .....
Starting Debug data in
3
2
1
0
*****
Debug -- (Offset Calibration)
*****
-339.04      -357.77      -165.18
-336.63      -356.54      -168.80
-337.83      -359.00      -167.59
-337.83      -355.32      -166.39
...
...
```

```

...
60.11 340.56 -194.12
86.56 328.27 -196.53
113.01 318.43 -200.15

*****
End Debug -- (Offset Calibration)
*****
Offset x = 27.65 mG
Offset y = 108.19 mG
Offset z = 185.68 mG

```

Τέλος θα επισυνάψουμε του κώδικες που χρησιμοποιήσαμε για τη παραπάνω διαδικασία. Η ονοματολογία των αρχείων είναι ίδια με παραπάνω, παρόλα αυτά τα αρχεία εντοπίζονται σε διαφορετικό φάκελο και έχουν διαφορετικό σκοπό [Ref 26].

Listing 15. Compass_calibration.h Report c++ language technical details

```

/*
-----
MSC HEP 2013-2014
-----
*/
/*
-----
Project      : Quadcopter
File         : Compass_HMC5883L_Calibration.ino
Description   : This code calibrates HMC5883L magnetometer sensor
Author       : Monahopoulos Konstantinos
-----
*/
/*
-----
COMPASS CLASS DECLARATION
-----
*/
#ifndef compass.h
#define compass.h

class COMPASS_CLASS{

public:

    COMPASS_CLASS(void); //class constructor
    void compass_offset_calibration(int select);
    void compass_init(int gain);

    float bearing;
    float compass_x_scaled;
    float compass_y_scaled;
    float compass_z_scaled;

    float compass_x_offset, compass_y_offset, compass_z_offset;
    float compass_x_gainError, compass_y_gainError, compass_z_gainError;
}

```

```

    int compass_debug;

private:

    float compass_gain_fact;
    int compass_x,compass_y,compass_z;
    void compass_read_XYZdata();

};

#endif

```

Listing 16. Compass_calibration.cpp Report c language technical details

```

/*
-----
----- MSc HEP 2013-2014 -----
-----

*/
/*
-----
----- Project      : Quadcopter
File        : Compass_HMC5883L_Calibration.ino
Description  : This code calibrates HMC5883L magnetometer sensor
Author       : Monahopoulos Konstantinos
-----
*/

/*
-----
----- Includes
-----
*/
#include <Arduino.h>
#include <Wire.h>
#include "compass.h"

/*
-----
----- DEFINITIONS
-----
*/
#define compass_address 0x1E          // The I2C address of the Magnetometer
#define compass_XY_excitation 1160 // The magnetic field excitation in X and
Y direction during Self Test (Calibration)
#define compass_Z_excitation 1080 // The magnetic field excitation in Z
direction during Self Test (Calibration)
#define compass_rad2degree 57.3

#define compass_cal_x_offset 116 // Manually calculated offset in X
direction
#define compass_cal_y_offset 225 // Manually calculated offset in Y
direction
#define compass_cal_x_gain 1.1 // Stored Gain offset at room temperature
#define compass_cal_y_gain 1.12 // Stored Gain offset at room temperature

```

```

/*
-----
Function Prototyping
-----
*/
// -----
//     Constructor function
// -----
COMPASS_CLASS::COMPASS_CLASS(void) {

compass_x_offset, compass_y_offset, compass_z_offset=0;
compass_x_scaled,compass_y_scaled,compass_z_scaled=0;
compass_x_gainError,compass_y_gainError,compass_z_gainError=1;
compass_x,compass_y,compass_z=0;

compass_gain_fact=1;
bearing=0;
compass_debug=0;
}

// -----
//     HMC5883L initialization
// -----
// This Function updates the gain_fact variable

void COMPASS_CLASS::compass_init(int gain){

byte gain_reg,mode_reg;
Wire.beginTransmission(compass_address);
Wire.write(0x01);

if (gain == 0){
    gain_reg = 0b00000000;
    compass_gain_fact = 0.73;
}
else if (gain == 1){
    gain_reg = 0b00100000;
    compass_gain_fact= 0.92;
}
else if (gain == 2){
    gain_reg = 0b01000000;
    compass_gain_fact= 1.22;
}
else if (gain == 3){
    gain_reg = 0b01100000;
    compass_gain_fact= 1.52;
}
else if (gain == 4){
    gain_reg = 0b10000000;
    compass_gain_fact= 2.27;
}
else if (gain == 5){
    gain_reg = 0b10100000;
    compass_gain_fact= 2.56;
}
else if (gain == 6){
    gain_reg = 0b11000000;
    compass_gain_fact= 3.03;
}
else if (gain == 7){
    gain_reg = 0b11100000;
    compass_gain_fact= 4.35;
}
}

```

```

    Wire.write(gain_reg); // bit configuration = g2 g1 g0 0 0 0 0 0, g2 g1 g0
= 0 0 1 for 1.3 guass and 0 1 0 for 1.9 Guass
    Wire.write(0b00000011); // Putting the Magnetometer in idle
    // Writing the register value 0000 0000 for continous mode
    // Writing the register value 0000 0001 for single
    // Writing the register value 0000 0011 for Idle
    Wire.endTransmission();

    Serial.print("Gain updated to = ");
    Serial.print(compass_gain_fact);
    Serial.println(" mG/bit");

}

// -----
//      HMC5883L Calibration
// -----
// This Function calculates the offset in the Magnetometer
// using Positive and Negative bias Self test capability
// This function updates X_offset Y_offset and Z_offset Global variables
// Call Initialize before

void COMPASS_CLASS::compass_offset_calibration(int select){
    // ****
    // offset_calibration() function performs two tasks
    // 1. It calculates the diffrence in the gain of the each axis
    magnetometer axis, using
    //      inbuilt self excitation function of HMC5883L (Which is useless if it
    is used as a compass
    //      unless you are very unlucy and got a crappy sensor or live at very
    High or low temperature)
    // 2. It calculates the mean of each axes magnetic field, when the
    Magnetometer is rotated 360 degree
    // 3. Do Both
    // ****

    // -----
    //      Gain offset estimation
    // -----
    if (select == 1 | select == 3){ // User input in the function
    // Configuring the Control register for Positive Bias mode
    Serial.println("Calibrating the Magnetometer ..... Gain");
    Wire.beginTransmission(compass_address);
    Wire.write(0x00);
    Wire.write(0b01110001); // bit configuration = 0 A A DO2 DO1 D00 MS1 MS2

    /*
     A A                               DO2  DO1  D00       Sample Rate [Hz]      MS1  MS0
Measurment Mode
     0 0 = No Average                 0   0   0   =   0.75                      0   0
= Normal
     0 1 = 2 Sample average           0   0   1   =   1.5                       0   1
= Positive Bias
     1 0 = 4 Sample Average          0   1   0   =   3                         1   0
= Negative Bias
     1 1 = 8 Sample Average          0   1   1   =   7.5                      1   1
= -
                           1   0   0   =   15 (Default)
                           1   0   1   =   30
                           1   1   0   =   75
                           1   1   1   =   -
    */
    Wire.endTransmission();

    compass_read_XYZdata(); // Disregarding the first data

    // Reading the Positive baised Data
}

```

```

        while(compass_x<200 | compass_y<200 | compass_z<200){ // Making sure the
data is with Positive baised
            compass_read_XYZdata();
        }

        compass_x_scaled=compass_x*compass_gain_fact;
        compass_y_scaled=compass_y*compass_gain_fact;
        compass_z_scaled=compass_z*compass_gain_fact;

        // Offset = 1160 - Data_positive
        compass_x_gainError = (float)compass_XY_excitation/compass_x_scaled;
        compass_y_gainError = (float)compass_XY_excitation/compass_y_scaled;
        compass_z_gainError = (float)compass_Z_excitation/compass_z_scaled;

        // Configuring the Control register for Negative Bais mode
        Wire.beginTransmission(compass_address);
        Wire.write(0x00);
        Wire.write(0b01110010); // bit configuration = 0 A A DO2 DO1 DO0 MS1 MS2

        /*
         *          DO2 DO1 DO0      Sample Rate [Hz]      MS1 MS0
         * Measurment Mode
         * 0 0 = No Average      0  0  0  =  0.75      0  0
         * = Normal
         * 0 1 = 2 Sample average 0  0  1  =  1.5       0  1
         * = Positive Bias
         * 1 0 = 4 Sample Average 0  1  0  =  3        1  0
         * = Negative Bais
         * 1 1 = 8 Sample Average 0  1  1  =  7.5      1  1
         * = -
         *                      1  0  0  =  15 (Default)
         *                      1  0  1  =  30
         *                      1  1  0  =  75
         *                      1  1  1  =  -
         */
        Wire.endTransmission();

        compass_read_XYZdata(); // Disregarding the first data
        // Reading the Negative baised Data
        while(compass_x>-200 | compass_y>-200 | compass_z>-200){ // Making sure
the data is with negative baised
            compass_read_XYZdata();
        }

        compass_x_scaled=compass_x*compass_gain_fact;
        compass_y_scaled=compass_y*compass_gain_fact;
        compass_z_scaled=compass_z*compass_gain_fact;

        // Taking the average of the offsets
        compass_x_gainError =
        (float)((compass_XY_excitation/abs(compass_x_scaled))+compass_x_gainError)/
        2;
        compass_y_gainError =
        (float)((compass_XY_excitation/abs(compass_y_scaled))+compass_y_gainError)/
        2;
        compass_z_gainError =
        (float)((compass_Z_excitation/abs(compass_z_scaled))+compass_z_gainError)/2
        ;

        Serial.print("x_gain_offset = ");
        Serial.println(compass_x_gainError);
        Serial.print("y_gain_offset = ");
        Serial.println(compass_y_gainError);
        Serial.print("z_gain_offset = ");
        Serial.println(compass_z_gainError);
    }
}

```

```

// Configuring the Control register for normal mode
Wire.beginTransmission(compass_address);
Wire.write(0x00);
Wire.write(0b01111000); // bit configuration = 0 A A DO2 DO1 DO0 MS1 MS2

/*
A A
Measurement Mode          DO2 DO1 DO0      Sample Rate [Hz]      MS1 MS0
0 0 = No Average          0  0  0   =   0.75           0  0
= Normal
0 1 = 2 Sample average    0  0  1   =   1.5            0  1
= Positive Bias
1 0 = 4 Sample Average    0  1  0   =   3              1  0
= Negative Bias
1 1 = 8 Sample Average    0  1  1   =   7.5           1  1
= -
                           1  0  0   =   15 (Default)
                           1  0  1   =   30
                           1  1  0   =   75
                           1  1  1   =   -
*/
Wire.endTransmission();

// -----
//      Offset estimation
// -----
if (select == 2 | select == 3){// User input in the function
Serial.println("Calibrating the Magnetometer ..... Offset");
Serial.println("Please rotate the magnetometer 2 or 3 times in complete
circles with in one minute ..... ");

for(byte i=0;i<10;i++){ // Disregarding first few data
    compass_read_XYZdata();
}

float x_max=-4000,y_max=-4000,z_max=-4000;
float x_min=4000,y_min=4000,z_min=4000;

// -----
//      Debug code
// -----
if (compass_debug == 1){
    Serial.println("Starting Debug data in ");
    delay(1000);
    Serial.println("3");
    delay(1000);
    Serial.println("2");
    delay(1000);
    Serial.println("1");
    delay(1000);
    Serial.println("0");
    Serial.println();
    for(byte i=0;i<10;i++){
        Serial.print("*");
    }
    Serial.println("*");
    Serial.println("Debug -- (Offset Calibration)");
    for(byte i=0;i<10;i++){
        Serial.print("*");
    }
    Serial.println("*");
}
// End Debug code

unsigned long t = millis();
while(millis()-t <= 30000) {

```

```

compass_read_XYZdata();

compass_x_scalled=(float)compass_x*compass_gain_fact*compass_x_gainError;
compass_y_scalled=(float)compass_y*compass_gain_fact*compass_y_gainError;
compass_z_scalled=(float)compass_z*compass_gain_fact*compass_z_gainError;

if (compass_debug == 1){ //----- Debug Data
    Serial.print(compass_x_scalled);
    Serial.print("\t");
    Serial.print(compass_y_scalled);
    Serial.print("\t");
    Serial.println(compass_z_scalled);
} //----- End Debug Data

x_max = max(x_max,compass_x_scalled);
y_max = max(y_max,compass_y_scalled);
z_max = max(z_max,compass_z_scalled);

x_min = min(x_min,compass_x_scalled);
y_min = min(y_min,compass_y_scalled);
z_min = min(z_min,compass_z_scalled);
}

/*
Debug code -----
*/
if (compass_debug == 1){
    Serial.println();
    for(byte i=0;i<10;i++){
        Serial.print("*");
    }
    Serial.println("*");
    Serial.println("End Debug -- (Offset Calibration)");
    for(byte i=0;i<10;i++){
        Serial.print("*");
    }
    Serial.println("*");
}
// End Debug code

compass_x_offset = ((x_max-x_min)/2)-x_max;
compass_y_offset = ((y_max-y_min)/2)-y_max;
compass_z_offset = ((z_max-z_min)/2)-z_max;

Serial.print("Offset x = ");
Serial.print(compass_x_offset);
Serial.println(" mG");
Serial.print("Offset y = ");
Serial.print(compass_y_offset);
Serial.println(" mG");
Serial.print("Offset z = ");
Serial.print(compass_z_offset);
Serial.println(" mG");

}

}

// -----
//      HMC5883L update values
// -----
// This function updates the Global X Y Z Variables

void COMPASS_CLASS::compass_read_XYZdata(){

```

```

Wire.beginTransmission(compass_address);
Wire.write(0x02);
Wire.write(0b10000001);
// Writing the register value 0000 0000 for continous mode
// Writing the register value 0000 0001 for single
Wire.endTransmission();
Wire.requestFrom(compass_address, 6);

if (6 <= Wire.available()){

    compass_x = Wire.read()<<8 | Wire.read();
    compass_z = Wire.read()<<8 | Wire.read();
    compass_y = Wire.read()<<8 | Wire.read();
}

}

```

Listing 17. Compass_HMC5883L_Calibration.ino Report c++ language technical details

```

/*
-----
MSC HEP 2013-2014
-----

*/
/*
Project      : Quadcopter
File         : Compass_HMC5883L_Calibration.ino
Description   : This code calibrates HMC5883L magnetometer sensor
Author       : Monahopoulos Konstantinos
-----

*/
// Structure:
// -----
// Arduino  |
//  UNO      |- 3.3 V -----| MPU 6050      |
//           |- GND -----| HMC5883,      |
//           |- SDA -----| MS5611       |
//           |- SCL -----|               |
//           |
//-----          | _____ |
//           V           Integrated IMU sensor
//-----


/*
----- Includes -----
*/
#include <Wire.h>
#include "compass.h"

/*
This will calibrate HMC5883L.

Run this Algorithm to calibrate HMC5883L magnetometer. During the
calibration

```

you must rotate the quadcopter 2-3 times in under a minute and write down the GAIN offset's values and the Calibration Offset. Let the program finish execution.

```
/*
 *
-----  

    Assign Classes  

-----  

*/  

COMPASS_CLASS COMPASSobj;  

/*  

-----  

    Global Variables  

-----  

*/  

int dt=0;  

unsigned long t;  

/*  

-----  

    Main Code  

-----  

*/  

void setup(){  

    Serial.begin(9600);  

    Wire.begin();  

    COMPASSobj.compass_x_offset = 122.17;  

    COMPASSobj.compass_y_offset = 230.08;  

    COMPASSobj.compass_z_offset = 389.85;  

    COMPASSobj.compass_x_gainError = 1.12;  

    COMPASSobj.compass_y_gainError = 1.13;  

    COMPASSobj.compass_z_gainError = 1.03;  

    COMPASSobj.compass_init(2);  

    COMPASSobj.compass_debug = 1;  

    COMPASSobj.compass_offset_calibration(3);  

}  

void loop(){  

    // Nothing to do here ...  

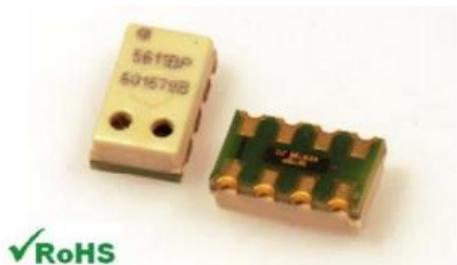
}
```

3.8.4 BAPOMETPO (MS5611)

To *MS5611-01BA* είναι ένας αισθητήρας υψομέτρου υψηλής ανάλυσης νέας γενιάς από την *MEAS Switzerland* με διεπιφάνειες *SPI* και *I2C*. Είναι βελτιστοποιημένος για υψομετρικές συσκευές και για βαρόμετρα. Έχει δυνατότητα ανάλυσης της τάξης των

10 εκατοστών. Ο αισθητήρας συμπεριλαμβάνει έναν αισθητήρα πίεσης υψηλής γραμμικότητας και έναν 24bit ΔΣ ADC πολύ χαμηλής κατανάλωσης. Έτσι παρέχονται ακριβείς ψηφιακές τιμές 24bit για την πίεση και την θερμοκρασία. Επίσης παρέχεται η δυνατότητα στο χρήστη μέσω διαφορετικών mode λειτουργίας να βελτιστοποιεί το ρυθμό μετατροπής και την κατανάλωση ρεύματος. Η παροχή τιμών υψηλής ανάλυσης για την θερμοκρασία επιτρέπει την εναλλαγή μεταξύ θερμόμετρου και υψομετρικού αισθητήρα χωρίς κάποιον επιπλέον αισθητήρα. Ο MS5611-01BA μπορεί να συνδεθεί σχεδόν με κάθε μικροελεγκτή.

Το πρωτόκολλο επικοινωνίας είναι απλό , χωρίς να χρειάζεται να προγραμματιστούν εσωτερικοί καταχωρητές στην συσκευή. Οι μικρές διαστάσεις της συσκευής (5mm x 3mm x 1.7mm) επιτρέπουν την ενσωμάτωση σε κινητές συσκευές. Αυτή η γενιά νέων αισθητήρων βασίζεται στην τεχνολογία MEAS και στην MEAS Switzerland.



Εικόνα 50. Ο αισθητήρας βαρομετρικής πίεσης MS5611-01BA .

Το τελευταίο component που ολοκληρώνει τον 10DOFIMU είναι το βαρόμετρο MS5611 το οποίο και μας παράγει τον δέκατο βαθμό ελευθερίας. Τέτοιου είδους αισθητήρες χρησιμοποιούνται κατά κόρον στα Quadcopter για να είναι σε θέση να διατηρούν το ύψος τους κατά την πτήση. Εφόσον μας δίνεται η δυνατότητα να συμπεριλάβουμε ένα τέτοιου είδους αισθητήρα στο Quadcopter που κατασκευάζουμε και μέσω των τιμών που εξάγονται από το βαρόμετρο μπορούμε να επηρεάσουμε τη ταχύτητα περιστροφής των μοτέρ. Πιο συγκεκριμένα οι τιμές αυτές, όπως και όλες οι προηγούμενες θα εισαχθούν σε ένα σύστημα PID το οποίο ανάλογα με την ενίσχυση σε κάθε βαθμίδα του , επιδρά καταλυτικά στη διαδικασία ισορροπίας του Quadcopter. Παρακάτω θα αναπτύξουμε τον κώδικα που εφαρμόσαμε στον μικροελεγκτή μέσω του οποίου πραγματοποιούμε ανάγνωση των τιμών του αισθητήρα, εκτυπώνοντας τα αποτελέσματα μέσω της σειριακής θύρας.

3.8.4.1 ΥΠΟΛΟΓΙΣΜΟΣ ΥΨΟΥΣ ΜΕ ΧΡΗΣΗ ΤΟΥ ARDUINO UNO

Αρχικά η συνδεσμολογία που πρέπει να πραγματοποιήσουμε για την σωστή εγκαθίδρυση των αισθητήρων στο *Arduino* ακολουθεί την εξής λογική. Θα πρέπει να συνδέσουμε την αναλογική είσοδο του *Arduino* (*pin Analog input 4*) που χρησιμοποιεί το *I2C Bus* στο *SDA* της συσκευής. Ανάλογα συνδέσουμε την αναλογική είσοδο του *Arduino* (*pin Analog input 5*) στο *SCL* της συσκευής. Τέλος συνδέουμε ως τάση εισόδου στον αισθητήρα την τάση εξόδου 3.3 volt του *Arduino* και την γείωση του μικροελεγκτή στο *GND* του *10DOF*. Πρέπει να σημειωθεί ότι είναι προτιμότερο να τροφοδοτούμε τον αισθητήρα με χαμηλή τάση (3.3v) παρά με 5v, αλλά για να πραγματοποιηθεί αυτό θα πρέπει να απενεργοποιήσουμε τις ενσωματωμένες *pull – up* αντιστάσεις του *Arduino*. Αυτό γίνεται αρχικά πηγαίνοντας στις βιβλιοθήκες του *Arduino* στο φάκελο *Arduino\libraries\Wire\utility* και εντοπίζοντας το αρχείο *twi.c* ανοίγοντας το με κάποιον *editor*. Και στη συνέχεια εντοπίζουμε τις γραμμές *digitalWrite(SDA, 1);* και *digitalWrite(SCL, 1);* και τις σχολιάζουμε. Με αυτό τον τρόπο έχουμε απενεργοποιήσει σε κάθε sketch που αναπτύσσουμε τις *pull – up* αντιστάσεις στα *pin A4* και *A5* του μικροελεγκτή.

Για την σωστή λειτουργία του προγράμματος πρέπει να συμπεριλάβουμε στο *project* το αρχείο *<Wire.h>* το οποίο και κάνει τη διαχείριση της συσκευής που έχει τοποθετηθεί στο διάδρομο *I2C* του *Arduino*. Επίσης το *MS5611* θα πρέπει να έχει ελεγχθεί ότι εντοπίζεται στη διεύθυνση *0x77* όπως ορίζεται στον εντοπισμό των συσκευών μέσω του *Scanner_port_report* που δείξαμε παραπάνω. Τα αρχεία που έχουν δημιουργηθεί για τον έλεγχο του επιτανχυσιομέτρου και του γυροσκοπίου είναι τα *MS5611.cpp* και *MS5611.h*. Σε αυτό το σημείο θα περιγράψουμε κομμάτια του κώδικα που αφορούν την εκτέλεση του *MS5611* και μόνο. Το *project* που αναλαμβάνει την εκτέλεση και εμφάνιση των αποτελεσμάτων είναι το *MS5611_Control.ino*.

Για την καλύτερη δόμηση του προγράμματος και για μεγαλύτερη ευκολία στη χρήση του, κατασκευάσαμε μία δομή στο αρχείο *MS5611.h*. Ο ορισμός της κλάσης γίνεται μέσω της δήλωσής της ως *class MS5611_CLASS*. Επιπροσθέτως έχουμε ορίσει μια *enum* (*enumeration*) δομή την οποία χρησιμοποιούμε κατάλληλα παρακάτω για να αντιληφθούμε το μοντέλο του του βαρόμετρου που χρησιμοποιούμε και αναλόγως να

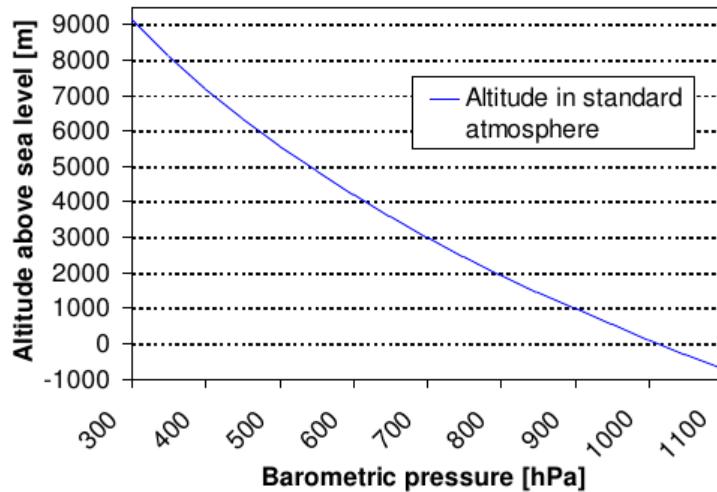
ορίσουμε τους χρόνους καθυστέρησης στο πρόγραμμα. Στη περίπτωση του *MS5611* που χρησιμοποιούμε, ο αισθητήρας αποτελεί ένα *high resolution* ολοκληρωμένο σύστημα, το οποίο συνεπάγει ότι το *delay time* (*ct*) παίρνει τη τιμή 5. Ακολούθως έχουμε αναπτύξει ένα σετ από συναρτήσεις οι οποίες με την ορθή χρήση τους εξάγουν τα επιθυμητά αποτελέσματα.

Πιο αναλυτικά οι συναρτήσεις `ReadTemperature(bool compensation = false)` και `ReadPressure(bool compensation = false)` μπορούν να διαβάσουν τις τιμές θερμοκρασίας και πίεσης χρησιμοποιώντας τις αντίστοιχες συναρτήσεις `ReadRawTemperature(void)` και `ReadRawPressure(void)`. Εφόσον έχουμε υπολογίσει τις τιμές αυτές καλώντας τη συνάρτηση `getAltitude(double pressure, double seaLevelPressure = 101325)` με *default* όρισμα εισόδου τη πίεση της θάλασσας. Η τιμή αυτή είναι σταθερή και με τον τρόπο δήλωσης της συνάρτησης μπορούμε να την χρησιμοποιήσουμε με δύο τρόπους.

Ο πρώτος τρόπος υπολογίζει την απόλυτη υψομετρική τιμή (σε σχέση με τη στάθμη της θάλασσας) και ο δεύτερος την σχετική υψομετρική τιμή σε σχέση με τη προηγούμενη τιμή και θέση του αισθητήρα. Γνωρίζοντας την επιφανειακή πίεση αναφοράς της θάλασσας (1013.25 *hPa*) και μετρώντας την πίεση στο σημείο που βρίσκεται ο αισθητήρας, μπορούμε να υπολογίσουμε τα υψόμετρα μέσω του τύπου :

$$\text{altitude} = 44330 * \left(1 - \left(\frac{p}{p_0} \right)^{\frac{1}{5.255}} \right)$$

Η σχέση μεταξύ υψομέτρου και πίεσης παρουσιάζεται στη παρακάτω χαρακτηριστική, η οποία δείχνει ότι κάθε αλλαγή στην πίεση της τάξης των $\Delta p = 1 \text{ hPa}$ αντιστοιχεί με διαφορά στο ύψος της τάξης $\Delta h = 8.43 \text{ m}$. Αντίστροφα, αλλαγή στη συνιστώσα του ύψους της τάξης $\Delta h = 10 \text{ m}$ αντιστοιχεί σε αλλαγή της πίεσης $\Delta p = 1.2 \text{ hPa}$.



Εικόνα 51. Σχέση βαρομετρικής πίεσης - υψομέτρου .

Επιπροσθέτως, μέσω των συναρτήσεων υπάρχει η δυνατότητα υπολογισμού των τιμών *offset* για θερμοκρασίες μικρότερες των 20°C και -15°C , πραγματοποιώντας τη διαδικασία αυτή μέσω της μεταβλητής *TEMP*. Παρακάτω εμφανίζουμε τα κομμάτια κώδικα που συναποτελούν τη κύρια λειτουργία του καθώς επίσης και τα αποτελέσματα που εξάγουμε μέσω της σειριακής θύρας του μικροελεγκτή [Ref 27] .

Listing 18. MS5611.h Report c++ language technical details

```
/*
-----
MSC HEP 2013-2014
-----
*/
/*
-----
Project      : Quadcopter
File         : MS5611_Control.ino
Description   : This code test MS5611 sensor
Author       : Monahopoulos Konstantinos
-----
*/
/*
-----
Includes
-----
*/
#ifndef MS5611_h
```

```

#define MS5611_h

#include <Arduino.h>

// -----
//      Enumeration MS5611
// -----
typedef enum
{
    MS5611_ULTRA_HIGH_RES    = 0x08,
    MS5611_HIGH_RES          = 0x06,
    MS5611_STANDARD           = 0x04,
    MS5611_LOW_POWER          = 0x02,
    MS5611_ULTRA_LOW_POWER    = 0x00
} MS5611_enum;

/*
-----
----- MS5611 CLASS DECLARATION -----
-----

*/
class MS5611_CLASS
{
public:
    MS5611_CLASS(void);      //class constructor

    bool Init(MS5611_enum osr = MS5611_HIGH_RES);
    uint32_t ReadRawTemperature(void);
    uint32_t ReadRawPressure(void);
    double ReadTemperature(bool compensation = false);
    int32_t ReadPressure(bool compensation = false);
    double getAltitude(double pressure, double seaLevelPressure = 101325);
    double getSeaLevel(double pressure, double altitude);
    void setOversampling(MS5611_enum osr);

    double realTemperature;
    long realPressure;
    float absoluteAltitude;
    float relativeAltitude;
    double referencePressure;

private:
    uint16_t fc[6];
    uint8_t ct;
    uint8_t uosr;
    int32_t TEMP2;
    int64_t OFF2, SENS2;

    uint32_t D1;
    uint32_t D2;
    int32_t dT;
    int64_t OFF;
    int64_t SENS;
    int32_t TEMP;
    uint8_t vha_16;
    uint8_t vla_16;
    uint8_t vxa_24;
    uint8_t vha_24;
    uint8_t vla_24;

    void reset(void);
    void readPROM(void);
}

```

```

        uint16_t readRegister16(uint8_t reg);
        uint32_t readRegister24(uint8_t reg);
};

#endif

```

Listing 19. MS5611.cpp Report c++ language technical details

```

/*
-----
MSC HEP 2013-2014
-----
*/
/*
-----
Project      : Quadcopter
File         : MS5611_Control.ino
Description   : This code test MS5611 sensor
Author       : Monahopoulos Konstantinos
-----
*/
/*
-----
Includes
-----
*/
#define <Arduino.h>
#define <Wire.h>
#define <math.h>
#define "MS5611.h"

/*
-----
DEFINITIONS
-----
*/
#define MS5611_ADDRESS          (0x77)
#define MS5611_CMD_ADC_READ     (0x00)
#define MS5611_CMD_RESET         (0x1E)
#define MS5611_CMD_CONV_D1       (0x40)
#define MS5611_CMD_CONV_D2       (0x50)
#define MS5611_CMD_READ_PROM    (0xA2)
#define MS5611_PROM_REG_COUNT   6

/*
-----
Function Prototyping
-----
*/
// -----
// Constructor function

```

```

// -----
MS5611_CLASS::MS5611_CLASS(void) {

    realTemperature = 0;
    realPressure = 0;
    absoluteAltitude = 0;
    relativeAltitude = 0;
    referencePressure = 0;

    D1 = 0;
    D2 = 0;
    dT = 0;
    OFF = 0;
    SENS = 0;
}

// -----
//      MS5611 Initialization
// -----
bool MS5611_CLASS::Init(MS5611_enum osr)
{
    Serial.println("Initialize MS5611 Sensor");
    Wire.begin();
    reset();
    setOversampling(osr);

    delay(100);
    readPROM();

    return false;
}

// -----
//      MS5611 Reset
// -----
void MS5611_CLASS::reset(void)
{
    Wire.beginTransmission(MS5611_ADDRESS);
    Wire.write(MS5611_CMD_RESET);
    Wire.endTransmission();
}

// -----
//      Set oversampling MS5611 value
// -----
void MS5611_CLASS::setOversampling(MS5611_enum osr)
{
    switch (osr)
    {
        case MS5611_ULTRA_LOW_POWER:
            ct = 1;
            break;
        case MS5611_LOW_POWER:
            ct = 2;
            break;
        case MS5611_STANDARD:
            ct = 3;
            break;
        case MS5611_HIGH_RES:
            ct = 5;
            break;
        case MS5611_ULTRA_HIGH_RES:
            ct = 10;
            break;
    }
}

```

```

        }
        uosr = osr;
    }

// -----
//      Read Data From MS5611
// -----
void MS5611_CLASS::readPROM(void)
{
    for (uint8_t offset = 0; offset < 6; offset++)
    {
        fc[offset] = readRegister16(MS5611_CMD_READ_PROM + (offset * 2));
    }
}

// -----
//      Read Real Pressure From MS5611
// -----
int32_t MS5611_CLASS::ReadPressure(bool compensation)
{
    D1 = ReadRawPressure();

    D2 = ReadRawTemperature();
    dT = D2 - (uint32_t)fc[4] * 256;

    OFF = (int64_t)fc[1] * 65536 + (int64_t)fc[3] * dT / 128;
    SENS = (int64_t)fc[0] * 32768 + (int64_t)fc[2] * dT / 256;

    if (compensation)
    {
        TEMP = 2000 + ((int64_t) dT * fc[5]) / 8388608;

        OFF2 = 0;
        SENS2 = 0;

        if (TEMP < 2000)
        {
            OFF2 = 5 * ((TEMP - 2000) * (TEMP - 2000)) / 2;
            SENS2 = 5 * ((TEMP - 2000) * (TEMP - 2000)) / 4;
        }

        if (TEMP < -1500)
        {
            OFF2 = OFF2 + 7 * ((TEMP + 1500) * (TEMP + 1500));
            SENS2 = SENS2 + 11 * ((TEMP + 1500) * (TEMP + 1500)) / 2;
        }
    }

    OFF = OFF - OFF2;
    SENS = SENS - SENS2;
}

return (D1 * SENS / 2097152 - OFF) / 32768;
}

// -----
//      Read Pressure Raw Data From MS5611
// -----
uint32_t MS5611_CLASS::ReadRawPressure(void)
{
    Wire.beginTransmission(MS5611_ADDRESS);
    Wire.write(MS5611_CMD_CONV_D1 + uosr);
    Wire.endTransmission();
    delay(ct);

    return readRegister24(MS5611_CMD_ADC_READ);
}

```

```

}

// -----
//      Read Real Temperature From MS5611
// -----
double MS5611_CLASS::ReadTemperature(bool compensation)
{
    D2 = ReadRawTemperature();
    dT = D2 - (uint32_t)fc[4] * 256;

    TEMP = 2000 + ((int64_t) dT * fc[5]) / 8388608;

    TEMP2 = 0;

    if (compensation)
    {
        if (TEMP < 2000)
        {
            TEMP2 = (dT * dT) / (2 << 30);
        }
    }

    TEMP = TEMP - TEMP2;

    return ((double)TEMP/100);
}

// -----
//      Read Temperature Raw Data From MS5611
// -----
uint32_t MS5611_CLASS::ReadRawTemperature(void)
{
    Wire.beginTransmission(MS5611_ADDRESS);
    Wire.write(MS5611_CMD_CONV_D2 + uosr);
    Wire.endTransmission();
    delay(ct);

    return readRegister24(MS5611_CMD_ADC_READ);
}

// -----
//      Altitude Calculation Function
// -----
double MS5611_CLASS::getAltitude(double pressure, double seaLevelPressure)
{
    // Calculate altitude from Pressure & Sea level pressure
    return (44330.0f * (1.0f - pow((double)pressure /
(double)seaLevelPressure, 0.1902949f)));
}

// -----
//      Sea Level Calculation Function
// -----
double MS5611_CLASS::getSeaLevel(double pressure, double altitude)
{
    // Calculate sea level from Pressure given on specific altitude
    return ((double)pressure / pow(1.0f - ((double)altitude / 44330.0f),
5.255f));
}

// -----
//      Read 16-bit Function
// -----
// Read 16-bit from register (oops MSB, LSB)

```

```

uint16_t MS5611_CLASS::readRegister16(uint8_t reg)
{
    Wire.beginTransmission(MS5611_ADDRESS);
    Wire.write(reg);
    Wire.endTransmission();

    Wire.beginTransmission(MS5611_ADDRESS);
    Wire.requestFrom(MS5611_ADDRESS, 2);

    while(!Wire.available()) {};
    vha_16 = Wire.read();
    vla_16 = Wire.read();

    Wire.endTransmission();

    return (vha_16 << 8 | vla_16);
}

// -----
//      Read 24-bit Function
// -----
// Read 24-bit from register (oops XSB, MSB, LSB)
uint32_t MS5611_CLASS::readRegister24(uint8_t reg)
{
    Wire.beginTransmission(MS5611_ADDRESS);
    Wire.write(reg);
    Wire.endTransmission();

    Wire.beginTransmission(MS5611_ADDRESS);
    Wire.requestFrom(MS5611_ADDRESS, 3);

    while(!Wire.available()) {};
    vxa_24 = Wire.read();
    vha_24 = Wire.read();
    vla_24 = Wire.read();

    Wire.endTransmission();

    return ((int32_t)vxa_24 << 16) | ((int32_t)vha_24 << 8) | vla_24;
}

```

Ο αλγόριθμος για την βασική εκτέλεση του προγράμματος εμπεριέχεται στο αρχείο `MS5611_Control.ino` το οποίο και παρουσιάζουμε παρακάτω :

Listing 20. `MS5611_Control.ino` Report c++ language technical details

```

/*
-----
MSC HEP 2013-2014
-----
*/
/*
-----
Project      : Quadcopter
File        : MS5611_Control.ino
Description  : This code test MS5611 sensor
Author       : Monahopoulos Konstantinos
-----
```

```

/*
// Structure:
// -----
// Arduino | -----
// UNO      |- 3.3 V -----| MPU 6050   |
//           |- GND -----| HMC5883,   |
//           |- SDA -----| MS5611    |
//           |- SCL -----|
//           |
//-----|-----|-----|-----|-----|-----|-----|
//           |-----|-----|-----|-----|-----|-----|
//           V             Integrated IMU sensor
*/

/*
-----|-----|-----|-----|-----|-----|-----|
Includes
-----|-----|-----|-----|-----|-----|-----|
*/
#include <Wire.h>
#include "MS5611.h"
/*
-----|-----|-----|-----|-----|-----|-----|
Assign Classes
-----|-----|-----|-----|-----|-----|-----|
*/
MS5611_CLASS MS5611_obj;

/*
-----|-----|-----|-----|-----|-----|-----|
Main Code
-----|-----|-----|-----|-----|-----|-----|
*/
void setup()
{
  Serial.begin(9600);

// -----
// MS5611 Initialization
// -----
// MS5611_obj.Init();
if (MS5611_obj.Init()==0) {
  while(1){
    Serial.println(" False on initialization of MS5611 ");
    delay(500);
  }
}

// Get reference pressure for relative altitude
  MS5611_obj.referencePressure = MS5611_obj.ReadPressure();
}

void loop()
{
  // Calculate Temperature
}

```

```

MS5611_obj.realTemperature = MS5611_obj.ReadTemperature();
// Calculate Pressure
MS5611_obj.realPressure = MS5611_obj.ReadPressure();
// Calculate absolute altitude
MS5611_obj.absoluteAltitude =
MS5611_obj.getAltitude(MS5611_obj.realPressure);
// Calculate relative altitude
MS5611_obj.relativeAltitude =
MS5611_obj.getAltitude(MS5611_obj.realPressure,
MS5611_obj.referencePressure);

Serial.print("RealTemp = ");
Serial.print(MS5611_obj.realTemperature);
Serial.print(" *C");
Serial.print("\t");

Serial.print("RealPressure = ");
Serial.print(MS5611_obj.realPressure);
Serial.print(" Pa");
Serial.print("\t");

Serial.print("AbsoluteAltitude = ");
Serial.print(MS5611_obj.absoluteAltitude);
Serial.print(" m");
Serial.print("\t");

Serial.print("RelativeAltitude = ");
Serial.print(MS5611_obj.relativeAltitude);
Serial.print(" m");
Serial.println("\t");

delay(1000);
}

```

Εκτελόντας τον παραπάνω κώδικα μπορούμε να εποπτεύσουμε τις τιμές θερμοκρασίας, πίεσης, απόλυτη υψομετρική τιμή και σχετική υψομετρική τιμή σε μέτρα. Παρακάτω επισυνάπτουμε τα αποτελέσματα εξήγηθησαν μέσω της σειριακής.

| RealTemp | RealPressure | AbsoluteAltitude | RelativeAltitude |
|----------|--------------|------------------|------------------|
| 23.78 *C | 102053 Pa | -60.43 m | 0.00 m |
| 23.76 *C | 102053 Pa | -60.43 m | 0.00 m |
| 23.76 *C | 102054 Pa | -60.52 m | -0.08 m |
| 23.76 *C | 102052 Pa | -60.35 m | 0.08 m |
| 23.76 *C | 102054 Pa | -60.52 m | -0.08 m |
| 23.77 *C | 102055 Pa | -60.60 m | -0.17 m |
| 23.77 *C | 102056 Pa | -60.68 m | -0.25 m |
| 23.78 *C | 102053 Pa | -60.43 m | 0.00 m |
| 23.78 *C | 102056 Pa | -60.68 m | -0.25 m |
| 23.78 *C | 102050 Pa | -60.19 m | 0.25 m |
| 23.78 *C | 102055 Pa | -60.60 m | -0.17 m |
| 23.78 *C | 102053 Pa | -60.43 m | 0.00 m |
| 23.78 *C | 102054 Pa | -60.52 m | -0.08 m |
| 23.78 *C | 102052 Pa | -60.35 m | 0.08 m |
| 23.79 *C | 102052 Pa | -60.35 m | 0.08 m |
| 23.79 *C | 102052 Pa | -60.35 m | 0.08 m |

Εικόνα 52. Εκτύπωση αποτελεσμάτων debugging του MS5611 στη σειριακή οθόνη .

ΚΕΦΑΛΑΙΟ 4: ΑΝΑΠΤΥΞΗ ΕΛΕΓΚΤΗ ΙΣΣΟΡΟΠΙΑΣ

4.1 ΘΕΩΡΙΑ ΕΛΕΓΚΤΗ PID

Ένας *proportional-integral-derivative* ελεγκτής (*PID controller*) είναι ένας ελεγκτής που χρησιμοποιείται ευρέως στα βιομηχανικά συστήματα ελέγχου. Ο *PID* ελεγκτής υπολογίζει μία τιμή σφάλματος που αντιστοιχεί στη διαφορά μεταξύ της μετρούμενης τιμής και της επιθυμητής τιμής. Ο ελεγκτής επιχειρεί να ελαχιστοποιήσει το σφάλμα τροποποιώντας τη διαδικασία ελέγχου μέσω ελέγχου της τιμής μιας μεταβλητής.

Ο αλγόριθμος του ελεγκτή *PID* αφορά 3 διαφορετικές σταθερές και ονομάζεται έλεγχος 3 όρων. Ο αναλογικός (*proportional*), ο ολοκληρωτικός (*integral*) και ο παραγοντικός (*derivative*) όρος. Η λειτουργία των όρων αυτών επεξηγείται ως εξής: Ο *P* εξαρτάται από το σφάλμα εκείνη τη δεδομένη στιγμή. Ο *I* από το ολοκλήρωμα παρελθόντων τιμών σφάλματος και ο *D* από την πρόβλεψη μελλοντικών τιμών με βάση υπολογισμού την παράγωγο. Το σταθμισμένο άθροισμα όλων αυτών χρησιμοποιείται να τη ρύθμιση ενός στοιχείου ελέγχου όπως το άνοιγμα μιας βαλβίδας ή η ισχύς που παρέχεται σε ένα θερμαντικό στοιχείο

Με το συντονισμό των 3 παραμέντρων στον αλγόριθμο του *PID*, ο ελεγκτής μπορεί να ανταποκριθεί στις εκάστοτε απαιτήσεις. Η απόκριση του ελεγκτή μπορεί να εκτιμηθεί με την δυνατότητα ανταπόκρισης στο εμφανιζόμενο σφάλμα, με το βαθμό στον οποίο ο ελεγκτής υπερβαίνει την επιθυμητή τιμή εξόδου και το μέγεθος της ταλάντωσης του συστήματος (*system oscillation*). Να σημειωθεί εδώ ότι ο *PID* ελεγκτής δεν εγγυάται βέλτιστο έλεγχο του συστήματος ούτε σταθερότητα συστήματος.

Μερικές εφαρμογές μπορεί να χρειάζονται μόνο έναν ή δύο από τους τρεις ορους. Αυτό επιτυγχάνεται θέτοντας τον όρο που δεν χρησιμοποιείται στην τιμή μηδέν. Ο *PID* ελεγκτής θα ονομάζεται *PI*, *PD*, *P* ή *I* ελεγκτής με την έλλειψη των αντίστοιχων συντελεστών.

Ο *PID* έλεγχος πήρε το όνομα του από τους 3 όρους. Οι όροι αναλογικός, ολοκληρωτικός, παραγωγικός αθροίζονται για να υπολογιστεί η έξοδος του ελεγκτή. Αν υποθέσουμε ότι $u(t)$ είναι η έξοδος του ελεγκτή η τελική μορφή του *PID* αλγορίθμου είναι:

$$u(t) = MV(t) = K_p e(t) + K_i \int_0^t e(\tau) d\tau + K_d \frac{d}{dt} e(t)$$

Όπου:

K_p : Proportional gain, a tuning parameter

K_i : Integral gain, a tuning parameter

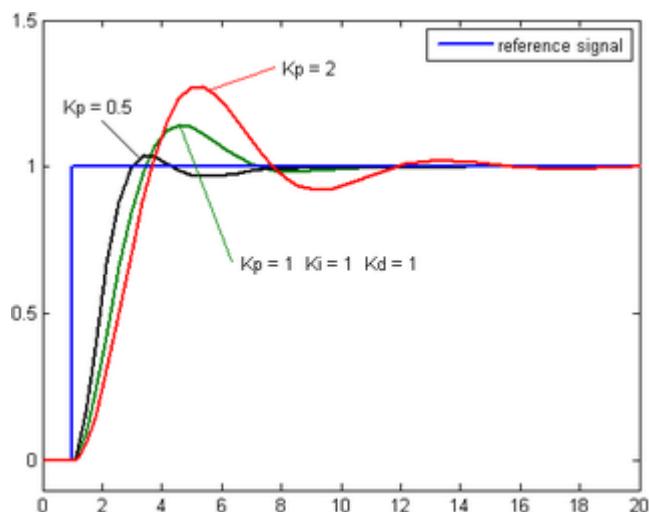
K_d : Derivative gain, a tuning parameter

e : Error = $SP - PV$

t : Time or instantaneous time (the present)

τ : Variable of integration; takes on values from time 0 to the present t .

4.1.1 ΑΝΑΛΟΓΙΚΟΣ ΟΡΟΣ (PROPORTIONAL)



Εικόνα 53. Απόκριση σήματος PID ελεγκτή με χρήση μόνο του αναλογικού όρου .

Ο αναλογικός όρος παράγει ένα αποτέλεσμα ανάλογο με την τιμή του τρέχοντος σφάλματος. Η απόκριση μπορεί να ρυθμιστεί πολλαπλασιάζοντας το σφάλμα με την τιμή K_p , που ονομάζεται σταθερά αναλογικού κέρδους .

Ο αναλογικός όρος δίνεται από :

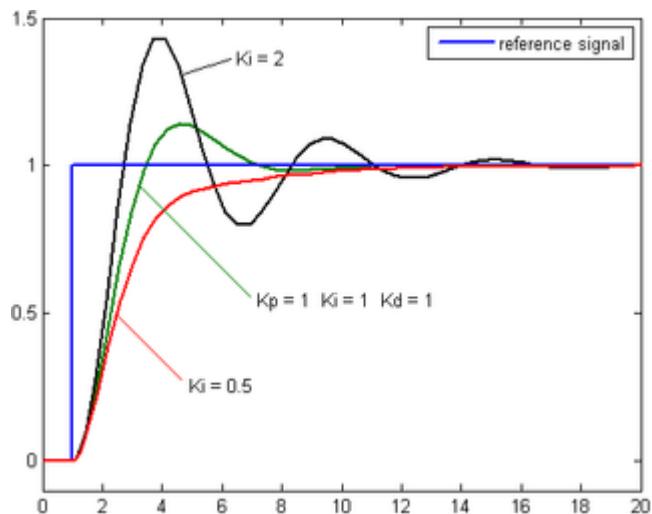
$$P_{out} = K_p e(t)$$

Μία μεγάλη τιμή για το αναλογικό κέρδος οδηγεί σε μεγάλη μεταβολή της εξόδου για μία δεδομένη αλλαγή στο σφάλμα. Αν το αναλογικό κέρδος είναι πολύ μεγάλο το σύστημα μπορεί να γίνει ασταθές. Επίσης μία μικρή τιμή για το αναλογικό κέρδος οδηγεί σε μικρή μεταβολή της εξόδου για μία δεδομένη αλλαγή στο σφάλμα(πχ την εμφάνιση μεγάλης τιμής σφάλματος). Αν το αναλογικό κέρδος είναι πολύ μικρό το σύστημα μπορεί να μην ανταποκρίνεται επαρκώς στις αναταράξεις. Η πράξη έχει αποδείξει ότι ο αναλογικός όρος πρέπει να έχει το μεγαλύτερο μέρος στην παραγωγή της εξόδου.

4.1.1.1 DROOP

Επειδή απαιτείται μη μηδενικό σφάλμα για να οδηγήσει τον αναλογικό ελεγκτή, χρησιμοποιείται ένα σφάλμα σταθερής κατάστασης ονομαζόμενο ως *droop*. Το *droop* είναι ανάλογο προς το *process gain* και αντιστρόφως ανάλογο προς το proportional gain. Το *droop* μπορεί να μετριαστεί προσθέτοντας έναν όρο στην ζητούμενη τιμή εξόδου ή με δυναμική διόρθωση με τη χρήση ενός ολοκληρωτικού όρου που είναι επίσης γνωστός ως *Offset*.

4.1.2 ΟΛΟΚΛΗΡΩΤΙΚΟΣ ΟΡΟΣ (INTEGRAL)



Εικόνα 54. Απόκριση σήματος PID ελεγκτή με χρήση αναλογικού - ολοκληρωτικού όρου .

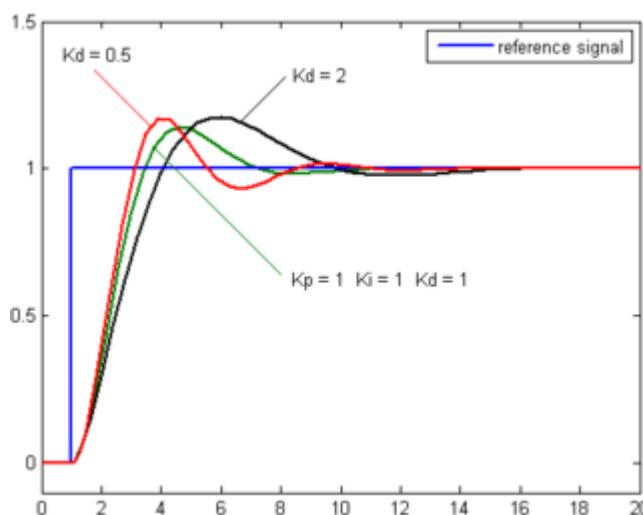
Η συνεισφορά του ολοκληρωτικού όρου είναι ανάλογη προς το μέγεθος της τιμής του σφάλματος. Είναι επίσης ανάλογη προς το χρόνο που το σφάλμα διατηρεί αυτήν την τιμή.

Το ολοκλήρωμα για έναν *PID* ελεγκτή είναι το άθροισμα των στοιχειωδών σφαλμάτων προς το χρόνο. Το αποτέλεσμα που δίνει είναι η συσσωρευτική απόκλιση (*offset*) που έπρεπε ήδη να είχε διορθωθεί. Το συσσωρευτικό σφάλμα πολλαπλασιάζεται με τον συντελεστή K_i και προστίθεται στην έξοδο του ελεγκτή. Ο ολοκληρωτικός όρος δίνεται από :

$$I_{\text{out}} = K_i \int_0^t e(\tau) d\tau$$

Ο ολοκληρωτικός όρος επιταχύνει την κίνηση προς την επιθυμητή τιμή για την έξοδο του συστήματος. Επίσης εξαλείφει το σφάλμα σταθερής κατάστασης. Παρόλα αυτά ο ολοκληρωτικός όρος αποκρίνεται στα συσσωρευμένα σφάλματα παρελθόντων τιμών. Έτσι μπορεί να οδηγήσει την τρέχουσα τιμή να υπερβεί την επιθυμητή.

4.1.3 ΠΑΡΑΓΩΓΙΚΟΣ ΟΡΟΣ (DERIVATIVE)



Εικόνα 55. Απόκριση σήματος *PID* ελεγκτή με χρήση αναλογικού - ολοκληρωτικού – παραγωγικού όρου

Η παράγωγος του σφάλματος της διεργασίας ορίζεται από την κλήση του σφάλματος σε σχέση με το χρόνο και με το πολλαπλασιασμό αυτόν του μεγέθους με το παραγωγικό κέρδος K_d .

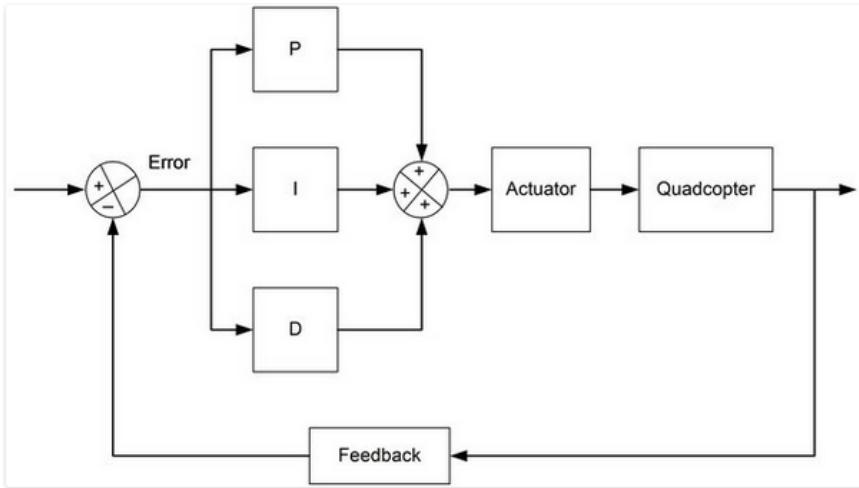
Ο παραγωγικός όρος δίνεται από :

$$D_{\text{out}} = K_d \frac{d}{dt} e(t)$$

Ο παραγωγικός ελεγκτής προβλέπει τη συμπεριφορά του συστήματος και βελτιώνει το χρόνο απόκρισης και τη σταθερότητα του συστήματος. Μία ιδανική παράγωγος είναι μη αιτιατή (not causal) έτσι ώστε να συμπεριλαμβάνεται ένα κατωδιαβατό φίλτρο για τον παραγωγικό όρο. Έτσι περιορίζεται ο θόρυβος και το κέρδος για τις υψηλές συγχονότητες.

4.2 ΕΠΙΡΡΟΗ PID ΣΤΟ QUADCOPTER

Όπως ήδη αναλύθηκε ο *PID* ελεγκτής (*proportional-integral-derivative controller*) είναι ένας μηχανισμός ανάδρασης για τον βρόχο ελέγχου. Ένας *PID* ελεγκτής υπολογίζει το «σφάλμα» ως τη διαφορά μεταξύ τρέχουσας τιμής και επιθυμητής. Ο ελεγκτής επιχειρεί να ελαχιστοποιήσει το σφάλμα. Όσον αφορά το quadcopter τα παραπάνω σημαίνουν ότι ο ελεγκτής λαμβάνει δεδομένα από τους αισθητήρες (*gyros / accelerometers* κτλ) και μετά από σύγκριση με τις επιθυμητές τιμές τροποποιεί κατάλληλα την ταχύτητα των μοτέρ ώστε να αντισταθμιστούν οι διαφορές και να διατηρηθεί ισορροπία. Ο *PID* έλεγχος αφορά τρεις διαφορετικές παραμέτρους. Ανάλογα με τον ελεγκτή πτήσης θα υπάρχουν *PID* ελεγκτές συσχετισμένοι με διαφόρους τύπους πτήσης



Εικόνα 56. Περιγραφή πλήρες συστήματος PID ελεγκτή .

Επιρροή του παράγοντα P

Το P είναι η κύρια μεταβλητή που συνεισφέρει στο σωστό έλεγχο του συστήματος. Μπορούμε να αφήσουμε τις τιμές I και D μηδενικές και το quadcopter να αιωρείται χωρίς προβλήματα. Όσο πιο μεγάλη τιμή έχει το P τόσο περισσότερο γίνεται διόρθωση στην τιμή ελέγχου. Αν το P είναι πολύ μεγάλο το quadcopter γίνεται πολύ ευαίσθητο και υπερδιορθώνει με αποτέλεσμα να υπάρχουν ταλαντώσεις υψηλής συχνότητας.

Παρόλα αυτά δεν προέρχονται όλες οι ταλαντώσεις από το συντελεστή P. Χρειάζεται να εξαλειφθούν οι ταλαντώσεις από άλλες πηγές (πλαίσιο, μοτέρ) όσο περισσότερο είναι εφικτό πριν αρχίσει η διαδικασία ρύθμισης .

Επιρροή του παράγοντα I

Έστω ότι έχουμε ρυθμίσει το κέρδος P σωστά. Αφήνοντας τα I και D στην τιμή 0. Κατα την πτήση μπορεί να παρατηρήσει κανείς ότι χρειάζεται να είναι πατημένος ο έλεγχος *pitch/roll* ώστε να κινηθεί το όχημα. Μόλις αφεθεί ο μοχλός ή το κουμπί το quadcopter επανέρχεται στη θέση ισορροπίας. Προσθέτοντας κέρδος I παρατηρούμε ότι η πτήση είναι πιο ομαλή. Άλλα όταν αφεθεί ο μοχλός ελέγχου το quadcopter συνεχίζει να κινείται. Για να γίνει αυτό πιο κατανοητό, το quadcopter ολοκληρώνει το σφάλμα σε βάθος χρόνου . Όταν ο συντελεστής I είναι μη μηδενικός υπάρχει πάντα ένα μικρό σφάλμα

Σε κάποιες περιπτώσεις το I χρησιμοποιείται για πιο ομαλή πτήση ενώ σε κάποιες άλλες για να επιτευχθεί διαρκής κίνηση. Παρόλα αυτά με πολύ ψηλή τιμή I το quadcopter θα αρχίσει να ταλαντώνεται σε χαμηλή συχνότητα και μπορεί να ακόμη να κάνει και απότομες μανούβρες.

Επιρροή του παράγοντα D

Ο συντελεστής D δεν χρησιμοποείται ιδιαίτερα. Μπορεί μάλιστα να παραληφθεί και το quadcopter θα συνεχίσει να πετάει. Ο συντελεστής D συμπεριφέρεται σαν ελατήριο που απορροφά τους κραδασμούς. Με την προσθήκη D κέρδους η κίνηση του quadcopter γίνεται με λιγότερους κραδασμούς και αναταράξεις. Παρόλα αυτά μεγάλη τιμή D θα δημιουργήσει ταλαντώσεις.

Πιο συγκεκριμένα ο D αλλάζει την δύναμη που εφαρμόζεται για τη διόρθωση εσφαλμένης θέσης. Όταν προσεγγίζεται η επιθυμητή τιμή η εφαρμοζόμενη δύναμη ελαττώνεται. Γι' αυτό παρατηρούμε μια πιο «μαλακή» κίνηση.

Η ρύθμιση των τιμών PID αφορά την εύρεση ισορροπίας μεταξύ της στιγμιαίας αδράνειας και της ώθησης που δημιουργούν τα μοτερ. Η στιγμιαία αδράνεια σχετίζεται με το βάρος του quadcopter αλλά και με την κατανομή του βάρους.

Στο quadcopter οι τιμές αυτές δεν μπορούν να αποτυπωθούν από άλλο ήδη έτοιμο μοντέλο, γι' αυτό το λόγο θα πρέπει να τις ρυθμίσουμε εμείς βρίσκοντας σταδιακά τις ενισχύσεις των βαθμίδων K_i , K_p , K_d και μέσω δοκιμαστικών πτήσεων να βρούμε σταδιακά τις κατάλληλες.

4.3 ΥΛΟΠΟΙΗΣΗ PID ΜΕ ΧΡΗΣΗ ΤΟΥ ARDUINO UNO

Για την σωστή λειτουργία του *PID controller* δημιουργήσαμε δύο αρχεία που συμπεριλαμβάνουμε στο γενικό project τα οποία είναι τα `pid.h` και `pid.cpp`. Τα αρχεία αυτά παρέχουν τις απαραίτητες μεταβλητές και συναρτήσεις και θα μας βοηθήσουν να υπολογίσουμε τα σφάλματα μέσω του *PID Controller*. Στο αρχείο `pid.h` εντοπίζουμε την δήλωση δύο κλάσεων και πιο συγκεκριμένα τη κλάση `PID_INFO` και `PID_EXEC`. Η πρώτη κλάση καλείται κατά την αρχικοποίηση του

προγράμματος και περιέχει όλες εκείνες τις πληροφορίες που χρειάζεται ο PID controller. Κατά τη κλήση της κλάσης αυτής πραγματοποιείται δήλωση των τιμών του P, του I και του D για όλους τους βαθμούς ελευθερίας που κατέχει το Quadcopter, όπως επίσης και την επιθυμητή γωνία κλίσης κατά την εκκίνηση.

Η κλάση `PID_EXEC` περιέχει όλες τις συναρτήσεις γιατ τον υπολογισμό της εξόδου του ελενγκτή η οποία αργότερα θα προστεθεί ή αφαιρεθεί ανάλογα, στη κινητήρια δύναμη των τεσσάρων μοτέρ. Κάποιες από τις συναρτήσεις που εντοπίζονται εντός της κλάσης είναι οι `SetMode(int Mode)` η οποία θέτει σε λειτουργία τον PID ελενγκτή, η `SetOutputLimits(double, double)` την οποία χρησιμοποιούμε για να ορίσουμε τη μέγιστη έξοδο σφάλματος κατά την εκτέλεση, η `SetTunings(double, double, double)` η οποία δέχεται σαν όρισμα τις ενισχύσεις των βαθμίδων του PID, η `SetControllerDirection(int)` της οποίας το όρισμα εισόδου είναι 0 ή 1 και θέτει το πρόσημο του σφάλματος και η `SetSampleTime(int)` στην οποία δηλώνεται ο ρυθμός εκτέλεσης του PID. Εκτός από τις συμπληρωματικές συναρτήσεις τις οποίες έχουμε ενσωματώσει προς εξαγωγή πληροφοριών για την κατάσταση του PID, στη κλάση αυτή έχουμε ενσωματώσει και τη βασική συνάρτηση `Compute()` η οποία είναι και η κύρια συνάρτηση εκτέλεσης σε κάθε βρόγχο επανάληψης που υπολογίζει κάθε στιγμή το σφάλμα.

Στο αρχείο `pid.cpp` κατά την κλήση της κλάσης του `PID_EXEC` ορίζουμε τα όρια εξόδου του σφάλματος σε (-200, 200) και το ρυθμό εκτέλεσης στα 60ms. Η συνάρτηση `Compute()` είναι η βασική συνάρτηση υπολογισμού των μεταβλητών εξόδου. Η συνάρτηση αυτή δε δέχεται σαν όρισμα κάποιες μεταβλητές αλλα αντιθέτως λειτουργεί υπολογίζοντας το σφάλμα και αποθηκευοντάς το σε διευθύνσεις μνήμης τις οποίες έχουμε δηλώσει κατά την αρχικοποίηση. Πιο συγκεκριμένα ο `Constructor` της κλάσης δέχεται σαν όρισμα τις διευθύνσεις των τιμών εισόδου, εξόδου και επιθυμητής γωνίας οι οποίες ανανεώνονται σε κάθε επανάληψη. Με τον τρόπο αυτό δε χρειάζεται να εισάγουμε μεταβλητές ως ορίσματα αλλά μόνο να ανανεώνουμε το περιεχόμενο των διευθύνσεων που αντιπροσωπεύουν τις τιμές αυτές.

Στη κύρια λειτουργία, εντοπίζεται η διαφορά της επιθυμητής τιμής με τη πραγματική και ύστερα υπολογίζεται το γενικό σφάλμα (`error`), το αθροιστικό σφάλμα (`ITerm`) όλων των προηγούμενων σφαλμάτων, όπως επίσης υπολογίζεται και η παράγωγος του σφάλματος (`dInput`), η οποία είναι η διαφορά του τωρινού και του προηγούμενου

σφάλματος. Τέλος, έχοντας υπολογίσει τις παραπάνω παραμέτρους λύνουμε την συνάρτηση του *PID controller* υπολογίζοντας την έξοδο και αποθηκεύοντάς την στη μεταβλητή *output*. Ο τύπος υπολογισμού του *PID controller* όπως ήδη αναφέραμε είναι ο παρακάτω :

$$\text{Output} = K_P e(t) + K_I \int e(t) dt + K_D \frac{d}{dt} e(t)$$

Προφανώς θα πρέπει να δημιουργήσουμε έναν *PID controller* για κάθε βαθμό ελευθερίας διαφορετικής κίνησης που χαρακτηρίζει το quadcopter. Συνεπώς θα πρέπει να δημιουργήσουμε 4 *PID controllers*, έναν για κάθε κίνηση (*PITCH*, *ROLL*, *YAW*, *ALTITUDE*). Παρακάτω παρουσιάζουμε τον κώδικα που περιλαμβάνουν τα αρχεία *pid.h* και *pid.cpp*. και πραγματοποιούν τα παραπάνω προ-απαιτούμενα.

Listing 21. pid.h Report c++ language technical details

```
/*
-----
MSC HEP 2013-2014
-----
*/
/*
-----
Project      : Quadcopter
File         : PID_Control.ino
Description   : This code defines PID process
Author       : Monahopoulos Konstantinos
-----
*/
/*
-----
DEFINITIONS
-----
*/
#ifndef PID_h
#define PID_h

//Constants used in some of the functions below
#define AUTOMATIC 1
#define MANUAL     0
#define DIRECT      0
#define REVERSE    1

/*

```

```

-----  

----- PID CLASS DECLARATION -----  

-----  

*/
class PID_INFO
{
    public:
        PID_INFO(double,double,double);
        double PIDOutput;
        double Setpoint;
        double kp, ki, kd;
};

/*
-----  

----- PID CLASS DECLARATION -----  

-----  

*/
class PID_EXEC
{
    public:
        // -----
        // commonly used functions
        // -----
        PID_EXEC(double*, double*, double*,double, double, double, int); /* constructor. links the PID to the Input, Output, and Setpoint. Initial tuning parameters are also set here*/
        void SetMode(int Mode);                                // * sets PID to either Manual (0) or Auto (non-0)
        bool Compute();                                     // * performs the PID calculation. it should be called every time loop()
        cycles. ON/OFF and                                // calculation frequency can be set using SetMode
        // SetSampleTime respectively
        void SetOutputLimits(double, double); //clamps the output to a specific range. 0-255 by default, but
        //it's likely the user will want to change this depending on
        //the application
        // -----
        // available but not commonly used functions
        // -----
        void SetTunings(double, double, double); // While most users will set the tunings once in the constructor, this function gives the user the option
        // of changing tunings during runtime for Adaptive control

```

```

    void SetControllerDirection(int);      // * Sets the Direction, or
"Action" of the controller. DIRECT          // means the output will increase when
error is positive. REVERSE                 // means the opposite. it's very
unlikely that this will be needed         // once it is set in the constructor.

    void SetSampleTime(int);              // * sets the frequency, in
Milliseconds, with which                  // the PID calculation is
performed. default is 100

// -----
// Display functions
// -----
// These functions
query the pid for interal values.          // they were created
double GetKp();                           // where it's
mainly for the pid front-end,             // inside the PID.
double GetKi();                           // 
important to know what is actually        //
int GetMode();                            //
int GetDirection();                       //
unsigned long lastTime, SampleTime;        //

double kp;                                // * (P)roportional Tuning Parameter
double ki;                                // * (I)ntegral Tuning Parameter
double kd;                                // * (D)erivative Tuning Parameter

private:

void Initialize();                         //
double dispKp;                           // * we'll hold on to the tuning
parameters in user-entered                // format for display purposes
double dispKi;                           //
double dispKd;                           //
int controllerDirection;                  //

double *myInput;                          // * Pointers to the Input, Output, and
Setpoint variables                        // This creates a hard link between the
double *myOutput;                         variables and the
double *mySetpoint;                      // PID, freeing the user from having to
constantly tell us                       // what these values are. with pointers
we'll just know.
double ITerm, lastInput;
double outMin, outMax;
bool inAuto;
};

#endif

```

Listing 22. pid.cpp Report c++ language technical details

```
/*
-----
MSc HEP 2013-2014
-----
*/
/*
-----
Project      : Quadcopter
File        : PID_Control.ino
Description   : This code defines PID process
Author       : Monahopoulos Konstantinos
-----
*/
/*
-----
Includes
-----
*/
/*
-----
Function Prototyping
-----
*/
/*
// -----
//      Information Constructor function
// -----
*/
/*
***** *Constructor (...)*****
*      The parameters specified here have been set by the user.
***** */
PID_INFO::PID_INFO(double Pre_Def_Kp ,double Pre_Def_Ki, double Pre_Def_Kd)
{
    kp = Pre_Def_Kp;
    ki = Pre_Def_Ki;
    kd = Pre_Def_Kd;
    Setpoint = 0; // On start initialization of setpoint
}

/*
// -----
//      Execution Constructor function
// -----
*/
/*
***** *
*      The input parameters here are specified in PID_INFO class .
***** */
/
```

```

PID_EXEC::PID_EXEC(double* Input, double* Output, double* Setpoint,
                   double Kp, double Ki, double Kd, int ControllerDirection)
{
    myOutput = Output;
    myInput = Input;
    mySetpoint = Setpoint;
    inAuto = false;

    PID_EXEC::SetOutputLimits(-200,200); //default
    output limit corresponds to - EXTHES TO EIXA -130,130 //the arduino pwm
limits

    SampleTime = 60; // -(miimum
sample time)- default Controller Sample Time is 0.1 seconds (SampleTime =
100)

    PID_EXEC::SetControllerDirection(ControllerDirection);
    PID_EXEC::SetTunings(Kp, Ki, Kd);

}

// -----
//      Compute function
// -----
*****/*
 *      This, as they say, is where the magic happens.  this function should
be called
 *      every time "void loop()" executes.  the function will decide for itself
whether a new
 *      pid Output needs to be computed.  returns true when the output is
computed,
 *      false when nothing has been done.

*****
bool PID_EXEC::Compute()
{
    if(!inAuto) return false;
    unsigned long now = millis();
    unsigned long timeChange = (now - lastTime);
    if(timeChange>=SampleTime)
    {
        /*Compute all the working error variables*/
        double input = *myInput;
        double error = *mySetpoint - input;
        ITerm+= (ki * error);
        if(ITerm > outMax) ITerm= outMax;
        else if(ITerm < outMin) ITerm= outMin;
        double dInput = (input - lastInput);

        /*Compute PID Output*/
        double output = kp * error + ITerm- kd * dInput;

        if(output > outMax) output = outMax;
        else if(output < outMin) output = outMin;
        *myOutput = output;

        /*Remember some variables for next time*/
        lastInput = input;
        lastTime = now;
        return true;
    }
}

```

```

        }
        else return false;
    }

// -----
//      SetTunings function
// -----
-----

/*********************************************
*
* This function allows the controller's dynamic performance to be adjusted.
* it's called automatically from the constructor, but tunings can also
* be adjusted on the fly during normal operation
*
********************************************

*/
void PID_EXEC::SetTunings(double Kp, double Ki, double Kd)
{
    if (Kp<0 || Ki<0 || Kd<0) return;

    dispKp = Kp; dispKi = Ki; dispKd = Kd;

    double SampleTimeInSec = ((double)SampleTime)/1000;
    kp = Kp;
    ki = Ki * SampleTimeInSec;
    kd = Kd / SampleTimeInSec;

    if(controllerDirection ==REVERSE)
    {
        kp = (0 - kp);
        ki = (0 - ki);
        kd = (0 - kd);
    }
}

// -----
//      SetSampleTime function
// -----
-----

/*********************************************
*
* sets the period, in Milliseconds, at which the calculation is performed
*
********************************************

*/
void PID_EXEC::SetSampleTime(int NewSampleTime)
{
    if (NewSampleTime > 0)
    {
        double ratio = (double)NewSampleTime
                      / (double)SampleTime;
        ki *= ratio;
        kd /= ratio;
        SampleTime = (unsigned long)NewSampleTime;
    }
}

// -----
//      SetOutputLimits function
// -----
-----
```

```

/*
 *      This function will be used far more often than SetInputLimits. while
 *      the input to the controller will generally be in the 0-1023 range (which
 *      is
 *      the default already,) the output will be a little different. maybe
 *      they'll
 *      be doing a time window and will need 0-8000 or something. or maybe
 *      they'll
 *      want to clamp it from 0-125. who knows. at any rate, that can all be
 *      done
 *      here.
 */
void PID_EXEC::SetOutputLimits(double Min, double Max)
{
    if(Min >= Max) return;
    outMin = Min;
    outMax = Max;

    if(inAuto)
    {
        if(*myOutput > outMax) *myOutput = outMax;
        else if(*myOutput < outMin) *myOutput = outMin;

        if(ITerm > outMax) ITerm= outMax;
        else if(ITerm < outMin) ITerm= outMin;
    }
}

// -----
//      SetMode function
// -----
// *****

/*
 *      Allows the controller Mode to be set to manual (0) or Automatic (non-
 *      zero)
 *      when the transition from manual to auto occurs, the controller is
 *      automatically initialized
 */
void PID_EXEC::SetMode(int Mode)
{
    bool newAuto = (Mode == AUTOMATIC);
    if(newAuto == !inAuto)
    { /*we just went from manual to auto*/
        PID_EXEC::Initialize();
    }
    inAuto = newAuto;
}

// -----
//      Initialize function
// -----
// *****

/*
 *      does all the things that need to happen to ensure a bumpless transfer
 *      from manual to automatic mode.
 */
void PID_EXEC::Initialize()
{

```

```

ITerm = *myOutput;
lastInput = *myInput;
if(ITerm > outMax) ITerm = outMax;
else if(ITerm < outMin) ITerm = outMin;
}

// -----
// ----- SetControllerDirection function
// -----
-----

/*********************************************
*
* The PID will either be connected to a DIRECT acting process (+Output
leads
* to +Input) or a REVERSE acting process(+Output leads to -Input.) we need
to
* know which one, because otherwise we may increase the output when we
should
* be decreasing. This is called from the constructor.

*****
*/
void PID_EXEC::SetControllerDirection(int Direction)
{
    if(inAuto && Direction != controllerDirection)
    {
        kp = (0 - kp);
        ki = (0 - ki);
        kd = (0 - kd);
    }
    controllerDirection = Direction;
}

// -----
// ----- Status function
// -----
-----

/*********************************************
*
* Just because you set the Kp=-1 doesn't mean it actually happened. these
* functions query the internal state of the PID. they're here for display
* purposes. this are the functions the PID Front-end uses for example

*****
*/
double PID_EXEC::GetKp(){ return dispKp; }
double PID_EXEC::GetKi(){ return dispKi; }
double PID_EXEC::GetKd(){ return dispKd; }
int PID_EXEC::GetMode(){ return inAuto ? AUTOMATIC : MANUAL; }
int PID_EXEC::GetDirection(){ return controllerDirection; }

```

4.4 ΑΛΛΗΛΕΠΙΔΡΑΣΗ ΜΟΤΕΡ - ΕΛΕΓΚΤΗ

Όπως ήδη αναφέραμε η μεταβλητή `PIDOutput` που εξάγεται από τον *PID Controller*, πρέπει να υπολογιστεί για κάθε κλάση που δημιουργήσαμε. Η μεταβλητή αυτή, όστερα, πρέπει να εισαχθεί σε μία συνάρτηση που θα είναι υπεύθυνη να υπολογίζει την

ταχύτητα του κάθε μοτέρ, έτσι ώστε το quadcopter να καταφέρνει να ισορροπεί. Για το σκοπό αυτό έχουμε δημιουργήσει δύο αρχεία, το Motors.h και το Motors.cpp, εμπεριέχοντας όλη τη διαδικασία υλοποίησης κίνησης των μοτέρ μέσω του PID σφάλματος. Στο αρχείο Motors.h έχουμε δημιουργήσει μία κλάση MOTORS_CLASS στη οποία εκτός από τις public και private μεταβλητές, έχουν δηλωθεί και οι απαραίτητες συναρτήσεις. Πριν προχωρήσουμε στη περιγραφή των αρχείων πρέπει να αναφέρουμε ότι στο *main sketch* έχουμε καλέσει τη κλάση τύπου Servo τέσσερις φορές, με διαφορετική ονοματολογία κάθε φορά αναπαριστώντας τα τέσσερα μοτέρ του quadcopter. Οι κλάσεις αυτές δημιουργούνται με χρήση του παρακάτω κώδικα.

```
Servo myescFL;
Servo myescFR;
Servo myescRL;
Servo myescRR;
```

Τα ακρωνύμια *FR*, *FL*, *RL*, *RR* αντιπροσωπεύουν τα μοτέρ (*Front Left*, *Front Right*, *Rear Left*, *Rear Right*) βοηθώντας μας έτσι να ορίσουμε την επιτάχυνση ή επιβράδυνση στο εκάστοτε μοτέρ. Επιπροσθέτως, μέσω κατάλληλης σύνδεσης των ESC και ακολούθως των μοτέρ στα εκάστοτε pin αρχικοποιούμε τα αντίστοιχα pins με χρήση του παρακάτω κώδικα.

```
myescFL.attach(3); // Attach the pins to esc
myescFR.attach(5); // Attach the pins to esc
myescRL.attach(6); // Attach the pins to esc
myescRR.attach(9); // Attach the pins to esc
```

Με αυτό τον τρόπο έχει γίνει η αρχικοποίηση των μοτέρ και μπορούμε πλέον να συνεχίσουμε στη περιγραφή των συναρτήσεων. Οι βασικές συναρτήσεις για το σκοπό ισορροπίας του *quadcopter* εντοπίζονται στη κλάση PID_CLASS και είναι η συνάρτηση *setSpeed(int Curretnspeed,double pitch,double roll,Servo &MotorFL ,Servo &MotorFR, Servo &MotorRL,Servo &MotorRR)* και η συνάρτηση *arm(Servo &MotorFL ,Servo &MotorFR, Servo &MotorRL,Servo &MotorRR)*. Η πρώτη συνάρτηση από ότι παρατηρούμε δέχεται τέσσερις pointers κλάσεων που ο κάθε ένας δείχνει στη διεύθυνση μνήμης της κάθε κλάσης τύπου Servo, τις οποίες έχουμε δημιουργήσει αντιπροσωπεύοντας τα τέσσερα μοτέρ που είναι εγκατεστημένα στο quadcopter. Η συνάρτηση *setSpeed* καλείται ύστερα από τον υπολογισμό της κάθε *PID* εξόδου, και ορίζει την ταχύτητα κίνησης στο κάθε μοτέρ.

Στην συνέχεια, συναντάμε τη συνάρτηση *arm* η οποία πρέπει να κληθεί κατά την εκκίνηση του προγράμματος και είναι υπεύθυνη για τον οπλισμό των μοτέρ. Μέσω της

συνάρτησης γράφουμε κατάλληλα σήματα *PWM* στο κάθε *ESC*, τα οποία αντιπροσωπεύουν ένα σήμα παλμοσειράς μέγιστης περιόδου που μπορούν να δεχτούν τα *ESC* και που έχουμε προ – δηλώσει στη διαδικασία του *Calibration*. Για να οπλιστούν τα μοτέρ η διάρκεια παραγωγής των δειγμάτων πρέπει να είναι τουλάχιστον 2 sec. Στη δικιά μας περίπτωση επιλέγουμε η διάρκεια να είναι 6 sec, το οποίο ορίζεται στο *manual* των *Afro ESC* που έχουμε επιλέξει. Παρακάτω επισυνάπτουμε τον κώδικα που περιλαμβάνεται στα αρχεία *Motors.h* και *Motors.cpp*.

Listing 23. Motors.h Report c++ language technical details

```
/*
-----
MSC HEP 2013-2014
-----
*/
/*
-----
Project      : Quadcopter
File         : Motors_Control.ino
Description   : This code defines Motor Speed process
Author        : Monahopoulos Konstantinos
-----
*/
/*
-----
Includes
-----
*/
#include <Servo.h>

/*
-----
DEFINITIONS
-----
*/
#ifndef MOTORS_H
#define MOTORS_H

/*
-----
MOTOR CLASS DECLARATION
-----
*/
class MOTORS_CLASS
```

```

{

    public:
        MOTORS_CLASS(void); //class constructor
        void setSpeed(int MotorFLCurretnspeed,int MotorFRCurretnspeed,int
MotorRLCurretnspeed,int MotorRRCurretnspeed,double pitch,double roll,double
yaw,double altitude,Servo &MotorFL ,Servo &MotorFR, Servo &MotorRL,Servo
&MotorRR);
        void arm(Servo &MotorFL ,Servo &MotorFR, Servo &MotorRL,Servo &MotorRR);
        int MotorSpeed[3];

    private:
        int MotorSpeedPitch;
        int MotorSpeedRoll;

};

#endif

```

Listing 24. Motors.cpp Report c++ language technical details

```

/*
-----
----- MSc HEP 2013-2014 -----
-----

*/
/*
-----
----- Project      : Quadcopter
File          : Motors_Control.ino
Description   : This code defines Motor Speed process
Author        : Monahopoulos Konstantinos
-----
*/

/*
-----
----- Includes
-----
*/
#include <Arduino.h>
#include "Motors.h"

/*
-----
----- DEFINITIONS
-----
*/
#define MAX_SIGNAL 2000 // maximum us at esc
#define MIN_SIGNAL 800 // minimum us at esc

/*
-----
----- Function Prototyping
-----
*/

```

```

// -----
//      Constructor function
// -----
MOTORS_CLASS::MOTORS_CLASS(void) {

    MotorSpeed[0]=1000;
    MotorSpeed[1]=1000;
    MotorSpeed[2]=1000;
    MotorSpeed[3]=1000;
}

// -----
// Set Speed to the motors defined by the PID
// -----
void MOTORS_CLASS::setSpeed(int MotorFLCurretnspeed,int
MotorFRCurretnspeed,int MotorRLCurretnspeed,int MotorRRCurretnspeed,double
pitch,double roll,double yaw,double altitude,Servo &MotorFL ,Servo &MotorFR,
Servo &MotorRL,Servo &MotorRR) {

    /* The desired roll or pitch force will
       be translated into the thrust ratio between the 2 motors that can effect
       that force.*/

    MotorFL.writeMicroseconds(round(MotorFLCurretnspeed - roll - pitch +
yaw));
    MotorFR.writeMicroseconds(round(MotorFRCurretnspeed + roll - pitch -
yaw));
    MotorRL.writeMicroseconds(round(MotorRLCurretnspeed - roll + pitch -
yaw));
    MotorRR.writeMicroseconds(round(MotorRRCurretnspeed + roll + pitch +
yaw));

}

// -----
// Send MIN_SIGNAL to esc for ARMing Procedure
// -----
void MOTORS_CLASS::arm(Servo &MotorFL ,Servo &MotorFR, Servo &MotorRL,Servo
&MotorRR) {
    MotorFL.writeMicroseconds(MIN_SIGNAL);
    MotorFR.writeMicroseconds(MIN_SIGNAL);
    MotorRL.writeMicroseconds(MIN_SIGNAL);
    MotorRR.writeMicroseconds(MIN_SIGNAL);
    delay(2000);
}

```

ΚΕΦΑΛΑΙΟ 5: ΤΗΛΕΧΕΙΡΙΣΜΟΣ

5.1 ΧΕΙΡΗΣΜΟΣ ΣΤΟ ΔΕΚΤΗ (QUADCOPTER)

Για να μπορεί ο χρήστης να ελέγξει την κίνηση του quadcopter πρέπει πρώτα να έχουμε εγκαταστήσει ένα χειριστήριο το οποίο θα αποτελεί το μέσω ελέγχου. Αφού ο αλγόριθμος που αναπτύσσουμε δεν αποτελεί μέρος ενός ήδη υπάρχοντος project αλλά έχει χτιστεί από το μηδέν, έχουμε την επιλογή να τοποθετήσουμε οποιοδήποτε χειρηστήριο επιθυμούμε και μέσω κατάλληλων σημάτων αποστολής, λήψης και αποκωδικοποίησης να πραγματοποιηθεί ο έλεγχος του quadcopter.

Το χειρηστήριο που χρησιμοποιούμε είναι ένα χειριστήριο *PS2* που μέσω κατάλληλων βιβλιοθηκών μπορούμε, να επιβληθούμε των λειτουργιών του, συνδέοντας το με ένα *Arduino Board*. Ύστερα, η αμφίδρομη επικοινωνία πραγματοποιείται μέσω δύο *Xbee* modules χρησιμοποιώντας το πρωτόκολλο ασυρμάτου δικτύου αισθητήρων *Zigbee* 802.15.4.

Σε αυτό το σημείο περιγράφεται μόνο ο αλγόριθμος ανόγνωσης των εντολών που παράγονται από το χειριστήριο προς τον μικροελεγκτή που είναι εγκατεστημένος στο *Quadcopter*. Η διαδικασία συνδεσμολογίας, επικοινωνίας και εκτέλεσης που έχουμε αναπτύξει στον μικροελεγκτή του *PS2* χειρηστηρίου θα ερευνηθεί σε επόμενο βήμα.

Όπως αναφέραμε το χειριστήριο απαιτεί ένα μίζον προαπαιτούμενο για την σωστή πτήση του *quadcopter*. Για το λόγο αυτό, αλλα και για λόγους ασφαλείας κατά την εκκίνηση του προγράμματος, εντός της συνάρτησης `void setup()` έχουμε επιλέξει τον έλεγχο εγκαθίδρυσης επικοινωνίας του χειριστηρίου.

Όπως θα δούμε και αργότερα κατά την εκκίνηση του προγράμματος που εκτελείται στο μικροελεγκτή έχουμε θέσει ένα `status flag`, το οποίο σηματοδοτεί την ορθή εκκίνηση λειτουργίας του. Όταν ολοκληρωθεί η αρχικοποίηση του χειρηστηρίου το `flag` αυτό παίρνει τη τιμή 1 και στέλνει ένα `byte` σήμανσης στο quadcopter. Κατά τον έλεγχο ύπαρξης του χειριστηρίου έχουμε τοποθετήσει μία `while loop` η οποία κάθε 150ms ελέγχει εάν το `status flag` εχει πάρει τη τιμή 1.

Όταν πραγματοποιηθεί η εγκαθίδρυση, είμαστε πλέον σίγουροι τόσο ότι το χειριστήριο έχει αρχικοποιηθεί σωστά, όσο και ότι η επικοινωνία μεταξύ των *Xbees* έχει ολοκληρωθεί. Για λόγους οπτικής επιβεβαίωσης, κρατάμε σε κατάσταση *HIGH* το *status led* που έχουμε εγκαταστήσει στο *pin 13*, για ένα δευτερόλεπτο. Η παραπάνω διαδικασία πραγματοποιείται με χρήση του παρακάτω κώδικα, ο οποίος αποτελεί μέρος του συνολικού αλγορίθμου.

```
#ifdef PS2_CONTROLLER
    while (1)
    {
        digitalWrite(STATUS_LED, HIGH);
        delay(150);
        if(Serial.available() > 0){

            PSControllerStatus = Serial.parseInt();

            if(PSControllerStatus == true){
                digitalWrite(STATUS_LED, HIGH);
                delay(1000);
                digitalWrite(STATUS_LED, LOW);
                break;
            }
        }
        digitalWrite(STATUS_LED, LOW);
        delay(150);
    }
#endif
```

Εισέρχοντας εντός της *void loop()* συνάρτησης η οποία κλίνεται επαναληπτικά, στην αρχή κάθε κύκλου, ελέγχουμε εάν έχει δοθεί από το χειριστήριο κάποια εντολή προς εκτέλεση από το *quadcopter*. Η διαδικασία αυτή γίνεται ελέγχοντας τυχόν λαμβανόμενη πληροφορία από τη σειριακή θύρα του μικροελεγκτή μέσω της εντολής *if(Serial.available() > 0)*. Έχοντας αρκετές επιλογές, λόγω της πληθώρας πλήκτρων που βρίσκονται στο χειριστήριο, μπορούμε με χρήση του κάθε κουμπιού να πραγματοποιήσουμε στο *quadcopter* όλες τις μανούβρες που επιτρέπουν οι βαθμοί ελευθερίας του.

Επιπροσθέτως, έχουμε ενσωματώσει κάποιες επιπλέον λειτουργίες, που μέσω αυτών μπορούμε να κινήσουμε τη ενσωματωμένη κάμερα τόσο οριζόντια όσο και κάθετα και να αλλάξουμε την ευελιξία κίνησης του *quadcopter* θέτοντας το *acrobatic mode on/off* αλλάζοντας αντίστοιχα τις τιμές του *PID Controller*. Επίσης υπάρχει η δυνατότητα εκτέλεσης *emergency stop* μέσω της οποίας μπορούμε να απενεργοποιήσουμε

ακαριαία την περιστροφή των μοτέρ σε περίπτωση έκτακτης ανάγκης. Ακόμη, μπορούμε να ενεργοποιήσουμε τη διαδικασία *Altitude Hold on/off* μέσω της οποίας το quadcopter πραγματοποιεί hovering σε σταθερό ύψος.

Για να γνωρίζουμε ότι κάποια εντολή δίνεται τη δεδομένη στιγμή, αναβοσβήνουμε με γρήγορο ρυθμό το Status Led που έχουμε ενσωματώσει, ενώ ύστερα αποκωδικοποιούμε την εντολή μέσω της συνάρτησης `Serial.readStringUntil('\n')`, αποθηκευοντας το αποτέλεσμα στη string μεταβλητή `StringCommand`. Κάθε εντολή που στέλνεται από το χειριστήριο, ακολουθείται από το χαρακτήρα του κενού `('\\n')`, άρα μέσω της παραπάνω εντολής μπορούμε να απομονώσουμε τη γραμματοσειρά (*ASCII*) της εντολής και αναλόγως να εισέλθουμε στο αντίστοιχο `if statement`.

Παρακάτω θα επισυνάψουμε τον βασικό κορμό του κώδικα που εκτελεί όλη την διαδικασία. Πρέπει να αναφέρουμε ότι κάποια κομμάτια επιπλέον λειτουργίας του quadcopter, μέσω εντολοδότης από το χειριστήριο δεν έχουν ακόμα ολοκληρωθεί. Πιο αναλυτικά δεν έχουμε ενσωματώσει στον αλγόριθμο σταθεροποίησης μέσω της συνιστώσα του βαρόμετρου. Όλα αυτά τα επιπλέον βήματα δεν μας επηρεάζουν στην πραγματοποίηση μιας απλής πτήσης του quadcopter, παρόλα αυτά στο τελικό `release` που θα συνοδεύεται με αυτό το έγγραφο ευελπιστούμε να έχουμε ολοκληρώσει όλα τα παραπάνω βήματα.

Listing 25. Main_Quadcopter_Code.cpp Report c++ language technical details

```
/*
-----
MSc HEP 2013-2014
-----
*/
/*
Project      : Quadcopter
File        : Quadcopter_Release_1_00_fixed.ino
Description   : This code calibrates, test and debug the Quadcopter
Author       : Monahopoulos Konstantinos
-----

*/
// Structure:
// -----
// Arduino | -----
// UNO     | - 3.3 V -----| MPU 6050 |
//           | - GND -----| HMC5883,  |
//           | - SDA -----| MS5611   |
//           | - SCL -----|           |
//           |             |
//-----|-----|-----|-----|
//           |-----|-----|-----|
//           V-----|-----|
//           Integrated IMU sensor

/*
-----
Includes
-----
*/
#include <Wire.h>
#include <Servo.h>
#include "gyro_accel.h"
#include "debug.h"
#include "Motors.h"
#include "Compass.h"
#include "MS5611.h"
#include "PID.h"

/*
-----
DEFINITIONS
-----
*/
/* DEFINE MPU6050_CALIBRATION TO CALIBRATE QUAD ON INITIALIZATION */
// #define MPU6050_CALIBRATION

/* DEFINE MPU6050_CALIBRATION TO USE PRECALIBRATION OFFSET QUAD ON
INITIALIZATION */
#define MPU6050_PRECALIBRATION
```

```

/* DEFINE HMC5883_CALIBRATION TO CALIBRATE QUAD ON INITIALIZATION */
//#define HMC5883_CALIBRATION

/* DEFINE HMC5883_CALIBRATION TO USE PRECALIBRATION OFFSET QUAD ON
INITIALIZATION */
#define HMC5883_PRECALIBRATION

/* DEFINE MATLAB_DATA_LOGGING TO DEBUG SENSOR VALUES TO MATLAB */
//#define MATLAB_DATA_LOGGING

/* DEFINE DEBUG TO DEBUG SENSOR VALUES TO ARDUINO */
//#define DEBUG

/* DEFINE PS2 CONTROLLER CONFIRMATION */
#define PS2_CONTROLLER

/* DEFINE PID COMPUTATION */
#define PID

/* DEFINE MOTORS OUTPUT */
#define MOTORS

#define rad2degree 57.3           // Radian to degree conversion

/* default Filter_gain = 0.95 . Lowering the Filter Gain means using more
the accelerometer but the more the Filter_gain
reduced the more noise you get from accelerometer. If you set Filter_gain=1
means using only Gyroscope measurement.
*/
#define Filter_gain_Gyro_Accel 0.99      // Accelerometer Gain
#define Filter_gain_Gyro_Compass 0.00    // Gyroscope Gain
#define MAX_SIGNAL 2000                 // maximum us at esc
#define MIN_SIGNAL 900                  // minimum us at esc
#define PID_LED 8                      // Led in pin 8
#define PS2_STATUS_LED 12               // Led in pin 12
#define SENSORS_STATUS_LED 13          // Led in pin 13
#define dt 80                          // time diffrence in millis
seconds
/*
-----
-----
Global Variables
-----
-----
*/
unsigned long t;                           // Time Variable
unsigned int PSControllerStatus=0;         // Status flag
String StringCommand;                     // Command from controller
int ServoValuesLR=90;                    // 
int ServoValuesUD=90;                    // 
String Acrobatic="OFF";                  // Acrobatic Mode is OFF at
initialization
String AltitudeHold="OFF";               // Altitude Hold is OFF at
initialization
String QuadAtGround="ON";                // While Quadcopter is still at
ground QuadAtGround is ON

/*
-----
-----
Assign Classes
-----
-----
*/
MPU6050_CLASS MPU6050obj;                // Mpu-6050 Object
HMC5883_CLASS HMC5883obj;                // HMC5883 Object

```

```

MS5611_CLASS MS5611obj;                                // MS5611 Object
DEBUG_CLASS DEBUGobj;                                  // Debug Object
MOTORS_CLASS MOTORSobj;                               // Motors Object

// -----
// PID Configuration
// -----


/* Set PID values for Pitch - Roll - Yaw */
PID_INFO PID_ROLL_info(0.859,0.618,0.319);
PID_INFO PID_PITCH_info(0.959,0.747,0.300);
PID_INFO PID_YAW_info(0.20,0.000,0.005);
PID_INFO PID_ALTITUDE_info(0.000,0.000,0.000);

/* Assign Pointer variables that will be constantly updated in main loop*/
PID_EXEC PID_ROLL_exec(&MPU6050obj.angle_y, &PID_ROLL_info.PIDOutput,
&PID_ROLL_info.Setpoint,PID_ROLL_info.kp,PID_ROLL_info.ki,PID_ROLL_info.kd,
DIRECT );
PID_EXEC PID_PITCH_exec(&MPU6050obj.angle_x, &PID_PITCH_info.PIDOutput,
&PID_PITCH_info.Setpoint,PID_PITCH_info.kp,PID_PITCH_info.ki,PID_PITCH_info.
kd, DIRECT );
PID_EXEC PID_YAW_exec(&HMC5883obj.bearing, &PID_YAW_info.PIDOutput,
&PID_YAW_info.Setpoint,PID_YAW_info.kp,PID_YAW_info.ki,PID_YAW_info.kd,
DIRECT );
PID_EXEC PID_ALTITUDE_exec(&MS5611obj.Absolute_Altitude,
&PID_ALTITUDE_info.PIDOutput,
&PID_ALTITUDE_info.Setpoint,PID_ALTITUDE_info.kp,PID_ALTITUDE_info.ki,PID_AL
TITUDE_info.kd, DIRECT );
Servo myescFL;                                         // Servo Front Left Object
Servo myescFR;                                         // Servo Front Right Object
Servo myescRL;                                         // Servo Rear Left Object
Servo myescRR;                                         // Servo Rear Right Object
Servo CameraServoLR;                                   // Camera Left - Right Object
Servo CameraServoUD;                                   // Camera Up - Down Object

/*
-----
----- Main Code -----
-----
*/
void setup(){
    Serial.begin(9600);
    Wire.begin();

// -----
// Camera Mount Configuration
// -----


CameraServoLR.attach(11);                             // Attach camera pins
CameraServoUD.attach(10);                            // Attach camera pins
CameraServoLR.write(ServoValuesLR);                  // Initiate camera servo 1
CameraServoUD.write(ServoValuesUD);                  // Initiate camera servo 2

// -----
// Motor Configuration
// -----


myescFL.attach(3);                                    // Attach motor pins to esc
myescFR.attach(5);                                    // Attach motor pins to esc
myescRL.attach(6);                                    // Attach motor pins to esc
myescRR.attach(9);                                    // Attach motor pins to esc
MOTORSobj.arm(myescFL,myescFR,myescRL,myescRR); // Arm the esc

// -----
// Led Configuration

```

```

// -----
pinMode(PS2_STATUS_LED, OUTPUT);           // Attach Led pin
pinMode(SENSORS_STATUS_LED, OUTPUT);      // Attach Led pin
pinMode(PID_LED, OUTPUT);                // Attach Led pin

// -----
// MPU6050 Configuration
// -----

MPU6050obj.MPU6050_ResetWake();          // Wake up MPU - 6050
MPU6050obj.MPU6050_SetGains(0,1);        // Setting the lows scale
MPU6050obj.MPU6050_SetDLPF(0);          // Setting the DLPF to inf

Bandwidth for calibration

// -----
// HMC5883 Configuration
// -----

HMC5883obj.HMC5883_init(2);             // Wake up HMC5883 */

Serial.print("Compass Gain updated to = ");
Serial.print(HMC5883obj.compass_gain_fact);
Serial.println(" mG/bit");
Serial.println(" ");

// -----
// MS5611 Configuration
// -----

MS5611obj.init(MS561101BA_ADDR_CS_B_LOW);

// -----
// Definitions Configuration
// -----

#ifndef MPU6050_CALIBRATION
    MPU6050obj.MPU6050_OffsetCal();
#endif

#ifndef MPU6050_PRECALIBRATION
    MPU6050obj.MPU6050_Set_Precalibration_offset();
#endif

#ifndef HMC5883_CALIBRATION
    HMC5883obj.compass_debug = 1;
    HMC5883obj.HMC5883_offset_calibration(3);
#endif

#ifndef HMC5883_PRECALIBRATION
    HMC5883obj.HMC5883_Set_Precalibration_offset();
#endif

MPU6050obj.MPU6050_SetDLPF(6);           // Set the DLPF
decreasing the bandwidth by increasing this number. [MAX BW,MIN BW] = [0,6]

// -----
// Sensor Configuration Status
// -----

digitalWrite(SENSORS_STATUS_LED, HIGH);    // All Sensors are ready
to use !

// -----
// PS2 controller identifier
// -----

```

```

/* Check correct initialization for PS2 controller*/

#ifndef PS2_CONTROLLER
    while (1)
    {
        //Serial.println(" Waiting PS2 Controller configuration.. ");
        digitalWrite(PS2_STATUS_LED, HIGH);
        delay(150);

        if(Serial.available() > 0){
            PSControllerStatus = Serial.parseInt();

            if(PSControllerStatus == true){
                digitalWrite(PS2_STATUS_LED, HIGH);
                delay(1000);
                digitalWrite(PS2_STATUS_LED, LOW);
                break;
            }
        }
        digitalWrite(PS2_STATUS_LED, LOW);
        delay(150);
    }
#endif

// -----
// Set Initialization setpoint for Compass
// -----

HMC5883obj.HMC5883_scaled_reading();
HMC5883obj.HMC5883_heading();
PID_YAW_info.Setpoint = HMC5883obj.bearing;
Serial.print("Initialization Yaw in degrees = ");
Serial.println(HMC5883obj.bearing);
Serial.println(" ");

/* Check initial degrees at StartUp */
if(PID_YAW_info.Setpoint <= 40 || PID_YAW_info.Setpoint >= 320){

    while(1){
        digitalWrite(SENSORS_STATUS_LED, HIGH);
        delay(150);
        digitalWrite(SENSORS_STATUS_LED, LOW);
        delay(150);
        Serial.print("Problem at initialization process in degrees = ");
        Serial.println(HMC5883obj.bearing);
    }
}

// -----
// PID meta_info
// -----

/* Start PID clocks for the first execution */
PID_ROLL_exec.lastTime = millis()-PID_ROLL_exec.SampleTime;
PID_PITCH_exec.lastTime = millis()-PID_PITCH_exec.SampleTime;
PID_YAW_exec.lastTime = millis()-PID_YAW_exec.SampleTime;
PID_ALTITUDE_exec.lastTime = millis()-PID_ALTITUDE_exec.SampleTime;
}

void loop(){

    /* AT THE BEGINNING FOR EVERY #number LOOPS I MUST CHECK THE STRENGTH OF
    THE XBEE RANGE. IF STRENGTH IS LOWER THAN A VALUE ...
    LOWER THE MOTOR SPEED UNTIL HIGH EQUALS TO ...
    */
}

```

```

    t=millis(); //Number of milliseconds since the program started (unsigned
long), This number will overflow (go back to zero), after approximately 50
days.
/*
-----
-----MAIN PROCESS-----
-----
*/
/*
-----
-----MPU - 6050 implementation-----
-----
*/
MPU6050obj.MPU6050_ReadData();

// -----
//      Read Gyro Values From MPU6050
// -----
MPU6050obj.angle_x_gyro =
(MPU6050obj.gyro_x_scaled*((double)dt/1000)+MPU6050obj.angle_x);
MPU6050obj.angle_y_gyro =
(MPU6050obj.gyro_y_scaled*((double)dt/1000)+MPU6050obj.angle_y);
MPU6050obj.angle_z_gyro =
(MPU6050obj.gyro_z_scaled*((double)dt/1000)+MPU6050obj.angle_z);

// -----
//      Read Accelerometer Values From MPU6050
// -----
MPU6050obj.angle_x_accel = atan(MPU6050obj.accel_y_scaled
/(sqrt(MPU6050obj.accel_x_scaled*MPU6050obj.accel_x_scaled+MPU6050obj.acce
l_z_scaled*MPU6050obj.accel_z_scaled)))*(double)rad2degree;
MPU6050obj.angle_y_accel = -
atan(MPU6050obj.accel_x_scaled/(sqrt(MPU6050obj.accel_y_scaled*MPU6050obj.
accel_y_scaled+MPU6050obj.accel_z_scaled*MPU6050obj.accel_z_scaled)))*(do
uble)rad2degree;
MPU6050obj.angle_z_accel =
atan(MPU6050obj.accel_z_scaled/(sqrt(MPU6050obj.accel_y_scaled*MPU6050obj.
accel_y_scaled+MPU6050obj.accel_x_scaled*MPU6050obj.accel_x_scaled)))*(do
uble)rad2degree;

// -----
//      Compute Overall Angle
// -----
MPU6050obj.angle_x = Filter_gain_Gyro_Accel*MPU6050obj.angle_x_gyro+(1-
Filter_gain_Gyro_Accel)*MPU6050obj.angle_x_accel;
MPU6050obj.angle_y = Filter_gain_Gyro_Accel*MPU6050obj.angle_y_gyro+(1-
Filter_gain_Gyro_Accel)*MPU6050obj.angle_y_accel;
MPU6050obj.angle_z = Filter_gain_Gyro_Accel*MPU6050obj.angle_z_gyro+(1-
Filter_gain_Gyro_Accel)*MPU6050obj.angle_z_accel;

/*
-----
-----HMC5883 implementation-----
-----
*/
HMC5883obj.HMC5883_scalled_reading();

```

```

HMC5883obj.HMC5883_heading();
if (HMC5883obj.bearing > 320 ) {
HMC5883obj.bearing =320;
} else if(HMC5883obj.bearing < 40 ) {
HMC5883obj.bearing =40;
}

/*
-----
MS5611 implementation
-----
*/
MS5611obj.temperature = MS5611obj.getTemperature(MS561101BA_OSР_4096);
MS5611obj.pressure = MS5611obj.getPressure(MS561101BA_OSР_4096);
MS5611obj.Absolute_Altitude=MS5611obj.getAltitude(MS5611obj.pressure,
MS5611obj.temperature);

/*
-----
PID implementation
-----
*/
#endif PID

/* Compute PID Output only if Quad left the ground*/
if (QuadAtGround.equals("OFF")){
PID_ROLL_exec.Compute();
PID_PITCH_exec.Compute();
PID_YAW_exec.Compute();
PID_ALTITUDE_exec.Compute();
}
#endif

// -----
// Motors implementation
// -----

#ifndef MOTORS

/* PID Output to Motors */
MOTORSobj.setSpeed(MOTORSobj.MotorSpeed[0],MOTORSobj.MotorSpeed[1],MOTORSobj
.MotorSpeed[2],MOTORSobj.MotorSpeed[3],PID_PITCH_info.PIDOutput,PID_ROLL_inf
o.PIDOutput,PID_YAW_info.PIDOutput,PID_ALTITUDE_info.PIDOutput,myescFL,myesc
FR,myescRL,myescRR);

#endif

// -----
// READ CONTROLLER COMMAND
// -----

if(Serial.available() > 0)
{
    digitalWrite(PS2_STATUS_LED, HIGH); // Turn Led on
    delay(5); // Watch out this
delay ..
    digitalWrite(PS2_STATUS_LED, LOW); // Turn Led off
    StringCommand = Serial.readStringUntil('\n'); // Fetch Command

// -----
// TURN CAMERA LEFT
// -----

```

```

if(StringCommand.equals("L1") && ServoValuesLR<160) {
    ServoValuesLR+=5;
    CameraServoLR.write(ServoValuesLR); }

// -----
// TURN CAMERA RIGHT
// -----

if(StringCommand.equals("R1")&& ServoValuesLR>20) {
    ServoValuesLR-=5;
    CameraServoLR.write(ServoValuesLR);
}

// -----
// TURN CAMERA DOWN
// -----

if(StringCommand.equals("L2") && ServoValuesUD<160) {
    ServoValuesUD+=5;
    CameraServoUD.write(ServoValuesUD);
}

// -----
// TURN CAMERA UP
// -----

if(StringCommand.equals("R2")&& ServoValuesUD>20) {
    ServoValuesUD-=5;
    CameraServoUD.write(ServoValuesUD);
}

// -----
// ACROBATIC ON - Not supported yet
// -----

if(StringCommand.equals("SELECT") && Acrobatic.equals("OFF")){
    Acrobatic="ON";
    StringCommand="VOID"; // Set command to VOID to avoid entering to
another Mode

    /* Set the Kp,Ki,Kd of PID_PITCHobj and PID_ROLLobj to big values
       for AGGRESSIVE maneuvers OR change the step of angle set. Led on acrobatic
led*/
}

// -----
// ACROBATIC OFF - Not supported yet
// -----

if(StringCommand.equals("SELECT") && Acrobatic.equals("ON")){
    Acrobatic="OFF";
    StringCommand="VOID"; // Set command to VOID to avoid enter to another
Mode

    /* Set the Kp,Ki,Kd of PID_PITCHobj and PID_ROLLobj to small values
       for SMOOTH maneuvers OR change the step of angle set. Led off acrobatic
led*/
}

// -----
// EMERGENCY STOP
// -----

if(StringCommand.equals("START")){
    while(1){
        MOTORSobj.setSpeed(0,0,0,0,0,PID_YAW_info.PIDOutput,PID_ALTITUDE_info.PIDO
utput,myescFL,myescFR,myescRL,myescRR);
}
}

```

```

        digitalWrite(PS2_STATUS_LED, HIGH);
    }

}

// -----
// ALTITUDE UP
// -----


if(StringCommand.equals("UP")) {

/* Read the decimal part of the command*/
StringCommand = Serial.readStringUntil('\n');
/* Maximum speed in writeMicroseconds is 1900,
   MAX CORRECTION IS 60,
   SETTING THE GAP TO 60,
   1900(max) - Gap = 1840*/



if (MOTORSobj.MotorSpeed[0]>1840 && MOTORSobj.MotorSpeed[1] >1840 &&
MOTORSobj.MotorSpeed[2] >1840 && MOTORSobj.MotorSpeed[3] >1840) {
    MOTORSobj.MotorSpeed[0]=1840;
    MOTORSobj.MotorSpeed[1]=1840;
    MOTORSobj.MotorSpeed[2]=1840;
    MOTORSobj.MotorSpeed[3]=1840;
} else{
/* Scale 0 - 255 to 0-25 and add it to the current motor speed*/
MOTORSobj.MotorSpeed[0]+=round(StringCommand.toInt()/10);
MOTORSobj.MotorSpeed[1]+=round(StringCommand.toInt()/10);
MOTORSobj.MotorSpeed[2]+=round(StringCommand.toInt()/10);
MOTORSobj.MotorSpeed[3]+=round(StringCommand.toInt()/10);
}

/* We know that when speed equals to 1600, Quadcopter left the Ground*/
if (MOTORSobj.MotorSpeed[0]>1600 && MOTORSobj.MotorSpeed[1] >1600 &&
MOTORSobj.MotorSpeed[2] >1600 && MOTORSobj.MotorSpeed[3] >1600 &&
QuadAtGround.equals("ON") ) {

/* Set QuadAtGround to OFF, trigger the PID LED and power on PID
implementation */
    QuadAtGround="OFF";
    digitalWrite(PID_LED, HIGH);
    PID_ROLL_exec.SetMode(AUTOMATIC);
    PID_PITCH_exec.SetMode(AUTOMATIC);
    PID_YAW_exec.SetMode(AUTOMATIC);
    PID_ALTITUDE_exec.SetMode(AUTOMATIC);
}
}

// -----
// ALTITUDE DOWN
// -----


if(StringCommand.equals("DOWN")){
/* Read the decimal part of the command*/
StringCommand = Serial.readStringUntil('\n');

/* min speed in writeMicroseconds is 1000 */
if (MOTORSobj.MotorSpeed[0]<1000 && MOTORSobj.MotorSpeed[1] <1000 &&
MOTORSobj.MotorSpeed[2] <1000 && MOTORSobj.MotorSpeed[3] <1000) {
    MOTORSobj.MotorSpeed[0]=1000;
    MOTORSobj.MotorSpeed[1]=1000;
    MOTORSobj.MotorSpeed[2]=1000;
    MOTORSobj.MotorSpeed[3]=1000;

} else{

/* Scale 0 - 255 to 0-25 and abstract it from the current motor speed*/
MOTORSobj.MotorSpeed[0]-=round(StringCommand.toInt()/10);
MOTORSobj.MotorSpeed[1]-=round(StringCommand.toInt()/10);
}
}

```

```

MOTORSobj.MotorSpeed[2]=round(StringCommand.toInt()/10);
MOTORSobj.MotorSpeed[3]=round(StringCommand.toInt()/10);

}

/* We know that when speed is smaller than 1600, Quadcopter is on the
Ground*/
if (MOTORSobj.MotorSpeed[0]<=1600 && MOTORSobj.MotorSpeed[1]<=1600 &&
MOTORSobj.MotorSpeed[2]<=1600 && MOTORSobj.MotorSpeed[3]<=1600 &&
QuadAtGround.equals("OFF")){
    /* Set QuadAtGround to ON, trigger the PID LED and power off PID
implementation */
    QuadAtGround="ON";
    digitalWrite(PID_LED, LOW);
    PID_ROLL_exec.SetMode(MANUAL);
    PID_PITCH_exec.SetMode(MANUAL);
    PID_YAW_exec.SetMode(MANUAL);
    PID_ALTITUDE_exec.SetMode(MANUAL);

}
// -----
// YAW LEFT - Needs test
// -----


if(StringCommand.equals("LEFT")){
/* Read the decimal part of the command*/
StringCommand = Serial.readStringUntil('\n');
if (PID_YAW_info.Setpoint<40){
    PID_YAW_info.Setpoint=40;
} else{
    /*Only for the Command Compute the setpoint for the PID*/
    PID_YAW_info.Setpoint+=(StringCommand.toInt()/20);
}
}

// -----
// YAW RIGHT - Needs test
// -----


if(StringCommand.equals("RIGHT")){
/* Read the decimal part of the command*/
StringCommand = Serial.readStringUntil('\n');
if (PID_YAW_info.Setpoint>320){
    PID_YAW_info.Setpoint=320;
} else{
    /*Only for the Command Compute the setpoint for the PID*/
    PID_YAW_info.Setpoint+=(StringCommand.toInt()/20);
}
}

// -----
// ROLL LEFT
// -----


if(StringCommand.equals("SQUARE")){
/* Read the decimal part of the command*/
StringCommand = Serial.readStringUntil('\n');
if (PID_ROLL_info.Setpoint>60){
    PID_ROLL_info.Setpoint=60;
} else{
    /* in this step we increase the desired angle from the Play station
controller*/
    /*Only for the Command Compute the setpoint for the PID*/
    PID_ROLL_info.Setpoint += (StringCommand.toInt()/20);
}
}

```

```

// -----
// ROLL RIGHT
// -----

if(StringCommand.equals("CIRCLE")){
/* Read the decimal part of the command*/
StringCommand = Serial.readStringUntil('\n');
if (PID_ROLL_info.Setpoint<-60){
    PID_ROLL_info.Setpoint=-60;
}else{

    /* in this step we decrease the desired angle from the Play station
controller*/
    /*Only for the Command Compute the setpoint for the PID*/
    PID_ROLL_info.Setpoint-=(StringCommand.toInt()/20);
}
}

// -----
// PITCH FRONT
// -----


if(StringCommand.equals("TRIANGLE")){
/* Read the decimal part of the command*/
StringCommand = Serial.readStringUntil('\n');

if (PID_PITCH_info.Setpoint>60){
PID_PITCH_info.Setpoint=60;
}else{

    /* in this step we increase the desired angle from the Play station
controller*/
    /*Only for the Command Compute the setpoint for the PID*/
    PID_PITCH_info.Setpoint += (StringCommand.toInt()/20);
}
}

// -----
// PITCH BACK
// -----


if(StringCommand.equals("X")){
/* Read the decimal part of the command*/
StringCommand = Serial.readStringUntil('\n');
if (PID_PITCH_info.Setpoint<-60){
    PID_PITCH_info.Setpoint=-60;
}else{

    /* in this step we decrease the desired angle from the Play station
controller*/
    /*Only for the Command Compute the setpoint for the PID*/
    PID_PITCH_info.Setpoint-=(StringCommand.toInt()/20);
}
}

// -----
// HOVER
// -----


if(StringCommand.equals("L3")) {
/*Set Pitch and roll angle values to Zero*/

    int straightquad = round((MOTORSobj.MotorSpeed[0] +
MOTORSobj.MotorSpeed[1])/2);

    MOTORSobj.MotorSpeed[0]= straightquad;
    MOTORSobj.MotorSpeed[2]= straightquad;
}

```

```

MOTORSobj.MotorSpeed[1]= straightquad;
MOTORSobj.MotorSpeed[3]= straightquad;

PID_ROLL_info.Setpoint=0;
PID_PITCH_info.Setpoint=0;
}

// -----
// Altitude Hold ON - Not supported
// -----

if(StringCommand.equals("R3") && AltitudeHold.equals("OFF")){
    /* Set Altitude hold on algorithm. Led on Altitude hold led*/
    AltitudeHold="ON";
    StringCommand="VOID"; // Set command to VOID to avoid enter to another
Mode
}

// -----
// Altitude Hold OFF - Not supported
// -----

if(StringCommand.equals("R3") && AltitudeHold.equals("ON")){
    /* Set Altitude hold off algorithm. Led off Altitude hold led*/
    AltitudeHold="OFF";
    StringCommand="VOID"; // Set command to VOID to avoid enter to another
SELECT if
}
} // End of Serial.available() - End of Command read from PS2 Controller

/* -----
----- Matlab Data Acquisition for Debug
-----
*/
/*You can Send to Matlab for plotting Only three variables at a time, you
have a choice Between : */

/* gyro_x_scaled - gyro_y_scaled - gyro_z_scaled      */
/* accel_x_scaled - accel_y_scaled - accel_z_scaled   */
/* angle_x_gyro - angle_y_gyro - angle_z_gyro        */
/* angle_x_accel - angle_y_accel - accel_z_scaled     */
/* angle_x - angle_y - angle_z                         */

#ifndef MATLAB_DATA_LOGGING
DEBUGObj.SensorArray[0]=MPU6050obj.angle_x;
DEBUGObj.SensorArray[1]=MPU6050obj.angle_y;
DEBUGObj.SensorArray[2]=MPU6050obj.angle_z;
DEBUGObj.Debug_XYZ_matlab_Print(DEBUGObj.SensorArray);
delay(200); // Delay 200 so matlab can catch the values ..

#endif

/*
-----
----- Debug to Arduino Serial implementation
-----
*/
#ifndef DEBUG
    Serial.print("angle_x = ");
    Serial.print(MPU6050obj.angle_x);

    Serial.print("\t");
    Serial.print("angle_y = ");

```

```

Serial.print(MPU6050obj.angle_y);

Serial.print("\t");
Serial.print("angle_z = ");
Serial.print(MPU6050obj.angle_z);

Serial.print("\t");
Serial.print("Yaw = ");
Serial.print((Filter_gain_Gyro_Compass*MPU6050obj.angle_z_accel) + (1-
Filter_gain_Gyro_Compass)*HMC5883obj.bearing);

Serial.print("\t");
Serial.print("PID_Roll_Output= ");
Serial.print(PID_ROLL_info.PIDOutput);

Serial.print("\t");
Serial.print("PID_Pitch_Output= ");
Serial.print(PID_PITCH_info.PIDOutput);

Serial.print("\t");
Serial.print("PID_Yaw_Output= ");
Serial.print(PID_YAW_info.PIDOutput);

Serial.print("\t");
Serial.print("Altitude = ");
Serial.print(MS5611obj.Absolute_Altitude); Serial.print(" m");

Serial.print("\t");
Serial.print("Time = ");
Serial.println(millis()-t);

#endif

while(millis()-t < dt){ /* do nothing */}

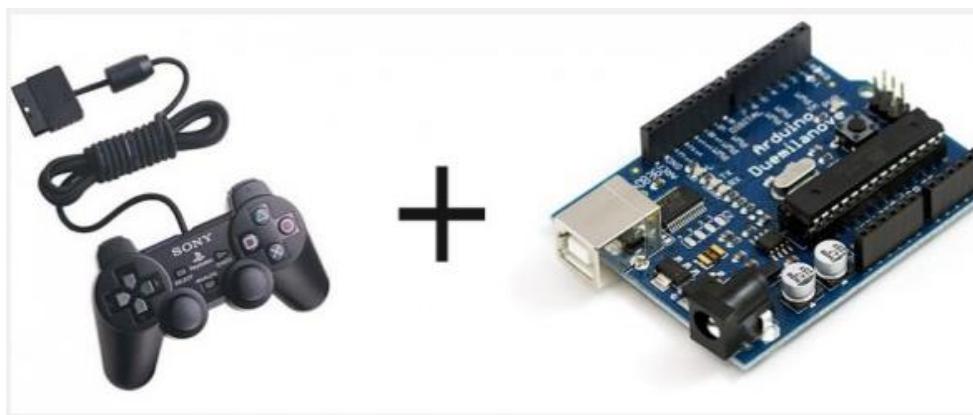
}

```

5.2 ΧΕΙΡΗΣΜΟΣ ΣΤΟ ΠΟΜΠΟ (PS2)

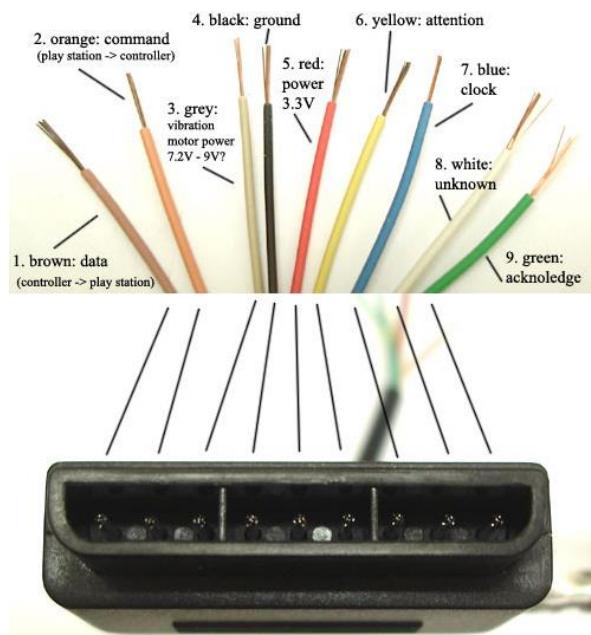
Ο χειρισμός του *quadcopter* πραγματοποιείται μέσω ενός χειριστηρίου *PS2*. Για το σκοπό αυτό έχουμε αναπτύξει ένα πρόγραμμα που είναι σε θέση να στέλνει κατάλληλους χαρακτήρες και να θέτει την κάθε λειτουργία. Ο συνολικός αλγόριθμος αποτελείται από τρία στο σύνολο αρχεία τα οποία είναι το *PS2X_QuadCptterController.ino*, το *PS2X_lib.h* και το *PS2X_lib.cpp*.

Η βιβλιοθήκη που περιλαμβάνεται μπορεί να χρησιμοποιηθεί για οποιαδήποτε εφαρμογή, αρκεί να έχουμε εγκαταστήσει σωστά τόσο το υλικό μέρος όσο και το προτόκολο επικοινωνίας. Το *baudrate* που θέτει το μήκος της λέξης έχει οριστεί στη τιμή των 9600, το οποίο δηλώνεται εντός της συνάρτησης *void setup()* τόσο στον αλγόριθμο του χειριστηρίου, όσο και στον αλγόριθμο του base station δηλαδή του *quadcopter*.



Εικόνα 57. PS2 ελεγκτής χειρισμού και Arduino Uno .

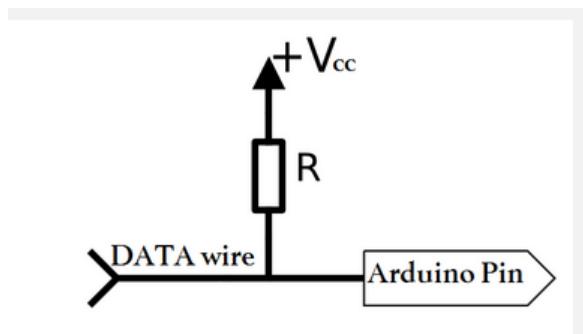
Εφόσων έχει οριστεί το baudrate καλείται η συνάρτηση ps2x.config_gamepad(13,11,10,12, true, true) της οποίας τα ορίσματα εισόδου σηματοδοτούν την συνδεσμολογία που έχουμε πραγματοποιήσει στα αντίστοιχα pins. Πιο συγκεκριμένα η συνάρτηση αυτή μεταφράζεται ως GamePad(clock, command, attention, data, Pressures?, Rumble?), το οποίο σημαίνει ότι πρέπει να εντοπίσουμε τα αντίστοιχα σήματα που εξάγονται από τον PS2 controller και να τα συνδέσουμε στα σωστά pins. Η σειρά των σημάτων εξόδου που είναι και αυτή που ακολουθήσαμε παρουσιάζονται στη παρακάτω εικόνα.



Εικόνα 58. Διαμόρφωση σημάτων εξόδου PS2 ελεγκτή .

Η μοναδική αλλαγή που πραγματοποιήσαμε είναι να εφαρμόσουμε μια *pull – up* αντίσταση των $10kohm$ μεταξύ της τάσης (5v) και του pin 12 που όπως βλέπουμε αντιπροσωπεύει το pin των δεδομένων. Για την πλήρη λειτουργία του χειριστηρίου χρειαστήκαμε δύο πηγές ενέργειας ($2 \times 9.6V$ batteries), μία εκ των οποίων είναι υπεύθυνη για την τροφοδοσία του *Arduino* και μία για να τροφοδοτεί απευθείας το χειριστήριο έτσι ώστε να ενεργοποιηθεί η λειτουργία της δόνησης όταν πατάμε κάποιο από τα κουμπιά χειρισμού.

Για τη τροφοδοσία του χειριστηρίου χρειαζόμαστε την τάση των 3.3v που εξάγεται από το αντίστοιχο pin του *Arduino*. Γενικώς όπως συμβαίνει σε πολλά χειριστήρια η τάση της παλμοσειράς που εξάγεται από το pin δεδομένων μπορεί να μην είναι αρκετή για να την αναγνωρίζει και αποκοδικοποιήσει ο μικροελενγκτής. Για να ενισχύσουμε το πλάτος των παλμών εφοδιαστήκαμε την απαραίτητη τάση από την τροφοδοσία των 5v και μέσω της *pull – up* αντίστασης οδηγήσαμε το σήμα προς ανάγνωση. Η διαδικασία αυτή φαίνεται στη παρακάτω εικόνα.



Εικόνα 59. Περιγραφή σύνδεσης *Pull – Up* αντίστασης στο PS2 ελεγκτή .

Έχοντας ολοκληρώσει την περιγραφή της υλικής συνδεσμολογίας θα συνεχίσουμε στο αλγορίθμικό κομμάτι, για να αντιληφθούμε τη λειτουργικότητα του χειριστηρίου. Ελένχοντας τη τιμή που επιστρέφει η συνάρτηση διαμώρφωσης (configuration) θέτουμε ανάλογα τη τιμή της μεταβλητής `PsStatusFlag` σε TRUE και τη στέλνουμε στο μικροελενγκτή του *quadcopter*. Αργότερα, αυτή η μεταβλητή ελέγχεται και ανάλογα ορίζει την συνέχιση του προγράμματος, όπως περιγράψαμε στο προηγούμενο βήμα.

Η επόμενη συνάρτηση που συναντάμε είναι η `ps2x.read_gamepad(false, SumOfVibration)` η οποία ενεργοποιεί τη διαδικασία της δόνησης κατά το πάτημα των κουμπιών που έχουμε ορίσει, με ένταση δόνησης αναλόγως της πίεσης του πατήματος. Ακολούθως γίνεται ανάγνωση του κουμπιού που πατήθηκε και ενεργοποιείται το ανάλογο ποσοστό δόνησης στέλνοντας το χαρακτήρα (`string`) που αντιπροσωπεύει το κουμπί, μαζί με την ανάλογη *8-bit* τιμή πίεσης. Η δεκαδική τιμή που στέλνεται μέσω της σειριακής θύρας, είναι η ίδια που στέλνεται και στη συνάρτηση δόνησης και οριοθετείται μεταξύ των τιμών 0 – 255. Από ότι μπορούμε να παρατηρήσουμε η αναλογική τιμή που αποτελεί έξοδος της συνάρτησης (`ps2x.Analog`) καλείται τόσο στο `if` statement για την επιλογή του κουμπιού, όσο και στο τέλος του προγράμματος θέτοντας το ποσοστό δόνησης. Τα κουμπιά που υποστηρίζουν τη λειτουργία αυτή είναι τα κουμπιά κίνησης του quadcopter.

Αντίστοιχα, έχουμε ορίσει και κάποια κουμπιά τα οποία θέλουμε μόνο να αλλάζουν κατάσταση από *High* σε *Low* και αντίστροφα. Το πάτημα των συγκεκριμένων κουμπιών ελέγχεται στο σημείο που αναγράφει `if (ps2x.NewButtonState())`, σηματοδοτώντας το mode που θα λειτουργεί το *quadcopter*. Τα κουμπιά κατάστασης είναι τα *L3, R3, START* και *SELECT*.

Παρακάτω επισυνάπτουμε τα δομικά μέρη του προγράμματος τα οποία περιγράψαμε σε αυτό το κεφάλαιο. Επιπροσθέτως θα αναπτύξουμε τη διαδικασία του testing και θα προβάλουμε τα αποτελέσματα μέσω της σειριακής θύρας. Τέλος θα αναπτύξουμε ένα χάρτη ανάγνωσης των κουμπιών του χειριστηρίου με την αντίστοιχη λειτουργία τους για περισσότερη ευκολία του χρήστη κατά τη πτήση του quadcopter.

Listing 26. PS2X.lib.h Report c++ language technical details

```
/*
-----
----- MSC HEP 2013-2014 -----
-----
*/
/*
-----
----- Project      : Quadcopter
File          : PS2X_QuadCpterController.ino
Description   : This code implements PS2 controller functionality
Author        : Monahopoulos Konstantinos
-----
```

```

-----  

*/  

/*-----  

-----  

    Includes  

-----  

*/  

#ifndef PS2X_lib_h  

#define PS2X_lib_h  

#if ARDUINO > 22  

#include "Arduino.h"  

#else  

#include "WProgram.h"  

#endif  

#include <math.h>  

#include <stdio.h>  

#include <stdint.h>  

#include <avr/io.h>  

/*-----  

-----  

DEFINITIONS  

-----  

*/  

//$$$$$$$$$$$$$ DEBUG ENABLE SECTION $$$$$$$$$$  

// to debug ps2 controller, uncomment these two lines to print out debug to  

uart  

#define PS2X_DEBUG  

//#define PS2X_COM_DEBUG  

#define CTRL_CLK      4  

#define CTRL_BYTE_DELAY 3  

//These are our button constants  

#define PSB_SELECT      0x0001  

#define PSB_L3          0x0002  

#define PSB_R3          0x0004  

#define PSB_START        0x0008  

#define PSB_PAD_UP       0x0010  

#define PSB_PAD_RIGHT    0x0020  

#define PSB_PAD_DOWN     0x0040  

#define PSB_PAD_LEFT     0x0080  

#define PSB_L2          0x0100  

#define PSB_R2          0x0200  

#define PSB_L1          0x0400  

#define PSB_R1          0x0800  

#define PSB_GREEN        0x1000  

#define PSB_RED          0x2000  

#define PSB_BLUE         0x4000  

#define PSB_PINK         0x8000  

#define PSB_TRIANGLE     0x1000  

#define PSB_CIRCLE        0x2000  

#define PSB_CROSS         0x4000  

#define PSB_SQUARE        0x8000  

//Guitar button constants  

#define GREEN_FRET      0x0200  

#define RED_FRET         0x2000  

#define YELLOW_FRET      0x1000

```

```

#define BLUE_FRET      0x4000
#define ORANGE_FRET    0x8000
#define STAR_POWER     0x0100
#define UP_STRUM       0x0010
#define DOWN_STRUM     0x0040
#define WHAMMY_BAR     8

//These are stick values
#define PSS_RX 5
#define PSS_RY 6
#define PSS_LX 7
#define PSS LY 8

//These are analog buttons
#define PSAB_PAD_RIGHT 9
#define PSAB_PAD_UP    11
#define PSAB_PAD_DOWN  12
#define PSAB_PAD_LEFT  10
#define PSAB_L2        19
#define PSAB_R2        20
#define PSAB_L1        17
#define PSAB_R1        18
#define PSAB_GREEN     13
#define PSAB_RED       14
#define PSAB_BLUE      15
#define PSAB_PINK      16
#define PSAB_TRIANGLE   13
#define PSAB_CIRCLE    14
#define PSAB_CROSS     15
#define PSAB_SQUARE    16

#define SET(x,y) (x|=(1<<y))
#define CLR(x,y) (x&=(~(1<<y)))
#define CHK(x,y) (x & (1<<y))
#define TOG(x,y) (x^=(1<<y))

/*
-----
----- PS2X CLASS DECLARATION -----
*/
class PS2X {

public:
    boolean Button(uint16_t);
    unsigned int ButtonDataByte();
    boolean NewButtonState();
    boolean NewButtonState(unsigned int);
    boolean ButtonPressed(unsigned int);
    boolean ButtonReleased(unsigned int);
    void read_gamepad();
    void read_gamepad(boolean, byte);
    byte readType();
    byte config_gamepad(uint8_t, uint8_t, uint8_t, uint8_t);
    byte config_gamepad(uint8_t, uint8_t, uint8_t, uint8_t, bool, bool);
    void enableRumble();
    bool enablePressures();
    byte Analog(byte);

private:
    unsigned char _gamepad_shiftinout (char);
    unsigned char PS2data[21];
    void sendCommandString(byte*, byte);
    void reconfig_gamepad();
    unsigned char i;
}

```

```

    unsigned int last_buttons;
    unsigned int buttons;
    uint8_t maskToBitNum(uint8_t);
    uint8_t _clk_mask;
    volatile uint8_t *_clk_oreg;
    uint8_t _cmd_mask;
    volatile uint8_t *_cmd_oreg;
    uint8_t _att_mask;
    volatile uint8_t *_att_oreg;
    uint8_t _dat_mask;
    volatile uint8_t *_dat ireg;
    unsigned long last_read;
    byte read_delay;
    byte controller_type;
    boolean en_Rumble;
    boolean en_Pressures;

};

#endif

```

Listing 27. PS2X.lib.cpp Report c++ language technical details

```

/*
-----
----- MSc HEP 2013-2014 -----
-----

*/
/*
-----
----- Project      : Quadcopter
File          : PS2X_QadCpterController.ino
Description   : This code implements PS2 controller functionallity
Author        : Monahopoulos Konstantinos
-----
*/

/*
-----
----- Includes
-----
*/
#include "Arduino.h"
#include "PS2X.lib.h"
#include <math.h>
#include <stdio.h>
#include <stdint.h>
#include <avr/io.h>
#if ARDUINO > 22
#else
#include "WProgram.h"
#include "pins_arduino.h"
#endif

/*
-----
----- Global Variables
-----

```

```

*/
static byte enter_config[]={0x01,0x43,0x00,0x01,0x00};
static byte set_mode[]={0x01,0x44,0x00,0x01,0x03,0x00,0x00,0x00,0x00};
static byte
set_bytes_large[]={0x01,0x4F,0x00,0xFF,0x03,0x00,0x00,0x00};
static byte exit_config[]={0x01,0x43,0x00,0x00,0x5A,0x5A,0x5A,0x5A,0x5A};
static byte enable_rumble[]={0x01,0x4D,0x00,0x00,0x01};
static byte type_read[]={0x01,0x45,0x00,0x5A,0x5A,0x5A,0x5A,0x5A};

/*
-----
----- Function Prototyping -----
-----

*/
// -----
//     Constructor function
// -----

boolean PS2X::NewButtonState() {
    return ((last_buttons ^ buttons) > 0);
}

// -----
//     Constructor function
// -----

boolean PS2X::NewButtonState(unsigned int button) {
    return (((last_buttons ^ buttons) & button) > 0);
}

// -----
//     Constructor function
// -----

boolean PS2X::ButtonPressed(unsigned int button) {
    return (NewButtonState(button) & Button(button));
}

// -----
//     Constructor function
// -----

boolean PS2X::ButtonReleased(unsigned int button) {
    return ((NewButtonState(button)) & (~last_buttons & button) > 0));
}

// -----
//     Constructor function
// -----

boolean PS2X::Button(uint16_t button) {
    return ((~buttons & button) > 0);
}

// -----
//     Constructor function
// -----

unsigned int PS2X::ButtonDataByte() {
    return (~buttons);
}

// -----

```

```

//      Constructor function
// -----
byte PS2X::Analog(byte button) {
    return PS2data[button];
}

// -----
//      Constructor function
// -----

unsigned char PS2X::_gamepad_shiftinout (char byte) {
    uint8_t old_sreg = SREG;           // *** KJE *** save away the current
state of interrupts

    unsigned char tmp = 0;
    cli();                           // *** KJE *** disable for now
for(i=0;i<8;i++) {

        if(CHK(byte,i)) SET(*_cmd_oreg,_cmd_mask);
        else CLR(*_cmd_oreg,_cmd_mask);
        CLR(*_clk_oreg,_clk_mask);

        SREG = old_sreg; // *** *** KJE *** *** Interrupts may be enabled
again
        delayMicroseconds(CTRL_CLK);
        cli();           // *** KJE ***

        if(CHK(*_dat_ireg,_dat_mask)) SET(tmp,i);
        SET(*_clk_oreg,_clk_mask);
    }
    SET(*_cmd_oreg,_cmd_mask);
    SREG = old_sreg; // *** *** KJE *** *** Interrupts may be enabled again
    delayMicroseconds(CTRL_BYTE_DELAY);
    return tmp;
}

// -----
//      Constructor function
// -----

void PS2X::read_gamepad() {
    read_gamepad(false, 0x00);
}

// -----
//      Constructor function
// -----

void PS2X::read_gamepad(boolean motor1, byte motor2) {
    double temp = millis() - last_read;
    uint8_t old_sreg = SREG;           // *** KJE **** save away the current
state of interrupts - *** *** KJE *** ***

    if (temp > 1500) //waited to long
        reconfig_gamepad();

    if(temp < read_delay) //waited too short
        delay(read_delay - temp);

    last_buttons = buttons; //store the previous buttons states

    if(motor2 != 0x00)
        motor2 = map(motor2,0,255,0x40,0xFF); //noting below 40 will make it
spin

```

```

cli();      //*** KJE ***
SET(*_cmd_oreg,_cmd_mask);
SET(*_clk_oreg,_clk_mask);
CLR(*_att_oreg,_att_mask); // low enable joystick
SREG = old_sreg; // *** KJE *** - Interrupts may be enabled again

delayMicroseconds(CTRL_BYTE_DELAY);
//Send the command to send button and joystick data;
char dword[9] = {0x01,0x42,0,motor1,motor2,0,0,0,0};
byte dword2[12] = {0,0,0,0,0,0,0,0,0,0,0,0};

for (int i = 0; i<9; i++) {
    PS2data[i] = _gamepad_shiftinout(dword[i]);
}
if(PS2data[1] == 0x79) { //if controller is in full data return mode, get
the rest of data
    for (int i = 0; i<12; i++) {
        PS2data[i+9] = _gamepad_shiftinout(dword2[i]);
    }
}

cli();
SET(*_att_oreg,_att_mask); // HI disable joystick
SREG = old_sreg; // Interrupts may be enabled again

#ifdef PS2X_COM_DEBUG
Serial.println("OUT:IN");
    for(int i=0; i<9; i++){
        Serial.print(dword[i], HEX);
        Serial.print(":");
        Serial.print(PS2data[i], HEX);
        Serial.print(" ");
    }
    for (int i = 0; i<12; i++) {
        Serial.print(dword2[i], HEX);
        Serial.print(":");
        Serial.print(PS2data[i+9], HEX);
        Serial.print(" ");
    }
    Serial.println("");
#endif

buttons = *(uint16_t*) (PS2data+3); //store as one value for multiple
functions
last_read = millis();
}

// -----
// Constructor function
// -----

byte PS2X::config_gamepad(uint8_t clk, uint8_t cmd, uint8_t att, uint8_t
dat) {
    return config_gamepad(clk, cmd, att, dat, false, false);
}

// -----
// Constructor function
// -----

byte PS2X::config_gamepad(uint8_t clk, uint8_t cmd, uint8_t att, uint8_t
dat, bool pressures, bool rumble) {

    uint8_t old_sreg = SREG; // *** KJE *** save away the current
state of interrupts
    byte temp[sizeof(type_read)];

    _clk_mask = maskToBitNum(digitalPinToBitMask(clk));

```

```

_clk_oreg = portOutputRegister(digitalPinToPort(clk));
_cmd_mask = maskToBitNum(digitalPinToBitMask(cmd));
_cmd_oreg = portOutputRegister(digitalPinToPort(cmd));
_att_mask = maskToBitNum(digitalPinToBitMask(att));
_att_oreg = portOutputRegister(digitalPinToPort(att));
_dat_mask = maskToBitNum(digitalPinToBitMask(dat));
_dat_ireg = portInputRegister(digitalPinToPort(dat));

pinMode(clk, OUTPUT); //configure ports
pinMode(att, OUTPUT);
pinMode(cmd, OUTPUT);
pinMode(dat, INPUT);

digitalWrite(dat, HIGH); //enable pull-up

cli(); // *** KJE *** disable for now
SET(*_cmd_oreg, _cmd_mask); // SET(*_cmd_oreg, _cmd_mask);
SET(*_clk_oreg, _clk_mask);
SREG = old_sreg; // *** *** KJE *** *** Interrupts may be enabled again

//new error checking. First, read gamepad a few times to see if it's
talking
read_gamepad();
read_gamepad();

//see if it talked
if(PS2data[1] != 0x41 && PS2data[1] != 0x73 && PS2data[1] != 0x79){ //see
if mode came back. If still anything but 41, 73 or 79, then it's not talking
#ifndef PS2X_DEBUG
    Serial.println("Controller mode not matched or no controller
found");
    Serial.print("Expected 0x41 or 0x73, got ");
    Serial.println(PS2data[1], HEX);
#endif

return 1; //return error code 1
}

//try setting mode, increasing delays if need be.
read_delay = 1;

for(int y = 0; y <= 10; y++)
{
    sendCommandString(enter_config, sizeof(enter_config)); //start config run

    //read type
    delayMicroseconds(CTRL_BYTE_DELAY);

    cli(); // *** KJE *** disable for now
    SET(*_cmd_oreg, _cmd_mask);
    SET(*_clk_oreg, _clk_mask);
    CLR(*_att_oreg, _att_mask); // low enable joystick
    SREG = old_sreg; // *** *** KJE *** *** Interrupts may be enabled again

    delayMicroseconds(CTRL_BYTE_DELAY);

    for (int i = 0; i<9; i++) {
        temp[i] = _gamepad_shiftinout(type_read[i]);
    }

    cli(); // *** KJE *** disable for now
    SET(*_att_oreg, _att_mask); // HI disable joystick
    SREG = old_sreg; // *** *** KJE *** *** Interrupts may be enabled again

    controller_type = temp[3];

    sendCommandString(set_mode, sizeof(set_mode));
}

```

```

        if(rumble){ sendCommandString(enable_rumble, sizeof(enable_rumble));
en_Rumble = true; }
        if(pressures){ sendCommandString(set_bytes_large,
sizeof(set_bytes_large)); en_Pressures = true; }
        sendCommandString(exit_config, sizeof(exit_config));

read_gamepad();

if(pressures){
    if(PS2data[1] == 0x79)
        break;
    if(PS2data[1] == 0x73)
        return 3;
}

if(PS2data[1] == 0x73)
    break;

if(y == 10){
    #ifdef PS2X_DEBUG
    /*Serial.println("Controller not accepting commands");
    Serial.print("mode stil set at");
    Serial.println(PS2data[1], HEX);
    Serial.println("TURN ANALOG ON");*/
    #endif
    return 2; //exit function with error
}

read_delay += 1; //add 1ms to read_delay
}

return 0; //no error if here
}

// -----
//     Constructor function
// -----
void PS2X::sendCommandString(byte string[], byte len) {

    uint8_t old_sreg = SREG;           // *** KJE *** save away the current
state of interrupts

#ifdef PS2X_COM_DEBUG
byte temp[len];
cli();                                // *** KJE *** disable for now
CLR(*_att_oreg,_att_mask); // low enable joystick
SREG = old_sreg; // *** *** KJE *** *** Interrupts may be enabled again

for (int y=0; y < len; y++)
    temp[y] = _gamepad_shiftinout(string[y]);

cli();                                // *** KJE *** disable for now
SET(*_att_oreg,_att_mask); //high disable joystick
SREG = old_sreg; // *** *** KJE *** *** Interrupts may be enabled again
delay(read_delay);                    //wait a few

Serial.println("OUT:IN Configure");
for(int i=0; i<len; i++){
    Serial.print(string[i], HEX);
    Serial.print(":");
    Serial.print(temp[i], HEX);
    Serial.print(" ");
}
Serial.println("");

#else
cli();                                // *** KJE *** disable for now

```

```

CLR(*_att_oreg,_att_mask); // low enable joystick
SREG = old_sreg; // *** *** KJE *** *** Interrupts may be enabled again
for (int y=0; y < len; y++)
    _gamepad_shiftinout(string[y]);

cli(); // *** KJE *** disable for now
SET(*_att_oreg,_att_mask); //high disable joystick
SREG = old_sreg; // *** *** KJE *** *** Interrupts may be enabled again
delay(read_delay); //wait a few
#endif
}

// -----
//     Constructor function
// -----
// -----
//     Constructor function
// -----


uint8_t PS2X::maskToBitNum(uint8_t mask) {
    for (int y = 0; y < 8; y++)
    {
        if(CHK(mask,y))
            return y;
    }
    return 0;
}

// -----
//     Constructor function
// -----


byte PS2X::readType() {
/*
    byte temp[sizeof(type_read)];
    sendCommandString(enter_config, sizeof(enter_config));
    delayMicroseconds(CTRL_BYTE_DELAY);

    SET(*_cmd_oreg,_cmd_mask);
    SET(*_clk_oreg,_clk_mask);
    CLR(*_att_oreg,_att_mask); // low enable joystick

    delayMicroseconds(CTRL_BYTE_DELAY);

    for (int i = 0; i<9; i++) {
        temp[i] = _gamepad_shiftinout(type_read[i]);
    }

    sendCommandString(exit_config, sizeof(exit_config));

    if(temp[3] == 0x03)
        return 1;
    else if(temp[3] == 0x01)
        return 2;

    return 0;
*/
    if(controller_type == 0x03)
        return 1;
    else if(controller_type == 0x01)
        return 2;

    return 0;
}
}

// -----
//     Constructor function
// -----

```

```

// -----
void PS2X::enableRumble() {

    sendCommandString(enter_config, sizeof(enter_config));
    sendCommandString(enable_rumble, sizeof(enable_rumble));
    sendCommandString(exit_config, sizeof(exit_config));
    en_Rumble = true;

}

// -----
//      Constructor function
// -----
bool PS2X::enablePressures() {

    sendCommandString(enter_config, sizeof(enter_config));
    sendCommandString(set_bytes_large, sizeof(set_bytes_large));
    sendCommandString(exit_config, sizeof(exit_config));

    read_gamepad();
    read_gamepad();

    if(PS2data[1] != 0x79)
        return false;

    en_Pressures = true;
    return true;
}

// -----
//      Constructor function
// -----
void PS2X::reconfig_gamepad(){

    sendCommandString(enter_config, sizeof(enter_config));
    sendCommandString(set_mode, sizeof(set_mode));
    if (en_Rumble)
        sendCommandString(enable_rumble, sizeof(enable_rumble));
    if (en_Pressures)
        sendCommandString(set_bytes_large, sizeof(set_bytes_large));
    sendCommandString(exit_config, sizeof(exit_config));

}

```

Listing 28. PS2X_QuadCpterController.ino Report c++ language technical details

```

/*
-----
MSc HEP 2013-2014
-----
*/
/*
-----
Project      : Quadcopter
File         : PS2X_QuadCpterController.ino
Description   : This code implements PS2 controller functionallity
Author       : Monahopoulos Konstantinos
-----

```

LOOKING AT THE PLUG

```

PS PIN ->| o   o   o | o   o   o | o   o   o |
             \_____/

```

PIN # USAGE

DATA

COMMAND

N/C (9 Volts unused)

GND

VCC

ATT

CLOCK

N/C

ACK

*/

/*

Includes

*/

```

#include "PS2X_lib.h" //for v1.6

```

/*

Assign Classes

*/

```

PS2X ps2x; // create PS2 Controller Class

```

/*

Global Variables

*/

```

int error = 0;
byte type = 0;
byte vibrate[8] = {0};
int PsStatusFlag=0;
int SumOfVibration=0;

```

/*

Main Code

*/

```

void setup(){
    Serial.begin(9600);

    error = ps2x.config_gamepad(13,11,10,12, true, true); //setup pins and
settings: GamePad(clock, command, attention, data, Pressures?, Rumble?)
check for error

    if(ps2x.readType() == 1 && error == 0){

        /*If controller type is PS2 controller && controller

```

```

    configured set PsStatusFlag = 1, the quadcopter get this value in
initialization*/
    PsStatusFlag=1;
    Serial.println(PsStatusFlag);
}
}

void loop() {
    /*you should call this at least once a second*/

    if(PsStatusFlag == 1){                                // skip loop if Controller
        Status Flag=0                                    // DualShock Controller
        Configuration OK

        for(int i=0;i<8;i++){                           // This will vibrate the
            controller for any of the selected buttons pressed
            SumOfVibration+=vibrate[i];
            if(SumOfVibration>255)
                SumOfVibration=255;
        }

        ps2x.read_gamepad(false, SumOfVibration);      // read controller and set
        large motor to spin at 'vibrate' speed
        SumOfVibration=0;                             // Re-initialize

        if(ps2x.Button(PSB_PAD_UP)) {                  // will be TRUE as long as
            button is pressed
            Serial.print("UP\n");
            Serial.print(ps2x.Analog(PSAB_PAD_UP), DEC);
            Serial.print("\n");
        }
        if(ps2x.Button(PSB_PAD_RIGHT)) {
            Serial.print("RIGHT\n");
            Serial.print(ps2x.Analog(PSAB_PAD_RIGHT), DEC);
            Serial.print("\n");
        }
        if(ps2x.Button(PSB_PAD_LEFT)) {
            Serial.print("LEFT\n");
            Serial.print(ps2x.Analog(PSAB_PAD_LEFT), DEC);
            Serial.print("\n");
        }
        if(ps2x.Button(PSB_PAD_DOWN)) {
            Serial.print("DOWN\n");
            Serial.print(ps2x.Analog(PSAB_PAD_DOWN), DEC);
            Serial.print("\n");
        }

        if(ps2x.Button(PSB_PINK)){                     // will be TRUE as long as
            button is pressed
            Serial.print("SQUARE\n");
            Serial.print(ps2x.Analog(PSAB_PINK), DEC);
            Serial.print("\n");
        }

        if(ps2x.Button(PSB_GREEN)){
            Serial.print("TRIANGLE\n");
            Serial.print(ps2x.Analog(PSAB_GREEN), DEC);
            Serial.print("\n");
        }

        if(ps2x.Button(PSB_RED)){
            Serial.print("CIRCLE\n");
            Serial.print(ps2x.Analog(PSAB_RED), DEC);
            Serial.print("\n");
        }
    }
}

```

```

if(ps2x.Button(PSB_BLUE)) {
    Serial.print("X\n");
    Serial.print(ps2x.Analog(PSAB_BLUE), DEC);
    Serial.print("\n");
}

if(ps2x.Button(PSB_L1))
    Serial.print("L1\n");

if(ps2x.Button(PSB_R1))
    Serial.print("R1\n");

if(ps2x.Button(PSB_L2))
    Serial.print("L2\n");

if(ps2x.Button(PSB_R2))
    Serial.print("R2\n");

if (ps2x.NewButtonState()) // will be TRUE if any
button changes state (on to off, or off to on)
{
    if(ps2x.Button(PSB_L3))
        Serial.print("L3\n");

    if(ps2x.Button(PSB_R3))
        Serial.print("R3\n");

    if(ps2x.Button(PSB_START))
        Serial.print("START\n");

    if(ps2x.Button(PSB_SELECT))
        Serial.print("SELECT\n");
}

/*this will set the large motor vibrate speed based on
how hard you press the Quadcopter Movements.. */

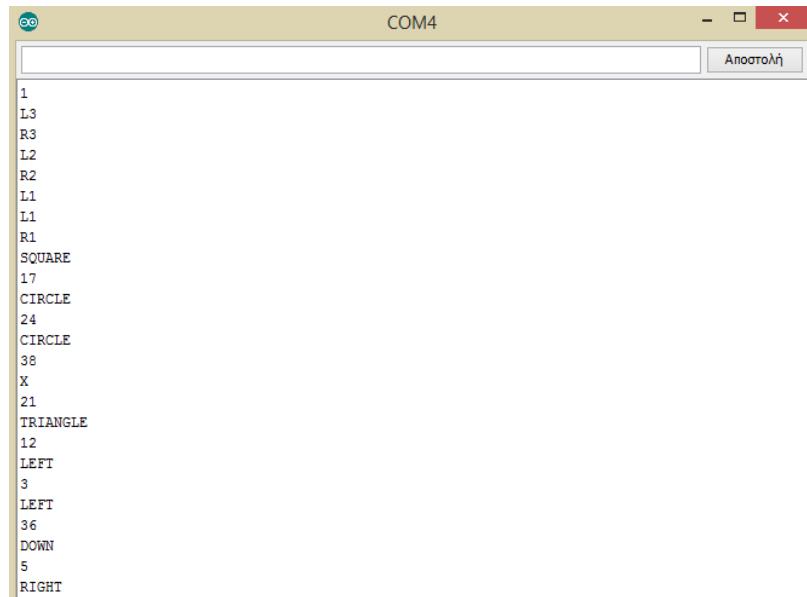
vibrate[0] = ps2x.Analog(PSAB_PAD_UP);
vibrate[1] = ps2x.Analog(PSAB_PAD_RIGHT);
vibrate[2] = ps2x.Analog(PSAB_PAD_LEFT);
vibrate[3] = ps2x.Analog(PSAB_PAD_DOWN);
vibrate[4] = ps2x.Analog(PSAB_PINK);
vibrate[5] = ps2x.Analog(PSAB_GREEN);
vibrate[6] = ps2x.Analog(PSAB_RED);
vibrate[7] = ps2x.Analog(PSAB_BLUE);

}
delay(50);
}

```

Εφόσον αναπτύξαμε τους κώδικες που υποστηρίζουν πλήρως τη λειτουργία του *PS2 controller*, παρακάτω θα δείξουμε τα αποτελέσματα που εξήχθησαν μέσω της σειριακής, δοκιμάζοντας το πάτημα τυχαίων κουμπιών του χειριστηρίου, με διαφορετικές πιέσεις πατήματος. Σε αυτό το σημείο θα πρέπει να αναφέρουμε ότι για να επικοινωνίσουμε με το *Arduino* μέσω της θύρας *USB* θα πρέπει να αλλάξουμε τη θέση του επιλογέα που βρίσκεται στο *X-bee shield* στη θέση που αναγράφει “*USB*”. Σε αντίθετη περίπτωση εφόσον φορτώσουμε το *.hex* αρχείο στη *flash* του *Arduino* ο

επιλογέας θα πρέπει να βρίσκεται στη θέση XBEE για να επικοινωνίσουμε με το quadcopter.



Εικόνα 60. Εκτύπωση αποτελεσμάτων debugging του PS2 ελεγκτή στη σειριακή οθόνη .

Τέλος θα επισυνάψουμε τη σημασιολογική έννοια των κουμπιών, θέτοντας ονομαστικά τις λειτουργίες του κάθε κουμπιού. Η ανάπτυξη αυτή πραγματοποιείται, όπως ήδη είπαμε για την ευκολία του χρήστη.



Εικόνα 61. Διαμόρφωση λειτουργίας των ενσωματωμένων κουμπιών του PS2 ελεγκτή .

Σε επόμενο το βήμα θα περιγράψουμε τον τρόπο επικοινωνίας του χειριστηρίου με το Base Station συμπεριλαμβανομένου όλων των θεωρητικών και πρακτικών μερών που

περιέχει. Θα ξεκινήσουμε την περιγραφή αναπτύσσοντας τη θεωρία του προτύπου 802.15.4 που υποστηρίζουν τα ασύρματα δίκτυα αισθητήρων και κατά ακολουθία το πρωτόκολλο *zigbee*. Υστερα θα περιγράψουμε τη διαδικασία συνδεσμολογίας που πραγματοποιήσαμε, τις αρχικοποιήσεις, τις ρυθμίσεις και τις παραμετροποιήσεις των X-bee modules, που χρησιμοποιούμε στην εφαρμογή αυτή.

5.3 ΠΡΟΤΥΠΟ IEEE 802.15.4

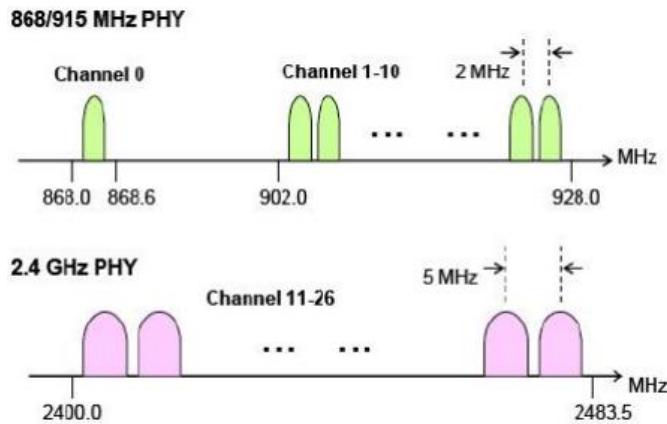
Το πρότυπο *IEEE 802.15.4* είναι μία σχετικά πρόσφατη τεχνολογία που ολοκληρώθηκε στις αρχές του 2003. Αφορά συσκευές που θα καταναλώνουν ελάχιστο ποσό ενέργειας και έχει σχεδιαστεί για να μπορεί να χρησιμοποιηθεί σε ένα ευρύ φάσμα εφαρμογών αυτοματισμού που το απαρτίζουν ακίνητες ή κινούμενες συσκευές. Το πρότυπο αυτό ορίζει το φυσικό επίπεδο (*PH-Layer*) και το επίπεδο ελέγχου πρόσβασης μέσου (*MAC-Layer*) για ασύρματα προσωπικά δίκτυα μικρής εμβέλειας και χαμηλής ταχύτητας (*LR-WPANs*).

Οι ασύρματες ζεύξεις υπό την επίβλεψη του προτύπου 802.15.4 μπορούν να λειτουργήσουν σε τρεις ζώνες συχνοτήτων με ανάλογους ρυθμούς δεδομένων. Στην Ευρώπη το πρότυπο λειτουργεί στην συχνότητα των 868 *Mhz* με ρυθμούς δεδομένων στα 20 *Kbps*, Στις ΗΠΑ στη συχνότητα των 914 *Mhz* με ρυθμό αποστολής στα 40 *Kbps* ενώ στην *ISM (Industrial Scientific Medical)* ζώνη συχνοτήτων λειτουργεί παγκοσμίως στη ζώνη των 2.4 *Ghz* με ρυθμό μετάδοσης τα 250 *Kbps*, επιπροσθέτως όλες οι μπάντες παρέχουν με τις ανάλογες διαμορφώσεις.

ΠΙΝΑΚΑΣ 9. Χαρακτηριστικά κόμβων διαφόρων σκοπών.

| | Ζώνη (Hz) | Ταχ. μετάδοσης | Διαμόρφωση |
|---------------|----------------|-----------------|------------|
| Ευρώπη | 868 <i>Mhz</i> | 20 <i>Kbps</i> | BPSK |
| ΗΠΑ | 914 <i>Mhz</i> | 40 <i>Kbps</i> | BPSK |
| ISM | 2.4 <i>Ghz</i> | 250 <i>Kbps</i> | O-QPSK |

Στο πρωτόκολλο 802.15.4 εκχωρούνται συνολικά 27 κανάλια εκ των οποίων 16 κανάλια ανήκουν στη ζώνη των 2.4 GHz , 10 κανάλια στη ζώνη των 915 MHz και 1 κανάλι στη ζώνη των 868 MHz . Η ζώνη των 2.4 GHz αποτελεί την πιο διαδεδομένη ζώνη συχνοτήτων, που είναι και η κοινή ζώνη συχνοτήτων λειτουργίας με τα υπόλοιπα ασύρματα δίκτυα άρα και επικάλυψης.

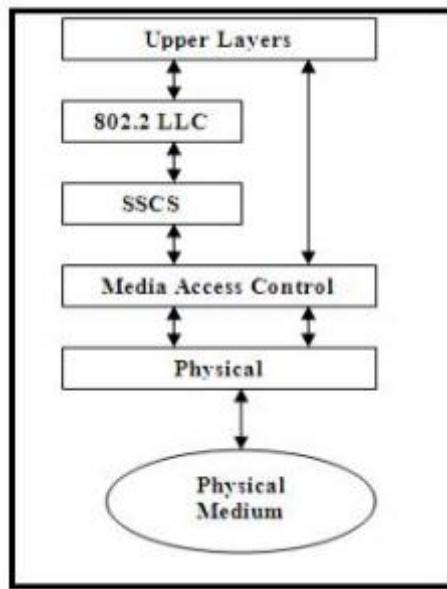


Εικόνα 62. Κανάλια συχνοτήτων μετάδοσης της οικογενείας IEEE 802.15.4.

Το IEEE 802.15.4 είναι ένα πρότυπο που ορίζει το φυσικό επίπεδο και τον έλεγχο πρόσβασης μέσου (*MAC*) για ασύρματα προσωπικά δίκτυα μικρής εμβέλειας και χαμηλής ταχύτητας (*LR-WPANs*), τροφοδοτούμενες από μπαταρίες ή κάποια άλλη πηγή περιορισμένης ενέργειας. Τα *Low-Rate WPANs* χαρακτηρίζονται από μικρό αριθμό καθηκόντων, χαμηλό κόστος, μεγάλη διάρκεια ζωής της μπαταρίας και υποστήριξη για μεγάλο αριθμό κόμβων.

Η αρχιτεκτονική κάθε *LR-WPAN* (*Low Rate – Wireless Personal Network*), στην κατηγορία των οποίων ανήκουν και τα WSNs, κατηγοριοποιείται σε μία σειρά από επίπεδα (*layers*), τα οποία διευκολύνουν τη μελέτη και το σχεδιασμό του δικτύου και προτυποποιούνται από μία σειρά πρωτοκόλλων. Αποτελείται από το φυσικό επίπεδο, το οποίο περιλαμβάνει έναν πομποδέκτη για τις ράδιο-συχνότητες μαζί με κάποιους μηχανισμούς ελέγχου χαμηλού επιπέδου, και το επίπεδο *MAC*, το οποίο παρέχει μηχανισμούς πρόσβασης στο φυσικό κανάλι, όπως το *CSMA/CA* (*Carrier Sense Multiple Access/Collision Avoidance*) για πρόσβαση στο κανάλι μέσω του φυσικού μέσου.

Η πρόσβαση στο υπο-επίπεδο *MAC* γίνεται μέσω του *Logical Link Control (LLC)* και του υπο-στρώματος σύγκλισης ειδίου ως προς την υπηρεσία (*Specific Convergence Sublayer-SSCS*). Κάθε επίπεδο επιτελεί συγκεκριμένες λειτουργίες και παρέχει υπηρεσίες μόνο στο υπερκείμενο επίπεδό του.



Εικόνα 63.. Δομή βασικών επιπέδων του πρωτοκόλλου 802.15.4

5.3.1 ΤΟ ΠΡΟΤΟΚΟΛΛΟ ZIGBEE

Πολλές φορές κυριαρχεί λανθασμένα η αντίληψη ότι το πρότυπο 802.15.4 και το *ZigBee* είναι ταυτόσημα. Το *ZigBee* αποτελεί επέκταση της στοίβας πρωτοκόλλων του 802.15.4, καθώς υλοποιεί τα επίπεδα δικτύου και εφαρμογών, βασιζόμενο στις υπηρεσίες που παρέχουν το φυσικό επίπεδο και το *MAC* υπο-επίπεδο του 802.15.4. Το πρότυπο στοχεύει στα ασύρματα προσωπικά δίκτυα και συγκεκριμένα στοχεύει στην αντικατάσταση του *Bluetooth* καθώς οι ραδιοσυσκευές του πρότυπου *Zigbee* είναι πιο φτηνές και απαιτούν περίπου το 50% του κώδικα που χρειάζεται μια συσκευή *Bluetooth* για τον έλεγχο τους (κατά συνέπεια απαιτούν λιγότερο χώρο στη μνήμη της φορητής συσκευής – γεγονός σημαντικό για το κόστος).

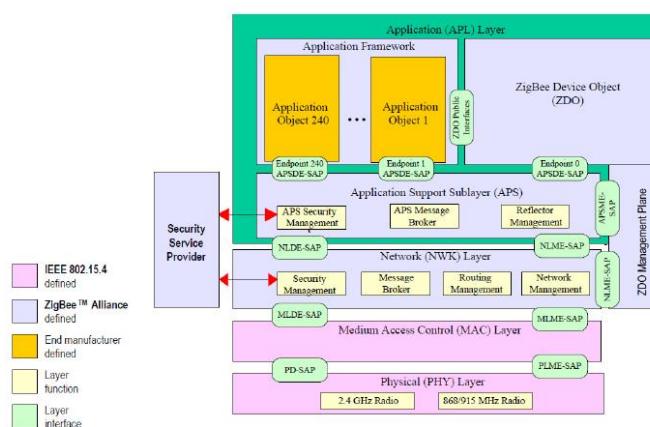
Τα βασικά χαρακτηριστικά του είναι ο χαμηλός ρυθμός μετάδοσης δεδομένων, η δυνατότητα να υποστηρίξει μέχρι 254 συσκευές σε τοπολογία αστέρα και η γρήγορη επαναφορά των συσκευών από κατάσταση *sleep*. Το δίκτυο *ZigBee* είναι πολλαπλής πρόσβασης, αφού όλες οι συσκευές έχουν ισότιμη πρόσβαση στο μέσο επικοινωνίας

και υπάρχουν δύο τύποι μηχανισμών πολλαπλής πρόσβασης. Αυτοί οι τύποι αναφέρθηκαν και στην περίπτωση του προτύπου 802.15.4 και είναι ο μηχανισμός με λειτουργία *beacon*, όπου οι συσκευές επιτρέπεται να εκπέμπουν μόνο σε προκαθορισμένες χρονοθυρίδες, και η λειτουργία *non-beacon*, στην οποία όλες οι συσκευές μπορούν να εκπέμψουν οποιαδήποτε χρονική στιγμή, εφόσον το κανάλι είναι ελεύθερο.



Εικόνα 64. Λογότυπο του ομίλου Zigbee.

Το πρωτόκολλο *Zigbee* καθορίζει τα υψηλότερα επίπεδα της στοίβας πρωτοκόλλων, από το επίπεδο δικτύου και πάνω. Η στοίβα πρωτοκόλλων του *ZigBee* βρίσκεται στο πάνω μέρος του φυσικού επιπέδου και του *MAC* υπο-επιπέδου, που έχουν καθοριστεί από το πρότυπο *IEEE 802.15.4*. Η *ZigBee Alliance* στηριζόμενη πάνω σε αυτό το πρωτόκολλο πρόσθεσε άλλα δύο επίπεδα, τα δικτύου (*NWK*) και εφαρμογής (*APL*). Το σχήμα 2.2 δείχνει σε λεπτομέρεια τα επίπεδα και τα υπό-επίπεδα τα οποία και αναλύονται στη συνέχεια της ενότητας αυτής.



Εικόνα 65. Η στοίβα πρωτοκόλλων του *ZigBee* (NWK-APS-APF).

5.3.1.1 NETWORK LAYER

To network επίπεδο αποτελεί την αλυσίδα επικοινωνίας μεταξύ του προτύπου *IEEE 802.15.4* και του προτύπου *Zigbee* καθώς εξασφαλίζει τη σωστή λειτουργία του *MAC* υπο-επιπέδου παρέχοντας κατάλληλες υπηρεσίες στο επίπεδο εφαρμογών, μέσω των μονάδων *NLDE – SAP* (*Netwotk Layer Data Entity - Service Access Point*) και *NLME – SAP* (*Netwotk Layer Management Entity - Service Access Point*). Στο επίπεδο αυτό ορίζεται η τοπολογία του δικτύου ενός (*WSN*) καθώς επίσης και η διαδικασία δρομολόγησης σε ένα *multi-hop* δίκτυο όπως αυτό. Επίσης υπάρχει η δυνατότητα εντοπισμού άλλων γειτονικών δικτύων που επικοινωνούν στην ίδια συχνότητα με αποτέλεσμα την αποφυγή συχνοτικής επικάλυψης.

Επιπρόσθετες αρμοδιότητες του επιπέδου αυτού είναι ο καθορισμός του ρόλου μίας νέας συσκευής, να δημιουργεί ένα νέο δίκτυο εφόσον η συσκευή είναι (*Full function device - FFD*), να επιτρέπει τη ενοποίηση ή την αποχώρηση μιας συσκευής σε ένα δίκτυο, να γνωρίζει τους γειτονικούς κόμβους, να κατέχει τα *routing tables* και τέλος να αλλάζει τη δρομολόγηση των δεδομένων μέσω του αλγορίθμου απόφασης για την καλύτερη διαδρομή εφόσον υπάρχει στη βάση δεδομένων της ένας πίνακας με πληροφορίες για τις συσκευές που βρίσκονται μέσα στην ακτίνα δράσης της.

5.3.1.2 APPLICATION LAYER

To *Application layer* βρίσκεται στη κορυφή του πρωτοκόλλου *Zigbee* και ως το υψηλότερο επίπεδο περιλαμβάνει και αυτό κάποια εσωτερικά επίπεδα. Αποτελείται από το *Application Framework*, τα *ZigBee Device Objects (ZDO)* και το *Application Sub Layer (APS)*. Στο *Application Framework* εδρεύουν έως και 240 διαφορετικά αντικείμενα *Application Objects (APO)* τα οποία μπορούν να οριστούν από το χρήστη και συναποτελούν την εφαρμογή που θα τρέξει σε μία *Zigbee* συσκευή.

Τα *ZigBee Device Objects (ZDO)* είναι αυτά που καθορίζουν το ρόλο της κάθε συσκευής στο δίκτυο, τον τρόπο λειτουργίας της και παρέχουν τη δυνατότητα για ανακάλυψη υπηρεσιών και συσκευών στις εφαρμογές. Τέλος το υποεπίπεδο *Application Sub Layer (APS)* προσφέρει μία διεπαφή για τη μεταφορά των δεδομένων των εφαρμογών μεταξύ δυο ή περισσοτέρων Αντικειμένων Εφαρμογών.

5.3.2 ΠΑΡΑΜΕΤΡΟΠΟΙΗΣΗ ΤΩΝ XBEE

Για να υλοποιήσουμε τη διαδικασία επικοινωνίας χρησιμοποιήσαμε δύο *Xbee* των οποίων τα χαρακτηριστικά περιγράφουμε παρακάτω. Το *Xbee XB24-Z7PIT-004* από την *Digi* αποτελεί τη δεύτερη έκδοση της σειράς, η οποία βελτιώνει την ισχύ επικοινωνίας για αποστολή και λήψη δεδομένων. Οι μονάδες της δεύτερης έκδοσης μας επιτρέπουν να δημιουργήσουμε πολύπλοκα δίκτυα "mesh" βασισμένα στο *Xbee ZigBee mesh firmware*. Οι μονάδες αυτές επιτρέπουν μια πολύ αξιόπιστη και απλή επικοινωνία μεταξύ μικροελεγκτών, υπολογιστών, συστημάτων, και οτιδήποτε κατέχει σειριακή θύρα! Υποστηρίζονται δίκτυα *point to point* το οποίο είναι και αυτό που χρησιμοποιούμε αλλά και πιο πολύπλοκα δίκτυα όπως τα *multi-point*.



Εικόνα 66. Η μονάδα *Xbee*.

ΠΙΝΑΚΑΣ 10. Τεχνικά χαρακτηριστικά του *Xbee*.

3.3V @ 40mA

250kbps Max data rate

2mW output (+3dBm)

400ft (120m) range

Built-in antenna

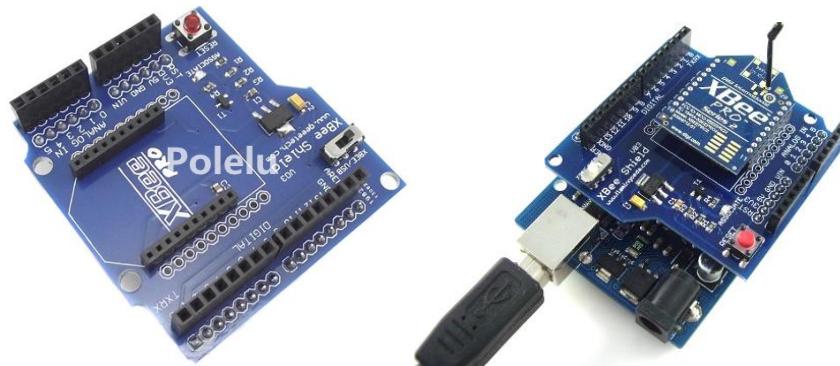
10-bit ADC input pins

8 digital IO pins

128-bit encryption

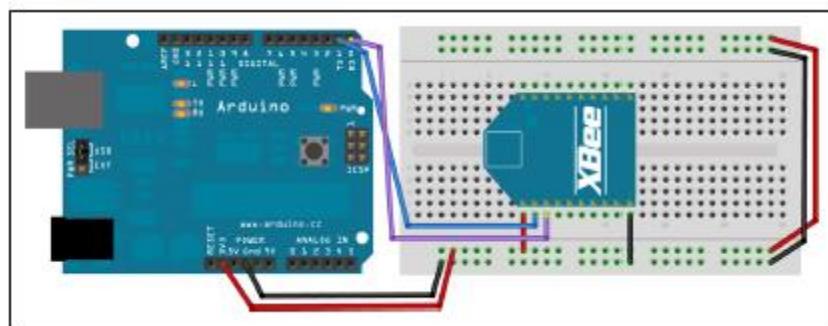
AT or API command set

Για να συνδέσουμε το *Xbee* με το *Arduino* επιλέξαμε να χρησιμοποιήσουμε ένα *Xbee shield* που μας παρέχει ευκολία τοποθέτησης του *Xbee* στο *Arduino Uno* επιτρέποντας έτσι την ασύρματη επικοινωνία μεταξύ των *Xbees* τοποθετημένων στο χώρο. Το *Xbee shield* μπορεί να λειτουργήσει τόσο με την *Version Series 1*, όσο και με την *Version Series 2.5*.



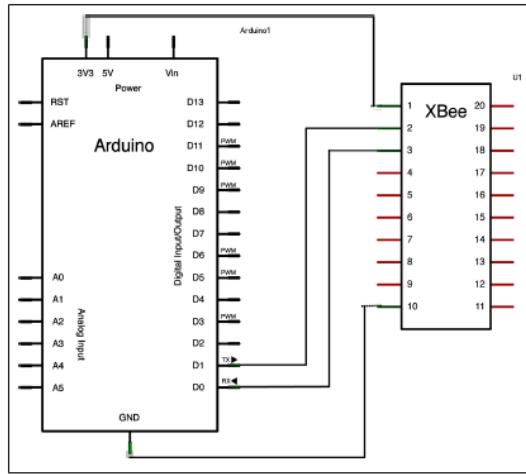
Εικόνα 67. Xbee Shield.

Παρακάτω έχουμε αναπτύξει ένα παράδειγμα συνδεσμολογίας κατά την οποία συνδέουμε το *Xbee* με το *arduino* χωρίς να μεσολαβεί κάποιο *module* παρα μόνο ένας *adapter* για τη διαφοροποίηση του πάχους των *pins* και την εγκατάσταση κατευθείαν επάνω στο *breadboard*.



Εικόνα 68. Συνδεσμολογία Xbee σε Breaboard.

Παρατηρούμε ότι η συνδεσμολογία είναι εξερετικά απλή και το μόνο που χρειάζεται είναι να συνδεθεί το *Xbee* στο *Pin* της τροφοδοσίας (3,3V) , το *Pin* της γείωσης, το *Dout* *pin* στο *Tx* του *arduino* και το *Din* *pin* στο *Rx* του *arduino* για τη σειριακή μεταφορά των δεδομένων (*Data*). Ακολουθεί το σχηματικό της συνδεσμολογίας.

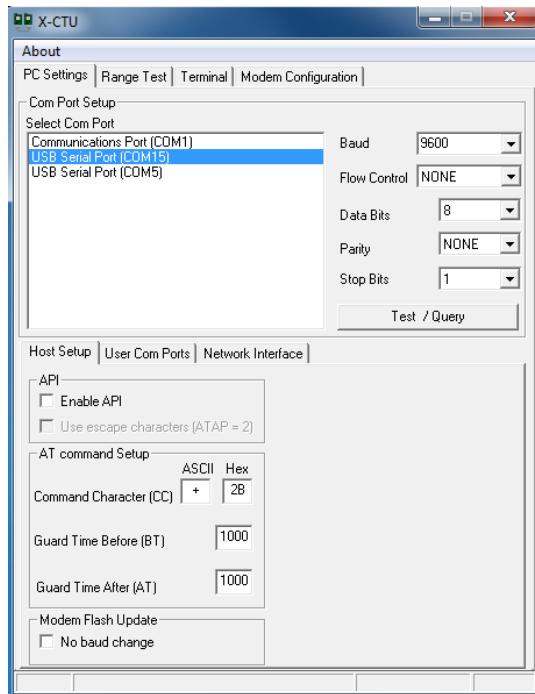


Εικόνα 69. Σχηματικό συνδεσμολογίας Xbee .

5.3.2.1 ΑΡΧΙΚΟΠΟΙΗΣΗ ΤΩΝ XBEE

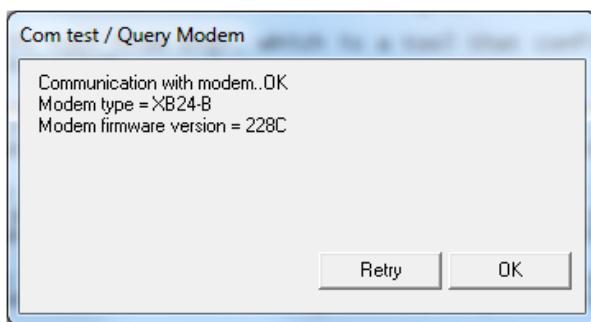
Στη συνέχεια θα παρουσιάσουμε πως μπορεί να γίνει η εγκατάσταση και η αρχικοποίηση των δύο *Xbees* ως δικτύωση *Point-to-Point*. Το *Xbee* που θα λειτουργήσει ως σταθμός βάσης θα εγκατασταθεί με τη συνδεσμολογία που αναφέραμε παραπάνω ενώ τα *Dout* και τα *Din* που αντιπροσωπεύουν τα αντίστοιχα *Tx* και *Rx* θα συνδευούν στις αντίστοιχες *Hardware Serial Ports*.

Αρχικά θα πρέπει να εγκαταστήσουμε το πρόγραμμα που θα χρησιμοποιήσουμε για την αρχικοποίηση των *Xbees*. Μέσω από μια επιλογή πολλαπλών παρεχόμενων προγραμμάτων, επιλέξαμε να λειτουργήσουμε το πρόγραμμα *X-ctu*. Μόλις εγκαταστήσουμε και τρέξουμε το πρόγραμμα *X-ctu* εμφανίζεται το παρακάτω παράθυρο.



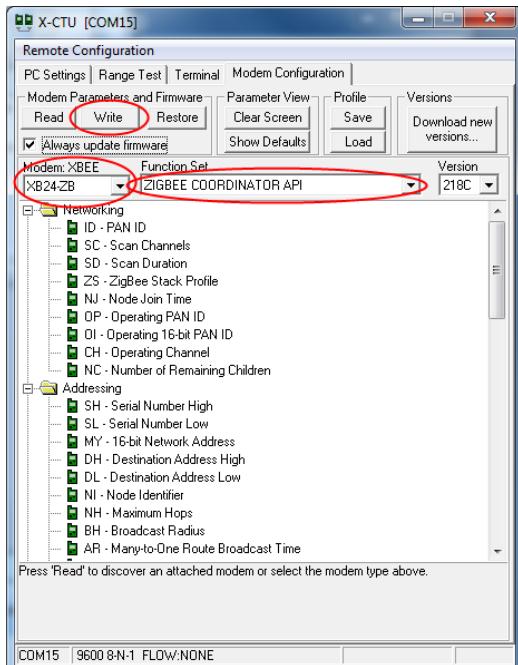
Εικόνα 70. Αναγνώριση αναπτυξιακού μέσω X-CTU .

Καθώς παρατηρούμε το *Xbee* έχει συνδεθεί μέσω της σειριακής θύρας (*Com15*) δηλαδή μέσω του *Arduino*, η επιλογή της σειριακής θύρας επιλέγεται αυτόματα και δεν έχει κάποια ιδιαίτερη σημασία. Για να αντιλειφθούμε εάν έχει γίνει σωστά η εγκατάσταση πατάμε το κουμπί *Test/Query* λαμβάνοντας το παρακάτω μήνυμα.



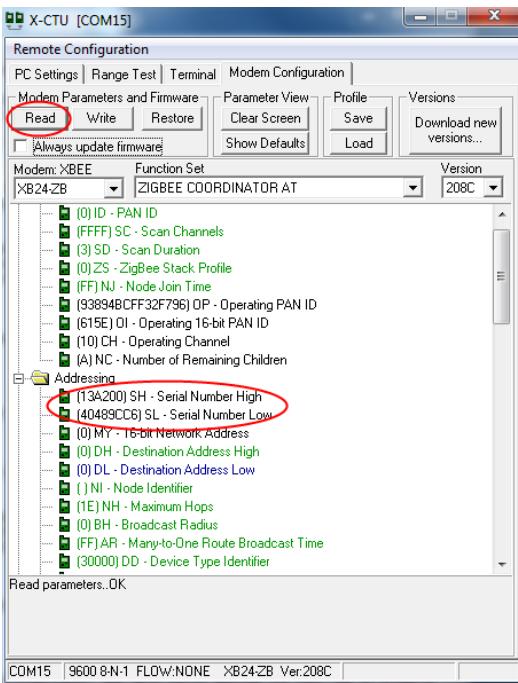
Εικόνα 71. Επιβεβαίωση αναγνώρισης αναπτυξιακού μέσω X-CTU .

Ξεκινώντας από το *X-bee* που θα λειτουργήσει ως *PAN Coordinator* στο σταθμό βάσης θα πρέπει αρχικά να αναβαθμίσουμε το *firmware* που είναι εγκατεστημένο by *default*. Πατόντας *Write* στο μενού *Modem Configuration* αφού έχουμε επιλέξει το *XB24-ZB* στο πλαίσιο *modem* : *Xbee* και τον τύπο του *Xbee* στο *function set* θα μορφοποιηθεί η παρακάτω εικόνα.



Εικόνα 72. Λιαμόρφωση Xbee μέσω X-CTU.

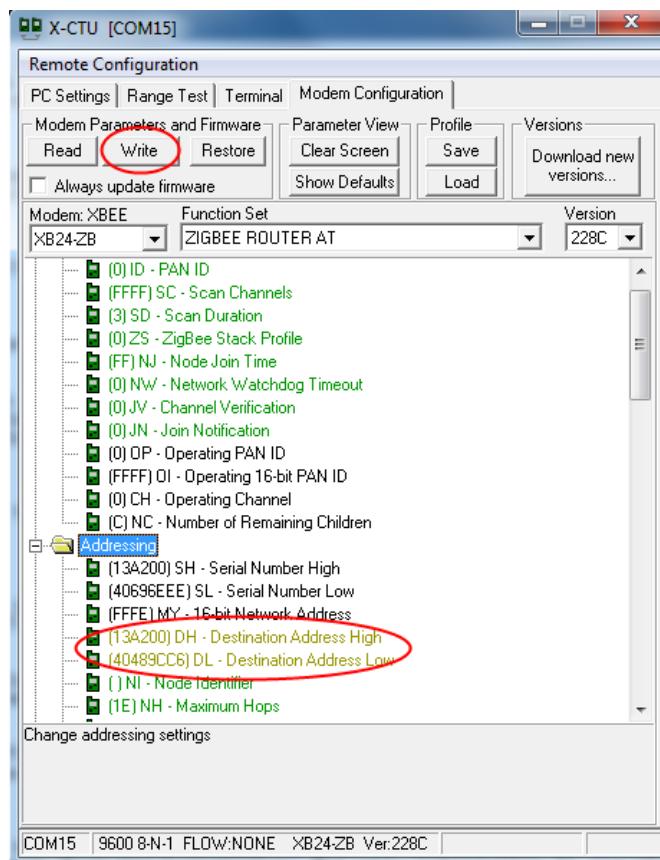
Όταν τελειώσει η αναβάθμιση του firmware, Πατάμε το κουμπί Read για να εισέλθουμε στο configuration του Xbee παράγοντας τη παρακάτω μορφή.



Εικόνα 73. Αναγνώριση διεύθυνσης Xbee μέσω X-CTU.

Οι διευθύνσεις στα πλαίσια *Serial number low* και *Serial number high* που έχουμε σημειώσει με κόκκινο είναι οι διευθύνσεις του *Coordinator* και που πρέπει να γνωρίζουν οι συσκευές (*Xbees*) για να επικοινωνίσουν σε μία *point to point* δικτύωση.

Εγκατιστόντας το δέυτερο *Xbee*, προς αρχικοποίηση επαναλαμβάνουμε την ίδια διαδικασία της αναβάθμισης του *firmware* μόνο που αυτή τη φορά επιλέγουμε στο πλαίσιο *function set* την επιλογή *Zigbee Router At*. Συνεχίζουμε διαβάζοντας το configuration του *Xbee* και τοποθετώντας την διεύθυνση που καταγράψαμε από τον coordinator στα παιδία *Serial number low* και *Serial number high* παίρνοντας την παρακάτω εικόνα.



Εικόνα 74. Τοποθέτηση διεύθυνσης *Xbee* μέσω *X-CTU*.

Με αυτή τη παραπάνω απλή διαδικασία μπορούμε να αρχικοποιήσουμε δύο *Xbees* τα οποία θα επικοινωνούν απευθείας μεταξύ τους. Να σημειωθεί ότι η διαδικασία υλοποίησης πιο περίπλοκων δικτυακών μορφολογιών είναι πολύ πιο σύνθετη διαδικασία.

ΚΕΦΑΛΑΙΟ 6: ΣΥΜΠΕΡΑΣΜΑΤΑ ΚΑΙ ΜΕΛΛΟΝΤΙΚΗ ΕΡΓΑΣΙΑ

6.1 ΣΥΜΠΕΡΑΣΜΑΤΑ

Η διαδικασία κατασκευής ενός *Quadcopter* από την αρχή αποτελεί μια αρκετά επίπονη, χρονοβόρα και πολυέξοδη διαδικασία η οποία απαιτεί αρκετές γνώσεις των φυσικών μεγεθών που διέπουν τέτοιου είδους συστήματα αλλά και πολύ καλές γνώσεις προγραμματίσουν. Η γλώσσα που επιλέχτηκε για την υλοποίηση του αλγορίθμου είναι η C++ μέσω της οποίας καταφέραμε και αναπτύξαμε την επιθυμητή λογική εκτέλεσης, ενώ ο συνολικός αριθμός γραμμών κώδικα που αναπτύξαμε σε όλη τη διαδικασία (*test – calibrate – use*) για όλα τα components του *Quadcopter* είναι 2000 γραμμές.

Η διαδικασία όλων των υπολογισμών που πραγματοποιεί σε κάθε κύκλο εκτέλεσης το Arduino Uno είναι αρκετά πολύπλοκή, παρόλα αυτά ο μικροελεγκτής καταφέρνει και ανταποκρίνεται στα απαιτούμενα που του έχουμε θέσει σε ικανοποιητικό χρόνο. Μέρος των υπολογισμών αποτελούν οι γωνίες κλήσης χρησιμοποιώντας τον αντίστοιχο αισθητήρα και τα σφάλματα που προκύπτουν σε κάθε άξονα μέσω της εκτέλεσης τεσσάρων *PID* ελεγκτών. Επιπροσθέτως, σε κάθε βρόγχο επανάληψης ελέγχεται μέσω της ασύρματης επικοινωνίας εντολοδοτήσεις που δέχεται από το *PS2 controller* ανταποκρίνοντας κατάλληλα στην εντολή αυτή.

Κατά τη διαδικασία των δοκιμών πριν τολμήσουμε κάποια πτήση, ελέγξαμε την ισορροπία του *Quadcopter* στις κινήσεις *Roll* και *Pitch* σε ελεγχόμενο περιβάλλον. Το γεγονός αυτό σημαίνει ότι για την εύρεση των κατάλληλων P,I και D τιμών για την ισορροπία της αντίστοιχης κίνησης επιτρέπαμε ελεγχόμενα τη κίνηση μόνο σε αυτό τον άξονα που αποτελεί και τον άξονα δοκιμής. Με τον τρόπο αυτό καταφέραμε και βρήκαμε τις κατάλληλες τιμές ισορροπίας του *Quadcopter*. Υστερα από την εφαρμογή των τιμών αυτών το *Quadcopter* είναι σε θέση να διορθώνει αρκετά γρήγορα το σφάλμα κλίσης και να αντιστέκεται σε εξωτερικές δυνάμεις χωρίς ιδιαίτερο *Overshooting*. Οι αντιπροσωπευτικές τιμές που αφορούν την ισσοροπία του *Yaw* βρίσκονται πειραματικά όταν το *Quadcopter* θα είναι στον αέρα χωρίς να δεσμεύεται από εφωτερικές συσιστώσες, έτσι ώστε να υπάρχει δυνατότητα περιστροφής γύρω από τον άξονά του.

Έγινε από την επιτυχημένη εύρεση των τιμών ισσοροπίας του *Quadcopter* προχωρήσαμε σε μία δοκιμαστική πτήση στην οποία αντιμετωπίσαμε επιπρόσθετα προβλήματα. Μέχρι και σε αυτή τη φάση υλοποίησης του *Quadcopter* δεν έχουμε καταφέρει μια ολοκληρωμένη επιτυχημένη πτήση, διότι αντιμετωπίζουμε προβλήματα κατά την απογείωση. Για το λόγο αυτό έχουμε επιλέξει να ενεργοποιούμε και να απενεργοποιούμε την εκτέλεση των *PID* ελεγκτών μόνο όταν το *Quadcopter* απογειώνεται ή αντίστοιχα προσγειώνεται από/προς το έδαφος. Η αλλαγή αυτή μας επέτρεψε μέχρι ένα σημείο την απογείωση του *Quadcopter* αλλα χρειάζεται επιπλέον ερευνητική δραστηριότητα για την αντιμετώπιση των προβλημάτων, έτσι ώστε να ισχυρηστούμε μια επιτυχημένη πτήση.



Εικόνα 75. Διαδικασία δοκιμών εύρεσης ισσοροπίας του *Quadcopter* (1/2).



Εικόνα 76. Διαδικασία δοκιμών εύρεσης ισσοροπίας του *Quadcopter* (2/2).

6.2 ΜΕΛΛΟΝΤΙΚΗ ΕΡΓΑΣΙΑ

Στα μελλοντικά σχέδια για την ολοκλήρωση του *Project* χρειάζεται πρώτα να αντιμετωπιστεί το πρόβλημα της απογείωσης του *Quadcopter*. Εφόσον πραγματοποιηθεί αυτό θα πρέπει να αντιληφθούμε πως ανταποκρίνεται το *Quadcopter* στις κινήσεις εντολοδότησης μέσω του χειριστηρίου και υλοποίηση τυχόν μικροδιορθώσεων όσο αυτό θα βρίσκεται στον αέρα.

Στο λειτουργικό κομμάτι, όσον αφορά τις υποστηριζόμενες λειτουργίες του *Quadcopter* θα πρέπει να ενσωματώσουμε έναν αλγόριθμο υπολογισμού εύρους ανάμεσα στα *Xbee*, ο οποίος θα εκτελείται σε συγκεκριμένο ρυθμό και θα υπολογίζει την ισχύ του σήματος του χειριστηρίου με της βάσης. Επιπροσθέτως θα πρέπει να αξιοποιήσουμε την λειτουργία του βαρομέτρου για ενσωμάτωση της λειτουργίας *Hovering* στην οποία θα ισορροπεί το *Quadcopter* σε ένα συγκεκριμένο ύψος. Τέλος απαιτείται η εύρεση αλγορίθμου όσον αφορά το *Acrobatic Mode* που θα υποστηρίζει μέσω του οποίου θα είμαστε σε θέση να εκτελέσουμε απότομες μανούβρες.

Εφόσον υλοποιηθούν τα παραπάνω και εφόσον έχουμε ήδη εγκαταστήσει βάση κάμερας την οποία είμαστε σε θέση να ελέγχουμε, θα μπορούσαμε να εγκαταστήσουμε μία *FPV* κάμερα για λήψη εναέριων βίντεο.

ΑΝΑΦΟΡΕΣ

- [Ref 1]
http://blog.oscarliang.net/types-of-multicopter/
- [Ref 2]
http://blog.tkjelectronics.dk/2012/03/quadcopters-how-to-get-started/
- [Ref 3]
http://robots.dacloughb.com/project-3/motor-orientation/
- [Ref 4]
http://el.wikipedia.org/wiki/Arduino
- [Ref 5]
http://blog.oscarliang.net/how-to-choose-motor-and-propeller-for-quadcopter/
- [Ref 6]
http://flitetest.com/articles/how-to-simple-propeller-balancing
- [Ref 7]
http://blog.tkjelectronics.dk/2012/03/quadcopters-how-to-get-started/
- [Ref 8]
http://techvalleyprojects.blogspot.gr/2012/06/arduino-control-escmotor-tutorial.html
- [Ref 9]
http://robots.dacloughb.com/project-2/esc-calibration-programming/
- [Ref 10]
http://en.wikipedia.org/wiki/Lithium_polymer_battery
- [Ref 11]
Reference -> http://www.rchelicopterfun.com/rc-lipo-batteries.html

- [Ref 12]
<http://www.roadtovr.com/oculus-rift-headtracker-adjacent-reality/>
- [Ref 13]
<http://playground.arduino.cc/Main/I2cScanner>
- [Ref 14]
<http://invensense.com/mems/gyro/mpu6000.html>
- [Ref 15]
<http://playground.arduino.cc/Main/MPU-6050>
- [Ref 16]
<http://www.livescience.com/40103-accelerometer-vs-gyroscope.html>
- [Ref 17]
<http://www.instructables.com/id/Accelerometer-Gyro-Tutorial/?ALLSTEPS>
- [Ref 18]
<http://www.livescience.com/40103-accelerometer-vs-gyroscope.html>
- [Ref 17]
<http://www.instructables.com/id/Accelerometer-Gyro-Tutorial/?ALLSTEPS>
- [Ref 18]
<http://www.livescience.com/40103-accelerometer-vs-gyroscope.html>
- [Ref 19]
<http://www.instructables.com/id/Accelerometer-Gyro-Tutorial/?ALLSTEPS>
- [Ref 20]
<http://www.livescience.com/40103-accelerometer-vs-gyroscope.html>
- [Ref 21]
https://www.sparkfun.com/pages/accl_gyro_guide

- [Ref 22]
<http://theboredengineers.com/2012/09/the-quadcopter-get-its-orientation-from-sensors/>
- [Ref 23]
<http://hobbylogs.me.pn/?p=47>
- [Ref 24]
http://www.adafruit.com/datasheets/HMC5883L_3-Axis_Digital_Compass_IC.pdf
- [Ref 25]
<https://github.com/kriswiner/MPU-6050/wiki/Affordable-9-DoF-Sensor-Fusion>
- [Ref 26]
<http://hobbylogs.me.pn/?p=17>
- [Ref 27]
<http://www.jarzebski.pl/arduino/czujniki-i-sensory/czujnik-cisnienia-i-temperatury-ms5611.html>