ΑΝΩΤΑΤΟ ΤΕΧΝΟΛΟΓΙΚΟ ΙΔΡΥΜΑ ΠΕΙΡΑΙΑ
ΣΧΟΛΕΣ ΤΕΧΝΟΛΟΓΙΚΩΝ ΕΦΑΡΜΟΓΩΝ
ΤΜΗΜΑ ΑΥΤΟΜΑΤΙΣΜΟΥ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

«Έλεγχος μαριονέτας με φυσική αλληλεπίδραση χρήστη»

ΑΡΓΥΡΙΟΥ ΕΜΜΑΝΟΥΗΛ

Α.Μ. 32798

ΕΠΙΒΛΕΠΩΝΤΕΣ: ΝΙΚΟΛΑΟΥ ΓΡΗΓΟΡΗΣ,
ΑΛΑΦΟΔΗΜΟΣ ΚΩΝΣΤΑΝΤΙΝΟΣ

ΑΘΗΝΑ, ΙΟΥΝΙΟΣ 2013

Πρόλογος

Θα ήθελα να ευχαριστήσω θερμά τον κο. Γρηγόρη Νικολάου για την ελευθερία στην επιλογή του θέματος αυτού και την αποδοχή του, αλλά κυρίως για την καθοδήγηση, την πολύτιμη βοήθεια και την διαθεσιμότητα του κατά την διάρκεια της εκπόνησης της παρούσας πτυχιακής εργασίας. Επίσης τους καθηγητές κο. Καλλιγερόπουλο, κο. Σύρκο και κο. Αλαφοδήμο για την διδασκαλία τους στα αντίστοιχα μαθήματα τους, τα όποια αποτελούν την ραχοκοκαλιά των συστημάτων αυτομάτου ελέγχου. Ιδιαίτερη αναφορά στο εργαστήριο Μηχατρονικής και στον τρόπο λειτουργίας του, με την εξαμηνιαία εργασία του, να αποτελεί την καλύτερη προετοιμασία για τις προκλήσεις που θα αντιμετώπιζα στην εκπόνηση της πτυχιακής εργασίας.
Ευχαριστώ τους γονείς μου και την αδερφή μου, για την υπομονή και την συμπαράσταση τους και τους φίλους μου για την πολύτιμη στήριξη και βοήθεια τους.

Πίνακας Περιεχομένων

3. Processing

Εισαγωγή

Η παρούσα πτυχιακή εργασία αφορά την σχεδίαση και τη κατασκευή συστήματος ελέγχου μιας μαριονέτας, με φυσική αλληλεπίδραση χρήστη. Όταν λέμε φυσική αλληλεπίδραση, εννοούμε την χρήση κάποιων μελών του σώματος μας, για επίτευξη μιας πράξης. Ειδικότερα, η μαριονέτα θα κινείται όπως κινείται ο χρήστης μπροστά από το Kinect, δηλαδή θα αντιγράφονται οι κινήσεις του και θα καθρεφτίζονται στην μαριονέτα.

Πιο συγκεκριμένα ο αισθητήρας Kinect, με όλες τις δυνατότητες που προσφέρει, θα καλύψει το κομμάτι της φυσικής αλληλεπίδρασης, σε συνεργασία με τον μικροελεγκτή Leonardo, της μεγάλης οικογένειας των μικροελεγκτών Arduino, που θα προσδώσει κίνηση στην μαριονέτα. Το περιβάλλον προγραμματισμού, που συνδυάζει αρμονικά τα παραπάνω και θα στεγάσει την εργασία, είναι το Processing IDE.

Σκοπός της εργασίας είναι η επίδειξη των δυνατοτήτων των τεχνολογιών που χρησιμοποιούνται και πως αυτές μπορούν να αποτελέσουν πηγές καινοτόμων ιδεών και υλοποιήσεων στο σύντομο μέλλον.

Κεφάλαιο 1ο

Arduino

1.1 Γενικά

Το Arduino είναι μία πλατφόρμα ηλεκτρονικών πρωτοτύπων ανοιχτού κώδικα, που αποτελείται από έναν μικροελεγκτή, μια γλώσσα προγραμματισμού και ένα ολοκληρωμένο περιβάλλον ανάπτυξης. Πρόκειται για ένα εργαλείο το οποίο δημιουργεί διαδραστικές εφαρμογές, σχεδιασμένο να απλοποιήσει το έργο αυτό για τους αρχάριους, που ταυτόχρονα όμως παραμένει αρκετά ευέλικτο για τους γνώστες, ώστε να αναπτύξουν πιο περίπλοκα project.

Από την έναρξη του το 2005, έχουν πωληθεί περισσότερες από 300.000 πλακέτες Arduino και υπάρχει ένας αυξανόμενος αριθμός από project, που χρησιμοποιούν το Arduino ως επεξεργαστικό τους πυρήνα.

Η κοινότητα του Arduino είναι τεράστια, συνεπώς υπάρχει πληθώρα από tutorials, βιβλία αναφοράς και βιβλιοθήκες. Οι προχωρημένοι χρήστες του συνεχώς αναπτύσσουν “ασπίδες” για τις διαθέσιμες πλακέτες Arduino, όπως επίσης εντελώς νέες πλακέτες συμβατές με το Arduino για ειδικευμένες εφαρμογές.

Η επιτηδευμένη απλότητα της πλατφόρμας Arduino έχει επιτρέψει την είσοδο στο physical computing για ανθρώπους, που δεν είχαν σκεφτεί ποτέ τη χρήση ή τον προγραμματισμό ενός μικροελεγκτή, πριν από την Arduino/Wiring εποχή.

1.2 Σύντομη ιστορία

Το Arduino γεννήθηκε το 2005 στο Ινστιτούτο Διαδραστικού Σχεδιασμού στην Ιβρέα της Ιταλίας, σαν διακλάδωση της πλατφόρμας ανοιχτού κώδικα Wiring. Οι ιδρυτές Massimo Banzi και David Cuartielles έδωσαν το όνομα Arduino από το κεντρικό ιστορικό πρόσωπο της πόλης τον Arduin της Ιβρέας.

Ο Hernando Barragan, φοιτητής στο παραπάνω Ινστιτούτο, μαζί με τον Diego Gonzalez Joven ανέπτυξαν την πλατφόρμα Wiring το 2003 για την διατριβή τους, με επιβλέπων τον Massimo Banzi και τον Casey Reas (ένας από τους μυητές του ολοκληρωμένου περιβάλλον ανάπτυξης Processing). Η ιδέα πίσω από την πλατφόρμα Wiring ήταν να επιτρέψουν την εύκολη εισαγωγή στον προγραμματισμό και τα σκίτσα με ηλεκτρονικά για καλλιτέχνες και σχεδιαστές. Η ίδια περίπου ιδέα κρύβεται πίσω από το ολοκληρωμένο περιβάλλον ανάπτυξης Processing των Casey Reas και Ben Fry.

Το Arduino σχεδιάστηκε γύρω από το Wiring αλλά αναπτύχθηκε ανεξάρτητα από το 2005. Οι δύο πλατφόρμες χρησιμοποιούνται ευρέως και οτιδήποτε υλοποιείται σε αυτήν την εργασία ισχύει και για την πλατφόρμα Wiring.

1.3 Εγκατάσταση και δοκιμή

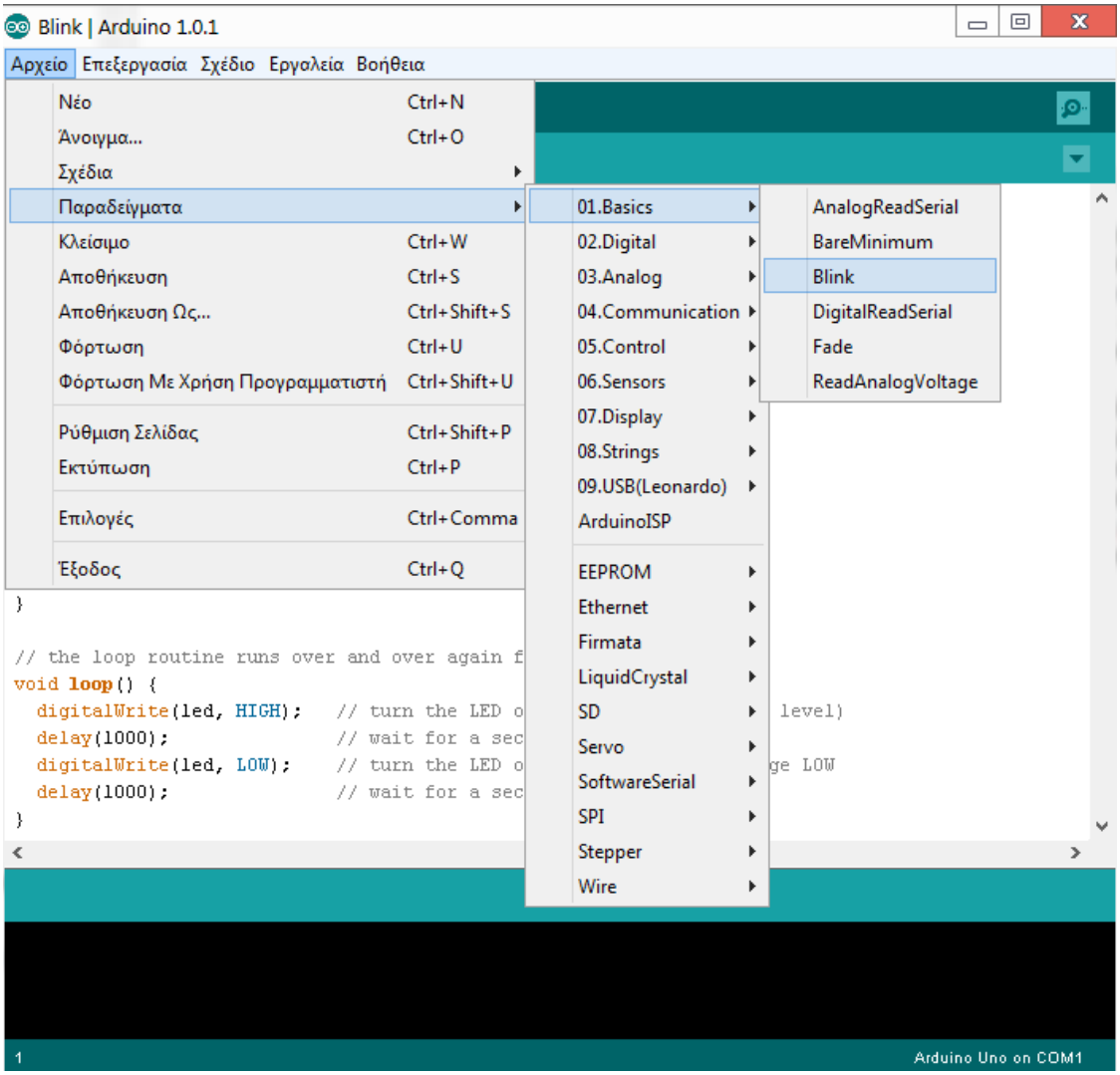
Πρώτο βήμα είναι η απόκτηση μιας πλακέτας Arduino και ενός καλωδίου USB. Το Arduino υποστηρίζει Windows, Mac OS X και Linux. Στο http://arduino.cc/en/Main/Software υπάρχει η κατάλληλη έκδοση για το αντίστοιχο λειτουργικό σύστημα.

Μετά την εγκατάσταση του λογισμικού και των drivers διενεργούμε ένα μικρό τεστ, προκειμένου να διαπιστώσουμε ότι όλα δουλεύουν σωστά.

Η πλακέτα έχει ένα ενσωματωμένο LED συνδεδεμένο στον ακροδέκτη 13, για να έχουμε την δυνατότητα να την δοκιμάσουμε χωρίς να συνδέσουμε επιπλέον εξαρτήματα. Ανάμεσα στα πολλά παραδείγματα, που υπάρχουν στο ολοκληρωμένο περιβάλλον ανάπτυξης του Arduino, υπάρχει ένα το οποίο ονομάζεται Blink και κάνει το προαναφερθέν LED να αναβοσβήνει κάθε δευτερόλεπτο.

Συνδέουμε την πλακέτα και εκτελούμε το λογισμικό. Στο μενού Εργαλεία, επιλέγουμε την πλακέτα που έχουμε. Μετά, πάλι στο μενού Εργαλεία επιλέγουμε την σειριακή θύρα που είναι συνδεδεμένη η πλακέτα. Στο μενού Αρχείο πάμε στα Παραδείγματα στη κατηγορία 1.Basics και επιλέγουμε το Blink (βλέπε εικόνα 1.1). Έπειτα, κάνουμε «Φόρτωση» πατώντας το αντίστοιχο κουμπί. Μετά από μερικά δευτερόλεπτα θα εμφανιστεί το μήνυμα «Ολοκλήρωση Φόρτωσης» και το LED θα αρχίσει να αναβοσβήνει.

Αν το παραπάνω παράδειγμα λειτουργήσει τότε σημαίνει ότι η πλακέτα είναι σωστά συνδεδεμένη και ρυθμισμένη.

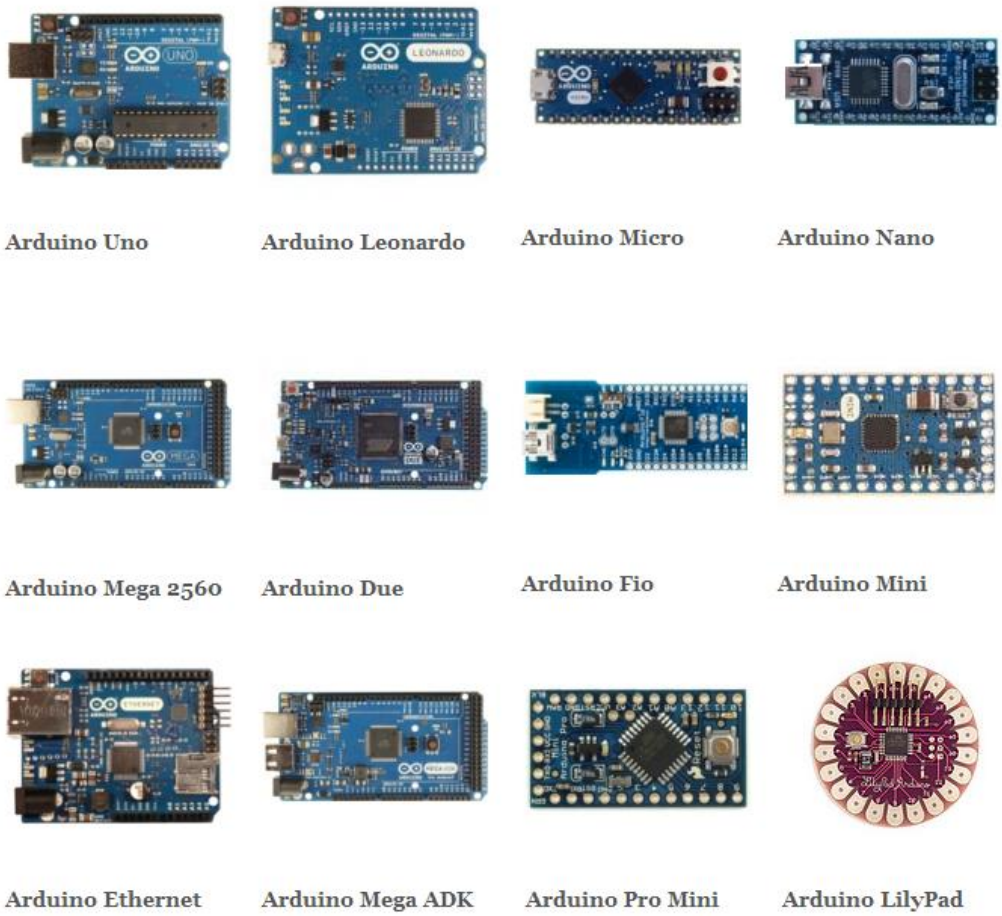


Εικόνα 1.1: Άνοιγμα του παραδείγματος Blink.

1.4 Πλακέτες Arduino

Μπορεί κανείς να σκεφτεί το Arduino σαν έναν μικρό εγκέφαλο που υποστηρίζει την σύνδεση πλήθους αισθητήρων και μηχανισμών κίνησης. Η πλακέτα βασίζεται σε έναν 8-bit Atmel AVR μικροελεγκτή. Ανάλογα με την πλακέτα υπάρχουν διάφορα chip, συμπεριλαμβανομένου του ATmega8, ATmega 168, ATmega 328, ATmega 1280 και ATmega 2560. Η πλακέτα του Arduino εκθέτει τους ακροδέκτες εισόδου και εξόδου του μικροελεγκτή, προκειμένου να χρησιμοποιηθούν σαν είσοδοι και έξοδοι για άλλα κυκλώματα σχηματισμένα γύρω από το Arduino.

Στη σελίδα http://arduino.cc/en/Main/Products υπάρχουν όλες οι διαφορετικές εκδοχές των πλακετών Arduino (βλέπε εικόνα 1.2).



Εικόνα 1.2: Οι επίσημες πλακέτες Arduino.

Κάθε μία έχει σχεδιαστεί για συγκεκριμένο σκοπό οπότε διαφέρουν μεταξύ τους στον αριθμό των ακροδεκτών και των εξαρτημάτων που ενσωματώνουν.
Μέχρι πριν λίγους μήνες το Arduino Uno ήταν η απλούστερη πλακέτα της σειράς, αλλά αντικαταστάθηκε από το Arduino Leonardo. Στη συγκεκριμένη εργασία χρησιμοποιείται το Leonardo.

1.5 Arduino Leonardo

Το Arduino Leonardo βασίζεται στον μικροελεγκτή ATmega32u4. Έχει 20 ψηφιακές εισόδους/εξόδους, ένα κρυσταλλικό ταλαντωτή στα 16MHz, σύνδεση

micro-USB, υποδοχή ρεύματος, κεφαλή ICSP και κουμπί reset. Περιέχει όλα όσα χρειάζονται για την υποστήριξη του μικροελεγκτή και απλά αρκεί η σύνδεση με USB καλώδιο για να είναι έτοιμο.

Μικροελεγκτής	ATmega32u4
Τάση λειτουργίας	5V
Τάση εισόδου (συνιστώμενη)	7-12V
Τάση εισόδου (όρια)	6-20V
Ψηφιακοί ακροδέκτες εισόδου/εξόδου	20
Κανάλια διαμόρφωσης εύρους παλμού	7
Κανάλια αναλογικής εισόδου	12
Ρεύμα DC ανά ακροδέκτη	40mA
Ρεύμα DC για τον ακροδέκτη 3,3V	50mA
Μνήμη Flash	32KB εκ των οποίων 4KB χρησιμοποιούνται από τον bootloader
EEPROM	2,5KB
SRAM	1KB
Ταχύτητα ρολογιού	16MHz

Πίνακας 1. Τεχνικά χαρακτηριστικά του Arduino Leonardo.

Γενικά, ο προγραμματισμός και η χρήση του Leonardo μοιάζει με τις άλλες πλακέτες Arduino. Υπάρχουν, ωστόσο, μερικές σημαντικές διαφορές.

Το Leonardo χρησιμοποιεί έναν μικροελεγκτή και για την εκτέλεση των σκίτσων και για την επικοινωνία μέσω USB με τον Η/Υ. Οι άλλες πλακέτες χρησιμοποιούν ξεχωριστούς μικροελεγκτές για αυτές τις δύο λειτουργίες, που σημαίνει ότι η επικοινωνία με τον Η/Υ παραμένει, άσχετα από την κατάσταση του κεντρικού μικροελεγκτή. Συνδυάζοντας τις δύο λειτουργίες αυτές σε έναν μικροελεγκτή, το Leonardo επιτρέπει περισσότερη ευελιξία στην επικοινωνία του με τον Η/Υ. Επίσης, βοηθάει στην μείωση του κόστους της πλακέτας.

Αφού η πλακέτα δεν έχει ολοκληρωμένο αφιερωμένο στη σειριακή επικοινωνία, σημαίνει ότι η σειριακή θύρα είναι εικονική, μία ρουτίνα του λογισμικού, στο λειτουργικό σύστημα όπως και στην πλακέτα. Έτσι όπως ο Η/Υ δημιουργεί υπόδειγμα του οδηγού της σειριακής θύρας όταν συνδέεται το Arduino, έτσι και το Leonardo δημιουργεί σειριακό υπόδειγμα όποτε εκτελείται ο bootloader του. Η πλακέτα είναι υπόδειγμα του USB CDC (Connected Device Class – Κλάση Συνδεδεμένης Συσκευής) οδηγού. Αυτό σημαίνει ότι κάθε φορά που γίνεται reset της πλακέτας, η σειριακή επικοινωνία USB διακόπτεται και αποκαθίσταται. Η πλακέτα εξαφανίζεται από την λίστα σειριακών συσκευών και η λίστα επαναριθμείται. Όποιο πρόγραμμα έχει ανοιχτή σειριακή επικοινωνία με το Leonardo θα χάσει τη σύνδεση του. Αυτό έρχεται σε αντίθεση με άλλες πλακέτες, στις όποιες όταν γίνεται reset δεν διακόπτεται η επικοινωνία με τον Η/Υ. Αυτή η διαφορά έχει επιπτώσεις στην εγκατάσταση του οδηγού, στη φόρτωση και στην επικοινωνία.

Αντίθετα με το Uno, στο Leonardo δεν γίνεται επανεκκίνηση του σκίτσου όταν ανοίγει μία σειριακή θύρα στον Η/Υ. Αυτό σημαίνει ότι δεν φαίνονται τα σειριακά δεδομένα που αποστέλλονται ήδη στον υπολογιστή από την πλακέτα, περιλαμβάνοντας για παράδειγμα τα δεδομένα που αποστέλλονται στην συνάρτηση `setup()`. Αυτή η αλλαγή σημαίνει ότι αν χρησιμοποιούνται οποιεσδήποτε σειριακές εντολές (`print()`, `println()`, `write()`), η έξοδος τους δεν θα φαίνεται στο σειριακό μόνιτορ. Για να επιλυθεί αυτό, ελέγχουμε αν η σειριακή θύρα είναι ανοικτή:

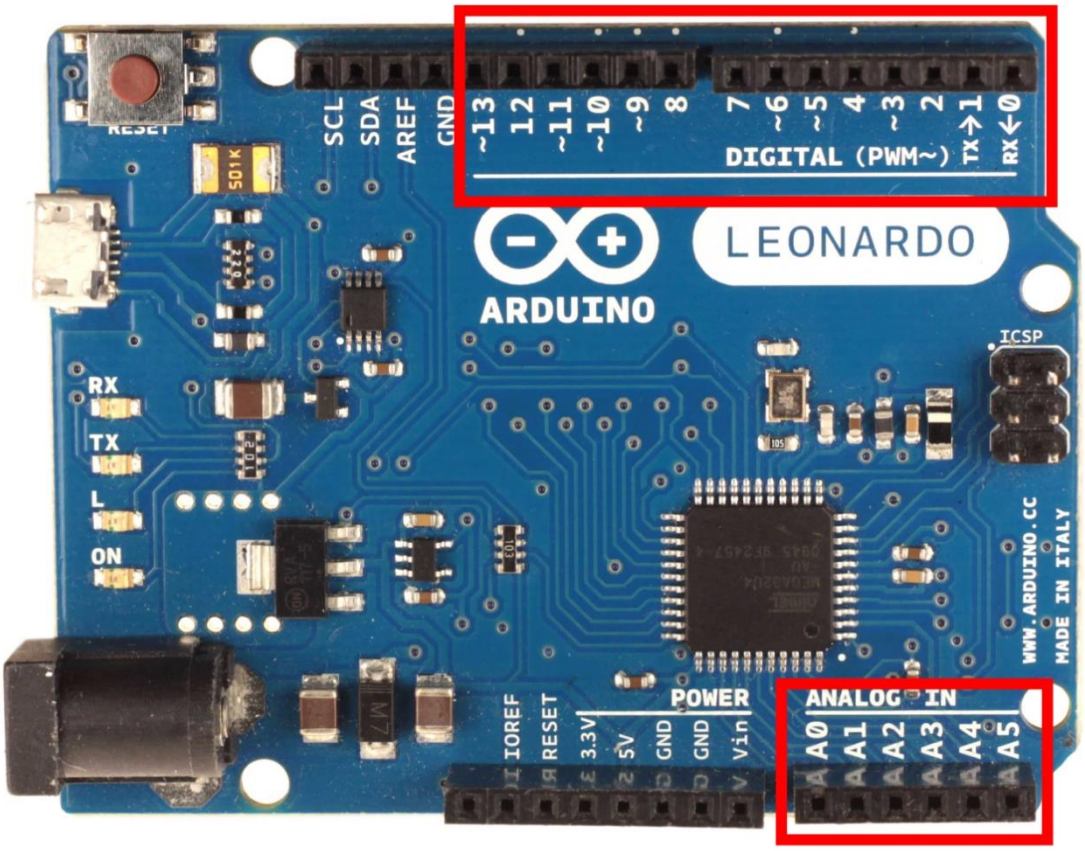
```
// while the serial stream is not open, do nothing:
   while (!Serial) ;
```

Ένα πλεονέκτημα της χρήσης ενός μοναδικού ολοκληρωμένου για τα σκίτσα και την επικοινωνία USB είναι η αύξηση της ευελιξίας στην επικοινωνία με τον υπολογιστή. Μολονότι η πλακέτα εμφανίζεται σαν εικονική σειριακή θύρα στο λειτουργικό σας σύστημα (CDC) για προγραμματισμό και επικοινωνία, μπορεί επίσης να συμπεριφερθεί σαν πληκτρολόγιο ή ποντίκι (HID – Οδηγός Ανθρώπινης Αλληλεπίδρασης).

Στο Leonardo η βασική κλάση `Serial` αναφέρεται στον εικονικό σειριακό οδηγό στη πλακέτα για την σύνδεση USB με τον υπολογιστή. Δεν συνδέεται στους ακροδέκτες 0 και 1 όπως στο Uno. Για να γίνει χρήση της σειριακής θύρας του υλικού (ακροδέκτες 0 και 1, RX και TX) επιστρατεύεται η ρουτίνα `Serial1`.

1.6 Ακροδέκτες εισόδου και εξόδου

Το Arduino Leonardo έχει 20 ψηφιακούς ακροδέκτες εισόδου/εξόδου, όπως φαίνεται στην εικόνα 1.3. Οι ακροδέκτες του Arduino μπορούν να ρυθμιστούν σε δύο καταστάσεις: είσοδος και έξοδος. Οι ακροδέκτες του ATmega λειτουργούν ως είσοδοι από προεπιλογή όποτε δεν χρειάζεται να οριστούν ρητά ως είσοδοι, αλλά συχνά το πράττουμε στον κώδικα για διαύγεια.



Εικόνα 1.3: Οι ψηφιακοί (επάνω) και οι αναλογικοί ακροδέκτες (κάτω) του Arduino.

1.6.1 Ψηφιακοί ακροδέκτες

Το Arduino έχει 14 ψηφιακούς ακροδέκτες, αριθμημένους από το 0 ως το 13. Οι ψηφιακοί ακροδέκτες μπορούν να οριστούν σαν INPUT (είσοδος) ή OUTPUT (έξοδος), χρησιμοποιώντας τη συνάρτηση `pinMode()`. Και στις δύο λειτουργίες, οι ψηφιακοί ακροδέκτες μπορούν μόνο να στείλουν ή να λάβουν ψηφιακά σήματα, που αποτελούνται από δύο καταστάσεις: ON (HIGH ή 5V) και OFF (LOW ή 0V).

Οι ακροδέκτες, που έχουν οριστεί σαν OUTPUT, μπορούν να τροφοδοτήσουν με ρεύμα εξωτερικές συσκευές ή κυκλώματα κατ' απαίτηση. Ακροδέκτες ορισμένοι σαν INPUT είναι έτοιμοι να διαβάσουν ρεύματα από τις συνδεδεμένες σε αυτούς συσκευές.

Επίσης, επτά από αυτούς τους ακροδέκτες (με την ένδειξη ~) μπορούν να χρησιμοποιηθούν σαν ακροδέκτες διαμόρφωσης εύρους παλμού (PWM, Pulse Width Modulation).

1.6.2 Αναλογικοί ακροδέκτες εισόδου

Οι μικροελεγκτές ATmega, που χρησιμοποιούνται στο Arduino, περιέχουν έξι μετατροπείς σήματος από αναλογικό σε ψηφιακό (ADC, Analog-to-Digital Converter). Ο ρόλος τους είναι να μετατρέψουν μια αναλογική τάση εισόδου σε ένα ψηφιακό αριθμό ανάλογο του μεγέθους της τάσης αυτής σε σχέση με τάση αναφοράς (5V).

Ο μετατροπέας στη πλακέτα Arduino έχει ανάλυση 10 bit, που σημαίνει ότι επιστρέφει ακέραιες τιμές από 0 ως 1023 (0 ως 2^{10}-1). Μία τάση εισόδου 0V επιστρέφει μηδενική τιμή, μια τάση εισόδου 5V επιστρέφει τιμή 1023 και μια τάση εισόδου 2,5V επιστρέφει τιμή 512.

Αυτοί οι ακροδέκτες μπορούν να οριστούν ως INPUT ή OUTPUT ακριβώς όπως οι ψηφιακοί, καλώντας τους A0, A1 κλπ.

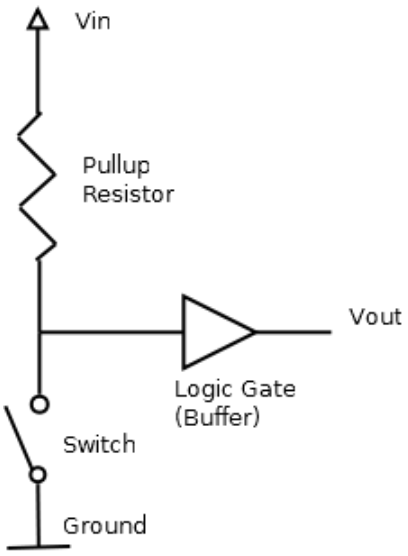
1.6.3 Αντιστάσεις έλξης (pull-up resistors)

Όποτε γίνεται ανάγνωση μιας ψηφιακής εισόδου από έναν ακροδέκτη του Arduino, στην πραγματικότητα γίνεται έλεγχος της εισερχόμενης τάσης σε αυτόν τον ακροδέκτη. Το Arduino επιστρέφει 0 αν δεν ανιχνευτεί τάση και 1 αν ανιχνευτεί τάση γύρω στα 5V. Για να έχουμε συνεπέστερες μετρήσεις πρέπει να εξασφαλίσουμε ότι τα εισερχόμενα σήματα είναι όσο πιο κοντά γίνεται στα 0V και 5V.

Ωστόσο, όταν οι ακροδέκτες είναι ορισμένοι σαν INPUT, λέγεται ότι είναι σε κατάσταση υψηλής εμπέδησης (high-impedance), που σημαίνει ότι χρειάζεται ελάχιστο ρεύμα για να αλλάξει κατάσταση ο ακροδέκτης. Αυτό μπορεί να χρησιμοποιηθεί σε εφαρμογές που εκμεταλλεύονται αυτό το φαινόμενο, όπως χωρητικοί αισθητήρες αφής (capacitive touch sensors). Γενικά, όμως, σημαίνει ότι η κατάσταση μιας αποσυνδεδεμένης εισόδου θα ταλαντεύεται τυχαία ανάμεσα στο 0 και το 1, λόγω του ηλεκτρικού θορύβου.

Για να αποφευχθεί αυτό πρέπει κάπως να προκαταλάβουμε την είσοδο σε μία γνωστή κατάσταση αν δεν υπάρχει σήμα εισόδου. Αυτός είναι ο ρόλος των ενσωματωμένων αντιστατών έλξης.

Παρατηρώντας την εικόνα 1.4 βλέπουμε ότι όταν ο διακόπτης είναι ανοιχτός, η τάση εξόδου V_{out}, η οποία είναι η τάση που θα διαβάσουμε στον ακροδέκτη εισόδου, είναι πολύ κοντά στην τάση V_{in} ή 5V. Όταν ο διακόπτης κλείσει, όλο το ρεύμα περνάει από τον αντιστάτη στην γείωση, διαμορφώνοντας την V_{out} σε 0V. Με αυτόν τον τρόπο, ο ακροδέκτης εισόδου οδηγείται πάντα σε μια συνεπής κατάσταση πολύ κοντά στα 0V ή 5V.



Εικόνα 1.4: Αντίσταση έλξης.

Το ολοκληρωμένο ATmega έχει 20KΩ αντιστάσεις έλξης ενσωματωμένους σε αυτό. Για να ενεργοποιηθούν χρησιμοποιούμε τον παρακάτω κώδικα:

```
pinMode(pin, INPUT);          //ορίζουμε το pin σαν είσοδο
digitalWrite(pin, HIGH);      //ενεργοποιούμε τις αντιστάσεις έλξης
```

1.7 Οι «ασπίδες» του Arduino

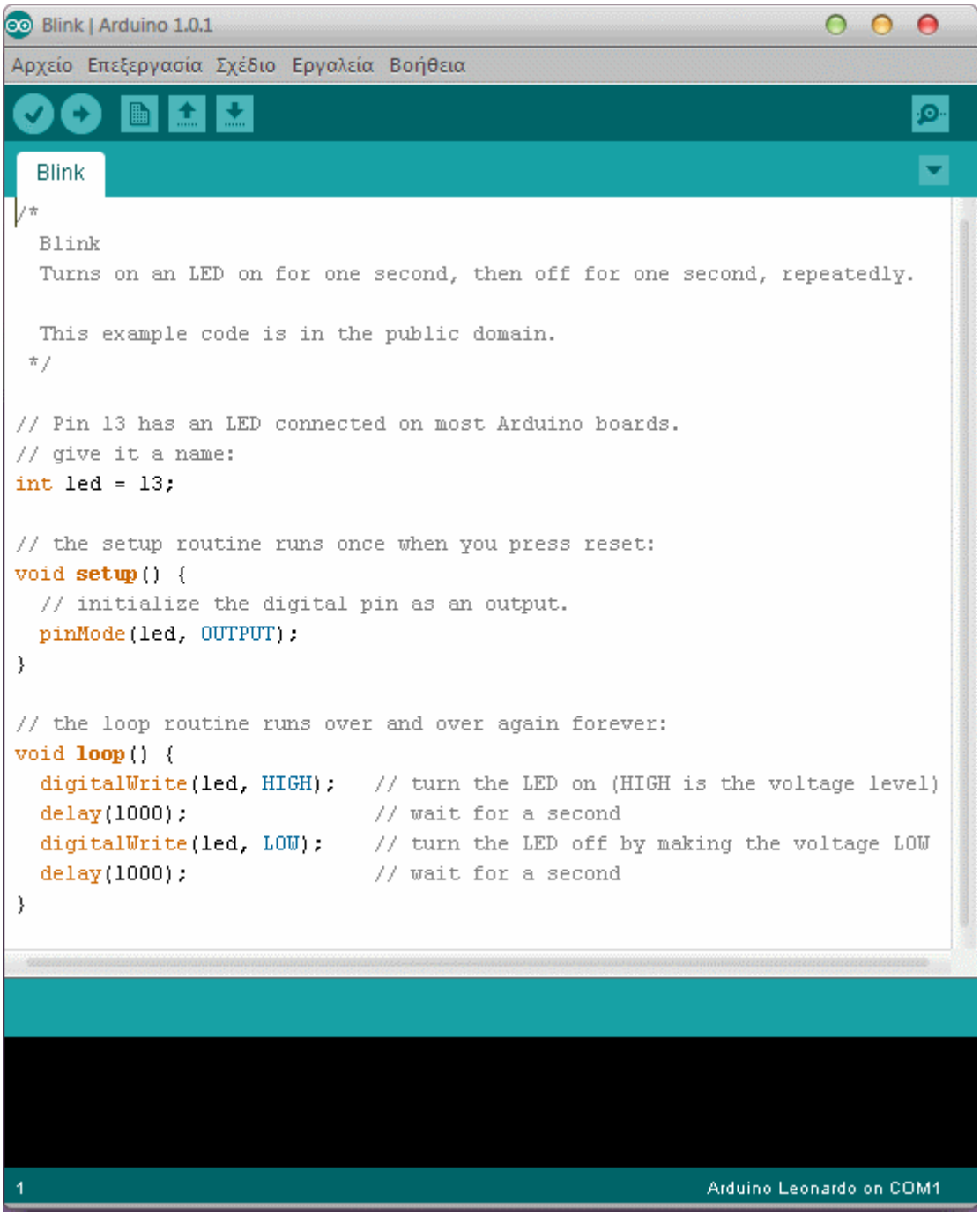
Οι «ασπίδες» του Arduino είναι πλακέτες που έχουν σχεδιαστεί ώστε να εκτελούν κάποια συγκεκριμένη εργασία και συνδέονται κατευθείαν πάνω από την υποστηριζόμενη πλακέτα Arduino, επεκτείνοντας τους ακροδέκτες του Arduino, ώστε να παραμένουν προσβάσιμοι. Υπάρχουν λίγες επίσημες ασπίδες (βλέπε εικόνα 1.5) και τεράστιος αριθμός από ανεπίσημες. Επίσης, ιδιαιτέρως θετικό είναι το γεγονός ότι μπορούμε να δημιουργήσουμε τις δικές μας ασπίδες σύμφωνα με τις ανάγκες μας.

Εικόνα 1.4: Οι επίσημες ασπίδες του Arduino.

1.8 Ολοκληρωμένο περιβάλλον ανάπτυξης (Arduino IDE)

Το Arduino IDE είναι μια εφαρμογή γραμμένη σε γλώσσα προγραμματισμού Java και προέρχεται από το IDE του Processing και του Wiring. Υποστηρίζει πολλαπλά λειτουργικά συστήματα και αποτελεί ένα περιβάλλον ανάπτυξης εφαρμογών. Περιλαμβάνει έναν συντάκτη πηγαίου κώδικα και έναν compiler που μεταφράζει τον κώδικα που είναι γραμμένος σε εντολές μηχανής.

Το Arduino IDE μας επιτρέπει να δημιουργήσουμε με ευκολία τα δικά μας σκίτσα και να φορτώσουμε στην πλακέτα με ένα κλικ. Η εικόνα 1.6 δείχνει τη μορφή του Arduino IDE.



Εικόνα 1.5: Το Arduino IDE.

1.9 Η γλώσσα του Arduino

Η γλώσσα του Arduino υλοποιείται σε C/C++ και βασίζεται στη γλώσσα Wiring. Όταν γράφουμε ένα σκίτσο Arduino, κάνουμε ταυτόχρονα χρήση της βιβλιοθήκης Wiring, η οποία περιλαμβάνεται στο Arduino IDE. Αυτό μας επιτρέπει να κάνουμε εκτελέσιμα προγράμματα χρησιμοποιώντας μόνο δύο

συναρτήσεις: την `setup()` και την `loop()`. Όπως αναφέραμε, η γλώσσα Wiring βασίζεται στο Processing και η δομή της γλώσσας του Arduino κληρονομείται από το Processing, όπου οι αντίστοιχες συναρτήσεις ονομάζονται `setup()` και `draw()`.

Πρέπει να συμπεριλάβουμε και τις δύο συναρτήσεις σε κάθε πρόγραμμα Arduino, ακόμα κι αν δεν χρειαζόμαστε μια από αυτές.

1.9.1 Η συνάρτηση `setup()`

Η συνάρτηση `setup()` εκτελείται μόνο μια φορά, αμέσως μόλις ολοκληρωθεί η φόρτωση του προγράμματος στη πλακέτα, και μετά κάθε φορά που ανάβουμε την πλακέτα. Χρησιμοποιείται για την αρχικοποίηση μεταβλητών, τον ορισμό των ακροδεκτών, την χρήση βιβλιοθηκών κλπ.

1.9.2 Η συνάρτηση `loop()`

Η συνάρτηση `loop()` εκτελείται συνεχώς σε έναν ατέρμονο βρόχο για όσο χρόνο η πλακέτα τροφοδοτείται με ρεύμα. Η συνάρτηση αυτή αποτελεί τον πυρήνα των περισσότερων προγραμμάτων.

1.9.3 Μεταβλητές

Οι μεταβλητές είναι συμβολικά ονόματα που δίνονται σε κάποια συγκεκριμένη ποσότητα πληροφορίας. Αυτό σημαίνει, ότι συνδέουμε μια πληροφορία με ένα συμβολικό όνομα ή αναγνωριστικό, με το οποίο θα αναφερόμαστε σε αυτή.
Προκείμενου να χρησιμοποιήσουμε μια μεταβλητή πρέπει πρώτα να την δηλώσουμε, καθορίζοντας τον τύπο δεδομένων της. Ο τύπος δεδομένων της, προδιαγράφει τον τύπο τον δεδομένων που θα αποθηκευτούν στη μεταβλητή και επηρεάζει το μέγεθος της μνήμης που θα δεσμευτεί για αυτήν. Η γλώσσα του Arduino βασίζεται στη C και στη C++ συνεπώς οι τύποι δεδομένων που μπορούμε να χρησιμοποιήσουμε είναι οι ίδιοι με αυτές.

Το μέρος μέσα στο κώδικα στο οποίο δηλώνουμε μια μεταβλητή έχει αντίκτυπο στην εμβέλεια της. Αν την δηλώσουμε έξω από οποιαδήποτε συνάρτηση, τότε θεωρείται παγκόσμια μεταβλητή και είναι προσβάσιμη από οπουδήποτε στον κώδικα. Αν την δηλώσουμε μέσα σε κάποια συνάρτηση τότε θεωρείται τοπική μεταβλητή και είναι προσβάσιμη μόνο μέσα από την ίδια συνάρτηση.

Κεφάλαιο 2ο

Kinect

2.1 Σύντομη ιστορία

Η τεχνολογία αντίληψης του βάθους πίσω από το Kinect εφευρέθηκε το 2005 από τους: Zeev Zalevsky, Alexander Shpunt, Aviad Maize και Javier Garcia. Το Kinect project ανακοινώθηκε την 1η Ιουνίου 2009 με το κωδικό όνομα Project Natal. Το κωδικό όνομα προέρχεται από την πόλη Νατάλ της Βραζιλίας, γενέτειρα του Alex Kipman, διευθυντή της Microsoft που επώασε το project αυτό.

Αρχικά ο αισθητήρας ήταν σχεδιασμένος να φέρει μικροεπεξεργαστή, ο οποίος θα επιφορτιζόταν με διαδικασίες όπως η ιχνηλασία σκελετού. Όμως τον Ιανουάριο του 2010 αποκαλύφθηκε ότι δεν θα φέρει τελικά αφιερωμένο επεξεργαστή, αλλά, αντιθέτως την επεξεργασία θα αναλάβει ένας από τους υπολογιστικούς πυρήνες του επεξεργαστή του Xbox 360. Σύμφωνα με δηλώσεις του Alex Kipman θα κατανάλωνε 10 – 15 τοις εκατό των υπολογιστικών πόρων του Xbox 360. Αργότερα, ο ίδιος διέψευσε την αρχική του δήλωση, λέγοντας ότι τελικά καταναλώνει μονοψήφιο ποσοστό επί τοις εκατό της επεξεργαστικής ισχύς.

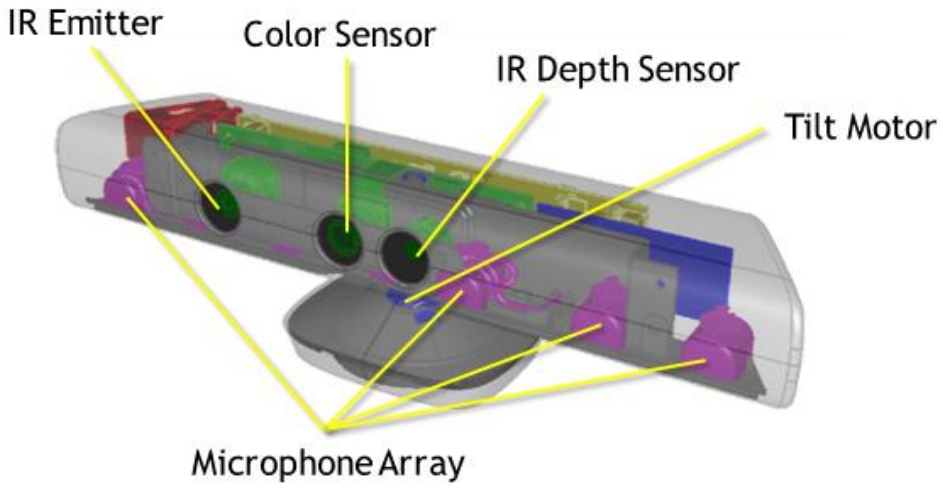
Σε εκδήλωση της Microsoft, στις 13 Ιουνίου 2010, στο πλαίσιο της έκθεσης E3, το όνομα άλλαξε σε Kinect και προέρχεται από τις λέξεις «κινητικός» (kinetic) και «συνδέω» (connect), που εκφράζουν τις ιδέες πίσω από την συσκευή. Το motto της διαφημιστικής εκστρατείας ήταν το ταιριαστό «Είσαι το χειριστήριο» (You are the controller).

Το Kinect (βλέπε εικόνα 2.1) λανσαρίστηκε στις 4 Νοεμβρίου του 2010 και πούλησε οκτώ εκατομμύρια μονάδες τις πρώτες εξήντα μέρες, μπαίνοντας στα Παγκόσμια Ρεκόρ Γκίνες. Το Kinect ήταν ο πρώτος καταναλωτικός αισθητήρας που επέτρεπε στον χρήστη να αλληλεπιδρά με την κονσόλα παιχνιδιών του, με φυσικό τρόπο (χρησιμοποιώντας χειρονομίες και λεκτικές εντολές αντί του κλασικού χειριστηρίου). Από το λανσάρισμα του μέχρι σήμερα έχουν πωληθεί 24 εκατομμύρια μονάδες.

Εικόνα 2.1: Το Kinect για Xbox 360.

2.2 Ο αισθητήρας

Τα χαρακτηριστικά του Kinect (βλέπε εικόνα 2.2) περιλαμβάνουν μια κάμερα RGB, έναν αισθητήρα βάθους (που αποτελείται από έναν υπέρυθρο λέιζερ προβολέα και έναν υπέρυθρο CMOS αισθητήρα) και μία διάταξη μικροφώνων (που κάνει δυνατό τον εντοπισμό ακουστικής πηγής και την καταστολή του περιβάλλοντος θορύβου). Επίσης περιέχει ένα LED, ένα επιταχυνσιομετρο τριών αξόνων και έναν μικρό σερβοκινητήρα που ελέγχει την κλίση της συσκευής.



Εικόνα 2.2: Τα επιμέρους χαρακτηριστικά του Kinect.

2.2.1 Κάμερα RGB

Η κάμερα RGB είναι μια 8-bit κάμερα, ανάλυσης SVGA μαζί με ένα φίλτρο χρώματος Bayer. Το οπτικό πεδίο της είναι 63×50 μοίρες, με 2,9 χιλιοστά εστιακή απόσταση και 2,8μm μέγεθος εικονοστοιχείου (pixel). Υποστηρίζει μέγιστη ανάλυση 1280×1024 με χαμηλό ρυθμό καρέ ανά δευτερόλεπτο (15fps) όπως επίσης και διαφορετικά φορμά χρώματος (YCbCr). Σε ανάλυση VGA (640×480) ο ρυθμός καρέ είναι υψηλότερος στα 30fps. Μπορεί να μην ακούγεται εντυπωσιακή, όμως αποδίδει πολύ καλά σε συνθήκες χαμηλού φωτισμού. Η πραγματική «μαγεία», όμως, λαμβάνει μέρος στον αισθητήρα βάθους, ο οποίος είναι εντελώς ανεξάρτητος από την RGB κάμερα.

2.2.2 Κάμερα IR

Η υπέρυθρη κάμερα είναι μονόχρωμη με μέγιστη ανάλυση 1280×1024 στα 30 καρέ το δευτερόλεπτο. Το οπτικό πεδίο της είναι 58×45 μοίρες, με 6,1 χιλιοστά εστιακή απόσταση και 5,2μm μέγεθος εικονοστοιχείου (pixel). Χρησιμοποιείται για να παρακολουθήσει και να αποκωδικοποιήσει το υπέρυθρο πρότυπο προβολής με σκοπό τον τριγωνισμό μιας τρισδιάστατης σκηνής.

Εικόνα 2.3: Από δεξιά στα αριστερά: Υπέρυθρη κάμερα, RGB κάμερα, LED και υπέρυθρος προβολέας.

2.2.3 Υπέρυθρος προβολέας λέιζερ

Ο προβολέας περιέχει μία δίοδο λέιζερ με μήκος κύματος 830nm. Η έξοδος είναι σταθερή και δεν υφίσταται κανενός είδους διαμόρφωση. Η ισχύς στην έξοδο του προβολέα είναι 60mW. Η θερμοκρασία του λέιζερ σταθεροποιείται από ένα θερμοστοιχείο τύπου Peltier, που είναι τοποθετημένο ανάμεσα από τον προβολέα και την αλουμινένια βάση στήριξης. Το θερμοστοιχείο αυτό μπορεί να θερμάνει και να ψύξει το λέιζερ, διατηρώντας μία σταθερή θερμοκρασία, προκειμένου να σταθεροποιηθεί το μήκος κύματος του.

Ο υπέρυθρος εκπομπός, προβάλει ένα ακανόνιστο μοτίβο υπέρυθρων κηλίδων σε διάφορες εντάσεις. Η υπέρυθρη κάμερα κατασκευάζει το χάρτη βάθους, αναγνωρίζοντας την παραμόρφωση στο μοτίβο αυτό. Η αντίληψη του βάθους επηρεάζεται έντονα από σχετική θέση του προβολέα και του αισθητήρα. Η παραμικρή κύρτωση στην αλουμινένια βάση στήριξης προκαλεί σημαντική διαταραχή στον χάρτη βάθους.

Η ύπαρξη μίας θερμικής ασφάλειας 102°C, συνδεδεμένη στο εξωτερικό μέρος του προβολέα με την κεντρική παροχή 12V του Kinect, είναι ένα αναγκαίο χαρακτηριστικό ασφαλείας. Η τοποθέτηση του σε εκείνο το σημείο υποδηλώνει

ότι εντοπίζει τις συνθήκες που μπορεί να προκαλέσουν ελαττωματική λειτουργία, μη επιτρέποντας στο λέιζερ να θερμάνει το εξωτερικό του περίβλημα, π.χ. αν το μπροστά πλαστικό παράθυρο έχει ζημιά ή αν καταρρεύσει το θερμοστοιχείο αφήνοντας το λέιζερ να αποδράσει σαν ακτίνα. Η ακατέργαστη ισχύς του λέιζερ είναι άκρως επικίνδυνη για τα μάτια αν τυχόν δεν απλωθεί σαν μοτίβο κηλίδων από τα οπτικά μέρη.

2.2.4 Καθορισμός θέσης

Το πρακτικό εύρος του Kinect είναι από 1,2 ως 3,5 μέτρα. Αν τα αντικείμενα είναι πολύ κοντά στον αισθητήρα, θα εμφανίζονται ως μαύρες βούλες, αν είναι πολύ μακριά, η ακρίβεια της σάρωσης θα είναι πολύ χαμηλή, κάνοντας τα να εμφανίζονται ως επίπεδα αντικείμενα. Στο Kinect για Windows, το εύρος είναι μικρότερο, από 0,4 ως 3 μέτρα.
Ο αισθητήρας έχει γωνιακό οπτικό πεδίο, 57° οριζόντια και 43° κάθετα, ενώ ο μηχανοκίνητος άξονας περιστροφής έχει δυνατότητα κλίσης μέχρι 27° πάνω ή κάτω.

2.2.5 Χαρακτηριστικά ήχου

Η διάταξη μικροφώνων περιλαμβάνει τέσσερα μικρόφωνα που καταγράφουν ήχο 24-bit με ρυθμό δειγματοληψίας 16kHz. Τα μικρόφωνα αυτά, είναι ικανά να διαχωρίσουν τις φωνές μπροστά από τον αισθητήρα από άλλους ήχους του περιβάλλοντος, δίνοντας την ικανότητα στον χρήστη να συνομιλήσει και να χρησιμοποιήσει φωνητικές εντολές.

Αν τοποθετήσουμε μικρόφωνα σε διαφορετικά μέρη, ο ήχος θα φτάσει σε αυτά, σε διαφορετικές στιγμές. Με αυτή τη λογική μπορούμε να υπολογίσουμε την πηγή του ήχου αν λάβουμε υπ’ όψιν την διαφορά ανάμεσα στα σήματα που λαμβάνουν τα μικρόφωνα και την ταχύτητα του ήχου στον αέρα. Περαιτέρω μπορούμε να προσδιορίσουμε την θέση από την οποία έρχεται ο ήχος.

Η διάταξη μικροφώνων προσομοιώνει την συμπεριφορά των αυτιών μας. Όταν ακούμε κάτι, ο εγκέφαλος μας υπολογίζει στο περίπου από πού προέρχεται, από τις διαφορές στη φάση του κύματος που φτάνει στο κάθε αυτί.

Αφού υπολογιστεί η θέση από την οποία έρχεται ο ήχος, ένας περίπλοκος αλγόριθμος συγχωνεύει τα σήματα από όλα τα μικρόφωνα, σε ένα ενιαίο σήμα που περιέχει τον ήχο από έναν φανταστικό κώνο μπροστά από το Kinect.

Επίσης, περιλαμβάνεται ένα φίλτρο αποκοπής συχνοτήτων πέρα από την ανθρώπινη ομιλία και ενίσχυσης της έντασης, με τέτοιο τρόπο ώστε οι θόρυβοι του περιβάλλοντος να καταστέλλονται και η ομιλία να ενισχύεται.

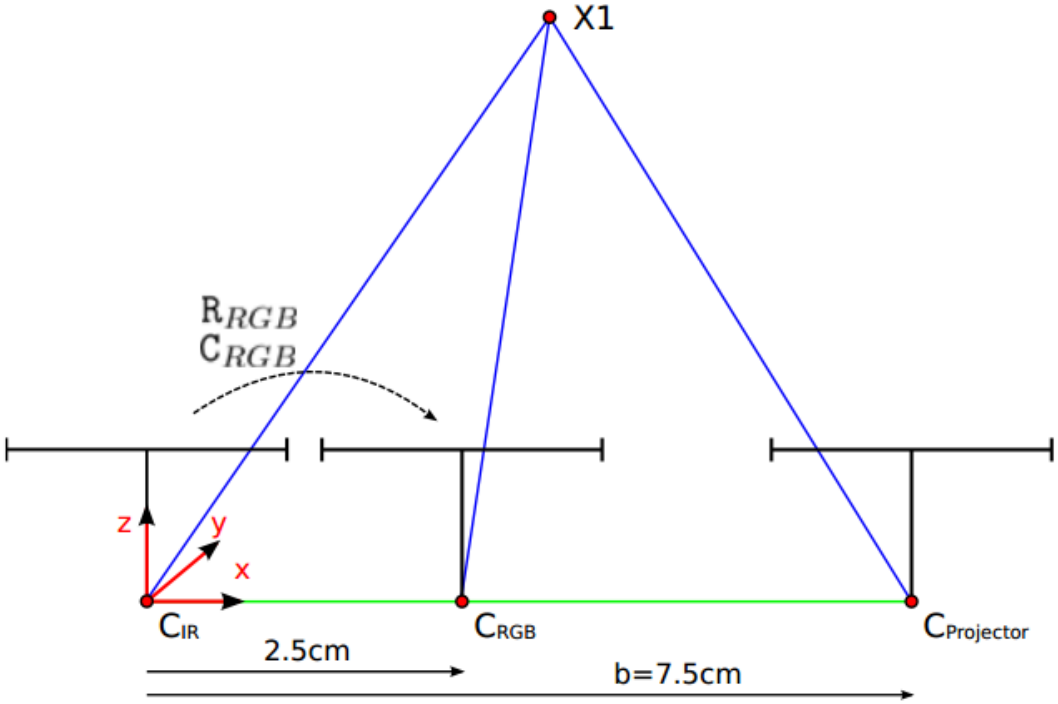
2.3 Γεωμετρικό μοντέλο

Το Kinect μοντελοποιείται σαν ένα σύστημα πολλαπλών θεάσεων, αποτελούμενο από τις κάμερες RGB, IR και βάθους. Το γεωμετρικό μοντέλο των RGB και IR καμερών, που προβάλλουν ένα τρισδιάστατο σημείο X στο σημείο εικόνας $[u, v]^T$, δίνεται από την σχέση:

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = K \begin{bmatrix} s \\ t \\ 1 \end{bmatrix} \quad (1)$$

$$\begin{bmatrix} s \\ t \\ 1 \end{bmatrix} = (1 + k_1 r^2 + k_2 r^4 + k_5 r^6) \begin{bmatrix} p \\ q \\ 0 \end{bmatrix} + \begin{bmatrix} 2k_3 pq + k_4(r^2 + 2p^2) \\ 2k_4 pq + k_3(r^2 + 2q^2) \\ 1 \end{bmatrix} \quad (2)$$

$$r^2 = p^2 + q^2, \quad \begin{bmatrix} pz \\ qz \\ z \end{bmatrix} = R(X - C) \quad (3)$$



Εικόνα 2.4: Γεωμετρικό μοντέλο του Kinect.

με παραμέτρους παραμόρφωσης $k=[k_1,k_2,...,k_5]$, πίνακα βαθμονόμησης κάμερας K, περιστροφή R και κέντρο κάμερας C.

Η κάμερα βάθους του Kinect συνδέεται με την γεωμετρία της υπέρυθρης (IR) κάμερας. Επιστρέφει το αντίστροφο βάθος d κατά μήκος του άξονα z (βλέπε εικόνα 2.4), για κάθε εικονοστοιχείο (pixel) $[u, v]^T$ της IR κάμερας όπως φαίνεται στη σχέση:

$$\begin{bmatrix} x \\ y \\ d \end{bmatrix} = \begin{bmatrix} u - u_0 \\ v - v_0 \\ \frac{1}{c_1}\frac{1}{z} - \frac{c_0}{c_1} \end{bmatrix} \quad (4)$$

όπου τα u και v δίνονται από την σχέση (2), το πραγματικό βάθος z από την σχέση (3), το $[u, v]^T$ από τον πίνακα 1, το x αντιπροσωπεύει τις τρισδιάστατες συντεταγμένες ενός τρισδιάστατου σημείου και τα c_1 και c_0 είναι οι παράμετροι του μοντέλου.

Εικόνα	1	2	3	4	Μέσος όρος
u_0	2,8	2,9	3,0	3,4	3,0
v_0	3,0	2,7	2,8	3,1	2,9

Πίνακας 2: Μετακίνηση θέσης εικονοστοιχείου από κάμερα IR σε κάμερα βάθους.

Συνδέουμε το σύστημα συντεταγμένων του Kinect με την υπέρυθρη κάμερα (IR) και ως εκ τούτου έχουμε $R_{IR}=I$ και $C_{IR}=0$. Ένα τρισδιάστατο σημείο X_{IR} φτιάχνεται από την μέτρηση $[x,y,d]$ στην εικόνα βάθους όπως δείχνει η σχέση:

$$X_{IR} = \frac{1}{c_1 d + c_0} dis^{-1}\left(K_{IR}^{-1}\begin{bmatrix} x+u_0 \\ y+v_0 \\ 1 \end{bmatrix}, k_{IR}\right) \quad (5)$$

και προβάλλεται στις εικόνες RGB όπως δείχνει η σχέση:

$$u_{RGB} = K_{RGB} dis(R_{RGB}(X_{IR} - C_{RGB}), k_{RGB}) \quad (6)$$

όπου dis είναι η συνάρτηση παραμόρφωσης που δίνεται από την εξίσωση (2), οι K_{IR} και K_{RGB} είναι οι αντίστοιχες παράμετροι παραμόρφωσης της υπέρυθρης και της κάμερας RGB, ο K_{IR} ο πίνακας βαθμονόμησης της υπέρυθρης κάμερας και οι K_{RGB}, R_{RGB}, C_{RGB} είναι ο πίνακας βαθμονόμησης, η περιστροφή και το κέντρο, της κάμερας RGB, αντίστοιχα.

2.4 Drivers και Frameworks

2.4.1 Το «χακάρισμα»

Αμέσως μετά το λανσάρισμα του Kinect τον Νοέμβριο του 2010, εκμεταλλευόμενη την USB σύνδεση του, η εταιρία Adafruit Industries, μία εταιρία παραγωγής ηλεκτρονικών kit ανοιχτού κώδικα με έδρα την Νέα Υόρκη, πρόσφερε αμοιβή χιλίων δολαρίων σε οποιονδήποτε μπορούσε να παρέχει

drivers για αυτήν την εξαιρετική συσκευή. Οι drivers και/ή η εφαρμογή θα πρέπει να λειτουργούν σε οποιοδήποτε λειτουργικό σύστημα και θα πρέπει να είναι πλήρως τεκμηριωμένοι και σύμφωνοι με την άδεια λογισμικού ανοιχτού κώδικα.

Η Microsoft, με ανακοίνωση της δήλωσε ότι δεν συναινεί σε τροποποιήσεις του Kinect και ότι θα συνεργαστεί στενά με τα όργανα επιβολής του νόμου, ώστε το Kinect να παραμείνει απαραβίαστο. Οι δηλώσεις αυτές, άθελα της, αποτέλεσαν κίνητρο για τους hackers και η Adafruit διπλασίασε την αμοιβή. Ξαφνικά έγινε αγώνας δρόμου. Η τεχνική αντίστροφης μηχανικής που χρησιμοποίησαν οι hacker είναι δύσκολη. Αφουγκράζονται την συνομιλία του Kinect με τον Η/Υ και τελικά αρχίζουν και καταλαβαίνουν τη γλώσσα που χρησιμοποιούν για να επικοινωνήσουν μεταξύ τους. Δυστυχώς όμως το Kinect παράγει τεράστια ροή δεδομένων. Για να διαβάσουν και να καταγράψουν τα δεδομένα αυτά χρειάζονται ειδικές συσκευές που ονομάζονται αναλυτές USB. Η Adafruit και άλλοι είχαν παραγγείλει τέτοιες συσκευές, άλλα η παράδοση τους θα αργούσε μερικές ημέρες.

Στις 6 Νοεμβρίου, ένας hacker με το ψευδώνυμο AlexP, απέκτησε τον έλεγχο των σερβοκινητήρων του Kinect. Η Microsoft προσπάθησε να αναιρέσει την είδηση δηλώνοντας ότι το Kinect δεν έχει «χακαριστεί» με κανέναν τρόπο. Η κοινότητα των hacker το πήρε σαν προσβολή και η Adafruit έγραψε: «Αυτό είναι σαχλό, έτσι τώρα η αμοιβή έγινε τρείς χιλιάδες δολάρια». Δύο μέρες αργότερα ο AlexP δημοσίευσε ένα βίντεο που έδειχνε τον έλεγχο στην εικόνα και τον χάρτη βάθους του Kinect. Το βραβείο ήταν δικό του αρκεί να δημοσίευε τον κώδικα, μία απόλυτα καθιερωμένη πρακτική των hacker. Αντιθέτως ανακοίνωσε ότι θα δημοσίευε τον κώδικα μόνο αν η κοινότητα συνέβαλε το ποσό των δέκα χιλιάδων δολαρίων για τη χρηματοδότηση του έργου του. Ήταν μία παραβίαση της εθιμοτυπίας των hacker.

Επιτέλους ο αναλυτής USB κατέφθασε και η Adafruit άρχισε να δημοσιοποιεί δεσμίδες δεδομένων καταγεγραμμένων από το Kinect. Η εναλλαγή της προόδου γινόταν σε ένα chatroom. Αρχικά κάποιος βρήκε την εντολή για να ενεργοποιηθεί το Kinect. Αργότερα κάποιος άλλος hacker «έσπασε» τον κώδικα ελέγχου των κινητήρων. Σταμάτησαν να δουλεύουν τις πρώτες πρωινές ώρες

της Τετάρτης σίγουροι πως θα τελειώσουν την επόμενη ημέρα και θα κατακτήσουν το τρόπαιο. Η 10η Νοεμβρίου ήταν η Ευρωπαϊκή ημερομηνία λανσαρίσματος του Kinect. Στις δέκα το πρωί ώρα Ισπανίας, ενώ οι άλλοι κοιμόντουσαν, ο Hector Martin αγόρασε το Kinect σε τοπικό κατάστημα ηλεκτρονικών ειδών κοντά στο Μπιλμπάο. Ο Martin, hacker και προπτυχιακός φοιτητής της επιστήμης των υπολογιστών είχε μείνει ξύπνιος μελετώντας τα δεδομένα που είχε δημοσιεύσει η Adafruit. Λίγο μετά τις έντεκα το πρωί κατάφερε να κάνει τον υπολογιστή του να επικοινωνεί με το Kinect. Λίγο αργότερα πληροφορίες εικόνας και χάρτη βάθους εμφανίζονταν στην οθόνη του από τον αισθητήρα. Δημοσίευσε το βίντεο και πήγε για ύπνο.

Όταν οι hackers των ΗΠΑ ξύπνησαν το μόνο που έμενε ήταν να επαληθευτεί ο κώδικας του Martin. Μόλις έγινε αυτό, η Adafruit του απένειμε τα τρεις χιλιάδες δολάρια, τα οποία δεσμεύτηκε να μοιράσει σε άλλα hacking projects. Μέχρι και ο AlexP έστειλε στον Martin τα 457 δολάρια που του είχαν δωρίσει μέχρι εκείνη τη στιγμή.

Ο Martin δημιούργησε drivers για λειτουργικά συστήματα Linux που επέτρεπαν την χρήση και της RGB κάμερας και της εικόνας βάθους του Kinect. Η ανακοίνωση των drivers ανοιχτού κώδικα συνεπήρε την κοινότητα ανάπτυξης εφαρμογών του Kinect, κάτι που τράβηξε την προσοχή την ΜΜΕ. Διάφορα site αφιερωμένα στον νέο κόσμο εφαρμογών του Kinect ξεφύτρωσαν σαν μανιτάρια σε όλο το διαδίκτυο. Το έκπληκτο κοινό μπορούσε να δει έναν αυξανόμενο αριθμό νέων εφαρμογών να εμφανίζονται σε κάθε γωνιά του κόσμου.

2.4.2 Επίσημα πλαίσια εργασίας (frameworks)

Οι hackers του Kinect δεν ήταν οι μόνοι που συνειδητοποίησαν τις απίστευτες δυνατότητες που εξαπόλυσε αυτή η νέα τεχνολογία. Οι εταιρίες που ενεπλάκησαν στον σχεδιασμό του Kinect σύντομα κατάλαβαν ότι το Kinect για το Xbox 360 ήταν ένα δειλό πρώτο βήμα προς μία τεχνολογική επανάσταση, και δεν είχαν καμία πρόθεση να μείνουν με τα χέρια σταυρωμένα.

Το 2010, η εταιρία PrimeSense (η εταιρία πίσω από την τρισδιάστατη απεικόνιση του Kinect), εξέδωσε τους δικούς της drivers και framework για το Kinect, το επονομαζόμενο OpenNI. Λίγο μετά από αυτό, ανακοίνωσε την συνεργασία της με την ASUS για την παραγωγή μιας όμοιας με το Kinect συσκευής, του Xtion.

Το 2011, η Microsoft λανσάρισε το μη εμπορικό λογισμικό ανάπτυξης του Kinect (Kinect SDK).

Αυτή τη στιγμή, υπάρχουν τρία πλαίσια εργασίας (frameworks) για το Kinect, το OpenKinect, το OpenNI και το Microsoft SDK.

2.4.3 OpenKinect drivers

Λίγο μετά την δημιουργία των drivers ανοιχτού κώδικα για την Adafruit, ο Hector Martin έγινε μέλος της κοινότητας OpenKinect, που δημιουργήθηκε από τον Josh Blake, με σκοπό να φέρει κοντά προγραμματιστές που ενδιαφέρονται για τις φυσικές διεπαφές χρήστη (Natural User Interface, NUI). Το OpenKinect αναπτύσσει και συντηρεί την βιβλιοθήκη libfreenect που δίνει πρόσβαση στο Kinect. Υποστηρίζει την πρόσβαση στις RGB εικόνες, στις εικόνες βάθους, στον σερβοκινητήρα, στο επιταχυνσιομετρο και το LED. Η πρόσβαση στα μικρόφωνα είναι υπό ανάπτυξη.

2.4.4 PrimeSense: OpenNI και NITE

Η Ισραηλινή εταιρία PrimeSense ανέπτυξε την τεχνολογία πίσω από τρισδιάστατη απεικόνιση του Kinect, και δούλεψε με την Microsoft στην εξέλιξη του αισθητήρα. Τον Δεκέμβριο του 2010, δημιούργησε έναν μη κερδοσκοπικό οργανισμό, τον OpenNI, που σημαίνει Open Natural Interaction (Ανοιχτή Φυσική Αλληλεπίδραση).

Ο οργανισμός αυτός σχηματίστηκε με σκοπό την πιστοποίηση και την βελτίωση της συμβατότητας και της χρησιμότητας των συσκευών και των εφαρμογών με

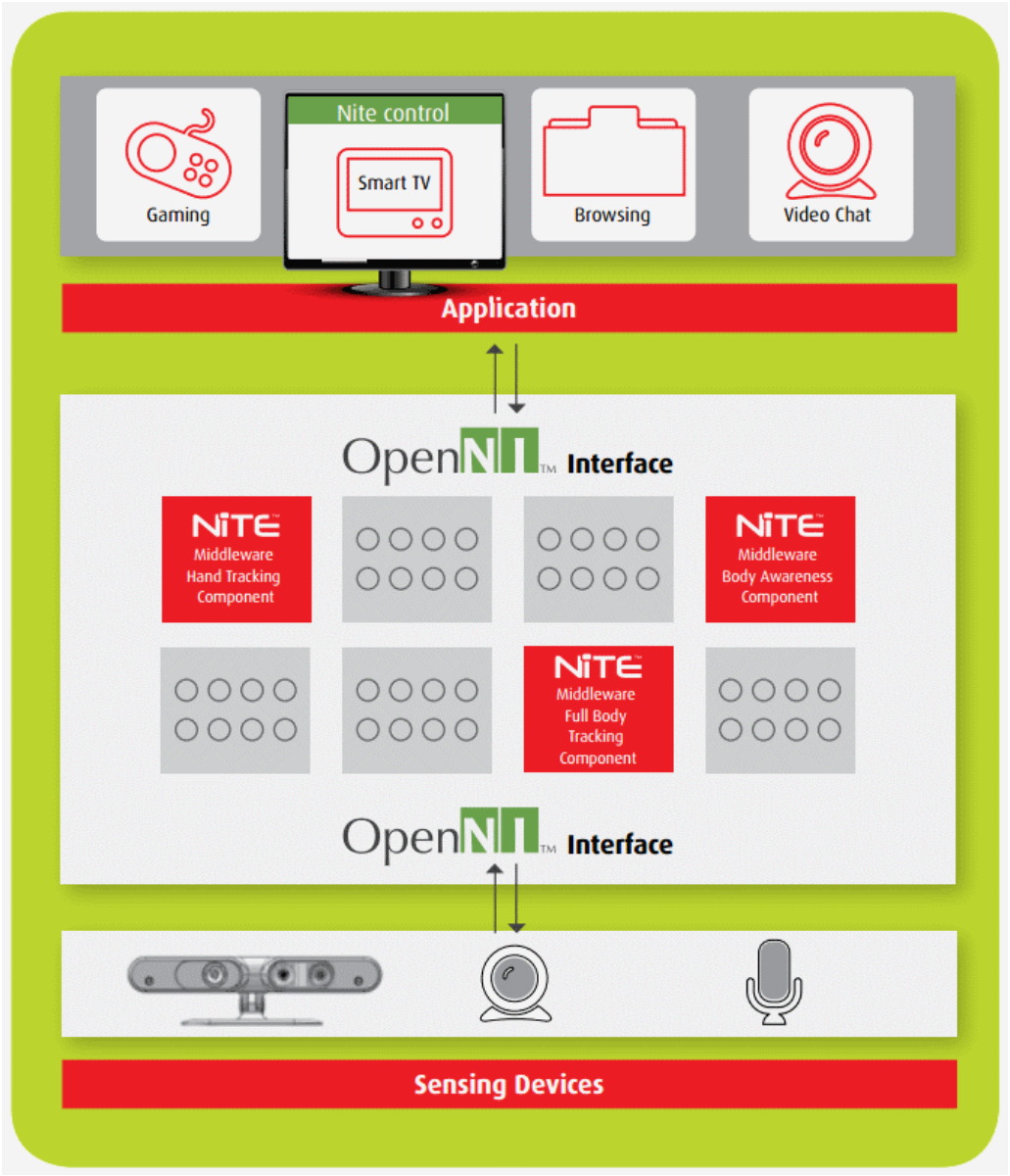
φυσική αλληλεπίδραση. Τα ιδρυτικά μέλη του OpenNI είναι η PrimeSense, η Willow Garage, η Side-Kick, η ASUS και η AppSide.

2.4.4.1 OpenNI

Για να εκπληρώσει τον στόχο του, ο οργανισμός OpenNI κυκλοφόρησε ένα framework ανοιχτού κώδικα που ονομάστηκε OpenNI Framework. Περιλαμβάνει μια διεπαφή προγραμματισμού εφαρμογών (Application Programming Interface, API) και ένα υψηλού επιπέδου ενδιάμεσο λογισμικό (middleware) που ονομάζεται NITE για την υλοποίηση της ιχνηλασίας χεριών/σκελετού και την αναγνώριση χειρονομιών.

Το OpenNI σπάει την εξάρτηση ανάμεσα στον αισθητήρα και το ενδιάμεσο λογισμικό (βλέπε εικόνα 2.5). Η διεπαφή προγραμματισμού εφαρμογών, καθιστά ικανούς τους προγραμματιστές, να αναπτύξουν αλγόριθμους πάνω σε ακατέργαστες δομές δεδομένων, ανεξάρτητα με τον αισθητήρα που παρέχει τα δεδομένα αυτά. Ομοίως, η βιομηχανία αισθητήρων, κατασκευάζει αισθητήρες που συνεργάζονται με οποιαδήποτε συσκευή συμμορφώνεται με τις προδιαγραφές του OpenNI.

Το Kinect ήταν η πρώτη υλοποίηση του σχεδίου αναφοράς της PrimeSense, με την ίδια να αναπτύσσει εξ’ ολοκλήρου τα οπτικά μέρη και τα ολοκληρωμένα κυκλώματα του. Σε δεύτερη φάση η Microsoft πρόσθεσε τον σερβοκινητήρα και το επιταχυνσιόμετρο στο σχέδιο. Ακριβώς γι’ αυτό το λόγο, το OpenNI δεν έχει πρόσβαση στα προαναφερθέντα στοιχεία, διότι είναι συγκεκριμένα της υλοποίησης του Kinect.



Εικόνα 2.5: Αποσπασματική κατά επίπεδα απεικόνιση του OpenNI.

Η PrimeSense είναι μία εταιρία ημιαγωγών χωρίς την δική της παραγωγή. Το εισόδημα της προέρχεται από την πώληση εξαρτημάτων και ολοκληρωμένων κυκλωμάτων με ανάθεση της παραγωγής σε τρίτους (outsourcing). Δραστηριοποιείται στον τομέα B2B (Business to Business) δηλαδή πουλάει λύσεις σε κατασκευαστές, οι οποίοι εισάγουν αυτές τις λύσεις σε καταναλωτικά προϊόντα. Με αυτόν ακριβώς τον τρόπο ενεπλάκη στην ανάπτυξη του Kinect μαζί με τη Microsoft. Αργότερα πουλάει την τεχνολογία αυτή σε κατασκευάστριες εταιρίες υπολογιστών ή τηλεοράσεων όπως η ASUS. Αλλά για να αναπτυχτεί αυτή η αγορά, πρέπει να υπάρχει μία κοινότητα από ανθρώπους, που

δημιουργούν περιεχόμενο και εφαρμογές φυσικής αλληλεπίδρασης. Έτσι δημιουργήθηκε το OpenNI, σαν ένας τρόπος ενδυνάμωσης των προγραμματιστών στην προσθήκη φυσικής αλληλεπίδρασης στις εφαρμογές τους.

2.4.4.2 NITE

Στον προγραμματιστή δεν αρκούν τα τρισδιάστατα σημεία για την υλοποίηση της φυσικής αλληλεπίδρασης. Τα πιο χρήσιμα χαρακτηριστικά προέρχονται από την δυνατότητα ιχνηλασίας του σκελετού και των χεριών. Δεν έχουν όλοι τις γνώσεις, τον χρόνο ή τους πόρους ώστε να αναπτύξουν αυτήν την δυνατότητα από το μηδέν, ιδιαίτερα εφόσον περιλαμβάνουν προχωρημένους αλγόριθμους. Η Primesense αποφάσισε να υλοποιήσει και να διανείμει αυτήν την δυνατότητα για εμπορικούς σκοπούς αλλά κράτησε τον κώδικα κλειστό. Το αποτέλεσμα είναι το ενδιάμεσο λογισμικό NITE.

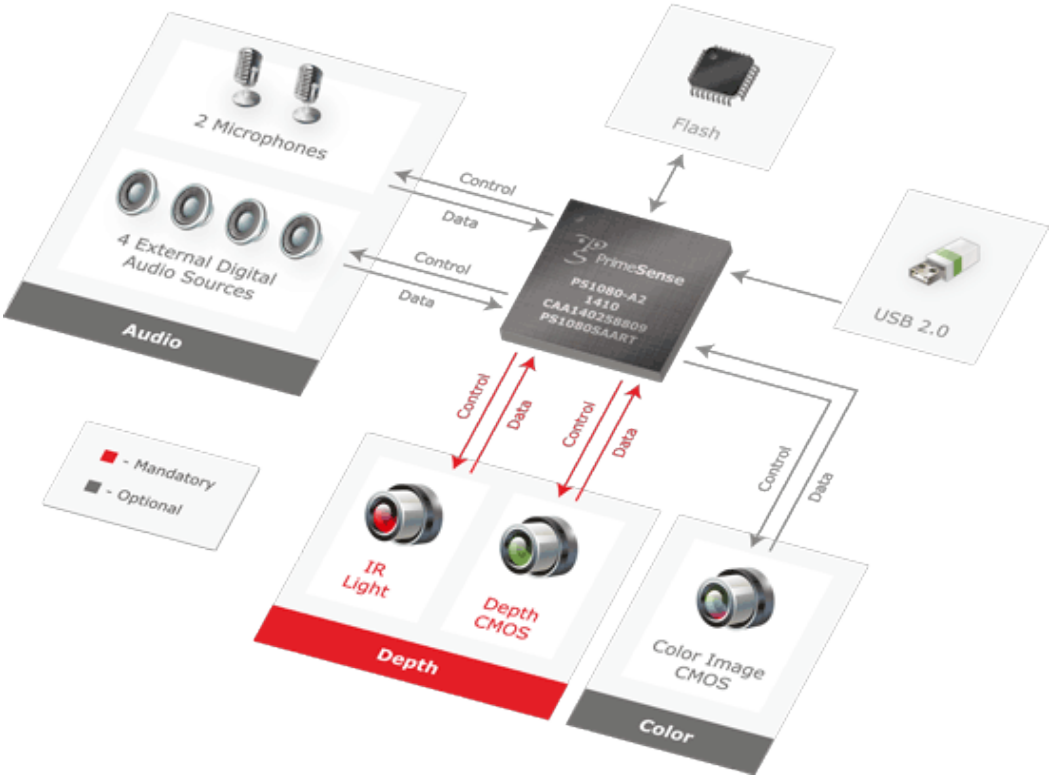
Συνεπώς χωρίς το NITE δεν μπορούμε να χρησιμοποιήσουμε την δυνατότητα ιχνηλασίας του σκελετού και των χεριών ή την αναγνώριση χειρονομιών, εκτός και αν αναπτύξουμε το δικό μας ενδιάμεσο λογισμικό. Σχεδόν σίγουρο είναι ότι στο μέλλον θα αναπτυχθούν και άλλα ενδιάμεσα λογισμικά από τρίτους, τα οποία θα συναγωνίζονται το OpenNI και το NITE, στον τομέα εφαρμογών φυσικής αλληλεπίδρασης.

Όταν το Kinect εγκατασταθεί σωστά και επικοινωνεί με τον Η/Υ, θα είμαστε σε θέση να προσπελάσουμε μια ροή από ακατέργαστα δεδομένα και άλλες δυνατότητες που παρέχονται από το συγκεκριμένο ενδιάμεσο λογισμικό (middleware).

2.4.4.2.1 Δυνατότητες NITE

Μπορούμε να χρησιμοποιήσουμε το Kinect σαν webcam ανάλυσης 640×480.
Επίσης, εφόσον διαθέτει υπέρυθρη κάμερα, μπορούμε να έχουμε πρόσβαση σε υπέρυθρη εικόνα ανάλυσης 1280×1024. Ο χάρτης βάθους είναι αποτέλεσμα

των εργασιών που εκτελούνται, από το ολοκληρωμένο PS1080 της PrimeSense (βλέπε εικόνα 2.6), στην υπέρυθρη εικόνα που τραβήχτηκε από την υπέρυθρη κάμερα. Η εικόνα αυτή έχει ακρίβεια 11 bit, δηλαδή παίρνει 2048 (2^{11}) διαφορετικές τιμές, η οποία αναπαριστάται γραφικά με αποχρώσεις του γκρι, από το άσπρο (2048) ως το μαύρο (0).



Εικόνα 2.6: Το διάγραμμα του ολοκληρωμένου PS1080.

2.4.5 Microsoft Kinect SDK

Τον Φεβρουάριο του 2011, η Microsoft ανακοίνωσε το μη εμπορικό λογισμικό ανάπτυξης (SDK) του Kinect για Windows. Η κυκλοφορία του έγινε τον Ιούνιο του 2011 για Windows 7 σε δώδεκα χώρες.

Το SDK περιλαμβάνει συμβατούς drivers για τον αισθητήρα. Παρέχει στους προγραμματιστές τη ικανότητα να κατασκευάσουν εφαρμογές σε γλώσσες C++, C# και Visual Basic χρησιμοποιώντας το Microsoft Visual Studio 2010. Οι δυνατότητες που παρέχει είναι: πρόσβαση σε ακατέργαστες ροές δεδομένων

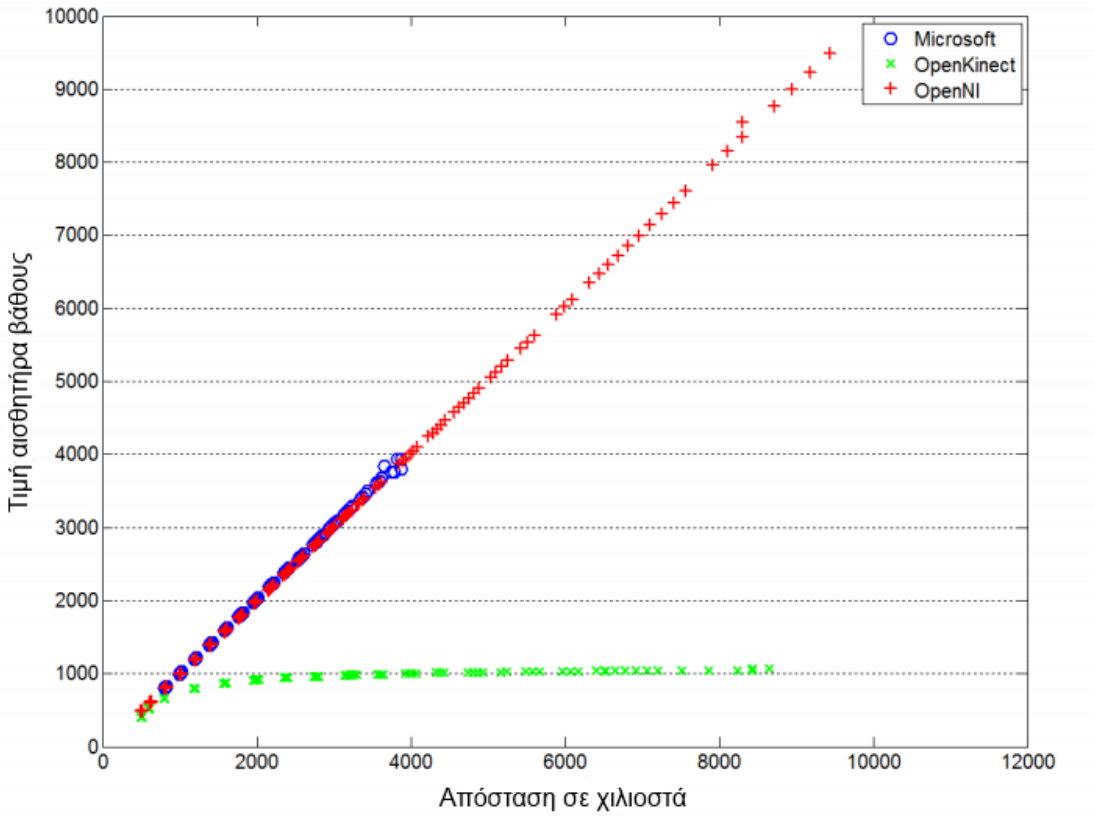
του αισθητήρα, ιχνηλασία σκελετού, προηγμένες δυνατότητες ήχου, δείγματα κώδικα και τεκμηρίωση.

Τον Μάρτιο του 2012 η Microsoft ανακοίνωσε την επόμενη έκδοση του Kinect for Windows. Τον Μάιο 2012, το Kinect for Windows 1.5 κυκλοφόρησε με την προσθήκη νέων δυνατοτήτων, υποστήριξη πολλών γλωσσών και πρωτοεμφανίστηκε σε 19 περισσότερες χώρες. Οι νέες δυνατότητες είναι οι εξής:
Περιλαμβάνει την εφαρμογή Kinect Studio που επιτρέπει τους προγραμματιστές την καταγραφή, αναπαραγωγή και αποσφαλμάτωση (debug) διαφόρων κλιπ από χρήστες που αλληλεπιδρούν με εφαρμογές. Υποστηρίζει νέο σύστημα σκελετού σε καθιστή θέση που επιτρέπει την ιχνηλασία κεφαλιού, λαιμού και χεριών των χρηστών είτε είναι καθιστοί είτε όρθιοι. Υποστηρίζει τέσσερις νέες γλώσσες φωνητικής αναγνώρισης (Γαλλικά, Ισπανικά, Ιταλικά και Γιαπωνέζικα). Επιπρόσθετα υποστηρίζει τις τοπικές διαλέκτους των γλωσσών αυτών μαζί με τα Αγγλικά. Είναι διαθέσιμο σε Χονγκ Κονγκ, Νότια Κορέα, Ταιβάν, Αυστρία, Βέλγιο, Βραζιλία, Δανία, Φινλανδία, Ινδία, Ολλανδία, Νορβηγία, Πορτογαλία, Ρωσία, Σαουδική Αραβία, Σιγκαπούρη, Νότιος Αφρική, Σουηδία, Ελβετία και Ηνωμένα Αραβικά Εμιράτα.

2.4.6 Σύγκριση πλαισίων εργασίας (frameworks)

Το Microsoft SDK είναι μόνο διαθέσιμο για Windows 7 ενώ τα άλλα δύο υποστηρίζουν πολλαπλά λειτουργικά συστήματα και είναι ανοιχτού κώδικα. Μία άλλη διαφορά είναι ότι το Microsoft SDK δίνει τιμές βάθους μέχρι τέσσερα μέτρα ενώ τα άλλα δύο μέχρι εννέα μέτρα. Αυτό είναι μάλλον επιλογή της Microsoft λόγω της χαμηλής ανάλυσης και του θορύβου σε μεγαλύτερες αποστάσεις. Εν τούτοις τα δεδομένα βάθους είναι εύκολα προσβάσιμα και σε αποστάσεις μεγαλύτερες των τεσσάρων μέτρων. Το Microsoft SDK δεν αντιλαμβάνεται βάθος κάτω από ογδόντα εκατοστά ενώ τα άλλα δύο κάτω από πενήντα εκατοστά.

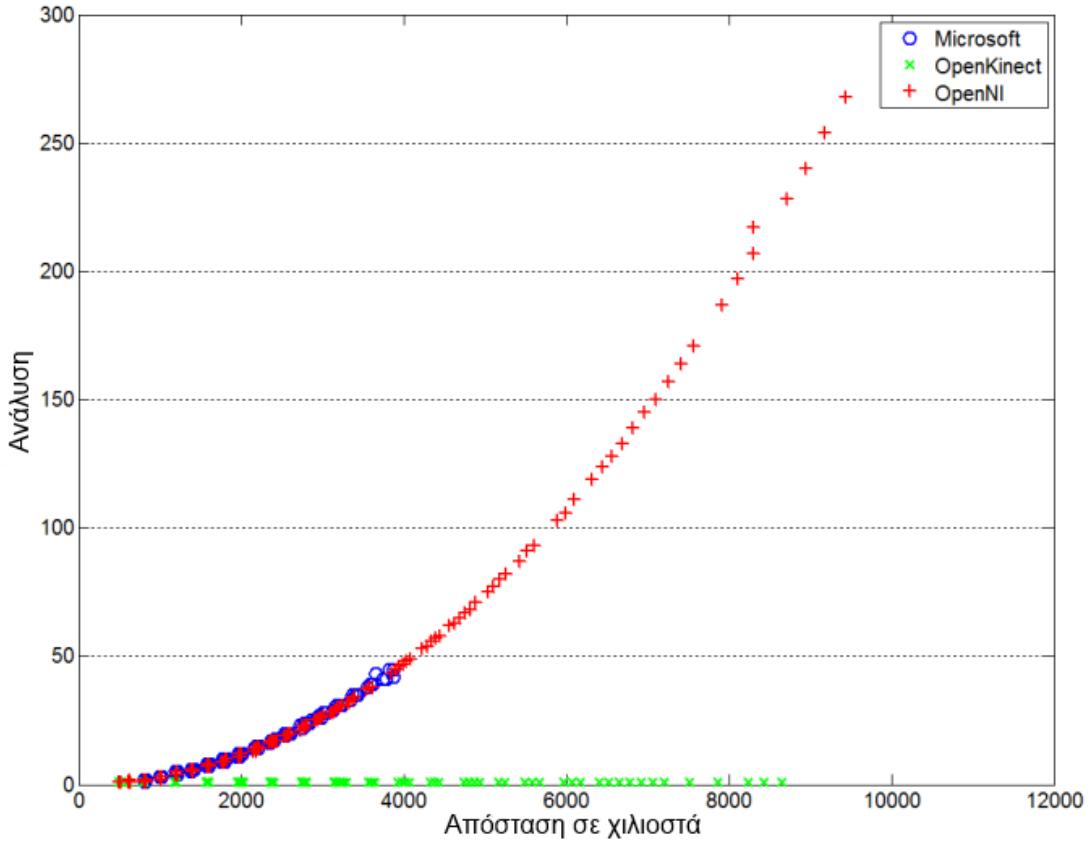
Η εικόνα 2.7 απεικονίζει τις δυαδικές τιμές σε σχέση με τις πραγματικές τιμές βάθους των τριών πλαισίων εργασίας. Οι δυαδικές τιμές υπολογίζονται μετατρέποντας τα δυαδικά μοτίβα σε θετικούς ακέραιους αριθμούς. Φαίνεται ότι το OpenKinect επιστρέφει μη βαθμονομημένα δεδομένα χωρίς γραμμικοποίηση. Αυτό σημαίνει ότι κάθε δυαδική τιμή έχει ξεχωριστή τιμή βάθους. Τα άλλα δύο πλαίσια εργασίας αντί αυτού γραμμικοποιούν τις δυαδικές τιμές έτσι ώστε να αντιπροσωπεύουν χιλιοστά με ακρίβεια. Αυτό προφανώς τα κάνει ευκολότερα να χρησιμοποιηθούν στην πράξη όμως, επίσης, συνεπάγεται ότι δεν είναι όλες οι δυαδικές τιμές δυνατές. Προφανώς, η γραμμικοποίηση αντιστοιχεί σε μετατροπή με πίνακα αναζήτησης.



Εικόνα 2.7: Οι δυαδικές τιμές σε σχέση με τις πραγματικές τιμές βάθους των τριών πλαισίων εργασίας.

Η εικόνα 2.8 δείχνει την ανάλυση βάθους των τριών πλαισίων εργασίας, η οποία αποτελεί την μικρότερη πιθανή απόσταση μεταξύ των δυαδικών τιμών. Το Microsoft SDK και OpenNI δίνουν περίπου όμοια αποτελέσματα χωρίς αυτό να αποτελεί έκπληξη, αφού χρησιμοποιούν το ίδιο υλικό και μέθοδο

γραμμικοποίησης. Η ανάλυση των δύο πλαισίων εργασίας μεταφράζεται στη μικρότερη δυνατή διαφορά βάθους που μπορεί να αντιληφθεί το Kinect σε συγκεκριμένη απόσταση. Παραδείγματος χάριν στα έξι μέτρα η μικρότερη μετρήσιμη διαφορά βάθους είναι περίπου δέκα εκατοστά (0,1 μέτρα). Η πιο σημαντική διαφορά των δύο πλαισίων εργασίας είναι το περιορισμένο εύρος του Microsoft SDK. Το OpenKinect, από την άλλη, δεν κάνει γραμμικοποίηση, επομένως έχει πάντα ανάλυση ενός δυαδικού ψηφίου (1 bit). Η μη γραμμικοποίηση του OpenKinect δεν σημαίνει ότι μπορεί να μετρήσει μικρότερες διαφορές βάθους.



Εικόνα 2.8: Οι αναλύσεις βάθους των τριών πλαισίων εργασίας.

Οι παραπάνω εικόνες προέρχονται με πειραματικό τρόπο. Ένα μεγάλο χαρτοκιβώτιο τοποθετήθηκε σε διάφορες αποστάσεις από τον αισθητήρα και ένα κομμάτι του κουτιού χρησιμοποιήθηκε για να αντιστοιχίσει τις δυαδικές τιμές σε πραγματικές τιμές βάθους και για να μετρήσει τις αναλύσεις βάθους. Οι

αναλύσεις βάθους βρέθηκαν, απλά, σαν η μικρότερη αλλαγή στις τιμές βάθους στο κομμάτι του χαρτόκουτου.

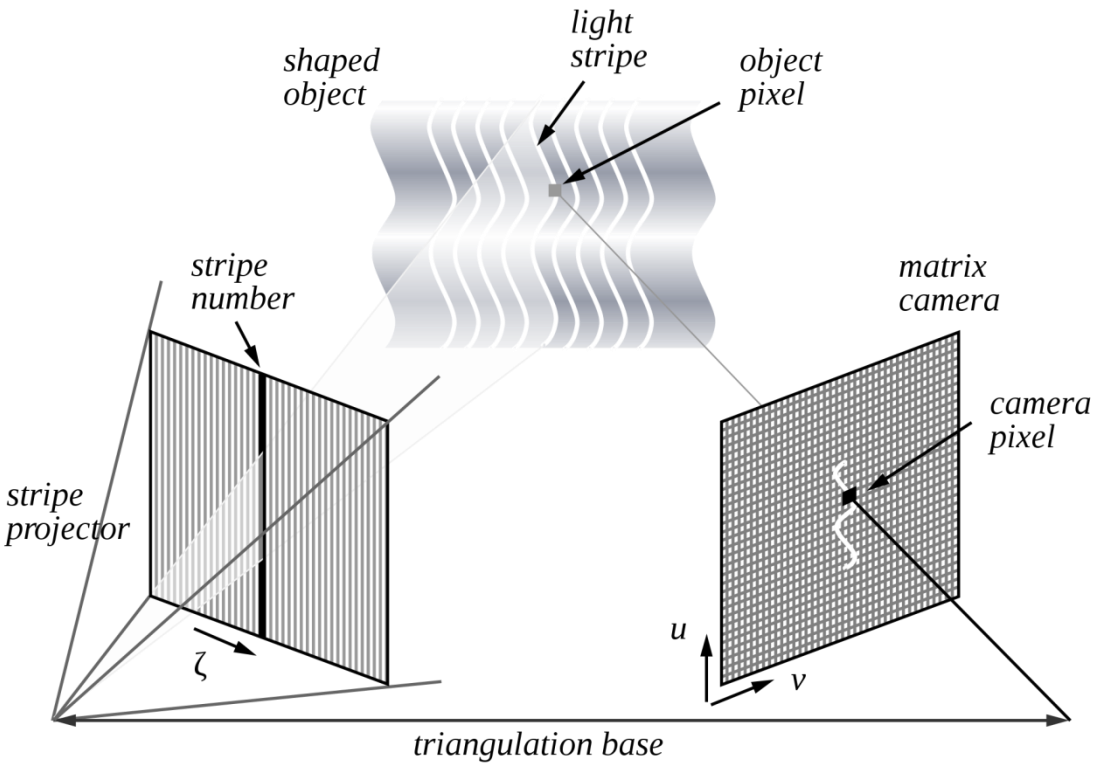
Μετά την αξιολόγηση των τριών πλαισίων εργασίας αποφασίστηκε να χρησιμοποιηθεί το OpenNI για λόγους που αναφέρονται παραπάνω.

2.5 Η θεωρία

Η τεχνική που χρησιμοποιεί η PrimeSense για το σύστημα τρισδιάστατης απεικόνισης λέγεται structured-light 3D scanning (τρισδιάστατη σάρωση αντικειμένου με χρήση του φωτός). Χρησιμοποιείται σε πολλές βιομηχανικές εφαρμογές για έλεγχο παραγωγής και μέτρηση ποσότητας παραγωγής. Συνήθως περιλαμβάνει υψηλού κόστους και υψηλής ακρίβειας σαρωτές. Το Kinect είναι η πρώτη συσκευή που χρησιμοποιεί αυτή την τεχνική σε εμπορικό επίπεδο.

2.5.1 Structured-light 3D scanning

Οι περισσότεροι σαρωτές αντικειμένων με χρήση φωτός βασίζονται στην προβολή μιας στενής λωρίδας φωτός σε ένα τρισδιάστατο αντικείμενο, χρησιμοποιώντας την παραμόρφωση της λωρίδας υπό διαφορετική γωνία παρατήρησης, για την μέτρηση της απόστασης του κάθε σημείου από την κάμερα. Με αυτόν τον τρόπο συνθέτεται ο τρισδιάστατος όγκος του αντικειμένου. Η μέθοδος αυτή μπορεί να επεκταθεί με την προβολή πολλών λωρίδων φωτός ταυτόχρονα, παρέχοντας μεγάλο αριθμό δειγμάτων (βλέπε εικόνα 2.9).

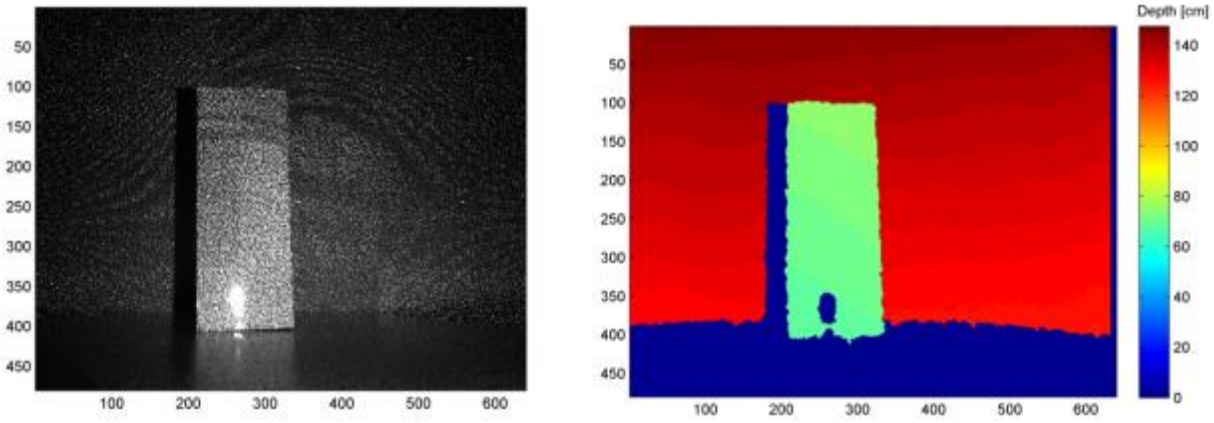


Εικόνα 2.9: Αρχές τριγωνισμού για σαρωτές αντικειμένων με χρήση φωτός.

2.5.2 Μετατροπή του προβαλλόμενου μοτίβου κηλίδων σε χάρτη βάθους

Το σύστημα στο Kinect είναι λίγο διαφορετικό. Αντί να προβάλλει λωρίδες ορατού φωτός, ο προβολέας υπέρυθρων του Kinect, προβάλλει ένα συνεχές μοτίβο κηλίδων. Το μοτίβο συλλαμβάνεται από την υπέρυθρη κάμερα και συσχετίζεται με ένα μοτίβο αναφοράς. Το μοτίβο αναφοράς είναι η λήψη ενός επίπεδου του αντικειμένου, από γνωστή απόσταση, η οποία αποθηκεύεται στη μνήμη του αισθητήρα. Όταν μία κηλίδα προβάλλεται σε ένα αντικείμενο, του οποίου η απόσταση από τον αισθητήρα είναι μεγαλύτερη ή μικρότερη από αυτή του επιπέδου αναφοράς τότε η θέση της κηλίδας στην υπέρυθρη εικόνα θα μετατοπιστεί στη κατεύθυνση της αρχικής, ανάμεσα στον προβολέα λέιζερ και το αντιλαμβανόμενο κέντρο της υπέρυθρης κάμερας. Αυτές οι μετατοπίσεις μετρούνται για όλες τις κηλίδες από μία απλή διαδικασία συσχέτισης εικόνας, η οποία αποδίδει μία εικόνα διαφορών. Για κάθε εικονοστοιχείο (pixel), η

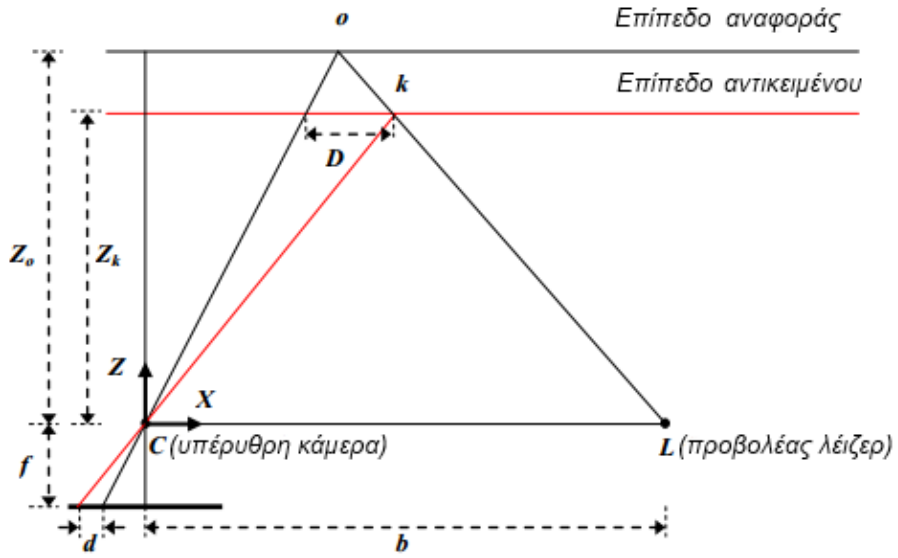
απόσταση από τον αισθητήρα μπορεί να ανακτηθεί από την αντίστοιχη εικόνα διαφορών.



Εικόνα 2.10: Υπέρυθρη εικόνα ενός μοτίβου κηλίδων (αριστερά) και η αντίστοιχη εικόνα βάθους (δεξιά).

2.5.3 Μαθηματικό μοντέλο παραγωγής βάθους

Η εικόνα 2.11 απεικονίζει την σχέση μεταξύ απόστασης ενός αντικειμένου k από τον αισθητήρα, σε σχέση προς το επίπεδο αναφοράς και τη μετρημένη διαφορά d. Για να εκφραστούν οι τρισδιάστατες συντεταγμένες του αντικειμένου, θεωρούμε ένα σύστημα συντεταγμένων βάθους, με την αρχή του στο αντιλαμβανόμενο κέντρο της υπέρυθρης κάμερας. Ο άξονας Z είναι ορθογώνιος ως προς το επίπεδο του αντικειμένου, ο άξονας X είναι κάθετος στον άξονα Z στη κατεύθυνση της γραμμής στοίχισης b, ανάμεσα στην υπέρυθρη κάμερα και τον προβολέα λέιζερ και ο άξονας Y είναι ορθογώνιος στον X και Z.



Εικόνα 2.11: Η σχέση μεταξύ του σχετικού βάθους και της μετρημένης διαφοράς.

Θεωρούμε ότι ένα αντικείμενο είναι στο επίπεδο αναφοράς σε απόσταση Z_o από τον αισθητήρα και μία κηλίδα του αντικειμένου συλλαμβάνεται στο επίπεδο εικόνας από την υπέρυθρη κάμερα. Αν το αντικείμενο μετατοπιστεί πιο κοντά στον αισθητήρα, η θέση της κηλίδας στο επίπεδο εικόνας θα εκτοπιστεί στην κατεύθυνση του X. Αυτό μετριέται στο χώρο της εικόνας σαν την διαφορά d που αντιστοιχεί σε ένα σημείο k στο πεδίο του αντικειμένου. Από την ομοιότητα τριγώνων έχουμε:

$$\frac{D}{b} = \frac{Z_o - Z_k}{Z_o} \tag{7}$$

και:

$$\frac{d}{f} = \frac{D}{Z_k} \tag{8}$$

όπου το Z_k υποδηλώνει την απόσταση (βάθος) του σημείου k στο χώρο του αντικειμένου, το b το βασικό μήκος, το f το εστιακό μήκος της υπέρυθρης κάμερας, το D την μετατόπιση του σημείου k στο χώρο του αντικειμένου και το d παρατηρούμενη διαφορά στο χώρο της εικόνας. Αντικαθιστώντας το D από την εξίσωση (8) στην εξίσωση (7) και εκφράζοντας το Z_k σε σχέση με άλλες μεταβλητές έχουμε:

$$Z_k = \frac{Z_o}{1 + \frac{Z_o}{fb} d} \tag{9}$$

Η εξίσωση (9) είναι το βασικό μαθηματικό μοντέλο για τη παραγωγή του βάθους από την παρατηρούμενη διαφορά, υπό τον όρο ότι οι σταθερές παράμετροι Z_o, f και b μπορούν να καθοριστούν με βαθμονόμηση. Η συντεταγμένη Z ενός σημείου μαζί με το f ορίζουν την κλίμακα απεικόνισης για αυτό το σημείο.

2.6 Διάφορα project βασισμένα στο Kinect

Το Kinect είναι ένας πανίσχυρος αισθητήρας, ο οποίος άφησε πίσω τον αρχικό του σκοπό (φυσική αλληλεπίδραση χρήστη με την κονσόλα παιχνιδιών του) και προχώρησε προς το μέλλον, με το να γίνει η ραχοκοκαλιά σε μια σειρά από ριζοσπαστικές εφαρμογές που δείχνουν τις πραγματικές του ικανότητες.

2.6.1 Fitnect

Το Fitnect (βλέπε εικόνα 2.12) είναι ένα διαδραστικό δοκιμαστήριο ρούχων. Το concept του εικονικού δοκιμαστηρίου έφτασε σε υψηλότατο επίπεδο επιτυχίας με την χρήση του Kinect. Το Fitnect είναι ένα τρισδιάστατο σύστημα αυξημένης πραγματικότητας (augmented reality) που αντικαθιστά το παραδοσιακό δοκιμαστήριο ρούχων. Επιτρέπει στους αγοραστές να δοκιμάσουν εικονικά διάφορα ρούχα, για να επιλέξουν αν τους ταιριάζουν και να καταλήξουν σε αγορά γρηγορότερα και ευκολότερα.

Εικόνα 2.12: Το Fitnect στη πράξη.

2.6.2 Kiwibank Welcome Experience

Το Kiwibank Welcome Experience (βλέπε εικόνα 2.13) είναι ένας τοίχος πληροφοριών.

Εικόνα 2.13: Το Kiwibank Interactive Wall στη διαδικασία του login.

Πιο συγκεκριμένα επιτρέπει στους χρήστες να παίζουν παιχνίδια, να «σερφάρουν» στο διαδίκτυο και να αντλούν τις πληροφορίες που χρειάζονται. Το περιβάλλον υποστηρίζει πολλαπλούς χρήστες και η καθοδήγηση σε αυτό γίνεται με χειρονομίες. Ο τοίχος είναι θεμελιώδης εφαρμογή του “cloud computing” και μπορεί σύντομα να συναντάμε κόσμο να αποκτά πρόσβαση στο παγκόσμιο ιστό από διάφορους τοίχους της πόλης.

2.6.3 JediBot

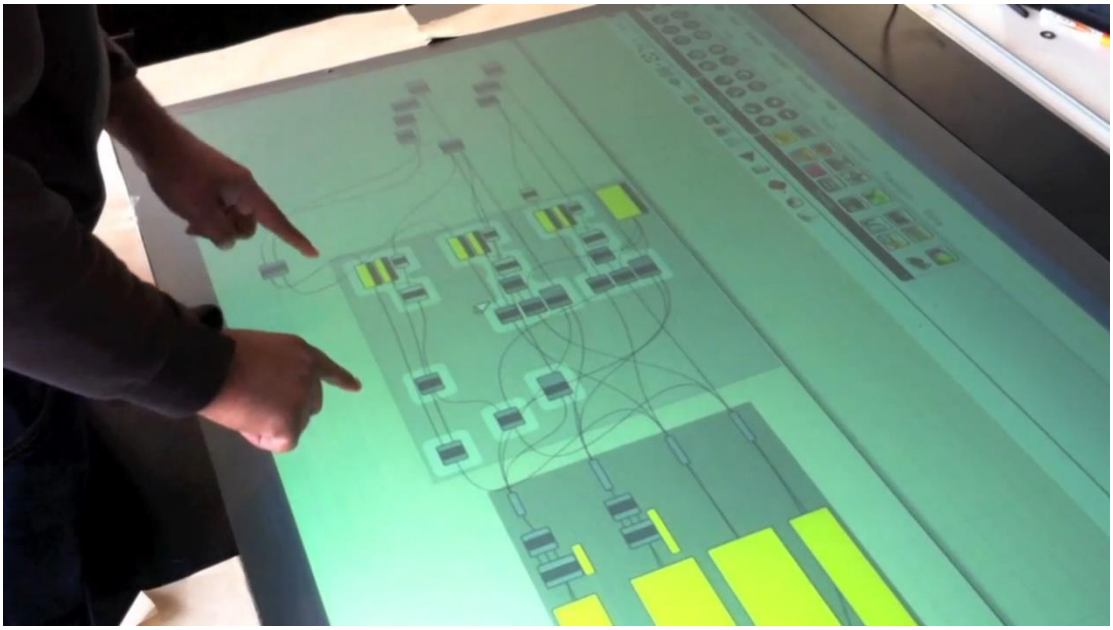
Εικόνα 2.14: Το JediBot επιτίθεται.

Συνδυάζοντας έναν ευκίνητο ρομποτικό βραχίονα, τις ικανότητες ιχνηλασίας του Kinect και ένα ευφυές λογισμικό, φοιτητές του Πανεπιστημίου Σταντφορντ δημιούργησαν το JediBot (βλέπε εικόνα 2.14).

Ο βραχίονας είναι εξοπλισμένος με ένα κόκκινο φωτόσπαθο και είναι προγραμματισμένος να σκοτώσει τον φοιτητή με το πράσινο φωτόσπαθο. Βασικά ο βραχίονας έχει κάποιες προκαθορισμένες επιθέσεις και αμύνεται χρησιμοποιώντας το Kinect για να εντοπίσει το πράσινο φωτόσπαθο. Το αποτέλεσμα είναι μία ανταγωνιστική μάχη ανθρώπου εναντίον μηχανής.

2.6.4 Kinect Multitouch Surface with Grasshopper

Αρχιτέκτονες και σχεδιαστές θα πρέπει σύντομα να προμηθευτούν αισθητήρες Kinect για επαγγελματική χρήση. Το Kinect MultiTouch Surface with Grasshopper (βλέπε εικόνα 2.15) είναι σχεδιασμένο να ενισχύει την επικοινωνία και την συνεργασία αρχιτεκτονικών σχεδίων και αλγορίθμων.



Εικόνα 2.15: Το λογισμικό Grasshopper με την βοήθεια του Kinect αλλάζει τον τρόπο εργασίας για αρχιτέκτονες και σχεδιαστές.

Το Kinect είναι στημένο από πάνω από το τραπέζι καθιστώντας την ακρίβεια και την αλληλεπίδραση άψογη. Διαφορετικά δεδομένα είναι δυνατό να σχηματιστούν και να μορφοποιηθούν χάρις το Kinect και το λογισμικό Grasshopper.

2.6.5 Kinect Real Hacking using Virtual Environment

Η τέχνη του hacking περνά σε εντελώς διαφορετικό επίπεδο με χρήση του Kinect. Το Kinect Real Hacking using Virtual Environment (βλέπε εικόνα 2.16) επιτρέπει τους χρήστες να παρακάμψουν τα τείχη ασφαλείας των Η/Υ καταστρέφοντας τα σε ένα περιβάλλον παιχνιδιού.

Εικόνα 2.16: Χακάρωντας έναν υπολογιστή με την βοήθεια του Kinect.

Ο χρήστης εισάγεται σε ένα σενάριο εικονικής πραγματικότητας χρησιμοποιώντας το Kinect για να κατευθυνθεί, με το πλαίσιο metasploit να είναι το θεμέλιο της εφαρμογής. Το εικονικό περιβάλλον είναι στη πραγματικότητα μία οπτική αναπαράσταση ενός άλλου Η/Υ ή προγράμματος ή τοίχου ασφαλείας. Έπειτα ο χρήστης καταστρέφει αντικείμενα που αντιπροσωπεύουν την ασφάλεια του υπολογιστή στόχου. Αυτό έχει ως αποτέλεσμα να πάρει τον έλεγχο ή αλλιώς να «χακάρει» αυτόν τον υπολογιστή.

2.6.6 Kinect Turntable 3D Scanner

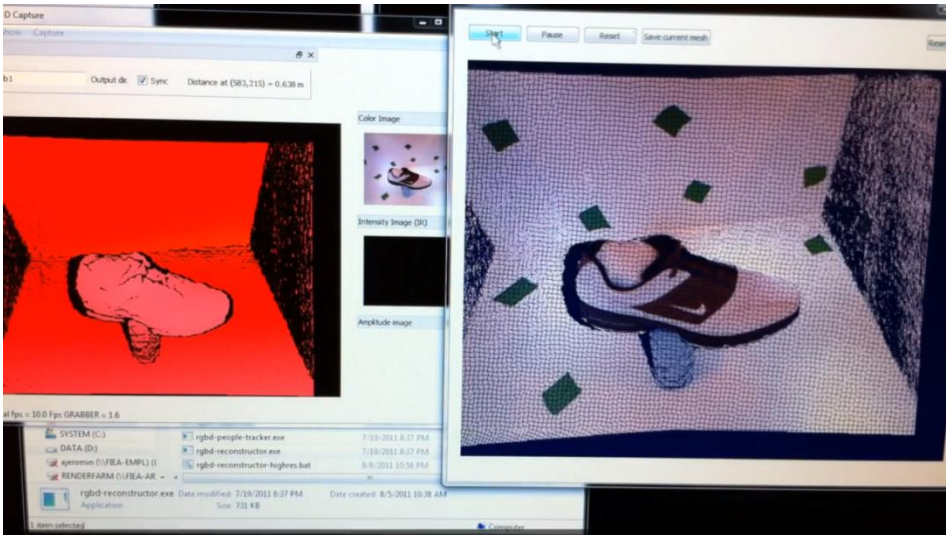
Με μικρό κόστος το Kinect μπορεί να μετατραπεί σε συσκευή σάρωσης τρισδιάστατων αντικειμένων. Το Kinect Turntable 3D Scanner είναι μία περιστρεφόμενη πλατφόρμα σάρωσης τρισδιάστατων αντικειμένων (βλέπε εικόνα 2.17).

Η σκέψη είναι εξαιρετικά απλή. Ένα κουτί κατασκευασμένο από αφρώδες χαρτί είναι τοποθετημένο πάνω σε μια περιστρεφόμενη πλατφόρμα. Στο κέντρο είναι

το αντικείμενο προς σάρωση, φωτιζόμενο από μικρά λαμπάκια LED σε κάθε γωνία του κουτιού. Με λίγο αρχικό κώδικα γίνεται η σύλληψη δεδομένων στο πλάι και από πάνω από το αντικείμενο καθώς περιστρέφεται.

Εικόνα 2.17: Το Kinect Turntable 3D Scanner.

Για να μετασχηματιστεί σε ψηφιακό τρισδιάστατο αντικείμενο στον υπολογιστή χρησιμοποιείται μία τροποποιημένη εκδοχή του προγράμματος RGBDemo (βλέπε εικόνα 2.18).



Εικόνα 2.18: Μετασχηματισμός τρισδιάστατου αντικειμένου.

2.7 Η επόμενη γενιά

Πολύ πρόσφατα (Μάιος 2013) ανακοινώθηκε το νέο Xbox One, ο διάδοχος του Xbox 360 και η τρίτη κονσόλα στην οικογένεια του Xbox. Αναμένεται να κυκλοφορήσει τον Νοέμβριο του 2013 με ανανεωμένη σχεδίαση, νέα αρχιτεκτονική κεντρικού επεξεργαστή και επεξεργαστή γραφικών και φυσικά με νέο αισθητήρα Kinect (βλέπε εικόνα 2.19).

Εικόνα 2.19: Το νέο Kinect.

Ο νέος αισθητήρας χρησιμοποιεί ευρυγώνια κάμερα, ανάλυσης 1920×1080, τύπου «χρόνου πτήσης». Οι κάμερες αυτές (Time-of-flight, ToF) βρίσκουν την απόσταση βασιζόμενες στην δεδομένη ταχύτητα του φωτός, μετρώντας δηλαδή τον χρόνο πτήσης του σήματος φωτός ανάμεσα στην κάμερα και το αντικείμενο. Το ανανεωμένο Kinect θα επεξεργάζεται δύο γιγαμπίτ ($2×10^9$ bits) δεδομένων το δευτερόλεπτο για να διαβάσει το περιβάλλον του. Έχει μεγαλύτερη ακρίβεια, με τρεις φορές αυξημένη πιστότητα σε σχέση με τον προκάτοχο του. Ακόμα έχει και την ικανότητα να βλέπει στο σκοτάδι χάρις το νέο ενεργό υπέρυθρο αισθητήρα. Έχει κατά 60% αυξημένο οπτικό πεδίο και μπορεί να ανιχνεύσει έξι σκελετούς μονομιάς. Μπορεί επίσης να ανιχνεύσει τον καρδιακό ρυθμό ενός χρήστη, τις εκφράσεις του προσώπου του, 25 ξεχωριστές αρθρώσεις (μέχρι και τον αντίχειρα) και την ακριβή περιστροφή τους, το βάρος που ασκείται σε κάθε άρθρωση, την ταχύτητα των κινήσεων και να ανιχνεύσει χειρονομίες με το χειριστήριο. Το μικρόφωνο θα παραμένει πάντα ενεργό για να είναι έτοιμο να λάβει φωνητικές εντολές από τον χρήστη, ακόμα και αν η κονσόλα είναι σε λειτουργία ύπνου.

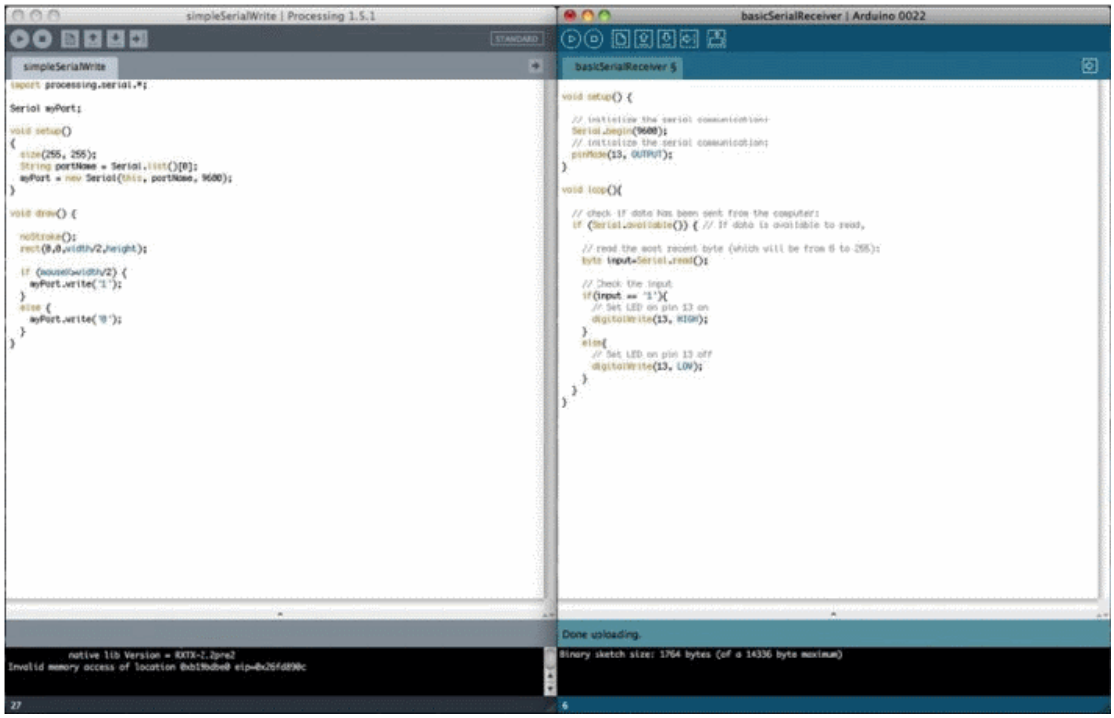
Αντίθετα με το Kinect για Xbox 360 (στο οποίο ήταν προαιρετικό αξεσουάρ), το Xbox One δεν θα λειτουργεί χωρίς συνδεδεμένο τον αισθητήρα Kinect. Ωστόσο οι χρήστες διατηρούν μέσω του λογισμικού την ικανότητα να απενεργοποιήσουν τις δυνατότητες του Kinect ενώ παραμένει συνδεδεμένο στην κονσόλα.

Κεφάλαιο 3ο

Processing

3.1 Γενικά

Ο πιο άμεσος τρόπος διασύνδεσης του Arduino και του Kinect είναι μέσω του Processing. Το Processing είναι μία γλώσσα προγραμματισμού ανοιχτού κώδικα, και ένα ολοκληρωμένο περιβάλλον ανάπτυξης, επίσης ανοιχτού κώδικα, βασισμένο στην Java. Το Arduino IDE μοιάζει πολύ με το Processing IDE διότι βασίζεται σε αυτό (βλέπε εικόνα 3.1). Ένα από τα καλύτερα χαρακτηριστικά του Processing είναι, η ύπαρξη μίας τεράστιας κοινότητας από ανθρώπους, που συνεχώς το εμπλουτίζουν με βιβλιοθήκες. Το Kinect δεν αποτελεί εξαίρεση, συνεπώς το Processing μπορεί να επικοινωνήσει με συσκευές Kinect, μέσω διαφόρων βιβλιοθηκών που είναι διαθέσιμες στην ιστοσελίδα του. Το Processing μπορεί επίσης να επικοινωνήσει με το Arduino μέσω σειριακής επικοινωνίας.



Εικόνα 3.1: Το Processing IDE και το Arduino IDE δίπλα δίπλα.

3.2 Γλώσσα προγραμματισμού

Ο Casey Reas και ο Ben Fry, πρώην μέλη του γκρουπ Αισθητικής και Υπολογισμού του Ψηφιακού Εργαστηρίου του MIT, δημιούργησαν το Processing το 2001. Ένας από τους βασικούς σκοπούς του Processing είναι να δράσει σαν εργαλείο προγραμματισμού για τους μη-προγραμματιστές. Στην ιστοσελίδα του Processing, αναγράφει την παρακάτω εισαγωγή:

«Το Processing είναι μια γλώσσα προγραμματισμού ανοιχτού κώδικα και ένα περιβάλλον για ανθρώπους που θέλουν να σχεδιάσουν εικόνες, animation και αλληλεπιδράσεις. Αρχικά αναπτύχθηκε σαν ψηφιακό μπλοκ ιχνηλασίας όπως επίσης και για να διδάξει τα θεμελιώδη στον προγραμματισμό εντός ενός οπτικού περιβάλλοντος. Αργότερα εξελίχθηκε σε εργαλείο δημιουργίας τελειωμένων επαγγελματικών project. Σήμερα, υπάρχουν δεκάδες χιλιάδες φοιτητές, καλλιτέχνες, σχεδιαστές, ερευνητές και χομπίστες που χρησιμοποιούν το Processing για εκμάθηση, δημιουργία πρωτοτύπων και παραγωγή.»

Το Processing βασίζεται στη Java, μία από τις πιο ευρέως διαδεδομένες γλώσσες προγραμματισμού. Η Java είναι μία αντικειμενοστραφής γλώσσα προγραμματισμού. Ο κώδικας που γράφεται σε Java μεταγλωττίζεται σε κώδικα μηχανής και έπειτα εκτελείται από την Εικονική Μηχανή της Java (Java Virtual Machine, JVM) που υπάρχει στον Η/Υ μας. Αυτό επιτρέπει στον προγραμματιστή να γράψει λογισμικό, χωρίς να ανησυχεί για το λειτουργικό σύστημα που αυτό θα εκτελεστεί. Αυτό αποτελεί μεγάλο πλεονέκτημα για τον προγραμματιστή, όταν αυτός προσπαθεί να γράψει λογισμικό που θα χρησιμοποιηθεί σε διαφορετικά συστήματα.

Τα προγράμματα του Processing ονομάζονται σκίτσα (sketches). Ο λόγος είναι ότι το Processing αρχικά είχε αναπτυχθεί σαν εργαλείο σχεδιαστών, για να καταγράψουν γρήγορα τις ιδέες τους, οι οποίες αργότερα θα αναπτύσσονταν σε άλλες γλώσσες προγραμματισμού. Κατά την διάρκεια της εξέλιξης του όμως, επέκτεινε τις δυνατότητες του, και τώρα χρησιμοποιείται σε πολλά περίπλοκα project παράλληλα με την αρχική του χρήση.

3.3 Εγκατάσταση

Η εγκατάσταση του Processing είναι εύκολη. Αρκεί μια επίσκεψη στο http://processing.org/download/ όπου προσφέρεται για download η κατάλληλη έκδοση, σύμφωνα με το λειτουργικό σας σύστημα.

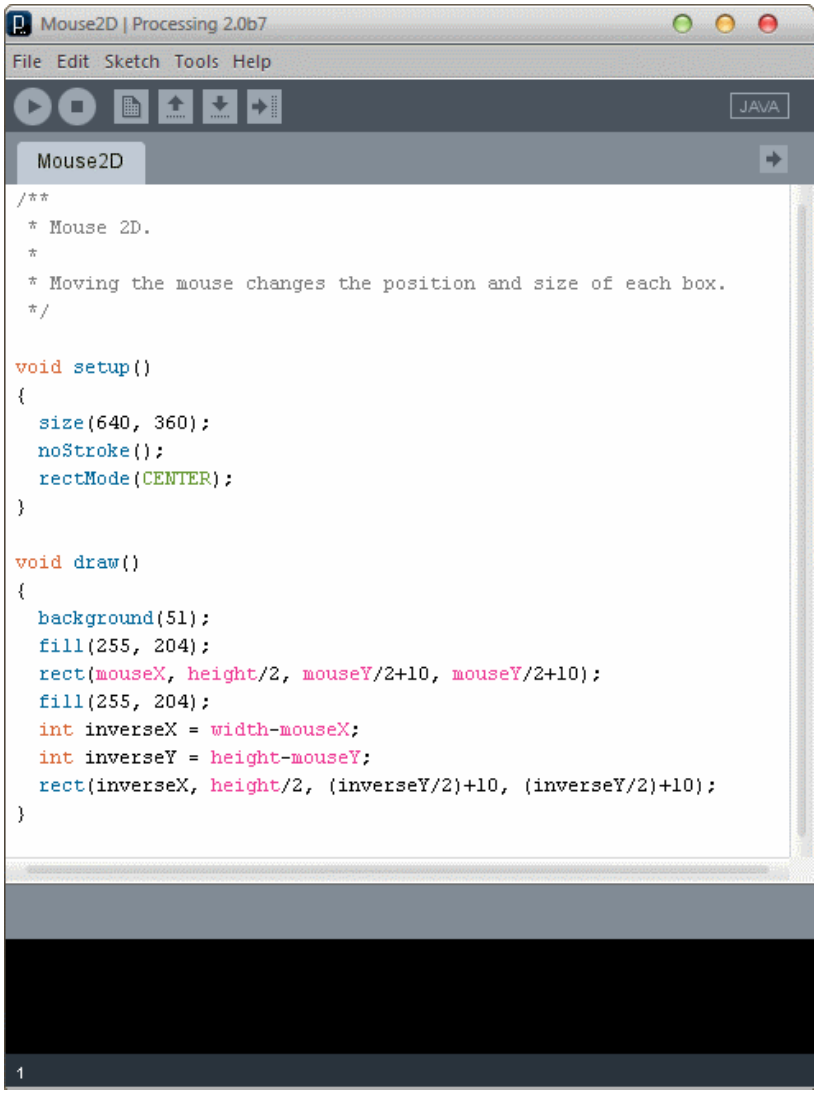
3.4 Ολοκληρωμένο περιβάλλον ανάπτυξης (Processing IDE)

Όπως προείπαμε το Processing IDE μοιάζει πολύ με το Arduino IDE (βλέπε εικόνα 3.2). Διαθέτει μία κονσόλα στο κάτω μέρος, μία περιοχή μηνυμάτων που προειδοποιεί για σφάλματα, έναν κειμενογράφο όπου γράφεται ο κώδικας και μια μπάρα εργαλείων στο πάνω μέρος.

Η μπάρα εργαλείων έχει όμοια δομή κουμπιών με το Arduino, εν τούτοις υπάρχουν μικρές διαφορές. Τα κουμπιά «Νέο», «Άνοιγμα» και «Αποθήκευση» λειτουργούν όπως και στο Arduino. Αντί για τα κουμπιά «Επαλήθευση» και

«Φόρτωση» στα αριστερά, υπάρχουν τα κουμπιά «Run» και «Stop». Η λειτουργία τους είναι η προφανής, το «Run» εκτελεί την εφαρμογή και το «Stop» σταματάει την εκτέλεση. Υπάρχει και ένα άλλο κουμπί που ονομάζεται «Export Application». Η λειτουργία του είναι να εξάγει τον κώδικα σε εφαρμογή Java και ανοίγει τον φάκελο που περιέχει όλα τα αρχεία της μετατροπής.

Το κουμπί «JAVA» στα δεξιά χρησιμοποιείται για την αλλαγή λειτουργίας του περιβάλλοντος. Οι επιλογές είναι: Java, Android, JavaScript και Experimental. Υπάρχει και η επιλογή Add Mode που μας επιτρέπει να προσθέσουμε κάποια διαφορετική λειτουργία από τις διαθέσιμες.



Εικόνα 3.2: Το Processing IDE.

3.5 Η γλώσσα του Processing

Το Processing σχεδιάστηκε για απλότητα στη χρήση και μας επιτρέπει να δημιουργήσουμε εκτελέσιμα προγράμματα μόνο με λίγες γραμμές κώδικα. Η γλώσσα του, βασίζεται στη Java. Όταν γράφουμε ένα σχεδιάγραμμα Processing, μπορούμε να κάνουμε χρήση των πολλών διαθέσιμων βιβλιοθηκών, που είτε περιλαμβάνονται στο Processing IDE, είτε μπορούμε να κατεβάσουμε από το διαδίκτυο. Όμοια με το Arduino τα εκτελέσιμα προγράμματα περιλαμβάνουν σε δύο βασικές συναρτήσεις: την `setup()` και την `draw()`.

3.5.1 Η συνάρτηση `setup()`

Η συνάρτηση `setup()` εκτελείται μόνο μια φορά στη ζωή του προγράμματος. Χρησιμοποιείται συνήθως για αρχικοποίηση μεταβλητών και την ρύθμιση των παραμέτρων του προγράμματος.

3.5.2 Η συνάρτηση `draw()`

Η συνάρτηση `draw()` εκτελείται συνεχώς σε έναν ατέρμονο βρόχο, ώσπου η εκτέλεση του προγράμματος τερματιστεί.

3.5.3 Μεταβλητές

Όπως αναφέρθηκε και στο πρώτο κεφάλαιο, οι μεταβλητές είναι συμβολικά ονόματα που χρησιμοποιούνται για την αποθήκευση δεδομένων σε ένα πρόγραμμα. Υπάρχουν οκτώ στοιχειώδης τύποι δεδομένων στη Java και υποστηρίζονται όλοι από το Processing. Οι τύποι αυτοί είναι: byte, short, int, long, float, double, boolean και char.

Οι ακέραιοι (int) χρησιμοποιούνται για την αποθήκευση θετικών και αρνητικών αριθμών χωρίς υποδιαστολή. Το εύρος των αριθμών είναι από το 2.147.483.648

ως το -2.147.483.647. Αν πρέπει να αποθηκεύσουμε μεγαλύτερους αριθμούς, χρησιμοποιούμε τον τύπο δεδομένων long.

Όποτε χρειαζόμαστε μεγαλύτερη ακρίβεια, κάνουμε χρήση του τύπου δεδομένων float. Οι δεκαδικοί (float) χρησιμοποιούνται για την αποθήκευση θετικών και αρνητικών αριθμών με κινητή υποδιαστολή. Ο τύπος δεδομένων double είναι όμοιος με τον float, όμως είναι ακόμα πιο ακριβής.

Ο τύπος δεδομένων boolean λαμβάνει δύο τιμές: αληθής και ψευδής (true και false).

Ο τύπος δεδομένων για χαρακτήρες (char) χρησιμοποιείται για την αποθήκευση τυπογραφικών συμβόλων.

3.5.4 Εμβέλεια μεταβλητών

Όποτε δηλώνουμε μία μεταβλητή, θέτουμε ταυτόχρονα και την εμβέλεια της ή αλλιώς την σφαίρα επιρροής της. Αν την δηλώσουμε μέσα σε κάποια συνάρτηση τότε είναι προσβάσιμη μόνο μέσα από την ίδια συνάρτηση. Αν την δηλώσουμε έξω από οποιαδήποτε συνάρτηση, τότε θεωρείται παγκόσμια μεταβλητή και είναι προσβάσιμη από οπουδήποτε συνάρτηση.

3.6 Οι βιβλιοθήκες

Οι βιβλιοθήκες του Processing είναι γραμμές κώδικα Java, πακεταρισμένες σε ένα αρχείο .jar και τοποθετημένες στο φάκελο βιβλιοθηκών του Processing. Υπάρχουν δύο είδη βιβλιοθηκών: οι βασικές και οι συμβαλλόμενες βιβλιοθήκες. Οι βασικές, όπως η OpenGL και η Serial, περιλαμβάνονται στο Processing ενώ οι συμβαλλόμενες δημιουργούνται και συντηρούνται από μέλη της κοινότητας. Για να χρησιμοποιήσουμε τις συμβαλλόμενες βιβλιοθήκες πρέπει πρώτα να τις κατεβάσουμε και έπειτα να τις αποθηκεύσουμε στον φάκελο βιβλιοθηκών του Processing.

Υπάρχουν τρείς βιβλιοθήκες διαθέσιμες για το Kinect (βλέπε πίνακα 3). Δύο από αυτές υποστηρίζουν συγκεκριμένα λειτουργικά συστήματα και μία που δουλεύει σε όλα. Επίσης, διαφοροποιούνται στις λειτουργίες που διαθέτουν.

Βιβλιοθήκη	Συγγραφέας	Βασίζεται στο	Λειτουργικό σύστημα
OpenKinect	Daniel Shiffman	OpenKinect/Libfreenect	Mac OS X
dLibs	Thomas Diewald	OpenKinect/Libfreenect	Windows
Simple-openni	Max Rheiner	OpenNI/NITE	Linux, Mac OS X, Windows

Πίνακας 3. Βιβλιοθήκες για το Kinect.

Στο προηγούμενο κεφάλαιο εξηγήθηκε πως αναπτύχθηκε το Kinect και ποιοι διαφορετικοί τρόποι πρόσβασης υπάρχουν στη ροή δεδομένων του. Παρατηρώντας τον πίνακα 3, συνειδητοποιούμε ότι δύο από τους τρόπους αυτούς βασίζονται στο OpenKinect και διατίθενται για ένα μόνο λειτουργικό σύστημα. Στη παρούσα εργασία θα χρησιμοποιήσουμε την βιβλιοθήκη Simple-openNI για δύο λόγους. Πρώτον επειδή υποστηρίζει πολλαπλά λειτουργικά συστήματα και δεύτερον επειδή κτίστηκε γύρω από το ενδιάμεσο λογισμικό (middleware) OpenNI/NITE που αναλύσαμε και επιλέξαμε στο προηγούμενο κεφάλαιο.

3.7 Simple-OpenNI

Η βιβλιοθήκη simple-openni είναι ένα «περίβλημα» του NITE και του OpenNI για το Processing. Δημιουργήθηκε από τον Max Rheiner, έναν καλλιτέχνη πολυμέσων και προγραμματιστή λογισμικού που διδάσκει στο Πανεπιστήμιο της Ζυρίχης.

Το NITE και το OpenNI είναι γραμμένα σε C++. Ο Rheiner έβαλε όλες τις λειτουργίες και τις μεθόδους σε ένα περίβλημα προκειμένου να είναι προσβάσιμες από την Java. Έπειτα, δημιούργησε μία βιβλιοθήκη για να κάνει

τον κώδικα Java διαθέσιμο στο Processing. Αυτό μας επιτρέπει να έχουμε το OpenNI και NITE διαθέσιμα και με πλήρη λειτουργικότητα.

Αναλυτικές οδηγίες εγκατάστασης υπάρχουν στο http://code.google.com/p/simple-openni/wiki/Installation

3.8 Λειτουργίες του OpenNI/NITE

Για να γίνει χρήση της ιχνηλασίας σκελετού και χεριών χρειαζόμαστε μία επιπλέον εξωτερική μονάδα, πέρα από το OpenNI. Η μονάδα αυτή είναι το NITE, το οποίο περιλαμβάνει όλους τους αλγόριθμους που ανέπτυξε η PrimeSense για την αλληλεπίδραση χρήστη.

3.8.1 Ιχνηλασία χεριών

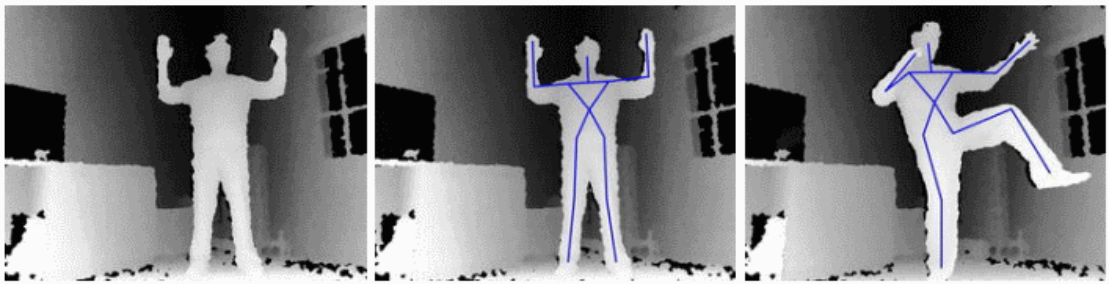
Η επιδέξια χρήση των άνω άκρων μας, καθιστά την ιχνηλασία χεριών πολύ σημαντική για την φυσική αλληλεπίδραση. Μπορούμε να τοποθετήσουμε τα χέρια μας με ακρίβεια στον χώρο και μπορούμε να κάνουμε μεγάλο αριθμό χειρονομιών. Οι ικανότητες μπορούν να χρησιμοποιηθούν για να οδηγήσουν παραμέτρους ή να προκαλέσουν διαφορετικές συμπεριφορές στις εφαρμογές μας.

3.8.2 Ιχνηλασία σκελετού

Η ιχνηλασία σκελετού είναι μάλλον η πιο εντυπωσιακή δυνατότητα του NITE, και του Kinect. Αυτή η δυνατότητα επιτρέπει στον υπολογιστή να καταλάβει την θέση του σώματος ενός ατόμου σε τρεις διαστάσεις και να έχει μία σχετικά ακριβής ιδέα για το που βρίσκονται σε κάθε στιγμή οι αρθρώσεις του. Ένα απίστευτο χαρακτηριστικό αν αναλογιστούμε ότι πριν από λίγο καιρό θα χρειαζόμασταν έναν αποκαρδιωτικό αριθμό υλικών (που περιλαμβάνει αρκετές κάμερες, ειδικές στολές κλπ.) για να αποκτήσουμε τα ίδια δεδομένα.

3.8.3 Βαθμονόμηση

Για να μπορέσει ο αλγόριθμος του OpenNI να ξεκινήσει την ιχνηλασία των αρθρώσεων ενός ατόμου, πρέπει το άτομο αυτό να σταθεί σε μία δεδομένη στάση σώματος (βλέπε εικόνα 3.3 αριστερά). Συγκεκριμένα πρέπει να σταθεί με τα πόδια ενωμένα και τα χέρια σηκωμένα πάνω από τους ώμους δίπλα από το κεφάλι. Αυτή η στάση έχει διάφορες ονομασίες. Στην τεχνική βιβλιογραφία και στην τεκμηρίωση της PrimeSense αναφέρεται ως «στάση Ψ», λόγω της ομοιότητας με το συγκεκριμένο γράμμα. Άλλοι προγραμματιστές την αποκαλούν «στάση υπακοής», επειδή παρομοιάζει την στάση σώματος που θα έπαιρνε κάποιος υπό την απειλή πυροβόλου όπλου.



Εικόνα 3.3: Αρχική στάση, βαθμονόμηση και ιχνηλασία.

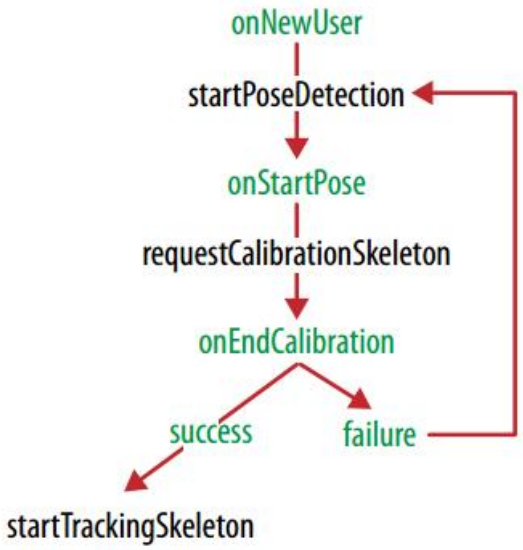
Με όποιο όνομα και να αναφέρεται, πάντα εγείρει σχόλια ανάμεσα σε προγραμματιστές και σχεδιαστές που δουλεύουν με το Kinect. Αφ' ενός μερικές φορές είναι τουλάχιστον άβολη. Αν για παράδειγμα η εφαρμογή που αναπτύσσει κάποιος ιχνηλατεί ανθρώπους σε καθημερινές δραστηριότητες, η απαίτηση βαθμονόμησης αποτελεί ένα αδέξιο εμπόδιο. Αφ' ετέρου κάποιοι ενοχλούνται από τη δυνατότητας ιχνηλασίας χωρίς σαφή βαθμονόμηση. Η ιδέα ότι κάμερες ιχνηλασίας μπορούν να παρατηρούν την κάθε μας κίνηση, εν αγνοία μας, σε δημόσιο μέρος είναι δυνητικά ενοχλητική.

Άσχετα από την εφαρμογή ή την ερμηνεία των παραπάνω, η βαθμονόμηση απαιτεί την προσοχή του προγραμματιστή. Είναι εκπληκτικό το πόσο έντονα αντιδρούν οι χρήστες στη διαδικασία αυτή. Εν τέλει η βαθμονόμηση αποτελεί μία

ενόχληση που πρέπει να ανεχτούμε. Στην τελευταία έκδοση του επίσημου SDK της Microsoft έχει επιδιορθωθεί και πιθανότατα θα εκλείψει και από το NITE.

3.8.3.1 Τα βήματα της βαθμονόμησης

Στη εικόνα 3.4 απεικονίζεται το διάγραμμα ροής της διαδικασίας βαθμονόμησης. Φαίνεται η ροή των οπισθοκλήσεων (callbacks) που πυροδοτούνται από το OpenNI και οι ενέργειες που πρέπει να γίνουν ανάμεσα τους για να συνεχίσει η βαθμονόμηση.



Εικόνα 3.4: Διάγραμμα ροής της διαδικασίας βαθμονόμησης.

Τα τρία βήματα της διαδικασίας βαθμονόμησης είναι: η βασική ανίχνευση χρήστη, η βαθμονόμηση σε εξέλιξη και πλήρως ανιχνευμένος χρήστης. Όταν το σκίτσο τρέξει για πρώτη φορά, το OpenNI δεν θα ανιχνεύσει κανέναν χρήστη. Οι συναρτήσεις εισόδου χρηστών, που παρέχει, θα επιστρέψουν μία κενή λίστα. Η διαδικασία θα ξεκινήσει όταν ένας χρήστης εισέλθει στη σκηνή. Εκείνη τη στιγμή το OpenNI ανιχνεύει την είσοδο του χρήστη και τον θέτει υποψήφιο προς βαθμονόμηση. Καλεί την συνάρτηση `onNewUser` μέσα στην οποία εισάγουμε την διαδικασία της βαθμονόμησης καλώντας την συνάρτηση `startPoseDetection`. Η συνάρτηση αυτή κάνει το Kinect να παρακολουθεί τον χρήστη και να ελέγχει αν έχει πάρει την αρχική στάση. Όταν γίνει αυτό θα

κληθεί η συνάρτηση `onStartPose`. Επόμενο βήμα είναι η κλήση της συνάρτησης `requestCalibrationSkeleton`, που πυροδοτεί την ανίχνευση του σκελετού του χρήστη. Όταν ολοκληρωθεί καλείται η συνάρτηση `onEndCalibration` οπού αναφέρει το αποτέλεσμα της βαθμονόμησης. Αν όλα πήγαν καλά καλείται η συνάρτηση `startTrackingSkeleton` για να ξεκινήσει η ιχνηλασία του σκελετού και η πρόσβαση στα δεδομένα. Αν η βαθμονόμηση δεν ολοκληρώθηκε σωστά η διαδικασία ξεκινά από την αρχή, από την συνάρτηση `startPoseDetection`.

Κεφάλαιο 4ο

Κατασκευή

4.1 Γενικά

Ένα από τα πιο απαιτητικά μέρη της εργασίας είναι η κατασκευή ενός στηρίγματος-βάσης για την μαριονέτα. Ταυτόχρονα είναι και από τα πιο κρίσιμα για την σταθερή στήριξη των σερβοκινητήρων που δίνουν ζωή στην μαριονέτα.

4.1.1 Πορεία σκέψης για την βάση

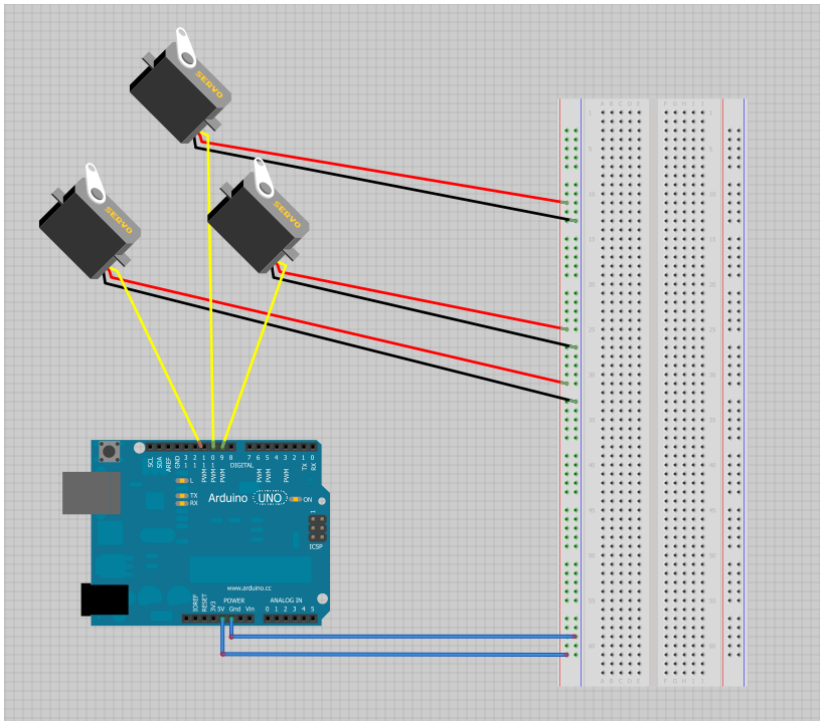
Το σχήμα της βάσης ήταν από την αρχή ξεκάθαρο ότι θα ήταν παραλληλόγραμμο, στα πρότυπα ενός μεγάλου χαρτοκιβωτίου. Η ιδέα του χαρτοκιβωτίου σαν βάση όμως, απορρίφθηκε λόγω ελλιπούς στήριξης και σταθερότητας. Σαν υλικό εντέλει, επιλέχθηκε το ξύλο και η βάση μετασχηματίστηκε από παραλληλόγραμμο χαρτοκιβωτίου σε ξύλινο «σκαμπό». Αυτό όμως, δεν ήταν αρκετά ψηλό για να στηρίξει επαρκώς την μαριονέτα, όποτε προστέθηκε μία «πλάτη». Ο σκοπός αυτού ήταν να υπερυψωθεί η μαριονέτα αρκετά ώστε να αιωρείται σε χαμηλό ύψος από το πάτωμα. Η βάση κατασκευάστηκε σε παλιό ξυλουργείο, χρησιμοποιώντας διάφορα ρετάλια που δεν είχαν χρησιμότητα στον ξυλουργό.

4.1.2 Τοποθέτηση σερβοκινητήρων

Οι σερβοκινητήρες έπρεπε να τοποθετηθούν από πάνω από την μαριονέτα για να είναι πιο εμφανής η κίνηση της. Τα σχοινιά της, έπρεπε να είναι όσο γίνεται πιο κάθετα από το έδαφος για τον ίδιο λόγο. Οι κινητήρες είναι τρείς, όσα και τα σχοινιά της μαριονέτας. Δύο για τα χέρια και ένα κοινό σχοινί για τα πόδια. Λαμβάνοντας υπ' όψιν αυτή την ιδιαιτερότητα, ο κινητήρας των ποδιών πρέπει να είναι πιο ψηλά από τους άλλους δύο, αλλιώς η κίνηση των ποδιών θα είναι πολύ μικρή και ίσως μη αντιληπτή. Γι' αυτό τον λόγο η «πλάτη» έχει τριγωνικό σχήμα, με τον κινητήρα υπεύθυνο για την κίνηση των ποδιών, να βρίσκεται στην κορυφή και τους άλλους δύο λίγο πιο κάτω, συμμετρικά μεταξύ τους. Ύστερα από λίγο πριόνισμα, οι κινητήρες τοποθετήθηκαν σύμφωνα με τις παραπάνω προδιαγραφές.

4.2 Σύνδεση

Τα καλώδια των κινητήρων πέφτουν από την πίσω μεριά της «πλάτης» όπου και συνδέονται με το Arduino στις εξόδους PWM με την βοήθεια ενός τυπικού breadboard (βλέπε εικόνα 4.1).



Εικόνα 4.1: Σύνδεση κινητήρων με το Arduino.

Το Arduino συνδέεται και τροφοδοτείται από φορητό υπολογιστή. Η τροφοδοσία αυτή είναι επαρκής για τους τρείς κινητήρες και δεν χρησιμοποιήθηκε εξωτερική τροφοδοσία. Το Kinect συνδέεται επίσης με τον υπολογιστή με μία ιδιαιτερότητα. Στην εργασία αυτή γίνεται χρήση του Kinect για Xbox 360. Το συγκεκριμένο Kinect, αντίθετα από το Kinect for Windows, δεν συνδέεται απευθείας στον υπολογιστή. Είναι απαραίτητη η χρήση ενός αντάπτορα-μετασχηματιστή (βλέπε εικόνα 4.2) προκειμένου να συνδεθεί επιτυχώς.

Εικόνα 4.2: Μετασχηματιστής-αντάπτορας του Kinect.

4.2 Τελική κατασκευή

Η τελική κατασκευή απεικονίζεται στην εικόνα 4.3 παρακάτω.

Εικόνα 4.3: Τελική κατασκευή.

Κεφάλαιο 5ο

Κώδικας

5.1 Ανάπτυξη του κώδικα

Το ποιο κρίσιμο μέρος της εργασίας είναι ο προγραμματισμός. Στην εργασία αυτή χωρίζεται σε δύο μέρη: Στον προγραμματισμό του Kinect και στον προγραμματισμό του Arduino.

Το Kinect είναι ο αισθητήρας της εργασίας. Με αυτόν θα ανιχνεύεται το σώμα του όποιου οι κινήσεις θα κατοπτρίζονται στη μαριονέτα. Για τον προγραμματισμό του, θα χρησιμοποιήσουμε το framework της Primesense, OpenNI μαζί με το αναπόσπαστο NITE το οποίο μας προσφέρει την πολυπόθητη ιχνηλασία σκελετού, όπως αναλύσαμε στα κεφάλαια 2.4.4 και 3.8. Συγκεκριμένα θα χρησιμοποιήσουμε την αντίστοιχη βιβλιοθήκη simple-openni υλοποιημένη για το Processing (βλέπε κεφάλαιο 3.7).

Η υλοποίηση θα λάβει μέρος στο Processing IDE (βλέπε κεφάλαιο 3.4), οι βιβλιοθήκες του οποίου επιτρέπουν την αρμονική συνεργασία Kinect και Arduino.

Το Arduino θα προσδώσει κίνηση στην εργασία. Είναι υπεύθυνο για τον έλεγχο των σερβοκινητήρων. Για τον προγραμματισμό του, θα χρησιμοποιήσουμε το

ομώνυμο Arduino IDE (βλέπε κεφάλαιο 1.7). Ο κώδικας, όπως σε κάθε μικροελεγκτή, θα «φορτωθεί» στη πλακέτα και θα παραμείνει εκεί (στη ROM του). Όταν ανοίξει η επικοινωνία από το Processing, θα είναι έτοιμος για τη συνεχή ροή δεδομένων που θα μετασχηματιστεί σε κίνηση από τους σερβοκινητήρες.

5.1.1 Περιγραφή κώδικα στο Arduino

Ο κώδικας στο Arduino είναι πολύ απλός. Πρώτο βήμα είναι η εισαγωγή της βιβλιοθήκης `Servo` του Arduino. Η βιβλιοθήκη αυτή θα διευκολύνει την σύνδεση και τον έλεγχο των σερβοκινητήρων. Μετά την εισαγωγή της, δηλώνονται τρία αντικείμενα τύπου `Servo` που ουσιαστικά είναι οι κινητήρες μας. Ένας για τα γόνατα, ένας για το δεξί αγκώνα και ένας για τον αριστερό αγκώνα.

Έπειτα δηλώνονται οι μεταβλητές που θα χρησιμοποιήσουμε για να παρακολουθήσουμε τις αλλαγές στις γωνίες των σερβοκινητήρων που θα έρχονται από το Processing. Δηλώνονται δύο μεταβλητές για αυτή την δουλειά: ο ακέραιος `nextServo` όπου καθορίζει ποιος κινητήρας θα κινηθεί και ο πίνακας ακεραίων `servoAngles`, τριών θέσεων, όπου αποθηκεύονται οι γωνίες για τους σερβοκινητήρες. Η μεταβλητή `nextServo` παίρνει τις τιμές 0,1,2 και χρησιμοποιείται στον πίνακα `servoAgles` για να διαβάσουμε τις αποθηκευμένες τιμές γωνιών.

Στη ρουτίνα `setup()` με την υλοποίηση της συνάρτησης `attach()`, λέμε στο Arduino σε ποια pin είναι συνδεδεμένοι οι κινητήρες. Τώρα θα μπορούμε να χρησιμοποιήσουμε τα αντικείμενα τύπου `Servo` που δηλώσαμε πριν για να κινήσουμε τον κατάλληλο κινητήρα. Τέλος προετοιμάζουμε την σειριακή επικοινωνία θέτοντας ταυτόχρονα το ρυθμό μετάδοσης σε 9600 bps (bytes per second). Αυτός ο ρυθμός μετάδοσης είναι τυπικός για τέτοιου είδους εφαρμογή όπου αποστέλλονται απλά δεδομένα και δεν θα προκαλέσει αισθητές καθυστερήσεις.

Στην ρουτίνα συνεχούς επανάληψης `loop()` αρχικά αφουγκραζόμαστε αν κάποιος έστειλε δεδομένα στη σειριακή θύρα. Η συνάρτηση `Serial.available()` θα γίνει αληθής αν υπάρχουν δεδομένα στη σειριακή θύρα και ψευδής αν δεν υπάρχουν. Όταν τα δεδομένα έρθουν, δηλώνεται μία ακέραια μεταβλητή `servoAngle` στην οποία και αποθηκεύονται. Έπειτα τοποθετούμε τις τιμές των δεδομένων στον πίνακα μας. Η μεταβλητή `nextServo` χρησιμοποιείται σαν δείκτης στον πίνακα που θα αποθηκεύσουμε την νέα τιμή δεδομένων. Η μεταβλητή `nextServo` παίρνει τις τιμές 0,1,2 που σημαίνει ότι η τιμή 0 είναι για την γωνία των γόνατων, η τιμή 1 για την γωνία του δεξιού αγκώνα και η τιμή 2 για την γωνία του αριστερού αγκώνα.

Η μεταβλητή `nextServo` ξεκινάει από το 0, όπου αναμένεται το πρώτο δεδομένο να είναι η γωνία των γονάτων. Μόλις γίνει η λήψη του δεδομένου αυτού, αυξάνεται η τιμή της μεταβλητής κατά ένα ώστε να περιμένει το επόμενο δεδομένο, που θα είναι η γωνία του δεξιού αγκώνα, για να αποθηκευτεί στη σωστή θέση του πίνακα. Το ίδιο γίνεται και για την γωνία του αριστερού αγκώνα. Επειδή η συνεχής αύξηση της μεταβλητής κατά ένα, δεν μας εξυπηρετεί, γιατί θέλουμε να παίρνει μόνο τις τιμές 0,1,2, ελέγχουμε αν έχει πάρει τιμή μεγαλύτερη από 2 και αν ισχύει αυτό, τη μηδενίζουμε. Έτσι πετυχαίνουμε την διακύμανση στις τιμές που επιθυμούμε.

Τέλος, το μόνο που μένει είναι να σταλθούν οι πιο πρόσφατες τιμές στους σερβοκινητήρες. Αυτό επιτυγχάνεται με την συνάρτηση `write()` και αντίστοιχη θέση του πίνακα: `servoAngles[0]` για τα γόνατα, `servoAngles[1]` για το δεξί αγκώνα και `servoAngles[2]` για τον αριστερό αγκώνα.

5.1.2 Περιγραφή κώδικα στο Processing

Και εδώ, προηγείται η εισαγωγή των κατάλληλων βιβλιοθηκών. Έτσι, εισάγεται η βιβλιοθήκη `SimpleOpenNI` και η βιβλιοθήκη `serial`. Δημιουργείται ένα αντικείμενο της βιβλιοθήκης `SimpleOpenNI` με όνομα `kinect`, παίζοντας τον ρόλο του αισθητήρα μας και ένα αντικείμενο της βιβλιοθήκης `serial`, παίζοντας τον ρόλο της σειριακής θύρας επικοινωνίας με το Arduino.

Στη ρουτίνα `setup()` αρχικά δημιουργείται ένα παράθυρο μεγέθους 640×480 pixel που είναι το μέγεθος της εικόνας που λαμβάνουμε από το Kinect. Αμέσως μετά, δίνουμε υπόσταση στο αντικείμενο `kinect` που δημιουργήσαμε προηγουμένως με την βιβλιοθήκη `SimpleOpenNI`. Με την συνάρτηση `kinect.enableDepth()` λέμε στην βιβλιοθήκη ότι θέλουμε να έχουμε πρόσβαση στην εικόνα βάθους. Έπειτα, με την συνάρτηση `kinect.SimpleOpenNI.SKEL.PROFILE_ALL` ενεργοποιείται η ανίχνευση χρήστη. Συγκεκριμένα, δηλώνεται στη βιβλιοθήκη ότι επιθυμούμε ιχνηλασία όλων των αρθρώσεων του χρήστη. Παρακάτω, με την συνάρτηση `kinect.setMirror()` αναστρέφεται η εικόνα ώστε η οθόνη να παριστάνει ένα καθρέπτη. Αυτό βοηθάει πολύ στην οπτική ανάδραση του χρήστη. Τέλος εκτυπώνεται στην κονσόλα η λίστα με τις σειριακές συσκευές ώστε να επιλεχθεί το Arduino. Διαβάζεται η πρώτη θύρα στον Η/Υ και τίθεται ως η προκαθορισμένη θύρα σειριακής επικοινωνίας με ρυθμό μετάδοσης 9600 bps.

Η ρουτίνα συνεχούς επανάληψης `draw()` είναι θεμελιώδης για την υλοποίηση της εφαρμογής. Ξεκινά με την συνάρτηση `kinect.update()` για ανανέωση των δεδομένων του αισθητήρα. Η συνάρτηση `kinect.enableDepth()` ζητά από την βιβλιοθήκη την πιο πρόσφατη εικόνα βάθους. Η εικόνα αυτή αποθηκεύεται στη μεταβλητή `PImage`, μία κλάση του Processing για την αποθήκευση δεδομένων εικόνας. Η κλάση αύτη προσφέρει πολλές χρήσιμες συναρτήσεις για την επεξεργασία τέτοιων εικόνων και το μεγαλύτερο πλεονέκτημα της είναι ότι μπορούμε αυτόματα να χρησιμοποιήσουμε τα δεδομένα που προέρχονται από το Kinect, με άλλες βιβλιοθήκες που δεν γνωρίζουν το Kinect αλλά γνωρίζουν πώς να επεξεργάζονται εικόνες. Η εικόνα αυτή σχεδιάζεται στην οθόνη από την συνάρτηση `image` του Processing με παραμέτρους `0,0` δηλαδή από την πάνω αριστερά γωνία του παράθυρου του σκίτσου.

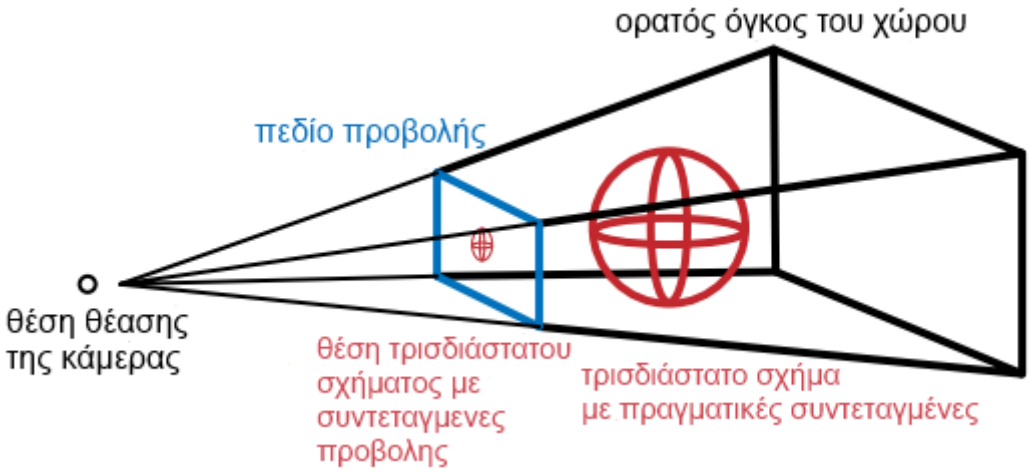
Μετά ζητάμε από την βιβλιοθήκη μία λίστα με όλους τους χρήστες που έχει ανιχνεύσει. Το OpenNI αντιπροσωπεύει τους χρήστες με ακέραιους αριθμούς. Σε κάθε χρήστη εκχωρείται ένας μοναδικός ακέραιος αριθμός που είναι η

ταυτότητα χρήστη (user ID). Δηλώνεται μία μεταβλητή `userList` ειδικού τύπου, ακέραιου διανύσματος (`IntVector`). Σε αυτή τοποθετείται η λίστα με τους χρήστες με την βοήθεια της συνάρτησης `kinect.getUsers()`. Αν η λίστα δεν είναι κενή, παίρνουμε το πρώτο στοιχείο του ακέραιου διανύσματος με την συνάρτηση `userList.get(0)` και του εκχωρούμε την τιμή που αντιπροσωπεύει τον χρήστη της συνάρτησης βαθμονόμησης `onNewUser()`. Αυτός είναι πλέον ο ενεργός χρήστης. Αφού ελεγχθεί αν η βαθμονόμηση πήγε καλά και αν εκτελείται ιχνηλασία σκελετού καλείται η ρουτίνα `drawSkeleton()` για να αποτυπώσει τον σκελετό στην οθόνη.

Έπειτα, ακολουθεί ένα μεγάλο κομμάτι κώδικα αποτελούμενο από τέσσερα βασικά κομμάτια:

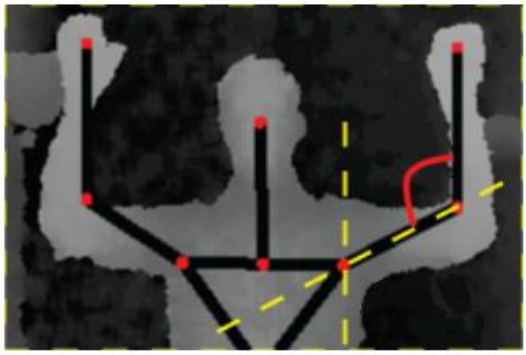
Το πρώτο κομμάτι λαμβάνει την θέση των αρθρώσεων στο χώρο. Για να γίνει αυτό δημιουργείται ένα διάνυσμα για να αποθηκευτεί η θέση της άρθρωσης. Το Processing παρέχει μία κλάση διανυσμάτων που ονομάζεται `PVector` όπου αποθηκεύονται όλες τις συντεταγμένες ενός σημείου σε μία μόνο μεταβλητή. Με την συνάρτηση `kinect.getJointPositionSkeleton()` και παραμέτρους το αναγνωριστικό χρήστη (`userId`) και μία σταθερά που υποδεικνύει την επιθυμητή άρθρωση, λαμβάνεται η θέση της άρθρωσης και αποθηκεύεται στο προαναφερθέν διάνυσμα. Αυτό γίνεται για όλες τις αρθρώσεις του σώματος.

Το δεύτερο κομμάτι αφορά την μετατροπή των πραγματικών συντεταγμένων σε συντεταγμένες «προβολής» (projective). Η συνάρτηση `convertRealWorldToProjective()` μεταφράζει την κίνηση του άξονα Z σε αλλαγές στους άξονες X και Y. Οι αρθρώσεις που είναι μακρύτερα στον άξονα Z κινούνται πιο αργά στους άξονες X και Y. Αυτό δεν είναι επιθυμητό στην συγκεκριμένη εφαρμογή οπότε απλά παραμερίζουμε εντελώς τις πληροφορίες από τον άξονα Z. Αυτό γίνεται δημιουργώντας ένα νέο διάνυσμα χρησιμοποιώντας μόνο τις συντεταγμένες X και Y του αρχικού τρισδιάστατου διανύσματος. Αυτό γίνεται για όλα τα παραπάνω διανύσματα αρθρώσεων.



Εικόνα 5.1: Η σχέση μεταξύ των συντεταγμένων προβολής και των πραγματικών συντεταγμένων. Οι πραγματικές συντεταγμένες τοποθετούν το αντικείμενο στον τρισδιάστατο χώρο. Οι συντεταγμένες προβολής περιγράφουν που θα το βλέπαμε στο πεδίο προβολής.

Το τρίτο κομμάτι αφορά τους άξονες στους οποίους θα υπολογιστούν οι γωνίες των μελών του σώματος. Το πρώτο βήμα για αυτό είναι να βρεθούν οι άξονες που θα ορίσουν τις γωνίες αυτές. Ειδικότερα η γωνία των αγκώνων θα υπολογιστεί σε σχέση με τον προσανατολισμό του αντίστοιχου χεριού (βλέπε εικόνα 5.2) και η γωνία των γονάτων σε σχέση με τον προσανατολισμό του αντίστοιχου ποδιού.



Εικόνα 5.2: Οι άξονες αναφοράς για την γωνία του αριστερού αγκώνα.

Δημιουργούνται τα διανύσματα που αντιστοιχούν σε αυτούς τους άξονες προσανατολισμού και υπολογίζονται με αφαίρεση διανυσμάτων. Για τους αγκώνες αφαιρείται το διάνυσμα του ώμου από το διάνυσμα του αγκώνα, ώστε

ο άξονας να αντιστοιχεί στον προσανατολισμό του χεριού. Για τα γόνατα αφαιρείται το διάνυσμα του ισχίου από το διάνυσμα του γόνατου ώστε ο άξονας να αντιστοιχεί στον προσανατολισμό του ποδιού.

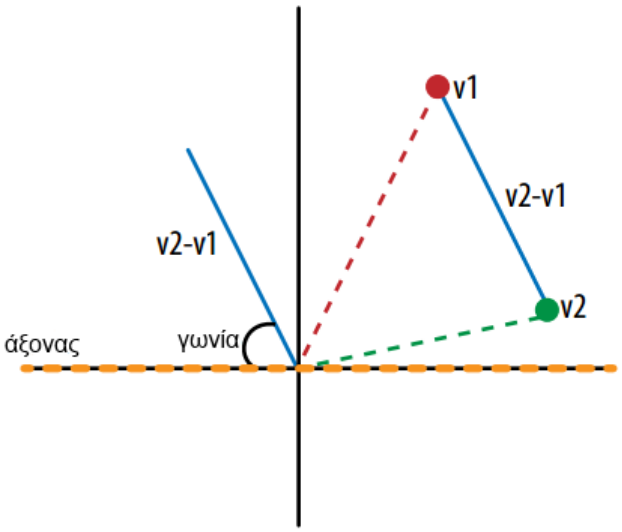
Το τέταρτο κομμάτι υπολογίζει τις γωνίες των μελών του σώματος με βάση τους άξονες προσανατολισμού που βρέθηκαν πριν. Καλείται η ρουτίνα `angleOf()`, η οποία λαμβάνει τρείς παραμέτρους. Τα δύο διανύσματα που αντιπροσωπεύουν τις άκρες του μέλους και ένα τρίτο διάνυσμα που είναι ο άξονας προσανατολισμού. Οι γωνίες αυτές πρέπει να υπολογιστούν λαμβάνοντας υπ' όψιν την τοποθέτηση των σερβοκινητήρων και την αρχική τους θέση.

Ακολουθεί η προβολή στην οθόνη των γωνιών που υπολογίσαμε παραπάνω υλοποιώντας την συνάρτηση `text` του Processing.

Το τελικό μέρος της `draw()` είναι η αποστολή των γωνιών μέσω σειριακής θύρας στο Arduino. Όπως αναφέρθηκε θέλουμε να στείλουμε τις γωνίες με εναλλασσόμενο τρόπο δηλαδή πρώτα τα γόνατα, μετά τον δεξί αγκώνα, μετά τον αριστερό και ξανά. Ο απλούστερος τρόπος να γίνει αυτό είναι να στείλουμε όλες τις γωνίες διαδοχικά. Αυτό το πράττουμε κατασκευάζοντας έναν πίνακα από bytes που αντιστοιχούν στις γωνίες και αποστέλλοντας τον στη σειριακή θύρα μονομιάς. Κάτι που πρέπει να σημειωθεί είναι ότι η συνάρτηση `byte` που καλείται πριν την αποστολή, εκτός από την μετατροπή στον κατάλληλο τύπο για την αποφυγή σφαλμάτων, στρογγυλοποιεί τα δεδομένα στον πλησιέστερο ακέραιο αριθμό. Αυτό είναι απαραίτητο για να «καταλάβει» το Arduino τα δεδομένα αυτά. Επίσης επειδή υπάρχει ένας μόνο κινητήρας για τα πόδια και εφόσον θα σηκώνουμε ένα πόδι τη φορά, οι γωνίες των γονάτων αθροίζονται και διαιρούνται με το δύο για να βγει ο μέσος όρος μίας γωνίας γονάτων.

Η ρουτίνα `angleOf()` αφαιρεί δύο διανύσματα αρθρώσεων δίνοντας ένα νέο που αναπαριστά το μέλος του σώματος που συνδέει αυτές τις αρθρώσεις. Χρησιμοποιεί την συνάρτηση `PVector.angleBetween()` για να υπολογίσει την γωνία μεταξύ του μέλους του σώματος και του άξονα αναφοράς (βλέπε

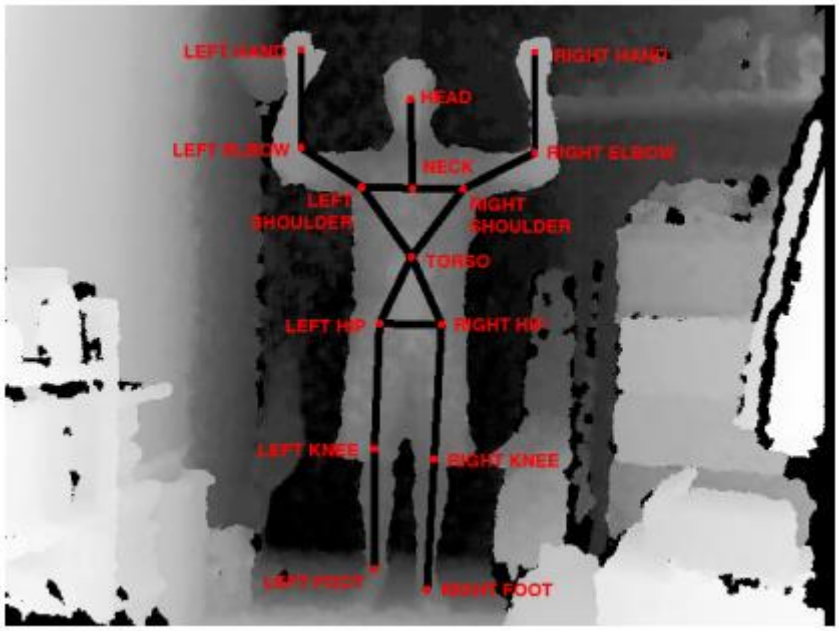
εικόνα 5.3). Το αποτέλεσμα είναι σε ακτίνια, όπου μετατρέπεται σε μοίρες με τη συνάρτηση `degrees()`.



Εικόνα 5.3: Η αφαίρεση δύο διανυσμάτων αρθρώσεων σε ένα διάνυσμα που αντιπροσωπεύει το μέλος του σώματος με δεδομένο προσανατολισμό.

Η ρουτίνα `drawSkeleton()` σχεδιάζει τον σκελετό στην οθόνη (βλέπε εικόνα 5.4). Χρησιμοποιεί την συνάρτηση `drawLimb()` για να ενώσει τις αρθρώσεις με γραμμές προσδιορίζοντας τον σκελετό. Δέχεται τρείς παραμέτρους: το αναγνωριστικό χρήστη (`userId`) και δύο σταθερές που αναγνωρίζουν αρθρώσεις στον σκελετό. Οι αρθρώσεις αυτές καθαυτές σχεδιάζονται σαν τελείες από την συνάρτηση `drawJoint()` που καλείται μετά.

Η ρουτίνα `drawJoint()` σχεδιάζει τις αρθρώσεις στην οθόνη. Δέχεται δύο παραμέτρους: το αναγνωριστικό χρήστη (`userId`) και το αναγνωριστικό της άρθρωσης (`jointID`). Δημιουργεί ένα διάνυσμα `PVector` το οποίο θα κατοικηθεί από πληροφορίες για τη ζητούμενη άρθρωση. Μετά επικυρώνει ότι η «σιγουριά» που σχετίζεται με την άρθρωση είναι αρκετά υψηλή για να συνεχίσει, μετατρέποντας την θέση της άρθρωσης σε συντεταγμένες προβολής για να ταυτίζονται με την εικόνα βάθους και τελικά σχεδιάζει μία έλλειψη βασισμένη σε αυτές τις συντεταγμένες.



Εικόνα 5.4: Ο πλήρης σκελετός από το OpenNI με όλα τα μέλη του σώματος σχεδιασμένα και όλες τις αρθρώσεις επισημασμένες.

Ο έλεγχος της «σιγουριάς» γίνεται με την χρήση της πληροφορίας θέσης της συνάρτησης `getJointPositionSkeleton()`. Αν το επίπεδο της «σιγουριάς» είναι χαμηλότερο από το κατώφλι που έχουμε θέσει, δηλαδή το OpenNI απλά μαντεύει τη θέση, καλείται η συνάρτηση `return` και τίποτα παρακάτω της δεν εκτελείται.

Οι ρουτίνες που ακολουθούν αποτελούν την διαδικασία βαθμονόμησης που αναφέρεται το κεφάλαιο 3.8.3.1. Επιγραμματικά η ρουτίνα `onNewUser()` ενεργοποιείται όταν εντοπιστεί χρήστης. Καλεί την ρουτίνα `startPoseDetection()` για να ξεκινήσει η αρχική στάση. Η ρουτίνα `onEndCalibration()` τυπώνει στην οθόνη την λήξη της βαθμονόμησης και αν ήταν επιτυχής ξεκινά την ιχνηλασία του σκελετού. Αν δεν ήταν επιτυχής καλεί την ρουτίνα `startPoseDetection()` για να εκκινήσει η διαδικασία αρχικής στάσης ξανά. Τέλος η ρουτίνα `onStartPose()` ενεργοποιείται μετά την αρχική στάση. Σταματάει να αναζητά την αρχική στάση (αφού έχει βρει ήδη) και αιτείται την έναρξη της βαθμονόμησης του σκελετού.

5.2 Κώδικας στο Arduino

```
#include <Servo.h>;

//declare servos
Servo Knees;
Servo rightElbow;
Servo leftElbow;

//setup the array of servo positions
int nextServo = 0;
int servoAngles[3] = {0,0};

void setup() {
  //attach servos to their pins
  Knees.attach(9);
  rightElbow.attach(10);
  leftElbow.attach(11);

  Serial.begin(9600);
}

void loop() {
  if (Serial.available()) {
    int servoAngle = Serial.read();

    servoAngles[nextServo] = servoAngle;
    nextServo++;
    if (nextServo > 2) {
      nextServo = 0;
    }

    Knees.write(servoAngles[0]);
    rightElbow.write(servoAngles[1]);
    leftElbow.write(servoAngles[2]);
  }
}
```

5.3 Κώδικας στο Processing

```
import SimpleOpenNI.*;
import processing.serial.*;
SimpleOpenNI kinect;
Serial port;

void setup() {
  size(640,480);

  kinect = new SimpleOpenNI(this);
  kinect.enableDepth();

  //turn on user tracking
  kinect.enableUser(SimpleOpenNI.SKEL_PROFILE_ALL);

  kinect.setMirror(true);

  println(Serial.list());
  String portName = Serial.list()[0];
  port = new Serial(this, portName, 9600);
}

void draw() {
  kinect.update();
  PImage depth = kinect.depthImage();
  image(depth, 0, 0);

  //make a vector of ints to store the list of users
  IntVector userList = new IntVector();

  //write the list of detected users into our vector
  kinect.getUsers(userList);

  //if we found any users
  if (userList.size() > 0) {
    //get the first user
    int userId = userList.get(0);

    //if we're successfully calibrated
    if (kinect.isTrackingSkeleton(userId)) {
      drawSkeleton(userId);

      //get the positions of the joints
      PVector rightHand = new PVector();
      kinect.getJointPositionSkeleton(userId,
                                      SimpleOpenNI.SKEL_RIGHT_HAND,
                                      rightHand);
      PVector rightElbow = new PVector();
      kinect.getJointPositionSkeleton(userId,
                                      SimpleOpenNI.SKEL_RIGHT_ELBOW,
                                      rightElbow);
      PVector rightShoulder = new PVector();
      kinect.getJointPositionSkeleton(userId,
                                      SimpleOpenNI.SKEL_RIGHT_SHOULDER,
                                      rightShoulder);
      PVector leftHand = new PVector();
      kinect.getJointPositionSkeleton(userId,
                                      SimpleOpenNI.SKEL_LEFT_HAND,
```

```
                                leftHand);
PVector leftElbow = new PVector();
kinect.getJointPositionSkeleton(userId,
                                SimpleOpenNI.SKEL_LEFT_ELBOW,
                                leftElbow);
PVector leftShoulder = new PVector();
kinect.getJointPositionSkeleton(userId,
                                SimpleOpenNI.SKEL_LEFT_SHOULDER,
                                leftShoulder);
PVector rightHip = new PVector();
kinect.getJointPositionSkeleton(userId,
                                SimpleOpenNI.SKEL_RIGHT_HIP,
                                rightHip);
PVector rightKnee = new PVector();
kinect.getJointPositionSkeleton(userId,
                                SimpleOpenNI.SKEL_RIGHT_KNEE,
                                rightKnee);
PVector rightFoot = new PVector();
kinect.getJointPositionSkeleton(userId,
                                SimpleOpenNI.SKEL_RIGHT_FOOT,
                                rightFoot);
PVector leftHip = new PVector();
kinect.getJointPositionSkeleton(userId,
                                SimpleOpenNI.SKEL_LEFT_HIP,
                                leftHip);
PVector leftKnee = new PVector();
kinect.getJointPositionSkeleton(userId,
                                SimpleOpenNI.SKEL_LEFT_KNEE,
                                leftKnee);
PVector leftFoot = new PVector();
kinect.getJointPositionSkeleton(userId,
                                SimpleOpenNI.SKEL_LEFT_FOOT,
                                leftFoot);

//reduce joint vectors to two dimensions
PVector rightHand2D = new PVector(rightHand.x, rightHand.y);
PVector rightElbow2D = new PVector(rightElbow.x, rightElbow.y);
PVector rightShoulder2D =
 new PVector(rightShoulder.x, rightShoulder.y);
PVector leftHand2D = new PVector(leftHand.x, leftHand.y);
PVector leftElbow2D = new PVector(leftElbow.x, leftElbow.y);
PVector leftShoulder2D =
 new PVector(leftShoulder.x, leftShoulder.y);
PVector rightHip2D = new PVector(rightHip.x, rightHip.y);
PVector rightKnee2D = new PVector(rightKnee.x, rightKnee.y);
PVector rightFoot2D = new PVector(rightFoot.x, rightFoot.y);
PVector leftHip2D = new PVector(leftHip.x, leftHip.y);
PVector leftKnee2D = new PVector(leftKnee.x, leftKnee.y);
PVector leftFoot2D = new PVector(leftFoot.x, leftFoot.y);

//calculate the axis against which we want to measure our angles
PVector rightArmOrientation =
 PVector.sub(rightShoulder2D, rightElbow2D);
PVector rightLegOrientation =
 PVector.sub(rightHip2D, rightKnee2D);
PVector leftArmOrientation =
 PVector.sub(leftElbow2D, leftShoulder2D);
PVector leftLegOrientation =
 PVector.sub(leftKnee2D, leftHip2D);

//calculate the angles between our joints
```

```
      float rightElbowAngle = angleOf(rightHand2D,
                                      rightElbow2D,
                                      rightArmOrientation);
      float leftElbowAngle = angleOf(leftHand2D,
                                     leftElbow2D,
                                     leftArmOrientation);
      float rightKneeAngle = angleOf(rightFoot2D,
                                     rightKnee2D,
                                     rightLegOrientation);
      float leftKneeAngle = angleOf(leftFoot2D,
                                    leftKnee2D,
                                    leftLegOrientation);

      //show the angles on the screen
      fill(255,0,0);
      scale(2);
      text("right elbow: " + int(rightElbowAngle) + "\n" +
           "left elbow: " + int(leftElbowAngle) + "\n" +
           "right knee: " + int(rightKneeAngle) + "\n" +
           "left knee; " + int(leftKneeAngle), 20, 20);


      //send angles through serial port
      byte out[] = new byte[3];
      out[0] = byte((rightKneeAngle+leftKneeAngle/2));
      out[1] = byte(rightElbowAngle);
      out[2] = byte(leftElbowAngle);
      port.write(out);
    }
  }
}

float angleOf(PVector one, PVector two, PVector axis) {
  PVector limb = PVector.sub(two, one);
  return degrees(PVector.angleBetween(limb, axis));
}

void drawSkeleton(int userId) {
  stroke(0);
  strokeWeight(5);

  kinect.drawLimb(userId, SimpleOpenNI.SKEL_HEAD,
                          SimpleOpenNI.SKEL_NECK);
  kinect.drawLimb(userId, SimpleOpenNI.SKEL_NECK,
                          SimpleOpenNI.SKEL_LEFT_SHOULDER);
  kinect.drawLimb(userId, SimpleOpenNI.SKEL_LEFT_SHOULDER,
                          SimpleOpenNI.SKEL_LEFT_ELBOW);
  kinect.drawLimb(userId, SimpleOpenNI.SKEL_LEFT_ELBOW,
                          SimpleOpenNI.SKEL_LEFT_HAND);
  kinect.drawLimb(userId, SimpleOpenNI.SKEL_NECK,
                          SimpleOpenNI.SKEL_RIGHT_SHOULDER);
  kinect.drawLimb(userId, SimpleOpenNI.SKEL_RIGHT_SHOULDER,
                          SimpleOpenNI.SKEL_RIGHT_ELBOW);
  kinect.drawLimb(userId, SimpleOpenNI.SKEL_RIGHT_ELBOW,
                          SimpleOpenNI.SKEL_RIGHT_HAND);
  kinect.drawLimb(userId, SimpleOpenNI.SKEL_LEFT_SHOULDER,
                          SimpleOpenNI.SKEL_TORSO);
  kinect.drawLimb(userId, SimpleOpenNI.SKEL_RIGHT_SHOULDER,
                          SimpleOpenNI.SKEL_TORSO);
  kinect.drawLimb(userId, SimpleOpenNI.SKEL_TORSO,
                          SimpleOpenNI.SKEL_LEFT_HIP);
```

```
  kinect.drawLimb(userId, SimpleOpenNI.SKEL_LEFT_HIP,
                          SimpleOpenNI.SKEL_LEFT_KNEE);
  kinect.drawLimb(userId, SimpleOpenNI.SKEL_LEFT_KNEE,
                          SimpleOpenNI.SKEL_LEFT_FOOT);
  kinect.drawLimb(userId, SimpleOpenNI.SKEL_TORSO,
                          SimpleOpenNI.SKEL_RIGHT_HIP);
  kinect.drawLimb(userId, SimpleOpenNI.SKEL_RIGHT_HIP,
                          SimpleOpenNI.SKEL_RIGHT_KNEE);
  kinect.drawLimb(userId, SimpleOpenNI.SKEL_RIGHT_KNEE,
                          SimpleOpenNI.SKEL_RIGHT_FOOT);
  noStroke();
  fill(255,0,0);

  drawJoint(userId, SimpleOpenNI.SKEL_HEAD);
  drawJoint(userId, SimpleOpenNI.SKEL_NECK);
  drawJoint(userId, SimpleOpenNI.SKEL_LEFT_SHOULDER);
  drawJoint(userId, SimpleOpenNI.SKEL_LEFT_ELBOW);
  drawJoint(userId, SimpleOpenNI.SKEL_NECK);
  drawJoint(userId, SimpleOpenNI.SKEL_RIGHT_SHOULDER);
  drawJoint(userId, SimpleOpenNI.SKEL_RIGHT_ELBOW);
  drawJoint(userId, SimpleOpenNI.SKEL_TORSO);
  drawJoint(userId, SimpleOpenNI.SKEL_LEFT_HIP);
  drawJoint(userId, SimpleOpenNI.SKEL_LEFT_KNEE);
  drawJoint(userId, SimpleOpenNI.SKEL_RIGHT_HIP);
  drawJoint(userId, SimpleOpenNI.SKEL_LEFT_FOOT);
  drawJoint(userId, SimpleOpenNI.SKEL_RIGHT_KNEE);
  drawJoint(userId, SimpleOpenNI.SKEL_LEFT_HIP);
  drawJoint(userId, SimpleOpenNI.SKEL_RIGHT_FOOT);
  drawJoint(userId, SimpleOpenNI.SKEL_RIGHT_HAND);
  drawJoint(userId, SimpleOpenNI.SKEL_LEFT_HAND);
}

void drawJoint(int userId, int jointID) {
  PVector joint = new PVector();

  float confidence =
    kinect.getJointPositionSkeleton(userId, jointID, joint);
  if (confidence < 0.5) {
    return;
  }
  PVector convertedJoint = new PVector();
  kinect.convertRealWorldToProjective(joint, convertedJoint);
  ellipse(convertedJoint.x, convertedJoint.y, 5, 5);
}

//user tracking callbacks
void onNewUser(int userId) {
  if (kinect.isTrackingSkeleton(userId)) return;
  println("Start pose detection");
  kinect.startPoseDetection("Psi", userId);
}

void onEndCalibration(int userId, boolean succesful) {
  if (succesful) {
    println("User calibrated");
    kinect.startTrackingSkeleton(userId);
  }
  else {
    println("Failed to calibrate user");
    kinect.startPoseDetection("Psi", userId);
  }
```

```
}

void onStartPose(String pose, int userId) {
  println("Started pose for user");
  kinect.stopPoseDetection(userId);
  kinect.requestCalibrationSkeleton(userId, true);
}
```

Δυσκολίες - Συμπεράσματα

Το Kinect αποτέλεσε το πρώτο βήμα για μία τεχνολογική επανάσταση. Έκανε την αρχή για μία μεγάλη γκάμα συσκευών και εφαρμογών, που χρησιμοποιούν αυτή την τεχνολογία, για να κάνουν τους υπολογιστές να καταλάβουν καλύτερα τον κόσμο γύρω τους. Λόγω των τεράστιων δυνατοτήτων του, επιλέχθηκε για την εκπόνηση της παρούσας εργασίας.

Από την αρχή υπήρχαν διάφορα προβλήματα προς επίλυση κυρίως οικονομικά. Δεν υπήρχε η δυνατότητα αγοράς μιας μονάδας Kinect λόγω αυξημένου κόστους. Τη λύση έδωσε κοντινός φίλος μου, ο οποίος μόλις είχε αγοράσει το Kinect για το Xbox του και παρόλα αυτά δέχτηκε να μου το δανείσει για μεγάλο χρονικό διάστημα. Αποφάσισα η ανάπτυξη του κώδικα να γίνει σε φορητό υπολογιστή (και όχι στον σταθερό μου) προκείμενου να είμαι σίγουρος ότι όλα τα τμήματα του λογισμικού είναι σωστά εγκατεστημένα και ρυθμισμένα, ώστε αν προέκυπτε κάποιο πρόβλημα να ήταν ευκολότερο να επιλυθεί αποκλείοντας συγκεκριμένες μεταβλητές. Επίσης πολύ σημαντικό είναι ότι η βιβλιοθήκη `Serial` του Processing δεν δουλεύει σε 64bit λειτουργικό. Έτσι και πάλι λόγω οικονομικής στενότητας επιστρατεύτηκε το netbook μου Asus eeePC 1000H με ηλικία τα πέντε έτη. Χρειάστηκε μία σειρά εργασιών για να μπορέσει να «σηκώσει» το λογισμικό και να το τρέχει όσο πιο ομαλά γίνεται. Ο υπερχρονισμός ήταν μονόδρομος και για να μην «ψηθεί» το netbook το άνοιξα ώστε να καθαριστεί και του τοποθέτησα νέα θερμοαγώγιμη πάστα υψηλής ποιότητας, αφαιρώντας την παλιά. Εγκατέστησα τροποποιημένη έκδοση των Windows 7 32bit για netbook και μαζί με τον υπερχρονισμό κατά 15% στον επεξεργαστή ανταπεξήλθε στο έργο που ανέλαβε. Αντιθέτως η αγορά του Arduino Leonardo ήταν απλή λόγω χαμηλού κόστους. Εύκολη αποδείχθηκε, επίσης, και η αναζήτηση του ειδικού μετασχηματιστή που διαχωρίζει τη σύνδεση του Kinect σε ξεχωριστή σύνδεση επικοινωνίας και παροχής ρεύματος.

Στο κατασκευαστικό κομμάτι οι δυσκολίες αναφέρονται στο κεφάλαιο 4. Στο κομμάτι της ανάπτυξης του κώδικα ήταν οι τυπικές σε κάθε προγραμματιστή που αναπτύσσει κώδικα. Αναζήτηση των κατάλληλων κλάσεων από τα αρχεία τεκμηρίωσης της βιβλιοθήκης simple-openni, με τις κατάλληλες μεταβλητές, τρέξιμο πολλών παραδειγμάτων, ανάπτυξη βήμα βήμα, διαμόρφωση του προγράμματος, αναζήτηση προτάσεων και συμβουλών σε τεχνικά βιβλία και στο διαδίκτυο, πολύ καφές και ακόμα περισσότερη σκέψη. Αρχικά η επικοινωνία μεταξύ Kinect και Arduino μου διέφευγε. Έτσι ασχολήθηκα μόνο με το Kinect και την ιχνηλασία σκελετού στο περιβάλλον του Processing. Αφού τέλειωσε αυτό το κομμάτι ασχολήθηκα με το Arduino και τον έλεγχο των σερβοκινητήρων. Βασίστηκα σε έτοιμα παραδείγματα και αποδείχτηκε εύκολη δουλειά. Πιο δύσκολη ήταν η επίλυση του προβλήματος της επικοινωνίας μεταξύ των συσκευών. Ύστερα από αρκετή σκέψη και δοκιμές και με την βοήθεια ενός βιβλίου ξεπεράστηκε και αυτό το εμπόδιο. Ο κώδικας ήταν ολοκληρωμένος και λειτουργικός· το γράψιμο ξεκινούσε.

Η συγγραφή της εργασίας ήταν αναπάντεχα το πιο χρονοβόρο κομμάτι. Σε συνδυασμό με υποχρεώσεις και περιστασιακή εργασία αποτέλεσε αναμφίβολα το πιο απαιτητικό μέρος αυτής. Η εύρεση υλικού, η μετάφραση - απόδοση, οι προσθήκες και οι αλλαγές στη δομή ήταν τα σημεία κλειδιά στη πορεία της.

Η προσέγγιση Processing IDE – OpenNI/NITE ήταν ιδανική λόγω των δυνατοτήτων τους αντίστοιχα. Το Processing με την ευκολία χρήσης του και τις βιβλιοθήκες που είναι διαθέσιμες για αυτό και τα OpenNI/NITE για τις συναρτήσεις που περιλαμβάνουν και την απλότητα στην υλοποίηση τους. Λόγω αυτών ο κώδικας θα πρέπει να εκτελείται σωστά σε οποιοδήποτε λειτουργικό σύστημα, χωρίς όμως να τον έχω προσωπικά δοκιμάσει.

Η εργασία αυτή αποτελεί εξέλιξη παραπλήσιας εργασίας φυσικής αλληλεπίδρασης συμφοιτητή μου. Η υλοποίηση του έγινε με χρήση απλής webcam, των βιβλιοθηκών OpenCV και χειρονομιών του χεριού.

Βιβλιογραφία

1. Greg Borestein, «Making Things See. 3D Vision with Kinect, Processing, Arduino and MakerBot», Εκδόσεις O'Reilly, 2012.
2. Enrique Ramos Melgar, Ciriaco Castro Diez and Przemek Jarowski. «Arduino and Kinect Projects», Εκδόσεις «Springer Science and Business Media», 2012.
3. Jan Smisek, Michael Jancosek and Tomas Pajdla. «3D with Kinect», Czech Technical University of Prague, 2011.
4. Technical Report. «Kinect Depth Sensor Evaluation for Computer Vision Applications», University of Aarhus, 2012.
5. Kourosh Khoshelham and Sander Oude Elberink. «Accuracy and Resolution of Kinect Depth Data for Indoor Mapping Applications», University of Twente, 2012.
6. http://arduino.cc/en/Main/ArduinoBoardLeonardo
7. http://arduino.cc/en/Guide/ArduinoLeonardoMicro
8. wikipedia.org/wiki/arduino
9. wikipedia.org/wiki/Pull-up-resistor
10. wikipedia.org/wiki/kinect
11. openkinect.org
12. msdn.microsoft.com/en-us/library/jj131033.aspx
13. http://www.ifixit.com/Teardown/Microsoft+Kinect+Teardown/4066/3
14. www.computableminds.com/post/Kinect/multiarray/microphone/how-works/xbox-360
15. http://www.newscientist.com/article/dn19762-inside-the-race-to-hack-the-kinect.html
16. www.primesense.com/en/technology/114-the-ps1080
17. www.primesense.com/en/press-room/resources/category/3-a4-ds
18. wikipedia.org/wiki/Structured-light-3D-scanner
19. www.kinecthacks.com/top-10-best-kinect-hacks
20. http://en.wikipedia.org/wiki/Time-of-flight_camera

21. wikipedia.org/wiki/Processing_(programming_language)
22. processing.org
23. www.czechmarionettes.com/Marionettes/Puppets/Scientist-man-puppet.html