Πανεπιστήμιο Θεσσαλίας, 2012-2013

Τμήμα Μηχανικών Η/Υ, Τηλεπικοινωνιών και Δικτύων

Μεταπτυχιακή εργασία

Θέμα:

« Ανάπτυξη web-based διεπαφής για έλεγχο κατευθυντικών κεραιών σε ασύρματα δίκτυα 802.11 »

Παπαποστόλου Αλέξανδρος

UNIVERSITY OF THESSALY

Επιβλέπων καθηγητής:

Κοράκης Αθανάσιος (Λέκτορας)

Συνεπιβλέποντες καθηγητές:

Λέανδρος Τασιούλας (Καθηγητής)

Αργυρίου Αντώνιος (Λέκτορας)

Βόλος, Οκτώβριος 2013

Ευχαριστίες

Με την εκπόνηση της παρούσας μεταπτυχιακής εργασίας, φέρνω εις πέρας τις μεταπτυχιακές μου σπουδές στο Τμήμα Μηχανικών Η/Υ Τηλεπικοινωνιών και Δικτύων του Πανεπιστημίου Θεσσαλίας.

Θα ήθελα αρχικά να ευχαριστήσω θερμά τον Λέκτορα του Τμήματος Μηχανικών Η/Υ Τηλεπικοινωνιών και Δικτύων κ. Κοράκη Αθανάσιο, και τον καθηγητή του Τμήματος Μηχανικών Η/Υ Τηλεπικοινωνιών και Δικτύων κ. Λέανδρο Τασιούλα, για τις χρήσιμες συμβουλές και υποδείξεις του καθώς και για την υποστήριξη που μου προσφέρανε κατά τη διάρκεια της φοίτησής μου, αλλά και κατά την εκπόνηση της διπλωματικής μου εργασίας. Επιπροσθέτως, να τους ευχαριστήσω που μου δώσαν τη δυνατότητα να ασχοληθώ στην μεταπτυχιακή μου εργασία με καινοτόμες τεχνολογίες και να συνεργαστώ με την ομάδα του NITlab.

Από καρδιάς να ευχαριστήσω όλη την ομάδα του NITlab, η οποία ήταν «σχολείο» από πλευράς γνώσεων και συνεργασίας. Ιδιαιτέρως, να ευχαριστήσω τον υποψήφιο διδάκτορα του Τμήματος Μηχανικών Η/Υ Τηλεπικοινωνιών και Δικτύων Καζδαρίδη Ιωάννη για την καθοδήγηση και την πολύτιμη βοήθειά του καθ’ όλη τη διάρκεια της μεταπτυχιακής εργασίας.

Τέλος, ευχαριστώ θερμά την οικογένειά μου για την αμέριστη συμπαράσταση που μου παρείχε όλα αυτά τα χρόνια για την ολοκλήρωση των μεταπτυχιακών και προπτυχιακών σπουδών μου.

Αφιερωμένο στην αδερφή μου,
Αθανασία Παπαποστόλου

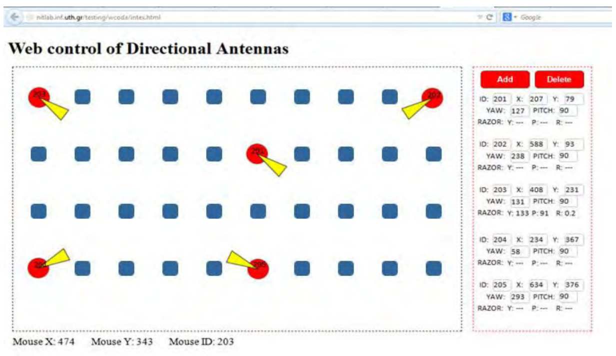
Contents

ABSTRACT

Towards the direction of extending the wireless experimentation testbed of NITOS FI facility, a directional antenna framework was developed to provide advanced capabilities to NITOS users. Through this framework, the research community is able to run their experiments and evaluate their protocols that need to run with a set of nodes equipped with directional antennas. The main idea of the developed framework is to control the direction of the antennas used for the experiment resulting in different gain of the received/transmitted signal. For the needs of this framework, a web-based control mechanism was developed providing the pan and tilt ability to a mounted directional antenna. It is controlled either by an interactive user interface or programmatically by NITOS' OMF scheduler. The control mechanism acts as a web server that receives position commands through http requests, adjusts the servos accordingly and sends back xml data with the servo status and the actual positions acknowledged from Razor sensors. The core module of the implementation is an Arduino board, an open-source hardware platform functioned by an AVR microcontroller, equipped with an Ethernet shield providing network communication capabilities. It is supplied with several digital/output pins where the servos through PWM and the 9DOF Razor IMU via serial are connected. Servos are ideal to control physical movement because they move to a fixed position instead of continuously rotating, but they usually lack of precision and need further calibration in order to map the desired angles with the corresponding input pulse width. A combination of an adequate number of measurements from Razor sensors and respective optical observations of the actual angle for every pulse width that the servo supports, were collected and processed in Matlab using the polynomial curve fitting function (polyfit) to define the polynomial relation. The coefficients of the polynomial of third degree were used to calculate the proper input servo value for the desired angle. In order to connect the servos and the 9DOF Razor IMU, a shield was designed that stacks on top of Arduino boards. Furthermore, for the needs of a central control of all the directional antennas that are installed at NITOS testbed, a web-based application was developed giving to experimenters the ability to interact with all the steering mechanisms through a graphical user interface.

1. Introduction

NITOS FI facility offers to research community the ability to perform experiments with numerous wireless nodes of heterogenous technologies allowing them to design, implement new algorithms, enabling new functionalities on wireless networks. In order to perform experiments and evaluate protocols with a set of nodes equipped with directional antennas, there was a need for a network-based framework to control the direction of the antennas that results in different gain of the received/transmitted signal in 802.11 networks. This master thesis presents the aforementioned framework. In the second chapter, we present the NITOS facility and OMF. In the third chapter, we present the components of the mechanism and how are connected. In the fourth, we present the web-based applications and the technologies that were used. Finally, in the fifth chapter we present the future improvements of the framework.



Web control of Directional Antennas

ID: 201 X: 207 Y: 79
YAW: 127 PITCH: 90
RAZOR: Y: --- P: --- R: ---

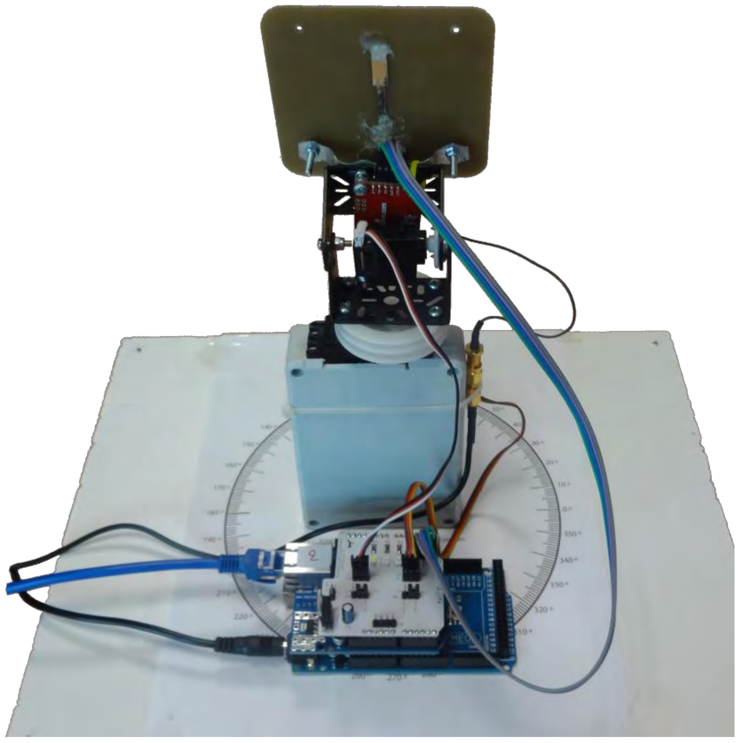
ID: 202 X: 588 Y: 93
YAW: 238 PITCH: 90
RAZOR: Y: --- P: --- R: ---

ID: 203 X: 408 Y: 231
YAW: 131 PITCH: 90
RAZOR: Y: 133 P: 91 R: 0.2

ID: 204 X: 234 Y: 367
YAW: 58 PITCH: 90
RAZOR: Y: --- P: --- R: ---

ID: 205 X: 634 Y: 376
YAW: 293 PITCH: 90
RAZOR: Y: --- P: --- R: ---

Mouse X: 474 Mouse Y: 343 Mouse ID: 203



2. NITOS FI facility

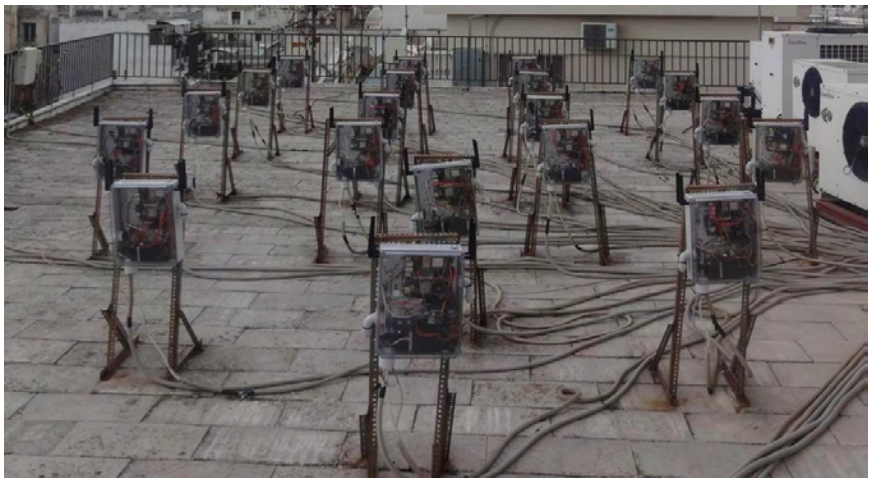


Figure 1: Outdoor testbed

The **NITOS** FI facility attracts numerous researchers from around the globe, thus rendering UTH a key player among the testbeds comprising the FIRE (Future Internet Research Initiative) initiative in Europe. The main experimental components of NITOS are:

A wireless experimentation testbed, which consists of powerful nodes (some of them mobile), that feature multiple wireless interfaces and allow for experimentation with heterogeneous (WiFi, Bluetooth, ZigBee) wireless technologies. **NITOS is about to be extended to a meso-scale testbed, with WiMAX and LTE Base Stations and by also enabling WiMAX/LTE connectivity to the wireless nodes.** The total number of fully operational nodes is 50, and is about to extend to over 100, by the deployment of a new RF-isolated indoor testbed.

A distributed Wireless Sensor Network (WSN) testbed of 40 nodes, able to sense and gather environmental measurements from agricultural installations. The deployed facility consists of multiple clusters, each one comprised of wireless sensor devices and Gateway nodes, creating mesh networks utilizing the ZigBee technology. Measurements that are gathered by the sensor network are aggregated, stored and processed at a centralized cluster of servers, which controls the irrigating system. The distributed wireless sensor infrastructure is located in the agricultural installation of UTH and enables experimentation on agricultural development by sensing environmental conditions as well as controlling multiple parameters on the agricultural process.

A software defined radio (SDR) testbed that consists of 10 Universal Software Radio Peripheral (USRP) devices attached to the NITOS wireless nodes. USRPs allow the researcher to program a number of physical layer features (e.g. modulation), thereby enabling dedicated PHY layer or cross-layer research.

A Software Defined Networking (SDN) testbed that consists of multiple **OpenFlow** technology enabled switches, connected to the NITOS nodes, thus enabling experimentation with switching and routing networking protocols. Experimentation using the OpenFlow technology can be combined with the wireless networking one, hence enabling the construction of more heterogeneous experimental scenarios. Although the physical devices that operate with OpenFlow are three, they are organized in a tree topology, interconnecting the testbed's nodes. With the use of the Open-VSwitch software, one can implement a software OpenFlow switch on the NITOS nodes, thus enabling more complex experiments.

A testbed for conducting video-transmission (wired or wireless) related experimentation, which consists of high definition digital cameras, mounted on the NITOS nodes. This component can be combined with the wired (OpenFlow) and wireless testbeds mentioned above, enabling the study of video transmission over heterogeneous communication technologies.

NITOS testbed is open to the research community 24/7 and it is **remotely accessible** through the NITOS reservation tool. Parallel experimentation of different users is enabled, through the utilization of the NITOS scheduler software. The testbed is based on open-source software that allows the design and implementation of new algorithms, enabling new functionalities on the existing hardware. Though OMF (cOntrol and Management Framework), NITOS supports evaluation of protocols and applications under real world settings. It is also designed to achieve reproducibility of experimentation though the CONCRETE (CONtrol and Classify REpeatable Testbed Experiments) tool developed in UTH [1].

3. Control Mechanism

The components of the mechanism are:

- an Arduino Ethernet or Arduino Mega with Ethernet shield
- a 360 degrees servo motor for yaw
- an 180 degrees servo motor for pitch
- a 9DOF Razor IMU with triple-axis gyro, accelerometer and magnetometer sensor
- a custom Arduino shield that servos, resistors, Razor IMU and relays are attached
- metal mounting brackets
- a 9 dbi directional antenna

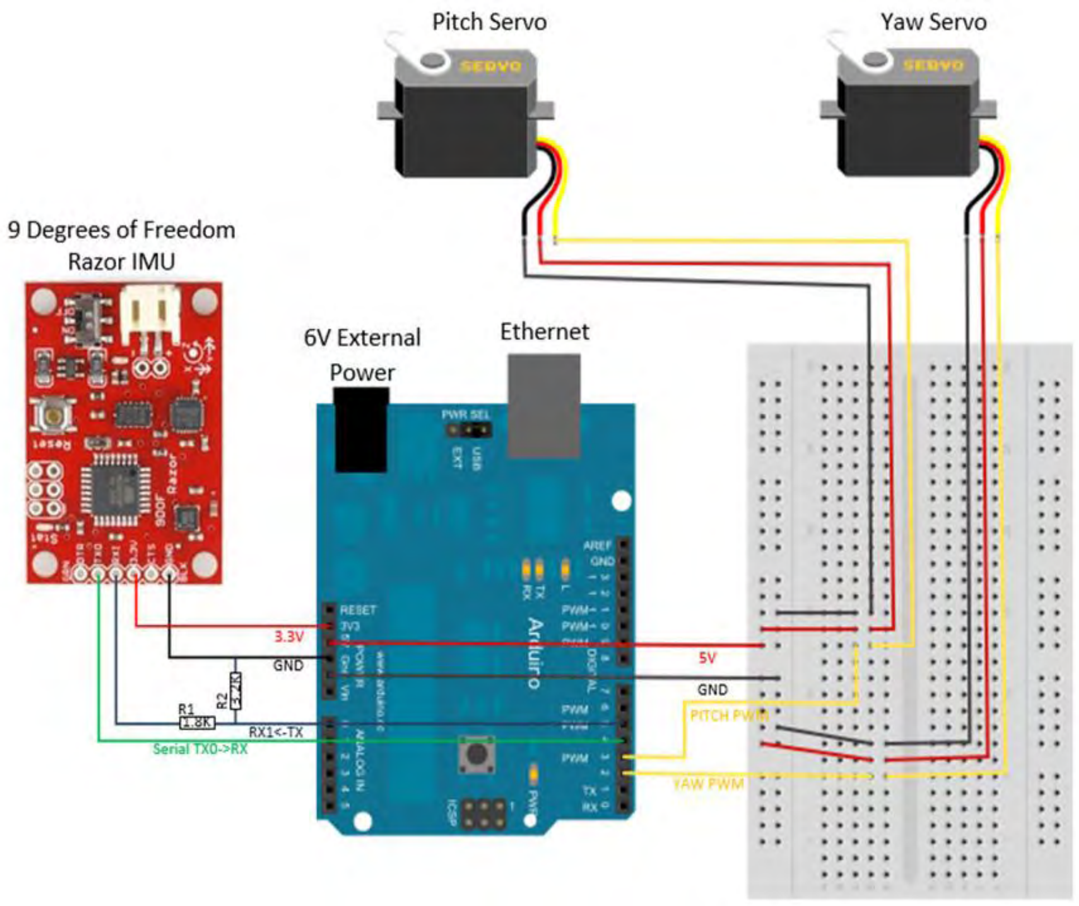


Figure 2: Schematic of the mechanism that clearly shows the connectivity of the components

3.1 Arduino

Arduino is a single-board microcontroller designed to make the process of using electronics in multidisciplinary projects more accessible. The hardware consists of a simple open source hardware board designed around an 8-bit Atmel AVR microcontroller, though a new model has been designed around a 32-bit Atmel ARM. The software consists of a standard programming language compiler and a boot loader that executes on the microcontroller [2].

Arduino boards can be purchased pre-assembled or do-it-yourself kits. Hardware design information is available for those who would like to assemble an Arduino by hand. There are sixteen official Arduinos that have been commercially produced to date.

Figure 3: Arduino Leonardo board

3.1.1 Arduino Hardware

An Arduino board consists of an Atmel 8-bit AVR microcontroller with complementary components to facilitate programming and incorporation into other circuits. An important aspect of the Arduino is the standard way that connectors are exposed, allowing the CPU board to be connected to a variety of interchangeable add-on modules known as shields. Some shields communicate with the Arduino board directly over various pins, but many shields are individually addressable via an I^{2}C serial bus, allowing many shields to be stacked and used in parallel. Official Arduinos have used the megaAVR series of chips, specifically the ATmega8, ATmega168, ATmega328, ATmega1280, and ATmega2560. A handful of other processors have been used by Arduino compatibles. Most boards include a 5 volt linear regulator and a 16 MHz crystal oscillator (or ceramic resonator in some variants), although some designs such as the LilyPad run at 8 MHz and dispense with the onboard voltage regulator due to specific form-factor restrictions. An Arduino's microcontroller is also pre-programmed with a boot loader that simplifies uploading of programs to the on-chip flash memory, compared with other devices that typically need an external programmer.

At a conceptual level, when using the Arduino software stack, all boards are programmed over an RS-232 serial connection, but the way this is implemented varies by hardware version. Serial Arduino boards contain a simple inverter circuit to convert between RS-232-level and TTL-level signals. Current Arduino boards are programmed via USB, implemented using USB-to-serial adapter chips such as the FTDI FT232. Some variants, such as the Arduino Mini and the unofficial Boarduino, use a detachable USB-to-serial adapter board or cable, Bluetooth or other methods. (When used with traditional microcontroller tools instead of the Arduino IDE, standard AVR ISP programming is used.)

The Arduino board exposes most of the microcontroller's I/O pins for use by other circuits. The Diecimila, Duemilanove, and current Uno provide 14 digital I/O pins, six of which can produce pulse-width modulated signals, and six analog inputs. These pins are on the top of the board, via female 0.1 inch headers. Several plug-in application shields are also commercially available.

The Arduino Nano, and Arduino-compatible Bare Bones Board and Boarduino boards may provide male header pins on the underside of the board to be plugged into solderless breadboards.

There are a great many Arduino-compatible and Arduino-derived boards. Some are functionally equivalent to an Arduino and may be used interchangeably. Many are the basic Arduino with the addition of commonplace output drivers, often for use in school-level education to simplify the construction of buggies and small robots. Others are electrically equivalent but change the form factor, sometimes permitting the continued use of Shields, sometimes not. Some variants even use completely different processors, with varying levels of compatibility.

3.1.2 Arduino Software

The Arduino integrated development environment (IDE) is a cross-platform application written in Java, and is derived from the IDE for the Processing programming language and the Wiring projects. It is designed to introduce programming to artists and other newcomers unfamiliar with software development. It includes a code editor with features such as syntax highlighting, brace matching, and automatic indentation, and is also capable of compiling and uploading programs to the board with a single click. There is typically no need to edit makefiles or run programs on a command-line interface.

Arduino programs are written in C or C++. The Arduino IDE comes with a software library called "Wiring" from the original Wiring project, which makes many common input/output operations much easier. Users only need define two functions to make a runnable cyclic executive program:

setup(): a function run once at the start of a program that can initialize settings

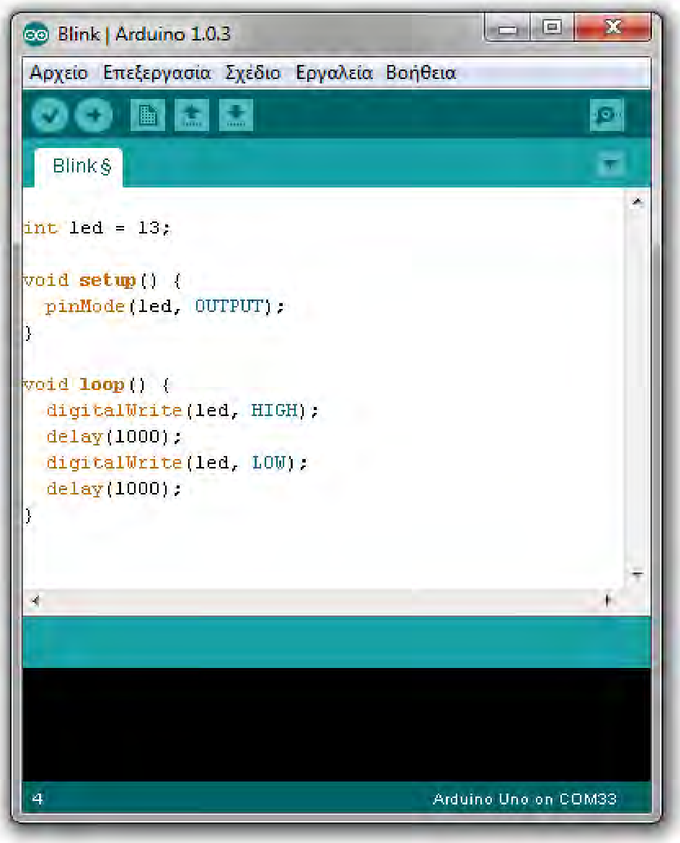
loop(): a function called repeatedly until the board powers off

A typical first program for a microcontroller simply blinks an LED on and off. In the Arduino environment, the user might write a program like this:

```
#define LED_PIN 13

void setup () {
 pinMode (LED_PIN, OUTPUT); // enable pin 13 for
digital output
}

void loop () {
 digitalWrite (LED_PIN, HIGH); // turn on the LED
 delay (1000); // wait one second (1000 milliseconds)
 digitalWrite (LED_PIN, LOW); // turn off the LED
 delay (1000); // wait one second
}
```



```
int led = 13;

void setup() {
  pinMode(led, OUTPUT);
}

void loop() {
  digitalWrite(led, HIGH);
  delay(1000);
  digitalWrite(led, LOW);
  delay(1000);
}
```

Figure 4: Arduino Software IDE

It is a feature of most Arduino boards that they have an LED and load resistor connected between pin 13 and ground, a convenient feature for many simple tests. The previous code would not be seen by a standard C++ compiler as a valid program, so when the user clicks the "Upload to I/O board" button in the IDE, a copy of the code is written to a temporary file with an extra include header at the top and a very simple main() function at the bottom, to make it a valid C++ program.

The Arduino IDE uses the GNU toolchain and AVR Libc to compile programs, and uses avrdude to upload programs to the board.

As the Arduino platform uses Atmel microcontrollers, Atmel's development environment, AVR Studio or the newer Atmel Studio, may also be used to develop software for the Arduino.

3.1.3 Arduino Boards

There are 16 versions of the Arduino Hardware, 2 of them were selected and tested for the mechanism.

3.1.3.1 Arduino Mega 2560

The Arduino Mega 2560 is a microcontroller board based on the ATmega2560. It has 54 digital input/output pins (of which 15 can be used as PWM outputs), 16 analog inputs, 4 UARTs (hardware serial ports), a 16 MHz crystal oscillator, a USB connection, a power jack, an ICSP header, and a reset button. It contains everything needed to support the microcontroller; simply connect it to a computer with a USB cable or power it with a AC-to-DC adapter or battery to get started.

The Mega2560 differs from all preceding boards in that it does not use the FTDI USB-to-serial driver chip. Instead, it features the ATmega16U2 (ATmega8U2 in the revision 1 and revision 2 boards) programmed as a USB-to-serial converter [3].

Power

The Arduino Mega can be powered via the USB connection or with an external power supply. The power source is selected automatically.
External (non-USB) power can come either from an AC-to-DC adapter (wall-wart) or battery. The adapter can be connected by plugging a 2.1mm center-positive plug into the board's power jack. Leads from a battery can be inserted in the Gnd and Vin pin headers of the POWER connector.

The board can operate on an external supply of 6 to 20 volts. If supplied with less than 7V, however, the 5V pin may supply less than five volts and the board may be unstable. If using more than 12V, the voltage regulator may overheat and damage the board. The recommended range is 7 to 12 volts.

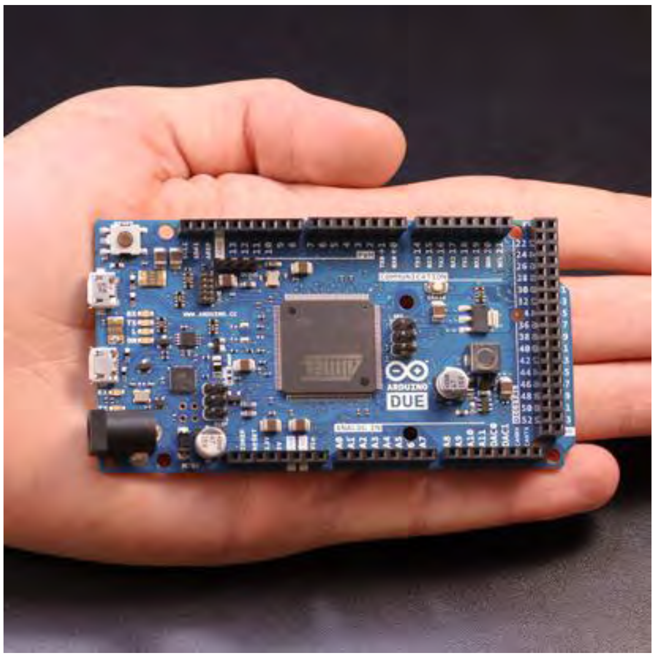


Figure 5: Arduino Mega 2560

The power pins are as follows:

- VIN. The input voltage to the Arduino board when it's using an external power source (as opposed to 5 volts from the USB connection or other regulated power source). You can supply voltage through this pin, or, if supplying voltage via the power jack, access it through this pin.
- 5V. This pin outputs a regulated 5V from the regulator on the board. The board can be supplied with power either from the DC power jack (7 - 12V), the USB connector (5V), or the VIN pin of the board (7-12V). Supplying voltage via the 5V or 3.3V pins bypasses the regulator, and can damage your board. We don't advise it.
- 3V3. A 3.3 volt supply generated by the on-board regulator. Maximum current draw is 50 mA.
- GND. Ground pins.
- IOREF. This pin on the Arduino board provides the voltage reference with which the microcontroller operates. A properly configured shield can read the IOREF pin voltage and select the appropriate power source or enable voltage translators on the outputs for working with the 5V or 3.3V.

Memory

The ATmega2560 has 256 KB of flash memory for storing code (of which 8 KB is used for the bootloader), 8 KB of SRAM and 4 KB of EEPROM (which can be read and written with the EEPROM library).

Input and Output

Each of the 54 digital pins on the Mega can be used as an input or output, using pinMode(), digitalWrite(), and digitalRead() functions. They operate at 5 volts. Each pin can provide or receive a maximum of 40 mA and has an internal pull-up resistor (disconnected by default) of 20-50 kOhms. In addition, some pins have specialized functions:

- **Serial: 0 (RX) and 1 (TX); Serial 1: 19 (RX) and 18 (TX); Serial 2: 17 (RX) and 16 (TX); Serial 3: 15 (RX) and 14 (TX).** Used to receive (RX) and transmit (TX) TTL serial data. Pins 0 and 1 are also connected to the corresponding pins of the ATmega16U2 USB-to-TTL Serial chip.
- **External Interrupts: 2 (interrupt 0), 3 (interrupt 1), 18 (interrupt 5), 19 (interrupt 4), 20 (interrupt 3), and 21 (interrupt 2).** These pins can be configured to trigger an interrupt on a low value, a rising or falling edge, or a change in value. See the attachInterrupt() function for details.
- **PWM: 2 to 13 and 44 to 46.** Provide 8-bit PWM output with the analogWrite() function.
- **SPI: 50 (MISO), 51 (MOSI), 52 (SCK), 53 (SS).** These pins support SPI communication using the SPI library. The SPI pins are also broken out on the ICSP header, which is physically compatible with the Uno, Duemilanove and Diecimila.
- **LED: 13.** There is a built-in LED connected to digital pin 13. When the pin is HIGH value, the LED is on, when the pin is LOW, it's off.
- **TWI: 20 (SDA) and 21 (SCL).** Support TWI communication using the Wire library. Note that these pins are not in the same location as the TWI pins on the Duemilanove or Diecimila.

The Mega2560 has 16 analog inputs, each of which provide 10 bits of resolution (i.e. 1024 different values). By default they measure from ground to 5 volts, though is it possible to change the upper end of their range using the AREF pin and analogReference() function.

There are a couple of other pins on the board:

- AREF. Reference voltage for the analog inputs. Used with analogReference().
- Reset. Bring this line LOW to reset the microcontroller. Typically used to add a reset button to shields which block the one on the board.

Communication

The Arduino Mega2560 has a number of facilities for communicating with a computer, another Arduino, or other microcontrollers. The ATmega2560 provides four hardware UARTs for TTL (5V) serial communication. An ATmega16U2 (ATmega 8U2 on the revision 1 and revision 2 boards) on the board channels one of these over USB and provides a virtual com port to software on the computer (Windows machines will need a .inf file, but OSX and Linux machines will recognize the board as a COM port automatically. The Arduino software includes a serial monitor which allows simple textual data to be sent to and from the board. The RX and TX LEDs on the board will flash when data is being transmitted via the ATmega8U2/ATmega16U2 chip and USB connection to the computer (but not for serial communication on pins 0 and 1).
A SoftwareSerial library allows for serial communication on any of the Mega2560's digital pins.
The ATmega2560 also supports TWI and SPI communication. The Arduino software includes a Wire library to simplify use of the TWI bus; see the documentation for details. For SPI communication, use the SPI library.

Programming

The Arduino Mega can be programmed with the Arduino software (download. The ATmega2560 on the Arduino Mega comes preburned with a bootloader that allows you to upload new code to it without the use of an external hardware programmer. It communicates using the original STK500 protocol (reference, C header files).

3.1.3.2 Ethernet Shield

The Arduino Ethernet Shield allows an Arduino board to connect to the internet. It is based on the **Wiznet W5100 ethernet chip** providing a network (IP) stack capable of both TCP and UDP. The Arduino Ethernet Shield supports up to four simultaneous socket connections. The Ethernet Shield has a standard RJ-45 connection, with an integrated line transformer.

The latest revision of the shield adds a micro-SD card slot, which can be used to store files for serving over the network. It is compatible with the Arduino Uno and Mega (using the Ethernet library). You can access the on-board SD card slot using the SD library which is included in the current Arduino build [4].

The latest revision of the shield also includes a reset controller, to ensure that the W5100 Ethernet module is properly reset on power-up. Previous revisions of the shield were not compatible with the Mega and need to be manually reset after power-up. The reset button on the shield resets both the W5100 and the Arduino board.

Arduino communicates with both the W5100 and SD card using the SPI bus (through the ICSP header). This is on digital pins 11, 12, and 13 on the Duemilanove and pins 50, 51, and 52 on the Mega. On both boards, pin 10 is used to select the W5100 and pin 4 for the SD card. These pins cannot be used for general i/o. On the Mega, the hardware SS pin, 53, is not used to select either the W5100 or the SD card, but it must be kept as an output or the SPI interface won't work.

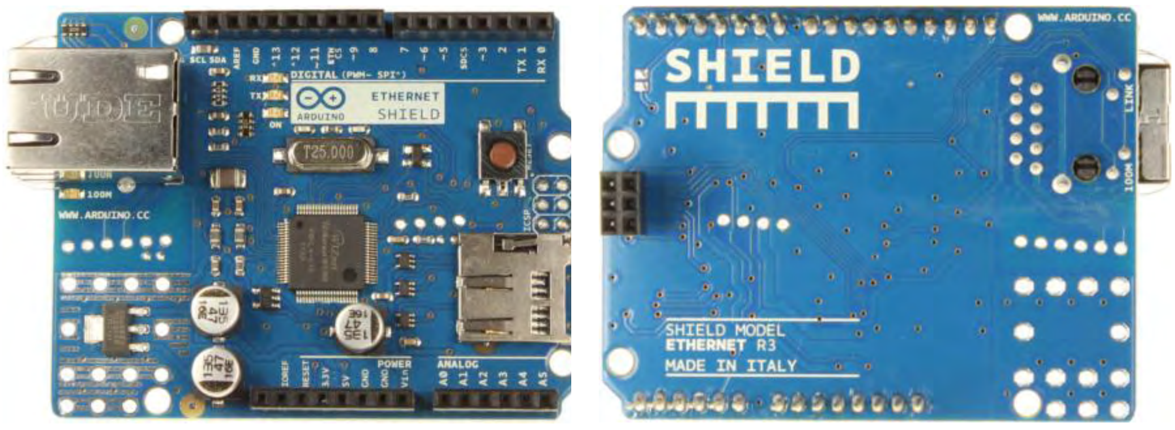


Figure 6: Arduino Ethernet Shield

Because the W5100 and SD card share the SPI bus, only one can be active at a time. If both peripherals are used in the program, this should be taken care of by the corresponding libraries. If none of the peripherals are used in the program, however, it should be deselected. In case of the SD card, pin 4 should be set as an output and writen a high. For the W5100, digital pin 10 must be set as a high output.

The shield contains a number of informational LEDs:

- **PWR**: indicates that the board and shield are powered
- **LINK**: indicates the presence of a network link and flashes when the shield transmits or receives data
- **FULLD**: indicates that the network connection is full duplex
- **100M**: indicates the presence of a 100 Mb/s network connection (as opposed to 10 Mb/s)
- **RX**: flashes when the shield receives data
- **TX**: flashes when the shield sends data
- **COLL**: flashes when network collisions are detected

The solder jumper marked "INT" can be connected to allow the Arduino board to receive interrupt-driven notification of events from the W5100, but this is not supported by the Ethernet library. The jumper connects the INT pin of the W5100 to digital pin 2 of the Arduino.

3.1.3.3 Arduino Ethernet

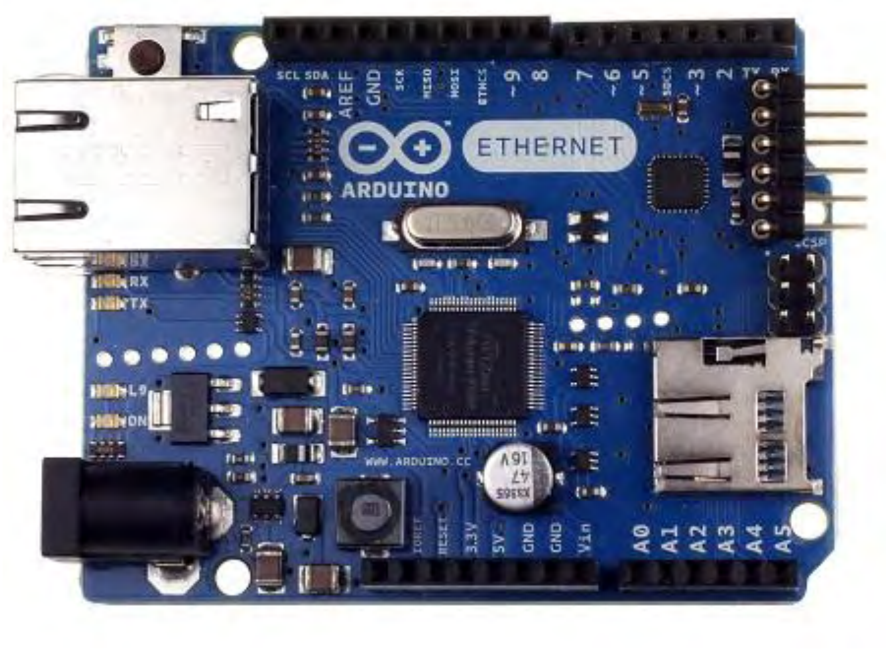


Figure 7: Arduino Ethernet board

The Arduino Ethernet is a microcontroller board based on the ATmega328 (datasheet). It has 14 digital input/output pins, 6 analog inputs, a 16 MHz crystal oscillator, a RJ45 connection, a power jack, an ICSP header, and a reset button.
NB: Pins 10, 11, 12 and 13 are reserved for interfacing with the Ethernet module and should not be used otherwise. This reduces the number of available pins to 9, with 4 available as PWM outputs [5].
An optional Power over Ethernet module can be added to the board as well.
The Ethernet differs from other boards in that it does not have an onboard USB-to-serial driver chip, but has a Wiznet Ethernet interface. This is the same interface found on the Ethernet shield.
An onboard microSD card reader, which can be used to store files for serving over the network, is accessible through the SD Library. Pin 10 is reserved for the Wiznet interface, SS for the SD card is on Pin 4.
The 6-pin serial programming header is compatible with the USB Serial adapter and also with the FTDI USB cables or with Sparkfun and Adafruit FTDI-style basic USB-to-

serial breakout boards. It features support for automatic reset, allowing sketches to be uploaded without pressing the reset button on the board. When plugged into a USB to Serial adapter, the Arduino Ethernet is powered from the adapter.

Power

The board can also be powered via an external power supply, an optional Power over Ethernet (PoE) module, or by using a FTDI cable/USB Serial connector [6].

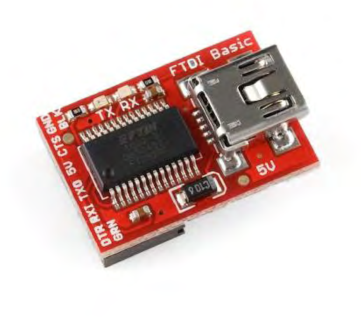


Figure 8: FTDI Basic Breakout – 5V

External power can come either from an AC-to-DC adapter (wall-wart) or battery. The adapter can be connected by plugging a 2.1mm center-positive plug into the board's power jack. Leads from a battery can be inserted in the Gnd and Vin pin headers of the POWER connector.
The board can operate on an external supply of 6 to 20 volts. If supplied with less than 7V, however, the 5V pin may supply less than five volts and the board may be unstable. If using more than 12V, the voltage regulator may overheat and damage the board. The recommended range is 7 to 12 volts.
The power pins are as follows:

- VIN. The input voltage to the Arduino board when it's using an external power source (as opposed to 5 volts from the USB connection or other regulated power source). You can supply voltage through this pin, or, if supplying voltage via the power jack, access it through this pin.
- 5V. This pin outputs a regulated 5V from the regulator on the board. The board can be supplied with power either from the DC power jack (7 - 12V), the USB connector (5V), or the VIN pin of the board (7-12V). Supplying voltage via the 5V or 3.3V pins bypasses the regulator, and can damage your board. We don't advise it.
- 3V3. A 3.3 volt supply generated by the on-board regulator. Maximum current draw is 50 mA.
- GND. Ground pins.
- IOREF. This pin on the Arduino board provides the voltage reference with which the microcontroller operates. A properly configured shield can read

the IOREF pin voltage and select the appropriate power source or enable voltage translators on the outputs for working with the 5V or 3.3V.

When using the power adapter, power can come either from an AC-to-DC adapter (wall-wart) or battery. The adapter can be connected by plugging a 2.1mm center-positive plug into the board's power jack. Leads from a battery can be inserted in the Gnd and Vin pin headers of the POWER connector.
The board can operate on an external supply of 6 to 20 volts. If supplied with less than 7V, however, the 5V pin may supply less than five volts and the board may be unstable. If using more than 12V, the voltage regulator may overheat and damage the board. The recommended range is 7 to 12 volts.

Memory

The ATmega328 has 32 KB (with 0.5 KB used for the bootloader). It also has 2 KB of SRAM and 1 KB of EEPROM (which can be read and written with the EEPROM library).

Input and Output

Each of the 14 digital pins on the Ethernet board can be used as an input or output, using pinMode(), digitalWrite(), and digitalRead() functions. They operate at 5 volts. Each pin can provide or receive a maximum of 40 mA and has an internal pull-up resistor (disconnected by default) of 20-50 kOhms. In addition, some pins have specialized functions:

- Serial: 0 (RX) and 1 (TX). Used to receive (RX) and transmit (TX) TTL serial data.
- External Interrupts: 2 and 3. These pins can be configured to trigger an interrupt on a low value, a rising or falling edge, or a change in value. See the attachInterrupt() function for details.
- PWM: 3, 5, 6, 9, and 10. Provide 8-bit PWM output with the analogWrite() function.
- SPI: 10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK). These pins support SPI communication using the SPI library.
- LED: 9. There is a built-in LED connected to digital pin 9. When the pin is HIGH value, the LED is on, when the pin is LOW, it's off. On most other arduino boards, this LED is found on pin 13. It is on pin 9 on the Ethernet board because pin 13 is used as part of the SPI connection.

The Ethernet board has 6 analog inputs, labeled A0 through A5, each of which provide 10 bits of resolution (i.e. 1024 different values). By default they measure from ground to 5 volts, though is it possible to change the upper end of their range using the AREF pin and the analogReference() function. Additionally, some pins have specialized functionality:

- TWI: A4 (SDA) and A5 (SCL). Support TWI communication using the Wire library.

There are a couple of other pins on the board:

- AREF. Reference voltage for the analog inputs. Used with analogReference().
- Reset. Bring this line LOW to reset the microcontroller. Typically used to add a reset button to shields which block the one on the board.

See also the mapping between Arduino pins and ATmega328 ports.

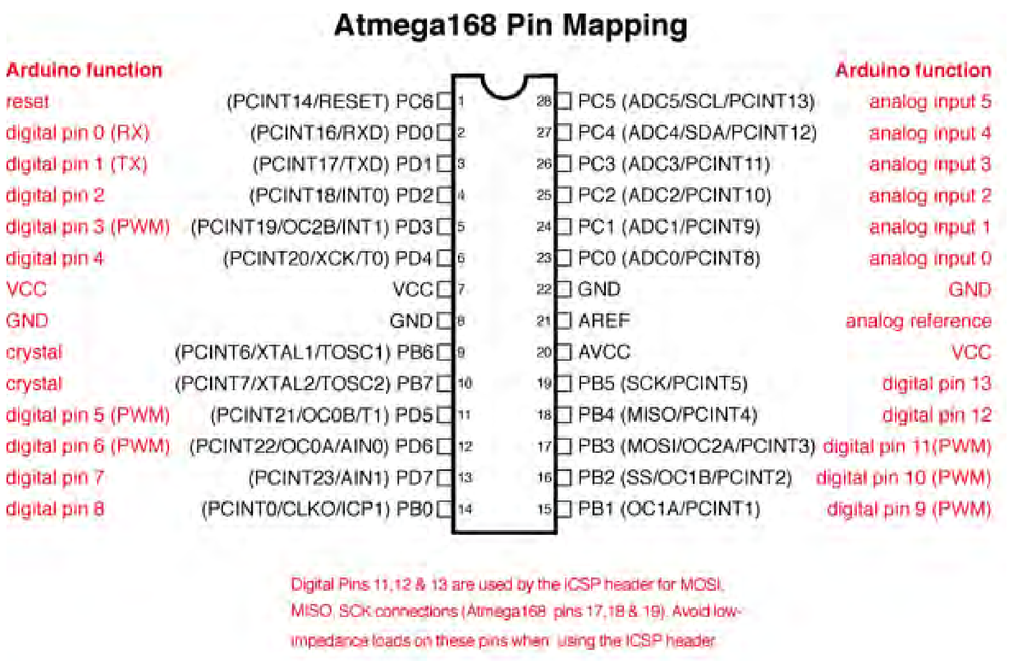


Figure 9: Atmega168 Pin Mapping

Communication

The Arduino Ethernet has a number of facilities for communicating with a computer, another Arduino, or other microcontrollers.

A **SoftwareSerial library** allows for serial communication on any of the Uno's digital pins.

The ATmega328 also supports TWI and SPI communication. The Arduino software includes a Wire library to simplify use of the TWI bus; see the documentation for details. For SPI communication, use the SPI library.

The board also can connect to a wired network via ethernet. When connecting to a network, you will need to provide an IP address and a MAC address. The Ethernet Library is fully supported.

The onboard microSD card reader is accessible through the SD Library. When working with this library, SS is on Pin 4.

Programming

It is possible to program the Arduino Ethernet board in two ways: through the 6 pin serial programming header, or with an external ISP programmer.

The 6-pin serial programming header is compatible with FTDI USB cables and the Sparkfun and Adafruit FTDI-style basic USB-to-serial breakout boards including the Arduino USB-Serial connector. It features support for automatic reset, allowing sketches to be uploaded without pressing the reset button on the board. When plugged into a FTDI-style USB adapter, the Arduino Ethernet is powered off the adapter.

Ethernet board can also be programmed with an external programmer like an AVRISP mkII or USBTinyISP. All the Ethernet example sketches work as they do with the Ethernet shield.

3.1.3.4 Comparison of Arduino Mega 2560 and Arduino Ethernet

In general, Arduino Mega 2560 has more processing power than Arduino Ethernet but it was used mostly for debugging reasons and because of its broader connectivity. On the other hand, Arduino Ethernet is more cost-effective device and the sketch was developed in order to be fully compatible and operational with this low cost board despite its limitations. In the following table you can see the comparison of their characteristics.

Arduino Board	Arduino Mega	Arduino Ethernet
Microcontroller	ATmega2560	ATmega328
Operating Voltage	5V	5V
Input Voltage	7-12V	7-12V
Digital I/O Pins	54 (15 PWM)	14 (4 PWM)
Analog Input Pins	16	6
DC Current per I/O	40 mA	40 mA
UARTs	4	1 (shared with USB)
USB serial	embedded	External
Flash Memory	256 KB (8KB	32KB (0.5KB bootloader)
SRAM, EEPROM	8 KB, 4 KB	2 KB, 1 KB
Clock Speed	16MHz	16MHz
Ethernet controller	W5100 TCP/IP (Shield)	W5100 TCP/IP
SD support	Ethernet Shield	Embedded

3.1.4 Arduino Libraries

The Arduino environment can be extended through the use of libraries, just like most programming platforms. Libraries provide extra functionality for use in sketches, e.g. working with hardware or manipulating data. A number of libraries come installed with the IDE, but here we will mention the ones and their functions that were mostly used [7].

3.1.4.1 Ethernet library

With the Arduino Ethernet Shield, this library allows an Arduino board to connect to the internet. It can serve as either a server accepting incoming connections or a client making outgoing ones. The library supports up to four concurrent connection (incoming or outgoing or a combination) [8].

Arduino communicates with the shield using the SPI bus. This is on digital pins 11, 12, and 13 on the Uno and pins 50, 51, and 52 on the Mega. On both boards, pin 10 is used as SS. On the Mega, the hardware SS pin, 53, is not used to select the W5100, but it must be kept as an output or the SPI interface won't work.

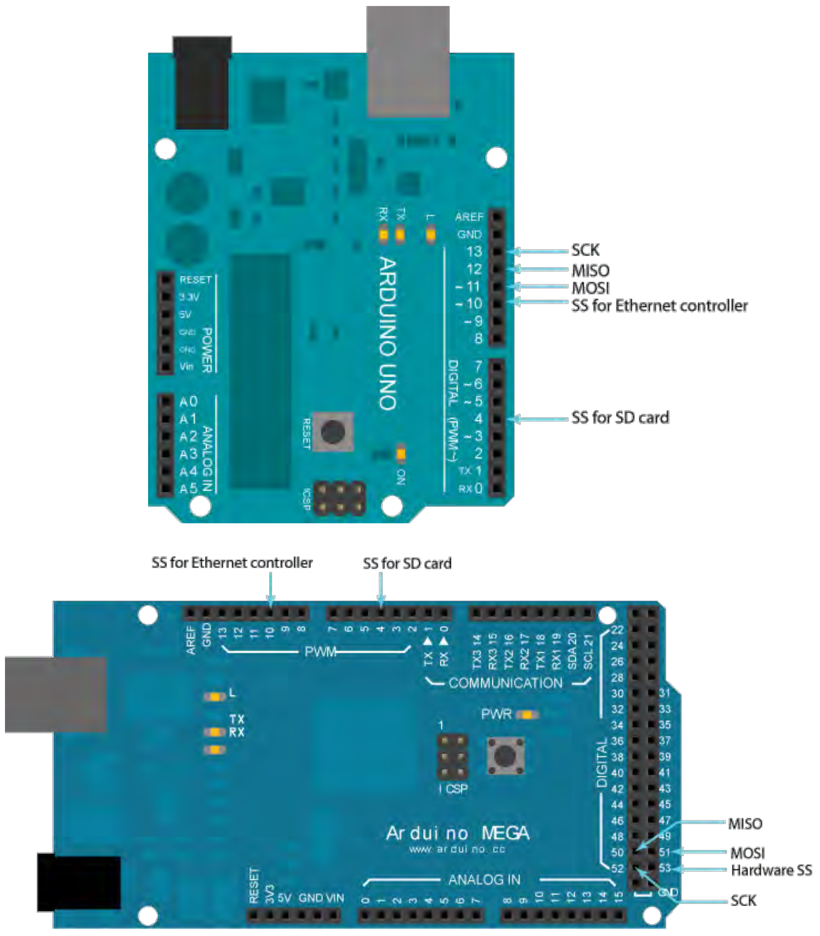


Figure 10: Arduino Pins affected by Ethernet shield

Ethernet class

The Ethernet class initializes the Ethernet library and network settings.

- **begin():** Initializes the Ethernet library and network settings.
- **localIP():** Obtains the IP address of the Ethernet shield. Useful when the address is auto assigned through DHCP.

IPAddress class

The IPAddress class works with local and remote IP addressing.

- **IPAddress():** Defines an IP address. It can be used to declare both local and remote addresses.

Server class

The Server class creates servers which can send data to and receive data from connected clients (programs running on other computers or devices).
Server

- **EthernetServer():** Create a server that listens for incoming connections on the specified port.
- **begin():** Tells the server to begin listening for incoming connections.
- **available():** Gets a client that is connected to the server and has data available for reading. The connection persists when the returned client object goes out of scope; you can close it by calling client.stop(). Returns a Client object; if no Client has data available for reading, this object will evaluate to false in an if-statement
- **write():** Write data to all the clients connected to a server. This data is sent as a byte or series of bytes.
- **print():** Print data to all the clients connected to a server. Prints numbers as a sequence of digits, each an ASCII character (e.g. the number 123 is sent as the three characters '1', '2', '3').
- **println():** Print data, followed by a newline, to all the clients connected to a server. Prints numbers as a sequence of digits, each an ASCII character (e.g. the number 123 is sent as the three characters '1', '2', '3').

Client class

The client class creates clients that can connect to servers and send and receive data.
Client: Client is the base class for all Ethernet client based calls. It is not called directly, but invoked whenever you use a function that relies on it.

- **EthernetClient():** Creates a client which can connect to a specified internet IP address and port (defined in the client.connect() function).
- **if (EthernetClient):** Indicates if the specified Ethernet client is ready.
- **connected():** Whether or not the client is connected. Note that a client is considered connected if the connection has been closed but there is still unread data.
- **connect():** Connects to a specified IP address and port. The return value indicates success or failure. Also supports DNS lookups when using a domain name.
- **write():** Write data to the server the client is connected to. This data is sent as a byte or series of bytes.
- **print():** Print data to the server that a client is connected to. Prints numbers as a sequence of digits, each an ASCII character (e.g. the number 123 is sent as the three characters '1', '2', '3').
- **println():** Print data, followed by a carriage return and newline, to the server a client is connected to. Prints numbers as a sequence of digits, each an ASCII character (e.g. the number 123 is sent as the three characters '1', '2', '3').

- **available():** Returns the number of bytes available for reading (that is, the amount of data that has been written to the client by the server it is connected to).
- **read():** Read the next byte received from the server the client is connected to (after the last call to read()).
- **flush():** Discard any bytes that have been written to the client but not yet read.
- **stop():** Disconnect from the server.

3.1.4.2 SD Library

The SD library allows for reading from and writing to SD cards, e.g. on the Arduino Ethernet Shield. It is built on sdfatlib by William Greiman. The library supports FAT16 and FAT32 file systems on standard SD cards and SDHC cards. It uses short 8.3 names for files. The file names passed to the SD library functions can include paths separated by forward-slashes, /, e.g. "directory/filename.txt". Because the working directory is always the root of the SD card, a name refers to the same file whether or not it includes a leading slash (e.g. "/file.txt" is equivalent to "file.txt"). As of version 1.0, the library supports opening multiple files [9].

The communication between the microcontroller and the SD card uses SPI, which takes place on digital pins 11, 12, and 13 (on most Arduino boards) or 50, 51, and 52 (Arduino Mega). Additionally, another pin must be used to select the SD card. This can be the hardware SS pin - pin 10 (on most Arduino boards) or pin 53 (on the Mega) - or another pin specified in the call to SD.begin(). Note that even if you don't use the hardware SS pin, it must be left as an output or the SD library won't work.

SD class

The SD class provides functions for accessing the SD card and manipulating its files and directories.

- **begin():** Initializes the SD library and card. This begins use of the SPI bus (digital pins 11, 12, and 13 on most Arduino boards; 50, 51, and 52 on the Mega) and the chip select pin, which defaults to the hardware SS pin (pin 10 on most Arduino boards, 53 on the Mega). Note that even if you use a different chip select pin, the hardware SS pin must be kept as an output or the SD library functions will not work.
- **exists():** Tests whether a file or directory exists on the SD card.
- **mkdir():** Create a directory on the SD card. This will also create any intermediate directories that don't already exists; e.g. SD.mkdir("a/b/c") will create a, b, and c.
- **open():** Opens a file on the SD card. If the file is opened for writing, it will be created if it doesn't already exist (but the directory containing it must already exist).
- **remove():** Remove a file from the SD card.
- **rmdir():** Remove a directory from the SD card. The directory must be empty.

File class

The File class allows for reading from and writing to individual files on the SD card.

- **available():** Check if there are any bytes available for reading from the file.

- **close():** Close the file, and ensure that any data written to it is physically saved to the SD card.
- **flush():** Ensures that any bytes written to the file are physically saved to the SD card. This is done automatically when the file is closed.
- **peek():** Read a byte from the file without advancing to the next one. That is, successive calls to peek() will return the same value, as will the next call to read().
- **position():** Get the current position within the file (i.e. the location to which the next byte will be read from or written to).
- **print():** Print data to the file, which must have been opened for writing. Prints numbers as a sequence of digits, each an ASCII character (e.g. the number 123 is sent as the three characters '1', '2', '3').
- **println():** Print data, followed by a carriage return and newline, to the File, which must have been opened for writing. Prints numbers as a sequence of digits, each an ASCII character (e.g. the number 123 is sent as the three characters '1', '2', '3').
- **seek():** Seek to a new position in the file, which must be between 0 and the size of the file (inclusive).
- **size():** Get the size of the file.
- **read():** Read a byte from the file.
- **write():** Write data to the file.
- **isDirectory():** Directories (or folders) are special kinds of files, this function reports if the current file is a directory or not.
- **openNextFile():** Reports the next file or folder in a directory.
- **rewindDirectory():** rewindDirectory() will bring you back to the first file in the directory, used in conjunction with openNextFile().

3.1.4.3 SoftwareSerial Library

The Arduino hardware has built-in support for serial communication on pins 0 and 1 (which also goes to the computer via the USB connection). The native serial support happens via a piece of hardware (built into the chip) called a UART. This hardware allows the Atmega chip to receive serial communication even while working on other tasks, as long as there room in the 64 byte serial buffer [10].

The SoftwareSerial library has been developed to allow serial communication on other digital pins of the Arduino, using software to replicate the functionality (hence the name "SoftwareSerial"). It is possible to have multiple software serial ports with speeds up to 115200 bps. A parameter enables inverted signaling for devices which require that protocol.

The library has the following known limitations:

- If using multiple software serial ports, only one can receive data at a time.

- Not all pins on the Mega and Mega 2560 support change interrupts, so only the following can be used for RX: 10, 11, 12, 13, 14, 15, 50, 51, 52, 53, A8 (62), A9 (63), A10 (64), A11 (65), A12 (66), A13 (67), A14 (68), A15 (69).
- Not all pins on the Leonardo support change interrupts, so only the following can be used for RX: 8, 9, 10, 11, 14 (MISO), 15 (SCK), 16 (MOSI).

SoftwareSerial class

- **SoftwareSerial(rxPin, txPin)**: A call to SoftwareSerial(rxPin, txPin) creates a new SoftwareSerial object, whose name you need to provide as in the example below.
- **available():** Get the number of bytes (characters) available for reading from a software serial port. This is data that's already arrived and stored in the serial receive buffer.
- **begin(speed):** Sets the speed (baud rate) for the serial communication. Supported baud rates are 300, 600, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 31250, 38400, 57600, and 115200.
- **isListening():** Tests to see if requested software serial port is actively listening.
- **overflow():** Tests to see if a software serial buffer overflow has occurred. Calling this function clears the overflow flag, meaning that subsequent calls will return false unless another byte of data has been received and discarded in the meantime.

The software serial buffer can hold 64 bytes.

- **peek():** Return a character that was received on the RX pin of the software serial port. Unlike read(), however, subsequent calls to this function will return the same character.

Note that only one SoftwareSerial instance can receive incoming data at a time (select which one with the listen() function).

- **read():** Return a character that was received on the RX pin of the software serial port. Note that only one SoftwareSerial instance can receive incoming data at a time (select which one with the listen() function).
- **print():** Prints data to the transmit pin of the software serial port. Works the same as the Serial.print() function.
- **println():** Prints data to the transmit pin of the software serial port, followed by a carriage return and line feed. Works the same as the Serial.println() function.
- **listen():** Enables the selected software serial port to listen. Only one software serial port can listen at a time; data that arrives for other ports will be discarded. Any data already received is discarded during the call to listen() (unless the given instance is already listening).
- **write():** Prints data to the transmit pin of the software serial port as raw bytes. Works the same as the Serial.write() function.

3.1.4.4 Servo Library

This library allows an Arduino board to control RC (hobby) servo motors. Servos have integrated gears and a shaft that can be precisely controlled. Standard servos allow the shaft to be positioned at various angles, usually between 0 and 180 degrees. Continuous rotation servos allow the rotation of the shaft to be set to various speeds.

The Servo library supports up to 12 motors on most Arduino boards and 48 on the Arduino Mega. On boards other than the Mega, use of the library disables analogWrite() (PWM) functionality on pins 9 and 10, whether or not there is a Servo on those pins. On the Mega, up to 12 servos can be used without interfering with PWM functionality; use of 12 to 23 motors will disable PWM on pins 11 and 12 [11].

Servo motors have three wires: power, ground, and signal. The power wire is typically red, and should be connected to the 5V pin on the Arduino board. The ground wire is typically black or brown and should be connected to a ground pin on the Arduino board. The signal pin is typically yellow, orange or white and should be connected to a digital pin on the Arduino board. Note that servos draw considerable power, so if you need to drive more than one or two, you'll probably need to power them from a separate supply (i.e. not the +5V pin on your Arduino). Be sure to connect the grounds of the Arduino and external power supply together.

- **attach():** Attach the Servo variable to a pin. Note that in Arduino 0016 and earlier, the Servo library supports only servos on only two pins: 9 and 10.
- **write():** Writes a value to the servo, controlling the shaft accordingly. On a standard servo, this will set the angle of the shaft (in degrees), moving the shaft to that orientation. On a continuous rotation servo, this will set the speed of the servo (with 0 being full-speed in one direction, 180 being full speed in the other, and a value near 90 being no movement).
- **writeMicroseconds():** Writes a value in microseconds (uS) to the servo, controlling the shaft accordingly. On a standard servo, this will set the angle of the shaft. On standard servos a parameter value of 1000 is fully counter-clockwise, 2000 is fully clockwise, and 1500 is in the middle.

Note that some manufactures do not follow this standard very closely so that servos often respond to values between 700 and 2300. Feel free to increase these endpoints until the servo no longer continues to increase its range. Note however that attempting to drive a servo past its endpoints (often indicated by a growling sound) is a high-current state, and should be avoided.

- **read():** Read the current angle of the servo (the value passed to the last call to write()).
- **attached():** Check whether the Servo variable is attached to a pin.
- **detach():** Detach the Servo variable from its pin. If all Servo variables are detached, then pins 9 and 10 can be used for PWM output with analogWrite().

3.2 Servos

Servos were used to accurately control physical movement as they generally move to a position instead of continuously rotating. Servos are commonly electrical, using an electric motor as the primary means of creating mechanical force. They are designed to set their position according to an input pulse width given in microseconds. Moreover, servos are easy to connect and control as the motor driver is built internally. Arduino supports Pulse Width Modulation output and additionally provides a built-in Servo library. Thereby, it is an ideal choice for the servomechanisms' control.
Servos contain a small motor connected through gears to an output shaft. The output shaft drives a servo arm and is also connected to a potentiometer to provide position feedback to an internal control circuit (see figure 18)

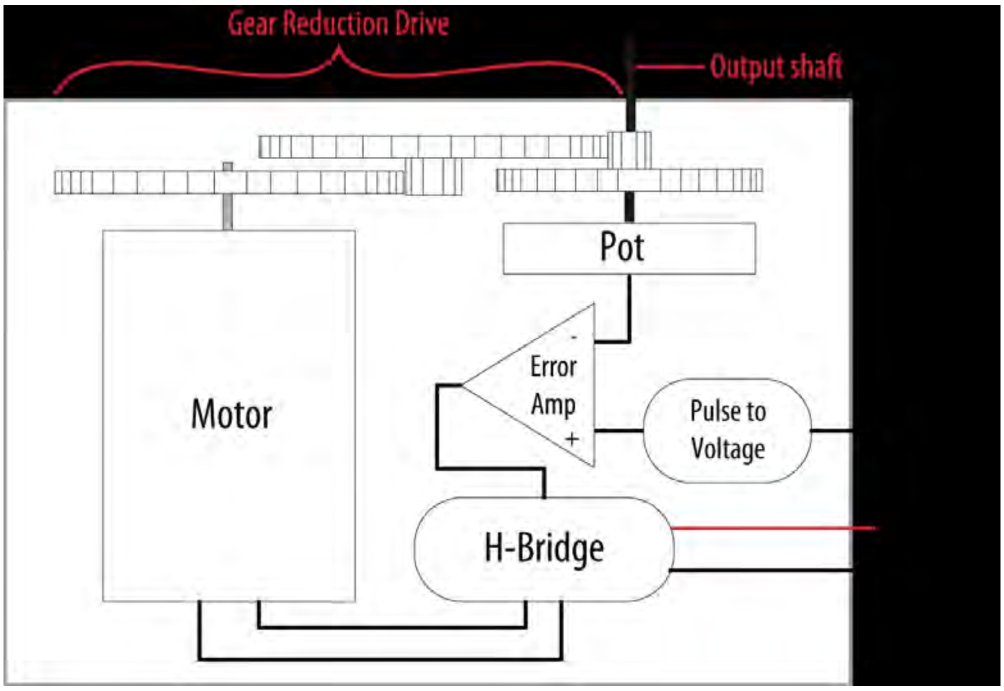


Figure 11: Elements inside a servo

Servos respond to changes in the duration of a pulse. A short pulse of 1 ms or less will cause the servo to rotate to one extreme; a pulse duration of 2 ms or so will rotate the servo to the other extreme (see Figure 19). Pulses ranging between these values will rotate the servo to a position proportional to the pulse width [12].

On the market there are available full rotations (360 degrees) and regular (usually 0-180 degrees) servos. The ones that rotate over a range of 0 to 180 are of the same structure but with a different gearbox. As a result, they lack of precision and they need further calibration to meet the requirements of the implementation.

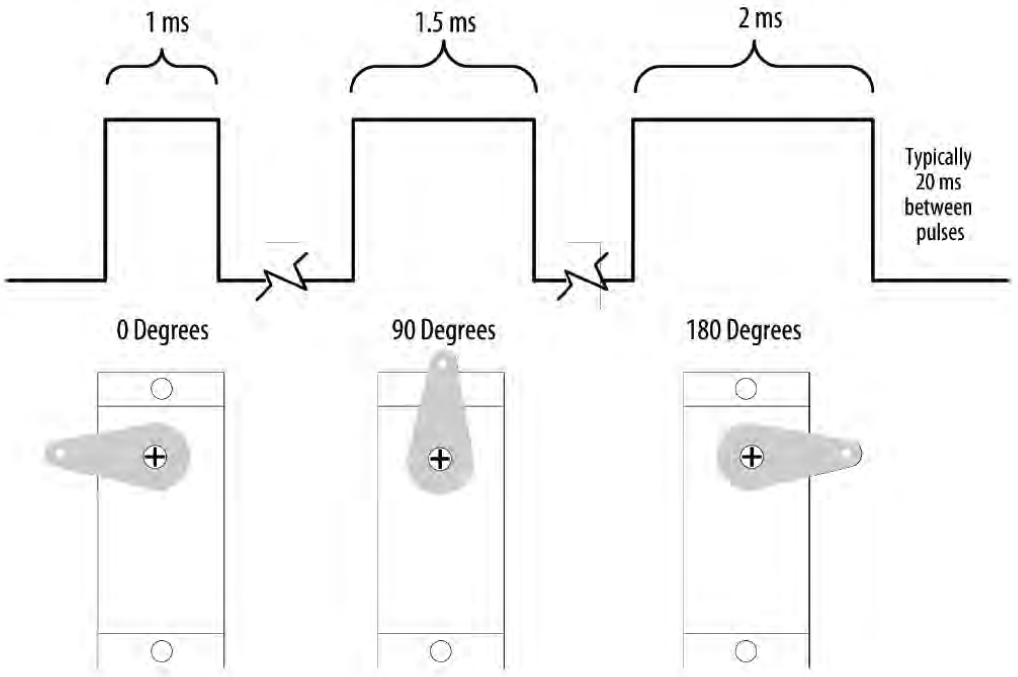


Figure 12: Relationship between the pulse width and the servo angle; the servo output arm moves proportionally as the pulse width increases from 1 ms to 2 ms

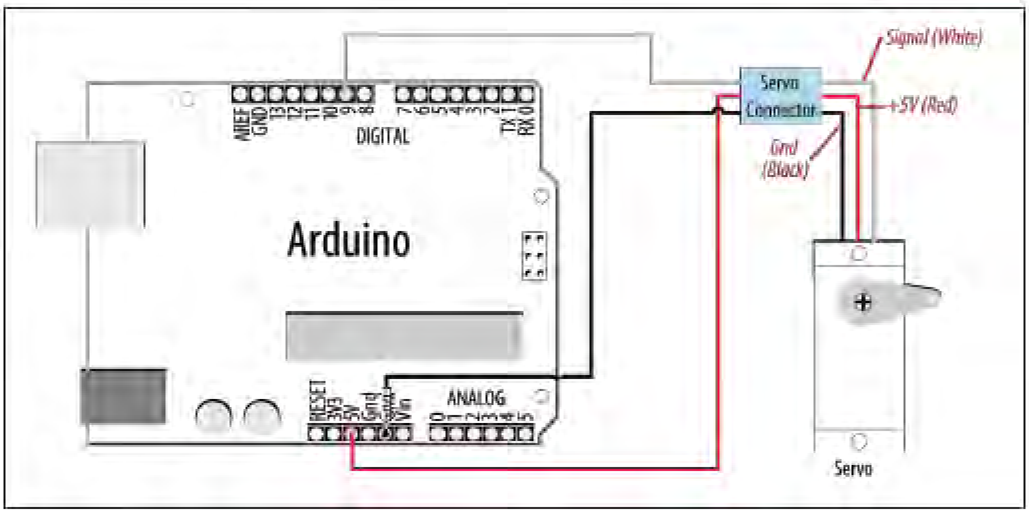


Figure 13: Connecting a servo with Arduino

3.2.1 Servos specification

Two different servos were selected, a full rotation GWS125-1T for the yaw movement and a 180 rotation sub-micro servo for the pitch. The sub-micro servo actually ranges from 0 to 160 but a range of 0-90 was adequate for the requirements.

GWS S125-1T specifications

Control System: +Pulse Width Control 1500usec Neutral
Required Pulse: 3-5 Volt Peak to Peak Square Wave
Operating Voltage: 4.8-6.0 Volts
Operating Speed (4.8V): 0.26sec/60° at no load
Operating Speed (6.0V): 0.21sec/60° at no load
Stall Torque (4.8V): 92 oz/in. (6.6kg.cm)
Stall Torque (6.0V): 106 oz/in. (7.6kg.cm)
Operating Angle: 360º
Bearing Type: Dual Ball Bearing [13]

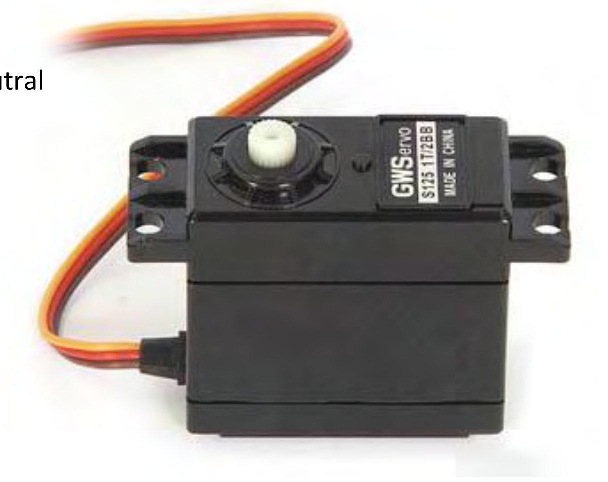


Figure 14: GWS S125-1T Full rotation Servo

Sub-micro servo specifications

Control System: +Pulse Width Control 1500usec Neutral
Required Pulse: 3-5 Volt Peak to Peak Square Wave
Operating Voltage: 4.8-6.0 Volts
Operating Speed (4.8V): 0.15sec/60° at no load
Operating Speed (6.0V): 0.10sec/60° at no load
Stall Torque (4.8V): 16.6 oz/in. (1.2kg.cm)
Stall Torque (6.0V): 20.8 oz/in. (1.5kg.cm)
Operating Angle: 160º
Bearing Type: Single Top Ball Bearing [14]

Figure 15: Sub-micro servo

3.2.2 Servos Calibration

3.2.2.1 Calibration using the minimum/maximum angles

The servo sets its position according to a pulse width. Consequently there is a minimum pulse width in microseconds, corresponding to the minimum angle (544 for 0 degrees) as well as a maximum pulse width, corresponding to the maximum angle. By giving the maximum pulse width as input to the servo, the maximum angle in degrees can be found using the protractor. In this way the calibration is achieved by mapping the minimum and maximum pulse width with the respective angle's values in degrees.

3.2.2.2 Calibration using a razor sensor

The servo is centered on a 360 degrees protractor according to the yaw rotation. Then an adequate number of measurements from razor sensors are taken. In combination with the respective optical measurements the actual angle for every pulse width, that the servo supports, is estimated. The data were processed in Matlab using the polynomial curve fitting function (polyfit) to define the polynomial relation that best fits the angles and pulse width microseconds.

The coefficients of the polynomial of third degree are used to calculate the proper input servo value for the desired angle.

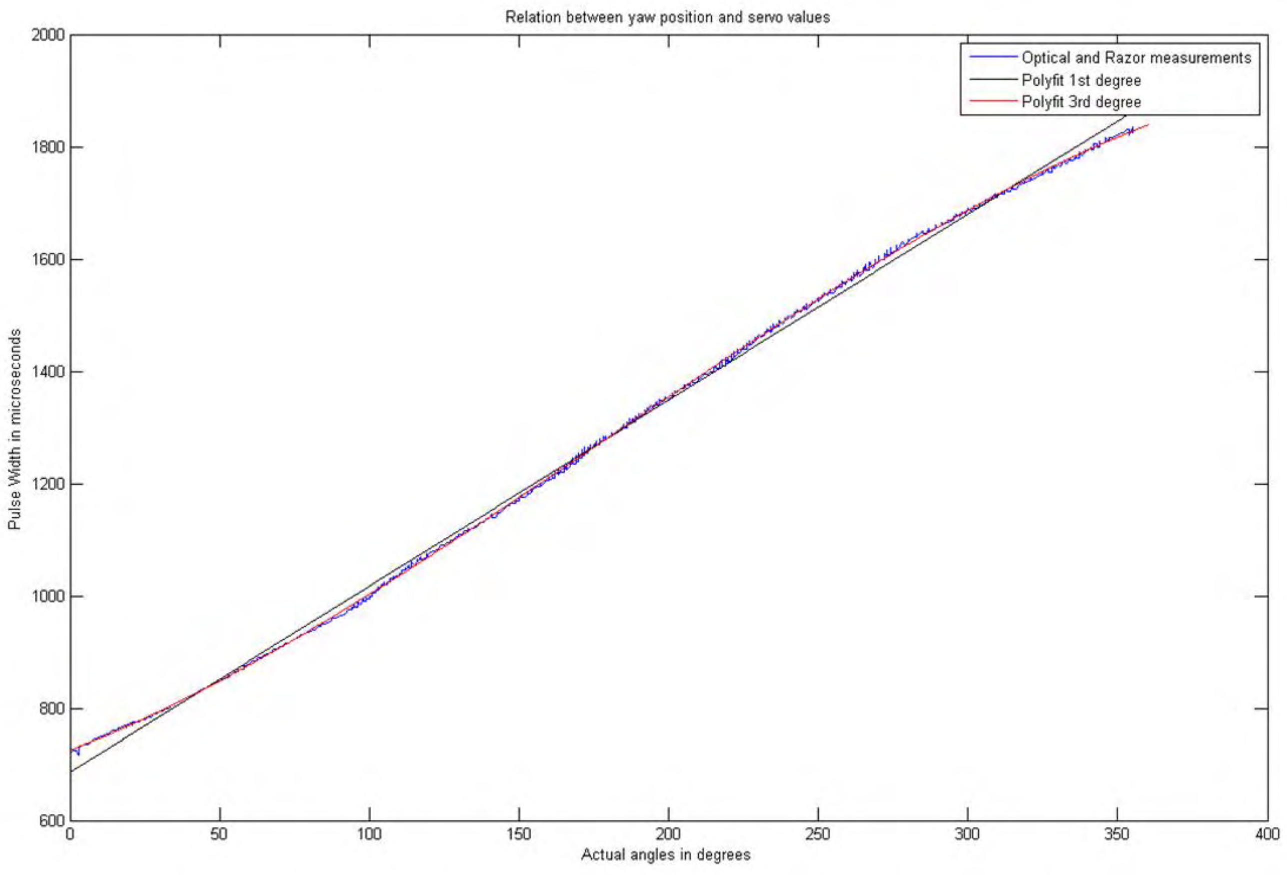


Figure 16: This figure shows that polyfit of 3rd degree perfectly fits the curve of the observations

In the same way, pitch servo was calibrated in order to have better precision with the difference that only the range from 0 to 90 degrees was important.

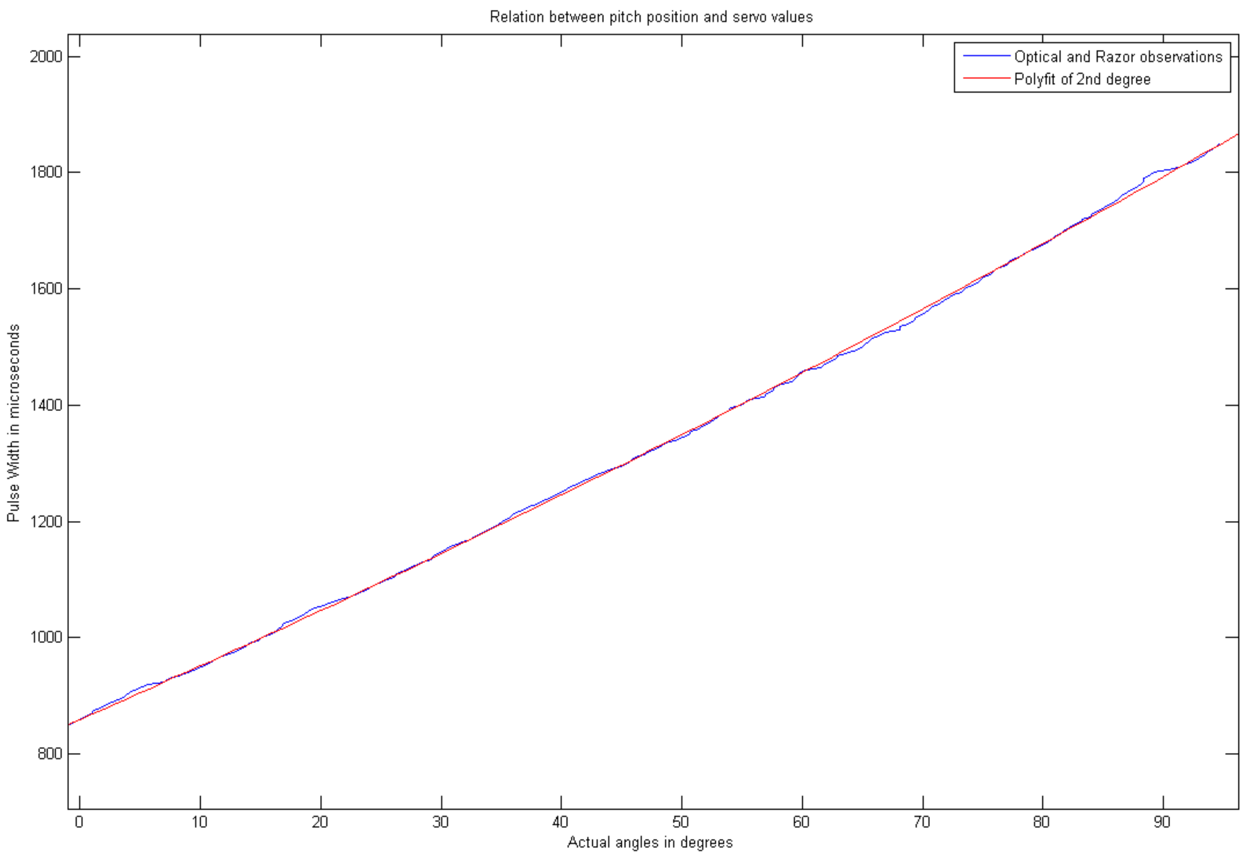


Figure 17: 2nd degree polynomial was adequate to predict the proper servo value.

3.3 Razor IMU

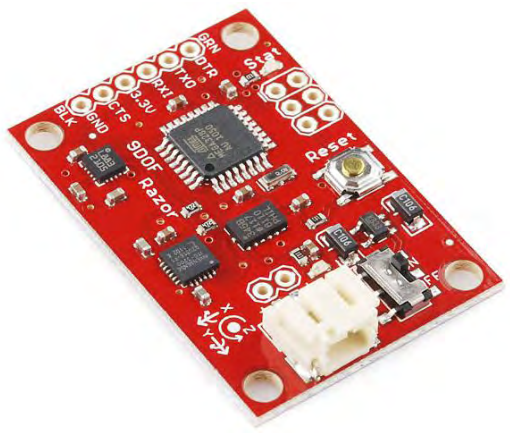


Figure 18: Razor IMU Board

The 9DOF Razor IMU incorporates three sensors:

- an ITG-3200 MEMS triple-axis digital-output gyroscope
- an ADXL345 - 13-bit resolution triple-axis accelerometer
- an HMC5883L triple-axis, digital magnetometer

It gives nine degrees of inertial measurement and specifically the yaw, pitch and roll position in degrees [15]. The outputs of all sensors are processed by an on-board ATmega328 and sent out over a serial interface. Its 3.5-16VDC input is powered from Arduino's 3.3V line. If Arduino Ethernet is used, it is connected via Serial Software library to Arduino's digital pins. In case of Arduino Mega, it is connected through one of the four serial interfaces. Due to limited computational power of Arduino, the Razor IMU was reprogrammed so that the serial clock operate lower to 9600 baud rate without losing data. In the case of Arduino Mega, it is connected to its hardware serial interface. Razor IMU is initialized not to send the output via stream but by demand of Arduino to avoid unnecessary computations. The serial commands are explained in the chapter 3.3.4.

The mechanism takes advantage of Razor sensors to send feedback information of actual position after movement command received and optionally, to correct servo positions until Razor's sensors values meet the desired angles.

3.3.1 Triple-Axis Digital-Output Gyroscope – ITG3200

Figure 19: ITG3200 Chip

Understanding gyros

Gyroscopes measure angular velocity, how fast something is spinning about an axis. In order to monitor the orientation of an object in motion, an accelerometer may not give enough information to know exactly how it's oriented. Unlike accelerometers gyros are not affected by gravity, so they make a great complement to each other. Usually, angular velocity is represented in units of rotations per minute (RPM), or degrees per second (°/s). The three axes of rotation are either referenced as x, y, and z, or roll, pitch, and yaw [16].

In the past, gyros have been used for space navigation, missile control, under-water guidance, and flight guidance. Now they are starting to be used alongside accelerometers for applications like motion-capture and vehicle navigation.

ITG3200 Features

- A full-scale range of ±2000°/sec
- Digital-output X-, Y-, and Z-Axis angular rate sensors (gyros) on one integrated circuit
- Digitally-programmable low-pass filter
- Low 6.5mA operating current consumption for long battery life
- Wide VDD supply voltage range of 2.1V to 3.6V
- Standby current: 5µA
- Digital-output temperature sensor
- Fast Mode I2C (400kHz) serial interface
- Optional external clock inputs of 32.768kHz or 19.2MHz to synchronize with system clock

3.3.2 Triple Axis Accelerometer - ADXL345

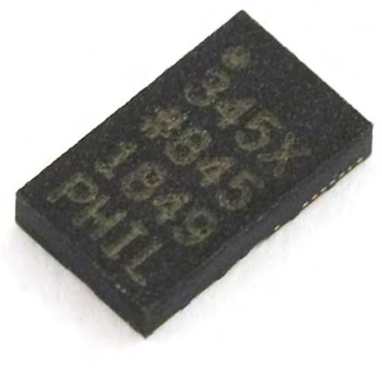


Figure 20: ADXL345 Chip

Understanding accelerometer

Accelerometers are devices that measure acceleration, which is the rate of change of the velocity of an object. They measure in meters per second squared (m/s2) or in G-forces (g). A single G-force for us here on planet Earth is equivalent to 9.8 m/s2, but this does vary slightly with elevation (and will be a different value on different planets due to variations in gravitational pull). Accelerometers are useful for sensing vibrations in systems or for orientation applications [17].

ADXL features

- 1.8V to 3.6V supply
- Low Power: 25 to 130uA @ 2.5V
- SPI and I2C interfaces
- Up to 13bit resolution at +/-16g
- Tap/Double Tap detection
- Activity/Inactivity monitoring
- Free-Fall detection

3.3.3 Magnetometer Digital Triple Axis - HMC5883L

Figure 21: HMC5883L Chip

Understanding magnetometers

A magnetometer is a measuring instrument used to measure the strength and, in some cases, the direction of magnetic fields. In recent years magnetometers have been miniaturised to the extent that they can be incorporated in integrated circuits at very low cost and are finding increasing use as compasses in consumer devices such as mobile phones and tablet computers [18].

HMC5883L features

- Simple I2C interface
- 2.16-3.6VDC supply voltage range
- Low current draw
- ±8 Guass and a resolution of up to 5 milli-Gauss

3.3.4 Razor AHRS firmware v1.4.1

The AHRS firmware was implemented by Quality & Usability Lab, Deutsche Telekom Laboratories, TU Berlin for the development of an attitude and heading reference system. It is released under GNU General Public License v3.0. The firmware was loaded with the help of an FTDI Basic Breakout – 3.3V board. The axis definition of the firmware is:

X axis pointing forward (towards the short edge with the connector holes)
Y axis pointing to the right
Z axis pointing down

Positive yaw : clockwise
Positive roll : right wing down
Positive pitch : nose up

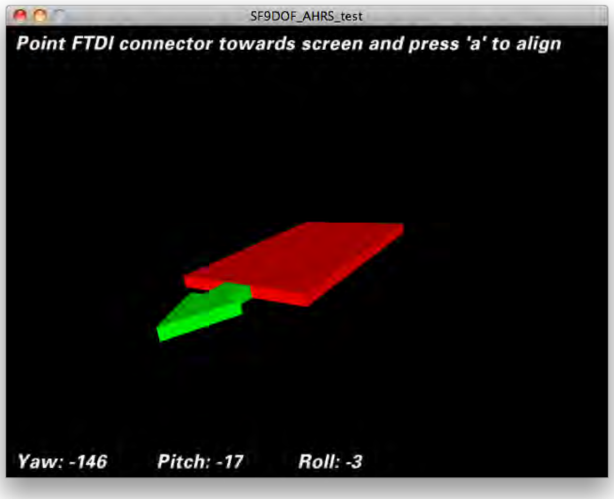


Figure 22: Axis definition

Serial commands that the firmware understands are:

"#o<params>" - Set OUTPUT mode and parameters. The available options are:
// Streaming output
"#o0" - DISABLE continuous streaming output. Also see #f below.
"#o1" - ENABLE continuous streaming output.

// Angles output
"#ob" - Output angles in BINARY format (yaw/pitch/roll as binary float, so one output frame is 3x4 = 12 bytes long).
"#ot" - Output angles in TEXT format ("#YPR=-142.28,-5.38,33.52", followed by carriage return and line feed [\r\n]).

// Sensor calibration
"#oc" - Go to CALIBRATION output mode.
"#on" - When in calibration mode, go on to calibrate NEXT sensor.

// Sensor data output

"#osct" - Output CALIBRATED SENSOR data of all 9 axes in TEXT format.
One frame consist of three lines - one for each sensor: acc, mag, gyr.

"#osrt" - Output RAW SENSOR data of all 9 axes in TEXT format.
One frame consist of three lines - one for each sensor: acc, mag, gyr.

"#osbt" - Output BOTH raw and calibrated SENSOR data of all 9 axes in TEXT format.
One frame consist of six lines - like #osrt and #osct combined (first RAW, then CALIBRATED).

NOTE: This is a lot of number-to-text conversion work for the little 8MHz chip on the Razor boards. In fact it's too much and an output frame rate of 50Hz can not be maintained. #osbb.

"#oscb" - Output CALIBRATED SENSOR data of all 9 axes in BINARY format.
One frame consist of three 3x3 float values = 36 bytes. Order is: acc x/y/z, mag x/y/z, gyr x/y/z.

"#osrb" - Output RAW SENSOR data of all 9 axes in BINARY format.
One frame consist of three 3x3 float values = 36 bytes. Order is: acc x/y/z, mag x/y/z, gyr x/y/z.

"#osbb" - Output BOTH raw and calibrated SENSOR data of all 9 axes in BINARY format.
One frame consist of 2x36 = 72 bytes - like #osrb and #oscb combined (first RAW, then CALIBRATED).

// Error message output

"#oe0" - Disable ERROR message output.

"#oe1" - Enable ERROR message output.

"#f" - Request one output frame - useful when continuous output is disabled and updates are required in larger intervals only. Though #f only requests one reply, replies are still bound to the internal 20ms (50Hz) time raster. So worst case delay that #f can add is 19.99ms.

"#s<xy>" - Request synch token - useful to find out where the frame boundaries are in a continuous binary stream or to see if tracker is present and answering. The tracker will send "#SYNCH<xy>\r\n" in response (so it's possible to read using a readLine() function). x and y are two mandatory but arbitrary bytes that can be used to find out which request the answer belongs to.

(**"#C"** and **"#D"** - Reserved for communication with optional Bluetooth module.)

Newline characters are not required. So you could send **"#ob#o1#s"**, which would set binary output mode, enable continuous streaming output and request a synch token all at once.

The status LED will be on if streaming output is enabled and off otherwise.

Byte order of binary output is little-endian: least significant byte comes first.

3.3.5 Sensor Calibration

Depending on how good or bad your sensors are, precision and responsiveness of Razor AHRS can be improved a lot by calibrating the sensors [19]. If not calibrated you may get effects like:

- drifts in yaw when you apply roll to the board.
- pointing up does not really result in an up attitude.

You have to know that the definition of the axes differs from what is printed on the board. The firmware uses:

- X axis pointing forward (towards the short edge with the connector holes)
- Y axis pointing to the right
- Z axis pointing down

which gives a right-handed coordinate system.

Standard calibration

It might be good to power up the Razor a few minutes before calibration, so the sensors can warm up.

- Open Arduino/Razor_AHRS/Razor_AHRS.pde using Arduino and find the section "USER SETUP AREA" / "SENSOR CALIBRATION". This is where you put the calibration values later.
- Connect the Razor AHRS to your computer, set the correct serial port in Arduino and open the Serial Monitor.
- If you didn’t change the firmware defaults, you should see lots of output like this:

```
#YPR=-155.73,-76.48,-129.51
```

Set the firmware output mode to calibration by sending the string #oc. You should now see output like this:

```
accel x,y,z (min/max) = -5.00/-1.00  25.00/29.00  225.00/232.00
```

Calibrating the accelerometer:

- We'll try to find the minimum and maximum output values for the earth gravitation on each axis. When you move the board, move it real slowly, so the acceleration you apply to it is as small as possible. We only want pure gravity!
- Take the board and point straight down with the x-axis (remember: x-axis = towards the short edge with the connector holes). While you do that, you can see the x-maximum (the second value) getting bigger.
- Hold the board very still and reset the measurement by sending #oc again.
- Now carefully tilt the board a little in every direction until the value does not get bigger any more and write down the x-maximum value.

- Do the same thing for the opposite side (x-axis pointing up) to get the x-minimum: bring into position, send #oc to reset measurement, find x-minimum value and write it down.
- Do the same thing for the z-axis (down and up) and the y-axis (right and left).
  - If you think you messed up the measurement by shaking or moving the board too fast, you can always reset by sending #oc.
- You should now have all the min/max values. Put them into Razor_AHRS.pde.
- NOTE: You have to be really careful when doing this! Even slightly tapping the board with the finger messes up the measurement (try it!) and leads to wrong calibration. Use #oc very often and double check your min/max values)

Calibrating the magnetometer:

- This time you can shake the board as much as you want, but move it away from magnetic distortions introduced by computers and other electronic devices and metal objects.
- We're still calibration mode for the accelerometer. Send #on, which will move calibration to the next sensor, which is the magnetometer.
- NOTE: This section stays here for reference, but you should use the newer "Extended magnetometer calibration (see next section) as it yields much better results! You can skip this and continue with the gyroscope.
- We'll try to find the minimum and maximum output values for the earth magnetic field on each axis. This basically works like calibrating the accelerometer, except the magnetic field of the earth does not point down straight. Depending on where on the planet you currently are, it points north-and-up (southern hemisphere) or north-and-down (northern hemisphere) at a certain angle. This angle is called inclination. Additionally there might be a tiny deviation from true geographic north, which is called declination. See Wikipedia. The following description assumes you're calibrating the magnetometer on the northern hemisphere.
- Hold the board flat like a compass with the x-axis (remember: x-axis = forward, towards the connector holes) pointing north. Then begin to rotate the board around the east-west axis so it starts pointing down. Observe the x-maximum (the second value) in the Serial Monitor and you will notice when you aligned the board's x-axis with the magnetic field of the earth. Stop rotating there and again tilt a little in every direction until the value does not get bigger any more.
- Do the same thing for the opposite side to get the x-minimum: first point north, then down.
- For the magnetometer we don't need to reset with #oc between measurements.
- Do the same thing for the z-axis (up/down) and the y-axis (left/right).
  - NOTE: The rotation of the board around the axis you want to measure doesn't matter, only that it points into the correct direction. E.g when you start measuring the z-axis, it doesn't matter if the x-axis points up or down or left or right.
- You should now have something like this in your Serial Monitor:

```
          magn x,y,z (min/max) = -564.00/656.00  -585.00/635.00  -550.00/564.00
```

Put these values into Razor_AHRS.pde.

Calibrating the gyroscope:

- Lay the Razor still on the table.
- We’re still calibration mode for the magnetometer. Send #on, which will move calibration to the next sensor, which is the gyroscope.
- Wait for 10 seconds, and do not move the Razor. It will collect and average the noise of the gyroscope on all three axes.
- You should now have output that looks like this:

```
○ gyro x,y,z (current/average) = -29.00/-27.98  102.00/100.51  -5.00/-5.85
```

    - If you think you messed up the measurement by shaking or moving the board, you can reset by sending #oc.
- Take the second values of each pair and put them into Razor_AHRS.pde.

3.4 Servo Shield

In order to connect the servos and the 9DOF Razor IMU, we designed a shield that stacks on top of Arduino boards. For the PCB design we exploited Fritzing design program. In **Σφάλμα! Το αρχείο προέλευσης της αναφοράς δεν βρέθηκε.** the designed PCB is illustrated, which was fabricated through Fritzing Fab.

3.4.1 Fritzing Software

Fritzing [20] is an open-source hardware initiative to support designers, artists, researchers and hobbyists to work creatively with interactive electronics. Through the developed program users can create their electronic circuits and then to design their Printed Circuit Board in order to assemble it and create their project.

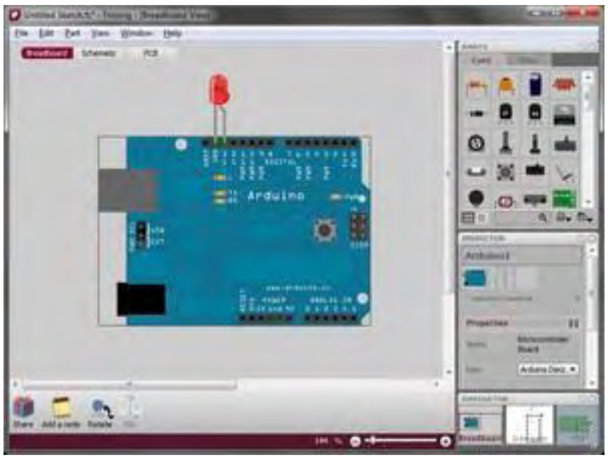
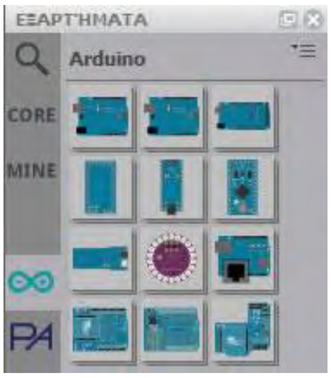
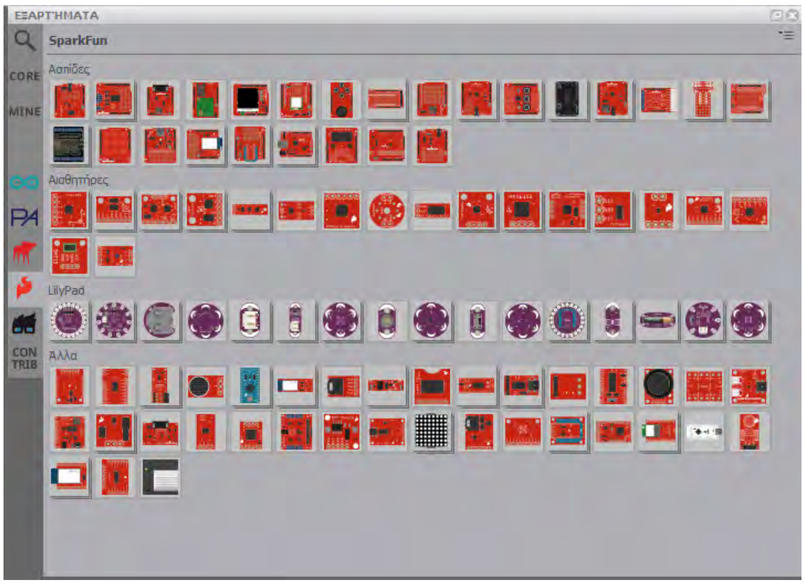


Figure 23: Fritzing Software

Apart from providing common electronics elements, Fritzing software also provides a variety of Arduino compatible boards [21]. Exploiting Fritzing users can add to their circuit usual components such as resistors capacitors transistors but the key novelty of Fritzing is the existing layout of the most Arduino platforms. This feature helps users to easily create their own designs by using the existing formats and pinouts, without having to design from scratch each platform they need. Below are depicted the available components from Sparkfun and Arduino while also the core components (the common).





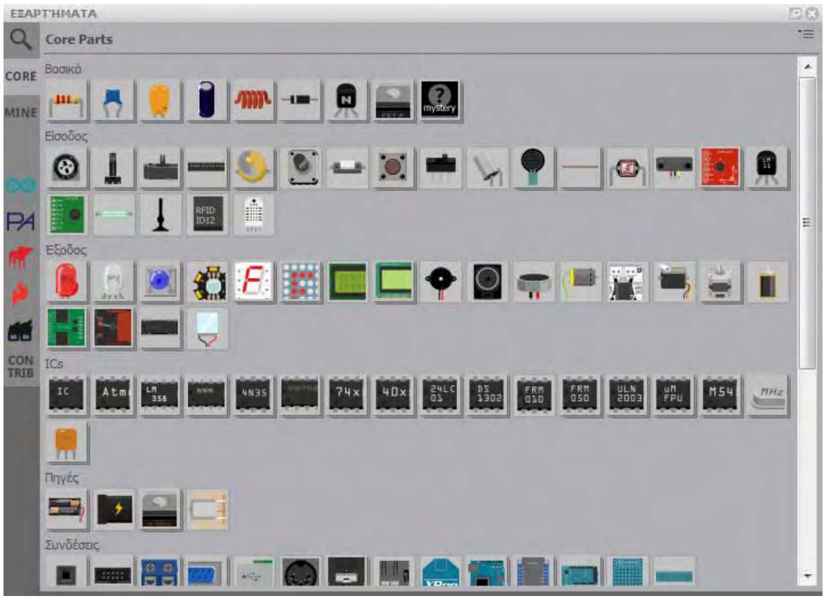
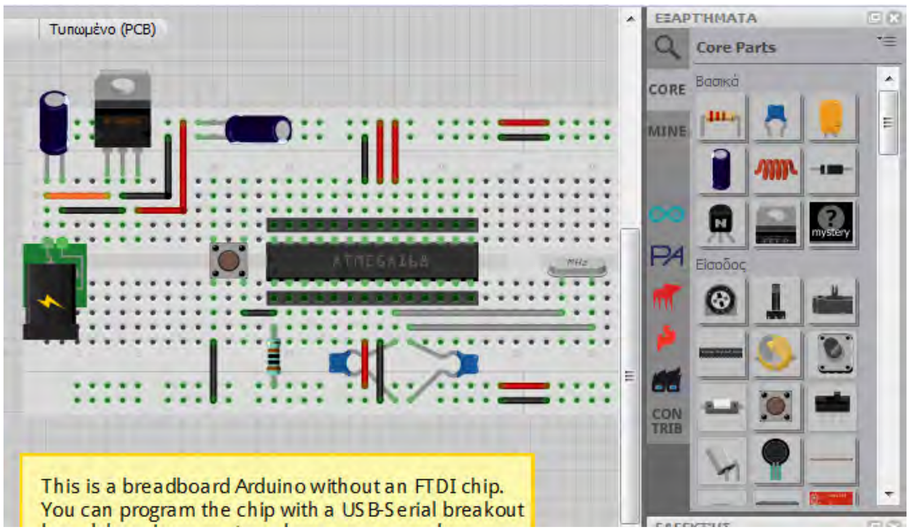


Figure 24: Fritzing Software Tools

Fritzing software features three different tabs:

- **Breadboard**: Users can add to their breadboard every element and connect it wherever, in the same way they do with their real accessories.



This is a breadboard Arduino without an FTDI chip.
You can program the chip with a USB-Serial breakout

Figure 25: Fritzing Breadboard

- **Design**: Users can see the electronic design (the connection among the picked elements)

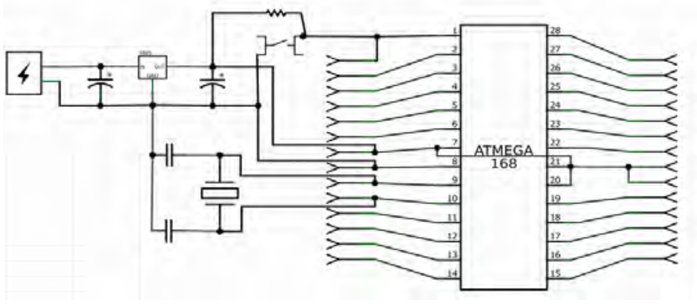


Figure 26: Fritzing Design

- **PCB**: Users can design the route of the wires and the location of each element on the board. After designing the PCB board users can order their boards to be printed through Fritzing Fab [22].

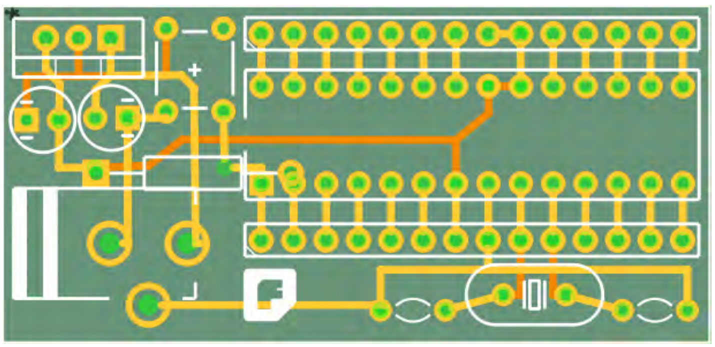


Figure 27: PCB Example

In the context of the present project the PCB for wireless sensors was designed through Fritzing software and was fabricated through Fritzing Fab [22]. Below is a schematic of the PCB as it is shown through Fritzing software at PCB tab.

3.4.2 PCB

In the context of the present project the PCB for attaching the servos steering the directional antennas was designed. Additional feature was added to the PCB to power the servos either from Arduino 5V line or from external power source. Below is a schematic of the PCB as it is shown through Fritzing software at PCB tab.

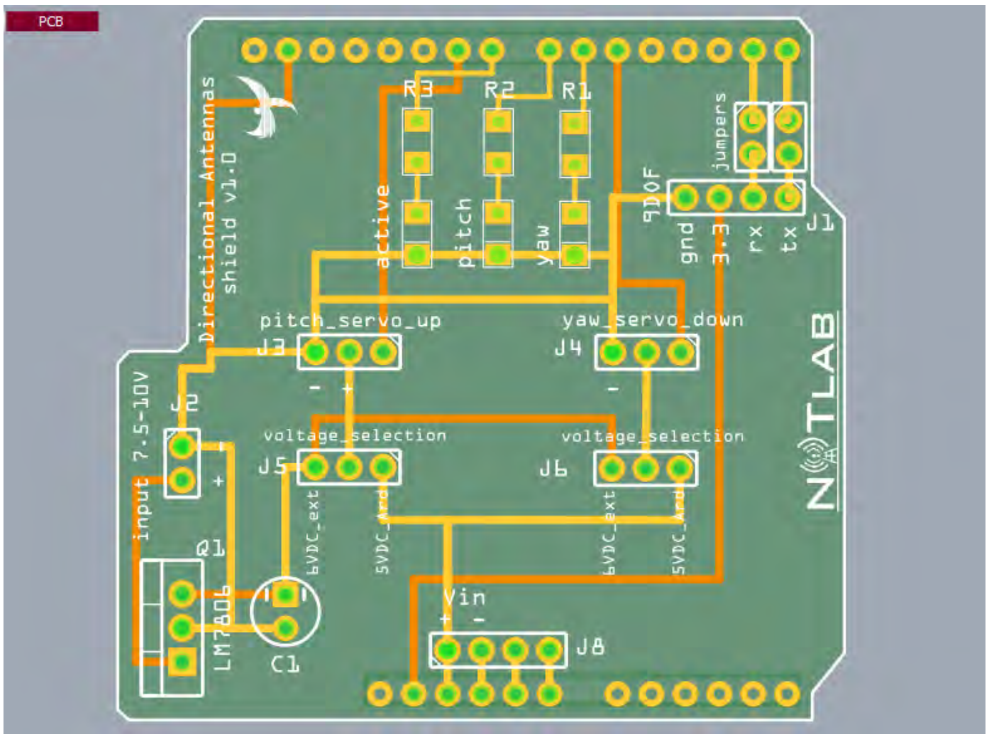


Figure 28: Servo Shield PCB design

3.5 Antenna

3.5.1 Basic Definitions and Antenna Concepts

An antenna gives the wireless system three fundamental properties: gain, direction and polarization. Gain is a measure of increase in power. Gain is the amount of increase in energy that an antenna adds to a radio frequency (RF) signal. Direction is the shape of the transmission pattern. As the gain of a directional antenna increases, the angle of radiation usually decreases. This provides a greater coverage distance, but with a reduced coverage angle. The coverage area or radiation pattern is measured in degrees. These angles are measured in degrees and are called beamwidths [22].

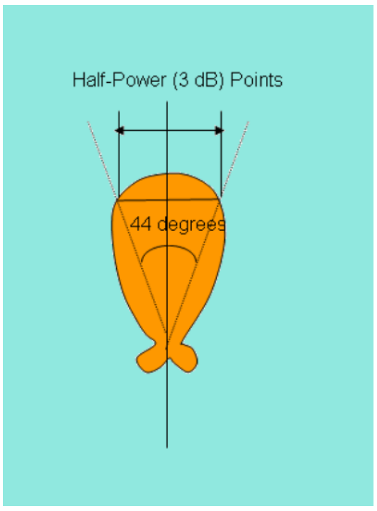


Figure 29: Beamwidth of Antenna

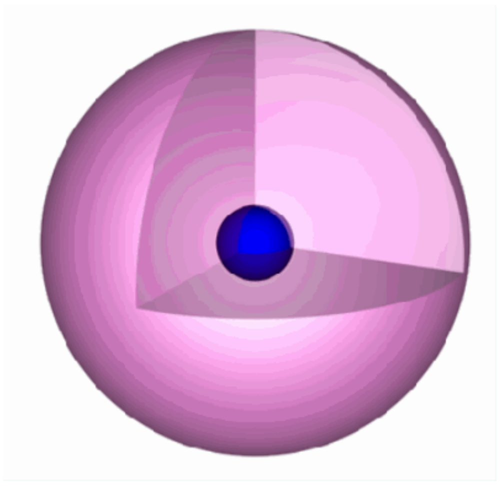


Figure 30: Radiation Pattern of an isotropic Antenna

An antenna is a passive device which does not offer any added power to the signal. Instead, an antenna simply redirects the energy it receives from the transmitter. The redirection of this energy has the effect of providing more energy in one direction, and less energy in all other directions.

Beamwidths are defined in both horizontal and vertical plains. Beamwidth is the angular separation between the half power points (3dB points) in the radiation pattern of the antenna in any plane. Therefore, for an antenna you have horizontal beamwidth and vertical beamwidth.

Antennas are rated in comparison to isotropic or dipole antennas. An isotropic antenna is a theoretical antenna with a uniform three-dimensional radiation pattern (similar to a light bulb with no reflector). In other words, a theoretical isotropic antenna has a perfect 360 degree vertical and horizontal beamwidth or a spherical radiation pattern. It is an ideal antenna which radiates in all directions and has a gain of 1 (0 dB), i.e. zero gain and zero loss. It is used to compare the power level of a given antenna to the theoretical isotropic antenna.

Antennas can be broadly classified as omnidirectional and directional antennas, which depends on the directionality.

Unlike isotropic antennas, dipole antennas are real antennas. The dipole radiation pattern is 360 degrees in the horizontal plane and approximately 75 degrees in the vertical plane (this assumes the dipole antenna is standing vertically) and resembles a donut in shape. Because the beam is slightly concentrated, dipole antennas have a gain over isotropic antennas of 2.14 dB in the horizontal plane. Dipole antennas are said to have a gain of 2.14 dBi, which is in comparison to an isotropic antenna. The higher the gain of the antennas, the smaller the vertical beamwidth is.

Imagine the radiation pattern of an isotropic antenna as a balloon, which extends from the antenna equally in all directions. Now imagine that you press in on the top and bottom of the balloon. This causes the balloon to expand in an outward direction, which covers more area in the horizontal pattern, but reduces the coverage area above and below the antenna. This yields a higher gain, as the

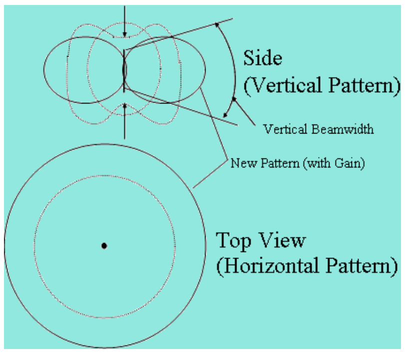


Figure 31: Radiation Pattern of an Omni Antenna

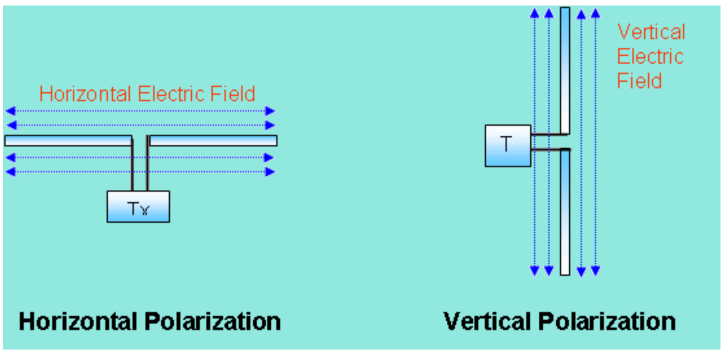


Figure 32: Antenna Polarization

antenna appears to extend to a larger coverage area.

Omnidirectional antennas have a similar radiation pattern. These antennas provide a 360 degree horizontal radiation pattern. These are used when coverage is required in all directions (horizontally) from the antenna with varying degrees of vertical coverage. Polarization is the physical orientation of the element on the antenna that actually emits the RF energy. An omnidirectional antenna, for example, is usually a vertical polarized antenna.

Directional antennas focus the RF energy in a particular direction. As the gain of a directional antenna increases, the coverage distance increases, but the effective coverage angle decreases. For directional antennas, the lobes are pushed in a certain

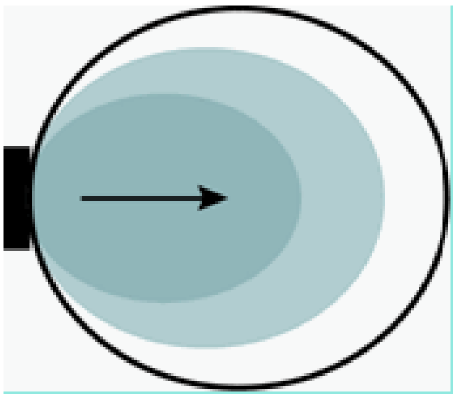


Figure 33: Radiation Pattern of a Directional Antenna

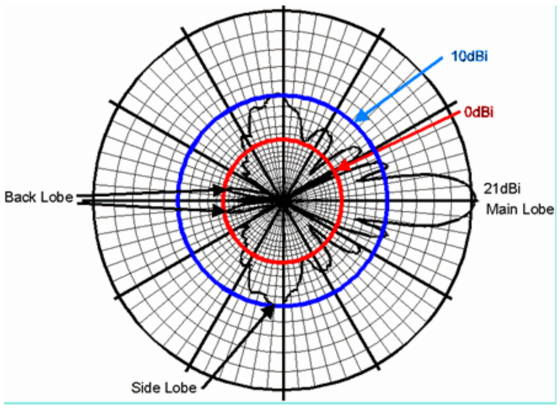


Figure 34: Typical Radiation Pattern of a Directional Antenna with Calibrated Lobes

direction and little energy is there on the back side of the antenna.

Another important aspect of the antenna is the front-to-back ratio. It measures the directivity of the antenna. It is a ratio of energy which antenna is directing in a particular direction, which depends on its radiation pattern to the energy which is left behind the antenna or wasted. The higher the gain of the antenna, the higher the front-to-back ratio is. A good antenna front-to-back ratio is normally 20 dB.

An antenna can have a gain of 21 dBi, a front-to-back ratio of 20 dB or a front-to-side ratio of 15 dB. This means the gain in the backward direction is 1 dBi, and gain off the side is 6 dBi. In order to optimize the overall performance of a wireless LAN, it is important to understand how to maximize radio coverage with the appropriate antenna selection and placement.

3.5.2 Indoor Effects

Wireless propagation can be effected by reflection, refraction or diffraction in a particular environment. Diffraction is the bending of waves around the corners. RF waves can take multipaths between the transmitter and receiver. A multipath is a combination of a primary signal and reflected, refracted or diffracted signal. So on the receiver side, the reflected signals combined with the direct signal can corrupt the signal or increase the amplitude of the signal, which depends on the phases of these signals. Because the distance traveled by the direct signal is shorter than the bounced signal, the time differential causes two signals to be received.

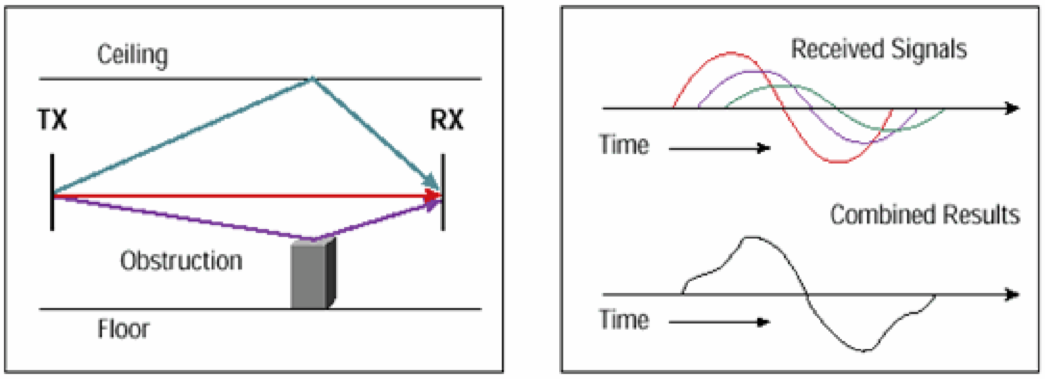


Figure 35: Multipath Effects in Indoor Environment

These signals are overlapped and combined into a single one. In real life, the time between the first received signal and the last echoed signal is called the delay spread. Delay spread is the parameter used to signify multipath. The delay of the reflected signals is measured in nano seconds. The amount of delay spread depends upon the amount of obstacles or infrastructure present between the transmitter and receiver. Therefore, delay spread has more value for the manufacturing floor due to lot of metallic structure present as compared to the home environment. Overall, multipath limits the data rate or lowers the performance.

3.5.3 Omni Antenna Pros and Cons

Omni antennas are very easy to install. Due to the 360 degrees horizontal pattern, it can even be mounted upside down from a ceiling in the indoor environment. Also, because of its shape it is very convenient to attach these antennas to the product. For example, you might see Rubber Duck antennas attached to the wireless APs. In order to obtain an omnidirectional gain from an isotropic antenna, energy lobes are pushed in from the top and the bottom, and forced out in a doughnut type pattern. If you continue to push in on the ends of the balloon (isotropic antenna pattern), a pancake effect with very narrow vertical beamwidth results, but with a large horizontal coverage. This type of antenna design can deliver very long communications distances, but has one drawback which is poor coverage below the antenna.
A low gain omni antenna provides a perfect coverage for an indoor environment. It covers more area near the AP or a wireless device in order to increase the probability of receiving the signal in a multipath environment.

3.5.4 Directional Antennas Pros and Cons

With the directional antennas, you can divert the RF energy in a particular direction to farther distances. Therefore, you can cover long ranges, but the effective beamwidth decreases. This type of antenna is helpful in near LOS coverage, such as covering hallways, long corridors, isle structures with spaces in between, etc. However, as the angular coverage is less, you cannot cover large areas. This is a disadvantage for general indoor coverage because you would like to cover a wider angular area around the AP.

Antenna arrays should face in the direction where the coverage is desired, which can sometimes make mounting a challenge.

Although directional antennas help to focus the energy in a particular direction which can help to overcome fading and multipath, multipath itself reduces the focusing power of a directional antenna. The amount of multipath seen by a user at a long distance from the AP can be much more.

Directional antennas used for the indoors typically have a lower gain, and as a result, have a lower front-to-back and front-to-side lobe ratios. This results in less ability to reject or reduce the interference signals received from directions outside the primary lobe area.

3.5.5 P-LINK ANT2409A Directional Antenna

The antennas used for this framework is the TP-LINK ANT2409A (in Figure 51) which provides gain of9dbi. For the reasons mentioned above and to overcome the fact that the nodes are closely located and that the antennas are designed to offer high gain, we placed fixed attenuators with 5dBm attenuation. The antenna is connected to the rp-sma RF pin at ICARUS nodes. In order to properly mount the antenna to the servomechanisms we removed the enclosure box.

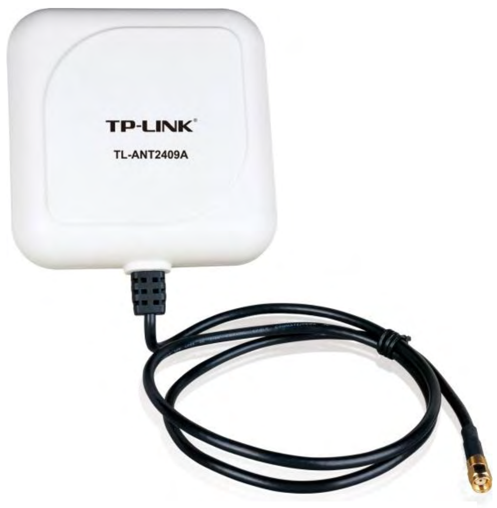


Figure 36: TP-LINK TL-ANT2409A [23]

4. Web Control

The Arduino Ethernet library supports a range of protocols that enable Arduino to be an Internet client or a server. This Arduino configuration acts as a simple web server that serves http requests, changes the values of PWM output ports as requested and sends back xml sensor data. That way, the web server acts as a service and can interact flexibly with the OMF scheduler or with a simple shell script.

4.1 Web server

Initialization

In the stage of initialization the libraries are loaded and specifically the Ethernet, SPI, Servo and SD Library. In case of Arduino Ethernet, SoftwareSerial library is also loaded as the default serial is used for programming the board and for debugging.

Global variables are defined:

- **String RAZOR_feed**: stores the Razor feed
- **String yawstr**: stores the string of the yaw value that is parsed from the GET http request
- **String pitchstr**: stores the string of the pitch
- **Int yawinput**: the integer value of the yaw servo
- **Int pitchinput**: the integer value of the pitch servo
- **Servo yaw**: the servo class for yaw
- **Servo pitch**: the servo class for pitch
- **Byte mac[]**: the mac address of the Arduino web server
- **Byte ip[]**: the IP address of the Arduino web server
- **EthernetServer server(80)**: the class EthernetServer that listens to the http port 80

Setup

This step runs once and setups all the modules:

- Ethernet server startups
- SD Card is initialized
- Serial for debugging is initialized
- Serial2 for razor is initialized, configured to 9600 baudrate and initialization commands are sent to prepare razor
- Servos are initialized

Server loop

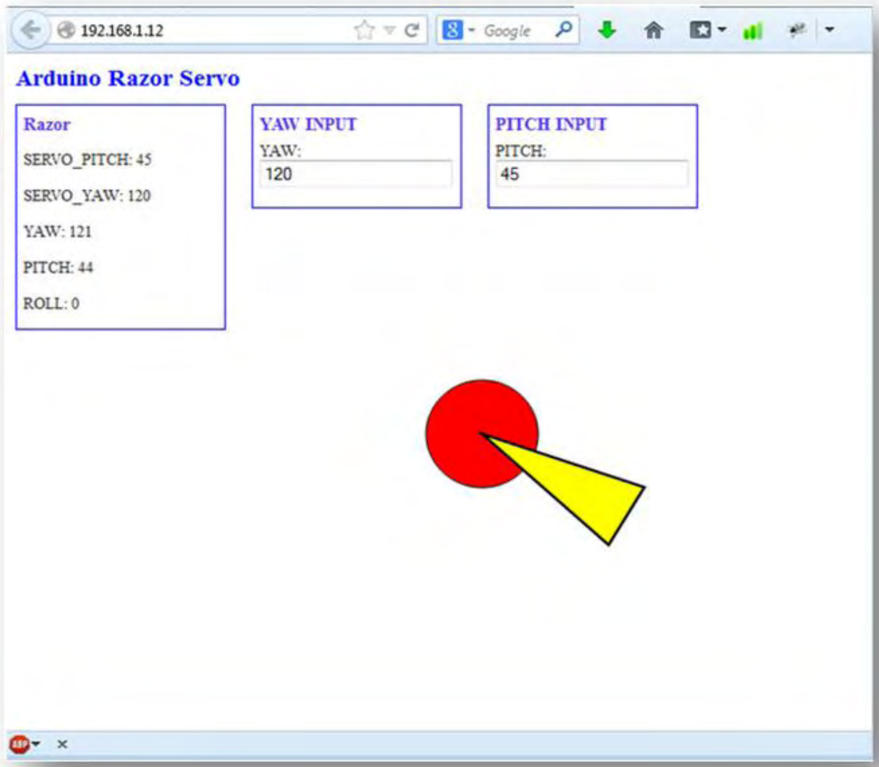
In this stage the server checks if a client exists when the server is available. If there are data available for reading, a persistent connection is established and the http request received is saved and parsed.

In case the attribute “ajax_inputs” exists, server looks for the yaw and pitch values that are between the “Y” - “W” and “W” – “END” respectively.

Immediately, the servos are triggered and take the desired values from the request. A delay is inserted so as to be sure that servo was moved to the ordered position. After that, XML_response is called and a request is sent to Razor (getrazor()). The Razor responds with the actual pitch, yaw and roll angles. The server creates an XML file with the Razor values and is sent to the client. The XML response is also triggered in the case that "Y" does not exist in the http request.
Now, if there is no "ajax_inputs" in the request, the server looks for the web page, loads the web page file "index.htm" that is stored in the SD card and sends it to the client.

4.2 Web server Interface - Browser

The HTML file sent from the web server by calling its IP address to the client browser, is the graphical user interface of the control mechanism. It is written in HTML5 and javascript. When the file is loaded, the client-browser sends silently Ajax HTTP requests every 3 seconds to receive XML data from Servo and Razor's sensors and updates the tables of the site. The user can write the desired angles of the servos to the forms or interact with the graphic that represents the mechanism and an extra 'GET' HTTP request will be sent with the proper pulse width values to



Arduino Razor Servo

Razor
SERVO_PITCH: 45
SERVO_YAW: 120
YAW: 121
PITCH: 44
ROLL: 0

YAW INPUT
YAW:
120

PITCH INPUT
PITCH:
45

Figure 37: Web user interface

Arduino. The proper pulse width nanoseconds are derived from a 3rd degree polynomial function for the yaw servo and from a 2nd degree for the pitch and are computed in the client side with the help of javascript. That way, Arduino server that has limited processing power is not charged with additional computations for the conversion of the desired angles to the corresponding pulse widths.

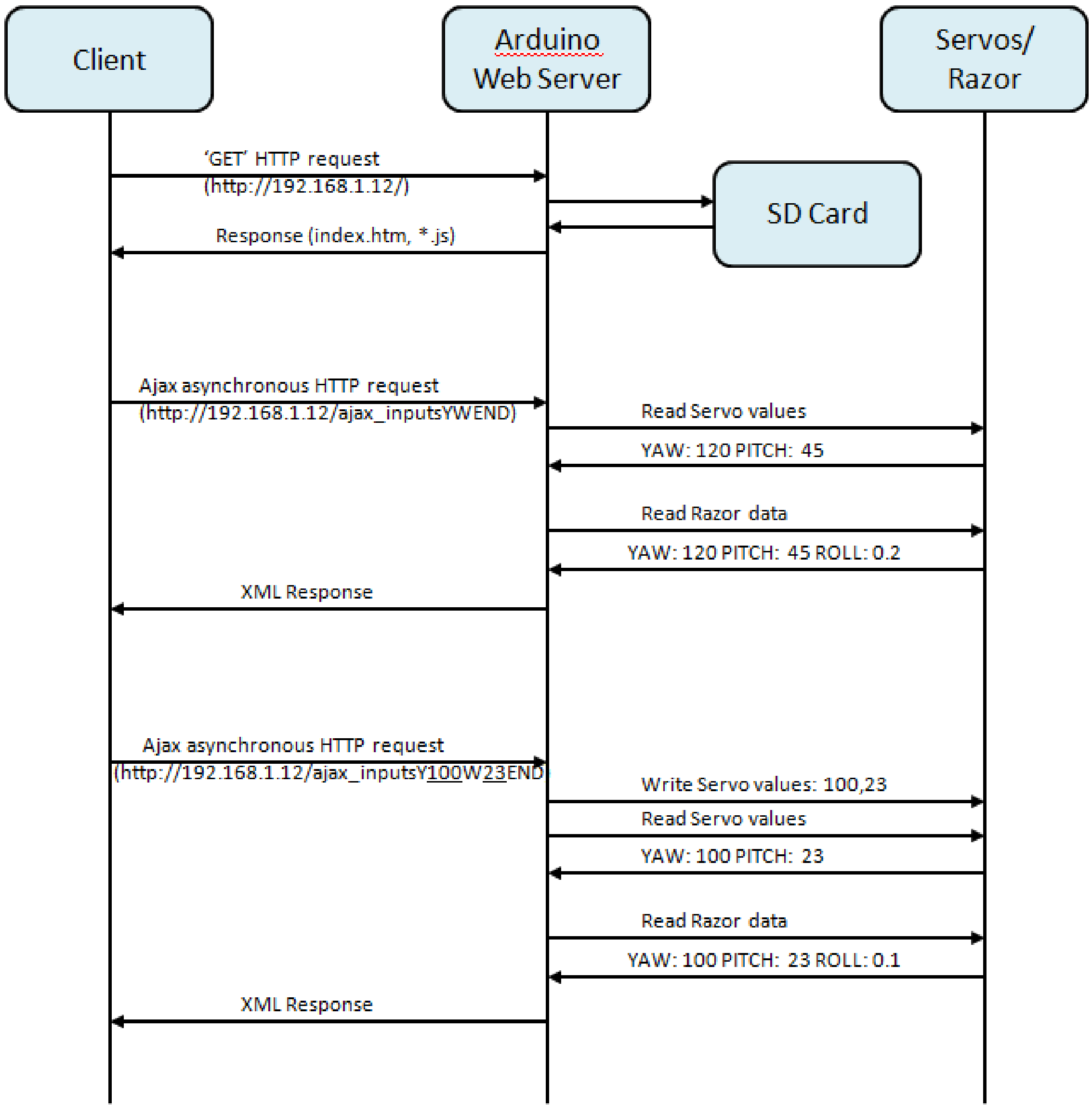

Client
Arduino Web Server
Servos/ Razor
SD Card
'GET' HTTP request (http://192.168.1.12/)
Response (index.htm, *.js)
Ajax asynchronous HTTP request (http://192.168.1.12/ajax_inputsYWEND)
Read Servo values
YAW: 120 PITCH: 45
Read Razor data
YAW: 120 PITCH: 45 ROLL: 0.2
XML Response
Ajax asynchronous HTTP request (http://192.168.1.12/ajax_inputsY100W23END)
Write Servo values: 100,23
Read Servo values
YAW: 100 PITCH: 23
Read Razor data
YAW: 100 PITCH: 23 ROLL: 0.1
XML Response

Figure 38: Example of Arduino Web Server flow

4.3 Control through bash shell script

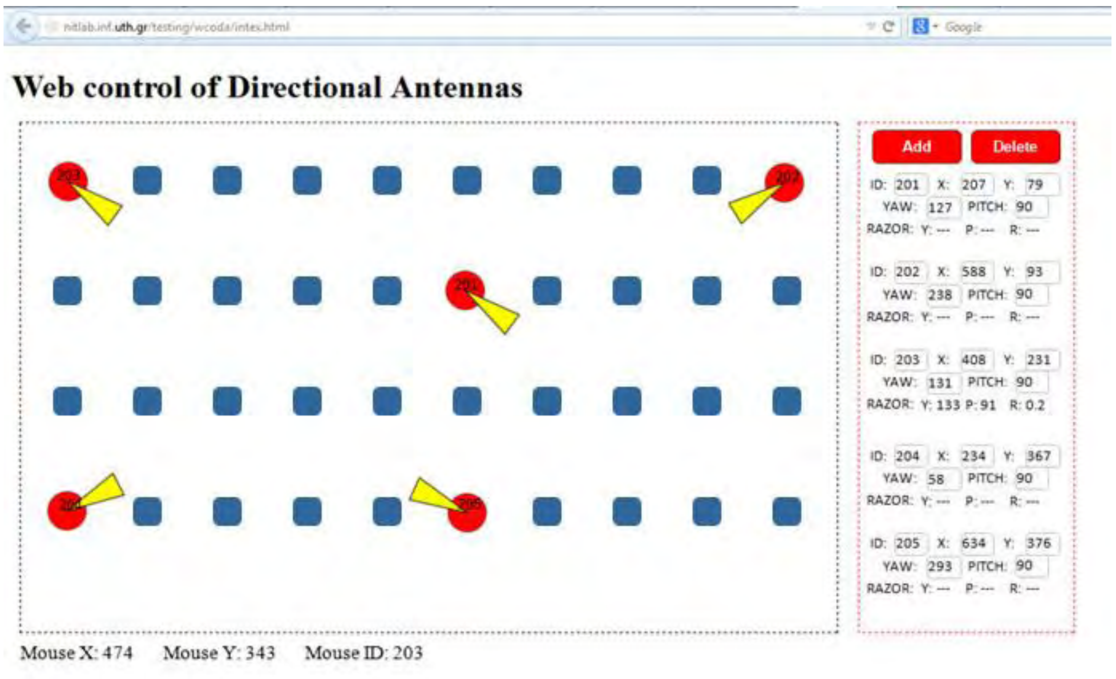
The mechanism can also be controlled by a Linux terminal through a bash shell script with help of the curl command. For the source code see Appendix.

4.4 Control through OMF

UTH developed an OMF Resource Controller (RC) which is able to control and steer the directional antennas. The Resource Controller sends HTTP requests to the web server which runs on each Arduino platform and listens on a specific IP address as described above. That way, a NITOS' user is able to organize an OMF experiment which uses the provided RC. Hence, the user has the capability to steer the directional antennas at the desirable angles, and at specific time slots during the execution of the experiment.

4.5 Central Control of Directional Antennas

An integrated web-based application was developed to centrally control all the directional antennas that are installed at NITOS testbed. Through this application, experimenters can control the five mechanisms that are currently installed at NITOS testbed. The browser sends every 3 seconds Ajax asynchronous HTTP requests to all the Arduino's web servers of the mechanisms and updates the status of the red nodes. If a red node is selected, (figure) one can change its antenna position by dragging its yellow triangle in the desired angle. One can also change the antenna's position by filling the input forms in the red box (figure). Every time the user changes the position of a directional antenna, an HTTP request including the new servo values is generated and sent out to the appropriate Arduino Web server. "Mouse X" and "Mouse Y" field indicate the mouse position in the canvas and Mouse ID indicates the id of the node that is currently selected.



Web control of Directional Antennas

ID: 201 X: 207 Y: 79
YAW: 127 PITCH: 90
RAZOR: Y: --- P: --- R: ---

ID: 202 X: 588 Y: 93
YAW: 238 PITCH: 90
RAZOR: Y: --- P: --- R: ---

ID: 203 X: 408 Y: 231
YAW: 131 PITCH: 90
RAZOR: Y: 133 P: 91 R: 0.2

ID: 204 X: 234 Y: 367
YAW: 58 PITCH: 90
RAZOR: Y: --- P: --- R: ---

ID: 205 X: 634 Y: 376
YAW: 293 PITCH: 90
RAZOR: Y: --- P: --- R: ---

Mouse X: 474 Mouse Y: 343 Mouse ID: 203

Figure 39: Web-based application to control directional antennas of NITOS testbed

5. Experiment

Icarus Nodes

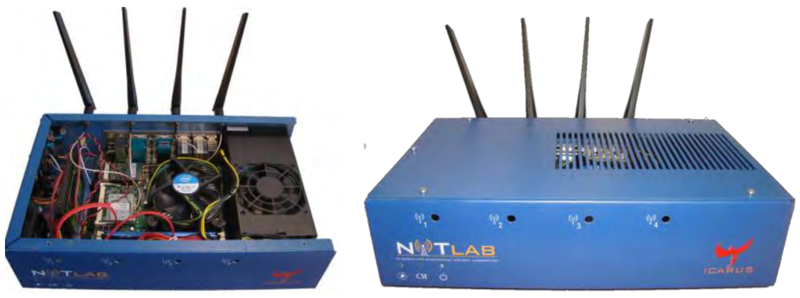


Figure 40: Icarus Node

Icarus have been developed by UTH, are equipped with 802.11a/b/g and 802.11a/b/g/n wireless interfaces and feature new generation Intel 4-core CPUs and new generation solid state drives. More details about their specification can be found in the following table.

Table: Icarus nodes specifications

Motherboard	Features 2 Gigabit network interfaces and supports 2 wireless interfaces
CPU	Intel Core i7-2600 Processor, 8M Cache at 3.40Ghz
RAM	4G HYPERX BLU DDR3
Wireless interfaces	Atheros 802/11 a/b/g & Atheros 802.11 a/b/g/n (MIMO)
Chassis Manager Card	UTH's CM card
Storage	Solid state drive
Power supply	350 Watt mini-ATX
Antennas	Multi-band 5dbi, operates both on 2,4Ghz & 5Ghz
Pigtails	High quality pigtails (UFL to RP-SMA)

Description

An Icarus wifi node was used and acted as a client. It featured a wireless interface which was connected with the directional antenna through a 5db attenuator. A laptop , far located, equipped with a wireless card acted as a server.

The experiment that was performed generates traffic load using the iperf command in order to compare the achieved throughput in different antenna's directions. UDP packets were sent at a rate of 54Mbps for 60 seconds.

Iperf is a linux tool to measure the bandwidth and the quality of a network link. The following commands were executed:

```
ICARUS:  iperf -c 192.168.2.1 -u -b 54M -t 60 -i 1
```

```
LAPTOP: iperf -s 192.168.2.2 -u -t 60 -i 1
```

The indoor environment that the experiment took place was not the appropriate place because of reflection and deffraction, but by lowering the gain with attenuators, we succeeded better throughput while focusing the antenna to the direction of the client-laptop.

6. Future proposed improvements/developments

Some further improvements are proposed so as the mechanism and its API would be robust and operational through time and helpful for research experiments:

1. The XML response could be enriched with time stamps in order to know the exact time that the directional antenna has moved position
2. An Arduino configuration could be developed and embedded in the existing one so that Razor sensors could be calibrated automatically by moving the servos and sending proper commands from the core Arduino board to Razor board. This is essential because the sensors need calibration from time to time and sensors' results should be accurate. The reverse procedure, calibrating the servos by using Razor sensors should be also added to configuration. Auto calibrations should be done when the mechanism is not used for experiments.
3. More accurate and high torque servos should be selected to lift up easily the antenna

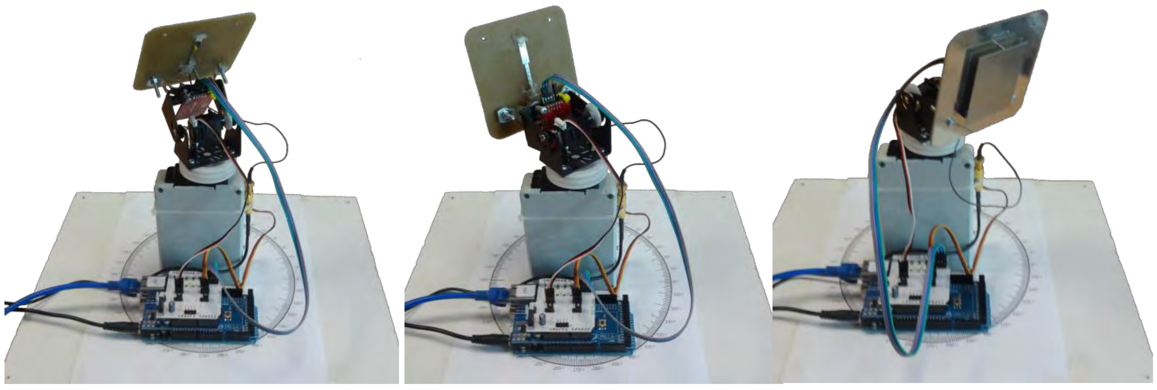


Figure 55: Directional Antennas mechanism, rotated in different directions

7. References

[1] NITOS: OpenLab – Seventh Framework Programme – Deliverable D3.3 Provision of wireless testbeds, Network Lab, UTH

[2] Arduino: http://en.wikipedia.org/wiki/Arduino

[3] Arduino Mega 2560: http://arduino.cc/en/Main/arduinoBoardMega2560

[4] Arduino Ethernet Shield: http://arduino.cc/en/Main/ArduinoEthernetShield

[5] Arduino Ethernet Board: http://arduino.cc/en/Main/ArduinoBoardEthernet

[6] FTDI Basic Breakout – 5V: https://www.sparkfun.com/products/9716

[7] Arduino Libraries: http://arduino.cc/en/Reference/Libraries

[8] Ethernet Library: http://arduino.cc/en/Reference/Ethernet

[9] SD Library: http://arduino.cc/en/Reference/SD

[10] SoftwareSerial Library: http://arduino.cc/en/Reference/SoftwareSerial

[11] Servo Library: http://arduino.cc/en/Reference/Servo

[12] Servo: Arduino Cookbook 2nd edition O'REILLY, Michael Margolis

[13] GWS125 -1T Specifications: http://www.pololu.com/catalog/product/522/specs

[14] Sub-micro servo's specifications: https://www.sparkfun.com/products/9065

[15] Razor IMU: https://www.sparkfun.com/products/10736

[16] Triple-Axis Gyroscope – ITG3200: https://www.sparkfun.com/products/9793

[17] Triple-Axis Accelerometer – ARDL345:https://www.sparkfun.com/products/9045

[18] Magnetometer HMC5883L: https://www.sparkfun.com/products/10494

[19] Razor Firmware & Calibration: https://github.com/ptrbrtz/razor-9dof-ahrs/wiki/Tutorial

[20] Fritzing Software : http://fritzing.org/

[21] Fritzing PCB: Development of a wireless sensor platform, Master Thesis Report, Ioannis Kazdaridis, UTH

[22] Fritzing Fab : http://fab.fritzing.org/fritzing-fab

[23] Antennas: http://www.cisco.com/en/US/tech/tk722/tk809/technologies_tech_note09186a00807f34d3.shtml

[24] TP-LINK ANT2409A: http://www.tp-link.com/en/products/details/?model=TL-ANT2409A

APPENDIX

Curlcontrol.sh

….."Curlcontrol.sh 192.168.1.12 236 45"…..

```
#!/bin/bash
if [ -z $1 ]; then
        IPDUINO="192.168.1.12"
        YAW="0"
        PITCH="45"
elif [ -z $2 ] | [ -z $3 ]; then
        IPDUINO=$1
        YAW="0"
        PITCH="45"
else
        IPDUINO=$1
        YAW=$2
        PITCH=$3
fi

YAW_SERVO=$(echo "0.393288011027541*$YAW - 0.179436042587114" | bc -l
| xargs printf "%1.0f")
if [ $YAW_SERVO -le "0" ]; then
        YAW_SERVO="0"
fi
echo $YAW_SERVO
PITCH_SERVO=$(echo "0.01445203370824 * ($PITCH * $PITCH) +
9.071565715837409*$PITCH + 859.7024322029524" | bc -l | xargs printf
"%1.0f")
echo $PITCH_SERVO

if [ -z $2 ] | [ -z $3 ]; then
        curl $1/ajax_inputs
else
        echo | curl
$1/ajax_inputsY${YAW_SERVO}W${PITCH_SERVO}END&nocache=
        echo -ne '\n'
fi
```

Index.htm

```
<!DOCTYPE html>
<html>
    <head>
        <title>Arduino Razor Servo</title>
        <script>
            strYAW = "";
            var YAW_state = 0;
            strPITCH = "";
            var PITCH_state = 0;
            function GetArduinoIO()
            {
                  nocache = "&nocache=" + Math.random() * 1000000;
                  var request = new XMLHttpRequest();
                  request.onreadystatechange = function()
                  {
                        if (this.readyState == 4) {
                              if (this.status == 200) {
                                    if (this.responseXML != null) {
                                          // XML file received -
contains analog values, switch values and LED states
                                          var count;
                                          // get analog inputs
                                          var num_an =
this.responseXML.getElementsByTagName('analog').length;
                                          for (count = 0; count <
num_an; count++) {

      document.getElementsByClassName("analog")[count].innerHTML =

      this.responseXML.getElementsByTagName('analog')[count].childNod
es[0].nodeValue;
                                          }

                                    }
                              }
                        }
                  }
                  // send HTTP GET request with LEDs to switch on/off
if any
                  request.open("GET", "ajax_inputs" + strYAW +
strPITCH + "END" + nocache, true);
                  request.send(null);
                  setTimeout('GetArduinoIO()', 3000);
                  strYAW = "";
                  strPITCH = "";
            }
            // service LEDs when checkbox checked/unchecked
            function GetCheck()
            {
                  var PITCH_val = PITCH_form.PITCH.value;
                  strPITCH = "W"+PITCH_val;
                  var YAW_val = YAW_form.YAW.value;
                  strYAW = "Y"+YAW_val;
            }

      </script>
```

```
        <style>
              .IO_box {
                    float: left;
                    margin: 0 20px 20px 0;
                    border: 1px solid blue;
                    padding: 0 5px 0 5px;
                    width: 120px;
              }
              h1 {
                    font-size: 120%;
                    color: blue;
                    margin: 0 0 10px 0;
              }
              h2 {
                    font-size: 85%;
                    color: #5734E6;
                    margin: 5px 0 5px 0;
              }
              p, form, button {
                    font-size: 80%;
                    color: #252525;
              }
              .small_text {
                    font-size: 70%;
                    color: #737373;
              }
        </style>
      </head>
      <body onload="GetArduinoIO()">
          <h1>Arduino Razor Servo</h1>
          <div class="IO_box">
                    <h2>Razor</h2>
                    <p>SERVO_PITCH: <span class="analog">...</span></p>
                    <p>SERVO_YAW: <span class="analog">...</span></p>
                    <p>YAW: <span class="analog">...</span></p>
                    <p>PITCH: <span class="analog">...</span></p>
                    <p>ROLL: <span class="analog">...</span></p>
              </div>
              <div class="IO_box">
                    <h2>YAW INPUT</h2>
                    <form id="YAW_ID" name="YAW_form">
                          YAW: <input type="text" name="YAW"
onclick="GetCheck()" />  <br /><br />
                    </form>
              </div>
              <div class="IO_box">
                    <h2>PITCH INPUT</h2>
                    <form id="PITCH_ID" name="PITCH_form">
                          PITCH: <input type="text" name="PITCH"
onclick="GetCheck()" />  <br /><br />
                    </form>
              </div>
      </body>
</html>
```