جامعة أبوظبي
ABU DHABI UNIVERSITY

Bachelor's Thesis

Hand Gesture Controlled Emergency Aerial Assistance Using Smartphone Based 4G Quadcopter

Authors:
Muhammad Obaidullah
Sifat Sultan

Supervisor:
Dr. Mohammed Assad Ghazal

A thesis submitted in fulfillment of the requirements for the degree of Bachelors of Science in Electrical Engineering

in

Department of Electrical & Computer Engineering
College of Engineering

September 22, 2014

بِسْمِ ٱللَّٰهِ ٱلرَّحْمَٰنِ ٱلرَّحِيمِ

In The Name of Allah, The Most Beneficent, The Most Merciful.

Declaration of Authorship

We, Muhammad Obaidullah & Sifat Sultan, declare that this thesis titled, *'Hand Gesture Controlled Emergency Aerial Assistance Using Smartphone Based 4G Quadcopter'* and the work presented in it are our own. We confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Muhammad Obaidullah	***Sifat Sultan***
Signed:	Signed:
Date:	Date:

"Attainment of knowledge is a must for every Muslim"

Holy Prophet of Allah Muhammad (S.A.W.)

"Stay hungry, stay foolish"

Steve Jobs
(1955 - 2011)

"The real asset of any advanced nation is its people, especially the educated ones, and the prosperity and success of the people are measured by the standard of their education."

Sheikh Zayed bin Sultan Al Nahyan
(1918 - 2004)

ABU DHABI UNIVERSITY

Abstract

Faculty of Electrical & Computer Engineering
Department of Electrical & Computer Engineering

Bachelor of Science Electrical Engineering

Hand Gesture Controlled Emergency Aerial Assistance Using Smartphone Based 4G Quadcopter

by Muhammad OBAIDULLAH
Sifat SULTAN

This project aims to solve the problem of providing emergency medical or any other type of assistance in a un-accessible area by using a 4G quadcopter which is controlled by hand gestures. The hand gestures provided by the user will be used to maneuver the quadcopter to reach the emergency location and provide assistance. The uniqueness of this project lies in the fact that it uses a 4G smartphone to connect the quadcopter to the base station for getting the user commands, providing the GPS location, and live video stream of the quadcopter's vision.

Acknowledgements

We would like to thank first of all Allah (S.W.T.) for everything. Secondly our parents who were there at all times for us when we needed them throughout this project. This project would not be possible if our parents did not encourage and support us when we were at our lowest and near the brink of giving up.

We would like to acknowledge various faculty members of Abu Dhabi University. Our supervisor and adviser, Dr. Mohammed Assad Ghazal, gave us a chance to realize our idea and our vision. His continuous efforts and chain of ideas proved to be priceless in many aspects of this project. We would also like to acknowledge Eng. Ibrahim for priceless hardware solutions he provided regarding PCB and ESC programming. Eng. Ahmed Sweleh for cooperation, firmware programming and many PCB solutions. Eng. Nasir for the idea of wooden protection box and quadcopter weight calculations. Also Dr. Hamdi Sheibani for allowing us to use the mechanical lab various times.

Last but not the least, our dear friends and brother Abid Abdul Aziz, and Shihab Sultan for their groundbreaking mechanical skills and passion. Bilal Ahmad and Mohammed Ali Sadi for helping in different aspects of project. Bilal Arshad for strategical tackling and negotiating with various people. For continuous motivation and encouragement we would like to acknowledge Owais Ali.

Contents

List of Figures

List of Tables

Abbreviations

ESC	**E**lectronic **S**peed **C**ontroller
PWM	**P**ulse **W**idth **M**odulation
LDR	**L**ight **D**ependent **R**esistor
OP AMP	**O**perational **A**mplifier
IR	**I**nfra **R**ed
RPM	**R**evolutions **P**er **M**inute
PROP	**P**ropeller
PCB	**P**rinted **C**ircuit **B**oard
CCTV	**C**losed **C**ircuit **T**ele**v**ison
PAN	**P**ermanent **A**ccount **N**umber
IEEE	**I**nstitute (of) **E**lectrical (&) **E**lectronics **E**ngineers
GND	**G**round
AIL	**A**ileron
ELE	**E**levation
THR	**T**hrust
RUD	**R**udder
POT	**P**otentiometer
CHAR	**C**haracter
IP	**I**nternet **P**rotocol
WAN	**W**ide **A**rea **N**etwork

Symbols

a	distance	m
V	Volts	W (V)
C	Current	W (A)
bps	Bits Per Second	bps
g	Gram	g
Kg	Kilogram	Kg
ms	Millisecond	ms
$bits/packet$	Bits Per Packet	bits/packet
P	power	W (Js^{-1})
ω	angular frequency	rads^{-1}

Dedicated to our Mothers

Chapter 1

Introduction

1.1 Project Background

The number of researches and developments on the unmanned aerial vehicles have increased over the past years. Many hobbyists and robotics enthusiasts are particularly interested in quadcopters because of their ability to hover in place, take-off and land vertically. [4] People can easily now buy and assemble a quadcopter with low cost. Within seconds a quadcopter can lift-off reach a point and land back. This ultra fast response and ease of control make the quadcopter ideal for reaching areas dangerous for humans and performing tasks.

Parallel to this, the field of Human-Computer Interaction has been a growing field in the past few years due to the introduction of innovative interaction devices such as Nintendo Wii Remote and Microsoft Kinect. The idea of using gestures as a mean of interaction has become the recent trend as it makes the experience more natural and hence the possibility of application.

Quadcopter: The term quadcopter or quadrotor comes from the four narrow chord propellers used by the air vehicle in order to maneuver in air. The quadcopter uses these four propellers to push the air downwards in order to generate lift force. It maneuvers by varying the speeds of these four motors.

Leap Motion: In 2012, the Leap Motion controller was introduced. Leap motion comes with a 150 degree field of view and makes use of a depth sensor to follow and scan the hand features up to 1/100th of millimeter. This extremely precise control gives us the opportunity to use it as our interface over Kinect which tracks body instead of hand since our gestures will be mainly finger movements.

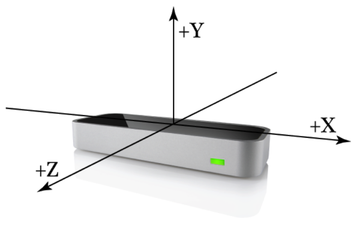


Figure 1.1: A leap motion device with 3D coordinates.

The past few years there have been extensive research on autonomous micro-aerial vehicles. The application ranges from hobby to serious circumstances.

Quad Copter Application XPROHELI developed quad copter that is used to take high quality video and photography. http://www.youtube.com/watch?v=HoQQlRzybmA#t=14 This video was taken using the XPROHELI quad copter.

FAE developed tri copter that is used to take long range aerial surveillance over oil field. http://www.youtube.com/watch?v=a5H5paygWQw&feature=youtu.be Quad-copter has the potential to be used in a variety of application and we decided to work on a case where quad copter deserves to be used; to save human lives where time is of paramount.

1.2 Problem Discussion

In this stone hearted world, everyday accident happen and people die. These lives can be saved by timely medical response. But the response faces countless obstacles sometimes like traffic jams, remote places, radiation etc. This is a growing problem causing engineers and specialists to think and come up with different solutions.

UAE itself has a mortality rate of about 37 per 100,000 which is one of the highest in the world. [1] This mortality rate can be further skimmed down by providing timely first aid kit to seriously injured people.

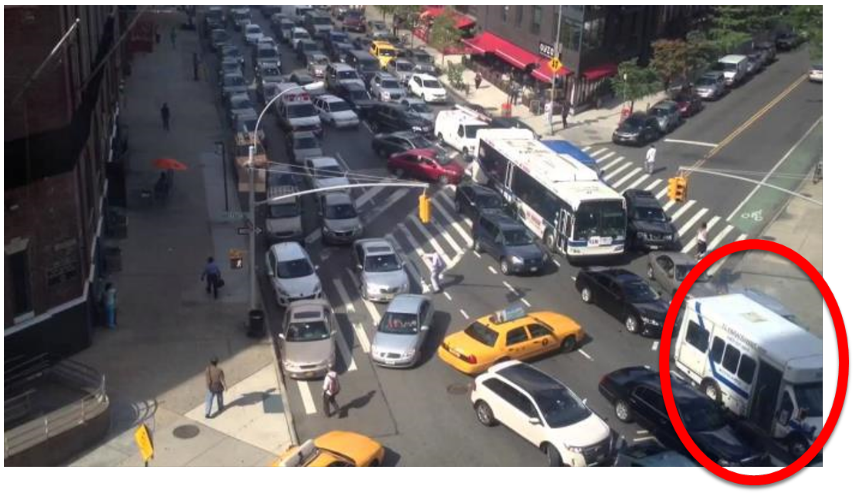


FIGURE 1.2: Lives are lost as fast-aids do not reach on time.

So all in all, our problem statement is how to provide quick and timely emergency assistance without the restriction of the range by using low-cost quadcopter.

Chapter 2

Design

2.1 Project Description

We proposed a system which is complete package and will solve all the problems of providing quick emergency assistance. The proposed system was widely accepted by the faculty members and approved.

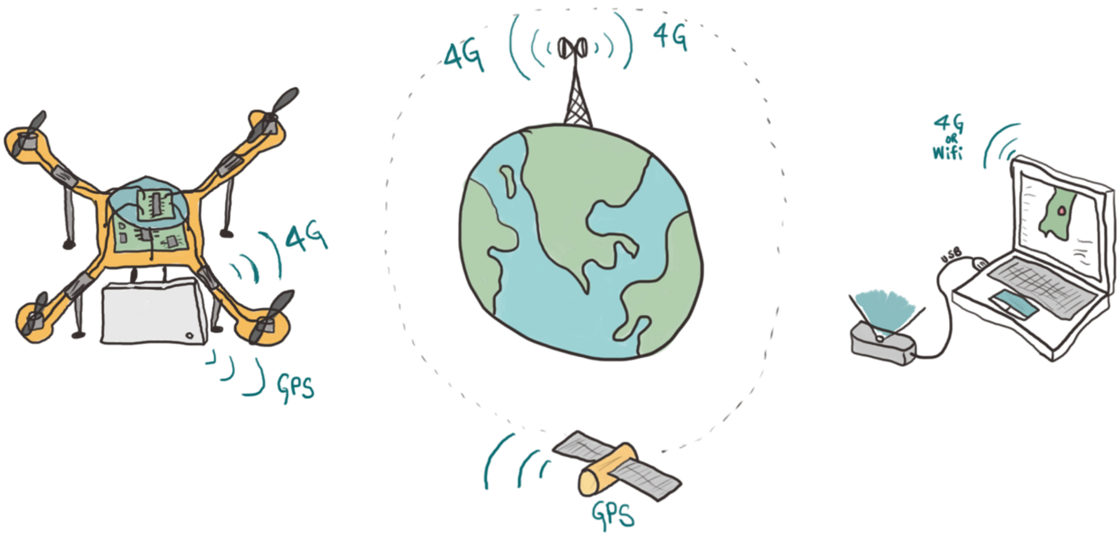


FIGURE 2.1: The system diagram of the project.

Our system has a Quadcopter with a gyro control chip which controls the basic maneuvers for the UAV and we propose to build a custom designed chip to decode the signals coming from the on-board android smartphone and send commands to the controller chip. We used a android smartphone to provide the quadcopter with GPS, Camera, and 4G internet connection. So this quadcopter is now capable of performing completely unmanned flight and be controlled using http requests coming to it through 4G connection. The quadcopter also provides a http feedback to the computer about its GPS position. The GPS can also be used to make a flight plan for the quadcopter. The on-board mobile also provides a UDP live stream of the camera vision and can be controlled using.

Lets take a typical emergency assistance scenario and apply our proposed solution to it. A call comes to the police about a car accident that has happened on a highway in Abu Dhabi. This is emergency case it might take several hours for the police to arrive and analyze the situation. The police places a pin on the map in the quadcopter software in computer. The computer sends the http request to the quadcopter to move to that specific location. The quadcopter immediately starts up and goes to the GPS location. This is done with a feedback loop which calculates the current GPS location and the required GPS location. While all of this is done, the policeman can see the current location of the quadcopter on the map. When the quadcopter reaches the end of its destination, the policeman turns on the gesture control mode and starts controlling the movement of the quadcopter precisely using hand. Now the policeman can access the situation and can also possibly drop the first aid kit if the situation is too serious.

2.1.1 Using a UAV for agility and accessibility

Unmanned Air Vehicles are light, fast and can be designed to carry the essential aid equipment. Along with that unmanned vehicles can be used to perform critical operations in areas where human life cannot be risked. Furthermore, quick response to a distress call can differentiate between life and death of a person. Therefore, air emergency medical assistance is needed very quickly in some cases and the delay cased by set-up and start of manned vehicles cannot be afforded at this point.

2.1.2 Use of a quadcopter for ease of control and high payload delivery

With the use of four propellers pushing the air downwards with immense speed, a typical quadcopter is capable of lifting high payloads. UAVs for search and rescue operations are required to be small and at the same time carry high payloads. Quadcopter is perfect for such kinds of ease of control applications because of its usage of four high rpm brushless motor run propellers. These brushless motor speeds can be intelligently varied to provide 6 degrees of freedom by sensing the gyroscopic values of Pitch, Yaw, and roll which.

The quadcopter can be controlled using two types of configurations, one is the X configuration and the other is the + configuration. For a very stable flight and camera view being not blocked, we propose to use a X configuration quadcopter.

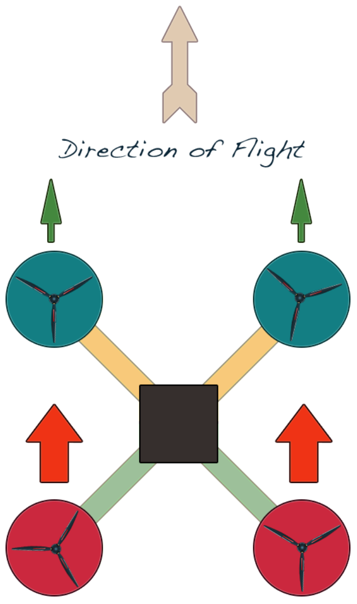


FIGURE 2.2: X configuration forward motion of quadcopter.

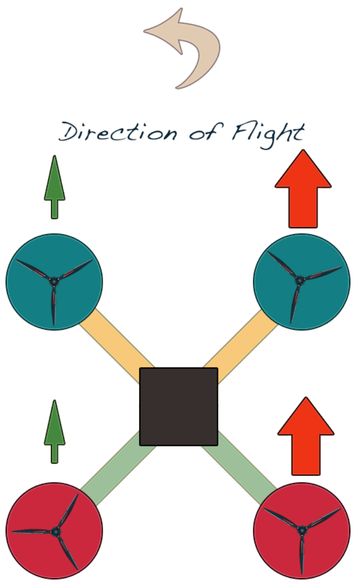


FIGURE 2.3: Left motion of quadcopter in X configuration

Forward Direction: By speeding up the back motors, a forward direction motion can be accomplished. This makes the thrust of the motors at an angle to the z-axis. The thrust has force components in the weight direction to balance or cancel the weight and a unbalanced forward component to make the quadcopter go forward.

Left Direction: By speeding up the right motors, a left direction motion can be accomplished. The thrust has force components in the weight direction to balance or cancel the weight and a unbalanced forward component to make the quadcopter go left.

Right Direction: By speeding up the left motors, a right direction motion can be accomplished. The thrust has force components in the weight direction to balance or cancel the weight and a unbalanced forward component to make the quadcopter go right.

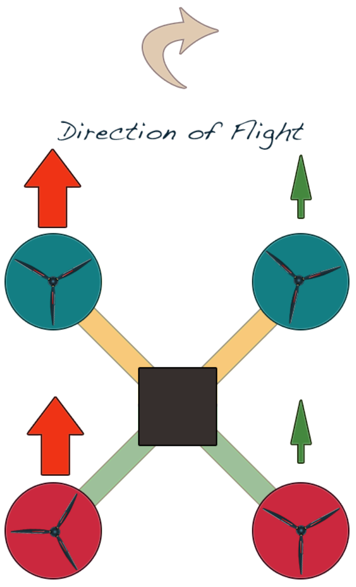


FIGURE 2.4: X configuration forward motion of quadcopter.

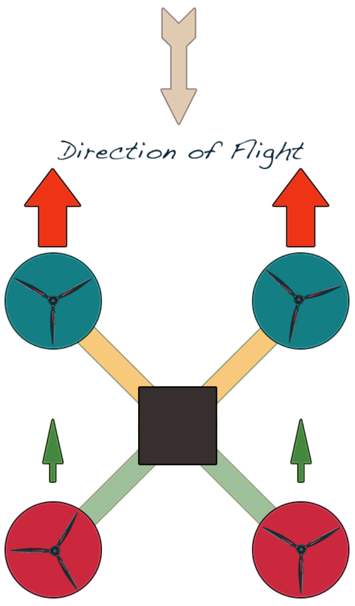


FIGURE 2.5: Left motion of quadcopter in X configuration

Backward Direction: By speeding up the front motors, a backward direction motion can be accomplished. This makes the thrust of the motors at an angle to the z-axis. The thrust has force components in the weight direction to balance or cancel the weight and a unbalanced backward component to make the quadcopter go forward.

2.1.3 Global control by using on-board 4G smartphone

Usually quadcopters have a limited range radio link but emergency assistance should not have limited range of control. It was decided to use 4G network for control and communication with the quadcopter as 4G network is accessible from within 35Km range of a cellular tower. Furthermore, a typical city is divided into cells and at almost each node a telecommunication tower is located. This is a huge advantage for our quadcopter since it can have a connection to base control all around the city where the 3G radiation pattern is in range. A smartphone will be mounted on the quadcopter for providing a 4G internet connection. The internet connection on the quadcopter can be used for several things such as providing the GPS location, video stream, and controlling the quadcopter maneuvers.

2.1.4 Live streaming video vision by using UDP

Video streaming of quadcopter's vision is possible by using the on-board android smartphone. An app it to be designed in such a way that it streams video back to the base station by use of UDP. This is possible through 4G connectivity since use of WiFi for such long range is not advisable.

p A video stream is acceptable if there is any fault of packages lost in the way. It is because if during the video streaming the packages get lost due to the router being overloaded for instance then the video streaming can still be resumed with a reduced quality.

The TCP/IP protocol would force the video stream to wait until the dropped packages are found to resume processing newer packages. The point that is usually misunderstood is that when few packages are lost it doesn't mean a whole frame is lost but rather few pixels from a certain frame are lost and hence waiting for these lost pixels to come back is a waste since the video stream has already moved on to new frames. However TCP could have been a good option if the case was about recorded video streaming since when a package is lost the system pauses the frame until the all the data for the next frame has been collected in the buffer giving the term 'buffering'.

Therefore, we decided to use UDP for our live video streaming.

2.1.5 Smartphone to Controller Board USB Connection

The receiver in our schematics is going to be a smartphone which will be using 4G internet to receive the data. The smartphone onboard the quadcopter will act as client while the PC with the user will act as the server to send the controls. The data will then be passed to an Atmega328 chip using USB-Serial FTDI chip that will in turn send the flight data to controller board. The application will send byte array through serial to Atmega328 Chip that will contain the flight values such as aileron value to control the roll, elevation value to control the elevation angle with horizontal, rudder value to control the yaw movement, and thrust value to control lift force. [**?**]

2.2 Hand Gesture Models

In order to control the quad-copter using a leap motion device, a model of the hand gestures should be made in order to approximate the finger positions in three dimensional space.

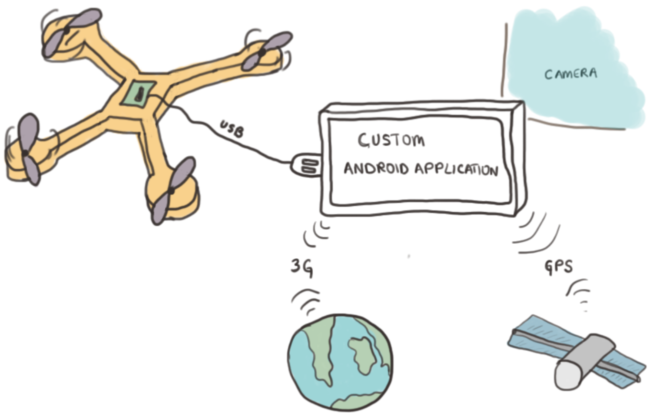


FIGURE 2.6: A custom designed and programmed AtMega 328 chip will act as a intermediate connector to decode the signals coming from the android smartphone through USB and send appropriate commands to the control board.

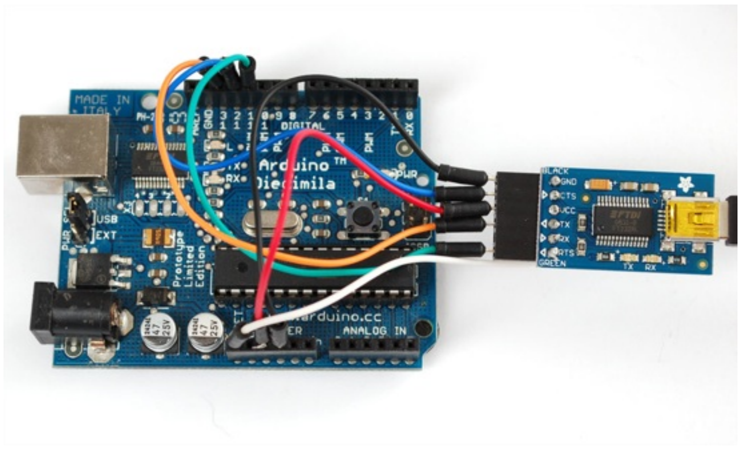


FIGURE 2.7: The figure shows how an AtMega 328 is connected to the FTDI USB-Serial converter chip. [2]

Leap Motion provides a library to use for creating hand objects and approximates the finger positions in the 3 dimensional space. According to these positions of the fingers, we can code the Leap Motion to generate http requests to the android on the quad-copter through internet.

We will program a java application to detect the hand gestures and then according to the gesture input, a particular http request will be made to the android. Eclipse IDE will be used to program the desktop application which will provide the internet connection to the quad-copter.

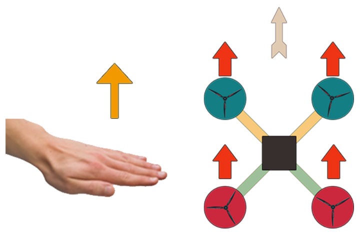


FIGURE 2.8: Moving the hand upwards in the z-axis in front of the Leap motion should provide us with a +ve change in z value.

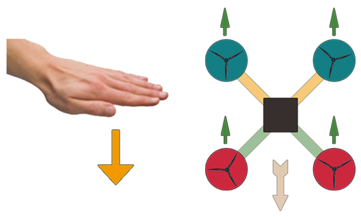


FIGURE 2.9: Moving the hand downwards in the z-axis in front of the Leap motion should provide us with a -ve change in z value.

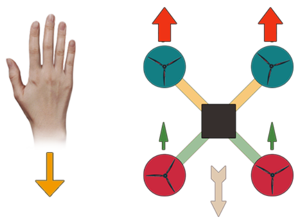


FIGURE 2.10: Moving the hand towards the body can give us -ve value of movement in y-axis.

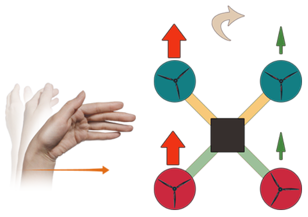


FIGURE 2.11: A +ve value in x-axis can be obtained when the user moves the hand to the right.

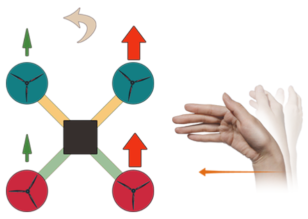


FIGURE 2.12: A -ve value in x-axis can be obtained when the user moves the hand to the left.

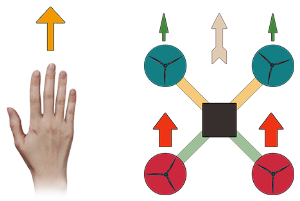


FIGURE 2.13: Moving the hand away from the body can give us +ve value of movement in y-axis.

.

2.3 Kalman Filter

There is a module in the controller board that processes the huge amount of samples the microprocessors take from the sensors. To put that into perspective let's say one have a micro-controller that has a clock speed of 15 MHz, that means the micro-controller will be taking 15 million samples from the sensor every second. All this amount of data is meaningless if the right data is not processed from this, Kalman Filter is a sequence of steps that will initially remove the sampled data that are purely noise and then average out the important data to pick the most appropriate one for input.

The leap motion captures about 200 frames per second or more depending upon the computer processing power. This is huge data speed and sudden errors can cause the system to In our project, we will implement a Kalman Filter on the hand gesture data.

2.4 Communication

2.4.1 XBee

2.4.1.1 The Advantages of Zigbee over other Technologies

We are all aware of standard and popular technologies like Bluetooth and Wifi which are perfect for mid to high data rates for voice, video, etc. However, the industry is not quite satisfied to use these technologies in applications like sensors arrays and control up unitll now. Application requires less lagging, very low power consumption for long battery lives and for large device arrays.

2.4.1.2 Zigbee for Quadcopter-User Communication

The industry is beginning to realize a new mode of communication; Zigbee Communication. The main advantages include low power consumption and unlike Wi-Fi ; it doesnt depend on internet. Hence if the internet is down due to any reasons; which is very common; than the data flow will vary.

Our project will be using this device for sending the user commands to the microcontroller located in the controller board of flying quadcopter. It is in application like this that the power of Zigbee is heavily felt. Since our copter will require real time data flow for a smooth flights and manouvers of the copter, we cannot afford any lag of data transfer; which is a grey area for both 3g and Wi-Fi. These two standard mode of connection depends on the traffic while our implementation of Zigbee is one to one. At all point of time the chip is ready to transfer data.

Zigbee is popular for other important reasons as well; it is easy to set up and it consumes very low power. Once again notice that our project has significant limitation when it comes to power. The battery we are carrying is heavily drained by the four motors so quickly that the flight time can barely carry it to a desired location. On top of that if we try to implement a Wi-Fi module or RF module for communication; the flight time will only get smaller. A Zigbee; on the other hand; consumes almost innoticable amount of power which gives the second reason of why it is an ideal choice for the purpose of communication between the quadcopter and the user.

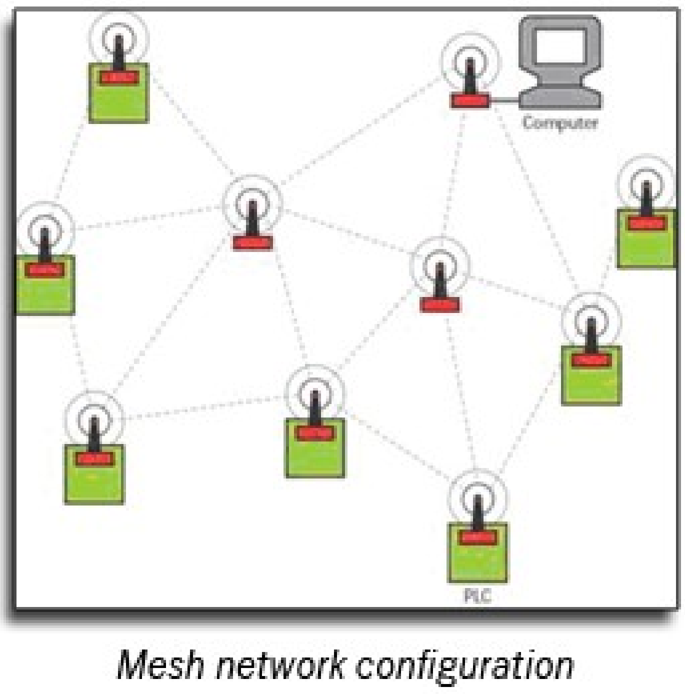
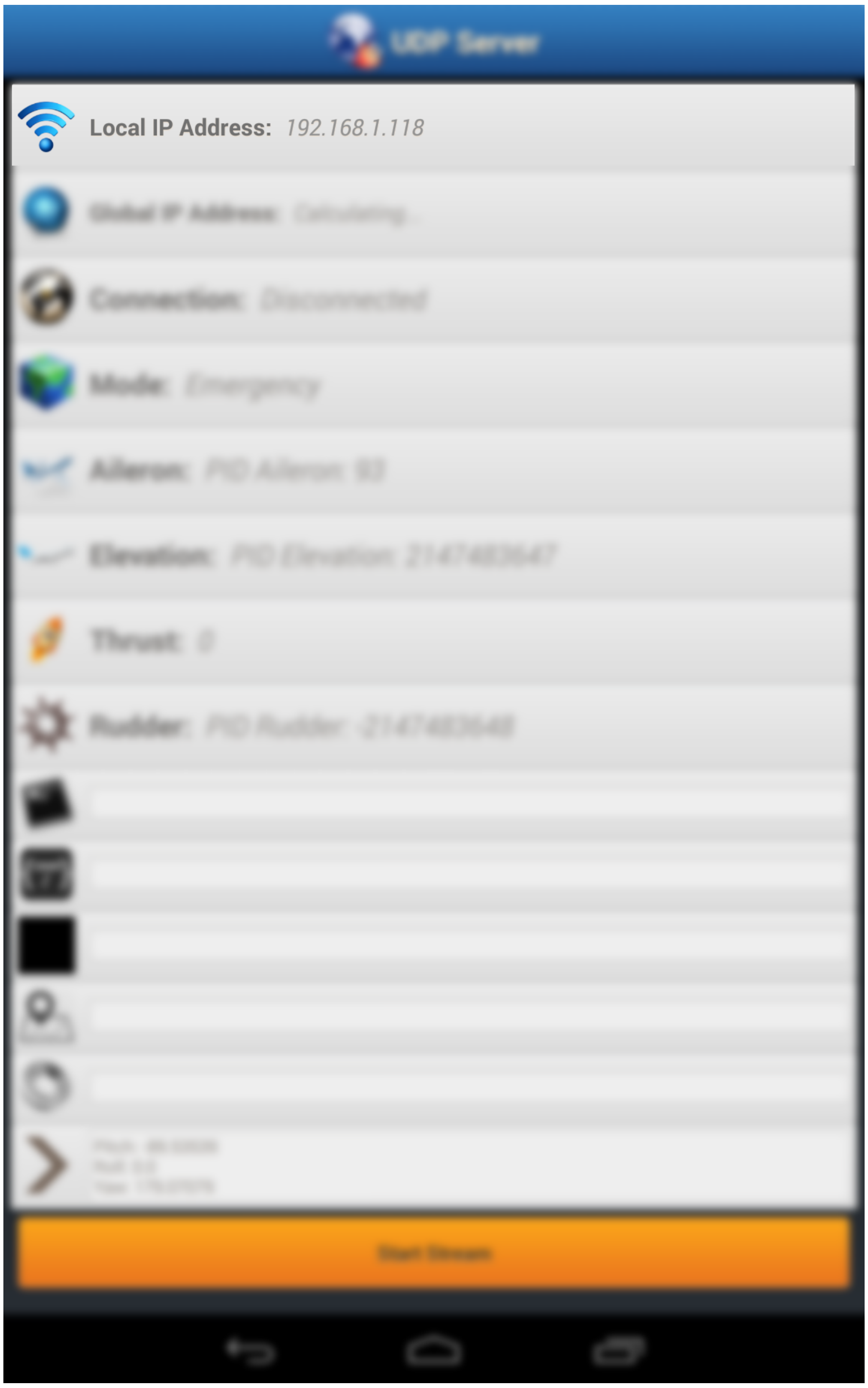


FIGURE 2.14: Zigbee mesh network configuration.

"In June 2009, the Continua Health Alliance, a consortium of health technology vendors led by Intel, announced that it was endorsing ZigBee as its low power local area network standard, bypassing rival low power technologies Sensium, ANT+, BodyLAN (used in Nike+) and Z-Wave, as well as traditional WiFi and RF technologies. "

2.5 Android



Local IP Address: 192.168.1.118

Global IP Address: Calculating...

Connection: Disconnected

Mode: Emergency

Aileron: PID Aileron: 93

Elevation: PID Elevation: 2147483647

Thrust: 0

Rudder: PID Rudder: -2147483648

FIGURE 2.15: Local IP Address: it shows the IP address given to the device inside the LAN

This the IP address that we need to know when trying to connect the smartphone on the quadcopter from the user phone/laptop/pc. This was achieved in the following code from. It is achieved in the following code:

```java
//get ip address using Wi-Fi manager in int format..
WifiManager wifiManager = (WifiManager) activity
    .getSystemService(activity.WIFI_SERVICE);
WifiInfo wifiInfo = wifiManager.getConnectionInfo();
int ipInt = wifiInfo.getIpAddress();


//convert the int to proper String format for the IP
String ipStrng = String.format("%d.%d.%d.%d",
        (ipInt & 0xff),
        (ipInt >> 8 & 0xff),
        (ipInt >> 16 & 0xff),
        (ipInt >> 24 & 0xff));

//show the ip in our TextView
peace.setText(ip_value, ipStrng);
```

Code 1: ServerUDP.java

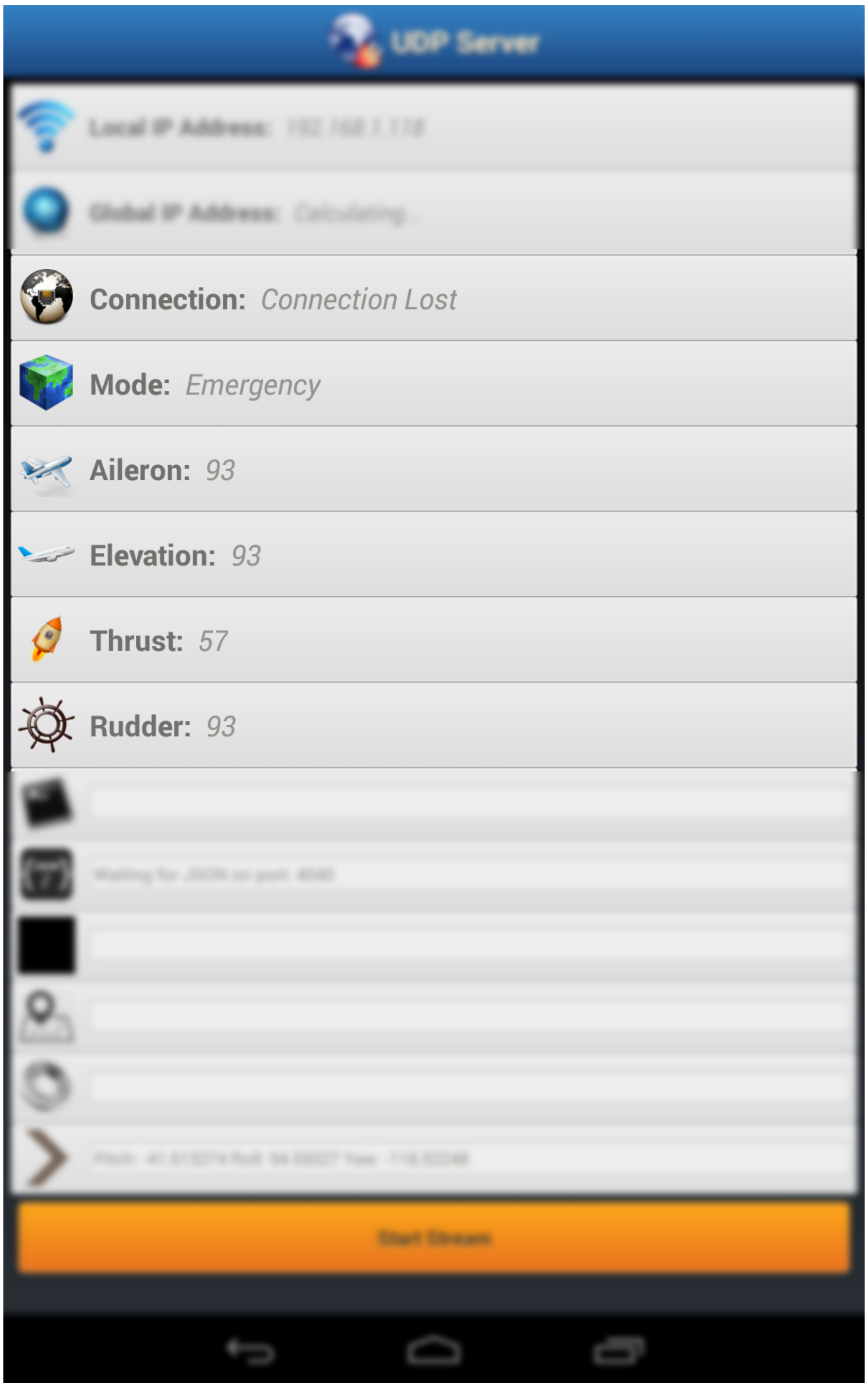
As with all other threads and services; the ServerUDP is initiated in the MainActivity as shown below:

```java
ServerUDP serverUdp = new ServerUDP(3030);
serverUdp.uiQuad(activity, aileron_value, elevation_value,
        thrust_value, rudder_value, connection_value, mode_value,
        ip_value, global_ip_value, reply_value);
serverUdp.start();
```

Code 2: MainActivity.java

Find it here: server–¿src–¿server–¿MainActivity.java

In order for the thread to talk to the text views in the MainActivity I passed the text views to the thread through the *uiQuad()* to avoid re-initiating these views inside this thread.



Local IP Address: 192.168.1.118

Global IP Address: Calculating...

Connection: *Connection Lost*

Mode: *Emergency*

Aileron: *93*

Elevation: *93*

Thrust: *57*

Rudder: *93*

Waiting for JSON on port 4040

Pitch: [illegible] Roll: [illegible] Yaw: [illegible]

FIGURE 2.16: Views to Display the Mode and the Control

The text views essentially reassures and confirms packets being received and the state of connection. This helps one to be certain that whether it is safe to approach the copter and un-plug the battery and also confirm if it is receiving the control values from the ground station.

Mode view tells in which block the arduino is currently on; the possible modes we have is the hand-gesture mode and emergency mode. **Aileron** view shows how much aileron value is being sent from the user, **Elevation** value show how much elevation value is being sent from the user, **Thust** value shows how much thrust value is being sent from the user and finally **Rudder** value shows how much rudder value is being sent from the user. The data is continuously read from the input stream using *receive(DatagramPacket dp)* and sent to the Atmega328P using *send(byte[] msgAr)*.

```java
while (true) {
  ds.receive(dp);
  if (dp != null) {
    // send the data received to usb
    usb.send(msgAr);
    // show the values received in the views
    updateUi();
  }
}
```

Code 3: ServerUDP.java

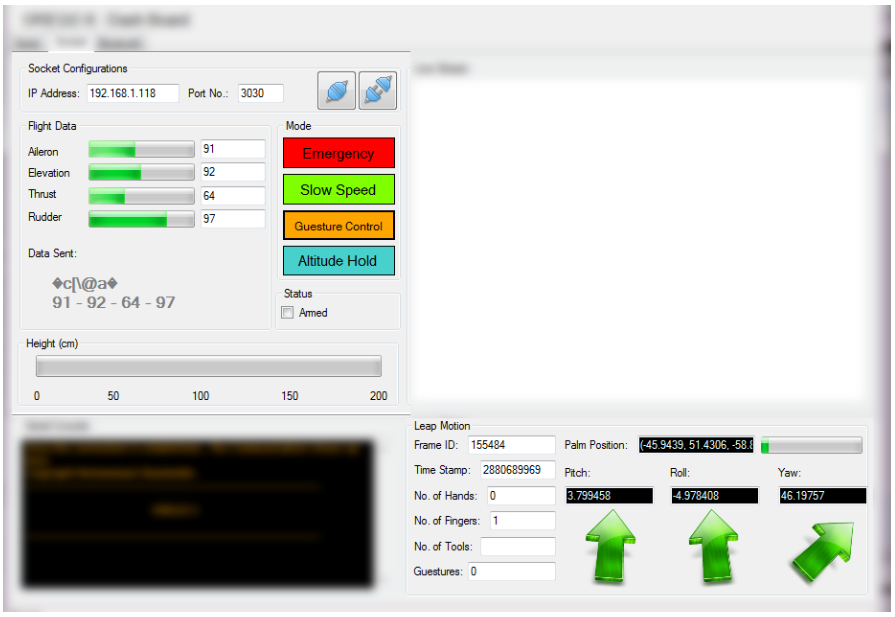
```java
private void updateUi() {
  byte[] data = dp.getData();
  if (data[1] == Character.valueOf('a')) {
    peace.setText(ConnectionValue, "Connected");
    peace.setText(ModeValue, "Emergency");
  } else if (data[1] == Character.valueOf('b')) {
    peace.setText(ConnectionValue, "Connected");
    peace.setText(ModeValue, "Slow Speed");
  else if
  ....
  peace.setText(ThrustValue, String.valueOf(data[4]));
  peace.setText(RudderValue, String.valueOf(data[5]));
  .....
}
```

Code 4: ServerUDP.java

We did a simple if-else statements to find the state from the incoming packets and update the *Mode* view. Then the bytes are converted to string for display using *String.valueOf(byte b)*
Finally for safety a connection timeout is setup inside the ***while(true)*** loop which starts a timer as soon as it finds the input stream empty and runs it until it finds any data there and when the time reaches 3 seconds there is an exception to kill the flight

```java
while (true) {
  try{
    ds.setSoTimeout(3000);
    ds.receive(dp);
    ...
  } catch (SocketTimeoutException  es){
    ...
    usb.send(kill);
  } catch (IOException e) {}
}
```

2.5.1 Leap Motion



IP Address: 192.168.1.118 Port No.: 3030

Aileron 91
Elevation 92
Thrust 64
Rudder 97

Data Sent:
◆c[\@a◆
91 - 92 - 64 - 97

Armed ☐

Height (cm)

Frame ID: 155484
Time Stamp: 2880689969
No. of Hands: 0
No. of Fingers: 1
No. of Tools:
Guestures: 0
Palm Position: (-45.9439, 51.4306, -58.8
Pitch: 3.799458
Roll: -4.978408
Yaw: 46.19757

FIGURE 2.17: Leap Control in Hand-gesture Mode

Leap sdk provides some powerful libraries that basically takes the raw data and makes meaning out of them so that developers may use it for their own application. Among them we used the ***Controller*** class and ***Frame*** class. The hand goes through states at every second and the leap motion is capturing the state of hand every microsecond and they get processed by the Controller. These data can be obtained as Frames which we use it to deduce gestures. Now to access these incoming frames from the controller and react to these frames we use a ***Listener*** which simply means that within every given interval check the frame and act according to the current frame.

```
/*
The most important elements declared for the gesture detection
*/
Frame currentHandFrame;
SampleListener listener;
Controller controller;
/*
The system calls this function within the given interval...
*/
this.LeapFrameUpdate.Interval = 50;
this.LeapFrameUpdate.Tick += new System.EventHandler(this.LeapFrameUpdate_Tick);
/*
At the loading of the 'Form'....
Create a sample listener and controller
*/
Controller controller = new Controller();
SampleListener listener = new SampleListener();
/*
Have the sample listener receive events from the controller
*/
controller.AddListener(listener);

LeapFrameUpdate.Start();
```

Every 50 millisecond we extract the frame and analyze it. Analysis includes the detection of the number of hands and fingers, then we deduce the roll, pitch and other values from the movement of the first hand. These values are then updated on the user interface and sent to the receiver smart phone through Datagram socket.

```
/*
Inside LeapFrameUpdate_Tick() that acts as a listener we start by
detecting hand and do the calculation based on first hand
*/
currentHandFrame = controller.Frame();
Hand hand = frame.Hands[0];
/*
Check if the hand has any fingers
 */
FingerList fingers = hand.Fingers;
/*
Calculate the hand's average finger tip position
and display to the user interface...
*/
Leap.Vector avgPos = Leap.Vector.Zero;
foreach (Finger finger in fingers)
{
  avgPos += finger.TipPosition;
}
avgPos /= fingers.Count;
/*
Show the 3D coordinate position of hand in text box
*/
PositionBox.Text = hand.PalmPosition.ToString();
/*
Get the hand height from sensor and check its valid range and update progress bar
*/
Thrust = (int) ((((500.0 - hand.PalmPosition.y) / 500.0) * 100.0) + 50.0);
positionProgress.Value = (int)hand.PalmPosition.y;
/*
We load the arrow image and define a point that will act as
an anchor for the arrow rotation
 */
System.IO.Stream file = thisExe.GetManifestResourceStream("OregoController.Properties
System.Drawing.Point p = new System.Drawing.Point();
p.X = 128;
p.Y = 128;
```

To have a more natural understanding of our hand position that the leap motion detected we thought of using arrows. The rotation of arrow is mapped to the values that were calculated from each frames; roll, pitch, yaw.

```
/*
Float convert the pitch value and apply on the arrow...
 */
float pitchVal = (direction.Pitch * 180.0f / (float)Math.PI);
pictureBox2.BackgroundImage = RotateImage(global::OregoController.Properties.Resource
    p, pitchVal);
/*
Float convert roll value and apply on the arrow...
 */
float rollVal = (normal.Roll * 180.0f / (float)Math.PI);
pictureBox3.BackgroundImage = RotateImage(global::OregoController.Properties.Resource
    p, rollVal);
/*
Float convert the yaw value and apply on the arrow...
 */
float yawVal = (direction.Yaw * 180.0f / (float)Math.PI);
pictureBox4.BackgroundImage = RotateImage(global::OregoController.Properties.Resource
    p, yawVal);
```

Initially a tcp socket was used but after discovering the more real time responsiveness of a udp socket we implemented our communication through Datagram Socket. Its also much easier to implement as the only thing it requires is the byte array and the destination address. As one may notice the Datagram works in a clever way; they simply spray out data in the network where each of the data includes the destination address and the right receiver will pick it up for himself.

```
/*
Send values to receiver smart phone on the quad-copter...
*/
byte[] packet = new byte[] {
  0xAA,
  Convert.ToByte('a'),
  Convert.ToByte(Aileron),
  Convert.ToByte(Elevation),
  Convert.ToByte(Thrust),
  Convert.ToByte(Rudder),
  0xAA
};
UdpClient udpClient = new UdpClient()
udpClient.Send(packet,7);
```

2.6 Arduino

The Atmega328P in our PCB reads the data in the serial format and decides which mode to enter and consequently what values to forward to controller board.

Any data received in the usb triggers the ***void serialEvent()*** function and this is where we take the packet, hold it and then analyze it as to what mode to go. The following analysis is taking place inside the ***void serialEvent()***.

```
while (Serial.available()) {
  char commands[7];
  delay(5);
  Serial.readBytes(commands,7);
  if(commands[0] == commands[6]){
  // Valid command detected
    if((char)commands[1] == 'a'){
      // Emergency State
      CurrentState = 0;
    }
    ...
else{
  Serial.println("Wrong Command!");
  Serial.println(commands);
}
```

Everything starts to change at this point, no more the Atmega328P is sending the same fixed value of 93, 93 ,93, 54 and 93 for Aileron, Elevation, Thrust and Rudder respectively. It looks at the orientation of the hand to determine Aileron , Elevation and Rudder while the distance of the hand from the *Leap* to determine the Thrust. Hence this is also a very tensed moment for those standing nearby since it may be un-predictable at times as to what the Leap will mistakenly send if the hand position suddenly falls.

```
else if((char)commands[1] == 'c'){
    // Free Hand Guesture Control State
    CurrentState = 2;
    AileronValue = commands[2];
    ElevationValue = commands[3];
    ThrustValue = commands[4];
    RudderValue = commands[5];
}
```

In the hand gesture we put our current state value to '2' and extract the Aileron , Elevation , Thrust and Rudder from array packet 2, 3, 4 and 5.

2.7 Navigation Design

The aim was to send way-point coordinates from user smartphone to quad-copter smartphone for navigation purpose.To achieve the purpose I used JSON to send the coordinates to avoid developing any protocol for parsing it as JSON is already a standard and well developed protocol. Initially the idea was to upload the coordinates form the user to the cloud and this coordinates will stay safe there; from there the smartphone on the quad-copter will retrieve the coordinates when required; come to think of it; it will also save memory space in the quadcopter even it is very insignificant. Another benifit one can think of is that if the battery of the smart-phone on the copter dies the coordinates will still remain in the cloud and later a history of checkpoints selected and visited may be demonstrated.

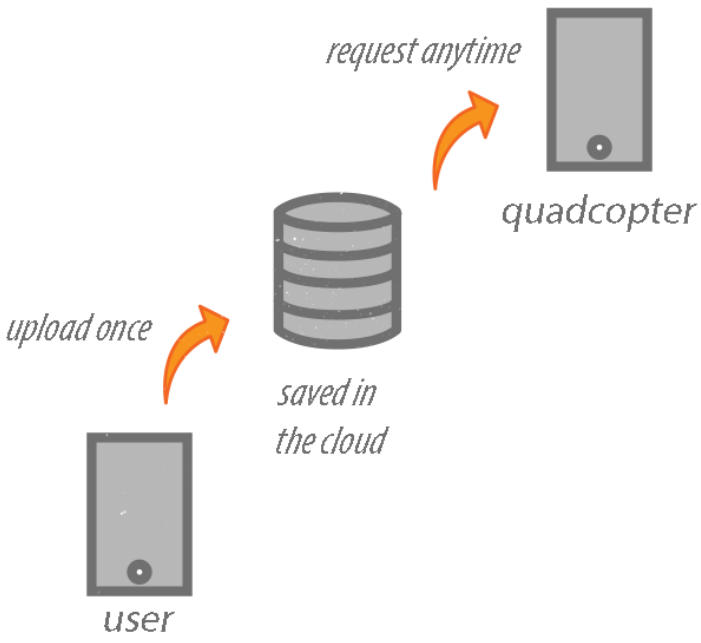


FIGURE 2.18: Way-Point Initial Concept

The only other option before json was to convert the coordinates into a string separated with a "." so that in the other side it can be parsed by splitting the string at the occurance of dots. After splitting it can be stored in an array. 2.19

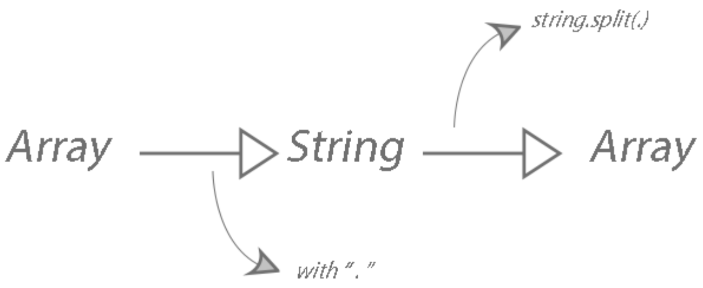


FIGURE 2.19: Split Coordinate Array

Finally this became more simple with JSON. JSON allows the use of multiple arrays with key word that can easily extracted on the other side by referring to the key word 2.20.

Figure 2.20: JSON Array Concept

The overall logic are the following:

1. User sends the coordinates
2. Quad-copter receives the coordinates
3. The coordinates are parsed
4. The coordinates are stored
5. The coordinates are used for navigation

```
//Store lat and lng in separate json array at the client...
JSONArray latJsonArray = new JSONArray();
latJsonArray.put(lat);

//Combine the two separate json array in
//a json object...
JSONObject jsonObject = new JSONObject();
jsonObject.put("lat", latJsonArray);
jsonObject.put("lng", lngJsonArray);

//extract the json array and store in an array list
//of coordinates at the server...
JSONObject jsonObject = new JSONObject(message);
JSONArray lngJsnArray = jsonObject.getJSONArray("lat");
JSONArray latJsnArray = jsonObject.getJSONArray("lng");

for (int i = 0; i < latJsnArray.length(); i++) {
 →      Coordinates coord = new Coordinates(latJsnArray.getDouble(i),
 →  lngJsnArray.getDouble(i), i);
        coordList.add(coord);
}

//Calculate bearing for navigation...
Location destination = new Location("Destination");
destination.setLatitude(coordList.get(1).getLatitude());
destination.setLongitude(coordList.get(1).getLongitude());
double bearing = currentLocation.bearingTo(destination);
```

2.8 Data Communication Design

The challenge was to control the quad-copter from distance which in other word meant wireless communication. The wireless communication is one of the fundamental element that makes controlling quadcopter or any hobby toys something very entertaining and consequently various people implemented different technologies based on the performance and the goal they wished to achieve.
There are various options to achieve that a wireless communication:

- Cellular Communication
- Internet Protocol

- Bluetooth Communication
- Xbee Communication
- Bluetooth

Classic Bluetooth is for "battery-intensive" operations such as streaming and communicating between devices whereas *Bluetooth Low Energy* is for application with low power consumption. ***Wi-Fi Direct*** devices can connect to each other without having to go through an accesspoint, so no router needed. This technology allows Wi-Fi direct installed devices to see other devices in the network and find out what each of these devices are; for instance there could be two Wi-Fi direct devices showing up the in the list of Wi-Fi direct devices in a smart-phone but is very vague as in what kind of devices they since the only thing that we are aware is that it has somewhere near the Wi-Fi adapter this chip integrated. Hence the importance of knowing it's *service.* Service I believe is what the device is capable of. Some of the applications of this technology are the following:

- Print from laptop or smartphone to a wireless printer
- Store images with someone across a room.
- Send video from phone to the TV.

Android has their own library of classes and interfaces(API) which lets us discover the available devices and know what the available devices can perform(as in image storing, video sharing or printing) called Direct Device Discovery and Service Discovery. ***Internet Protocol***is a very general concept that essentially means data being transferred between a network of devices. The difference starts in the technique of this data transferred; is it cable or signal.

This internet technology has been implemented wireless by the company called "Wi-Fi". Nowadays all router has a Wi-Fi adapter that generates a 2.4 GHz RF signals to create a LAN. This router also acts as the 'gate-way' between this LAN and other LANs aka the Web. Our aim was to use this Wi-Fi technology initially for relatively short range communication. In Android this is achieved by something known as 'socket' communication. Socket is a class that interacts with the Wi-Fi Service of the Android Layer which in-turn talks to the Wi-Fi adapter of the Linux Layer. This class basically does all the talks and the only thing a user has to input is the address for the destination device and the app of the device. Through socket class we can retrieve the data received in the Wi-Fi adapter and at the same put our data in the buffer for the Wi-Fi adapter to modulate it and stream it out. The functions are *.inputstream()* and *.outputstream()* respectively 2.21.

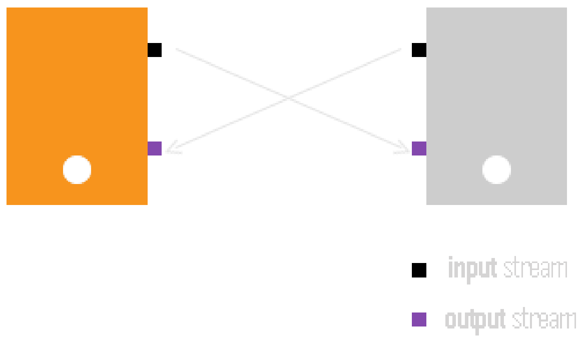


Figure 2.21: Socket Communication in JAVA

2.9 Sources Explained

2.9.1 Client

The ***Coordinates.java*** class was made to store the LatLng retrieved after parsing the json and have few functions to optimize the uploading task to server simple and easy.

```java
/*
It holds the lat and lng of a particular coordinate...
*/
public Coordinates(double lat, double lng, int i) {
  this.latitude = lat;
  this.longitude = lng;
  this.id = i+1;
}

/*
setter and getter...
*/
public double getLatitude() {
  return latitude;
}
public void setLatitude(double latitude) {
  this.latitude = latitude;
}

/*
and finally a future plan to have the upload to server functionality
*/
public void upload(){}
```

The ***MainActivity.java*** in client will allow users to use GoogleMap to navigate to the current location of the user and select way-points around that location by tapping on the screen. With every tap; the activity will render a marker on the map and also store the Lng-Lat in two separate *json* array. If not happy with the waypoints; the user may opt to remove the previous choice by clicking the *remove* which will do two action:

- remove the markers from the google map.
- reset the json arrays for the new selection.

Another thing to notice is the polyline between the waypoints. The polyline requires two coordinates as parameter to render the polyline; however since in our case we wished to build a polyline between the previous coordinate and the current coordinate I had to a little brainstorm to come up with the following algorithm:

1. every tap on the map will increment a counter;
   ```
   count++;
   ```

2. the previous coordinate is obtained by typing ; jsonAr[counter - 1] while the current coordinate is jsonAr[counter]

```
/*
*/
client = new Client(mainAcitivity, portInt, ip);
client.start();
```

2.9.2 Server

Location class is a service that connects to Location Services in Google Play Services using LocationClient and receives the coordinates from the LocationListener interface whose refresh rate can be configured in the beginning

First we initiate *LocationRequest* and *LocationClient* in the *onCreate()* method of our service. Now that things are initialized, we wait for the *LocationClient* to get connected to the *Location Services* and then we do two important things; connect the *LocationClient* in the *onStartCommand()* and start the update on *onConnected()*.The values are sent to the ***MainActivity*** through *Message* by a *Messenger* and MainActivity receives it using *Handler*.

```
/*
inside void onCreate() set the:
- update intervals and
- data precision
*/
mLocationRequest = LocationRequest.create();
mLocationRequest.setPriority(LocationRequest.PRIORITY_HIGH_ACCURACY);
mLocationRequest.setInterval(UPDATE_INTERVAL);
mLocationRequest.setFastestInterval(FASTEST_INTERVAL);

/*
inside void onCreate() initiate location client by feeding it:
-ConnectionCallbacks and,
-onConnectionFailedListeners
*/
mLocationClient = new LocationClient(this, this, this);

/*
inside void onStartCommand()
connect the location client to location service...
*/
mLocationClient.connect();

/*
Access the location coordinates from the location service through
onLocationChanged(Location location) interface...
Take the location and convert them to string
Send them to MainActivity
*/
String locString = "Latitude :" + location.getLatitude()
  + "Longitude :" + location.getLongitude();
sendMessage(locString);
```

The smart-phone to usb device communication in our ***USB*** class is achieved by a library developed and regularly maintained by Mike Wakerly [**?**].
His library contains the following important classes among a list of classes in the driver package:

1. UsbSerialDriver.java

2. UsbSerialProber.java
3. FtdiSerialDriver.java

He also provided an input listener in his util package called *SerialInputOutputManager.java* which we used for listening to incoming data to the USB and accessed it through an instance of ***SerialInputOutpuManager.listener***. These classes were used in making the ***USB*** class which is a worker thread in our app. In the run() method I wait for the USB to connect and once it connects I take its driver and configure its reading listener and writing parameter. The ***void write(char[] msg)*** is designed for any app component to use it to send data to the ftdi using this function. Initially the ***UsbManager*** of the android layer is requested to keep an eye to detect any peripheral device on the usb port.Once the driver is found, we configure the writing parameter and input listeners.

```
/*
Android UsbManager is initialized that keeps
an eye on any usb connection...
*/
UsbManager usbManager = (UsbManager) activity.getApplicationContext()
  .getSystemService(Context.USB_SERVICE);

/*
The usb manager will constantly look for a device
and extract the driver using UsbSerialProber...
*/
while (usbSerialDriver == null) {
  usbSerialDriver = UsbSerialProber.findFirstDevice(usbManager);
}

/*
The driver is opened for communication...
*/
usbSerialDriver.open();

/*
configure output parameter...
*/
usbSerialDriver.setParameters(1200, 8,
    UsbSerialDriver.STOPBITS_1,
    UsbSerialDriver.PARITY_NONE);

/*
and the input listeners...
*/
mSerialIoManager = new SerialInputOutputManager(
    usbSerialDriver, mListener);
```

JSON [?]; short for JavaScript Object Notation ; is a data-interchange format that can be used to send-receive values between devices whose strength lies at the ease of parsing.
Hence our ***Parse.java*** class is worker thread that receives the coordinates sent from the user in JSON format and parse it using JSON class for navigation purpose and also displaying them in the activity.

```
// wait at port 4040 for any request from the user...
serverSocket = new ServerSocket(4040);
client = serverSocket.accept();

//parse once a data is received in that port....
if (msg != null)
  parse();

//parse it inside the void parse() function...
JSONObject jsonObject = new JSONObject(msg);
JSONArray lngJsonArr = jsonObject.getJSONArray("lat");
JSONArray latJsonArr = jsonObject.getJSONArray("lng");
```

The ***ServerUDP.java*** class will receive the DatagramPacket containing the mode value and the control values: Aileron, Elevation, Thrust, Pitch. We are using udp ***DatagramSocket*** which is much faster than the standard tcp ***Socket***. The fundamental difference between udp and tcp is that once the connections gets lost in udp there is no need to re-establish the connection since there is no concept of connection in udp. A DatagramSocket simply sprays around packets in the network and the packets itself finds its way to the destination according to the destination address included in every single packet in the case of a client. While in the case of a udb server its even more easier, we tell the capacity of a packet and then mention the port of the socket before initiating it.

```
//create an array of byte to hold our data...
byte[] msgAr = new byte[MAX_UDP_DATAGRAM_LEN];

// put this array inside our packet...
DatagramPacket dp = new DatagramPacket(msgAr, msgAr.length);

//initialize a datagram server socket
DatagramSocket ds = new DatagramSocket(port);

// receive any data and put inside this packet...
ds.receive(dp);

//send it to the Atmega328p in our pcb...
if (dp != null)
  usb.send(msgAr);
```

MainActivity.java is the process that is dedicated to the users interaction. It seeks the required data from the other app components and also sends the data to them by using Messenger-Handler. Hence we find the layout views by querying the resource id and feed them to the view instances in the MainActivity. Then the important app components ; threads and services are initiated that will carry out the main task of acting as the receiver module for the quadcopter.

```
/*
TextView , Button and ImageButton are declared...
*/
TextView ip_value,global_ip_value, connection_value, rudder_value,
 →  mode_value, aileron_value, elevation_value, thrust_value,
 →  replay_value, json_value;
Button start_cameraStrea;
ImageButton serviceStart;

/*
These view are attached with instances in the following way;
*/
elevation_value = (TextView) findViewById(R.id.elevation_value);
serviceStart = (ImageButton) findViewById(R.id.startService);
start_cameraStream = (Button) findViewById(R.id.startStream);

/*
Right after initializing these layout components
threads and services are started.
Location service called up using intent
*/
Intent intent = new Intent(activity, Location.class);
startService(intent);
```

```
/*
The streaming activity inside the third party app IPCam app
is called up using intent...
*/
Intent intent_stream = new Intent().setClassName(
    "com.pas.webcam", "com.pas.webcam.Rolling");
startActivity(intent_stream);

/*
The ServerUDP is first initialized and then its run function is
→   triggered...
*/
ServerUDP serverUdp = new ServerUDP(3030);
serverUdp.start();

/*
Finally a separate port is used for the incoming way-points in json to
→   be parsed
*/
Parse parse = new Parse(4040);
parse.start();
```

2.10 3D Model Design

A 3D Model of the quadcopter was custom designed to fit our needs. The 3D Model was supposed to have arms of specific lengths to allow the propellers to rotate freely. We designed the whole quadcopter in Google Sketch Up taking care of these parameters. The design also was supposed to have a holder for the mobile phone. We designed a intelligent mechanism for controlling the tilt angle of the mobile. We used 10 inch diameter propellers with 3.8 inch of pitch. The pitch controls the amount of air scoped with each rotation. The diameter controls two things, amount of air the propeller pushes down, and the weight of the propeller. Careful calculations and testings were done to calculate the amount of weight of each motor can lift.

Features of the Model:

- Four less infill light 10" motor arms.
- Three center platforms with screw holes to mount the control chip.
- A raft to hold and rotate the mounted android smart-phone.
- Four stands for landing gear.

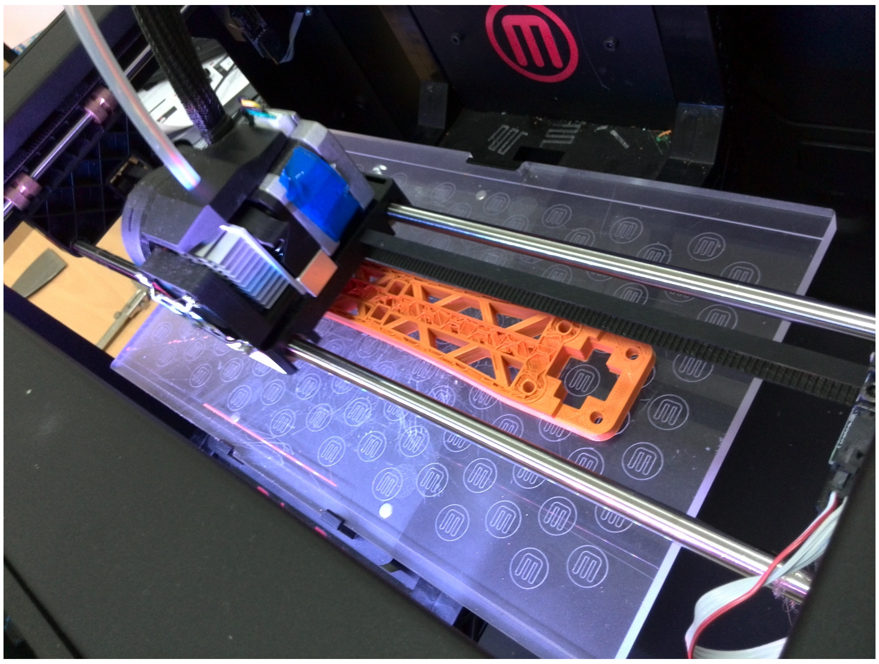


FIGURE 2.22: A arm of the quadcopter is being printed by the MakerBot.

2.11 Gyroscope Sensor

A quadcopter is highly prone to instability due to irregular distribution of mass or wind.Basic moves of a quadcopter like lifting straight off the ground to complex moves like pitch(right) and roll or yaw cannot be achieved without stabilizing the quadcopter.

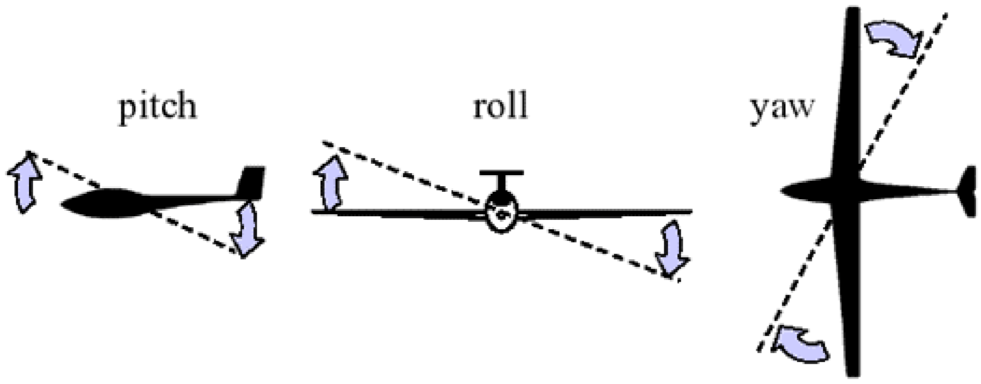


FIGURE 2.23: Explanation of the traditional 3 Dimensional rotation movements. These movements were defined for airplanes but as time passed by, these same terminologies were adopter by quad-copters although quadcopters don't bend their wings for roll(aileron).

It has become the industry standard to implement sensors to know the precise measurment of imbalance instantly and exert the proper combination of force to bring it back to balance. These sensors are known as Gyroscope and they use the law of physics to measure this phenomenon.

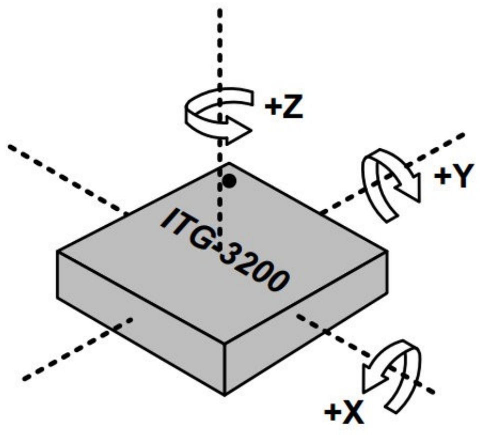


FIGURE 2.24: The gyroscope sensor is designed to measure the rotation per second and the angle of rotation of the object its mounted to when it experiences an imbalance.Below is a typical 3-axis gyroscope ITG3200. [6]

Hence, when the object tilts a little to the right(roll), the gyroscope on the y-axis will record a reading while the other three will not record any reading, or when the device tilts up(pitch) the sensor in x-axis will record a value while the rest wont record any value and finally if the device rotate in the horizontal plane the sensor in z plane will measure a current due to angular rotation around the z-axis

The working of the gyroscope is as fascinating as it is clever. There is a small mass inside each gyroscope on a particular axis which is held by very small springs. When there is a rotation, this will cause a centripetal force to form whose direction will depend on the direction of rotation. For instance; if the gyroscope is rotating right there will be a force that is exerted toward the center while it the gyroscope is rotating outward there will be a force that opposite to the center.

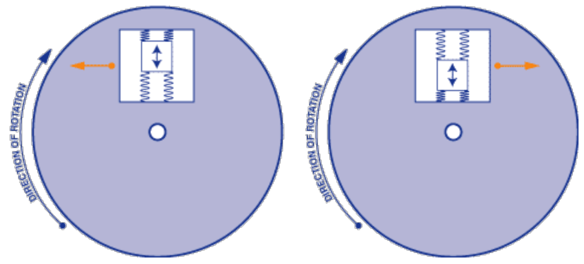


FIGURE 2.25: These forces are exerts the spring and this movement generates very low-current electrical signal that is amplified and later read by the micro-controller. [6]

2.12 An unexpected tragedy

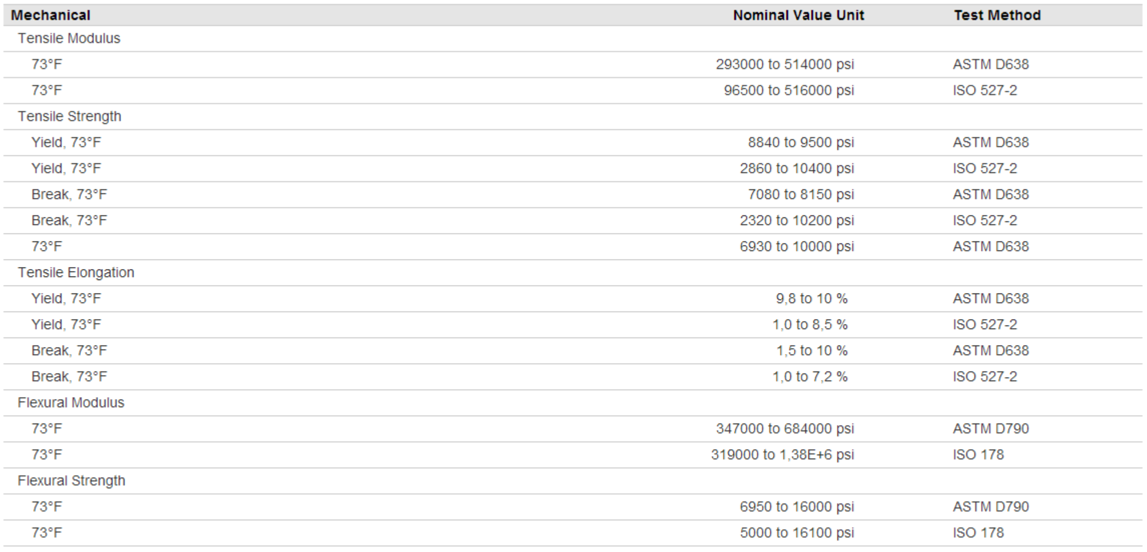
After designing the whole body using the lab's 3D printer, the quadcopter's body melted and twisted away one day due to sun heat. This gave us a strong lesson that the plastic used to manufacture the quadcopter body is very sensitive to the high temperatures. Since UAE's temperature is usually very high in major part of the year, the 3D plastic cannot be used for building the quadcopter.

2.12.1 Temperature Sensitivity of PLA plastic used in 3D printer

The plastic which is used in the 3D printer is Polylactic Acid thermoplastic aliphatic polyester. It is usually derived from renewable resources such as corn starch. It has a melting point of 150-160 degrees centigrade. The temperature in UAE reaches 45 degree centigrade in summers and even with a small amount of insulation, this temperature can reach high enough to just twist or deform the plastic. The choice of PLA plastic for quadcopter body is really bad when it comes to temperature sensitivity.

2.12.2 Strength of PLA plastic

In the first two-three attempts, the quadcopter crashed several times and broke the arms within milliseconds after coming in contact with the ground. This was also a big issue as we want the final product to have strength enough to withstand huge jerks and shocks in case the quadcopter falls down from huge height.



Mechanical	Nominal Value Unit	Test Method
Tensile Modulus		
73°F	293000 to 514000 psi	ASTM D638
73°F	96500 to 516000 psi	ISO 527-2
Tensile Strength		
Yield, 73°F	8840 to 9500 psi	ASTM D638
Yield, 73°F	2860 to 10400 psi	ISO 527-2
Break, 73°F	7080 to 8150 psi	ASTM D638
Break, 73°F	2320 to 10200 psi	ISO 527-2
73°F	6930 to 10000 psi	ASTM D638
Tensile Elongation		
Yield, 73°F	9,8 to 10 %	ASTM D638
Yield, 73°F	1,0 to 8,5 %	ISO 527-2
Break, 73°F	1,5 to 10 %	ASTM D638
Break, 73°F	1,0 to 7,2 %	ISO 527-2
Flexural Modulus		
73°F	347000 to 684000 psi	ASTM D790
73°F	319000 to 1,38E+6 psi	ISO 178
Flexural Strength		
73°F	6950 to 16000 psi	ASTM D790
73°F	5000 to 16100 psi	ISO 178

FIGURE 2.26: The table shows strength of the plastic used for 3D printing in our lab according to different standards. [3]

2.13 New Body Design

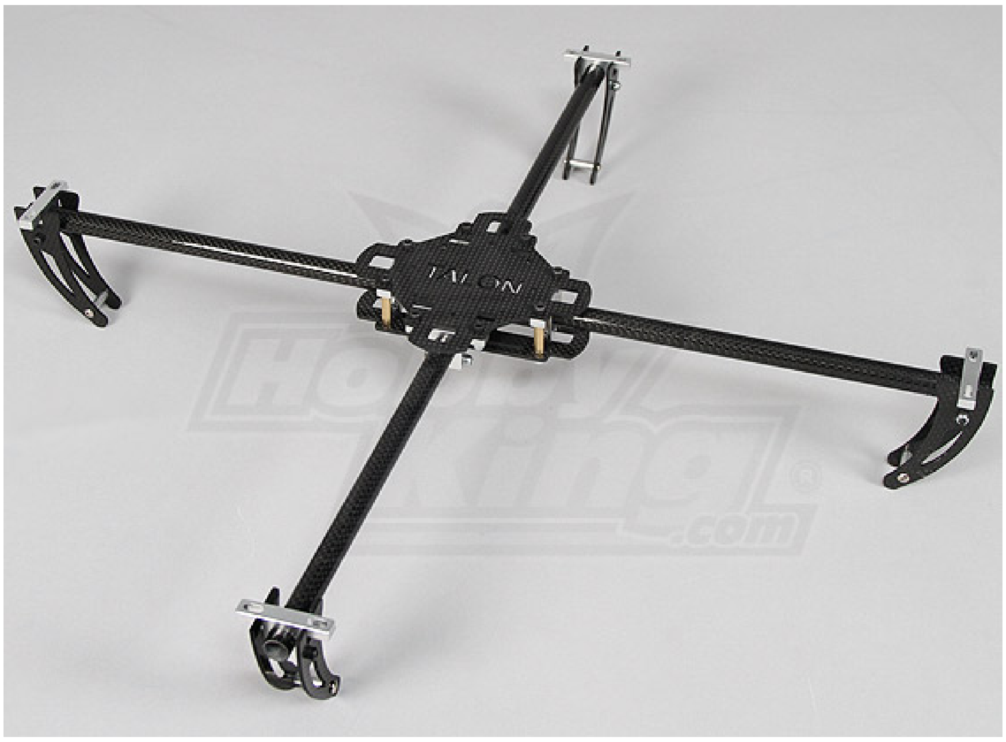


FIGURE 2.27: Turningy Talon Carbon Fiber Quadcopter Frame

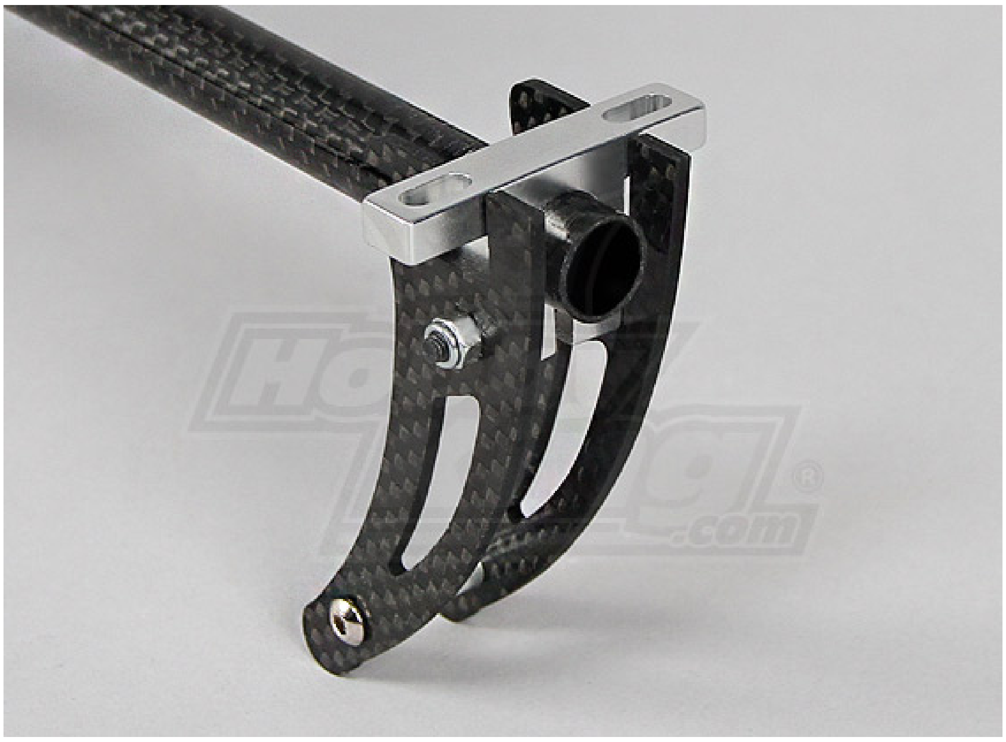


FIGURE 2.28: Motor Brackets Holder

After getting the lesson, we changed the body from PLA 3D printed plastic to Carbon Fiber body. The body was bought as a kit from HobbyKing named " Turnigy Talon Carbon Fiber Quadcopter Frame". Link: http://hobbyking.com/hobbyking/store/__22397__Turnigy_Talon_Carbon_Fiber_Quadcopter_Frame.html

2.14 Circuit Design

2.14.1 Circuit Design Version 1

Initially when we started our research on Quadcopter controller board, we found that we needed a PCB to find out what signals exactly are required by the quadcopter in order for it work perfectly. Additionally, connecting and disconnecting all the wires on the breadboard was very hectic and time consuming. For this purpose we created the version 1 of the PCB. It was the pre-final version of the PCB. This PCB's job is to transmit the appropriate Aileron, Rudder, Thrust and Elevation PWM signals to the controller board so that it can further perform PID control on brush-less motors. Essentially, this PCB is a bridge between the controller board and the smartphone. The following were the features of the first circuit.

- Servo connectors for four ESCs.
- Servo connector for mini servo which will be used for changing the mobile angle.
- Reset push button.
- Tilt Compensated Compass output pins.
- SPI programming header.
- 16 MHz crystal with 22pF capacitors for providing clock signal to AtMega328.
- Ultrasonic height sensor connection.
- FTDI connection pin.
- 3.3V Regulator for stepping down he voltage from 5V to 3.3V for the XBee.
- 3 indicator LEDs of different colors.
- XBee connection support.
- Bluetooth module connection support using the FTDI pins.

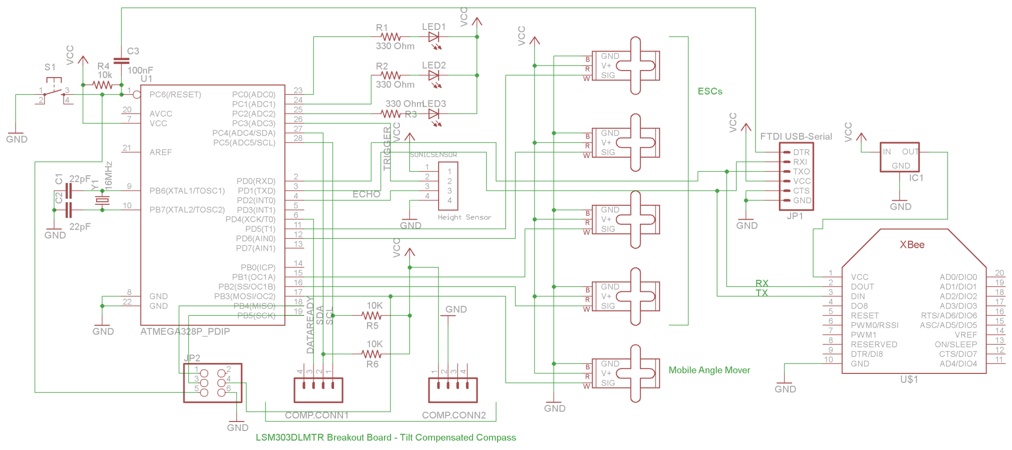


FIGURE 2.29: A circuit diagram of our prototype board design version 1.

2.14.2 Circuit Design Version 2

We quickly realized that several improvements can be done to the initial circuit to make the quadcopter have more and more features. The following are the additional features of the new PCB.

- IR Receiver for receiving Infra red signals. Basically this includes TV, CD player and other IR transmitters. The IR receiver already has a electronic circuit to amplify the IR signal and provide a signal when a change in IR level is detected. Also the Sensor already has the background noise removal circuit. SO we can just connect the infra-red sensor directly to the interrupt pin to read the bits transmitted and change the states of the quadcopter. This IR Receiver will be used.
- 5V Barrel Jack connector for providing 5V to the circuit.
- USB A jack for supplying power.
- USB B jack for supplying power.

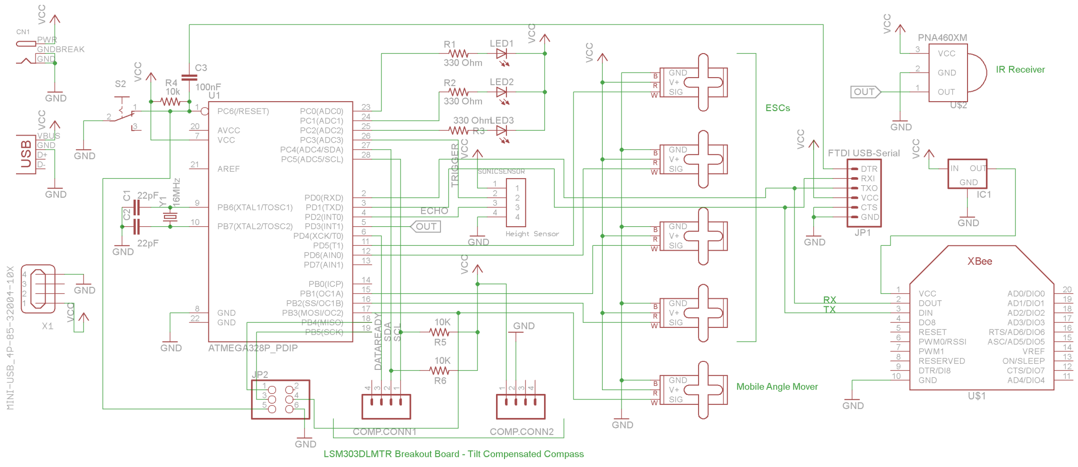


FIGURE 2.30: A circuit diagram of our prototype board design version 2.

2.15 PCB Design

The PCB was designed to be compact and at the same time to contain all the essential components for communication. There are two ways a communication link can be made with the on-board ATmega 328 on our PCB. The first method is by using XBee and the second method is by using the USB Port.

Features of the Prototype PCB

- 4 servo connectors for connecting to the Electronic Speed Controllers.
- 8 Pin connector for connecting a tilt compensated compass.
- 6 pin connector for FTDI chip to convert USB data signals to serial data.
- 4 pin connector for connecting a height measurement sensor.
- A 3 pin connector to connect the mobile tilting sensor.
- A SPI JTAG programming header.
- XBee connector for wireless communication.
- 3 LEDs to indicate the status of the AtMega microcontroller.

In the above figure, the prototype board is shown where we have used XBee to communicate with the computer and receive the serial data to control the motor speed. This is just for testing purposes and in the final design we will use the data coming from the android to determine the speed of the motors. Furthermore, the final design will use the XBee as a kill switch to completely turned off the quadcopter in case it reaches a forbidden area or any similar situation.

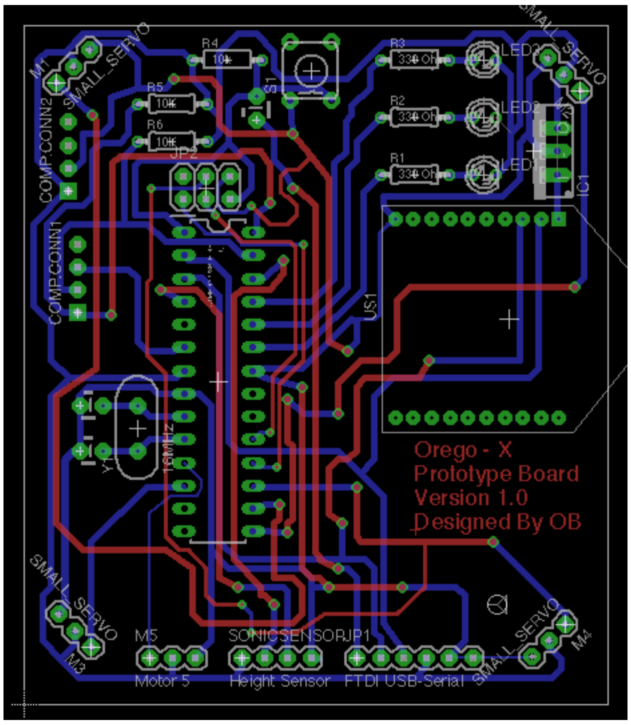


FIGURE 2.31: A x-ray vision of our prototype board design version 1.

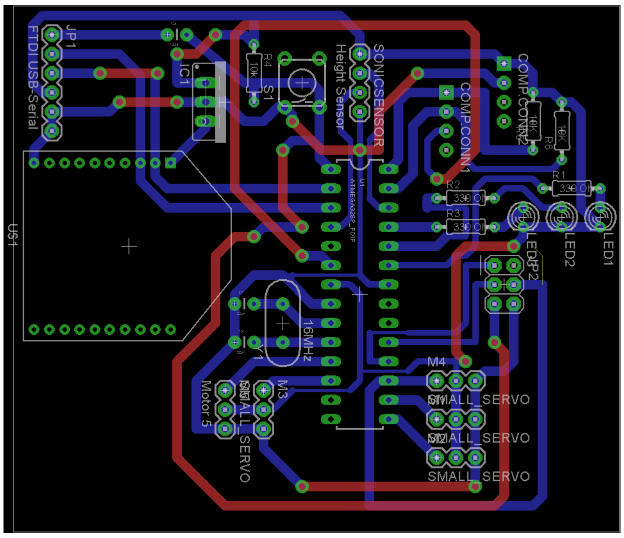


FIGURE 2.32: A x-ray vision of our prototype board design version 2.

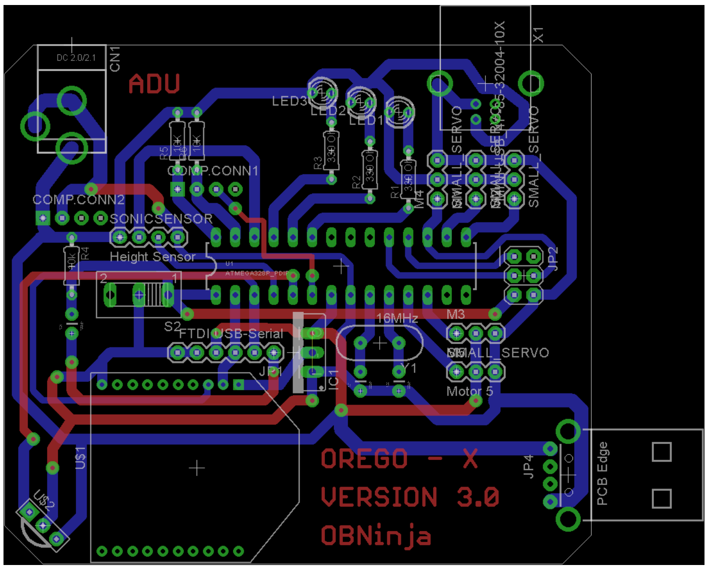


FIGURE 2.33: A x-ray vision of our prototype board design version 3.

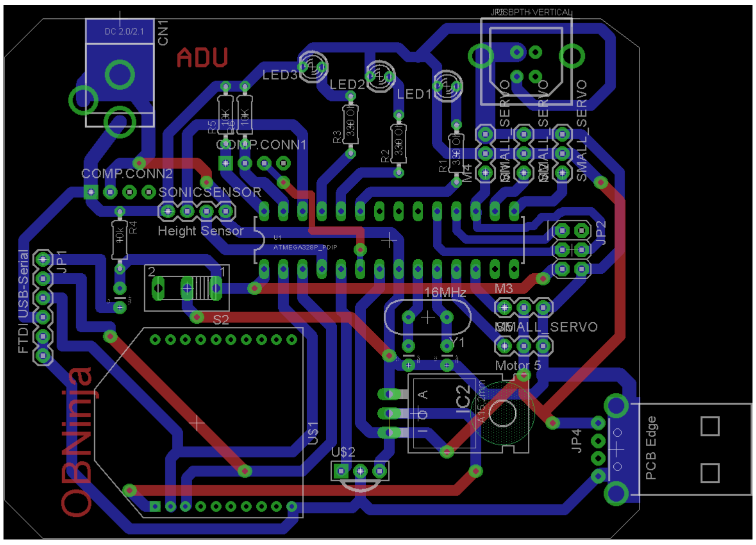


FIGURE 2.34: A x-ray vision of our final board design version 4.

2.16 Safety Design

Safety is the most important aspect when it comes to our project and our aim was not only to stop the expensive propellers form breaking down but also from hurting those standing nearby.To keep the propellers from hurting anyone a prop(eller)g-guard is often used. Some convincing designs by hobbyists that inspired our design were one made from Styrofoam and other PVC.

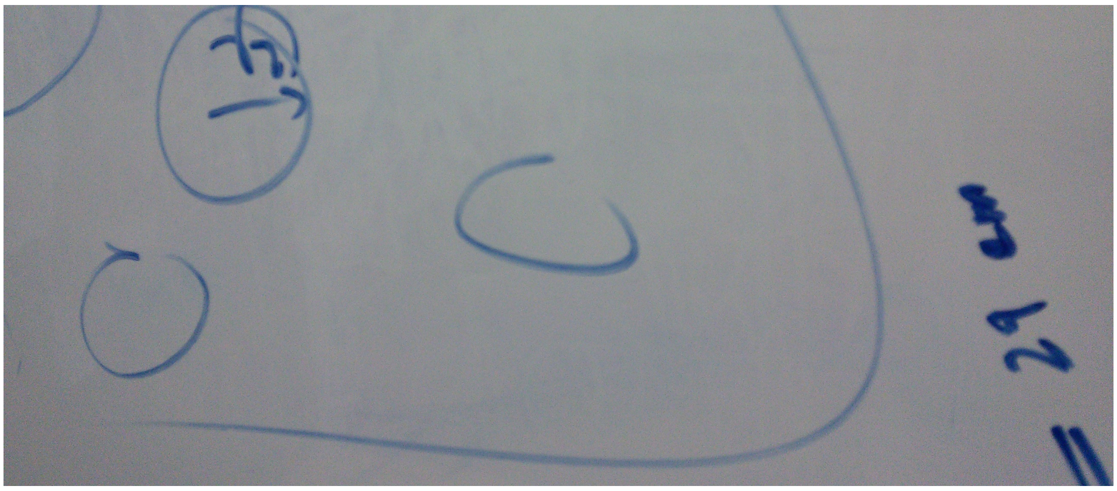


FIGURE 2.35: Prop-guard Design on White Board

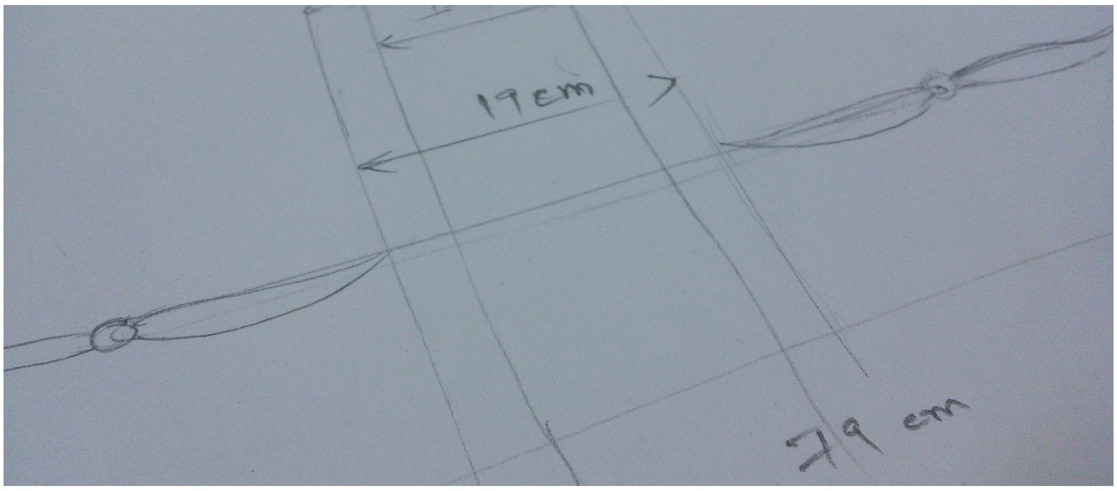


FIGURE 2.36: Prop-guard Built with Notepaper

The design went through a couple of phases;2.35, 2.36 which finally ended in a detailed auto-cad design 2.37 . Using Styrofoam lying around as waste the final prop guard was built which is yet to be tested. 2.39

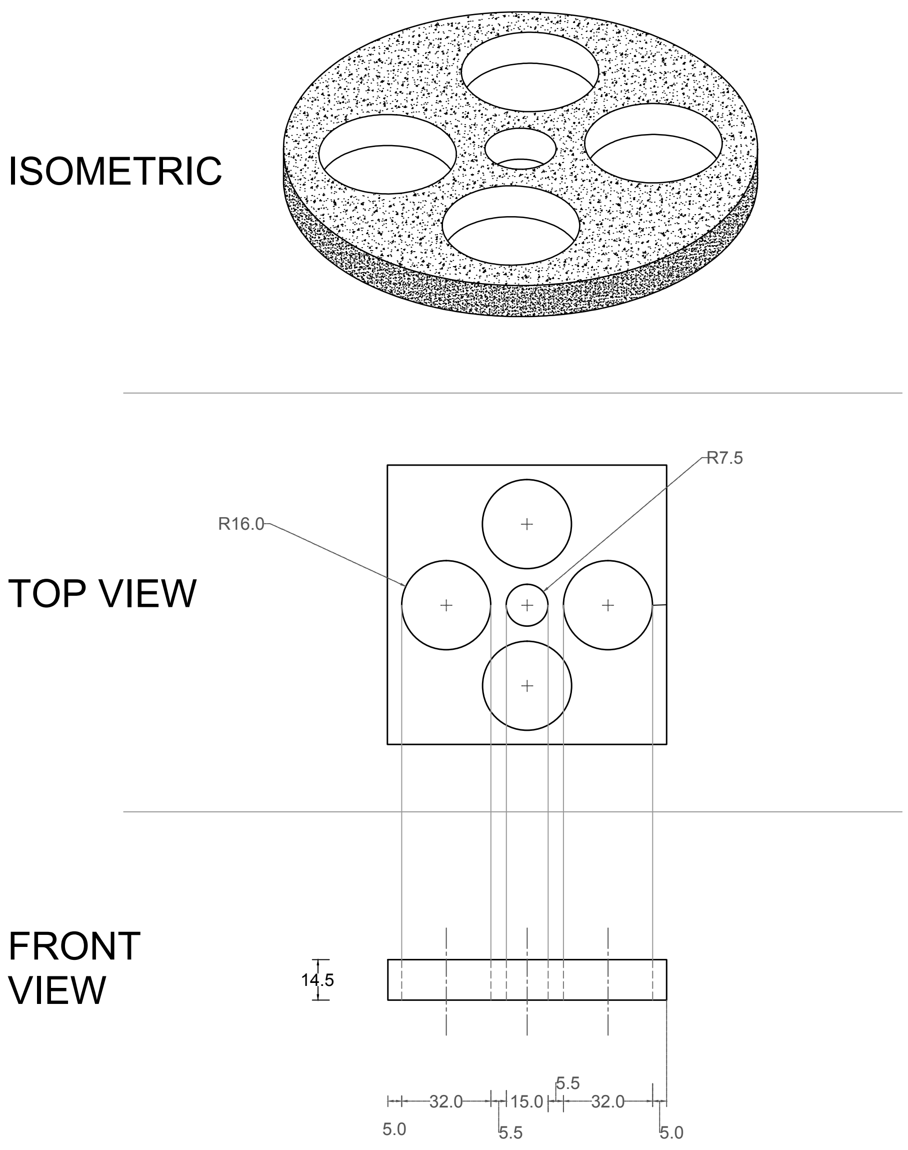


Figure 2.37: Prop-guard CAD Design

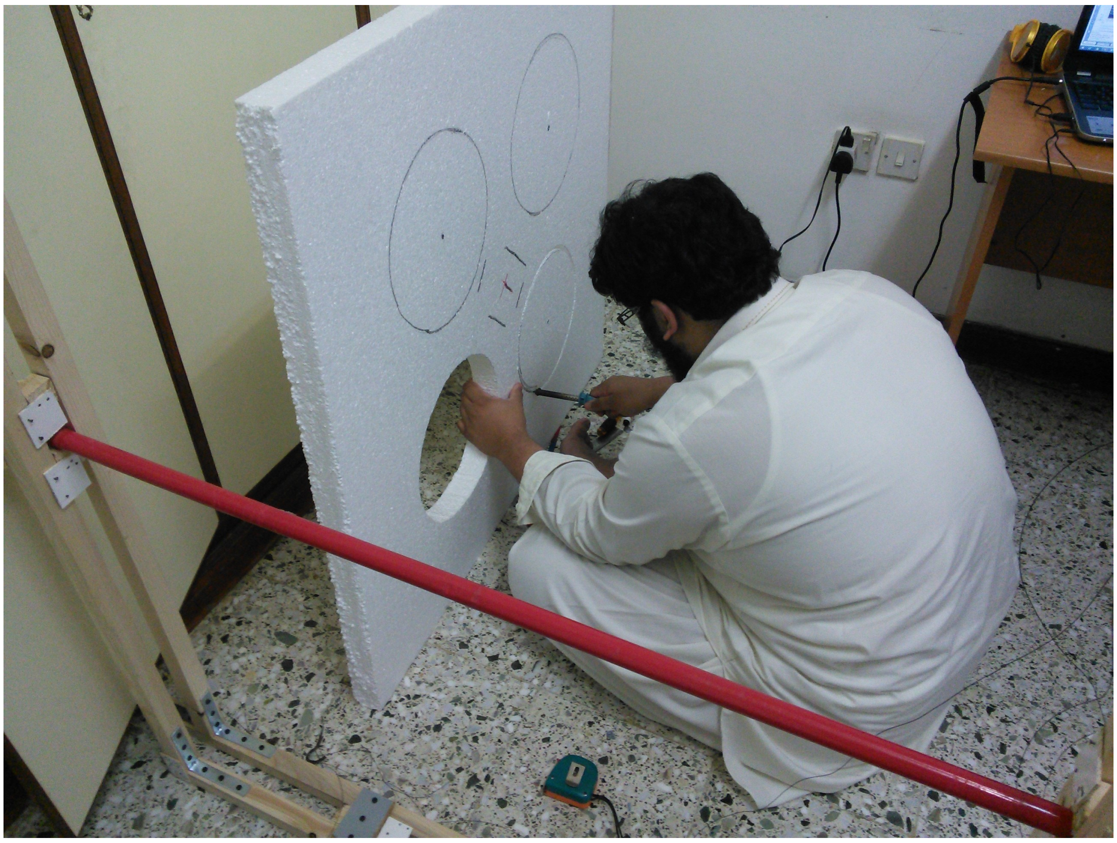


Figure 2.38: Prop-guard Cutting

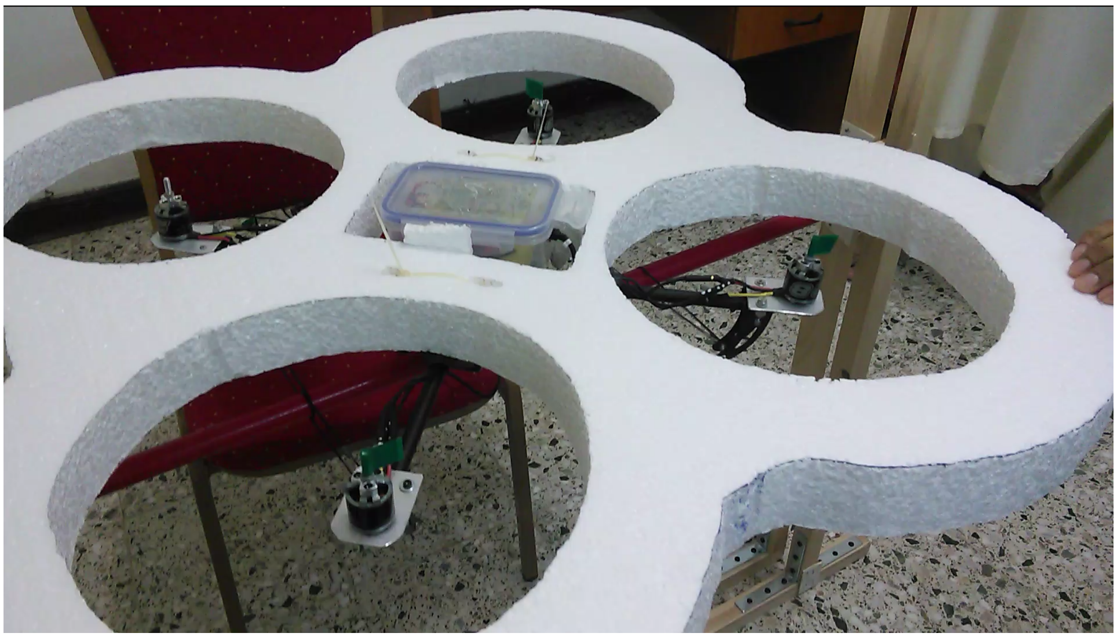


Figure 2.39: Prop-guard Built with Styrofoam

2.16.1 Emergency Button

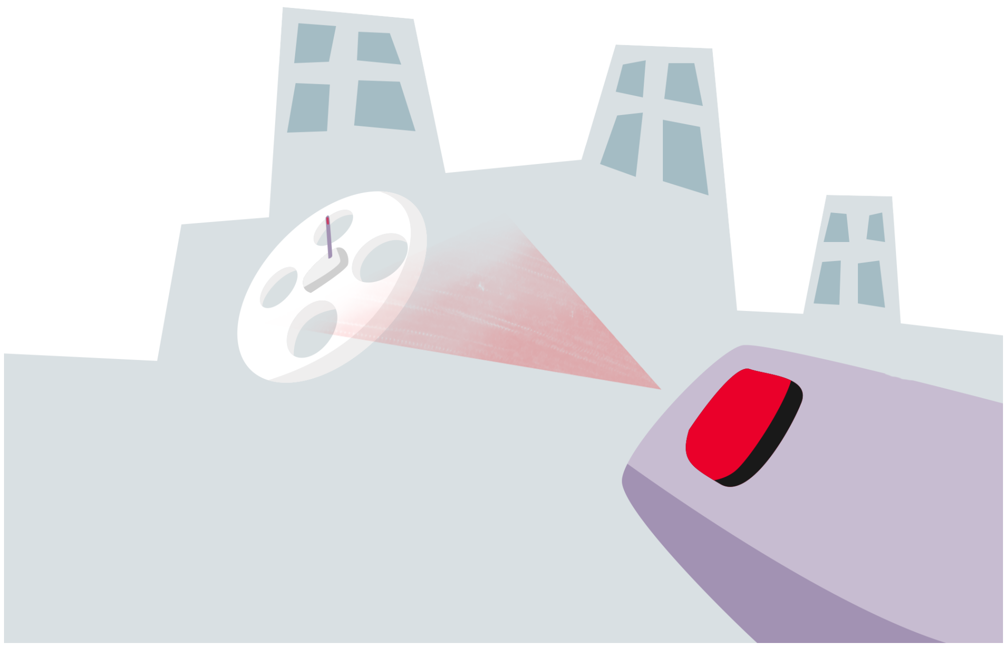


FIGURE 2.40: Emergency Button Concept

The quad-copter had a big chance of going out of control and that's when the idea of an ***Emergency Button*** was badly felt. Sullivan [1] wrote an easy to understand tutorial; Using an IR remote to Control an Arduino Project. He uses a IR receiver module and sets it up in an Arduino environment. Then he uses a IR Remote library from Github to receive data and display in the serial monitor of Arduino. Find this detailed tutorial in the pdf document *Using an IR Remote to Control an Arduino Project.*

We are using TVs in our daily life and we know how easy changing the channels through a remote control is, compared to changing the channels through buttons mounted on the actual device itself. The beauty of IR-remote is that it informs the device which button the user has pressed. This is done through the process of coding each button value and sending it through the IR LED and turning it ON or OFF in a particular manner. We cannot see the IR light as it falls well below the visible light frequency, hence the name "infra" red. However, we can see the IR light of our home TV remotes through a digital camera.

Modulating and demodulating the IR LED digitally is much more simpler and accurate than modulating and decoding it in analogue method. Analogous method of controlling can lead to errors when decoding as the brightness of IR LED is inversely proportional to the distance of the transmitter and receiver.

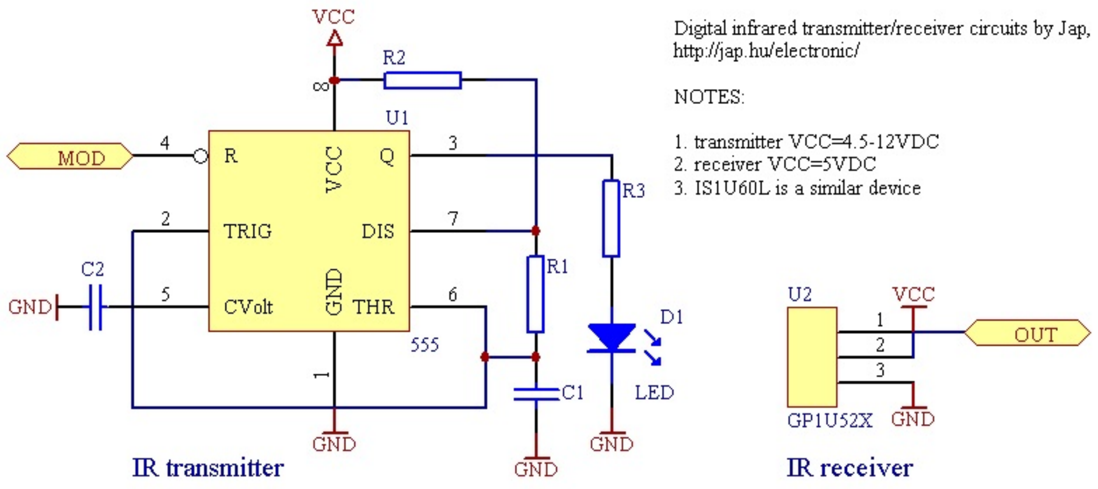


FIGURE 2.41: Peter Jakab on his website gives a schematics of electronic circuit to receive and transmit infrared pulses using IR LED and IR detector.[12] The circuit uses a 555 Timer Multivibrator IC tuned to generate timed pulses with the help of capacitor and resistor frequency selective network.

Many coding schemes have been evolving through years of research and experimentation by several companies. The manufacturer can opt for any coding scheme according to their design because as of yet, there is no standard set for the IR remote. Coding scheme depends on the micro-processor's optimum operating frequency and the number of buttons/functions on the remote. More number of functions in a IR remote leads to a longer message signal. One of the coding scheme for IR remote is RC-5 which was developed by Philips in December 1992 and is shown below in the figure.

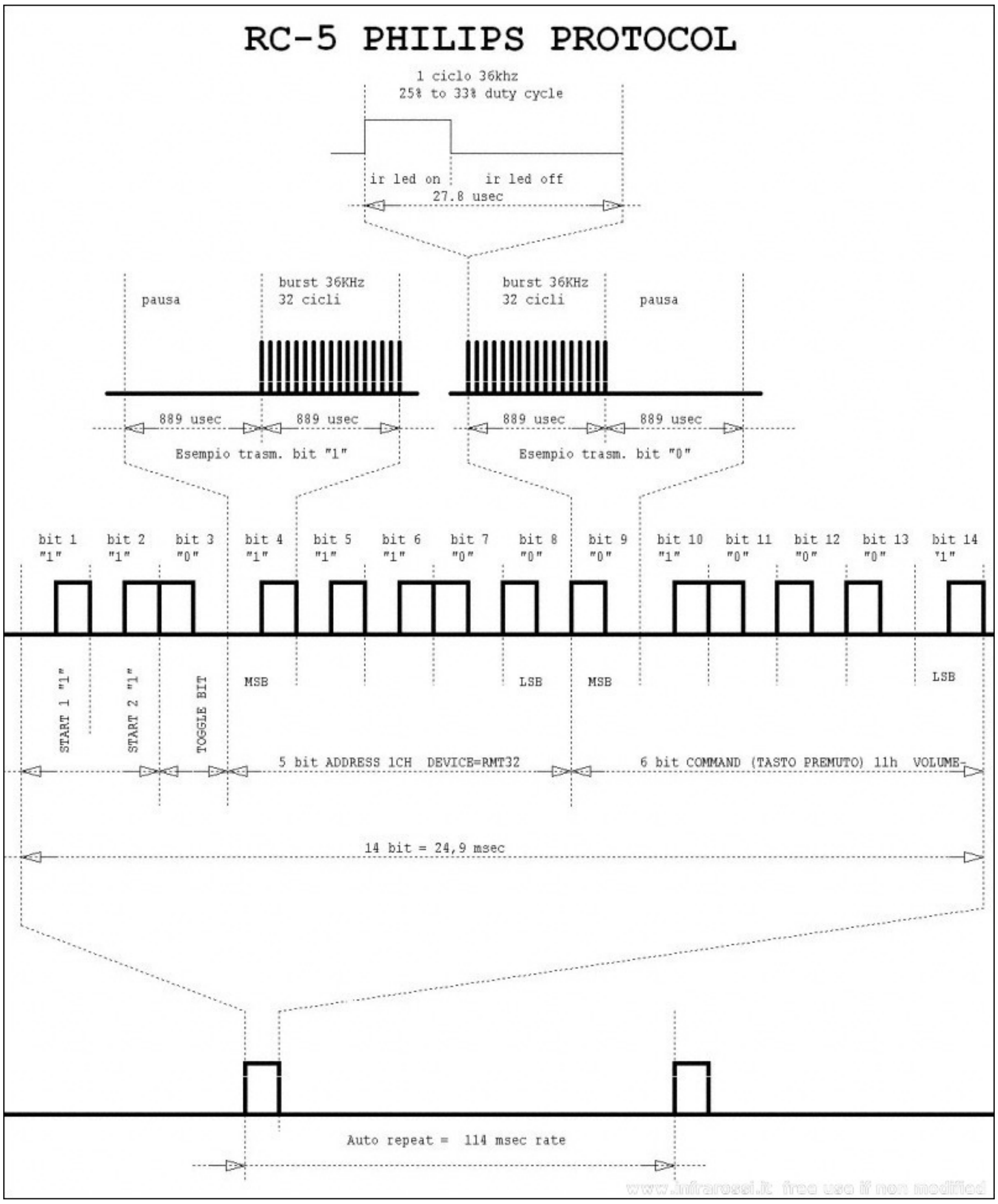


FIGURE 2.42: RC-5 Philips Protocol [7] uses pulse position modulation to send either a 1 or a 0 to the receiver. If the pulse is in the first half of 1778 micro-second then a 0 is received and if the LED is ON for the second 889 micro-seconds, then a 1 is received. The whole information packet consists of 14 bits and the starting three bits are meta-data bits and five bits after meta-data bits are to selects one of $2^5 = 32$ possible systems. The bits after that are to select one of the $2^6 = 64$ commands. Therefore, RC-5 protocol is limited to have 64 commands and 114 millisecond command transfer rate.

Following the tutorial discussed in the research we obtained a receiver module 2.44 and implemented the Arduino IR library. Any button pressed in the tv remote 2.43 will be received by the receiver that will cause an interrupt. The interrupt will invoke an emergency function which will force all the values be reset to zero. With another press it will restart the system.

2.16.2 Control Box

There are sensitive elements like the controller board, pcb and the mobile phone which needs to protected in the worst case scenario. That's where we thought of a way these could be protected in those situation 2.45. To make the emergency button more reliable a different model was thought out. In this concept we wanted the emergency remote to respond from every direction and that was when this concept came up **??**.

FIGURE 2.43: Remote for Emergency Power

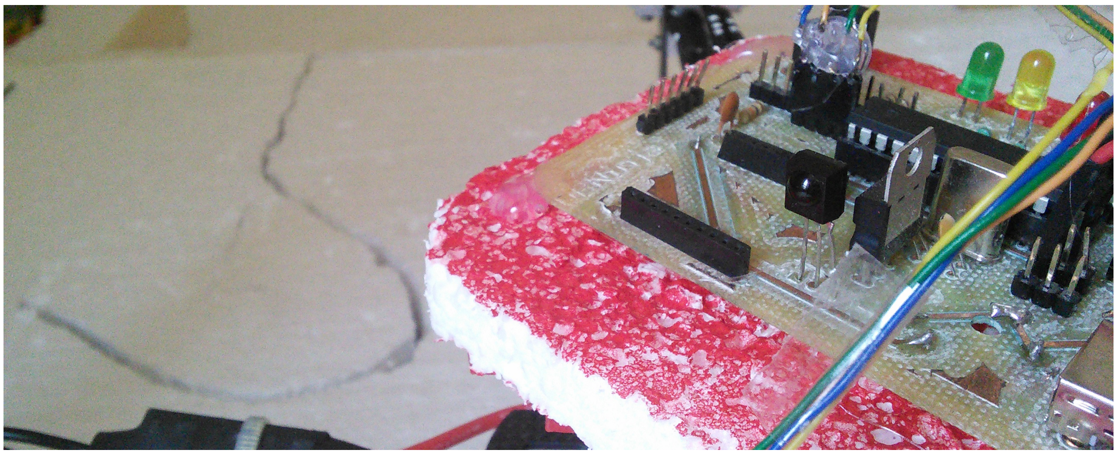


FIGURE 2.44: IR Receiver Module

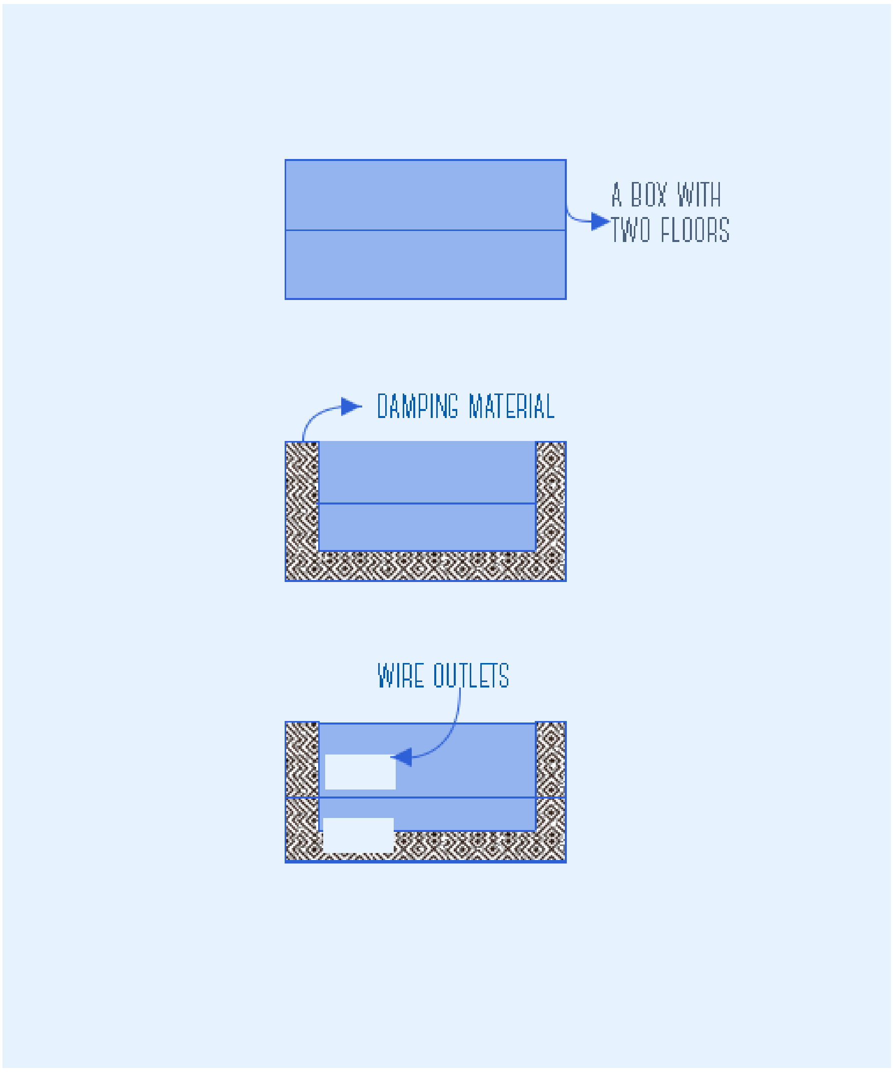


Figure 2.45: Designing the Box for Controller Board, PCB

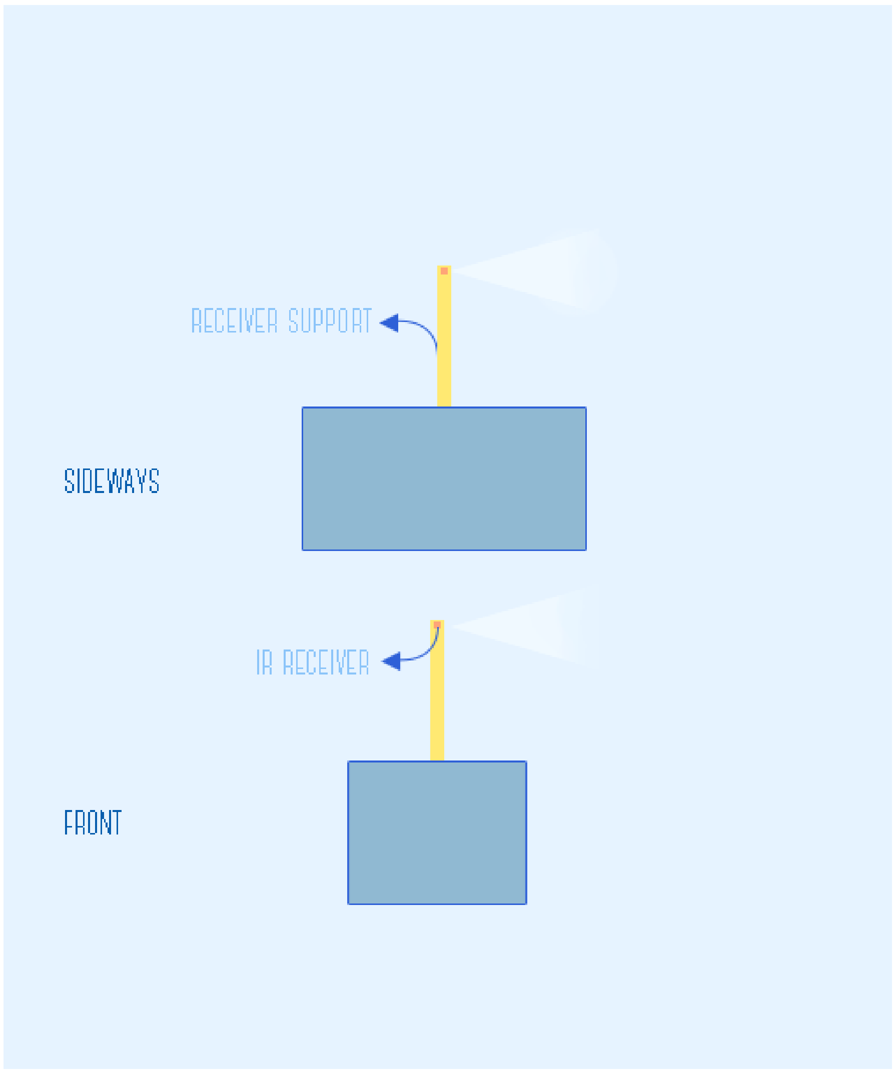


FIGURE 2.46: Improved design for the Box

Finally, the best option for this protective box was a lunch box with heavy customization 2.48, 2.47 .

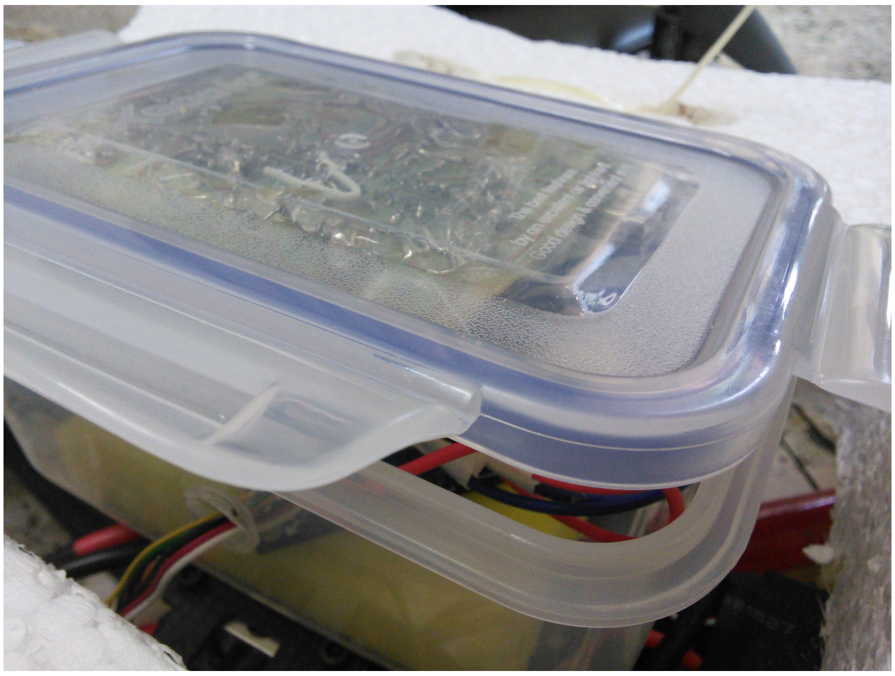


Figure 2.47: Protective Box Built

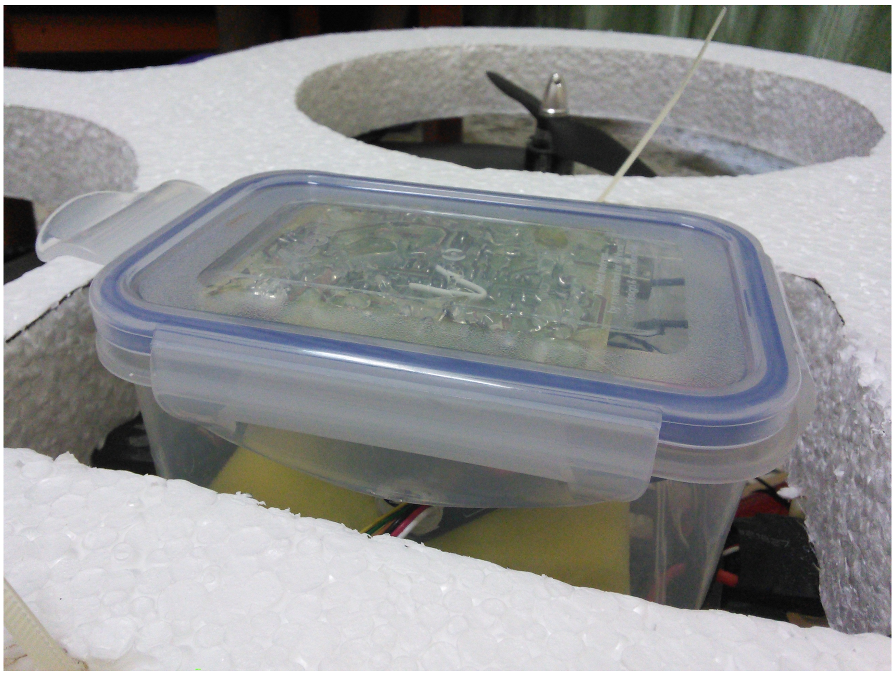


Figure 2.48: Protective Box Close Usp

2.16.3 The Wooden Frame

2.16.3.1 Introduction

The most important aspect of a project does not depend on how good the object works. It is about the safety of the project itself. The Quad Copter project required a lot of testing needed to be done in it to make sure it is safe for the public.

The Quad Copter project that we undertook is one of the most of the dangerous projects ever undertaken in the history of Abu Dhabi University. The casualties that happened during the course of the trial period included a broken finger and numerous shaken hearts. This prompted the team members to come up with a method to make sure the Quad Copter could be tested safely. Thus came the idea of the "Wooden Frame".

2.16.3.2 Use

The "Wooden Frame" is designed with the intention of keeping the quad copter under control: in other words, to make sure the Quad Copter does not fly out of control and harm the testers. The introduction of the wooden frame gave the testers the peace of mind to try out all the features of the Quad Copter.

2.16.3.3 Materials Used

- Wood Sticks (1.6 meter in length).
- L Shaped Metal Brackets.
- Wood Screws.
- Long Thin Aluminum Tube (Handle of a broom/ mop).
- Nylon Strings.

2.16.3.4 Structure

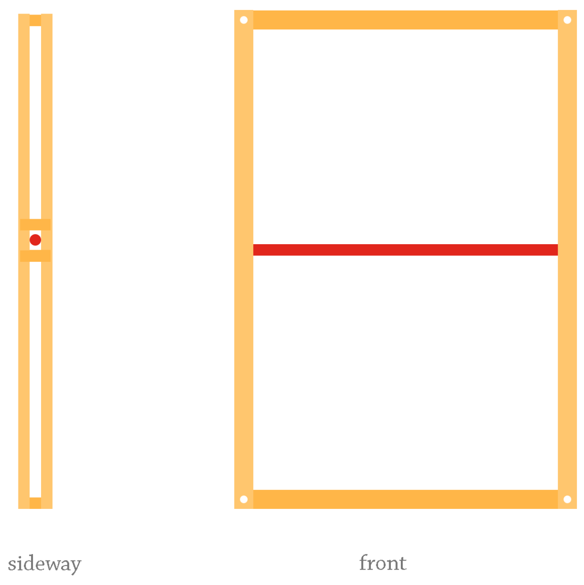


FIGURE 2.49: Frame Design

The Wooden Frame is made up of various wooden sticks cut to a length of 1.6 meters. The wooden sticks are connected with each other to form two rectangular shapes.

The Wooden sticks are attached perpendicular to each other with the help of L Shaped Metal Brackets and wood screws. Additional wooden pieces were hammered with nails on the junctions to improve the strength of the joints.

The two rectangular shapes are then connected with each other keeping a slight gap in between the rectangular frames. The gap allows a light Aluminum tube to move up and down the rectangular figure. This ensures ONLY vertical motion.

Strings from the four ends of the Rectangular frame are connected to the wall of the room in which the Quad Copter was tested. The strings ensured the Wooden Frame was stable and did not fall down.

The quad copter is mounted on the Aluminum tube with strong polyester strings. The Quad Copter is free to rotate along the axis of the Aluminum tube. Small wood pieces are attached at the end of the tube to avoid it from coming out of the gap.

2.16.3.5 Problems Faced

The biggest problem we faced due to the use of the wooden frame is the difficulty of the quad copter flying. The use of the aluminum tube means the Quad Copter has to fly an extra amount of weight – which it found was quite difficult.

Chapter 3

Implementation

3.1 Balancing the Quad-copter

Although the controller board is programmed with the task of keeping the quad-copter balanced after tuning the gyros the quad-copter was still unable to balance itself. There after we had to put the copter inside a wooden frame to start finding the value for balancing 3.11, ??, 3.13, ??. We discovered that it balances at the following values:

```
Aileron.write(93);
Elevation.write(93);
Thrust.write(57);
Rudder.write(93);
```

3.2 Custom Circuit Board

The job of the circuit board is to process the packets coming from the socket or FTDI chip and produce appropriate PWM signals for the controller board. The 7 byte packets coming from the socket or FTDI chips are first checked for validity by comparing the first and last byte. Then the state byte is processed and the quadcopter is put in the required state.

3.3 Setting up Quadcopter for flight

It is of utmost importance that the components of the quadcopter be turned ON and checked in proper order otherwise the motors can receive garbage values and respond in an unexpected way.

1. Make sure the Smart-phone is connected to the same network as ground station or in case of 4G, internet is connected.

2. Connect the Smart-phone to the FTDI Chip using the USB (male - male) connector.
3. Once the ESCs are programmed (Beeps are finished and red LED is ON in controller board), start up the ground base application.
4. Enter the IP Address and port number in the ground application after checking from the Smart-phone interface.
5. Click Connect button and check the status on Smart-phone. It should say "Connected" in Connection Status.
6. Make sure the quadcopter is in emergency mode by checking the Mode in Smart-phone.
7. Take the hand as far as possible from the Leap motion sensor.
8. Press on Guesture Control button.
9. Slowly bring the hand near the sensor and enjoy flying with hands.

3.4 Final Product

The final product includes the following components:-

1. Carbon Fibre Body Quadcopter.
2. Leap Motion Device.
3. Leap Motion USB Cable.
4. Orego X Desktop Application.
5. Orego X Callibration Application.
6. Android Server Application.
7. USB mini to micro (male-male) Cable.
8. Custom built Decoding PCB.
9. HobbyKing Controller Board.
10. 5V FTDI USB-Serial converter chip.

3.5 PCB

The printed circuit board consist of an Atmega328P that sends out four pwm values to the controller board using and communicates with peripheral devices through tx and rx connected to an FTDI outlet.It is also built with the capability of sonar and ir sensor for height sensing and emergency button respectively. ??, 3.3

3.6 Frame

The frame served two important purpose; safety and debugging. Due to the dramatic nature of esc it was very scary to come close for checking simple things like whether all the motors are responding or what was the direction of the air flow. Once it was strapped inside the frame the fear was much less and debugging process easily made progress. ??, 3.18, ??, 3.20, ??, 3.22, ??, 3.24, ??.

3.7 Module Requirement and Performance

The different modules that are on play are the following:

- Transmitter
- Receiver
- Control Board
- ESC
- Motor

3.7.1 Transmitter

3.7.1.1 C Sharp Ground Application

After designing the interface for the application, we tested the flight and control using the C Sharp application. There were several problems in the sensitivity of the hand gestures. We fixed these problems by remapping and tuning the values sensed by the leap sensor.

3.7.1.2 Remapping the Pitch

The problem with the pitch was that it had to be remapped from range -90 to +90 to range 87 to 97. 87 to 97 range was selected because 93 was the middle value found out by experimentation and even a small change from this value causes the quadcopter to tilt a lot.

```
/*
the remapping done for Pitch...
*/
Elevation = (int)pitchVal+93;
Elevation = (int)(((double)Elevation / 180) * 10) + 87;
```

The remapping equation becomes:

$$P_{remapped} = \left(\frac{P_{raw} + 93}{180} \times 10 \right) + 87 \tag{3.1}$$

The minimum value the formula will produce is:

$$P_{min} = 87 \tag{3.2}$$

The maximum value the formula will produce is:

$$P_{max} = 97 \tag{3.3}$$

3.7.1.3 Remapping the Roll

There were two problems with roll, the first one was that it had to be remapped from range -90 to +90 to range 87 to 97. The second problem was that the range needed to be reversed. In essence, the left tilt of the hand should produce higher than 93 value and right tilt of the hand should produce lower than 93 values and not the other way around.

```
/*
the remapping done for Roll...
*/
Aileron = (int)rollVal + 93;
Aileron = (int)(97 - (((double)Aileron / 180) * 10));
```

The remapping equation becomes:

$$R_{remapped} = 97 - \left(\frac{R_{raw} + 93}{180} \times 10 \right) \tag{3.4}$$

The minimum value the formula will produce is:

$$R_{min} = 87 \tag{3.5}$$

The maximum value the formula will produce is:

$$R_{max} = 97 \tag{3.6}$$

3.7.1.4 Remapping the Thrust

Leap motion device detects hand up to 700 cm vertical distance away from the sensor. As the hand goes away from the sensor, the accuracy of the finger positions and other parameters goes down. Since, higher accuracy is needed at high speeds and at lower speeds accuracy can be compromised, the control for thrust was reversed. Although leap sensor is capable of sensing distances up to 700cm, a margin of 200cm was kept and the acceptable range was made from 0-500.

Moving the hand away from the sensor makes the thrust low and moving the hands closer to the sensor increases the thrust. Therefore there are two requirements the mathematical formula should suffice.

```
/*
the remapping done for Thrust...
*/
Thrust = (int) ((((500.0 - hand.PalmPosition.y) / 500.0) * 100.0)
+ 50.0);
```

1. Re-map the input from the leap sensor from range 0-700 to range 50-150.

2. Reverse the range (700 maps to 50 and 0 maps to 150).

The final equation becomes:

$$T_{remapped} = \left(\frac{500 - T_{raw}}{500} \times 100 \right) + 50 \tag{3.7}$$

The minimum value the formula will produce is:

$$T_{min} = 50 \tag{3.8}$$

The maximum value the formula will produce is:

$$T_{max} = 150 \tag{3.9}$$

3.7.1.5 Emergency

During the course of flight testing we had to find reassurance in the emergency button, as discussed in design we implemented an IR sensor that when sensed will trigger an interrupt. In this case 3.7 , ?? we are trying to turn off the quad copter when there is any hint of it going out of control.

3.7.2 Control Board

To stabilize the quadcopter, a HobbyKing controller board was tuned and used to control the motor speeds at all time. HobbyKing Multi-Rotor Control Board V3.0 (Atmega328 PA) uses three Murata Piezo gyroscopes to measure acceleration or change in angle in three directions, pitch, roll and yaw.

The output signal from the gyroscopes is multiplied by the potentiometers or "pots" to reduce or increase the sensitivity of the signal. The controller board contains a simple voltage divider circuit to divide the voltage in the range 0-5V. This voltage is then send to the on-board AtMega micro-controller's ADC (Analogue to Digital Converter) pin.

It was necessary to tune these three gyroscopes for a stable flight. Several boom tests were done to ensure that the quadcopter balances correctly.

3.7.2.1 Balancing the Quad-copter

Although the controller board is programmed with the task of keeping the quad-copter balanced after tuning the gyros the quad-copter was still unable to balance itself. There after we had to put the copter inside a wooden frame to start finding the value for balancing 3.11, ??, 3.13, ??. We discovered that it balances at the following values:

```
Aileron.write(93);
Elevation.write(93);
Thrust.write(57);
Rudder.write(93);
```

3.7.2.2 Tuning Elevation

This was an important step in balancing the quadcopter as it determines the front and back motion of the quadcopter in air. The front and back motors should respond to different values sent and should tilt the quadcopter in forward direction if higher or tilt backwards if lower than the medium value.

Step 1: Tie the quadcopter from the right and left motor sides.

Step 2: Establish either USB (Serial) or socket communication using the Orego-X Calibration Application.

Step 3: As the thrust of 58 just starts the motors, and it can go up to 140 value, slowly increase the thrust up to about 90 value.

Step 4: Now change the value of elevation in the numeric up-down box and study the effect.

Step 5: Tune the value until the quadcopter balances out and both brushless motors are at the same speed.

3.7.3 Tuning Aileron

Tuning the aileron determines how the quadcopter moves left and right in the air. The left and right motors should respond to different values sent and should tilt the quadcopter in right direction if higher or tilt left if lower than the medium value.

Step 1: Tie the quadcopter from the front and back motor sides.

Step 2: Establish either USB (Serial) or socket communication using the Orego-X Calibration Application.

Step 3: As the thrust of 58 just starts the motors, and it can go up to 140 value, slowly increase the thrust up to about 90 value.

Step 4: Now change the value of aileron in the numeric up-down box and study the effect.

Step 5: Tune the value until the quadcopter balances out and both brushless motors are at the same speed.

3.7.4 Tuning Rudder

Rudder determines the spinning motion of the whole quadcopter body in the z-axis. This is also often known as azimuth angle. Correcting or balancing the rudder is quite tricky as it is difficult to restrict all other motions of the quadcopter and allow only azimuth rotation. This was done actual flight tests.

3.7.4.1 Blocking Rudder Change

Several tests and crashes from different drivers gave the feedback of difficulty in control because of Azimuth angle of hand. The azimuth angle of hand determined the rudder of the quadcopter.

The Quad-copter can be manouvred in all directions using only Pitch, Roll, and Thrust so the rudder value sent from the ground application while in gesture control mode is always in the middle (93). This heavily improved the hand gesture flight control.

3.7.5 ESC

The ESC takes the ppm values from the controller board and controls the current flow between the LiPo battery and the brush-less motor accordingly. The ESC must be equipped with the proper contacts to receive the ppm values otherwise it might not arm. After making sure the soldering contacts are robust we made it safer using shrink tubes and thereafter it performed perfectly.

3.7.5.1 Propeller

3.8 Air Flow

One important condition of the quad-copter to lift off the ground is that the air flow generated by the propellers should be directed to the ground so that it may force itself upward, however as common sense as it may sound it quite difficult to make sure that the air flow is downward. A simple trick that we learn was to take off the propellers and put some tape in its place **??**, 3.15, **??**. This way its both safe and one can put finger in those tape to asses the direction of the the propeller. Later on we discovered another technique to achieve this purpose. Put the propellers on but run the motor at slow speed. Throw some paper on top of it and notice the direction of the path of the paper. If it is pushed back then the air is pushed upward but if it is sucked in than the air is pushing downward.

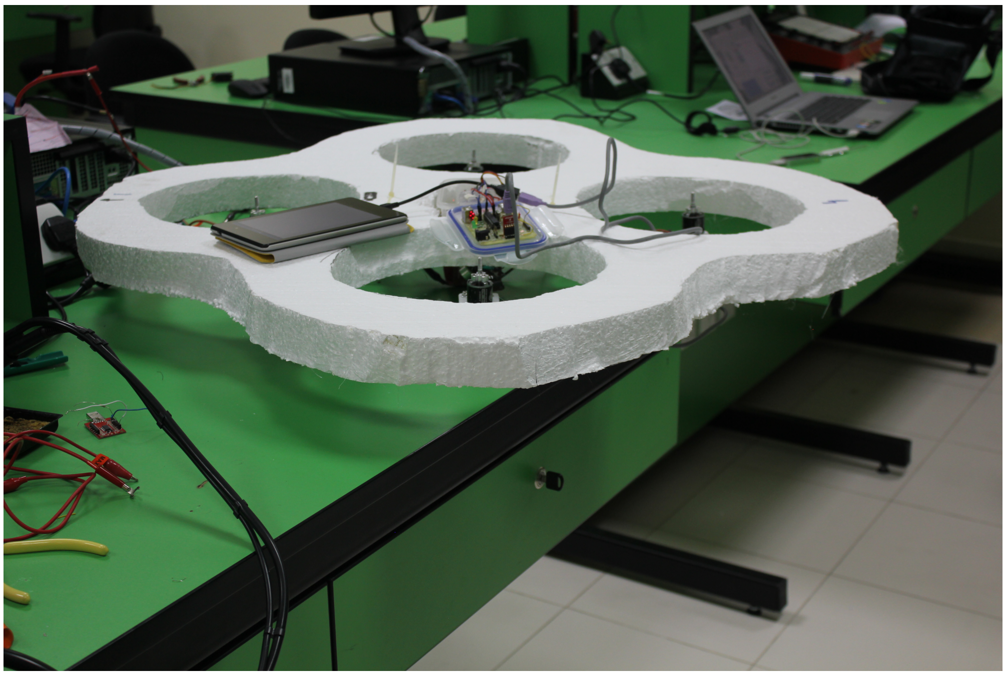


FIGURE 3.1: Quad-Copter Testing

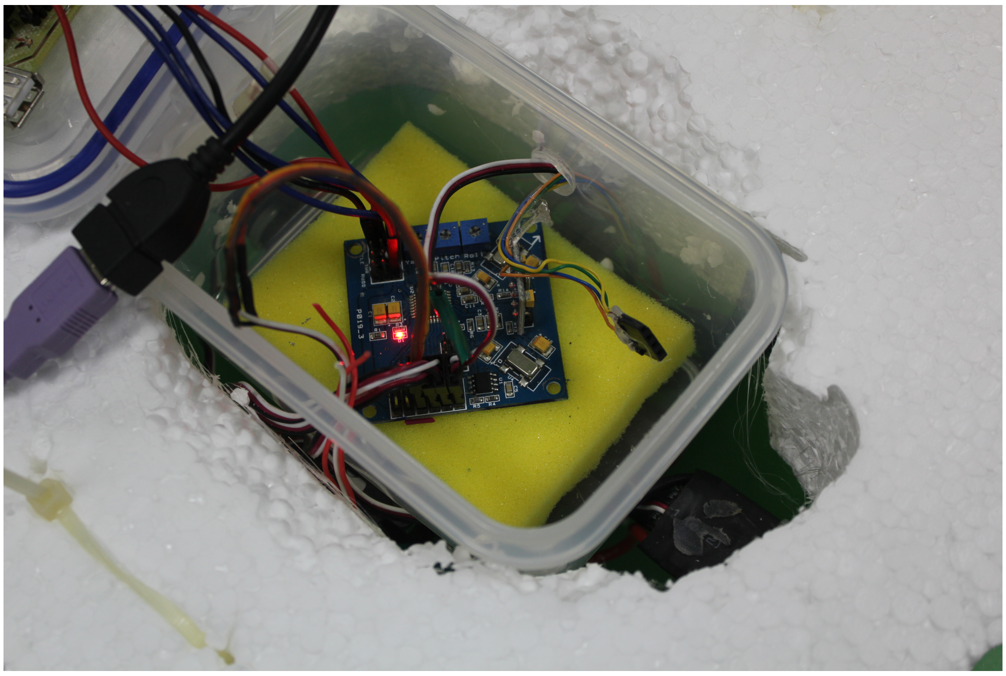


FIGURE 3.2: ESC Armed Indication

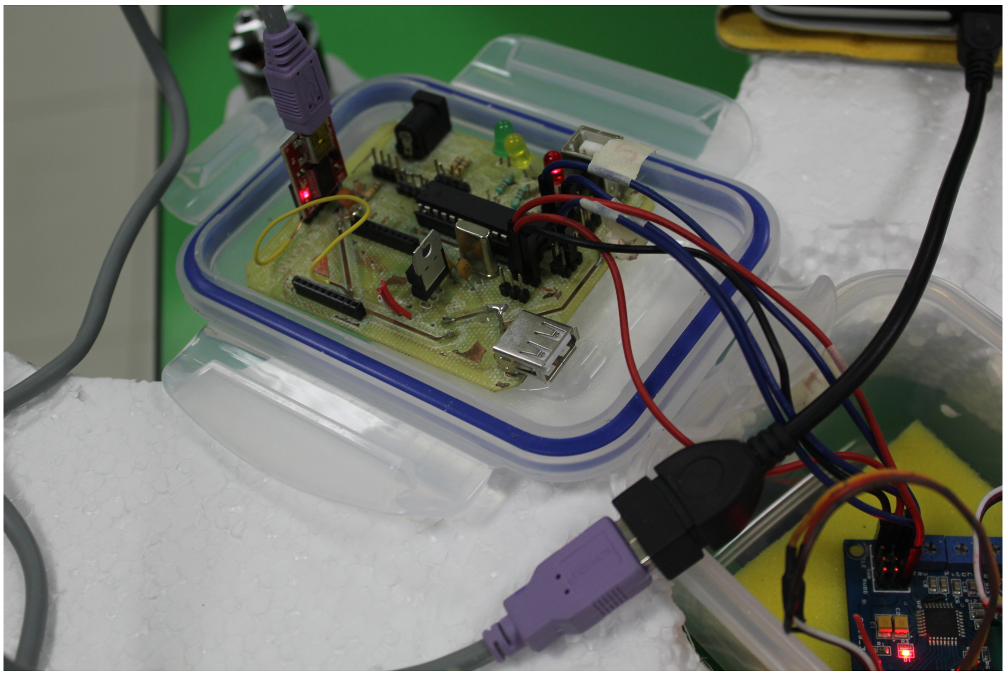


Figure 3.3: PCB Acting as the Receiver

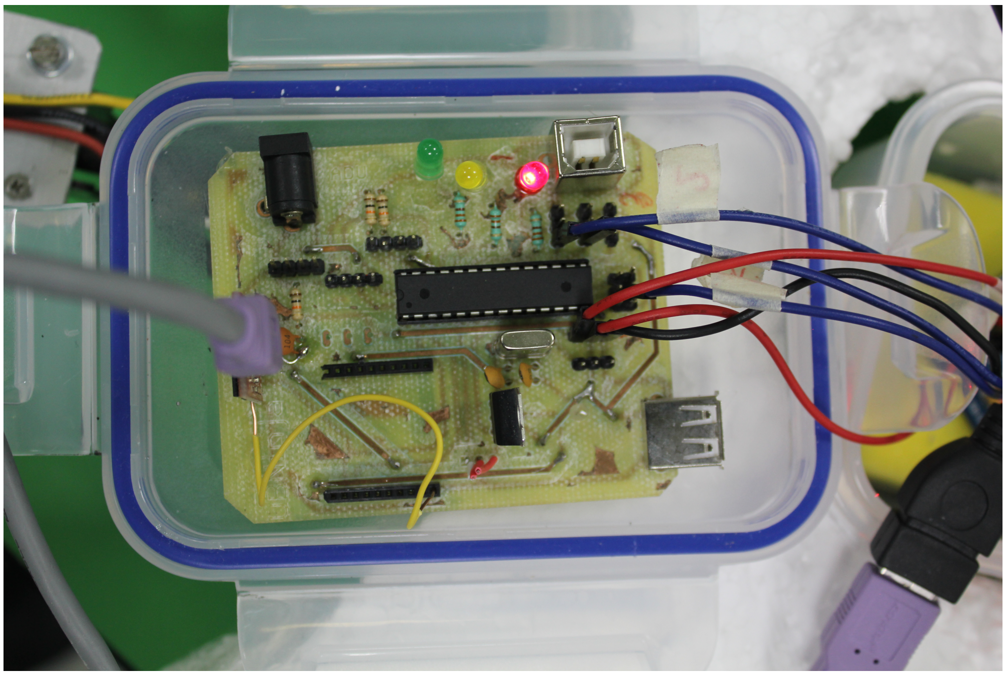


Figure 3.4: PCB with USB Plugged

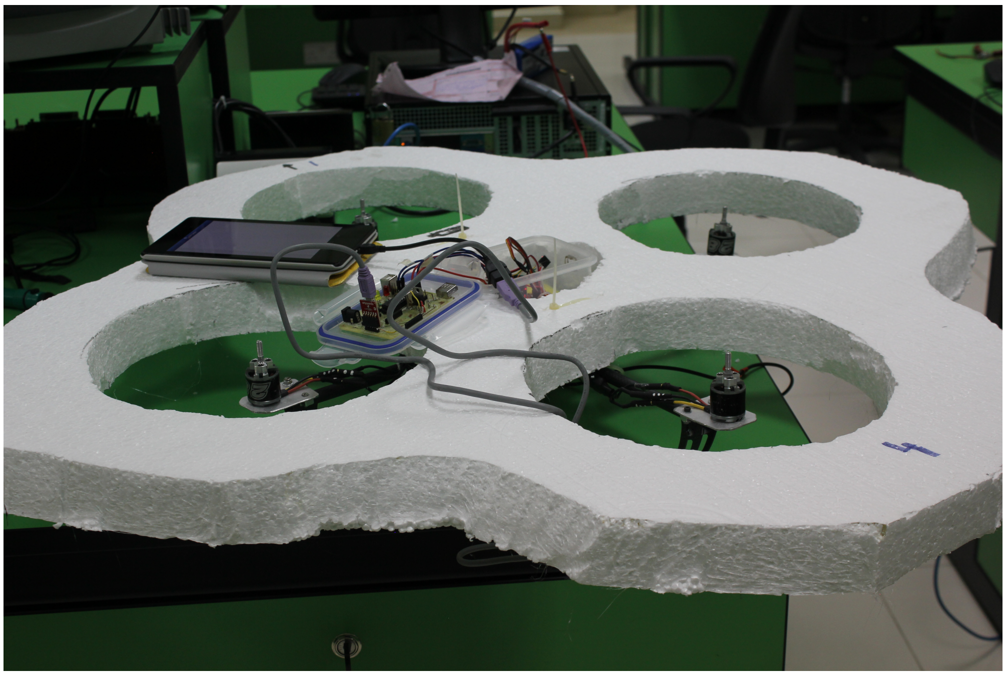


FIGURE 3.5: Checking Motor Response

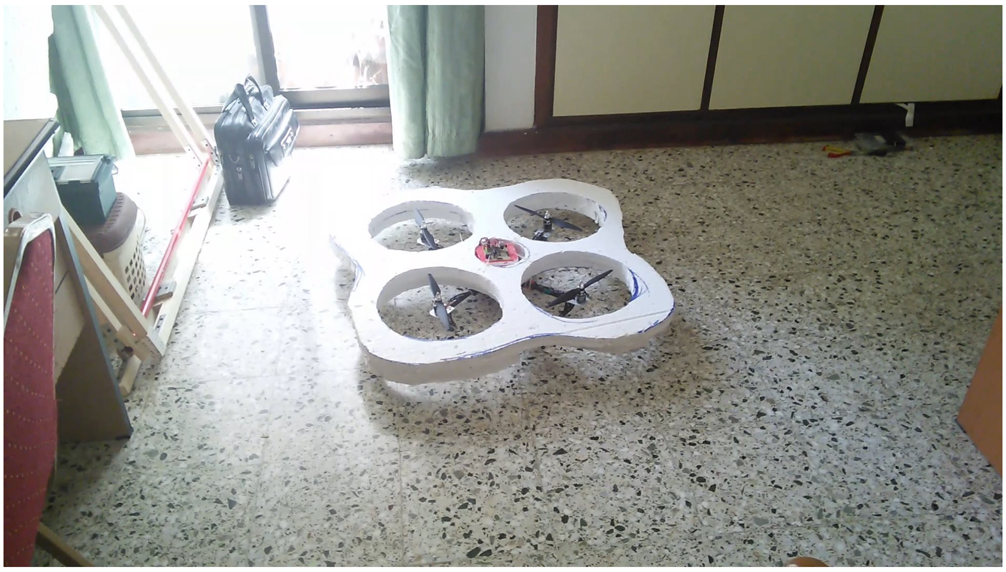


FIGURE 3.6: Test Flight in Home

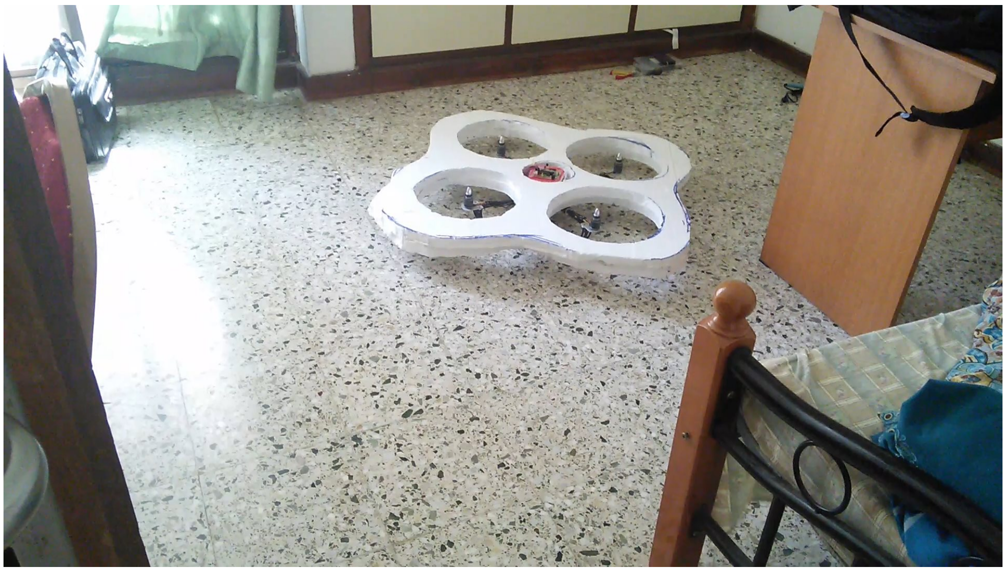


FIGURE 3.7: Test Flight in Home

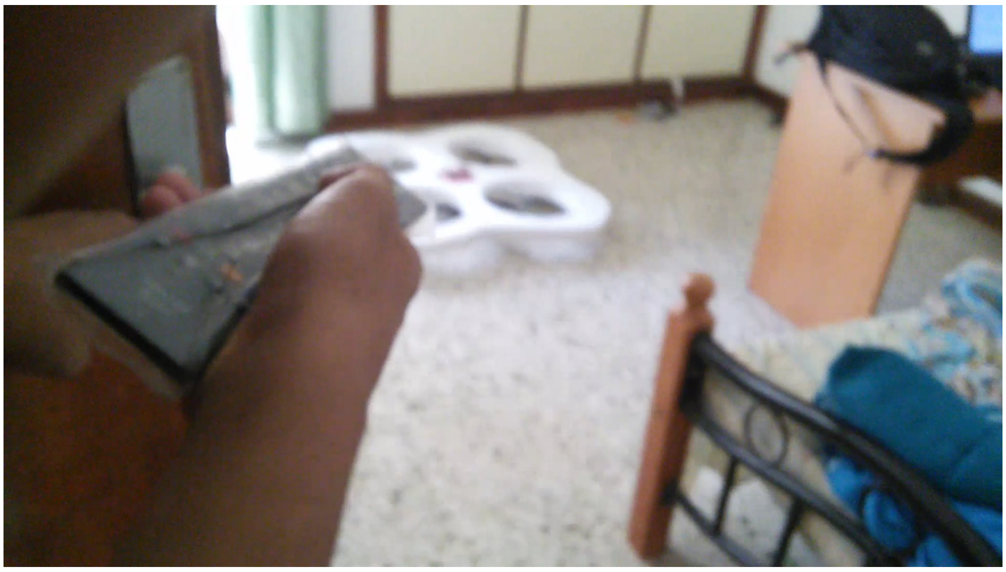


FIGURE 3.8: Emergency Button Test

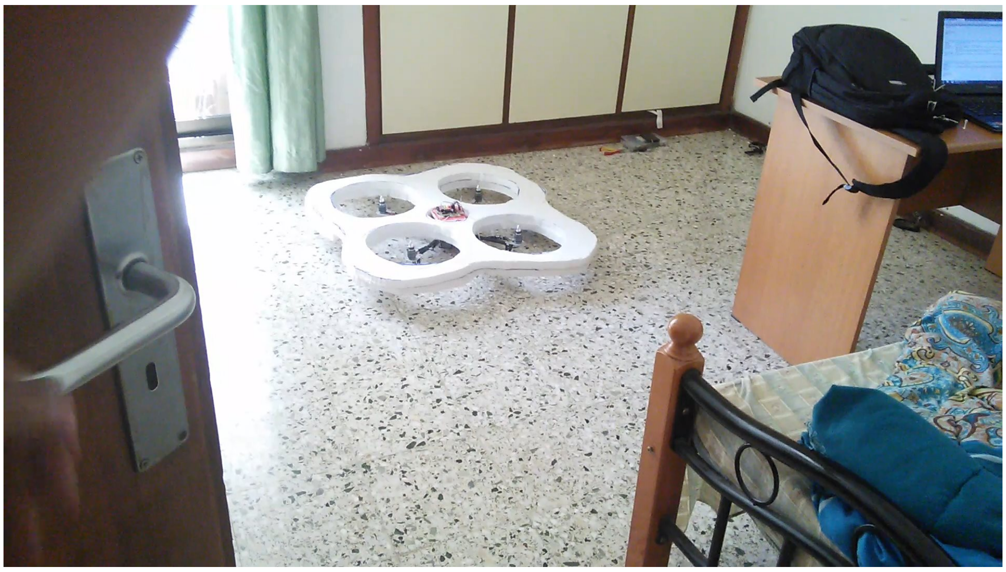


FIGURE 3.9: Test Flight in Home

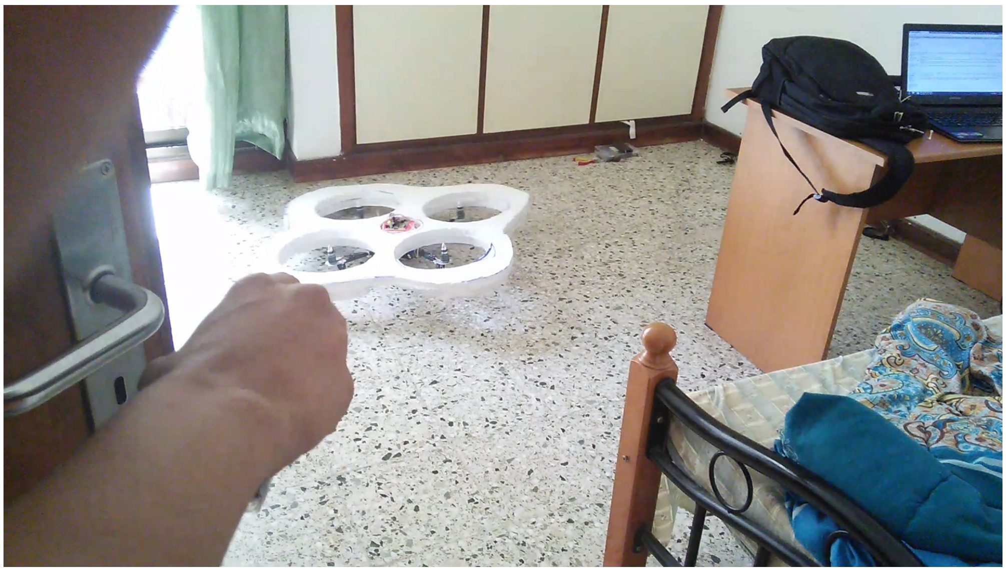


FIGURE 3.10: Emergency Button Test

Figure 3.11: Emergency Button Test

Figure 3.12: Balancing the Quadcopter

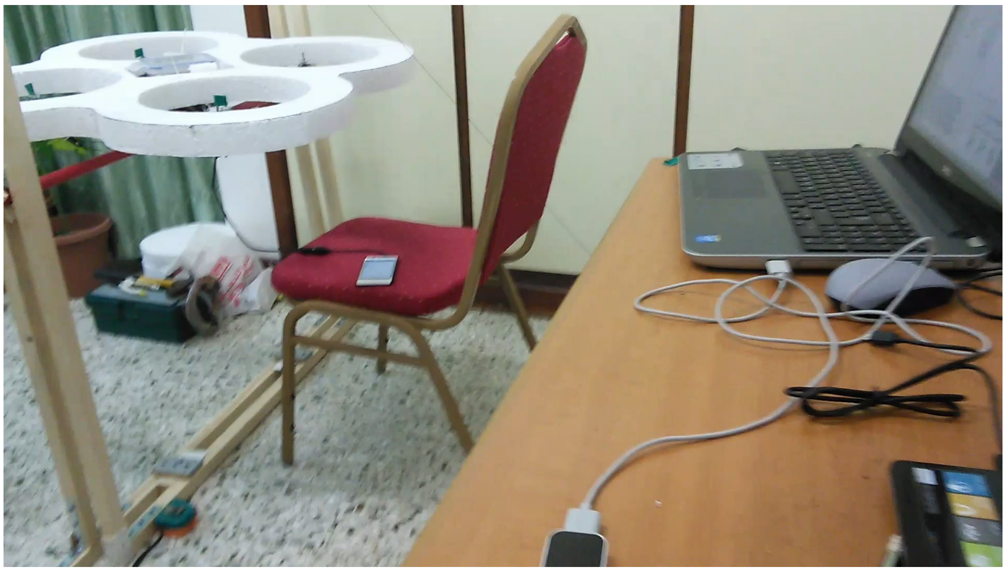


Figure 3.13: Quad-copter Balanced

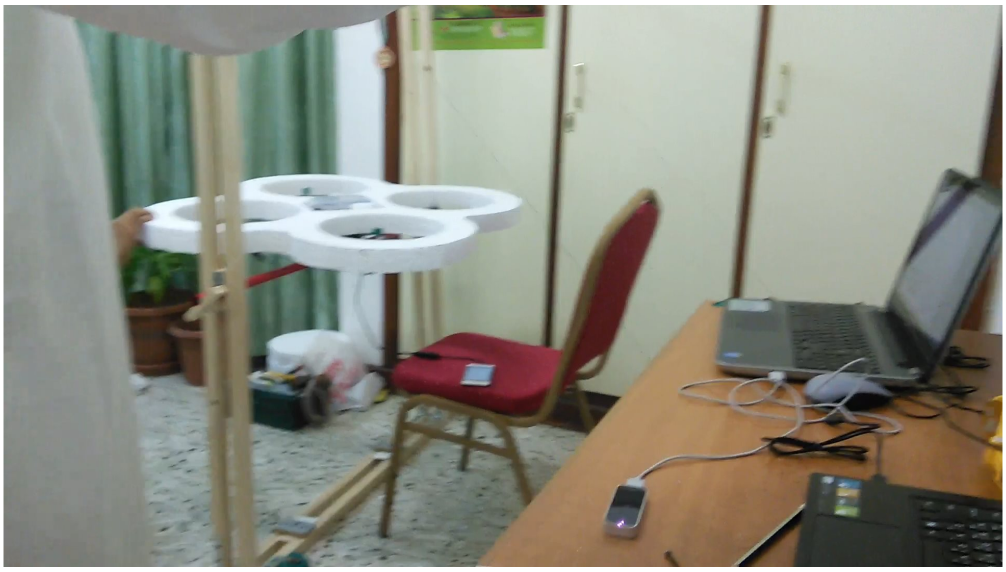


Figure 3.14: Hand-Gesture Mode in Quad-copter

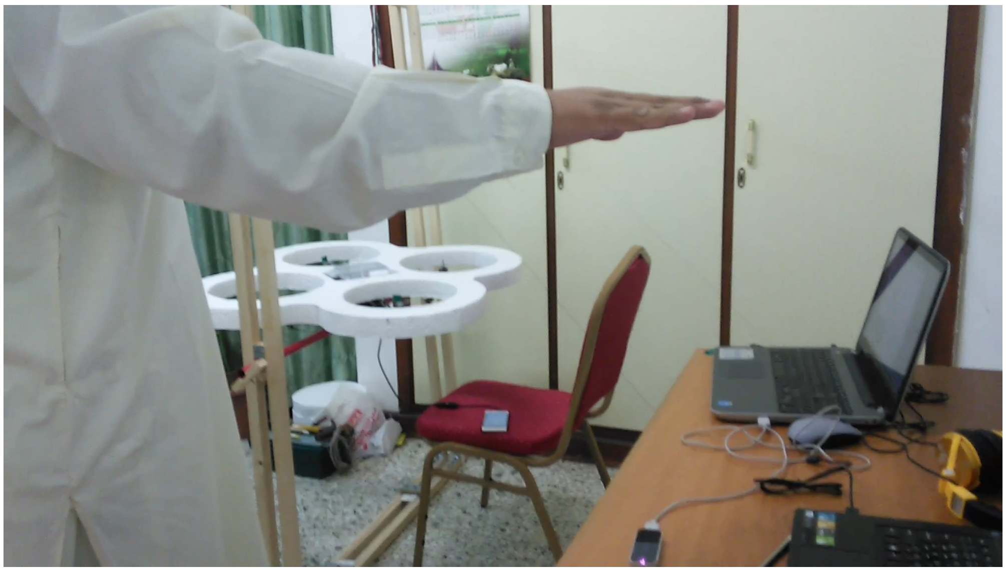


FIGURE 3.15: Hand-Gesture Mode in Quad-copter

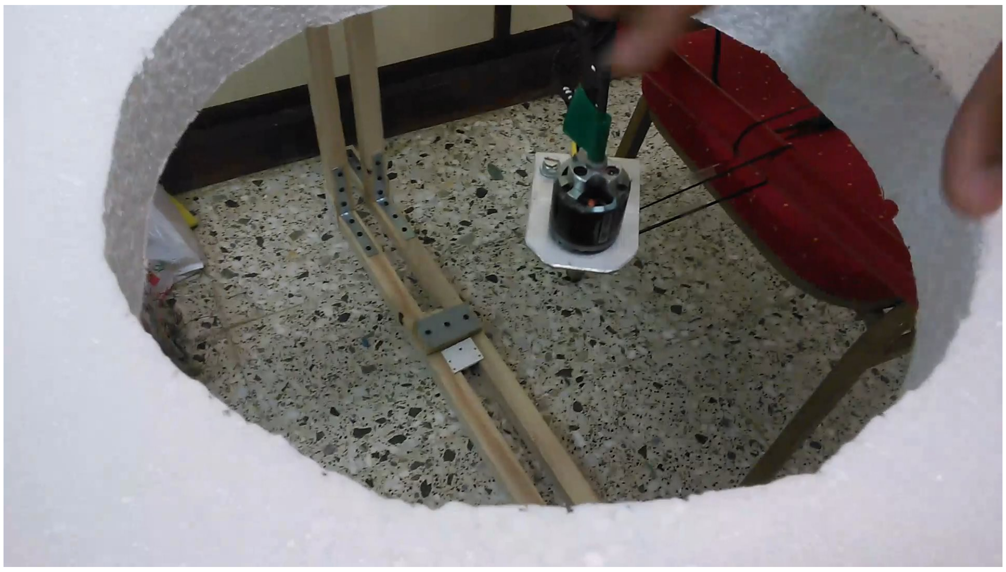


FIGURE 3.16: Air Flow Test

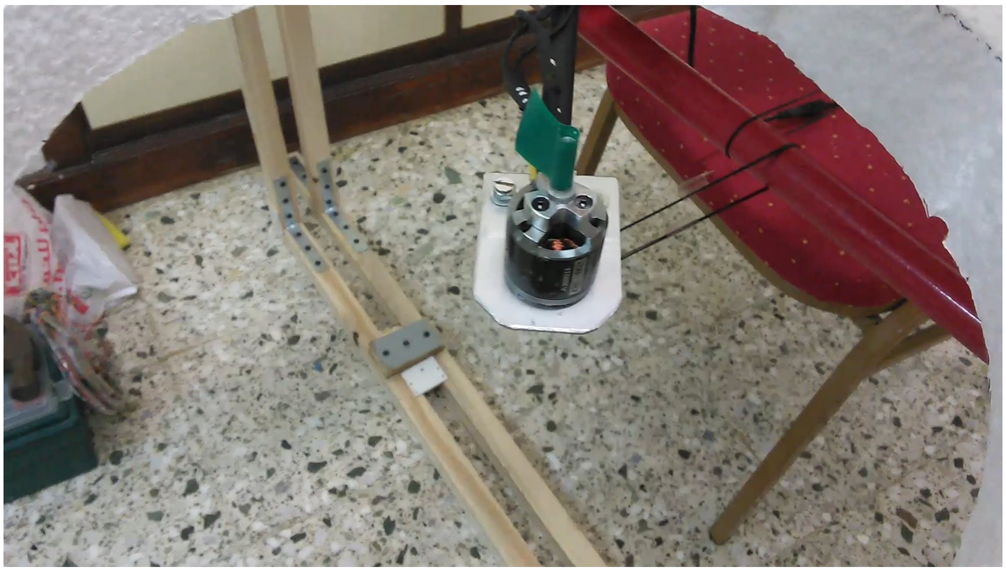


Figure 3.17: Air Flow Test

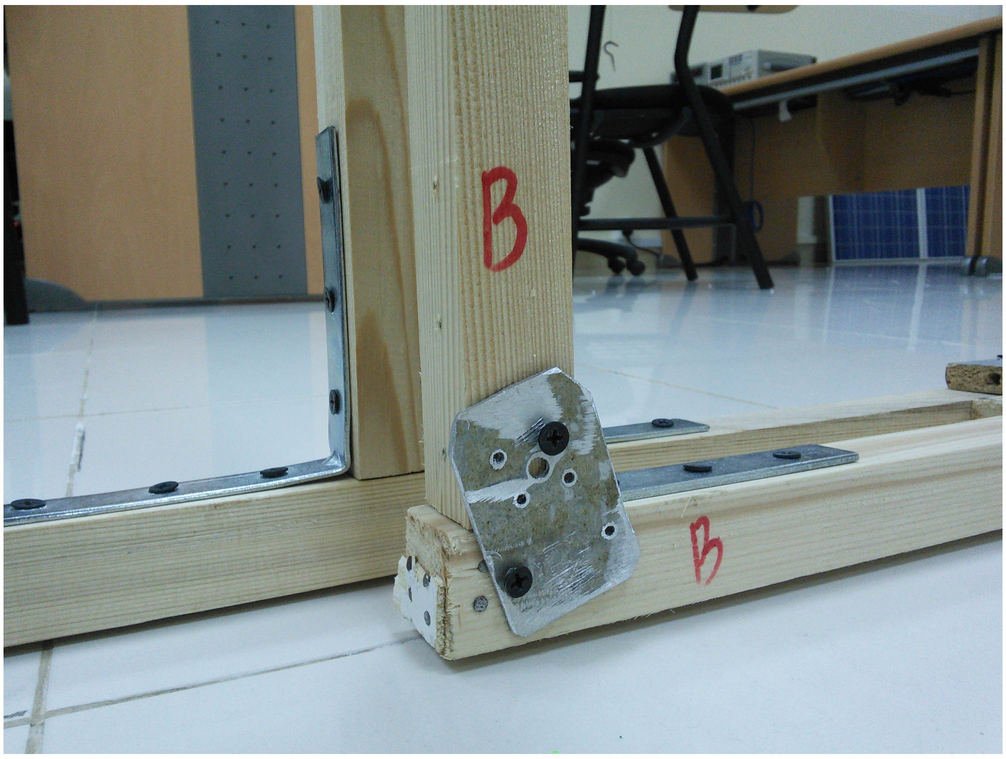


FIGURE 3.18: Frame

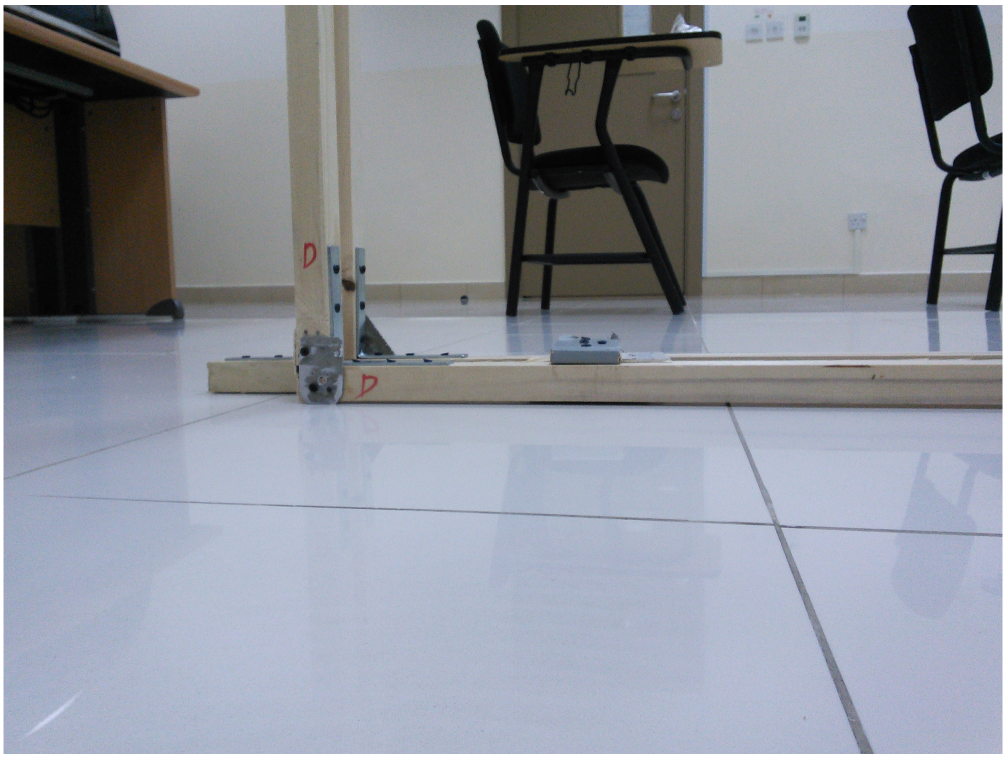


FIGURE 3.19: Frame

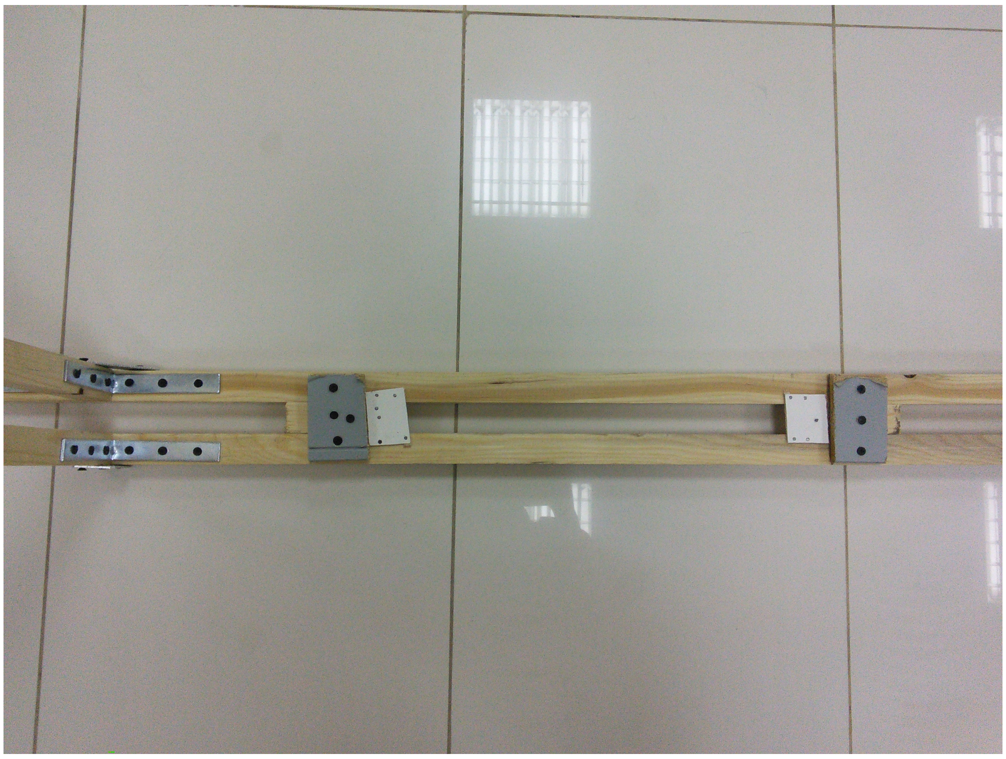


FIGURE 3.20: Frame

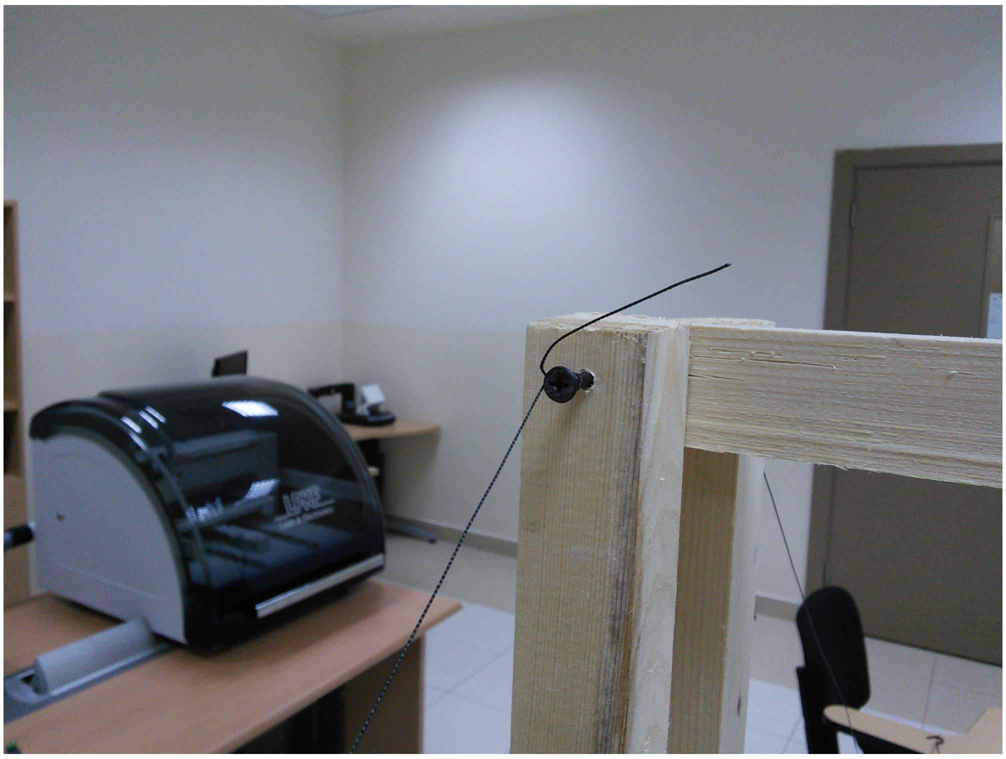


FIGURE 3.21: Frame Corner

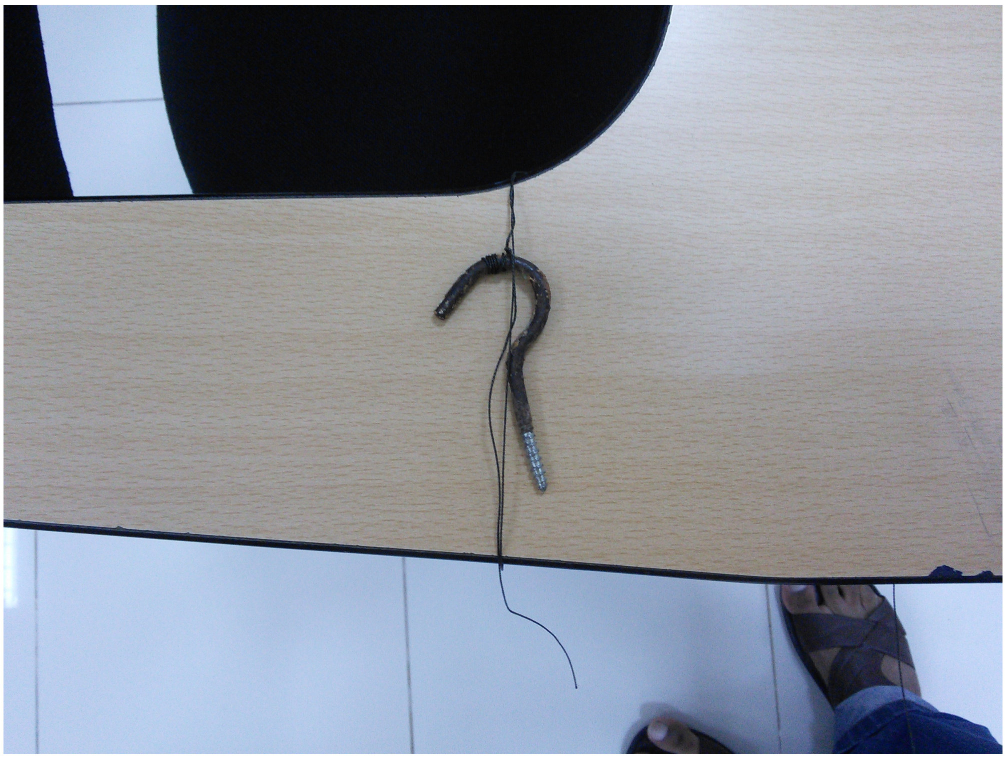


Figure 3.22: Frame Holding Screw

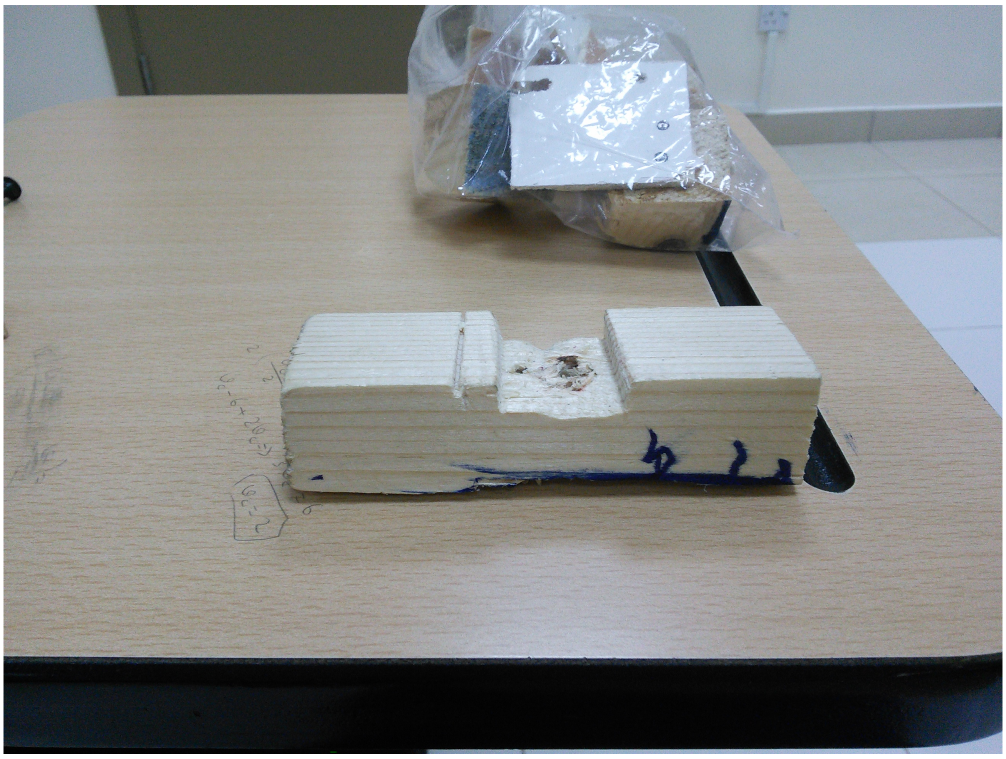


Figure 3.23: Balance Pipe Plug

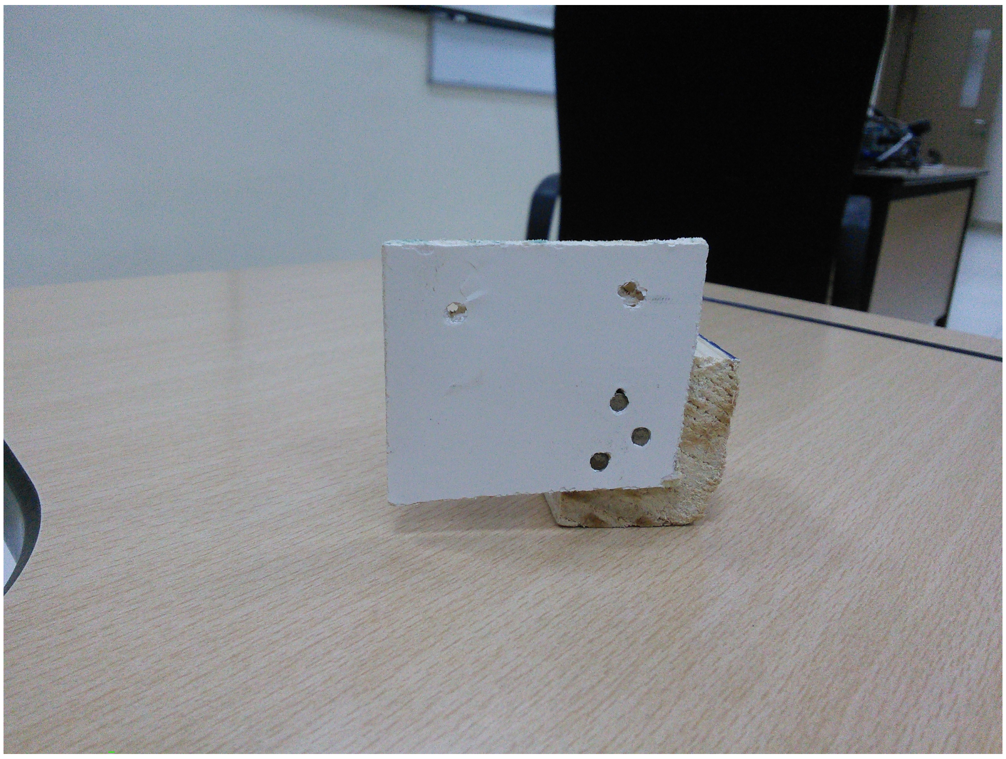


Figure 3.24: Balance Pipe Holder

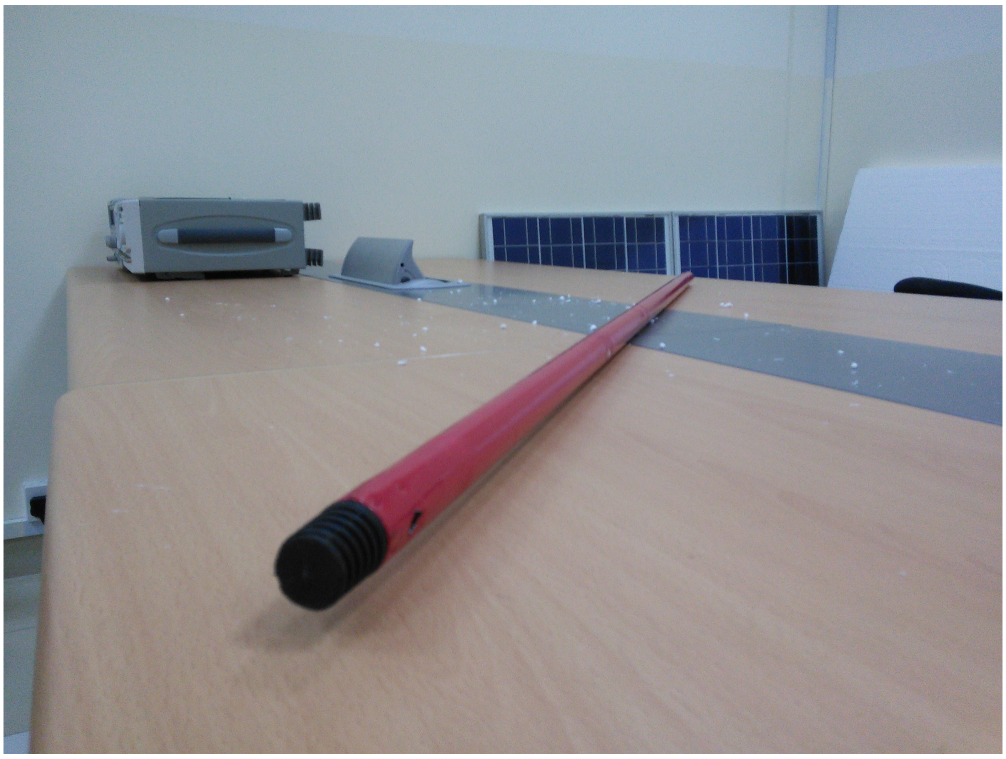


Figure 3.25: Balance Pipe

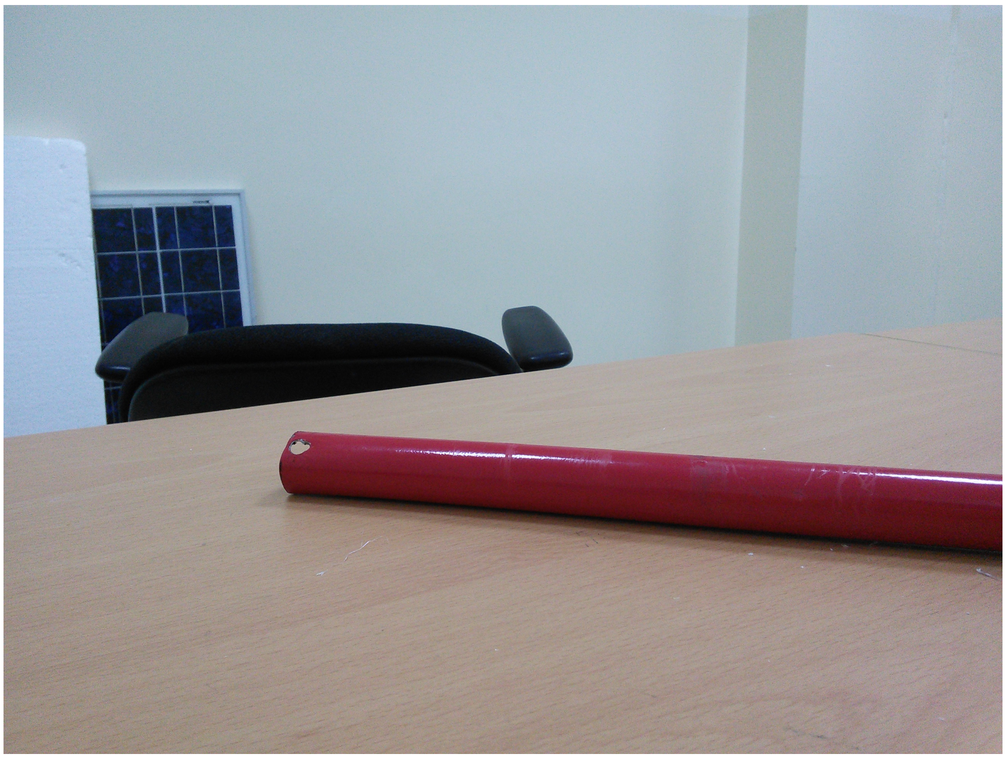


Figure 3.26: Balance Pipe

Chapter 4

Results & Discussion

4.1 Custom 3D Designed Body

Successful printing of the body was achieved but arms had to be manually drilled for the holes so that the motors fit precisely. The body was assembled using the screws with washers to prevent the plastic to breaking down. The hols for the circuit board to mount were made and the place for the mobile angle mover servo was also cut out.

In the website from Makerbot named Thingiverse (www.thingiverse.com), there are several 3D models designed for several sized quadcopters. Although these models are similar to what our final product required, these models need to be modified in several places to fit out needs and requirements.

4.1.1 Version 1 - PLA Plastic, Unchanged print

We printed the 3D quadcopter model as is from the thingiverse website without modifying any part of the quadcopter. This was done in order to make sure that the existing printer in the lab can print the model, all the parts fit together, and allow additional electronic components to be mounted onto the central body.

We found that all the parts indeed fit together but a major increase in the height of the arms from ground was needed. Also the motors we had did not fit on top of the arms perfectly. An increase in the diameter of motor mounts was also needed.

4.1.2 Version 2 - PLA Plastic, Modified print

We increased the arm motor mount diameters and increased the height of the landing pads and printed the parts using the 3D printer.

We found that all the parts indeed fit together but a tragedy occured when we left the model in the car for two days. Under the heat of the sun and no air flow, the quadcopter body melted and got twisted.

4.1.3 Version 3 - Carbon Fiber Body

We bought the carbon fiber body from Hobby King and made sure that the dimensions matched rest of the parts. The carbon fiber is composite material which is many times stronger than steel and quite light in weight.

4.2 Leap Motion Library for Gesture Control

There is a huge documentation online for programming the leap motion controller in Java. The gesture detection can be achieved using a Controller Object which can return the hand tracking data by using frame object within this class. [5] For example in the following code:

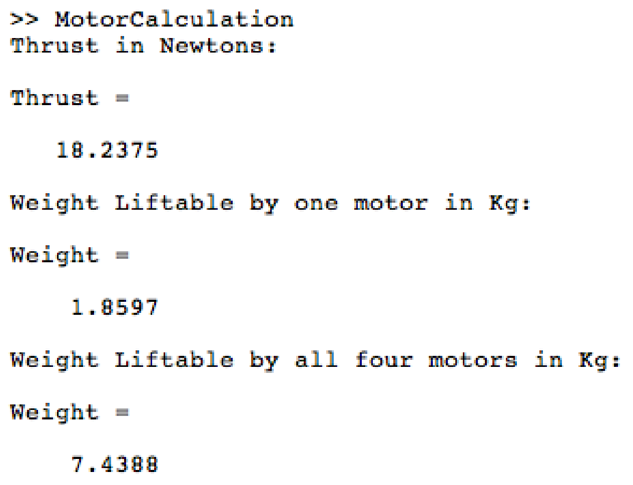
```java
SampleListener listener = new SampleListener();
Controller controller = new Controller();
// Have the sample listener receive events from the controller
controller.addListener(listener);
Frame frame = controller.frame();
System.out.println("Frame id: " + frame.id()
                 + ", timestamp: " + frame.timestamp()
                 + ", hands: " + frame.hands().count()
                 + ", fingers: " + frame.fingers().count()
                 + ", tools: " + frame.tools().count());
```

4.3 Payload Capability calculation using MATLAB

A MATLAB code was written according to the propeller efficiency, power of the motors and the propeller size to calculate the payload the quadcopter is capable of carrying.

```
% Propeller hover efficiency
eta = 0.75;
% Power of the motor Max. for our motor is 125
Power = 110;
% Propeller Radius in meters diameter = 10 inches = 0.2794
R = 0.2540;
% Usual Air Desnity kg/m^3
rho = 1.22;
Thrust = ((eta+Power)^2 * 2 * pi * R^2 * rho)^(1/3);
disp('Thrust in Newtons:');
Thrust
disp('Weight Liftable by one motor in Kg:');
Weight = Thrust/9.80665002864;
Weight
disp('Weight Liftable by all four motors in Kg:');
Weight = (Thrust/9.80665002864)*4;
Weight
```

The following was the result of the simulation:



```
>> MotorCalculation
Thrust in Newtons:

Thrust =

   18.2375

Weight Liftable by one motor in Kg:

Weight =

    1.8597

Weight Liftable by all four motors in Kg:

Weight =

    7.4388
```

FIGURE 4.1: Moving the hand away from the body can give us +ve value of movement in y-axis.

4.4 Weight Calculation of the Quadcopter.

A weight load estimation excluding the extra payload was tabulated:

Component	**Type**	**Qty.**	**Unit**	**Total**
ESC	30Amps	4	34.92g	139.7g
Motor & propellers	Super Tigre	4	59.45g	237.8g
Plastic Frame	3D Custom	1	248g	248g
Screws	Metallic	14	9.0g	125.0g
Servo	Tiny	1	11.3g	11.3g
Control Brd. & wires	Hobby King	1	21.0g	21.0g
AtMega Brd.	Custom	1	29.9g	29.9g
Android Smartphone	Huawei P6	1	120g	120g
11.V Battery	ZOP	1	369.7g	369.7g
Ultrasonic Sensor	N/A	1	10.2g	10.2g
Wiring	N/A	1	20g	20g
Total				1332.6g

The weight calculation showed that the quadcopter parts were no heavier than 1332.6 grams. in order for a quadcopter to have nice thrust, typically the thrust produced by the motors should be twice the weight. Our first test runs of the motors show that the quadcopter is easily capable of lifting these weights.

4.5 Serial Communication Protocol Development Trials

There were several protocols developed in order to communicate with the mobile.

4.5.1 Trial 1

One of the first trial was to send a string containing all the flight data and apply substring function to squeeze out the important flight variables. Some properties of the packet were as follows:

1. The length of the string should be same at all times.
2. Each variable would be mapped from whatever its original range is to be from 100 to 206 so that the string length remains the same.
3. Each variable will be separated by a '+' character.

Application was a little tricky and required many trial and error corrections after which the working code was as follows:

```
void getMotorSpeeds(){
  // Initializing the string to be null for tracking
  String dataString = "ONO+ONO+ONO+ONO"; // data string format "Motor 1 Percentage + Motor 2 Percentage + Motor 3 Percentage + Motor 4 Percentage"
  dataString = getSerialString();
  if(dataString == ""){
    // Nothing was received
    dataString = "000+000+000+000";
  }
  else{
    // Check is valid string is received from the correct '+' character placement
    if(dataString.substring(3,4) == "+" && dataString.substring(7,8) == "+" && dataString.substring(11,12) == "+"){
      // Print the 2nd raw value
      Serial.print("data 2: "+dataString.substring(4,7)+"\n");
      motorValues[2] = (((stringToNumber(dataString.substring(8,11))-100)/100.0)*76.0
      // Print the 3rd raw value
      Serial.print("data 3: "+dataString.substring(8,11)+"\n");
      motorValues[3] = (((stringToNumber(dataString.substring(12,15))-100)/100.0)*76.
      Serial.print("\n");
      // Set the PWM outputs
      motor1.write(motorValues[0]);
      motor2.write(motorValues[1]);
      motor3.write(motorValues[2]);
      motor4.write(motorValues[3]);
      // Turn OFF the yellow LED
      digitalWrite(yellowled, HIGH);
    }
    else{
      Serial.print("Don't Bullshit with me. -_-\n");
    }

  }
}
```

Disadvantages This protocol was working and for several weeks, it was used to communicating with the controller board.

1. The packet size is 15 bytes which is huge for high speed data transfer and is prone to many bit errors while data transfer.
2. Mapping the original values from 0-180 (typical servo angle signals) to values ranging from 100-206 (106 quantized levels) reduced the resolution of control variables.
3. String processing is often messy and takes a lot of time as we are calling function substring() to split the string.
4. Once there is an error in a packet, the serial buffer goes out of sync and the preceding packets will not be accepted.
5. Huge amount of serial printing back to the sender is not recommended and can cause the ATMega to reset and the smart phone to restart if the data transfer rate is high.

4.5.1.1 Packet Transfer Time

The serial transfer rate is about 1200 bps (bits per second). This was chosen after many trials after finding out that the slowest speed works best.

The number of bytes in our string are 15. Since each byte contains 8 bits, the number of bits in our string packet are:

$$bits/packet = 15 \times 8 = 120\ bits \tag{4.1}$$

Time required for transferring 112 bits

$$t_{transfer} = \frac{120}{1200} = 0.1\ seconds = 100\ milliseconds \tag{4.2}$$

After incorporating processing time, this transfer time increases and this allows next packet to come after 110 ms without aliasing or buffer fill up. The packet transfer rate then becomes maximum at 9 packets/second. This is quite low.

4.5.2 Trial 2

One of the trials included sending a string containing the data and on the other side decode it. In order to properly decode the string on the other side, the string should follow certain rules.

1. The length of the string should be same at all times.
2. The first character in the string should be 'b' to indicate the beginning of the string.

3. The last character in the string should be 'e' to indicate the ending of the string.
4. Each variable would be mapped from whatever its original range is to be from 100 to 900 so that the string length remains the same.

```
void getVariables(){
  // Let me set up a protocol for communication
  // This is a typical string sent from the mobile
  // 'b' symbolizes the begining of the string and 'e' for ending of the string
  // The motor values go from 100 to 900, so 500 is the middle value.
  // Order of Command Aileron -> Elevation -> Thrust -> Rudder
  // example string "b400400400400e"
  // Let me setup the initial Value
  String dataString = "b555666777888e";
  // data string format "Motor 1 Percentage + Motor 2 Percentage + Motor 3 Percentage + Motor 4 Percentage"

  if (dataString == ""){
    // Do Nothing
  }
  else if(dataString == "stopstopstopst"){
      // Emergency Mode, Stop Everything
      Aileron.write(0);
      Elevation.write(0);
      Thrust.write(0);
      Rudder.write(0);
      Serial.print("I Stopped \n");
  }
  else{
    // Check if it is a valid command beginning with b and ending with e (Error Checking)
    if(dataString.substring(0,1) == "b" && dataString.substring(13,14) == "e"){
      // Substring the data and indicate by Yellow LED
      digitalWrite(yellowled, LOW);
      RudderValue = (stringToNumber(dataString.substring(10,13)));
      setValues(AileronValue,ElevationValue,ThrustValue,RudderValue);
      digitalWrite(yellowled, HIGH);
    }
    else{
      // Nothing was received
      Serial.print("Something was wrong in the string received.\n");
      Serial.println(dataString);
      // Clear the Serial buffer
      Serial.flush();
    }
  }
}
```

Disadvantages Initially, the system was working flawlessly unless over time, we found out some of the defects of the protocol.

1. The packet size is 14 bytes which is huge for high speed data transfer and is prone to many bit errors while data transfer.
2. String processing is often messy and takes a lot of time as we are calling function substring() to split the string.
3. Once there is an error in a packet, the serial buffer goes out of sync and the preceding packets will not be accepted by the 'b' and 'e' criteria.
4. Serial flush function exists in Arduino serial library but does not do anything to the buffer in our experience.

4.5.2.1 Helper Function - Converting String to Number

```
int stringToNumber(String thisString) {
  int i, value, length;
  length = thisString.length();
  char blah[(length+1)];
  for(i=0; i<length; i++) {
    blah[i] = thisString.charAt(i);
  }
  blah[i]=0;
  value = atoi(blah);
  return value;
}
```

4.5.2.2 Packet Transfer Time

The serial transfer rate is about 1200 bps (bits per second). This was chosen after many trials after finding out that the slowest speed works best.

The number of bytes in our string are 14. Since each byte contains 8 bits, the number of bits in our string packet are:

$$bits/packet = 14 \times 8 = 112 \; bits \tag{4.3}$$

Time required for transferring 112 bits

$$t_{transfer} = \frac{112}{1200} = 0.0933 \; seconds \approx 93 \; milliseconds \tag{4.4}$$

After incorporating processing time, this transfer time increases and this allows next packet to come after 100 ms without aliasing or buffer fill up. The packet transfer rate then becomes maximum at 10 packets/second. This is quite low.

4.5.3 Trial 3 - Applied Method

A byte contains 8 bits and can be decoded into ASCII character or an unsigned integer. 8 bits can store $2^8 = 256$ different integers. The ESC signals range from 0-180 only. This allows a single byte to represent a speed. In other words, an ASCII character can represent any number from 0 to 180.

Instead of sending long strings from the server which contains commands and values, we can send a single byte for each of the values and a single byte to represent all of the states. The quadcopter does not need more than 256 states, infact in this project there are only 5 states. Emergency, Slow speed, Hand Guesture, Altitude Hold, and GPS Waypoint. 5 characters can be used to represent each state.

1. The length of the packet should be same at all times.
2. The first and last bytes in the packet should be $'0b10101010'$ for receiver, sender synchronization purposes.
3. The second byte indicates the state of the quadcopter.

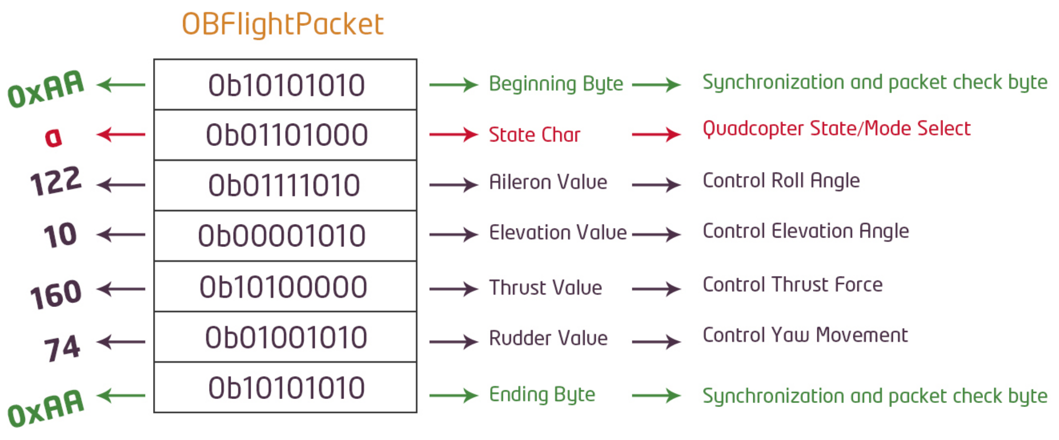


FIGURE 4.2: The figure explains the network protocol which was designed by us for sending flight data from Android USB to micro-controller.

4.5.3.1 Packet Transfer Time

The serial transfer rate is about 1200 bps (bits per second). This was chosen after many trials after finding out that the slowest speed works best.

The number of bytes in our packet are 7. Since each byte contains 8 bits, the number of bits in our string packet are:

$$bits/packet = 7 \times 8 = 56 \; bits \tag{4.5}$$

Time required for transferring 112 bits

$$t_{transfer} = \frac{56}{1200} = 0.0467 \; seconds \approx 47 \; milliseconds \tag{4.6}$$

After incorporating processing time, this transfer time increases and this allows next packet to come after 50 ms without aliasing or buffer fill up. The packet transfer rate then becomes maximum at 20 packets/second. This is acceptable.

4.6 IR Emergency Button Trials

Since there was a essential need for using an Emergency button in order to stop the quadcopter in case it is not responding. Taking the quadcopter to emergency mode should be done in a separate communication channel so that emergency command is uninterrupted or not lost in the communication link.

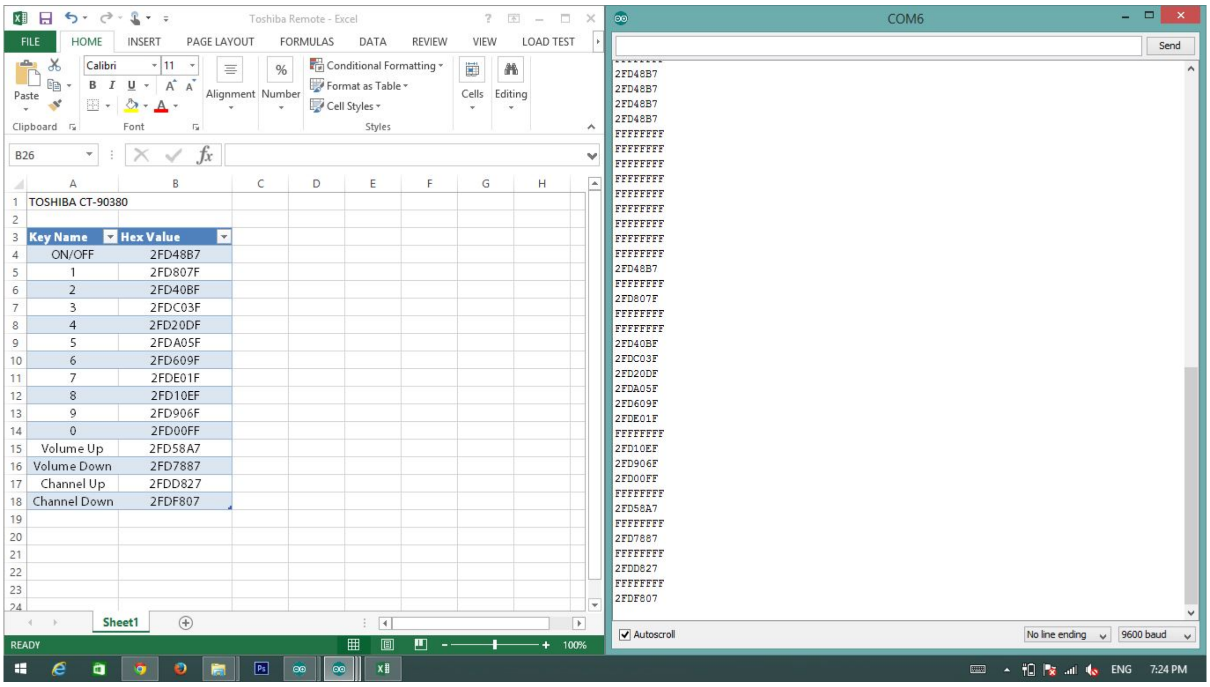
There were several options to implement the IR emergency button. After several trials, tests, and experiments we arrived at the interrupt driven emergency button.

4.6.1 Trial 1 - Code Message Driven

Arduino has a library called IRremote.h which uses the timer of the ATMega to detect IR signal pulses and returns a decoded result. The idea of using message coded IR emergency button was to put quadcopter in emergency mode once a particular message is received through IR.

```
#include <IRremote.h>

int RECV_PIN = 11;

IRrecv irrecv(RECV_PIN);

decode_results results;

void setup()
{
  Serial.begin(1200);
  irrecv.enableIRIn(); // Start the receiver
}

void loop() {
  if (irrecv.decode(&results)) {
    Serial.println(results.value, HEX);
    irrecv.resume(); // Receive the next value
  }
}
```

After writing the code, we took a standard TV remote and started pressing buttons on it to see what values are seen on the arduino terminal.



TOSHIBA CT-90380

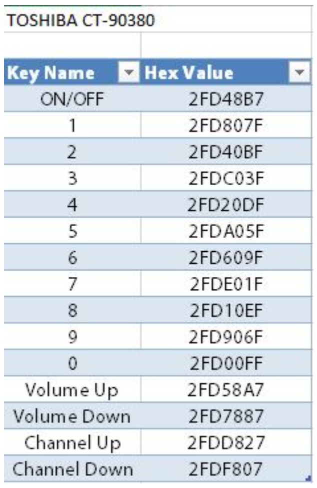
Key Name	Hex Value
ON/OFF	2FD48B7
1	2FD807F
2	2FD40BF
3	2FDC03F
4	2FD20DF
5	2FDA05F
6	2FD609F
7	2FDE01F
8	2FD10EF
9	2FD906F
0	2FD00FF
Volume Up	2FD58A7
Volume Down	2FD7887
Channel Up	2FDD827
Channel Down	2FDF807

```
2FD48B7
2FD48B7
2FD48B7
2FD48B7
FFFFFFFF
FFFFFFFF
FFFFFFFF
FFFFFFFF
FFFFFFFF
FFFFFFFF
FFFFFFFF
FFFFFFFF
FFFFFFFF
2FD48B7
FFFFFFFF
2FD807F
FFFFFFFF
FFFFFFFF
2FD40BF
2FDC03F
2FD20DF
2FDA05F
2FD609F
2FDE01F
FFFFFFFF
2FD10EF
2FD906F
2FD00FF
FFFFFFFF
2FD58A7
FFFFFFFF
2FD7887
FFFFFFFF
2FDD827
FFFFFFFF
2FDF807
```

Autoscroll: ✓ No line ending 9600 baud

FIGURE 4.3: The excel file with all the button results and the terminal screenshot.

We got the following results using a Toshiba CT-90380 IR remote:



TOSHIBA CT-90380

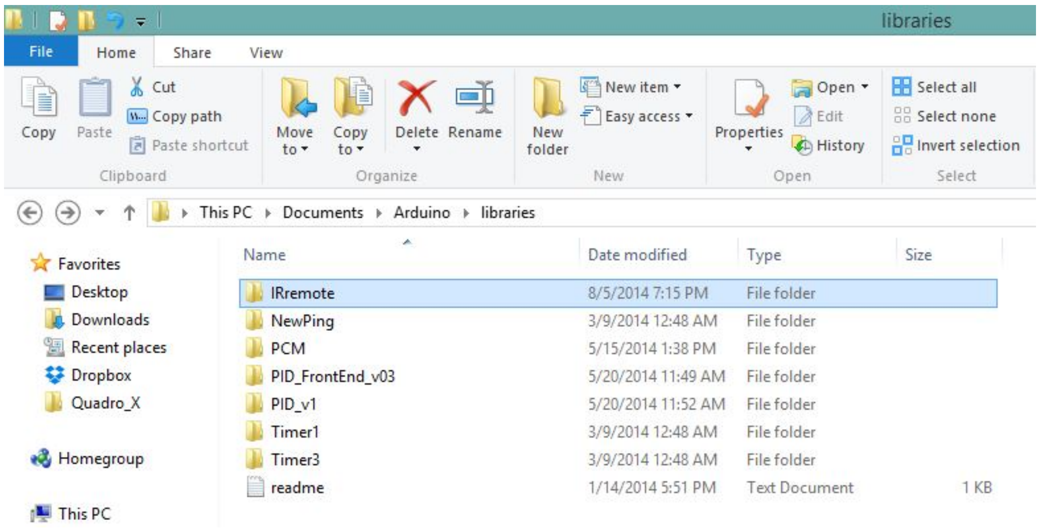
Key Name	Hex Value
ON/OFF	2FD48B7
1	2FD807F
2	2FD40BF
3	2FDC03F
4	2FD20DF
5	2FDA05F
6	2FD609F
7	2FDE01F
8	2FD10EF
9	2FD906F
0	2FD00FF
Volume Up	2FD58A7
Volume Down	2FD7887
Channel Up	2FDD827
Channel Down	2FDF807

FIGURE 4.4: The excel file with all major button IR signal results.

The problem: The IRremote library uses timer 0 by default for detecting IR pulses. Timer 0 is already occupied by our code for giving the signals to the ESCs (PWM generation).

Changing the timer from timer 0 to any other timer in 'IRremote.h' file

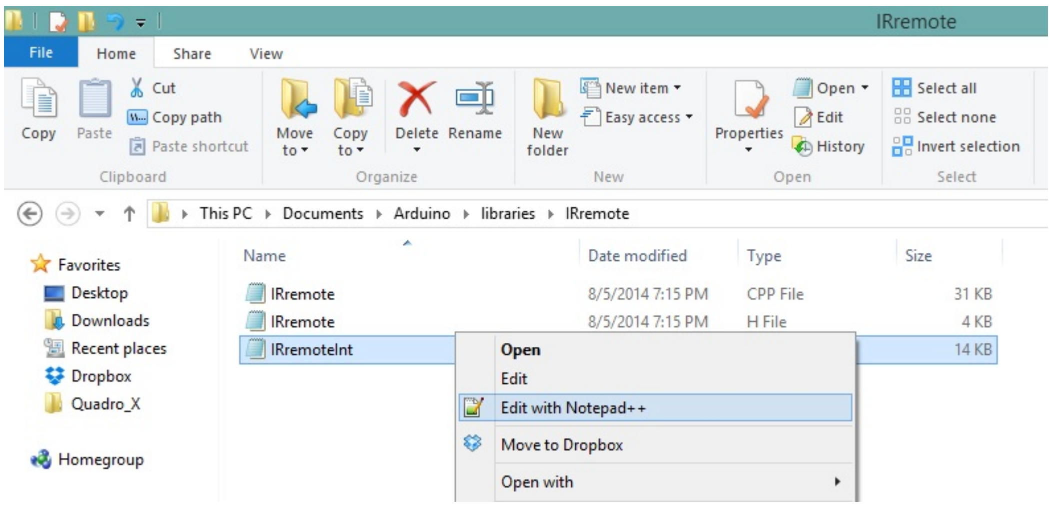
Step 1: Accessing the header file for change of definition.



Name	Date modified	Type	Size
IRremote	8/5/2014 7:15 PM	File folder	
NewPing	3/9/2014 12:48 AM	File folder	
PCM	5/15/2014 1:38 PM	File folder	
PID_FrontEnd_v03	5/20/2014 11:49 AM	File folder	
PID_v1	5/20/2014 11:52 AM	File folder	
Timer1	3/9/2014 12:48 AM	File folder	
Timer3	3/9/2014 12:48 AM	File folder	
readme	1/14/2014 5:51 PM	Text Document	1 KB

FIGURE 4.5: After importing the IRremote library using the Arduino IDE, the library is located in $C:\backslash Users\backslash Muhammad\backslash Documents\backslash Arduino\backslash libraries$.

Step 2: Opening the header file for change of definition.



Name	Date modified	Type	Size
IRremote	8/5/2014 7:15 PM	CPP File	31 KB
IRremote	8/5/2014 7:15 PM	H File	4 KB
IRremoteInt			14 KB

FIGURE 4.6: Open the 'IRremoteInt.h'.

Step 3: Changing the header file.

```
#if defined(__AVR_ATmega1280__) || defined(__AVR_ATmega2560__)
  //#define IR_USE_TIMER1   // tx = pin 11
  #define IR_USE_TIMER2     // tx = pin 9
  //#define IR_USE_TIMER3   // tx = pin 5
  //#define IR_USE_TIMER4   // tx = pin 6
  //#define IR_USE_TIMER5   // tx = pin 46

// Teensy 1.0
#elif defined(__AVR_AT90USB162__)
  #define IR_USE_TIMER1     // tx = pin 17

// Teensy 2.0
#elif defined(__AVR_ATmega32U4__)
  //#define IR_USE_TIMER1   // tx = pin 14
  //#define IR_USE_TIMER3   // tx = pin 9
  #define IR_USE_TIMER4_HS  // tx = pin 10

// Teensy++ 1.0 & 2.0
#elif defined(__AVR_AT90USB646__) || defined(__AVR_AT90USB1286__)
  //#define IR_USE_TIMER1   // tx = pin 25
  #define IR_USE_TIMER2     // tx = pin 1
  //#define IR_USE_TIMER3   // tx = pin 16

// Sanguino
#elif defined(__AVR_ATmega644P__) || defined(__AVR_ATmega644__)
  //#define IR_USE_TIMER1   // tx = pin 13
  #define IR_USE_TIMER2     // tx = pin 14

// Atmega8
#elif defined(__AVR_ATmega8P__) || defined(__AVR_ATmega8__)
  #define IR_USE_TIMER1   // tx = pin 9

// Arduino Duemilanove, Diecimila, LilyPad, Mini, Fio, etc
#else
  // Change here, uncomment the IR_USE_TIMER2
  #define IR_USE_TIMER0   // tx = pin 9
  //#define IR_USE_TIMER2     // tx = pin 3
#endif
```

4.6.2 Trial 2 - Interrupt Driven

This is kind of a hack to detect the IR signal. The IR receiver has three pins, one ground, one VCC, and one signal. The IR receiver already has an electronic circuit to amplify the IR changes sensed in the environment. If the signal wire of the IR receiver is connected to the interrupt pin of the ATMega, every time there is the change of IR signal determined in the environment, an interrupt will be raised.

```
int pin = 13;
volatile int state = LOW;

void setup()
{
  pinMode(pin, OUTPUT);
  // Attach interrupt 1 and call blink() function whenever interrupt is
     raised.
  attachInterrupt(1, blink, CHANGE);
}

void loop()
{
  digitalWrite(pin, state);
}

void blink()
{
  state = !state;
  // Delay for ensuring that the multiple interrupts are not raised
     from single press of remote button.
  delay(800);
}
```

4.6.3 KK Control Board Malfunction

The controller board was bought solely for the reason of stabilizing. The kk controller board promised of stabilizing by detecting any changes in height and orientation relative to the desired value using the gyros and compensating the change through the on board PID algorithm. Although the gyros are detecting the changes the PID value are not optimally set to compensate and from the behavior an educated guess can be made that the P value is very small and D value is very dominating. The solution is to buy a controller board whose PID value can be accessed from outside and changed to meet the requirements.

Several tests and crashes from different drivers gave the feedback of difficulty in control because of Azimuth angle of hand. The azimuth angle of hand determined the rudder of the quadcopter.

The quadcopter can be manoeuvred in all directions using only Pitch, Roll, and Thrust so the rudder value sent from the ground application while in gesture control mode is always in the middle (93). This heavily improved the hand gesture flight control.

4.7 Setting up quadcopter for flight

It is of utmost importance that the components of the quadcopter be turned ON and checked in proper order otherwise the motors can receive garbage values and respond in an unexpected way.

1. Make sure the Smart-phone is connected to the same network as ground station or in case of 4G, internet is connected.
2. Connect the Smart-phone to the FTDI Chip using the USB (male - male) connector.
3. Once the ESCs are programmed (Beeps are finished and red LED is ON in controller board), start up the ground base application.
4. Enter the IP Address and port number in the ground application after checking from the Smart-phone interface.
5. Click Connect button and check the status on Smart-phone. It should say "Connected" in Connection Status.
6. Make sure the quadcopter is in emergency mode by checking the Mode in Smart-phone.
7. Android Server Application.
8. USB mini to micro (male-male) Cable.
9. Custom built Decoding PCB.
10. HobbyKing Controller Board.
11. 5V FTDI USB-Serial converter chip.

4.8 Final Product

The final product includes the following components:-

1. Carbon Fibre Body Quadcopter.
2. Leap Motion Device.
3. Leap Motion USB Cable.
4. Orego X Desktop Application.
5. Orego X Callibration Application.
6. Android Server Application.
7. USB mini to micro (male-male) Cable.
8. Custom built Decoding PCB.
9. HobbyKing Controller Board.
10. 5V FTDI USB-Serial converter chip.

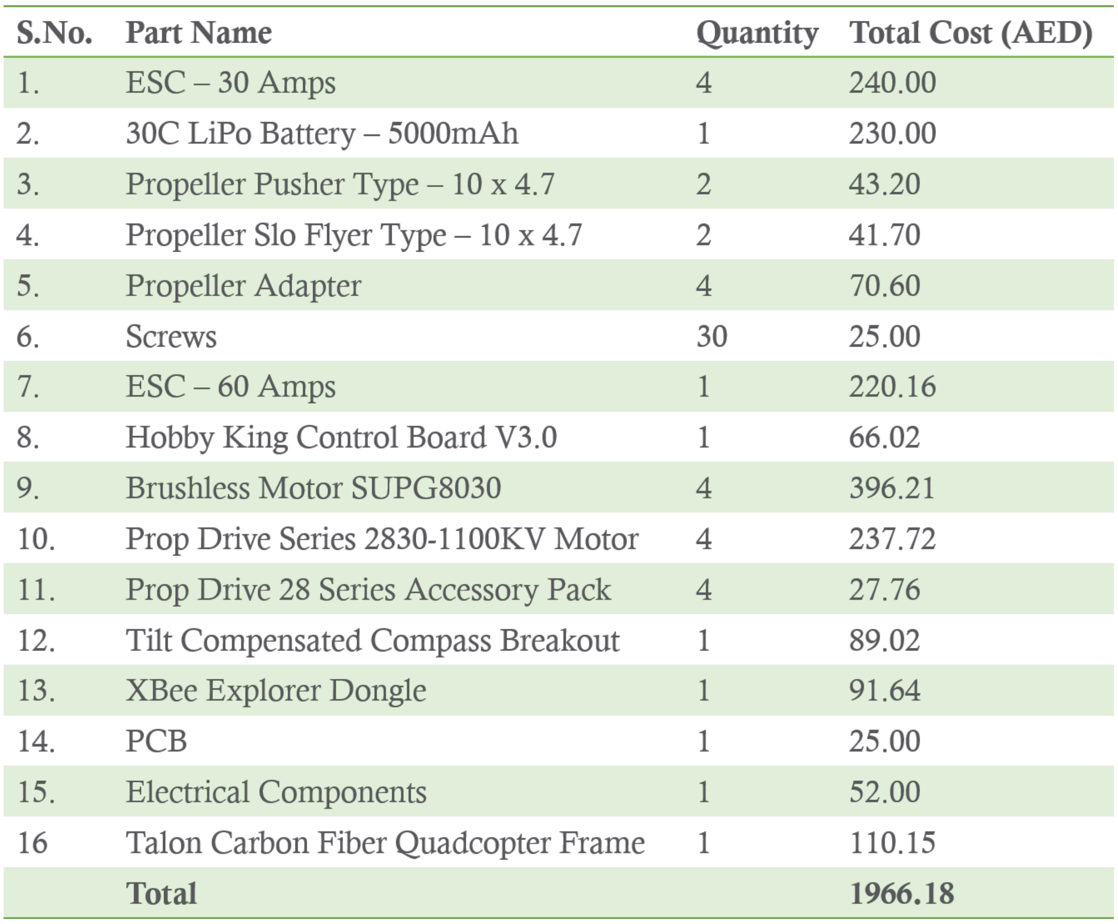
4.9 Tests Carried Out

The biggest part of the project was trial and error . This trial and error method mainly consisted of discovering the control values, gyro values, weight, battery positioning that would allow the quad rotor to lift and gain straight vertical altitude. We discovered that the quad rotor could lift itself with thrust value below 100 without the propeller guard but needs a higher value with propeller guard on.
The control board needs to be input with a minimum thrust value of 59 for the motors to start spinning. The yaw gyro pot was set to zero and the only way we configured the yaw value was sent through the user side which turned out to be 93. We also had to limit the roll and pitch value within +- 3 since they are very sensitive. The proper pitch and roll for vertical lift off is also 93.

Chapter 5

Project Management

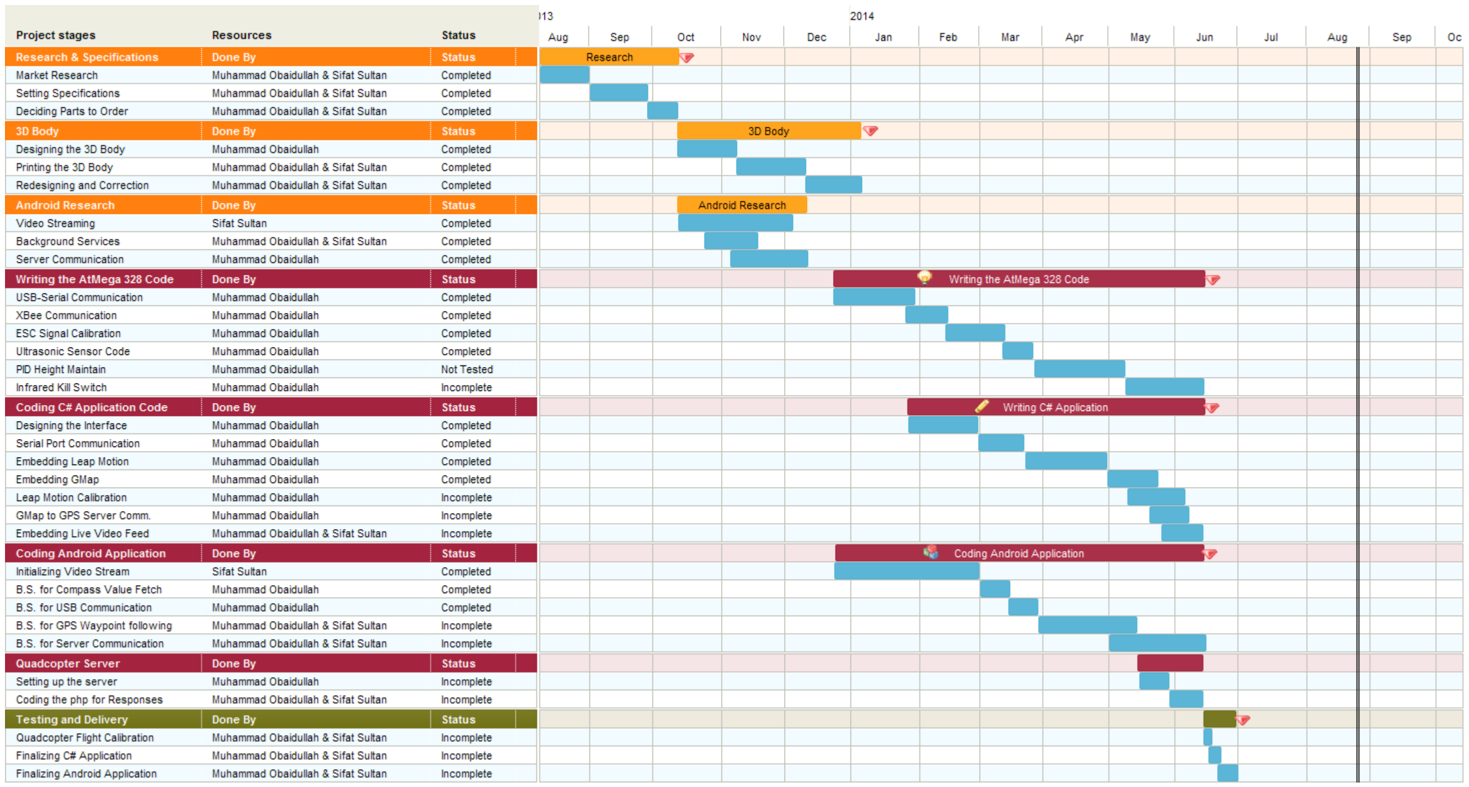
5.1 Updated Cost Table



S.No.	Part Name	Quantity	Total Cost (AED)
1.	ESC – 30 Amps	4	240.00
2.	30C LiPo Battery – 5000mAh	1	230.00
3.	Propeller Pusher Type – 10 x 4.7	2	43.20
4.	Propeller Slo Flyer Type – 10 x 4.7	2	41.70
5.	Propeller Adapter	4	70.60
6.	Screws	30	25.00
7.	ESC – 60 Amps	1	220.16
8.	Hobby King Control Board V3.0	1	66.02
9.	Brushless Motor SUPG8030	4	396.21
10.	Prop Drive Series 2830-1100KV Motor	4	237.72
11.	Prop Drive 28 Series Accessory Pack	4	27.76
12.	Tilt Compensated Compass Breakout	1	89.02
13.	XBee Explorer Dongle	1	91.64
14.	PCB	1	25.00
15.	Electrical Components	1	52.00
16	Talon Carbon Fiber Quadcopter Frame	1	110.15
	Total		**1966.18**

Figure 5.1: The table shows all the parts which were bought from various sources and countries including parts which were bought and brought personally from canada.

5.2 Gantt Chart



Project stages	Resources	Status
Research & Specifications	Done By	Status
Market Research	Muhammad Obaidullah & Sifat Sultan	Completed
Setting Specifications	Muhammad Obaidullah & Sifat Sultan	Completed
Deciding Parts to Order	Muhammad Obaidullah & Sifat Sultan	Completed
3D Body	Done By	Status
Designing the 3D Body	Muhammad Obaidullah	Completed
Printing the 3D Body	Muhammad Obaidullah & Sifat Sultan	Completed
Redesigning and Correction	Muhammad Obaidullah & Sifat Sultan	Completed
Android Research	Done By	Status
Video Streaming	Sifat Sultan	Completed
Background Services	Muhammad Obaidullah & Sifat Sultan	Completed
Server Communication	Muhammad Obaidullah	Completed
Writing the AtMega 328 Code	Done By	Status
USB-Serial Communication	Muhammad Obaidullah	Completed
XBee Communication	Muhammad Obaidullah	Completed
ESC Signal Calibration	Muhammad Obaidullah	Completed
Ultrasonic Sensor Code	Muhammad Obaidullah	Completed
PID Height Maintain	Muhammad Obaidullah	Not Tested
Infrared Kill Switch	Muhammad Obaidullah	Incomplete
Coding C# Application Code	Done By	Status
Designing the Interface	Muhammad Obaidullah	Completed
Serial Port Communication	Muhammad Obaidullah	Completed
Embedding Leap Motion	Muhammad Obaidullah	Completed
Embedding GMap	Muhammad Obaidullah	Completed
Leap Motion Calibration	Muhammad Obaidullah	Incomplete
GMap to GPS Server Comm.	Muhammad Obaidullah	Incomplete
Embedding Live Video Feed	Muhammad Obaidullah & Sifat Sultan	Incomplete
Coding Android Application	Done By	Status
Initializing Video Stream	Sifat Sultan	Completed
B.S. for Compass Value Fetch	Muhammad Obaidullah	Completed
B.S. for USB Communication	Muhammad Obaidullah	Completed
B.S. for GPS Waypoint following	Muhammad Obaidullah & Sifat Sultan	Incomplete
B.S. for Server Communication	Muhammad Obaidullah & Sifat Sultan	Incomplete
Quadcopter Server	Done By	Status
Setting up the server	Muhammad Obaidullah	Incomplete
Coding the php for Responses	Muhammad Obaidullah & Sifat Sultan	Incomplete
Testing and Delivery	Done By	Status
Quadcopter Flight Calibration	Muhammad Obaidullah & Sifat Sultan	Incomplete
Finalizing C# Application	Muhammad Obaidullah & Sifat Sultan	Incomplete
Finalizing Android Application	Muhammad Obaidullah & Sifat Sultan	Incomplete

5.3 Tasks

1

Task Name: Modifying 3D body design
Accomplished By: Muhammad Obaidullah
Starting Date: 4 December 2013
Completion Date: 26 December 2013
Obstacles Faced:

- The Sketch Up did not allow modification of the motor mount unless the landing pad was removed. Used an older version of the Sketch Up file provided online and printed the leg separately.

Percentage Completion: 100%
Output: The quadcopter arms and body is now completely designed in Google Sketch Up and can be exported as .stl format for printing using the Makerbot.

2

Task Name: Printing the 3D model and painting it
Accomplished By: Muhammad Obaidullah
Starting Date: 30 December 2013
Completion Date: 26 March 2014
Obstacles Faced:

- Several times, the electricity of the lab went and the model had to be printed again.
- After repeated printing and experimenting, the size of the holes and arms was modified for perfect fitting.
- Several test flights resulted in broken parts of the quadcopter, these parts were printed again.

Percentage Completion: 100%
Output: PLA plastic body is completely printed using the lab's 3D printer and painted using spray paint.

3

Task Name: Reverse engineering the HobbyKing flight control board
Accomplished By: Muhammad Obaidullah
Starting Date: 28 January 2014
Completion Date: 12 March 2014
Obstacles Faced:

- Surface mount resistor values were very hard to find out because of small size.
- The signals required by the flight control board were found and confirmed using the oscilloscope.

Percentage Completion: 100%
Output: We know exactly what signals are coming out from the HobbyKing flight control board and what input signals (Elevation, Aileron, Thrust, Rudder) signals are required as inputs. It was also of utmost importance to find out the V_{p-p} and duty cycle of the output PWM which is fed into the ESCs (Electronic Speed Controllers).

4

Task Name: AtMega 328 to Android Protocol Development
Accomplished By: Muhammad Obaidullah
Starting Date: 12 March 2014
Completion Date: 26 March 2014
Obstacles Faced:

- Serial communication gives low error rate under slow bit rate and XBee performs well close to 1200 bps serial data rate. Therefore the baud rate was reduced to 1200 bps. And appropriate delay was added in the Serial data read loop to wait for the next byte to arrive.

Percentage Completion: 100%
Output: Using this protocol, the quadcopter can be controlled completely. The android smart-phone can send flight data and control the global positioning of the quadcopter.

5

Task Name: Configuring XBee for serial communication
Accomplished By: Muhammad Obaidullah
Starting Date: 26 March 2014
Completion Date: 27 March 2014
Obstacles Faced:

- New version of XCTU does not recognize the XBees bought. Downloaded older version of XCTU and configured.
- The baud rate was reduced to 1200 bps to ensure low bit error rate. This rate was chosen after many tests using the oscilloscope channel 1 connected to the RX pin of XBee to detect whether it received every byte sent at different data speeds.

Percentage Completion: 100%
Output: The quadcopter can now be controlled by using either XCTU, C# windows application, or Android application. The following link was used: http://www.science.smith.edu/~jcardell/Courses/EGR328/Readings/XbeeGettingStarted.pdf

6

Task Name: Capture video using Camera object
Accomplished By: : Sifat Sultan
Starting Date: 1 January 2014
Completion Date: 5 June 2014
Obstacles Faced:

- Our quad copter is required to take a video stream of the surrounding and using any USB camera that are found in the market to carry out this task is bad choice as with every addition of weight the battery life of the quad copter is bound to reduce.

Percentage Completion: 100%
Output: Since the quad copter comes with smartphone which is intended to do the task of providing with 4g communication service, it only makes sense to use the other features that comes along with a standard android smartphone. It comes with a powerful built in camera that can capture HD video. Therefore we programmed the app to open the camera and capture the video through the following code.

7

Task Name: Play Live Stream using HttpURLConnection
Accomplished By: : Sifat Sultan
Starting Date: 15 January 2014
Completion Date: 5 June 2014
Obstacles Faced:

- The smartphone will stream the video to a certain IP and Port address. This stream is very efficiently encoded such that it needs powerful library to be decoded.
- I planned to use a WebView or a Media Player to play the video streaming. However; neither WebView nor Media Player in android is not powerful enough to decode the stream and play video.

Percentage Completion: 100%
Output: I will use HttpURLConnection object to establish socket connection between the user smartphone and the server smartphone that is mounted on the quad copter. Once the connection is established the input stream of the socket is opened. Then a Bitmap object will be used to decode the jpeg image that is stored in the input stream using the function BitampFactory.decodeStream(InputStream inputStream).

8

Task Name: Android background service for live video streaming
Accomplished By: Sifat Sultan
Starting Date: 30 March 2014
Completion Date: 15 April 2014
Obstacles Faced:

- The activity in the foreground should be able to bind to the background service and access all data.
- Almost no documentation existed for streaming video from background.

Percentage Completion: 100%
Output: The andorid device which is mounted on top of the quadcopter can stream live video using the 4G network, 3G network or WiFI internet.

9

Task Name: Android background service for getting direction from compass
Accomplished By: Muhammad Obaidullah
Starting Date: 5 April 2014
Completion Date: 15 April 2014
Obstacles Faced:

- Tried to implement the tilt compensated compass using the libraries provided by researchers in the field but found out that the tilt compensation reduces the accuracy of the smart-phone compass.
- A smartphone compass does not provide the tilt compensated direction. It is the raw compass direction where the value can be extremely wrong if the orientation of the android phone itself is not correct.

Percentage Completion: 100%
Output: The andorid device which is mounted on top of the quadcopter can know the direction of the quadcopter and is ready for performing flight calculations.

10

Task Name: Ground Station Windows *C#* application interface design
Accomplished By: Muhammad Obaidullah
Starting Date: 2 February 2014
Completion Date: 20 February 2014
Obstacles Faced:

- Completely new to *C#* programming and **no part of the *C#* application is copied from internet. All lines of code written by Obaidullah.**

Percentage Completion: 100%
Output: Base station was designed to incorporate XBee communication control.

11

Task Name: Embedding GPS functionality in the $C\#$ application.
Accomplished By: Muhammad Obaidullah
Starting Date: 28 April 2014
Completion Date: 23 May 2014
Obstacles Faced:

- GMap library was used to embed the map into the $C\#$ application.
- GMap provides many pre-built functions like drawing a polygon and defining GPS point using Latitude and longitude.

Percentage Completion: 100%
Output: The ground station can be used to set way-points for the quadcopter to move and also get the updated longitude and latitude position of the quadcopter back from the quadcopter.

12

Task Name: Embedding the XBee serial port in the $C\#$ application.
Accomplished By: Muhammad Obaidullah
Starting Date: 19 February 2014
Completion Date: 20 February 2014
Obstacles Faced:

- Programming the Serial port and changing the settings for the port by using the combo box.

Percentage Completion: 100%
Output: The user can select the port to start communicating with the XBee explorer and select the baud rate by which the user can communicate. The data can be sent automatically using the hand gestures or it can be manually entered in the text-boxes and sent.

13

Task Name: GPS follower background service for way-point determined flight control.
Accomplished By: Muhammad Obaidullah & Sifat Sultan
Starting Date: 25 May 2014
Completion Date: 7 June 2014
Obstacles Faced:

- No previous knowledge of php and databases.
- Figuring out the perfect closed-loop control for reaching the way-point.

Percentage Completion: 45%
Output: The user can select the waypoints on the *C#* application and those waypoints will be uploaded to the database. From the database, the waypoints are grabbed by this background service and accordingly the OBFlightPackets are generated to make the quadcopter go to the desired way-point.

14

Task Name: Embedding live update of the altitude in the *C#* application.
Accomplished By: Muhammad Obaidullah
Starting Date: 24 February 2014
Completion Date: 25 February 2014
Obstacles Faced:

- Getting the serial string from the XBee and performing string operations to extract the height data.

Percentage Completion: 100%
Output: The sonar sensor which is mounted on the quadcopter can give live height feedback to the micro-controller and XBee. The *C#* application shows the current height in the nice progress bar.

15

Task Name: Writing the Arduino C code for the AtMega328.

Accomplished By: Muhammad Obaidullah

Starting Date: 11 February 2014

Completion Date: 27 May 2014

Obstacles Faced:

- Sonar sensor requires very slow triggers of about 20 Hertz. The microprocessor cannot wait for the sonar sensor to send the sound signal, receive the signal back and calculate the distance.
- Drawing a clear line between the inputs and the outputs of the quadcopter system and figuring out the finite state machine diagram.
- Implementing the OBFLightPacket protocol in the C language by casting the incoming data into byte array and send to the ESCs.

Percentage Completion: 100%

Output: The AtMega 328 chip can be mounted on the controller board and the quadcopter is fully functional and ready with features such as AtMega 328 to android communication, height stabilization, XBee control, and mode select.

Task Name: Designing & updating the circuit diagram for the controller board.
Accomplished By: Muhammad Obaidullah
Starting Date: 4 February 2014
Completion Date: 26 March 2014
Obstacles Faced:

- The tilt compensated compass is connected via I^2C Master-slave configuration and the JTAG SPI programmer header is also connected to AtMega328.
- Providing the 5V logic power to the circuit board by using the ESC 3 pin headers.
- 16 Kill switch is an essential and necessary requirement especially when in initial testing stages. Implemented the kill switch using the infrared receiver.

Percentage Completion: 100%
Output: The quadcopter can communicate with the tilt compensated compass and be programmed using the JTAG SPI header directly using the *.hex* format binary file. Additionally, the power is given to the circuit board using the ESC's logic output. This improves the efficiency of the system because the ESCs have a buck converter inside to step the voltage down from 11.1V to 5V using the PWM switching regulation technique. Any TV remote with infrared LEDs can be used to shut the quadcopter OFF completely as any change in the infrared signal causes and interrupt to be raised whereby the quadcopter switches immediately to the emergency mode/state.

17

Task Name: Soldering & mounting the equipment on the PCB

Accomplished By: Muhammad Obaidullah

Starting Date: 24 December 2013

Completion Date: 28 May 2014

Obstacles Faced:

- There was a vias under the AtMega so it had to be soldered before all bracket was to be soldered.
- The PCB machine failed several times before a good PCB was printed. A lot of time and effort was wasted in this. This also caused the project to be delayed.
- The first versions had some faults in them so had to be printed again and again.

Percentage Completion: 100%

Output: The quadcopter PCB is completely soldered with all the features and ready to be mounted on top of the quadcopter for autonomous flight.

Chapter 6

Conclusion

A quad-copter might just be one of the more complicated project a beginner in this field of RC drone can take. By ‘complicated’ we mean to say that it is not only unpredictable but also ‘life-threatening’ at times. During the course of the project there were incidents when the propeller caused sever injuries which forced us to dedicate a considerable amount of our time in developing the safety features of it; propeller guard, wooden frame, the emergency button and the socket timeout feature.The sheer sound of the ‘Turningy’ brushless motor can make the heart beat go fast. To imagine that all of our complications are related to this motor it was very difficult to debug it or diagnose with such fear from the motors. This of course have taught us some handy tricks like; always take off the propellers when testing.

A visit to the RC drone forums gives a glimpse of how unpredictable this device can behave as there are threads after threads in those forums written on the same issue but had different solutions. This could be credited to the fact that there are various modules in play; the receiver, controller board, gyro tuning, the esc and finally the brushless motors, not to mention the quad copter body which underwent two changes and other parts that had to be custom remade in the mechanical lab.

The significance of data types was one of those topics that we have come to appreciate during this project. Data type should be chosen keeping in mind all the different languages used in the modules and whether or not they will be able to interpret it correctly.

This project was a tremendous learning experience and the sight of the quad copter flying was nothing less then sheer delight.

Appendix A

C

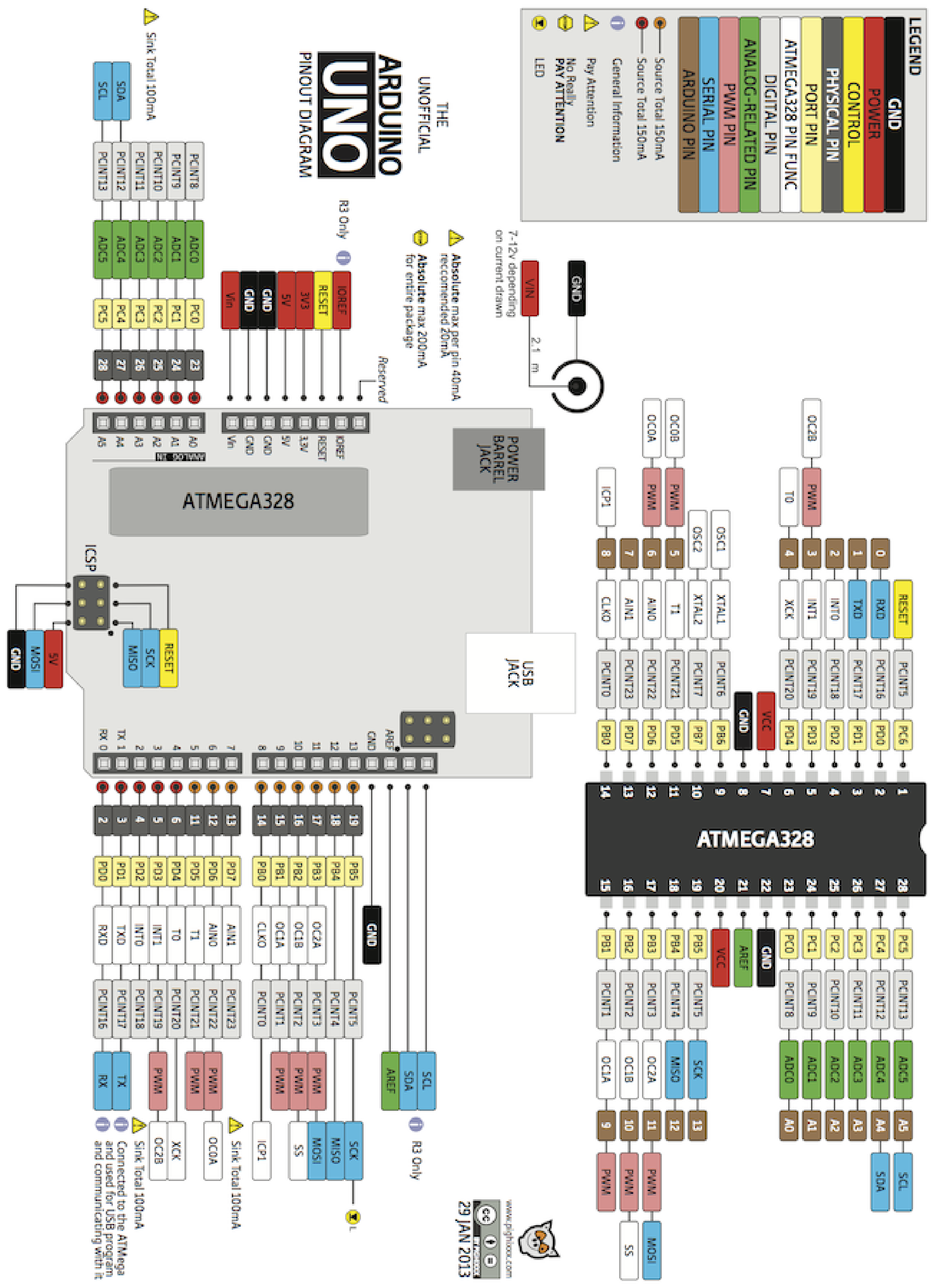
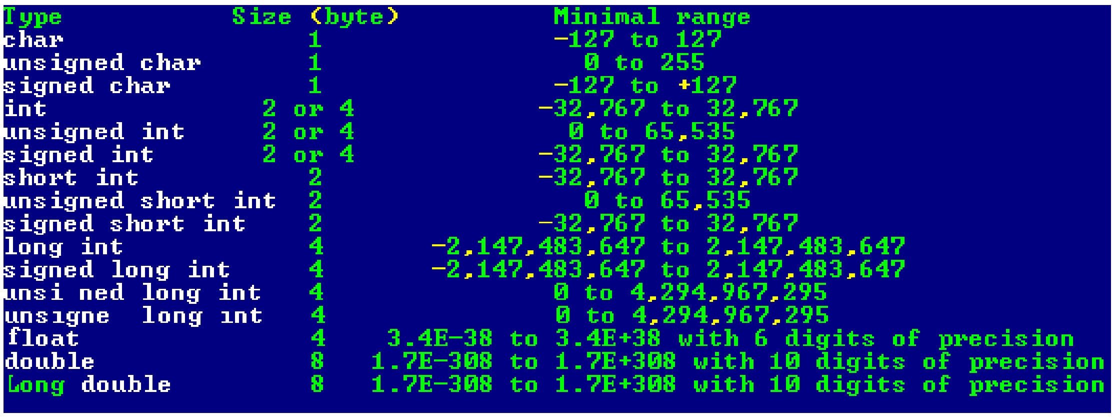


FIGURE A.1: Arduino Uno Pinout



Type	Size (byte)	Minimal range
char	1	-127 to 127
unsigned char	1	0 to 255
signed char	1	-127 to +127
int	2 or 4	-32,767 to 32,767
unsigned int	2 or 4	0 to 65,535
signed int	2 or 4	-32,767 to 32,767
short int	2	-32,767 to 32,767
unsigned short int	2	0 to 65,535
signed short int	2	-32,767 to 32,767
long int	4	-2,147,483,647 to 2,147,483,647
signed long int	4	-2,147,483,647 to 2,147,483,647
unsi ned long int	4	0 to 4,294,967,295
unsigne long int	4	0 to 4,294,967,295
float	4	3.4E-38 to 3.4E+38 with 6 digits of precision
double	8	1.7E-308 to 1.7E+308 with 10 digits of precision
Long double	8	1.7E-308 to 1.7E+308 with 10 digits of precision

Figure A.2: C Data Type

Appendix B

Hardware used Manuals

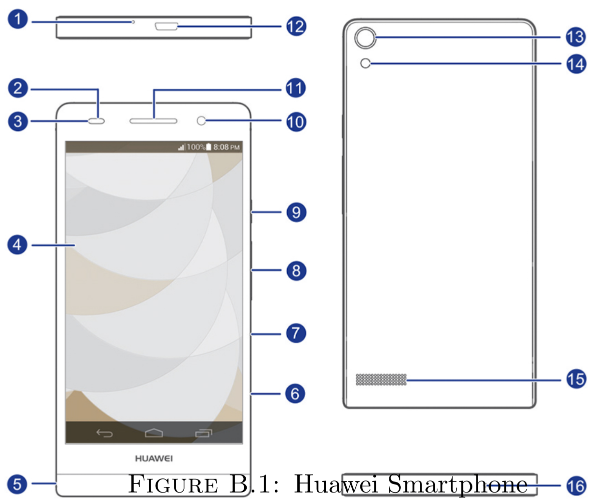


Figure B.1: Huawei Smartphone

1	Noise reduction microphone	2	Proximity sensor
3	Status indicator	4	Screen
5	Headset jack	6	Micro-SIM card slot
7	microSD card slot	8	Volume button
9	Power button ⏻	10	Front camera
11	Earpiece	12	Micro USB port
13	Rear camera	14	Flashlight

ATmega48A/PA/88A/PA/168A/PA/328/P

ATMEL 8-BIT MICROCONTROLLER WITH 4/8/16/32KB IN-SYSTEM PROGRAMMABLE FLASH

SUMMARY DATASHEET

Features

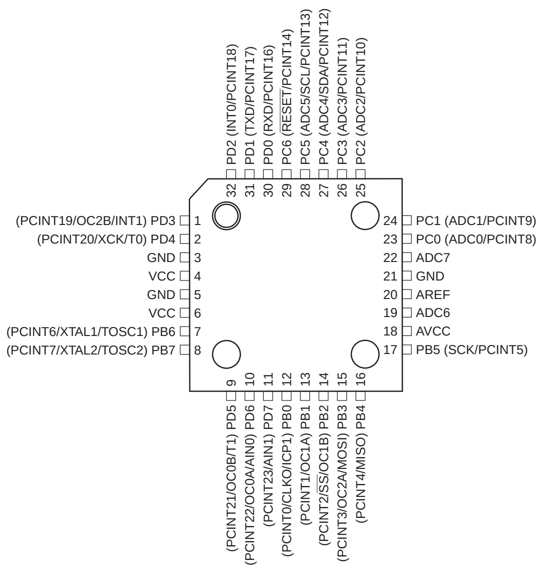
- High Performance, Low Power Atmel®AVR® 8-Bit Microcontroller Family
- Advanced RISC Architecture
  - 131 Powerful Instructions – Most Single Clock Cycle Execution
  - 32 x 8 General Purpose Working Registers
  - Fully Static Operation
  - Up to 20 MIPS Throughput at 20MHz
  - On-chip 2-cycle Multiplier
- High Endurance Non-volatile Memory Segments
  - 4/8/16/32KBytes of In-System Self-Programmable Flash program memory
  - 256/512/512/1KBytes EEPROM
  - 512/1K/1K/2KBytes Internal SRAM
  - Write/Erase Cycles: 10,000 Flash/100,000 EEPROM
  - Data retention: 20 years at 85°C/100 years at 25°C[1]
  - Optional Boot Code Section with Independent Lock Bits
    - In-System Programming by On-chip Boot Program
    - True Read-While-Write Operation
  - Programming Lock for Software Security
- Atmel® QTouch® library support
  - Capacitive touch buttons, sliders and wheels
  - QTouch and QMatrix® acquisition
  - Up to 64 sense channels
- Peripheral Features
  - Two 8-bit Timer/Counters with Separate Prescaler and Compare Mode
  - One 16-bit Timer/Counter with Separate Prescaler, Compare Mode, and Capture Mode
  - Real Time Counter with Separate Oscillator
  - Six PWM Channels
  - 8-channel 10-bit ADC in TQFP and QFN/MLF package
    - Temperature Measurement
  - 6-channel 10-bit ADC in PDIP Package
    - Temperature Measurement
  - Programmable Serial USART
  - Master/Slave SPI Serial Interface
  - Byte-oriented 2-wire Serial Interface (Philips I^{2}C compatible)
  - Programmable Watchdog Timer with Separate On-chip Oscillator
  - On-chip Analog Comparator
  - Interrupt and Wake-up on Pin Change

- Special Microcontroller Features
  - Power-on Reset and Programmable Brown-out Detection
  - Internal Calibrated Oscillator
  - External and Internal Interrupt Sources
  - Six Sleep Modes: Idle, ADC Noise Reduction, Power-save, Power-down, Standby, and Extended Standby
- I/O and Packages
  - 23 Programmable I/O Lines
  - 28-pin PDIP, 32-lead TQFP, 28-pad QFN/MLF and 32-pad QFN/MLF
- Operating Voltage:
  - 1.8 - 5.5V
- Temperature Range:
  - -40°C to 85°C
- Speed Grade:
  - 0 - 4MHz@1.8 - 5.5V, 0 - 10MHz@2.7 - 5.5.V, 0 - 20MHz @ 4.5 - 5.5V
- Power Consumption at 1MHz, 1.8V, 25°C
  - Active Mode: 0.2mA
  - Power-down Mode: 0.1μA
  - Power-save Mode: 0.75μA (Including 32kHz RTC)

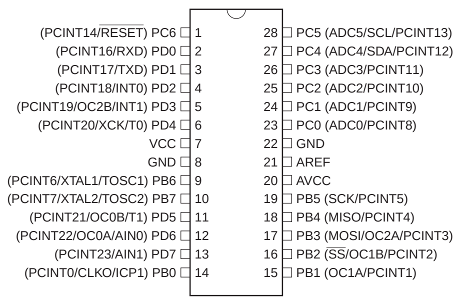
1. Pin Configurations

Figure 1-1. Pinout ATmega48A/PA/88A/PA/168A/PA/328/P

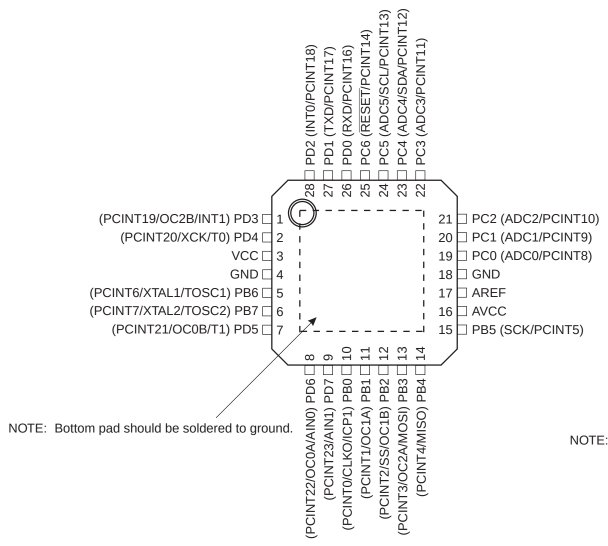
32 TQFP Top View



28 PDIP



28 MLF Top View



32 MLF Top View

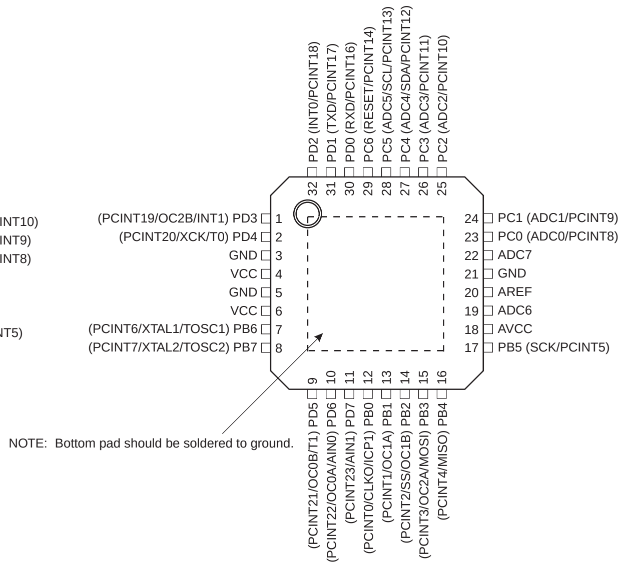


Table 1-1. 32UFBGA - Pinout ATmega48A/48PA/88A/88PA/168A/168PA

	1	2	3	4	5	6
A	PD2	PD1	PC6	PC4	PC2	PC1
B	PD3	PD4	PD0	PC5	PC3	PC0
C	GND	GND			ADC7	GND
D	VDD	VDD			AREF	ADC6
E	PB6	PD6	PB0	PB2	AVDD	PB5
F	PB7	PD5	PD7	PB1	PB3	PB4

1.1 Pin Descriptions

1.1.1 VCC

Digital supply voltage.

1.1.2 GND

Ground.

1.1.3 Port B (PB7:0) XTAL1/XTAL2/TOSC1/TOSC2

Port B is an 8-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The Port B output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port B pins that are externally pulled low will source current if the pull-up resistors are activated. The Port B pins are tri-stated when a reset condition becomes active, even if the clock is not running.

Depending on the clock selection fuse settings, PB6 can be used as input to the inverting Oscillator amplifier and input to the internal clock operating circuit.

Depending on the clock selection fuse settings, PB7 can be used as output from the inverting Oscillator amplifier.

If the Internal Calibrated RC Oscillator is used as chip clock source, PB7...6 is used as TOSC2...1 input for the Asynchronous Timer/Counter2 if the AS2 bit in ASSR is set.

The various special features of Port B are elaborated in ”Alternate Functions of Port B” on page 83 and ”System Clock and Clock Options” on page 27.

1.1.4 Port C (PC5:0)

Port C is a 7-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The PC5...0 output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port C pins that are externally pulled low will source current if the pull-up resistors are activated. The Port C pins are tri-stated when a reset condition becomes active, even if the clock is not running.

1.1.5 PC6/$\overline{\text{RESET}}$

If the RSTDISBL Fuse is programmed, PC6 is used as an I/O pin. Note that the electrical characteristics of PC6 differ from those of the other pins of Port C.

If the RSTDISBL Fuse is unprogrammed, PC6 is used as a Reset input. A low level on this pin for longer than the minimum pulse length will generate a Reset, even if the clock is not running. The minimum pulse length is given in Table 29-16 on page 311. Shorter pulses are not guaranteed to generate a Reset.

The various special features of Port C are elaborated in ”Alternate Functions of Port C” on page 86.|

1.1.6 Port D (PD7:0)

Port D is an 8-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The Port D output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port D pins that are externally pulled low will source current if the pull-up resistors are activated. The Port D pins are tri-stated when a reset condition becomes active, even if the clock is not running.

The various special features of Port D are elaborated in ”Alternate Functions of Port D” on page 89.

1.1.7 AV_{CC}

AV_{CC} is the supply voltage pin for the A/D Converter, PC3:0, and ADC7:6. It should be externally connected to V_{CC}, even if the ADC is not used. If the ADC is used, it should be connected to V_{CC} through a low-pass filter. Note that PC6...4 use digital supply voltage, V_{CC}.

1.1.8 AREF

AREF is the analog reference pin for the A/D Converter.

1.1.9 ADC7:6 (TQFP and QFN/MLF Package Only)

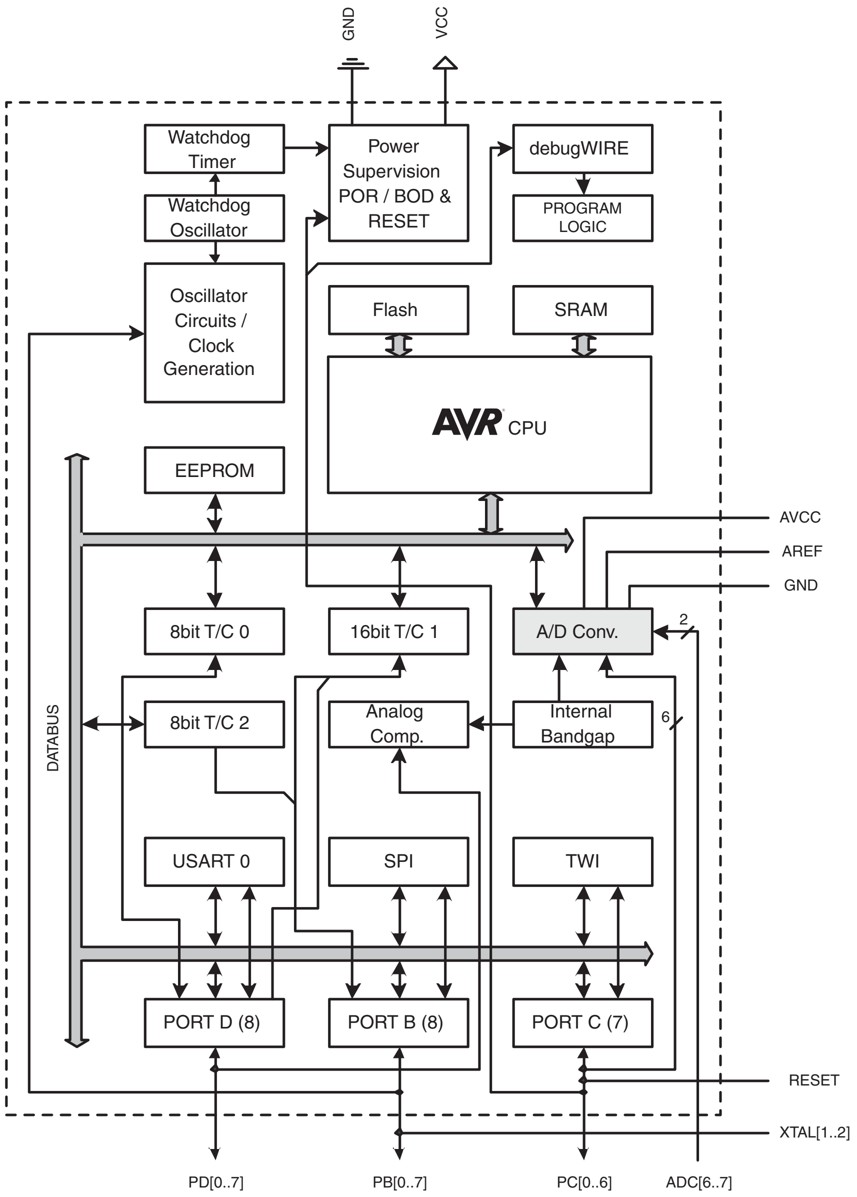
In the TQFP and QFN/MLF package, ADC7:6 serve as analog inputs to the A/D converter. These pins are powered from the analog supply and serve as 10-bit ADC channels.

2. Overview

The ATmega48A/PA/88A/PA/168A/PA/328/P is a low-power CMOS 8-bit microcontroller based on the AVR enhanced RISC architecture. By executing powerful instructions in a single clock cycle, the ATmega48A/PA/88A/PA/168A/PA/328/P achieves throughputs approaching 1 MIPS per MHz allowing the system designer to optimize power consumption versus processing speed.

2.1 Block Diagram

Figure 2-1. Block Diagram



The AVR core combines a rich instruction set with 32 general purpose working registers. All the 32 registers are directly connected to the Arithmetic Logic Unit (ALU), allowing two independent registers to be accessed in one single instruction executed in one clock cycle. The resulting architecture is more code efficient while achieving throughputs up to ten times faster than conventional CISC microcontrollers.

The ATmega48A/PA/88A/PA/168A/PA/328/P provides the following features: 4K/8Kbytes of In-System Programmable Flash with Read-While-Write capabilities, 256/512/512/1Kbytes EEPROM, 512/1K/1K/2Kbytes SRAM, 23 general purpose I/O lines, 32 general purpose working registers, three flexible Timer/Counters with compare modes, internal and external interrupts, a serial programmable USART, a byte-oriented 2-wire Serial Interface, an SPI serial port, a 6-channel 10-bit ADC (8 channels in TQFP and QFN/MLF packages), a programmable Watchdog Timer with internal Oscillator, and five software selectable power saving modes. The Idle mode stops the CPU while allowing the SRAM, Timer/Counters, USART, 2-wire Serial Interface, SPI port, and interrupt system to continue functioning. The Power-down mode saves the register contents but freezes the Oscillator, disabling all other chip functions until the next interrupt or hardware reset. In Power-save mode, the asynchronous timer continues to run, allowing the user to maintain a timer base while the rest of the device is sleeping. The ADC Noise Reduction mode stops the CPU and all I/O modules except asynchronous timer and ADC, to minimize switching noise during ADC conversions. In Standby mode, the crystal/resonator Oscillator is running while the rest of the device is sleeping. This allows very fast start-up combined with low power consumption.

Atmel® offers the QTouch® library for embedding capacitive touch buttons, sliders and wheels functionality into AVR® microcontrollers. The patented charge-transfer signal acquisition offers robust sensing and includes fully debounced reporting of touch keys and includes Adjacent Key Suppression® (AKS™) technology for unambiguous detection of key events. The easy-to-use QTouch Suite toolchain allows you to explore, develop and debug your own touch applications.

The device is manufactured using Atmel's high density non-volatile memory technology. The On-chip ISP Flash allows the program memory to be reprogrammed In-System through an SPI serial interface, by a conventional non-volatile memory programmer, or by an On-chip Boot program running on the AVR core. The Boot program can use any interface to download the application program in the Application Flash memory. Software in the Boot Flash section will continue to run while the Application Flash section is updated, providing true Read-While-Write operation. By combining an 8-bit RISC CPU with In-System Self-Programmable Flash on a monolithic chip, the Atmel ATmega48A/PA/88A/PA/168A/PA/328/P is a powerful microcontroller that provides a highly flexible and cost effective solution to many embedded control applications.

The ATmega48A/PA/88A/PA/168A/PA/328/P AVR is supported with a full suite of program and system development tools including: C Compilers, Macro Assemblers, Program Debugger/Simulators, In-Circuit Emulators, and Evaluation kits.

2.2 Comparison Between Processors

The ATmega48A/PA/88A/PA/168A/PA/328/P differ only in memory sizes, boot loader support, and interrupt vector sizes. Table 2-1 summarizes the different memory and interrupt vector sizes for the devices.

Table 2-1. Memory Size Summary

Device	Flash	EEPROM	RAM	Interrupt Vector Size
ATmega48A	4KBytes	256Bytes	512Bytes	1 instruction word/vector
ATmega48PA	4KBytes	256Bytes	512Bytes	1 instruction word/vector
ATmega88A	8KBytes	512Bytes	1KBytes	1 instruction word/vector
ATmega88PA	8KBytes	512Bytes	1KBytes	1 instruction word/vector
ATmega168A	16KBytes	512Bytes	1KBytes	2 instruction words/vector
ATmega168PA	16KBytes	512Bytes	1KBytes	2 instruction words/vector
ATmega328	32KBytes	1KBytes	2KBytes	2 instruction words/vector
ATmega328P	32KBytes	1KBytes	2KBytes	2 instruction words/vector

ATmega48A/PA/88A/PA/168A/PA/328/P support a real Read-While-Write Self-Programming mechanism. There is a separate Boot Loader Section, and the SPM instruction can only execute from there. In ATmega 48A/48PA there is no Read-While-Write support and no separate Boot Loader Section. The SPM instruction can execute from the entire Flash

3. Resources

A comprehensive set of development tools, application notes and datasheets are available for download on http://www.atmel.com/avr.

4. Data Retention

Reliability Qualification results show that the projected data retention failure rate is much less than 1 PPM over 20 years at 85°C or 100 years at 25°C.

5. About Code Examples

This documentation contains simple code examples that briefly show how to use various parts of the device. These code examples assume that the part specific header file is included before compilation. Be aware that not all C compiler vendors include bit definitions in the header files and interrupt handling in C is compiler dependent. Please confirm with the C compiler documentation for more details.

For I/O Registers located in extended I/O map, “IN”, “OUT”, “SBIS”, “SBIC”, “CBI”, and “SBI” instructions must be replaced with instructions that allow access to extended I/O. Typically “LDS” and “STS” combined with “SBRS”, “SBRC”, “SBR”, and “CBR”.

6. Capacitive Touch Sensing

The Atmel® QTouch® Library provides a simple to use solution to realize touch sensitive interfaces on most Atmel AVR® microcontrollers. The QTouch Library includes support for the Atmel QTouch and Atmel QMatrix® acquisition methods.

Touch sensing can be added to any application by linking the appropriate Atmel QTouch Library for the AVR Microcontroller. This is done by using a simple set of APIs to define the touch channels and sensors, and then calling the touch sensing APIs to retrieve the channel information and determine the touch sensor states.

The QTouch Library is FREE and downloadable from the Atmel website at the following location: www.atmel.com/qtouchlibrary. For implementation details and other information, refer to the Atmel QTouch Library User Guide - also available for download from Atmel website.

7. Register Summary

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Page
(0xFF)	Reserved	–	–	–	–	–	–	–	–	
(0xFE)	Reserved	–	–	–	–	–	–	–	–	
(0xFD)	Reserved	–	–	–	–	–	–	–	–	
(0xFC)	Reserved	–	–	–	–	–	–	–	–	
(0xFB)	Reserved	–	–	–	–	–	–	–	–	
(0xFA)	Reserved	–	–	–	–	–	–	–	–	
(0xF9)	Reserved	–	–	–	–	–	–	–	–	
(0xF8)	Reserved	–	–	–	–	–	–	–	–	
(0xF7)	Reserved	–	–	–	–	–	–	–	–	
(0xF6)	Reserved	–	–	–	–	–	–	–	–	
(0xF5)	Reserved	–	–	–	–	–	–	–	–	
(0xF4)	Reserved	–	–	–	–	–	–	–	–	
(0xF3)	Reserved	–	–	–	–	–	–	–	–	
(0xF2)	Reserved	–	–	–	–	–	–	–	–	
(0xF1)	Reserved	–	–	–	–	–	–	–	–	
(0xF0)	Reserved	–	–	–	–	–	–	–	–	
(0xEF)	Reserved	–	–	–	–	–	–	–	–	
(0xEE)	Reserved	–	–	–	–	–	–	–	–	
(0xED)	Reserved	–	–	–	–	–	–	–	–	
(0xEC)	Reserved	–	–	–	–	–	–	–	–	
(0xEB)	Reserved	–	–	–	–	–	–	–	–	
(0xEA)	Reserved	–	–	–	–	–	–	–	–	
(0xE9)	Reserved	–	–	–	–	–	–	–	–	
(0xE8)	Reserved	–	–	–	–	–	–	–	–	
(0xE7)	Reserved	–	–	–	–	–	–	–	–	
(0xE6)	Reserved	–	–	–	–	–	–	–	–	
(0xE5)	Reserved	–	–	–	–	–	–	–	–	
(0xE4)	Reserved	–	–	–	–	–	–	–	–	
(0xE3)	Reserved	–	–	–	–	–	–	–	–	
(0xE2)	Reserved	–	–	–	–	–	–	–	–	
(0xE1)	Reserved	–	–	–	–	–	–	–	–	
(0xE0)	Reserved	–	–	–	–	–	–	–	–	
(0xDF)	Reserved	–	–	–	–	–	–	–	–	
(0xDE)	Reserved	–	–	–	–	–	–	–	–	
(0xDD)	Reserved	–	–	–	–	–	–	–	–	
(0xDC)	Reserved	–	–	–	–	–	–	–	–	
(0xDB)	Reserved	–	–	–	–	–	–	–	–	
(0xDA)	Reserved	–	–	–	–	–	–	–	–	
(0xD9)	Reserved	–	–	–	–	–	–	–	–	
(0xD8)	Reserved	–	–	–	–	–	–	–	–	
(0xD7)	Reserved	–	–	–	–	–	–	–	–	
(0xD6)	Reserved	–	–	–	–	–	–	–	–	
(0xD5)	Reserved	–	–	–	–	–	–	–	–	
(0xD4)	Reserved	–	–	–	–	–	–	–	–	
(0xD3)	Reserved	–	–	–	–	–	–	–	–	
(0xD2)	Reserved	–	–	–	–	–	–	–	–	
(0xD1)	Reserved	–	–	–	–	–	–	–	–	
(0xD0)	Reserved	–	–	–	–	–	–	–	–	
(0xCF)	Reserved	–	–	–	–	–	–	–	–	
(0xCE)	Reserved	–	–	–	–	–	–	–	–	
(0xCD)	Reserved	–	–	–	–	–	–	–	–	
(0xCC)	Reserved	–	–	–	–	–	–	–	–	
(0xCB)	Reserved	–	–	–	–	–	–	–	–	
(0xCA)	Reserved	–	–	–	–	–	–	–	–	
(0xC9)	Reserved	–	–	–	–	–	–	–	–	
(0xC8)	Reserved	–	–	–	–	–	–	–	–	
(0xC7)	Reserved	–	–	–	–	–	–	–	–	
(0xC6)	UDR0	USART I/O Data Register								192
(0xC5)	UBRR0H					USART Baud Rate Register High				196
(0xC4)	UBRR0L	USART Baud Rate Register Low								196
(0xC3)	Reserved	–	–	–	–	–	–	–	–	
(0xC2)	UCSR0C	UMSEL01	UMSEL00	UPM01	UPM00	USBS0	UCSZ01 /UDORD0	UCSZ00 / UCPHA0	UCPOL0	194/205
(0xC1)	UCSR0B	RXCIE0	TXCIE0	UDRIE0	RXEN0	TXEN0	UCSZ02	RXB80	TXB80	193
(0xC0)	UCSR0A	RXC0	TXC0	UDRE0	FE0	DOR0	UPE0	U2X0	MPCM0	192
(0xBF)	Reserved	–	–	–	–	–	–	–	–	
(0xBE)	Reserved	–	–	–	–	–	–	–	–	

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Page
(0xBD)	TWAMR	TWAM6	TWAM5	TWAM4	TWAM3	TWAM2	TWAM1	TWAM0	–	235
(0xBC)	TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE	232
(0xBB)	TWDR	2-wire Serial Interface Data Register								234
(0xBA)	TWAR	TWA6	TWA5	TWA4	TWA3	TWA2	TWA1	TWA0	TWGCE	234
(0xB9)	TWSR	TWS7	TWS6	TWS5	TWS4	TWS3	–	TWPS1	TWPS0	233
(0xB8)	TWBR	2-wire Serial Interface Bit Rate Register								232
(0xB7)	Reserved	–		–	–	–	–	–	–	
(0xB6)	ASSR	–	EXCLK	AS2	TCN2UB	OCR2AUB	OCR2BUB	TCR2AUB	TCR2BUB	159
(0xB5)	Reserved	–	–	–	–	–	–	–	–	
(0xB4)	OCR2B	Timer/Counter2 Output Compare Register B								158
(0xB3)	OCR2A	Timer/Counter2 Output Compare Register A								158
(0xB2)	TCNT2	Timer/Counter2 (8-bit)								158
(0xB1)	TCCR2B	FOC2A	FOC2B	–	–	WGM22	CS22	CS21	CS20	157
(0xB0)	TCCR2A	COM2A1	COM2A0	COM2B1	COM2B0	–	–	WGM21	WGM20	154
(0xAF)	Reserved	–	–	–	–	–	–	–	–	
(0xAE)	Reserved	–	–	–	–	–	–	–	–	
(0xAD)	Reserved	–	–	–	–	–	–	–	–	
(0xAC)	Reserved	–	–	–	–	–	–	–	–	
(0xAB)	Reserved	–	–	–	–	–	–	–	–	
(0xAA)	Reserved	–	–	–	–	–	–	–	–	
(0xA9)	Reserved	–	–	–	–	–	–	–	–	
(0xA8)	Reserved	–	–	–	–	–	–	–	–	
(0xA7)	Reserved	–	–	–	–	–	–	–	–	
(0xA6)	Reserved	–	–	–	–	–	–	–	–	
(0xA5)	Reserved	–	–	–	–	–	–	–	–	
(0xA4)	Reserved	–	–	–	–	–	–	–	–	
(0xA3)	Reserved	–	–	–	–	–	–	–	–	
(0xA2)	Reserved	–	–	–	–	–	–	–	–	
(0xA1)	Reserved	–	–	–	–	–	–	–	–	
(0xA0)	Reserved	–	–	–	–	–	–	–	–	
(0x9F)	Reserved	–	–	–	–	–	–	–	–	
(0x9E)	Reserved	–	–	–	–	–	–	–	–	
(0x9D)	Reserved	–	–	–	–	–	–	–	–	
(0x9C)	Reserved	–	–	–	–	–	–	–	–	
(0x9B)	Reserved	–	–	–	–	–	–	–	–	
(0x9A)	Reserved	–	–	–	–	–	–	–	–	
(0x99)	Reserved	–	–	–	–	–	–	–	–	
(0x98)	Reserved	–	–	–	–	–	–	–	–	
(0x97)	Reserved	–	–	–	–	–	–	–	–	
(0x96)	Reserved	–	–	–	–	–	–	–	–	
(0x95)	Reserved	–	–	–	–	–	–	–	–	
(0x94)	Reserved	–	–	–	–	–	–	–	–	
(0x93)	Reserved	–	–	–	–	–	–	–	–	
(0x92)	Reserved	–	–	–	–	–	–	–	–	
(0x91)	Reserved	–	–	–	–	–	–	–	–	
(0x90)	Reserved	–	–	–	–	–	–	–	–	
(0x8F)	Reserved	–	–	–	–	–	–	–	–	
(0x8E)	Reserved	–	–	–	–	–	–	–	–	
(0x8D)	Reserved	–	–	–	–	–	–	–	–	
(0x8C)	Reserved	–	–	–	–	–	–	–	–	
(0x8B)	OCR1BH	Timer/Counter1 - Output Compare Register B High Byte								136
(0x8A)	OCR1BL	Timer/Counter1 - Output Compare Register B Low Byte								136
(0x89)	OCR1AH	Timer/Counter1 - Output Compare Register A High Byte								136
(0x88)	OCR1AL	Timer/Counter1 - Output Compare Register A Low Byte								136
(0x87)	ICR1H	Timer/Counter1 - Input Capture Register High Byte								136
(0x86)	ICR1L	Timer/Counter1 - Input Capture Register Low Byte								136
(0x85)	TCNT1H	Timer/Counter1 - Counter Register High Byte								135
(0x84)	TCNT1L	Timer/Counter1 - Counter Register Low Byte								135
(0x83)	Reserved	–	–	–	–	–	–	–	–	
(0x82)	TCCR1C	FOC1A	FOC1B	–	–	–	–	–	–	135
(0x81)	TCCR1B	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10	134
(0x80)	TCCR1A	COM1A1	COM1A0	COM1B1	COM1B0	–	–	WGM11	WGM10	132
(0x7F)	DIDR1	–	–	–	–	–	–	AIN1D	AIN0D	238
(0x7E)	DIDR0	–	–	ADC5D	ADC4D	ADC3D	ADC2D	ADC1D	ADC0D	253
(0x7D)	Reserved	–	–	–	–	–	–	–	–	
(0x7C)	ADMUX	REFS1	REFS0	ADLAR	–	MUX3	MUX2	MUX1	MUX0	250
(0x7B)	ADCSRB	–	ACME	–	–	–	ADTS2	ADTS1	ADTS0	253
(0x7A)	ADCSRA	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0	251

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Page
(0x79)	ADCH	ADC Data Register High byte								252
(0x78)	ADCL	ADC Data Register Low byte								252
(0x77)	Reserved	–	–	–	–	–	–	–	–	
(0x76)	Reserved	–	–	–	–	–	–	–	–	
(0x75)	Reserved	–	–	–	–	–	–	–	–	
(0x74)	Reserved	–	–	–	–	–	–	–	–	
(0x73)	Reserved	–	–	–	–	–	–	–	–	
(0x72)	Reserved	–	–	–	–	–	–	–	–	
(0x71)	Reserved	–	–	–	–	–	–	–	–	
(0x70)	TIMSK2	–	–	–	–	–	OCIE2B	OCIE2A	TOIE2	158
(0x6F)	TIMSK1	–	–	ICIE1	–	–	OCIE1B	OCIE1A	TOIE1	136
(0x6E)	TIMSK0	–	–	–	–	–	OCIE0B	OCIE0A	TOIE0	110
(0x6D)	PCMSK2	PCINT23	PCINT22	PCINT21	PCINT20	PCINT19	PCINT18	PCINT17	PCINT16	75
(0x6C)	PCMSK1	–	PCINT14	PCINT13	PCINT12	PCINT11	PCINT10	PCINT9	PCINT8	75
(0x6B)	PCMSK0	PCINT7	PCINT6	PCINT5	PCINT4	PCINT3	PCINT2	PCINT1	PCINT0	75
(0x6A)	Reserved	–	–	–	–	–	–	–	–	
(0x69)	EICRA	–	–	–	–	ISC11	ISC10	ISC01	ISC00	72
(0x68)	PCICR	–	–	–	–	–	PCIE2	PCIE1	PCIE0	
(0x67)	Reserved	–	–	–	–	–	–	–	–	
(0x66)	OSCCAL	Oscillator Calibration Register								38
(0x65)	Reserved	–	–	–	–	–	–	–	–	
(0x64)	PRR	PRTWI	PRTIM2	PRTIM0	–	PRTIM1	PRSPI	PRUSART0	PRADC	43
(0x63)	Reserved	–	–	–	–	–	–	–	–	
(0x62)	Reserved	–	–	–	–	–	–	–	–	
(0x61)	CLKPR	CLKPCE	–	–	–	CLKPS3	CLKPS2	CLKPS1	CLKPS0	38
(0x60)	WDTCSR	WDIF	WDIE	WDP3	WDCE	WDE	WDP2	WDP1	WDP0	55
0x3F (0x5F)	SREG	I	T	H	S	V	N	Z	C	10
0x3E (0x5E)	SPH	–	–	–	–	–	(SP10) [5.]	SP9	SP8	13
0x3D (0x5D)	SPL	SP7	SP6	SP5	SP4	SP3	SP2	SP1	SP0	13
0x3C (0x5C)	Reserved	–	–	–	–	–	–	–	–	
0x3B (0x5B)	Reserved	–	–	–	–	–	–	–	–	
0x3A (0x5A)	Reserved	–	–	–	–	–	–	–	–	
0x39 (0x59)	Reserved	–	–	–	–	–	–	–	–	
0x38 (0x58)	Reserved	–	–	–	–	–	–	–	–	
0x37 (0x57)	SPMCSR	SPMIE	(RWWSB)[5.]	SIGRD	(RWWSRE)[5.]	BLBSET	PGWRT	PGERS	SPMEN	280
0x36 (0x56)	Reserved	–	–	–	–	–	–	–	–	
0x35 (0x55)	MCUCR	–	BODS[(6)]	BODSE[(6)]	PUD	–	–	IVSEL	IVCE	46/69/92
0x34 (0x54)	MCUSR	–	–	–	–	WDRF	BORF	EXTRF	PORF	55
0x33 (0x53)	SMCR	–	–	–	–	SM2	SM1	SM0	SE	41
0x32 (0x52)	Reserved	–	–	–	–	–	–	–	–	
0x31 (0x51)	Reserved	–	–	–	–	–	–	–	–	
0x30 (0x50)	ACSR	ACD	ACBG	ACO	ACI	ACIE	ACIC	ACIS1	ACIS0	237
0x2F (0x4F)	Reserved	–	–	–	–	–	–	–	–	
0x2E (0x4E)	SPDR	SPI Data Register								170
0x2D (0x4D)	SPSR	SPIF	WCOL	–	–	–	–	–	SPI2X	169
0x2C (0x4C)	SPCR	SPIE	SPE	DORD	MSTR	CPOL	CPHA	SPR1	SPR0	168
0x2B (0x4B)	GPIOR2	General Purpose I/O Register 2								26
0x2A (0x4A)	GPIOR1	General Purpose I/O Register 1								26
0x29 (0x49)	Reserved	–	–	–	–	–	–	–	–	
0x28 (0x48)	OCR0B	Timer/Counter0 Output Compare Register B								
0x27 (0x47)	OCR0A	Timer/Counter0 Output Compare Register A								
0x26 (0x46)	TCNT0	Timer/Counter0 (8-bit)								
0x25 (0x45)	TCCR0B	FOC0A	FOC0B	–	–	WGM02	CS02	CS01	CS00	
0x24 (0x44)	TCCR0A	COM0A1	COM0A0	COM0B1	COM0B0	–	–	WGM01	WGM00	
0x23 (0x43)	GTCCR	TSM	–	–	–	–	–	PSRASY	PSRSYNC	141/160
0x22 (0x42)	EEARH	(EEPROM Address Register High Byte) [5.]								22
0x21 (0x41)	EEARL	EEPROM Address Register Low Byte								22
0x20 (0x40)	EEDR	EEPROM Data Register								22
0x1F (0x3F)	EECR	–	–	EEPM1	EEPM0	EERIE	EEMPE	EEPE	EERE	22
0x1E (0x3E)	GPIOR0	General Purpose I/O Register 0								26
0x1D (0x3D)	EIMSK	–	–	–	–	–	–	INT1	INT0	73
0x1C (0x3C)	EIFR	–	–	–	–	–	–	INTF1	INTF0	73
0x1B (0x3B)	PCIFR	–	–	–	–	–	PCIF2	PCIF1	PCIF0	
0x1A (0x3A)	Reserved	–	–	–	–	–	–	–	–	
0x19 (0x39)	Reserved	–	–	–	–	–	–	–	–	
0x18 (0x38)	Reserved	–	–	–	–	–	–	–	–	
0x17 (0x37)	TIFR2	–	–	–	–	–	OCF2B	OCF2A	TOV2	159
0x16 (0x36)	TIFR1	–	–	ICF1	–	–	OCF1B	OCF1A	TOV1	137

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Page
0x15 (0x35)	TIFR0	–	–	–	–	–	OCF0B	OCF0A	TOV0	
0x14 (0x34)	Reserved	–	–	–	–	–	–	–	–	
0x13 (0x33)	Reserved	–	–	–	–	–	–	–	–	
0x12 (0x32)	Reserved	–	–	–	–	–	–	–	–	
0x11 (0x31)	Reserved	–	–	–	–	–	–	–	–	
0x10 (0x30)	Reserved	–	–	–	–	–	–	–	–	
0x0F (0x2F)	Reserved	–	–	–	–	–	–	–	–	
0x0E (0x2E)	Reserved	–	–	–	–	–	–	–	–	
0x0D (0x2D)	Reserved	–	–	–	–	–	–	–	–	
0x0C (0x2C)	Reserved	–	–	–	–	–	–	–	–	
0x0B (0x2B)	PORTD	PORTD7	PORTD6	PORTD5	PORTD4	PORTD3	PORTD2	PORTD1	PORTD0	93
0x0A (0x2A)	DDRD	DDD7	DDD6	DDD5	DDD4	DDD3	DDD2	DDD1	DDD0	93
0x09 (0x29)	PIND	PIND7	PIND6	PIND5	PIND4	PIND3	PIND2	PIND1	PIND0	93
0x08 (0x28)	PORTC	–	PORTC6	PORTC5	PORTC4	PORTC3	PORTC2	PORTC1	PORTC0	92
0x07 (0x27)	DDRC	–	DDC6	DDC5	DDC4	DDC3	DDC2	DDC1	DDC0	92
0x06 (0x26)	PINC	–	PINC6	PINC5	PINC4	PINC3	PINC2	PINC1	PINC0	93
0x05 (0x25)	PORTB	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0	92
0x04 (0x24)	DDRB	DDB7	DDB6	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0	92
0x03 (0x23)	PINB	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0	92
0x02 (0x22)	Reserved	–	–	–	–	–	–	–	–	
0x01 (0x21)	Reserved	–	–	–	–	–	–	–	–	
0x0 (0x20)	Reserved	–	–	–	–	–	–	–	–	

Note:
1. For compatibility with future devices, reserved bits should be written to zero if accessed. Reserved I/O memory addresses should never be written.
2. I/O Registers within the address range 0x00 - 0x1F are directly bit-accessible using the SBI and CBI instructions. In these registers, the value of single bits can be checked by using the SBIS and SBIC instructions.
3. Some of the Status Flags are cleared by writing a logical one to them. Note that, unlike most other AVRs, the CBI and SBI instructions will only operate on the specified bit, and can therefore be used on registers containing such Status Flags. The CBI and SBI instructions work with registers 0x00 to 0x1F only.
4. When using the I/O specific commands IN and OUT, the I/O addresses 0x00 - 0x3F must be used. When addressing I/O Registers as data space using LD and ST instructions, 0x20 must be added to these addresses. The ATmega48A/PA/88A/PA/168A/PA/328/P is a complex microcontroller with more peripheral units than can be supported within the 64 location reserved in Opcode for the IN and OUT instructions. For the Extended I/O space from 0x60 - 0xFF in SRAM, only the ST/STS/STD and LD/LDS/LDD instructions can be used.
5. Only valid for ATmega88A/88PA/168A/168PA/328/328P.
6. BODS and BODSE only available for picoPower devices ATmega48PA/88PA/168PA/328P

8. Instruction Set Summary

Mnemonics	Operands	Description	Operation	Flags	#Clocks
ARITHMETIC AND LOGIC INSTRUCTIONS					
ADD	Rd, Rr	Add two Registers	Rd ← Rd + Rr	Z,C,N,V,H	1
ADC	Rd, Rr	Add with Carry two Registers	Rd ← Rd + Rr + C	Z,C,N,V,H	1
ADIW	Rdl,K	Add Immediate to Word	Rdh:Rdl ← Rdh:Rdl + K	Z,C,N,V,S	2
SUB	Rd, Rr	Subtract two Registers	Rd ← Rd - Rr	Z,C,N,V,H	1
SUBI	Rd, K	Subtract Constant from Register	Rd ← Rd - K	Z,C,N,V,H	1
SBC	Rd, Rr	Subtract with Carry two Registers	Rd ← Rd - Rr - C	Z,C,N,V,H	1
SBCI	Rd, K	Subtract with Carry Constant from Reg.	Rd ← Rd - K - C	Z,C,N,V,H	1
SBIW	Rdl,K	Subtract Immediate from Word	Rdh:Rdl ← Rdh:Rdl - K	Z,C,N,V,S	2
AND	Rd, Rr	Logical AND Registers	Rd ← Rd • Rr	Z,N,V	1
ANDI	Rd, K	Logical AND Register and Constant	Rd ← Rd • K	Z,N,V	1
OR	Rd, Rr	Logical OR Registers	Rd ← Rd v Rr	Z,N,V	1
ORI	Rd, K	Logical OR Register and Constant	Rd ← Rd v K	Z,N,V	1
EOR	Rd, Rr	Exclusive OR Registers	Rd ← Rd ⊕ Rr	Z,N,V	1
COM	Rd	One's Complement	Rd ← 0xFF – Rd	Z,C,N,V	1
NEG	Rd	Two's Complement	Rd ← 0x00 – Rd	Z,C,N,V,H	1
SBR	Rd,K	Set Bit(s) in Register	Rd ← Rd v K	Z,N,V	1
CBR	Rd,K	Clear Bit(s) in Register	Rd ← Rd • (0xFF - K)	Z,N,V	1
INC	Rd	Increment	Rd ← Rd + 1	Z,N,V	1
DEC	Rd	Decrement	Rd ← Rd – 1	Z,N,V	1
TST	Rd	Test for Zero or Minus	Rd ← Rd • Rd	Z,N,V	1
CLR	Rd	Clear Register	Rd ← Rd ⊕ Rd	Z,N,V	1
SER	Rd	Set Register	Rd ← 0xFF	None	1
MUL	Rd, Rr	Multiply Unsigned	R1:R0 ← Rd x Rr	Z,C	2
MULS	Rd, Rr	Multiply Signed	R1:R0 ← Rd x Rr	Z,C	2
MULSU	Rd, Rr	Multiply Signed with Unsigned	R1:R0 ← Rd x Rr	Z,C	2
FMUL	Rd, Rr	Fractional Multiply Unsigned	R1:R0 ← (Rd x Rr) << 1	Z,C	2
FMULS	Rd, Rr	Fractional Multiply Signed	R1:R0 ← (Rd x Rr) << 1	Z,C	2
FMULSU	Rd, Rr	Fractional Multiply Signed with Unsigned	R1:R0 ← (Rd x Rr) << 1	Z,C	2
BRANCH INSTRUCTIONS					
RJMP	k	Relative Jump	PC ← PC + k + 1	None	2
IJMP		Indirect Jump to (Z)	PC ← Z	None	2
JMP[(1)]	k	Direct Jump	PC ← k	None	3
RCALL	k	Relative Subroutine Call	PC ← PC + k + 1	None	3
ICALL		Indirect Call to (Z)	PC ← Z	None	3
CALL[(1)]	k	Direct Subroutine Call	PC ← k	None	4
RET		Subroutine Return	PC ← STACK	None	4
RETI		Interrupt Return	PC ← STACK	I	4
CPSE	Rd,Rr	Compare, Skip if Equal	if (Rd = Rr) PC ← PC + 2 or 3	None	1/2/3
CP	Rd,Rr	Compare	Rd – Rr	Z, N,V,C,H	1
CPC	Rd,Rr	Compare with Carry	Rd – Rr – C	Z, N,V,C,H	1
CPI	Rd,K	Compare Register with Immediate	Rd – K	Z, N,V,C,H	1
SBRC	Rr, b	Skip if Bit in Register Cleared	if (Rr(b)=0) PC ← PC + 2 or 3	None	1/2/3
SBRS	Rr, b	Skip if Bit in Register is Set	if (Rr(b)=1) PC ← PC + 2 or 3	None	1/2/3
SBIC	P, b	Skip if Bit in I/O Register Cleared	if (P(b)=0) PC ← PC + 2 or 3	None	1/2/3
SBIS	P, b	Skip if Bit in I/O Register is Set	if (P(b)=1) PC ← PC + 2 or 3	None	1/2/3
BRBS	s, k	Branch if Status Flag Set	if (SREG(s) = 1) then PC←PC+k + 1	None	1/2
BRBC	s, k	Branch if Status Flag Cleared	if (SREG(s) = 0) then PC←PC+k + 1	None	1/2
BREQ	k	Branch if Equal	if (Z = 1) then PC ← PC + k + 1	None	1/2
BRNE	k	Branch if Not Equal	if (Z = 0) then PC ← PC + k + 1	None	1/2
BRCS	k	Branch if Carry Set	if (C = 1) then PC ← PC + k + 1	None	1/2
BRCC	k	Branch if Carry Cleared	if (C = 0) then PC ← PC + k + 1	None	1/2
BRSH	k	Branch if Same or Higher	if (C = 0) then PC ← PC + k + 1	None	1/2
BRLO	k	Branch if Lower	if (C = 1) then PC ← PC + k + 1	None	1/2
BRMI	k	Branch if Minus	if (N = 1) then PC ← PC + k + 1	None	1/2
BRPL	k	Branch if Plus	if (N = 0) then PC ← PC + k + 1	None	1/2
BRGE	k	Branch if Greater or Equal, Signed	if (N ⊕ V= 0) then PC ← PC + k + 1	None	1/2
BRLT	k	Branch if Less Than Zero, Signed	if (N ⊕ V= 1) then PC ← PC + k + 1	None	1/2
BRHS	k	Branch if Half Carry Flag Set	if (H = 1) then PC ← PC + k + 1	None	1/2
BRHC	k	Branch if Half Carry Flag Cleared	if (H = 0) then PC ← PC + k + 1	None	1/2
BRTS	k	Branch if T Flag Set	if (T = 1) then PC ← PC + k + 1	None	1/2
BRTC	k	Branch if T Flag Cleared	if (T = 0) then PC ← PC + k + 1	None	1/2
BRVS	k	Branch if Overflow Flag is Set	if (V = 1) then PC ← PC + k + 1	None	1/2
BRVC	k	Branch if Overflow Flag is Cleared	if (V = 0) then PC ← PC + k + 1	None	1/2
BRIE	k	Branch if Interrupt Enabled	if (I = 1) then PC ← PC + k + 1	None	1/2
BRID	k	Branch if Interrupt Disabled	if (I = 0) then PC ← PC + k + 1	None	1/2

Mnemonics	Operands	Description	Operation	Flags	#Clocks
BIT AND BIT-TEST INSTRUCTIONS					
SBI	P,b	Set Bit in I/O Register	I/O(P,b) ← 1	None	2
CBI	P,b	Clear Bit in I/O Register	I/O(P,b) ← 0	None	2
LSL	Rd	Logical Shift Left	Rd(n+1) ← Rd(n), Rd(0) ← 0	Z,C,N,V	1
LSR	Rd	Logical Shift Right	Rd(n) ← Rd(n+1), Rd(7) ← 0	Z,C,N,V	1
ROL	Rd	Rotate Left Through Carry	Rd(0)←C,Rd(n+1)← Rd(n),C←Rd(7)	Z,C,N,V	1
ROR	Rd	Rotate Right Through Carry	Rd(7)←C,Rd(n)← Rd(n+1),C←Rd(0)	Z,C,N,V	1
ASR	Rd	Arithmetic Shift Right	Rd(n) ← Rd(n+1), n=0...6	Z,C,N,V	1
SWAP	Rd	Swap Nibbles	Rd(3...0)←Rd(7...4),Rd(7...4)←Rd(3...0)	None	1
BSET	s	Flag Set	SREG(s) ← 1	SREG(s)	1
BCLR	s	Flag Clear	SREG(s) ← 0	SREG(s)	1
BST	Rr, b	Bit Store from Register to T	T ← Rr(b)	T	1
BLD	Rd, b	Bit load from T to Register	Rd(b) ← T	None	1
SEC		Set Carry	C ← 1	C	1
CLC		Clear Carry	C ← 0	C	1
SEN		Set Negative Flag	N ← 1	N	1
CLN		Clear Negative Flag	N ← 0	N	1
SEZ		Set Zero Flag	Z ← 1	Z	1
CLZ		Clear Zero Flag	Z ← 0	Z	1
SEI		Global Interrupt Enable	I ← 1	I	1
CLI		Global Interrupt Disable	I ← 0	I	1
SES		Set Signed Test Flag	S ← 1	S	1
CLS		Clear Signed Test Flag	S ← 0	S	1
SEV		Set Twos Complement Overflow.	V ← 1	V	1
CLV		Clear Twos Complement Overflow	V ← 0	V	1
SET		Set T in SREG	T ← 1	T	1
CLT		Clear T in SREG	T ← 0	T	1
SEH		Set Half Carry Flag in SREG	H ← 1	H	1
CLH		Clear Half Carry Flag in SREG	H ← 0	H	1
DATA TRANSFER INSTRUCTIONS					
MOV	Rd, Rr	Move Between Registers	Rd ← Rr	None	1
MOVW	Rd, Rr	Copy Register Word	Rd+1:Rd ← Rr+1:Rr	None	1
LDI	Rd, K	Load Immediate	Rd ← K	None	1
LD	Rd, X	Load Indirect	Rd ← (X)	None	2
LD	Rd, X+	Load Indirect and Post-Inc.	Rd ← (X), X ← X + 1	None	2
LD	Rd, - X	Load Indirect and Pre-Dec.	X ← X - 1, Rd ← (X)	None	2
LD	Rd, Y	Load Indirect	Rd ← (Y)	None	2
LD	Rd, Y+	Load Indirect and Post-Inc.	Rd ← (Y), Y ← Y + 1	None	2
LD	Rd, - Y	Load Indirect and Pre-Dec.	Y ← Y - 1, Rd ← (Y)	None	2
LDD	Rd,Y+q	Load Indirect with Displacement	Rd ← (Y + q)	None	2
LD	Rd, Z	Load Indirect	Rd ← (Z)	None	2
LD	Rd, Z+	Load Indirect and Post-Inc.	Rd ← (Z), Z ← Z+1	None	2
LD	Rd, -Z	Load Indirect and Pre-Dec.	Z ← Z - 1, Rd ← (Z)	None	2
LDD	Rd, Z+q	Load Indirect with Displacement	Rd ← (Z + q)	None	2
LDS	Rd, k	Load Direct from SRAM	Rd ← (k)	None	2
ST	X, Rr	Store Indirect	(X) ← Rr	None	2
ST	X+, Rr	Store Indirect and Post-Inc.	(X) ← Rr, X ← X + 1	None	2
ST	- X, Rr	Store Indirect and Pre-Dec.	X ← X - 1, (X) ← Rr	None	2
ST	Y, Rr	Store Indirect	(Y) ← Rr	None	2
ST	Y+, Rr	Store Indirect and Post-Inc.	(Y) ← Rr, Y ← Y + 1	None	2
ST	- Y, Rr	Store Indirect and Pre-Dec.	Y ← Y - 1, (Y) ← Rr	None	2
STD	Y+q,Rr	Store Indirect with Displacement	(Y + q) ← Rr	None	2
ST	Z, Rr	Store Indirect	(Z) ← Rr	None	2
ST	Z+, Rr	Store Indirect and Post-Inc.	(Z) ← Rr, Z ← Z + 1	None	2
ST	-Z, Rr	Store Indirect and Pre-Dec.	Z ← Z - 1, (Z) ← Rr	None	2
STD	Z+q,Rr	Store Indirect with Displacement	(Z + q) ← Rr	None	2
STS	k, Rr	Store Direct to SRAM	(k) ← Rr	None	2
LPM		Load Program Memory	R0 ← (Z)	None	3
LPM	Rd, Z	Load Program Memory	Rd ← (Z)	None	3
LPM	Rd, Z+	Load Program Memory and Post-Inc	Rd ← (Z), Z ← Z+1	None	3
SPM		Store Program Memory	(Z) ← R1:R0	None	-
IN	Rd, P	In Port	Rd ← P	None	1
OUT	P, Rr	Out Port	P ← Rr	None	1
PUSH	Rr	Push Register on Stack	STACK ← Rr	None	2
POP	Rd	Pop Register from Stack	Rd ← STACK	None	2
MCU CONTROL INSTRUCTIONS					
NOP		No Operation		None	1
SLEEP		Sleep	(see specific descr. for Sleep function)	None	1

Mnemonics	Operands	Description	Operation	Flags	#Clocks
WDR		Watchdog Reset	(see specific descr. for WDR/timer)	None	1
BREAK		Break	For On-chip Debug Only	None	N/A

Note: 1. These instructions are only available in ATmega168PA and ATmega328P.

9. Ordering Information

9.1 ATmega48A

Speed (MHz)	Power Supply (V)	Ordering Code[2]	Package[1]	Operational Range[6]
20[3]	1.8 - 5.5	ATmega48A-AU ATmega48A-AUR[5] ATmega48A-CCU ATmega48A-CCUR[5] ATmega48A-MMH[4] ATmega48A-MMHR[4][5] ATmega48A-MU ATmega48A-MUR[5] ATmega48A-PU	32A 32A 32CC1 32CC1 28M1 28M1 32M1-A 32M1-A 28P3	Industrial (-40°C to 85°C)

Note:
1. This device can also be supplied in wafer form. Please contact your local Atmel sales office for detailed ordering information and minimum quantities.
2. Pb-free packaging complies to the European Directive for Restriction of Hazardous Substances (RoHS directive). Also Halide free and fully Green.
3. See ”Speed Grades” on page 308.
4. NiPdAu Lead Finish.
5. Tape & Reel.
6. Use ”ATmega48PA” on page 17, industrial (-40°C to 105°C) as the ATmega48A (-40°C to 105°C) is not presently offered.

Package Type	
32A	32-lead, Thin (1.0 mm) Plastic Quad Flat Package (TQFP)
32CC1	32-ball, 4 x 4 x 0.6 mm package, ball pitch 0.5 mm, Ultra Thin, Fine-Pitch Ball Grill Array (UFBGA)
28M1	28-pad, 4 x 4 x 1.0 body, Lead Pitch 0.45 mm Quad Flat No-Lead/Micro Lead Frame Package (QFN/MLF)
32M1-A	32-pad, 5 x 5 x 1.0 body, Lead Pitch 0.50 mm Quad Flat No-Lead/Micro Lead Frame Package (QFN/MLF)
28P3	28-lead, 0.300” Wide, Plastic Dual Inline Package (PDIP)

9.2 ATmega48PA

Speed (MHz)[3]	Power Supply (V)	Ordering Code[2]	Package[1]	Operational Range
20	1.8 - 5.5	ATmega48PA-AU ATmega48PA-AUR[5] ATmega48PA-CCU ATmega48PA-CCUR[5] ATmega48PA-MMH[4] ATmega48PA-MMHR[4][5] ATmega48PA-MU ATmega48PA-MUR[5] ATmega48PA-PU	32A 32A 32CC1 32CC1 28M1 28M1 32M1-A 32M1-A 28P3	Industrial (-40°C to 85°C)
		ATmega48PA-AN ATmega48PA-ANR[5] ATmega48PA-MMN[4] ATmega48PA-MMNR[4][5] ATmega48PA-MN ATmega48PA-MNR[5] ATmega48PA-PN	32A 32A 28M1 28M1 32M1-A 32M1-A 28P3	Industrial (-40°C to 105°C)

Note:
1. This device can also be supplied in wafer form. Please contact your local Atmel sales office for detailed ordering information and minimum quantities.
2. Pb-free packaging complies to the European Directive for Restriction of Hazardous Substances (RoHS directive). Also Halide free and fully Green.
3. See "Speed Grades" on page 308.
4. NiPdAu Lead Finish.
5. Tape & Reel.

Package Type	
32A	32-lead, Thin (1.0mm) Plastic Quad Flat Package (TQFP)
32CC1	32-ball, 4 x 4 x 0.6mm package, ball pitch 0.5mm, Ultra Thin, Fine-Pitch Ball Grill Array (UFBGA)
28M1	28-pad, 4 x 4 x 1.0 body, Lead Pitch 0.45mm Quad Flat No-Lead/Micro Lead Frame Package (QFN/MLF)
32M1-A	32-pad, 5 x 5 x 1.0 body, Lead Pitch 0.50mm Quad Flat No-Lead/Micro Lead Frame Package (QFN/MLF)
28P3	28-lead, 0.300" Wide, Plastic Dual Inline Package (PDIP)

9.3 ATmega88A

Speed (MHz)	Power Supply (V)	Ordering Code[2]	Package[1]	Operational Range[6]
20[3]	1.8 - 5.5	ATmega88A-AU ATmega88A-AUR[5] ATmega88A-CCU ATmega88A-CCUR[5] ATmega88A-MMH[4] ATmega88A-MMHR[4][5] ATmega88A-MU ATmega88A-MUR[5] ATmega88A-PU	32A 32A 32CC1 32CC1 28M1 28M1 32M1-A 32M1-A 28P3	Industrial (-40°C to 85°C)

Note: 1. This device can also be supplied in wafer form. Please contact your local Atmel sales office for detailed ordering information and minimum quantities.
2. Pb-free packaging complies to the European Directive for Restriction of Hazardous Substances (RoHS directive).Also Halide free and fully Green.
3. See ”Speed Grades” on page 308.
4. NiPdAu Lead Finish.
5. Tape & Reel.
6. Use ”ATmega88PA” on page 19, industrial (-40°C to 105°C) as the ATmega48A (-40°C to 105°C) is not presently offered.

Package Type	
32A	32-lead, Thin (1.0mm) Plastic Quad Flat Package (TQFP)
32CC1	32-ball, 4 x 4 x 0.6mm package, ball pitch 0.5mm, Ultra Thin, Fine-Pitch Ball Grill Array (UFBGA)
28M1	28-pad, 4 x 4 x 1.0 body, Lead Pitch 0.45mm Quad Flat No-Lead/Micro Lead Frame Package (QFN/MLF)
32M1-A	32-pad, 5 x 5 x 1.0 body, Lead Pitch 0.50mm Quad Flat No-Lead/Micro Lead Frame Package (QFN/MLF)
28P3	28-lead, 0.300” Wide, Plastic Dual Inline Package (PDIP)

9.4 ATmega88PA

Speed (MHz)[3]	Power Supply (V)	Ordering Code[2]	Package[1]	Operational Range
20	1.8 - 5.5	ATmega88PA-AU ATmega88PA-AUR[5] ATmega88PA-CCU ATmega88PA-CCUR[5] ATmega88PA-MMH[4] ATmega88PA-MMHR[4][5] ATmega88PA-MU ATmega88PA-MUR[5] ATmega88PA-PU	32A 32A 32CC1 32CC1 28M1 28M1 32M1-A 32M1-A 28P3	Industrial (-40°C to 85°C)
		ATmega88PA-AN ATmega88PA-ANR[5] ATmega88PA-MMN[4] ATmega88PA-MMNR[4][5] ATmega88PA-MN ATmega88PA-MNR[5] ATmega88PA-PN	32A 32A 28M1 28M1 32M1-A 32M1-A 28P3	Industrial (-40°C to 105°C)

Note:
1. This device can also be supplied in wafer form. Please contact your local Atmel sales office for detailed ordering information and minimum quantities.
2. Pb-free packaging complies to the European Directive for Restriction of Hazardous Substances (RoHS directive).Also Halide free and fully Green.
3. See "Speed Grades" on page 308.
4. NiPdAu Lead Finish.
5. Tape & Reel.

Package Type	
32A	32-lead, Thin (1.0mm) Plastic Quad Flat Package (TQFP)
32CC1	32-ball, 4 x 4 x 0.6mm package, ball pitch 0.5 mm, Ultra Thin, Fine-Pitch Ball Grill Array (UFBGA)
28M1	28-pad, 4 x 4 x 1.0 body, Lead Pitch 0.45 mm Quad Flat No-Lead/Micro Lead Frame Package (QFN/MLF)
32M1-A	32-pad, 5 x 5 x 1.0 body, Lead Pitch 0.50 mm Quad Flat No-Lead/Micro Lead Frame Package (QFN/MLF)
28P3	28-lead, 0.300" Wide, Plastic Dual Inline Package (PDIP)

9.5 ATmega168A

Speed (MHz)[3]	Power Supply (V)	Ordering Code[2]	Package[1]	Operational Range[6]
20	1.8 - 5.5	ATmega168A-AU ATmega168A-AUR[5] ATmega168A-CCU ATmega168A-CCUR[5] ATmega168A-MMH[4] ATmega168A-MMHR[4][5] ATmega168A-MU ATmega168A-MUR[5] ATmega168A-PU	32A 32A 32CC1 32CC1 28M1 28M1 32M1-A 32M1-A 28P3	Industrial (-40°C to 85°C)

Note: 1. This device can also be supplied in wafer form. Please contact your local Atmel sales office for detailed ordering information and minimum quantities.

2. Pb-free packaging complies to the European Directive for Restriction of Hazardous Substances (RoHS directive).Also Halide free and fully Green.
3. See "Speed Grades" on page 308
4. NiPdAu Lead Finish.
5. Tape & Reel.
6. Use "ATmega168PA" on page 21, industrial (-40°C to 105°C) as the ATmega48A (-40°C to 105°C) is not presently offered.

Package Type	
32A	32-lead, Thin (1.0mm) Plastic Quad Flat Package (TQFP)
32CC1	32-ball, 4 x 4 x 0.6 mm package, ball pitch 0.5mm, Ultra Thin, Fine-Pitch Ball Grill Array (UFBGA)
28M1	28-pad, 4 x 4 x 1.0 body, Lead Pitch 0.45mm Quad Flat No-Lead/Micro Lead Frame Package (QFN/MLF)
32M1-A	32-pad, 5 x 5 x 1.0 body, Lead Pitch 0.50mm Quad Flat No-Lead/Micro Lead Frame Package (QFN/MLF)
28P3	28-lead, 0.300" Wide, Plastic Dual Inline Package (PDIP)

9.6 ATmega168PA

Speed (MHz)[3]	Power Supply (V)	Ordering Code[2]	Package[1]	Operational Range
20	1.8 - 5.5	ATmega168PA-AU ATmega168PA-AUR[5] ATmega168PA-CCU ATmega168PA-CCUR[5] ATmega168PA-MMH[4] ATmega168PA-MMHR[4][5] ATmega168PA-MU ATmega168PA-MUR[5] ATmega168PA-PU	32A 32A 32CC1 32CC1 28M1 28M1 32M1-A 32M1-A 28P3	Industrial (-40°C to 85°C)
20	1.8 - 5.5	ATmega168PA-AN ATmega168PA-ANR[5] ATmega168PA-MN ATmega168PA-MNR[5] ATmega168PA-PN	32A 32A 32M1-A 32M1-A 28P3	Industrial (-40°C to 105°C)

Note:
1. This device can also be supplied in wafer form. Please contact your local Atmel sales office for detailed ordering information and minimum quantities.
2. Pb-free packaging complies to the European Directive for Restriction of Hazardous Substances (RoHS directive).Also Halide free and fully Green.
3. See "Speed Grades" on page 308.
4. NiPdAu Lead Finish.
5. Tape & Reel.

Package Type	
32A	32-lead, Thin (1.0mm) Plastic Quad Flat Package (TQFP)
32CC1	32-ball, 4 x 4 x 0.6mm package, ball pitch 0.5mm, Ultra Thin, Fine-Pitch Ball Grill Array (UFBGA)
28M1	28-pad, 4 x 4 x 1.0 body, Lead Pitch 0.45mm Quad Flat No-Lead/Micro Lead Frame Package (QFN/MLF)
32M1-A	32-pad, 5 x 5 x 1.0 body, Lead Pitch 0.50mm Quad Flat No-Lead/Micro Lead Frame Package (QFN/MLF)
28P3	28-lead, 0.300" Wide, Plastic Dual Inline Package (PDIP)

9.7 ATmega328

Speed (MHz)	Power Supply (V)	Ordering Code[2]	Package[1]	Operational Range[6]
20[3]	1.8 - 5.5	ATmega328-AU ATmega328-AUR[5] ATmega328-MMH[4] ATmega328-MMHR[4][5] ATmega328-MU ATmega328-MUR[5] ATmega328-PU	32A 32A 28M1 28M1 32M1-A 32M1-A 28P3	Industrial (-40°C to 85°C)

Note: 1. This device can also be supplied in wafer form. Please contact your local Atmel sales office for detailed ordering information and minimum quantities.
2. Pb-free packaging complies to the European Directive for Restriction of Hazardous Substances (RoHS directive).Also Halide free and fully Green.
3. See Figure 29-1 on page 309.
4. NiPdAu Lead Finish.
5. Tape & Reel
6. Use ”ATmega328P” on page 23, industrial (-40°C to 105°C) as the ATmega48A (-40°C to 105°C) is not presently offered.

Package Type	
32A	32-lead, Thin (1.0mm) Plastic Quad Flat Package (TQFP)
28M1	28-pad, 4 x 4 x 1.0 body, Lead Pitch 0.45mm Quad Flat No-Lead/Micro Lead Frame Package (QFN/MLF)
28P3	28-lead, 0.300” Wide, Plastic Dual Inline Package (PDIP)
32M1-A	32-pad, 5 x 5 x 1.0 body, Lead Pitch 0.50mm Quad Flat No-Lead/Micro Lead Frame Package (QFN/MLF)

9.8 ATmega328P

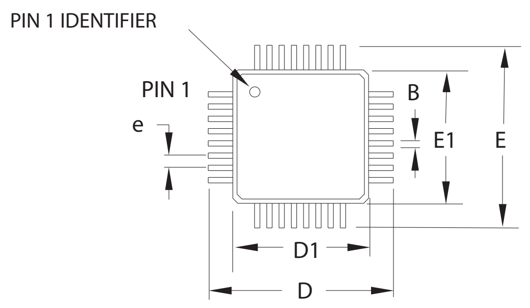
Speed (MHz)[3]	Power Supply (V)	Ordering Code[2]	Package[1]	Operational Range
20	1.8 - 5.5	ATmega328P-AU ATmega328P-AUR[5] ATmega328P-MMH[4] ATmega328P-MMHR[4][5] ATmega328P-MU ATmega328P-MUR[5] ATmega328P-PU	32A 32A 28M1 28M1 32M1-A 32M1-A 28P3	Industrial (-40°C to 85°C)
		ATmega328P-AN ATmega328P-ANR[5] ATmega328P-MN ATmega328P-MNR[5] ATmega328P-PN	32A 32A 32M1-A 32M1-A 28P3	Industrial (-40°C to 105°C)

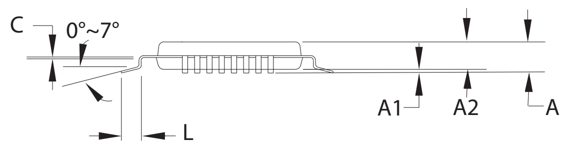
Note:
1. This device can also be supplied in wafer form. Please contact your local Atmel sales office for detailed ordering information and minimum quantities.
2. Pb-free packaging complies to the European Directive for Restriction of Hazardous Substances (RoHS directive).Also Halide free and fully Green.
3. See Figure 29-1 on page 309.
4. NiPdAu Lead Finish.
5. Tape & Reel.

Package Type	
32A	32-lead, Thin (1.0mm) Plastic Quad Flat Package (TQFP)
28M1	28-pad, 4 x 4 x 1.0 body, Lead Pitch 0.45mm Quad Flat No-Lead/Micro Lead Frame Package (QFN/MLF)
28P3	28-lead, 0.300" Wide, Plastic Dual Inline Package (PDIP)
32M1-A	32-pad, 5 x 5 x 1.0 body, Lead Pitch 0.50mm Quad Flat No-Lead/Micro Lead Frame Package (QFN/MLF)

10. Packaging Information

10.1 32A





COMMON DIMENSIONS
(Unit of measure = mm)

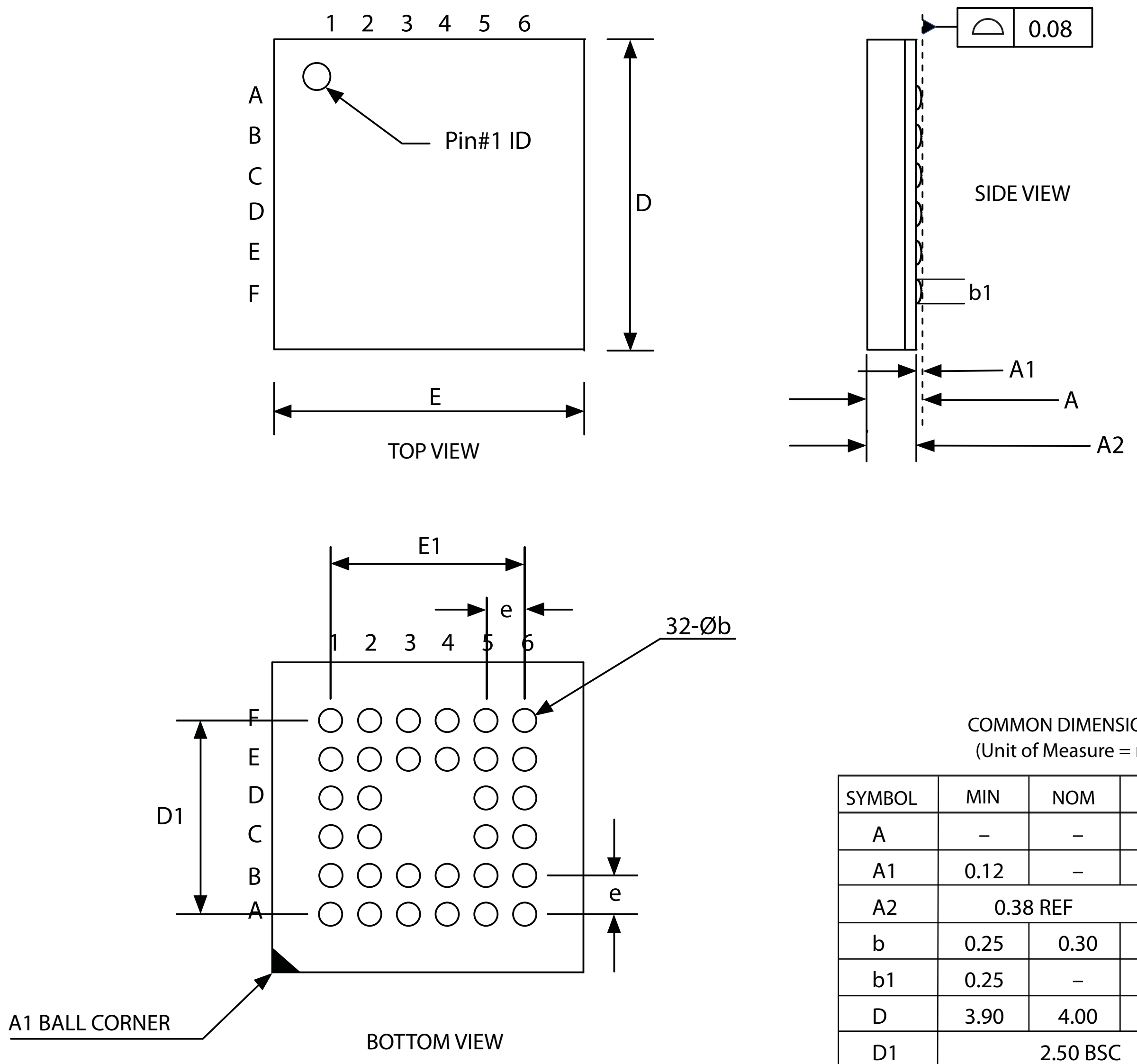
SYMBOL	MIN	NOM	MAX	NOTE
A	–	–	1.20	
A1	0.05	–	0.15	
A2	0.95	1.00	1.05	
D	8.75	9.00	9.25	
D1	6.90	7.00	7.10	Note 2
E	8.75	9.00	9.25	
E1	6.90	7.00	7.10	Note 2
B	0.30	–	0.45	
C	0.09	–	0.20	
L	0.45	–	0.75	
e	0.80 TYP			

Notes:
1. This package conforms to JEDEC reference MS-026, Variation ABA.
2. Dimensions D1 and E1 do not include mold protrusion. Allowable protrusion is 0.25mm per side. Dimensions D1 and E1 are maximum plastic body size dimensions including mold mismatch.
3. Lead coplanarity is 0.10mm maximum.

2010-10-20

	TITLE	DRAWING NO.	REV.
Atmel	32A, 32-lead, 7 x 7mm body size, 1.0mm body thickness, 0.8mm lead pitch, thin profile plastic quad flat package (TQFP)	32A	C

10.2 32CC1



COMMON DIMENSIONS
(Unit of Measure = mm)

SYMBOL	MIN	NOM	MAX	NOTE
A	–	–	0.60	
A1	0.12	–	–	
A2	0.38 REF			
b	0.25	0.30	0.35	1
b1	0.25	–	–	2
D	3.90	4.00	4.10	
D1	2.50 BSC			
E3.90	4.00	4.10		
E1	2.50 BSC			
e	0.50 BSC			

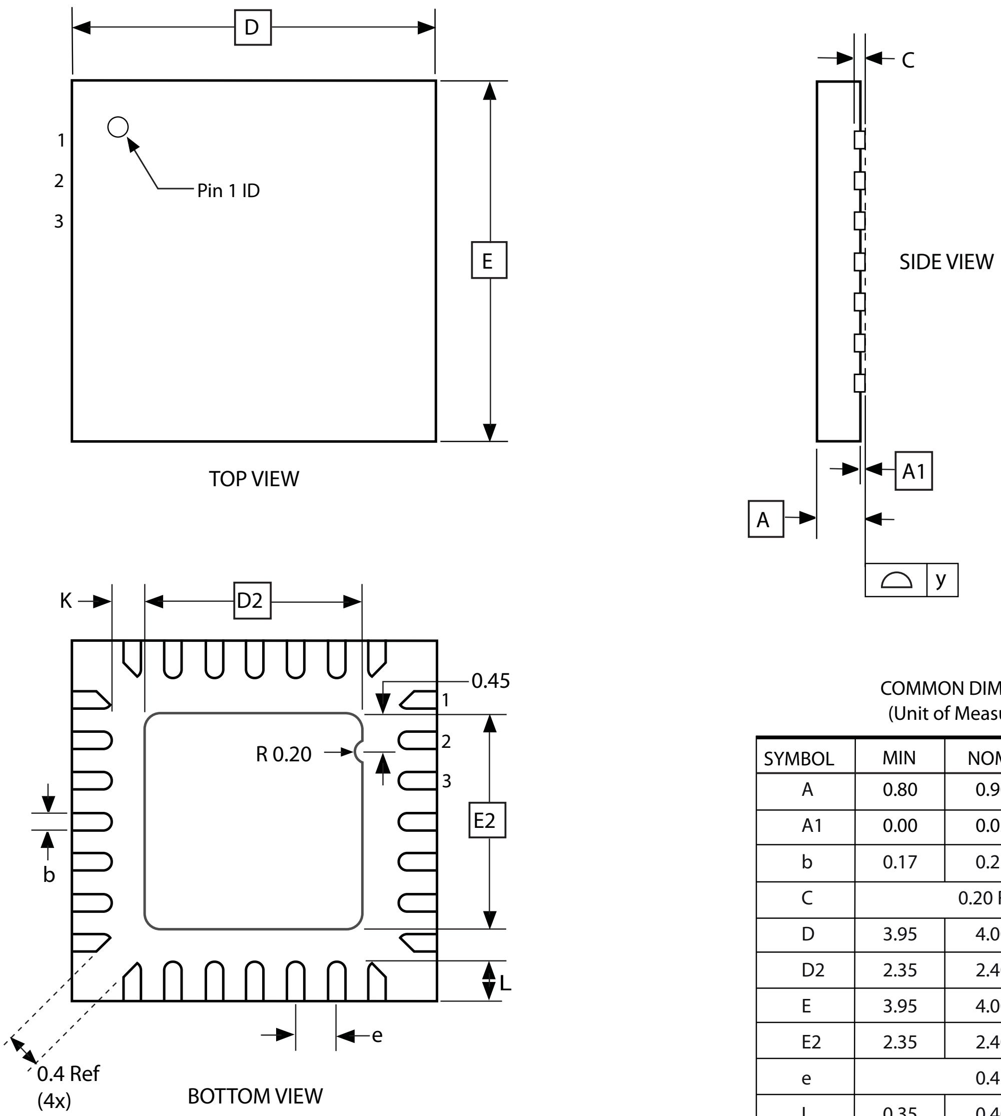
Note1: Dimension "b" is measured at the maximum ball dia. in a plane parallel to the seating plane.

Note2: Dimension "b1" is the solderable surface defined by the opening of the solder resist layer.

07/06/10

	TITLE	GPC	DRAWING NO.	REV.
Atmel Package Drawing Contact: packagedrawings@atmel.com	32CC1, 32-ball (6 x 6 Array), 4 x 4 x 0.6 mm package, ball pitch 0.50 mm, Ultra Thin, Fine-Pitch Ball Grid Array (UFBGA)	CAG	32CC1	B

10.3 28M1



COMMON DIMENSIONS
(Unit of Measure = mm)

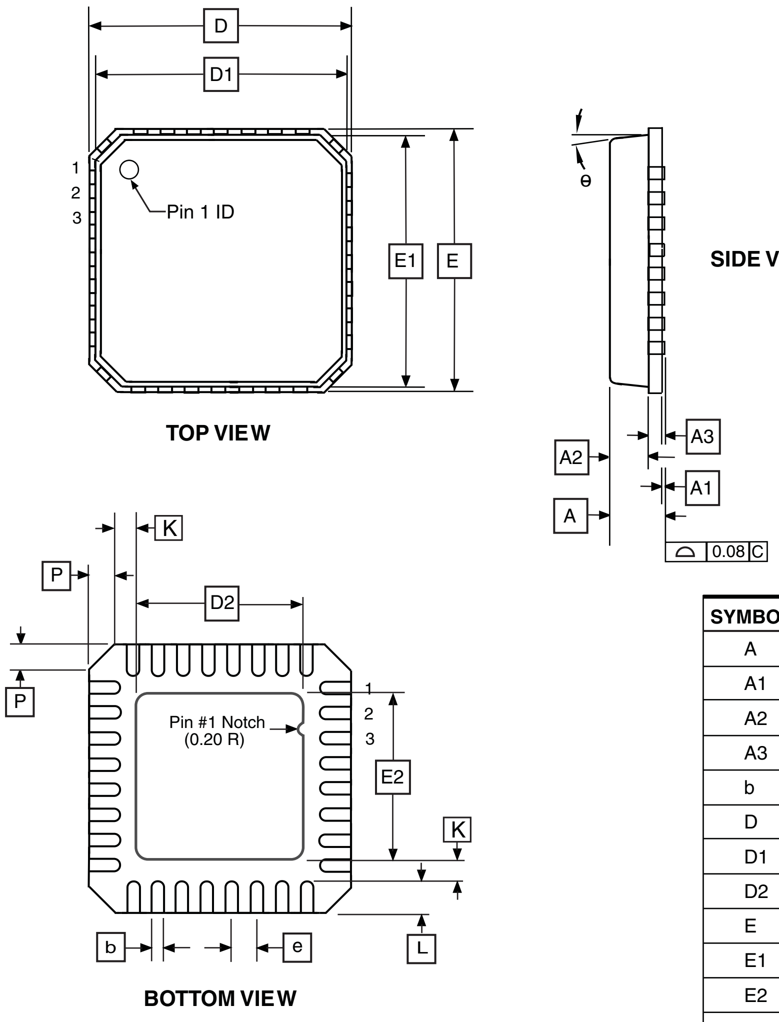
SYMBOL	MIN	NOM	MAX	NOTE
A	0.80	0.90	1.00	
A1	0.00	0.02	0.05	
b	0.17	0.22	0.27	
C		0.20 REF		
D	3.95	4.00	4.05	
D2	2.35	2.40	2.45	
E	3.95	4.00	4.05	
E2	2.35	2.40	2.45	
e		0.45		
L	0.35	0.40	0.45	
y	0.00	–	0.08	
K	0.20	–	–	

Note: The terminal #1 ID is a Laser-marked Feature.

10/24/08

	TITLE	GPC	DRAWING NO.	REV.
Atmel Package Drawing Contact: packagedrawings@atmel.com	28M1, 28-pad, 4 x 4 x 1.0mm Body, Lead Pitch 0.45mm, 2.4 x 2.4mm Exposed Pad, Thermally Enhanced Plastic Very Thin Quad Flat No Lead Package (VQFN)	ZBV	28M1	B

10.4 32M1-A



COMMON DIMENSIONS
(Unit of Measure = mm)

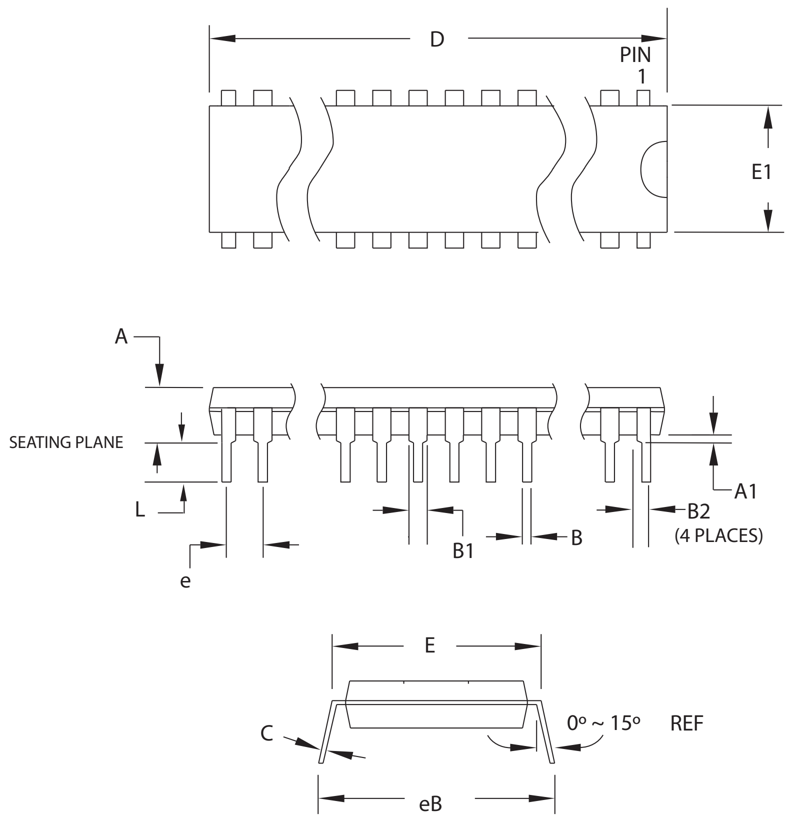
SYMBOL	MIN	NOM	MAX	NOTE
A	0.80	0.90	1.00	
A1	–	0.02	0.05	
A2	–	0.65	1.00	
A3		0.20 REF		
b	0.18	0.23	0.30	
D	4.90	5.00	5.10	
D1	4.70	4.75	4.80	
D2	2.95	3.10	3.25	
E	4.90	5.00	5.10	
E1	4.70	4.75	4.80	
E2	2.95	3.10	3.25	
e		0.50 BSC		
L	0.30	0.40	0.50	
P	–	–	0.60	
θ	–	–	12^{o}	
K	0.20	–	–	

Note: JEDEC Standard MO-220, Fig. 2 (Anvil Singulation), VHHD-2.

03/14/2014

	TITLE	DRAWING NO.	REV.
Atmel Package Drawing Contact: packagedrawings@atmel.com	**32M1-A** , 32-pad, 5 x 5 x 1.0mm Body, Lead Pitch 0.50mm, 3.10mm Exposed Pad, Micro Lead Frame Package (MLF)	32M1-A	F

10.5 28P3



Note: 1. Dimensions D and E1 do not include mold Flash or Protrusion. Mold Flash or Protrusion shall not exceed 0.25mm (0.010").

COMMON DIMENSIONS
(Unit of Measure = mm)

SYMBOL	MIN	NOM	MAX	NOTE
A	–	–	4.5724	
A1	0.508	–	–	
D	34.544	–	34.798	Note 1
E	7.620	–	8.255	
E1	7.112	–	7.493	Note 1
B	0.381	–	0.533	
B1	1.143	–	1.397	
B2	0.762	–	1.143	
L	3.175	–	3.429	
C	0.203	–	0.356	
eB	–	–	10.160	
e	2.540 TYP			

09/28/01

Atmel 2325 Orchard Parkway San Jose, CA 95131	TITLE 28P3, 28-lead (0.300"/7.62mm Wide) Plastic Dual Inline Package (PDIP)	DRAWING NO. 28P3	REV. B

11. Errata

11.1 Errata ATmega48A

The revision letter in this section refers to the revision of the ATmega48A device.

11.1.1 Rev. D

- **Analog MUX can be turned off when setting ACME bit**
- **TWI Data setup time can be too short**

1. **Analog MUX can be turned off when setting ACME bit**

   If the ACME (Analog Comparator Multiplexer Enabled) bit in ADCSRB is set while MUX3 in ADMUX is '1' (ADMUX[3:0]=1xxx), all MUXes are turned off until the ACME bit is cleared.

   Problem Fix/Workaround

   Clear the MUX3 bit before setting the ACME bit.

2. **TWI Data setup time can be too short**

   When running the device as a TWI slave with a system clock above 2MHz, the data setup time for the first bit after ACK may in some cases be too short. This may cause a false start or stop condition on the TWI line.

   Problem Fix/Workaround

   Insert a delay between setting TWDR and TWCR.

11.2 Errata ATmega48PA

The revision letter in this section refers to the revision of the ATmega48PA device.

11.2.1 Rev. A

- **Power consumption in power save modes**
- **Startup time for the device**

1. **Power consumption in power save modes**

   Power consumption in power save modes will be higher due to improper control of internal power management.48

   Problem Fix/Workaround

   This problem will be corrected in Rev B.

2. **Startup time for the device**

   Due to implementation of a different NVM structure, the startup sequence for the device will require longer startup time.

   Problem Fix/Workaround

   There is no fix for this problem.

11.2.2 Rev. D

- **Analog MUX can be turned off when setting ACME bit**
- **TWI Data setup time can be too short**

1. **Analog MUX can be turned off when setting ACME bit**

If the ACME (Analog Comparator Multiplexer Enabled) bit in ADCSRB is set while MUX3 in ADMUX is '1' (ADMUX[3:0]=1xxx), all MU Xes are turned off until the ACME bit is cleared.

Problem Fix/Workaround

Clear the MUX3 bit before setting the ACME bit.

2. **TWI Data setup time can be too short**

   When running the device as a TWI slave with a system clock above 2MHz, the data setup time for the first bit after ACK may in some cases be too short. This may cause a false start or stop condition on the TWI line.

   Problem Fix/Workaround

   Insert a delay between setting TWDR and TWCR.

11.3 Errata ATmega88A

The revision letter in this section refers to the revision of the ATmega88A device.

11.3.1 Rev. F

- **Analog MUX can be turned off when setting ACME bit**
- **TWI Data setup time can be too short**

1. **Analog MUX can be turned off when setting ACME bit**

   If the ACME (Analog Comparator Multiplexer Enabled) bit in ADCSRB is set while MUX3 in ADMUX is '1' (ADMUX[3:0]=1xxx), all MU Xes are turned off until the ACME bit is cleared.

   Problem Fix/Workaround

   Clear the MUX3 bit before setting the ACME bit.

2. **TWI Data setup time can be too short**

   When running the device as a TWI slave with a system clock above 2MHz, the data setup time for the first bit after ACK may in some cases be too short. This may cause a false start or stop condition on the TWI line.

   Problem Fix/Workaround

   Insert a delay between setting TWDR and TWCR.

11.4 Errata ATmega88PA

The revision letter in this section refers to the revision of the ATmega88PA device.

11.4.1 Rev. F

- **Analog MUX can be turned off when setting ACME bit**
- **TWI Data setup time can be too short**

1. **Analog MUX can be turned off when setting ACME bit**

   If the ACME (Analog Comparator Multiplexer Enabled) bit in ADCSRB is set while MUX3 in ADMUX is '1' (ADMUX[3:0]=1xxx), all MUXes are turned off until the ACME bit is cleared.

   Problem Fix/Workaround

   Clear the MUX3 bit before setting the ACME bit.

2. **TWI Data setup time can be too short**

When running the device as a TWI slave with a system clock above 2MHz, the data setup time for the first bit after ACK may in some cases be too short. This may cause a false start or stop condition on the TWI line.

Problem Fix/Workaround

Insert a delay between setting TWDR and TWCR.

11.4.2 Rev. A

- **Power consumption in power save modes**
- **Startup time for the device**

1. **Power consumption in power save modes**

   Power consumption in power save modes will be higher due to improper control of internal power management.48

   Problem Fix/Workaround

   This problem will be corrected in Rev B.

2. **Startup time for the device**

   Due to implementation of a different NVM structure, the startup sequence for the device will require longer startup time.

   Problem Fix/Workaround

   There is no fix for this problem.

11.5 Errata ATmega168A

The revision letter in this section refers to the revision of the ATmega168A device.

11.5.1 Rev. E

- **Analog MUX can be turned off when setting ACME bit**
- **TWI Data setup time can be too short**

1. **Analog MUX can be turned off when setting ACME bit**

   If the ACME (Analog Comparator Multiplexer Enabled) bit in ADCSRB is set while MUX3 in ADMUX is '1' (ADMUX[3:0]=1xxx), all MUXes are turned off until the ACME bit is cleared.

   Problem Fix/Workaround

   Clear the MUX3 bit before setting the ACME bit.

2. **TWI Data setup time can be too short**

   When running the device as a TWI slave with a system clock above 2MHz, the data setup time for the first bit after ACK may in some cases be too short. This may cause a false start or stop condition on the TWI line.

   Problem Fix/Workaround

   Insert a delay between setting TWDR and TWCR.

11.6 Errata ATmega168PA

The revision letter in this section refers to the revision of the ATmega168PA device.

11.6.1 Rev E

- **Analog MUX can be turned off when setting ACME bit**
- **TWI Data setup time can be too short**

1. **Analog MUX can be turned off when setting ACME bit**

   If the ACME (Analog Comparator Multiplexer Enabled) bit in ADCSRB is set while MUX3 in ADMUX is '1' (ADMUX[3:0]=1xxx), all MUXes are turned off until the ACME bit is cleared.

   Problem Fix/Workaround

   Clear the MUX3 bit before setting the ACME bit.

2. **TWI Data setup time can be too short**

   When running the device as a TWI slave with a system clock above 2MHz, the data setup time for the first bit after ACK may in some cases be too short. This may cause a false start or stop condition on the TWI line.

   Problem Fix/Workaround

   Insert a delay between setting TWDR and TWCR.

11.7 Errata ATmega328

The revision letter in this section refers to the revision of the ATmega328 device.

11.7.1 Rev D

- **Analog MUX can be turned off when setting ACME bit**
- **TWI Data setup time can be too short**

1. Analog MUX can be turned off when setting ACME bit

If the ACME (Analog Comparator Multiplexer Enabled) bit in ADCSRB is set while MUX3 in ADMUX is '1' (ADMUX[3:0]=1xxx), all MUX es are turned off until the ACME bit is cleared.

Problem Fix/Workaround

Clear the MUX3 bit before setting the ACME bit.

2. TWI Data setup time can be too short

When running the device as a TWI slave with a system clock above 2MHz, the data setup time for the first bit after ACK may in some cases be too short. This may cause a false start or stop condition on the TWI line.

Problem Fix/Workaround

Insert a delay between setting TWDR and TWCR.

11.7.2 Rev C

Not sampled.

11.7.3 Rev B

- **Analog MUX can be turned off when setting ACME bit**
- **Unstable 32kHz Oscillator**

1. Analog MUX can be turned off when setting ACME bit

If the ACME (Analog Comparator Multiplexer Enabled) bit in ADCSRB is set while MUX3 in ADMUX is '1' (ADMUX[3:0]=1xxx), all MUXes are turned off until the ACME bit is cleared.

Problem Fix/Workaround

Clear the MUX3 bit before setting the ACME bit.

2. Unstable 32kHz Oscillator

The 32kHz oscillator does not work as system clock. The 32kHz oscillator used as asynchronous timer is inaccurate.

Problem Fix/ Workaround

None.

11.7.4 Rev A

- **Analog MUX can be turned off when setting ACME bit**
- **Unstable 32kHz Oscillator**

1. Analog MUX can be turned off when setting ACME bit

If the ACME (Analog Comparator Multiplexer Enabled) bit in ADCSRB is set while MUX3 in ADMUX is '1' (ADMUX[3:0]=1xxx), all MUXes are turned off until the ACME bit is cleared.

Problem Fix/Workaround

Clear the MUX3 bit before setting the ACME bit.

2. **Unstable 32kHz Oscillator**

The 32kHz oscillator does not work as system clock. The 32kHz oscillator used as asynchronous timer is inaccurate.

Problem Fix/ Workaround

None.

11.8 Errata ATmega328P

The revision letter in this section refers to the revision of the ATmega328P device.

11.8.1 Rev D

- **Analog MUX can be turned off when setting ACME bit**
- **TWI Data setup time can be too short**

1. **Analog MUX can be turned off when setting ACME bit**

If the ACME (Analog Comparator Multiplexer Enabled) bit in ADCSRB is set while MUX3 in ADMUX is '1' (ADMUX[3:0]=1xxx), all MUXes are turned off until the ACME bit is cleared.

Problem Fix/Workaround

Clear the MUX3 bit before setting the ACME bit.

2. **TWI Data setup time can be too short**

When running the device as a TWI slave with a system clock above 2MHz, the data setup time for the first bit after ACK may in some cases be too short. This may cause a false start or stop condition on the TWI line.

Problem Fix/Workaround

Insert a delay between setting TWDR and TWCR.

11.8.2 Rev C

Not sampled.

11.8.3 Rev B

- **Analog MUX can be turned off when setting ACME bit**
- **Unstable 32kHz Oscillator**

1. **Analog MUX can be turned off when setting ACME bit**

If the ACME (Analog Comparator Multiplexer Enabled) bit in ADCSRB is set while MUX3 in ADMUX is '1' (ADMUX[3:0]=1xxx), all MUXes are turned off until the ACME bit is cleared.

Problem Fix/Workaround

Clear the MUX3 bit before setting the ACME bit.

2. **Unstable 32kHz Oscillator**

The 32kHz oscillator does not work as system clock. The 32kHz oscillator used as asynchronous timer is inaccurate.

Problem Fix/ Workaround

None.

11.8.4 Rev A

- **Unstable 32kHz Oscillator**

1. Unstable 32kHz Oscillator

The 32kHz oscillator does not work as system clock. The 32kHz oscillator used as asynchronous timer is inaccurate.

Problem Fix/ Workaround

None.

12. Datasheet Revision History

Please note that the referring page numbers in this section are referred to this document. The referring revision in this section are referring to the document revision.

12.1 Rev. 8271H – 08/2013

1. Updated text in section Section 16.9.3 ”Fast PWM Mode” on page 124 concerning compare units allowing generation of PWM waveforms (on page 126), referring to table 16-2.
2. Updated WDT Assembly code example in Section 10.10.5 ”Watchdog Timer” on page 44 (and onwards)
3. Updated footnote 1 for tables giving DC Characteristics in ”ATmega48PA DC Characteristics – Current Consumption” on page 320, ”ATmega88PA DC Characteristics – Current Consumption” on page 321, ”ATmega168P DC Characteristics – Current Consumption” on page 321 and ”ATmega328P DC Characteristics – Current Consumption” on page 322.
4. Figure 31-1 on page 324 has been updated with the correct plot.
5. Figure 31-333 on page 499 has been updated with the correct plot.
6. Changed description of external interrupt behavior in deep sleep in Section 13. ”External Interrupts” on page 71.
7. Added wait delay for t_{WD_FUSE} in Table 28-18 on page 298.
7. Updated errata for rev A of 48PA and 88PA in Section 11.2 on page 29 and Section 11.4 on page 30.
8. Updated back page and footer according to datasheet template of 05/2014

12.2 Rev. 8271G – 02/2013

1. Added ”Electrical Characteristics (TA = -40°C to 105°C)” on page 319.
2. Added ”ATmega48PA Typical Characteristics – (TA = -40°C to 105°C)” on page 523.
3. Added ”ATmega88PA Typical Characteristics – (TA = -40°C to 105°C)” on page 547.
4. Added ”ATmega168PA Typical Characteristics – (TA = -40°C to 105°C)” on page 571.
5. Added ”ATmega328P Typical Characteristics – (TA = -40°C to 105°C)” on page 596.

12.3 Rev. 8271F – 08/2012

1. Added ”DC Characteristics” on page 301. The following tables for DC characteristics - T_A = -40°C to 105°C added:
   Table 29-4 on page 304
   Table 29-7 on page 305
   Table 29-10 on page 307
   Table 29-13 on page 308
2. Replaced the following typical characteristics by the plots that include les characteristics at “T_A = -40°C to 105°C”:
   ”ATmega48PA Typical Characteristics” on page 349
   ”ATmega88PA Typical Characteristics” on page 398
   ”ATmega168PA Typical Characteristics” on page 448
   ”ATmega328P Typical Characteristics” on page 498

3. Removed the Power Save (Psave) maximum numbers for all devices throughout ”Electrical Characteristics – (TA = -40°C to 85°C)” on page 301.
4. Changed the powerdown maximum numbers from 8.5 and 3µA to 10 and 5µA (ATmega48PA, ATmega88PA, ATmega168PA and ATmega328P).
5. Changed the table note “Maximum values are characterized values and not test limits in production” to “Max values are test limits in production throughout ”Electrical Characteristics – (TA = -40°C to 85°C)” on page 301.

12.4 Rev. 8271E – 07/2012

1. Updated Figure 1-1 on page 3. Overlined “RESET” in 28 MLF top view and in 32 MLF top view.
2. Added EEAR9 bit to the ”EEARH and EEARL – The EEPROM Address Register” on page 22 and updated the all bit descriptions accordingly.
3. Added a footnote “EEAR9 and EEAR8 are unused bits in ATmega48A/48PA and must always be written to zero” to ”EEARH and EEARL – The EEPROM Address Register” on page 22.
4. Updated Table 18-8 on page 156, “Waveform Generation Mode Bit Description” . WGM2, WGM1 and WGM0 changed to WGM22, WGM21 and WGM20 respectively.
5. Updated ”TCCR2B – Timer/Counter Control Register B” on page 157. bit 2 (CS22) and bit 3 (WGM22) changed from R (read only) to R/W (read/write).
6. Updated the definition of **fosc** on page 173. **fosc** is the system clock frequency (not XTAL pin frequency)
7. Updated ”SPMCSR – Store Program Memory Control and Status Register” on page 263. Bit 0 renamed SPMEN and added bit 5 “SIGRD”.
8. Replaced “SELFPRGEN” by “SPMEN” throughout the whole datasheet including in the “code examples”, except in ”Program And Data Memory Lock Bits” on page 282 and in ”Fuse Bits” on page 283.
9. Updated ”Register Summary” on page 9 to include the bits: SIGRD and SPMEN in the SMPCSR register.
10. Updated the Table 29-1 on page 301. Removed the footnote.
11. Updated the footnote of the Table 29-18 on page 312. Removed the footnote “Note 2”.
12. Updated ”Errata” on page 29. Added “Errata” TWI Data setup time can be too short.

12.5 Rev. 8271D – 05/11

1. Added Atmel QTouch Sensing Capability Feature
2. Updated ”Register Description” on page 92 with PINxn as R/W.
3. Added a footnote to the PINxn, page 92.
4. Updated “Ordering Information”,”ATmega328” on page 22. Added “ATmega328-MMH” and “ATmega328-MMHR”.
5. Updated “Ordering Information”,”ATmega328P” on page 23. Added “ATmega328P-MMH” and “ATmega328P-MMHR”.
6. Added “Ordering Information” for ATmega48PA/88PA/168PA/328P @ 105°C
7. Updated ”Errata ATmega328” on page 33 and ”Errata ATmega328P” on page 34
8. Updated the datasheet according to the Atmel new brand style guide.

12.6 Rev. 8271C – 08/10

1. Added 32UFBGA Pinout, Table 1-1 on page 3.
2. Updated the “SRAM Data Memory”, Figure 8-3 on page 19.
3. Updated ”Ordering Information” on page 16 with CCU and CCUR code related to “32CC1” Package drawing.
4. “32CC1” Package drawing added ”Packaging Information” on page 24.

12.7 Rev. 8271B – 04/10

1. Updated Table 9-8 with correct value for timer oscillator at xtal2/tos2
2. Corrected use of SBIS instructions in assembly code examples.
3. Corrected BOD and BODSE bits to R/W in Section 10.11.2 on page 46, Section 12.5 on page 69 and Section 14.4 on page 92
4. Figures for bandgap characterization added, Figure 31-34 on page 341, Figure 31-81 on page 366, Figure 31-128 on page 391, Figure 31-176 on page 417, Figure 31-223 on page 441, Figure 31-271 on page 467, Figure 31-318 on page 491 and Figure 31-365 on page 516.
5. Updated ”Packaging Information” on page 24 by replacing 28M1 with a correct corresponding package.

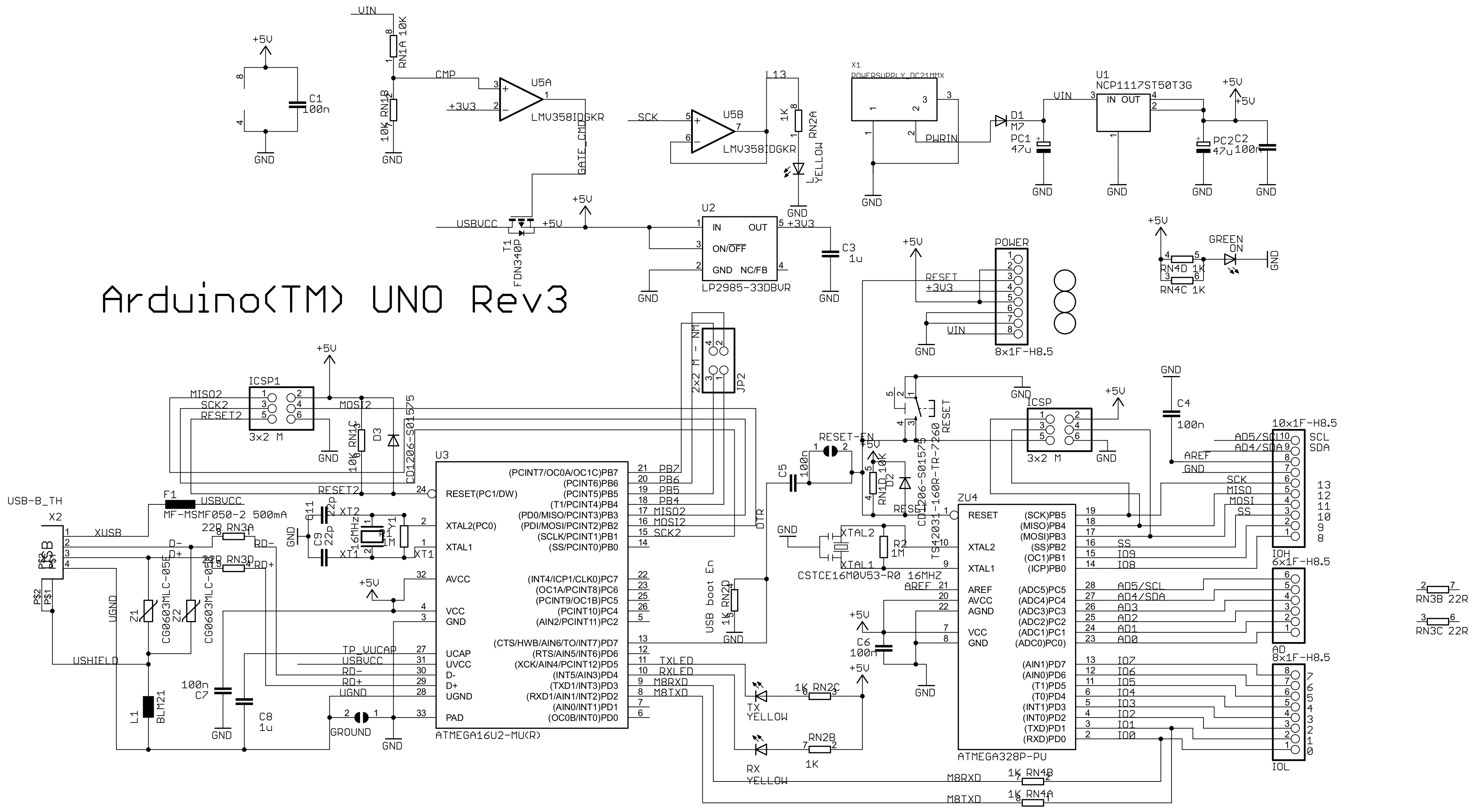
12.8 Rev. 8271A – 12/09

1. New datasheet 8271 with merged information for ATmega48PA, ATmega88PA, ATmega168PA and ATmega48A, ATmega88A andATmega168A. Also included information on ATmega328 and ATmega328P

2 Changes done:

– New devices added: ATmega48A/ATmega88A/ATmega168A and ATmega328
– Updated Feature Description
– Updated Table 2-1 on page 7
– Added note for BOD Disable on page 41.
– Added note on BOD and BODSE in ”MCUCR – MCU Control Register” on page 92 and ”Register Description” on page 280
– Added limitation information for the application ”Boot Loader Support – Read-While-Write Self-Programming” on page 265
– Added limitation information for ”Program And Data Memory Lock Bits” on page 282
– Added specified DC characteristics
– Added typical characteristics
– Removed exception information in ”Address Match Unit” on page 214.
–
–

Enabling Unlimited Possibilities®



Reference Designs ARE PROVIDED "AS IS" AND "WITH ALL FAULTS. Arduino DISCLAIMS ALL OTHER WARRANTIES, EXPRESS OR IMPLIED, REGARDING PRODUCTS, INCLUDING BUT NOT LIMITED TO, ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE Arduino may make changes to specifications and product descriptions at any time, without notice. The Customer must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined." Arduino reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The product information on the Web Site or Materials is subject to change without notice. Do not finalize a design with this information.

ARDUINO is a registered trademark.

Use of the ARDUINO name must be compliant with http://www.arduino.cc/en/Main/Policy

The Multi-Rotor Controller

Table of Contents

Introduction to the Multi-Rotor controller

The Multi-Rotor controller is a flight control board for 4 rotor Aircraft (Multi-Rotors). Its purpose is to stabilise the aircraft during flight. To do this it takes the signal from the three gyros on the board (roll, pitch and yaw) and feeds the information into the Integrated Circuit (Atmega IC). This then processes the information according the software and sends out a control signal to the Electronic Speed Controllers (ESCs) which are plugged onto the board and also connected to the motors. Depending upon the signal from the IC the ESCs will either speed up or slow down the motors in order to establish level flight.

The board also takes a control signal from the Remote Control Receiver (RX) and feeds this into the IC via the ail, ele, thr and rud pins on the board. After processing this information, the IC will then send out a signal to the motors (Via the M1 to M4 pins on the board) to speed up or slow down to achieve controlled flight (up, down, backwards, forwards, left, right, yaw) on the command from the RC Pilot sent via his Transmitter (TX).

Flight Configurations

The Multi-Rotor flight configurations depend on which firmware is loaded onto the chip.

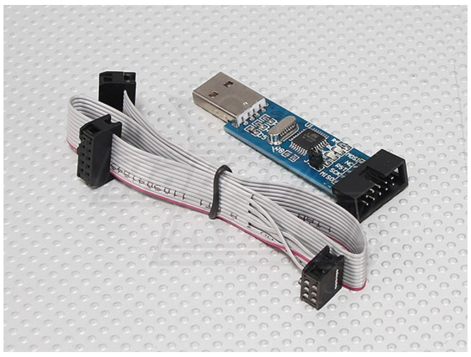
This configuration is Multi-Rotor (4 Rotor + configuration).

Updating the Firmware

The Multi-Rotor board has an Atmega328P chip on board which allows users to tweak and load non standard firmware.

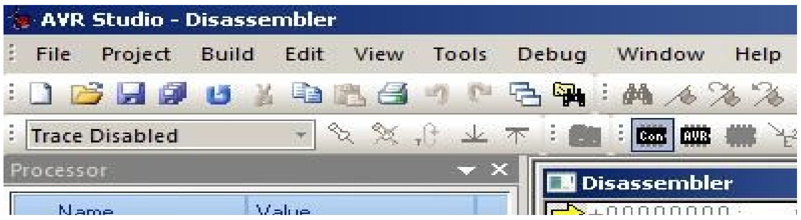
Set IC Fuses & Flashing the Firmware

Connect a USBasp Programmer to the six pin ISP header on the
Multi-Rotor controller board.

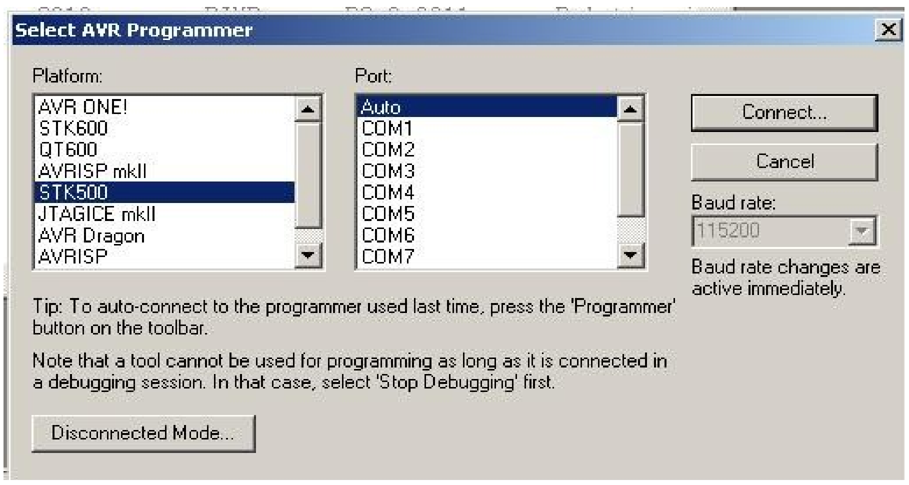


Connect your Programmer's 6 pin socket to the ISP header on the board. Pin 1 on the ISP header is usually marked with a small triangle. Then connect the a 5V DC power source to the PCB pins.

Open AVR Studio 4. It will ask you if you want to begin a new project, or open an existing project. Choose Cancel and click on the connect icon.



It will open a new window with a connection dialogue asking you to select your programmer and connection port. With a programmer like the AVRISP mkII it is easy because when you select that programmer it brings back only one choice of port... USB. The AVR-ISP500 from Olimex is recognised as a STK500 and has the option to auto choose the port. If it fails to recognise the port, you may need to manually set the port for the programmer in your Windows device settings to COM1 up to COM4 for AVR Studio to recognise it.



Baud rate: 115200

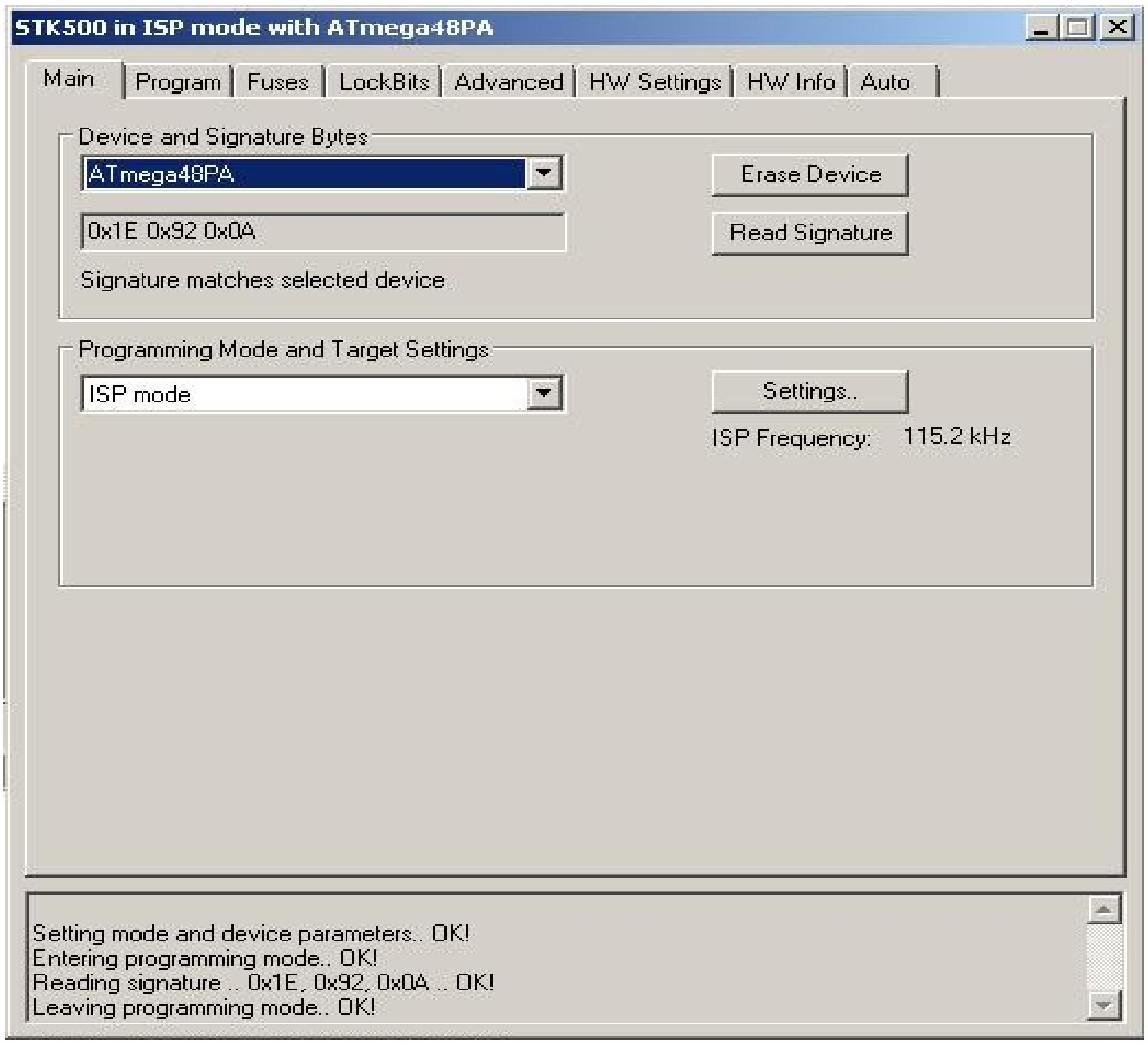
Baud rate changes are active immediately.

Tip: To auto-connect to the programmer used last time, press the 'Programmer' button on the toolbar.

Note that a tool cannot be used for programming as long as it is connected in a debugging session. In that case, select 'Stop Debugging' first.

When you have chosen your programmer and port, click connect and you will be taken to the AVR programming dialogue.

In the AVR programming window go to the "Main" tab and make sure that the chip you are programming (e.g. Atmega328P) is selected in the "Device and Signature Bytes" drop down menu. Also make sure that the" Programming mode and target settings are set to ISP. Make sure that the settings for the ISP mode have the ISP frequency set low enough to talk to the chip. Programmer's frequency can set to 115.2 kHz. This is quite an important setting to get right. If you click on "Read Signature" and you get the response "Signature matches selected device" you have successfully managed to connect to your IC.


STK500 in ISP mode with ATmega48PA

Device and Signature Bytes

ATmega48PA

0x1E 0x92 0x0A

Signature matches selected device

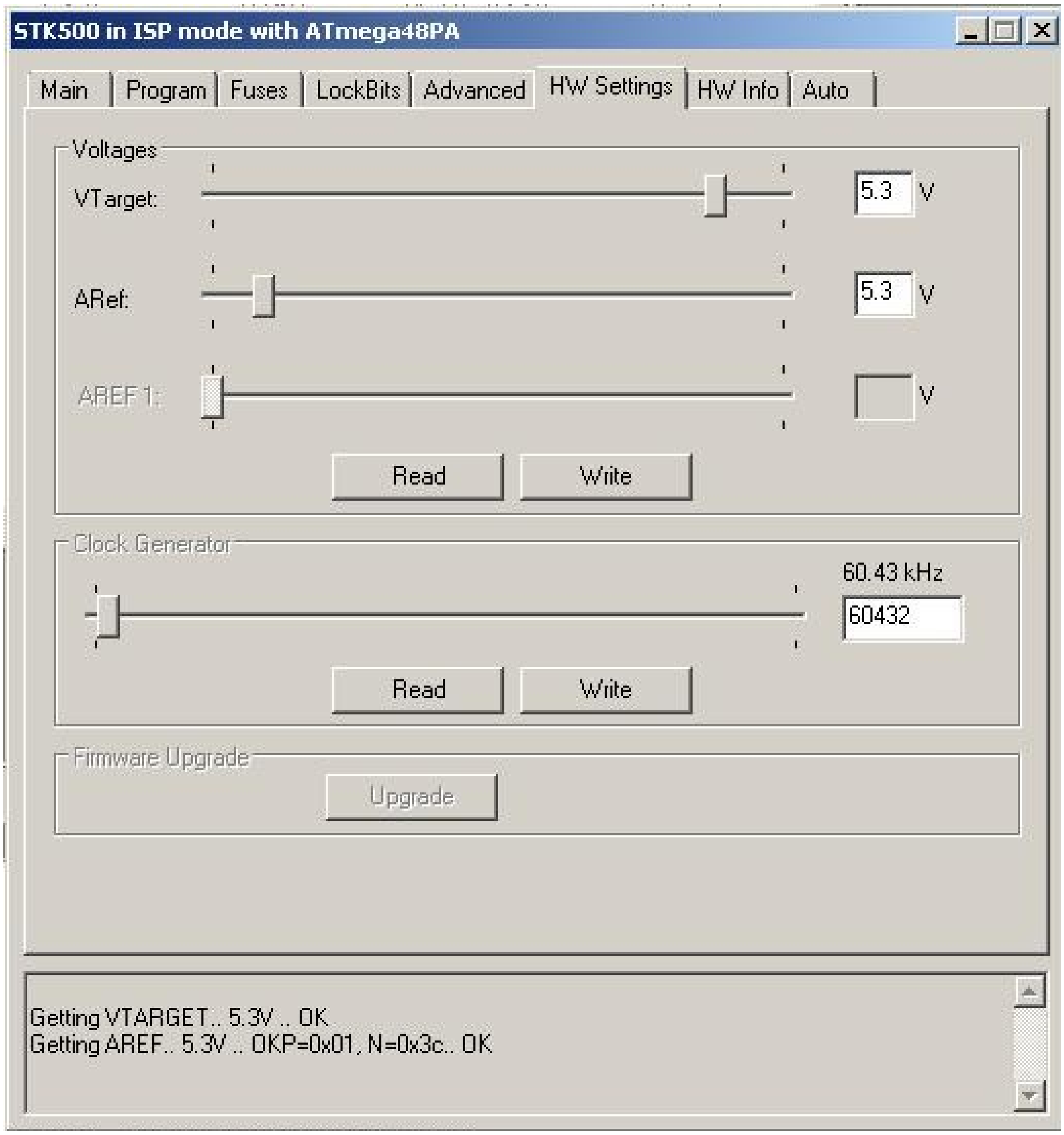
Programming Mode and Target Settings

ISP mode

ISP Frequency: 115.2 kHz

```
Setting mode and device parameters.. OK!
Entering programming mode.. OK!
Reading signature .. 0x1E, 0x92, 0x0A .. OK!
Leaving programming mode.. OK!
```

Also make sure that the target board or PCB is powered (You can check this by clicking on the HW Settings tab and checking if the programmer can see any voltage).



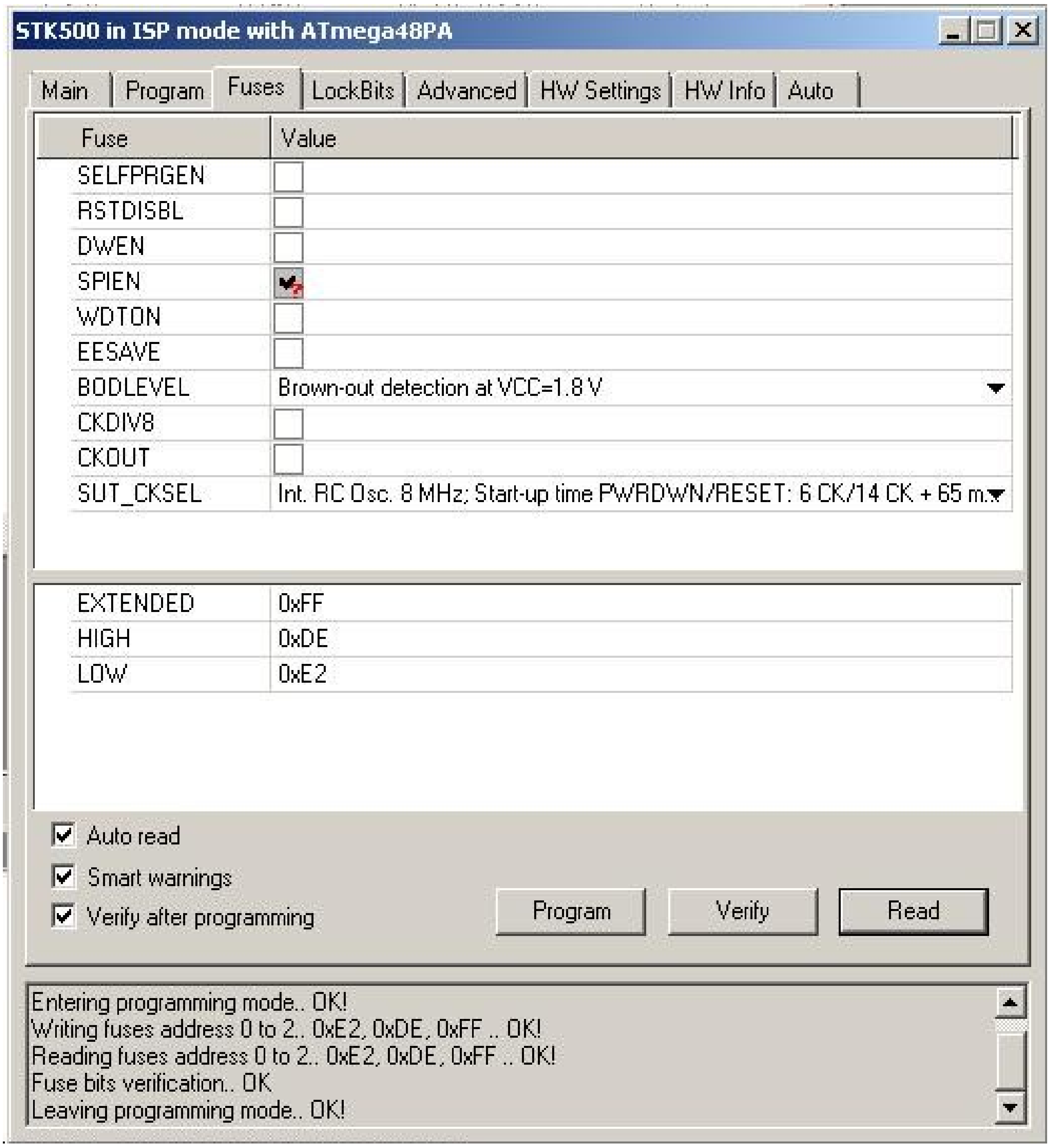
VTarget: 5.3 V
ARef: 5.3 V
AREF 1: V
60.43 kHz
60432

```
Getting VTARGET.. 5.3V .. OK
Getting AREF.. 5.3V .. OKP=0x01, N=0x3c.. OK
```

Now it is time to set the fuses so click on the "Fuses" tab. AVR Studio is very good in this respect as it will work out the fuse settings for your particular IC depending upon the check box options you choose.

Set the check boxes according to the following.

SELFPRGEN: unchecked
RSTDISBL: unchecked
DWEN: unchecked
SPIEN: **checked**
WDTON: unchecked
EESAVE: unchecked
BODLEVEL: **Brown-out detection at VCC=1.8 V**
CKDIV8 : unchecked
CKOUT: unchecked
SUT_CKSEL: **Int. RC Osc. 8 MHz; Start-up time PWRDWN/RESET: 6 CK/14 CK + 65 ms**
The fuse setting output will be displayed at the bottom of the window.



Fuse	Value
SELFPRGEN	☐
RSTDISBL	☐
DWEN	☐
SPIEN	✓
WDTON	☐
EESAVE	☐
BODLEVEL	Brown-out detection at VCC=1.8 V
CKDIV8	☐
CKOUT	☐
SUT_CKSEL	Int. RC Osc. 8 MHz; Start-up time PWRDWN/RESET: 6 CK/14 CK + 65 m

EXTENDED	0xFF
HIGH	0xDE
LOW	0xE2

```
Entering programming mode.. OK!
Writing fuses address 0 to 2.. 0xE2, 0xDE, 0xFF .. OK!
Reading fuses address 0 to 2.. 0xE2, 0xDE, 0xFF .. OK!
Fuse bits verification.. OK
Leaving programming mode.. OK!
```

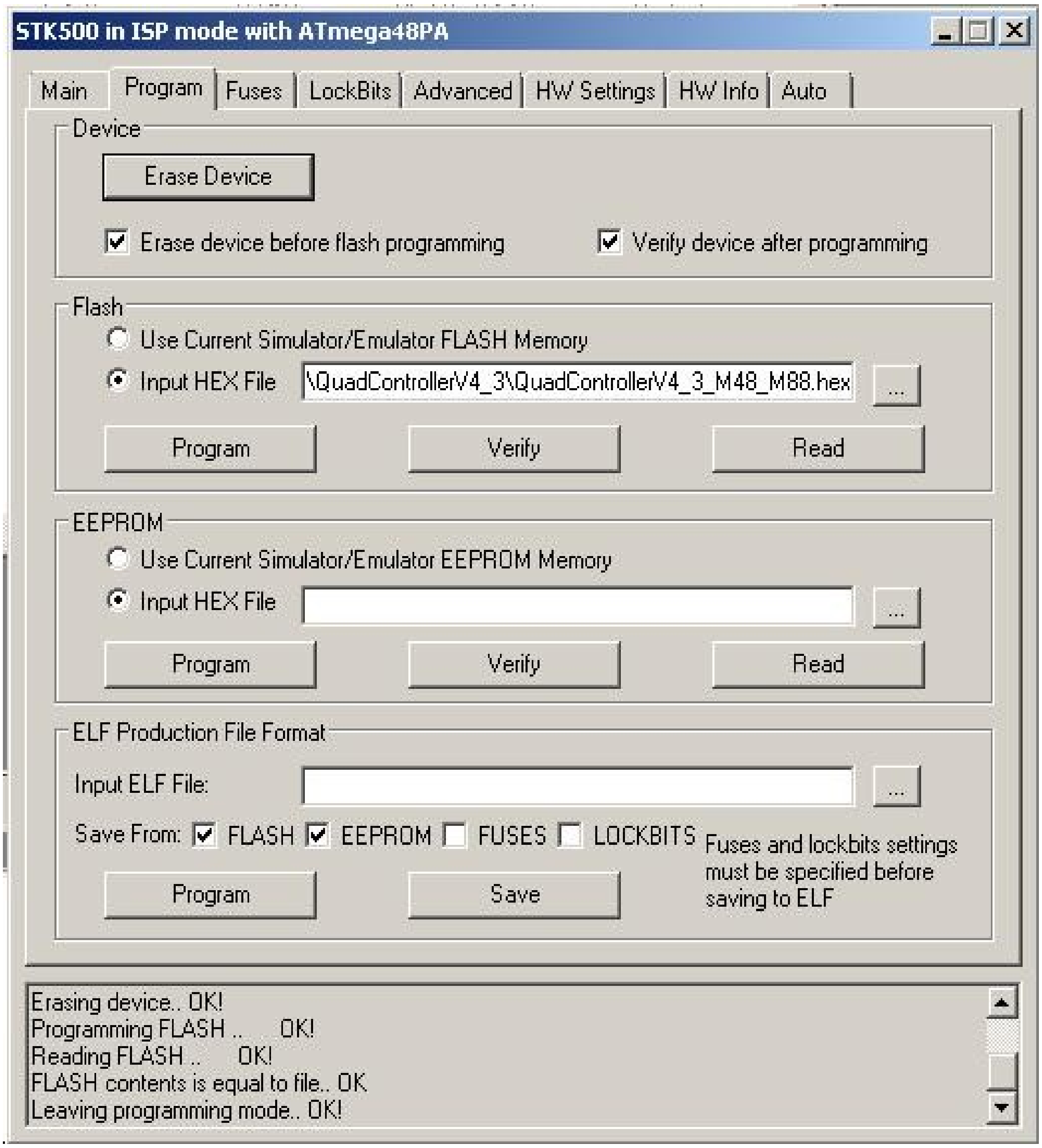
Check the "Auto read" "Smart warnings" and "Verify after programming" options at the bottom of the window and then click program.

If all goes to plan, you should get OK response messages in the output section at the bottom of your window for Entering Programming mode, writing fuse address, reading fuse address, Fuse bits verification and leaving programming mode.

If you get error messages, then recheck your chip version and all the connections from your programmer to the board and that the power is on. Also make sure that your fuse settings are as described above.

Flash the Firmware

Click on the "Program" tab and have a look at the "Flash" section which is the 2nd section down. Check the "Input HEX file" check box. Then browse the unzipped firmware folder and click on your firmware HEX file suitable for the chip you are programming for an Atmega328P. Then click "Program" in the Flash section of the window and you should get an OK response in the output section at the bottom of the window. Then click "Verify" to make sure that the program has been successfully uploaded and if you have an OK response coming back at the bottom of the window then you have successfully programmed the IC with the test program.


STK500 in ISP mode with ATmega48PA

Main | Program | Fuses | LockBits | Advanced | HW Settings | HW Info | Auto

Device

Erase Device

[✓] Erase device before flash programming [✓] Verify device after programming

Flash

○ Use Current Simulator/Emulator FLASH Memory

● Input HEX File \QuadControllerV4_3\QuadControllerV4_3_M48_M88.hex ...

Program Verify Read

EEPROM

○ Use Current Simulator/Emulator EEPROM Memory

● Input HEX File ...

Program Verify Read

ELF Production File Format

Input ELF File: ...

Save From: [✓] FLASH [✓] EEPROM [ ] FUSES [ ] LOCKBITS

Fuses and lockbits settings must be specified before saving to ELF

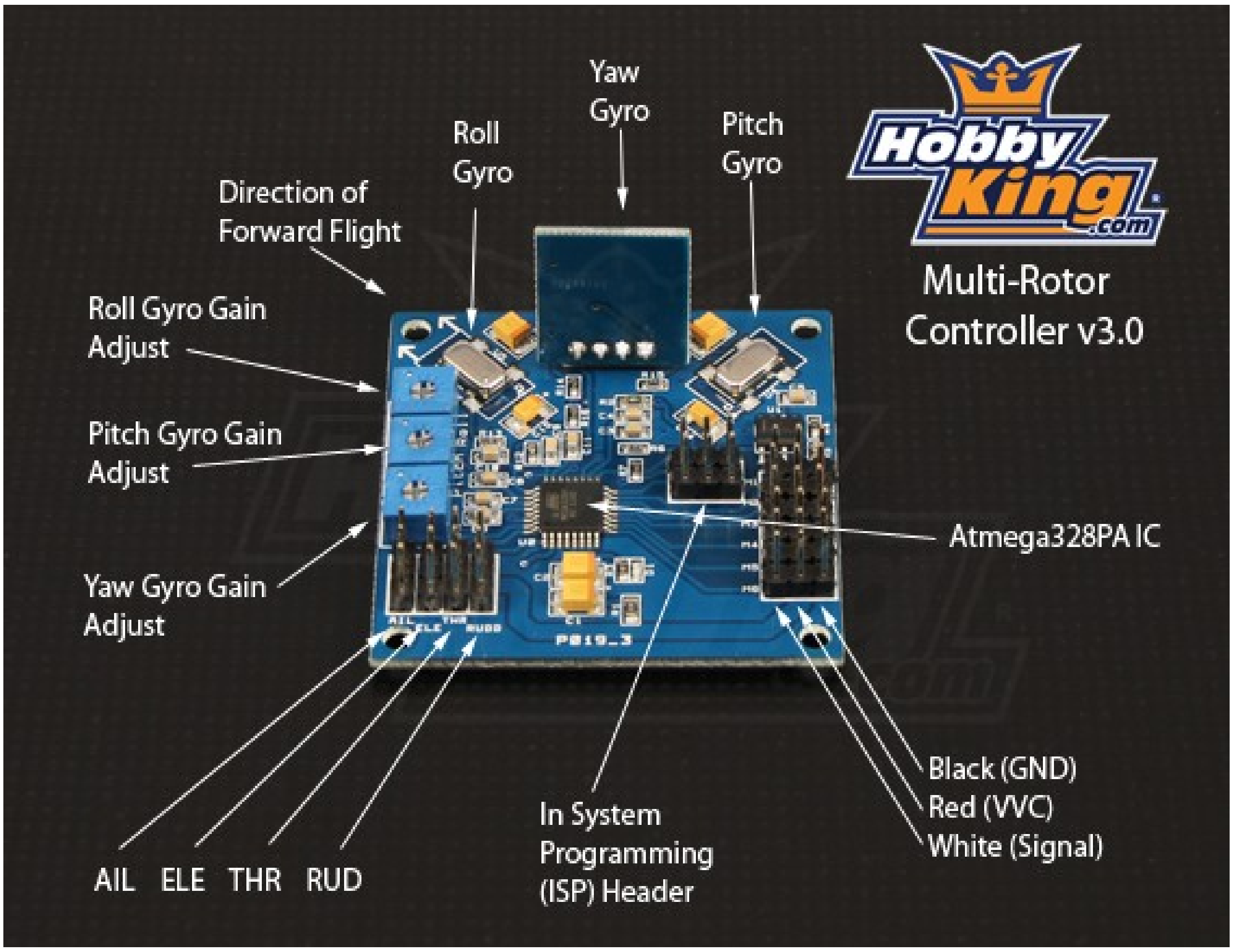
Program Save

```
Erasing device.. OK!
Programming FLASH ..      OK!
Reading FLASH ..      OK!
FLASH contents is equal to file.. OK
Leaving programming mode.. OK!
```

Mounting the Multi-Rotor controller in your Multi-Rotor.

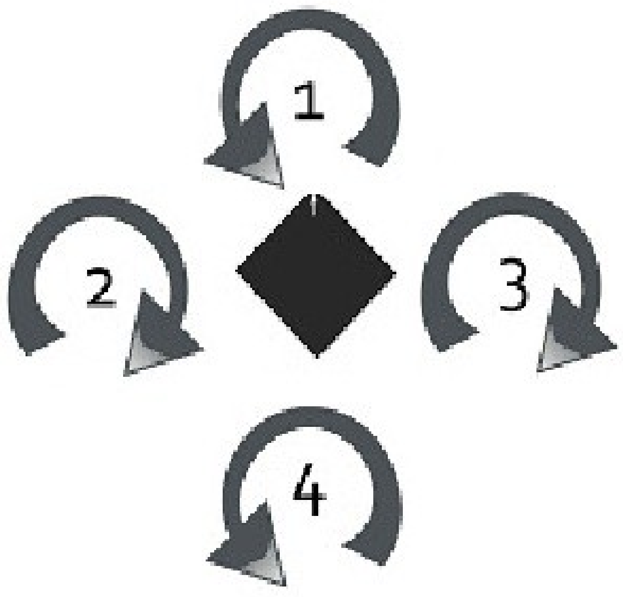
The Multi-Rotor controller uses Murata piezo gyros that are less sensitive to vibration than SMD type gyros, but it is still a good idea to mount the board on a vibration dampening material. The board must also be mounted with the white arrow facing the direction of forward flight.

When connecting your Remote Control Receiver (RX) you must connect the white signal wire of the channels (CH1, CH2, CH3 and CH4) from your RX corresponding to the aileron, elevator, throttle and rudder to the inner pins on the board while the red (VCC) wires are connected to the center pins, and the black (GND) wires are connected to the pins on the outer edge of your board.



The pins marked M1 to M4 are connected to the 3 pin BEC plug from your ESCs. They follow the same convention as the RX pins with the white wires connected to the inner pins, the red wires to the center pins and the black wires to the outer pins. The ESCs and the connected motors are plugged onto the pins M1 to M4 in the following order depending on flight rotor configuration. Note also the direction of rotation for each motor. This is achieved by connecting the three ESC wires to the motors and swapping two of the wires to achieve rotation in the opposite direction.

Multi-Rotor (+ Configuration)



Setting up the Multi-Rotor controller

1.Checking transmitter channels:

-Take off the propellers.
-Turn on transmitter and flight controller.
-Set throttle to about 1/4. Motors should start.
-Move pitch (elevator) stick forward. Back motor should speed up. If not, reverse pitch (elevator) channel.
-Move roll (aileron) stick to the left. Right motor should speed up. If not, reverse roll (aileron) channel.
-Move yaw (rudder) stick to the left. Front and back motor should speed up. If not, reverse yaw (rudder) channel.

2. Transmitter throttle adjustment:

- Turn on transmitter and flight controller.
- If led does not turn on and stays on, lower your trim.
- If still no go, you may need to reverse the throttle channel.
- Arm your board by putting the left stick down and to the right for the LED to come on. If this does not happen, adjust your throttle and yaw trim down and to the right on your transmitter. Make sure you do not have any mixing switches on your Transmitter enabled.

3. Initial transmitter ATV/servo range settings:

- Pitch (elevator): 50%
- Roll (aileron): 50%
- Yaw (rudder): 100%

4. ESC throttle range:

- Turn yaw pot to zero.
- Turn on transmitter.
- Throttle stick to full.
- Turn on flight controller.
- Wait until the ESCs beep twice after the initial beeps. (Depend on which ESC's)
- Throttle stick to off. ESCs beep.
- Turn off flight controller.
- Restore the yaw pot.

5. Initial Gyro gain pot value is 50%. Increase until it starts to oscillate rapidly, then back off until it is stable again. Fast forward flight needs lower gain.

Too low gain is recognised by the Multi-Rotor being hard to control and/or always wanting to tip over.

6. Checking gyro directions:

- Take off the propellers.
- Turn on transmitter and flight controller.
- Set throttle to about 1/4. Motors should start.
- Tilt Multi-Rotor forward. Forward motor should speed up. If not, reverse pitch gyro.
- Tilt Multi-Rotor to the left. Left motor should speed up. If not, reverse roll gyro.
- Turn Multi-Rotor CW. Front and back motor should speed up. If not, reverse yaw gyro.

7. Reversing gyros:

- Set roll gain pot to zero.
- Turn on flight controller.
- LED flashes rapidly 10 times.
- Move the stick for the gyro you want to reverse.
- LED will blink continually.
- Turn off flight controller.
- If there is more gyros to be reversed, go to step 2, else set roll gain pot back.

8.Final check:

Place the Multi-Rotor on the ground, stand back a safe distance and slowly advance to about 1/2 throttle. Hold it steady when you start increasing the throttle, because the Multi-Rotor controller calibrates its gyros when throttle leaves zero, and then the gyros need to be at rest.

If the Multi-Rotor tries to twist away, check propeller and motor directions, gyro placement and trim
settings. A slight twist is OK.

If not, try to twist the quad. It should resist your movements. More gyro gain gives more resistance. If it starts to oscillate, reduce the gain. You should not need to reduce the gain below 40%.

Note: the correct procedure for taking off from the ground is as following:
1: The quad and its propellers needs to be motionless.
2: Increase the throttle (collective). Just as the throttle leaves zero, gyro calibration is performed.
3: Enjoy! And remember to close the throttle if you lose control. Much less damage.

NOTES: Do not use bigger propellers than you need. Light propellers gives faster response and more stability. Try to get it to hover at about mid stick (1/3 to 2/3 throttle). Use smaller/bigger propeller, different motor Kv or more/less Battery cells to achieve that.

Bibliography

[1] Jennifer Bell. (2013) *Road accidents account for almost 70% of head injuries at one UAE hospital* [Online]. Available: http://www.thenational.ae/uae/health/road-accidents-account-for-almost-70-of-head-injuries-at-one-uae-hospital

[2] ADAFRUIT (2010) *FTDI Friend – Breakout Board+ (tutorial)* [Online]. Available: http://www.adafruit.com/blog/2010/09/16/ftdi-friend-breakout-board-tutorial/

[3] Prospector. *Polylactic Acid (PLA) Typical Properties* [Online]. Available: http://plastics.ides.com/generics/34/c/t/polylactic-acid-pla-properties-processing

[4] Mark Johnson Cutler. (2010) *Design and Control of an Autonomous Variable-Pitch Quadrotor Helicopter* [Online]. Available: http://acl.mit.edu/papers/Cutler_Masters12.pdf

[5] Leap Motion Documentation. (2013) *Understanding the Java Sample Application* [Online]. Available: https://developer.leapmotion.com/documentation/Languages/Java/Guides/Sample_Java_Tutorial.html

[6] Mando. *How a Gyro Works* [Online]. Available: https://learn.sparkfun.com/tutorials/gyroscope/how-a-gyro-works

[7] Philips Semiconductors *Remote Control System RC-5 Including Command Tables*,http://www.praktiker.at/download/itmubu06.pdf, Publication No. 9388 706 23011, December 1992.

[8] Wikipedia, the free encyclopedia *Radar gun*, http://en.wikipedia.org/wiki/Radar_gun, 2013.

[9] Pan, D., "A Tutorial on MPEG/Audio Compression," *IEEE Multimedia*, Vol.2, pp.60-74, Summer 1998.

[10] Neltronics, *"How does a Speed Camera or Radar Gun work?"*, http://www.neltronics.com.au/downloads/Bel%20-%20How%20does%20a%20Speed%20Camera%20or%20Radar%20Gun%20work.pdf, 18 May 2011 [6 June 2013].

[11] Wikipedia.org, *"Regenerative Receiver"*, http://en.wikipedia.org/wiki/File:Regenerative_Receiver.png, 4 January 2005 [6 June 2013].

[12] Peter Jakab, *"Infrared circuits for remote control"*, http://jap.hu/electronic/infrared.html, [6 June 2013].

[13] Stuart Clare, *"Functional MRI : Methods and Applications"*, Page 6, Paragraph 1 and 2, http://users.fmrib.ox.ac.uk/~stuart/thesis/fmri.pdf, October 1997 [6 June 2013].

[14] Michael Brady, *"Basics of MRI"*, Pages 14 - 20. http://www.robots.ox.ac.uk/~jmb/lectures/medimanallecture1.pdf, 2004 [6 June 2013].

[15] Ben Beiske, *"Living the Dream"*. http://www.travelblog.org/Photos/4009932, [6 June 2013].

[16] Radio Receivers, *"Chapter 2 Principles of radio transmission"*, Paragraph 2. http://www.mikroe.com/old/books/rrbook/chapter2/chapter2.htm, 2003 [6 June 2013].

Bibliography

[1] Sullivan. Using an ir remote to control an arduino project, 2013. URL http://sullivan-22-201-fall2013.blogspot.ae/2013/11/using-ir-remote-to-control-arduino.html.