ADAPTABILITY OF QUADCOPTER UNMANNED AERIAL VECHICLE FOR MOSQUITO CONTROL

BY

J.PRABHA

16PCS010

Project Report Submitted

In Partial fulfillment of the requirements for the award of

Master's Degree in Computer Science

Department of Computer Science,

Avinashilingam Institute for Home Science and Higher Education for Women, (Deemed to be University),

Coimbatore-641043

April-2018

ADAPTABILITY OF QUADCOPTER UNMANNED AERIAL VECHICLE FOR MOSQUITO CONTROL

BY

J.PRABHA

16PCS010

Project Report Submitted

In Partial fulfillment of the requirements for the award of

Master's Degree in Computer Science

Department of Computer Science,

Avinashilingam Institute for Home Science and Higher Education for Women, (Deemed to be University),

Coimbatore-641043

Signature of the Head of the Department Signature of the Supervisor

Viva Voce Examination Held on: _____

Signature of the Examiners

ACKNOWLEDGEMENT

ACKNOWLEDGEMENT

I would like to express my sincere thanks to God Almighty, for his constant love and grace that he showered upon me.

I would like to express my deep sense of reverential gratitude and sincere thanks to **Shri, Dr. P. R. Krishnakumar, Chancellor,** Avinashilingam Institute for Home Science and Higher Education for Women, Coimbatore, for his support and encouragement during the course of my study.

I owe my great deal of gratitude to **Dr. Premavathy Vijayan, Vice Chancellor**, Avinashilingam Institute for Home Science and Higher Education for Women, Coimbatore, for extending all resources that facilitated the conduct of the present work.

I express my gratitude to **Dr. S. Kowsalya, Registrar**, Avinashilingam Institute for Home Science and Higher Education for Women, Coimbatore, for providing all facilities necessary for the work.

I also thankful to **Dr. A. Parvathi, Dean Faculty of Science,** Avinashilingam Institute for Home Science and Higher Education for Women, Coimbatore, for granting the facility required.

I wish to place my deep sense of gratitude to **Dr. V. Radha, Professor and Head, Department of Computer Science,** for providing all the facilities required to complete the project.

I heartily thank my esteemed project guided by **Dr.N.Valliammal, Assistant Professor Department of Computer Science**, for imparting the tremendous assistance and well-timed support for triumph of my project.

I express my honorable thanks to my project coordinator **Dr. G. Sudhamathy, Assistant Professor, Department of Computer Science,** for her kind advice and knowledgeable suggestions which helped me to complete my project successfully.

I would like to extend my hearty thanks to one and all who helped me directly or indirectly for successful completion of my project.

Finally, I take pride to thank my beloved parents, my family members and my friends without whose support, encouragement and kind blessings .I would not have succeeded in my Endeavour.

A

ABSTRACT

BSTRACT

In today's world everyday new technologies are arising. Among them one of the popular technologies is drones which is also known as Unmanned Aerial Vehicle (UAV). Within in a short span of time UAV technology is developed to a great extent in both developed as well as developing countries. UAV has skyrocketed over the past decade driving costs down and the number of potential applications up, one being mosquito control. UAVs are used in numerous numbers of fields including agriculture, military, pest control and industrial plants. This project deals with the adaptability of quadcopter UAV for mosquito control. It is a prototype model for UAV in mosquito control. In this the quadcopter USB camera identifies the mosquito egg laying spots. The arduino controller fixed inside the quadcopter is coded with the functions like pitch, rotating angle etc. The ardunio controller connected to the system with Bluetooth and relay signals the mobile app trigger to spray the larvicide using the sprayer on the areas before they develop as adult blood feeding insects.

CONTENTS

TABLE OF CONTENTS.NO

1. INTRODUCTION

1.1 PROBLEM DEFINITION

Today many technologies are arising to control the insects which cause many environmental issues. Among them one of the major problems is by mosquitoes which spread many diseases. Mosquito control systems are tasked with surveillance and targeted control of nuisance mosquitoes and potential vectors of pathogens that cause disease. Many techniques are used for mosquito control by government. This project is a prototype model for UAV in mosquito control. It detects and destroys the mosquitoes before they develop as blood feeding insects. For this UAVs are used since they are available at affordable costs.

1.2 OVERVIEW OF THE PROJECT

In today's world everyday new technologies are arising among them one of the popular technologies is UAV. They are used in numerous numbers of fields including agriculture, military, pest control and industrial plants. There are many types of UAVs classified depending on various parameters such as battery range, application used etc. In this project the type of UAV used is a quadcopter. Through the wired USB camera fixed to the quadcopter the mosquito egg laying spots are identified. The aurdino controller, relay and the bluetooth module are connected with the system enables mobile app trigger for the sprayer to spray the larvicide on the detected areas.

SYSTEM STUDY

2. SYSTEM STUDY

2.1 EXISTING SYSTEM

In the existing system many methods are there to control the mosquitoes since they cause a major environmental issue to the people. They spread many harmful diseases so destroying them becomes an unavoidable thing. Both the government and the people as individual are using many methods to destroy them. Natural methods are used by people to destroy the mosquitoes in their own surroundings. The Government is trying to create awareness among people about how to keep places clean so that the mosquitoes will not even have places to lay the egg larva. In the same time they make use of environment department people to spray the larvicide or pesticide manually in the lakes, ponds and places where the mosquitoes lay eggs. But all these make more time and manpower. To overcome these factors and to make the work easy and affordable the proposed model is designed.

2.2 PROPOSED SYSTEM

UAVs are used in the proposed system to overcome the difficulties faced by the existing system. In this a prototype model of UAVs used in controlling mosquitoes is designed. UAVs are of many types. The type of UAV used in this is a Quadcopter. The camera in the quadcopter detects the places. Then the sprayer sprays the larvicide on the detected places. In this manpower is less when comparing with the existing system. With the onetime investment in the hardware part we can use it for long time and it is easy to include new technologies if needed.

SYSTEM DESIGN

3. SYSTEM DESIGN AND ANALYSIS

This project is the prototype model for the adaptability of quadcopter UAV for mosquito control.

3.1 QUADCOPTER

In this project the type of UAV used is a quadcopter. The quadcopter is activated using the remote controller. According to the movements in the joystick in the controller the quadcopter flies. The arduino controller inside is used to write and upload computer code for setting the drone pitch, rotating angles, motor control etc. The pitch values various with the place. The motor control differs according to the speed. The rotating angles changes with directions. All the movements coding are upload in the arduino controller and it is fixed inside the quadcopter. The quadcopter flies after receiving the signal from the controller. While flying the USB camera connected between the system and the quadcopter detects the mosquito egg laying spots for destroying them.

3.2 AURDINO CONTROLLER

Arduino is an open-source platform used for building projects. Arduino consists of both a physical programmable circuit board and a piece of software or IDE (Integrated Development Environment) that runs on the computer. It is used to write and upload computer code to the physical board. In this one aurdino controller is fixed with the quadcopter and the other one is connected to the system. With that the Bluetooth and the relay are connected. Arduino controller which is fixed inside quadcopter is for uploading the coding for it. Next one is for uploading the coding for relay to signal the sprayer. Relays are switches that open and close circuits electromechanically or electronically.

3.3 RELAY

Relays control one electrical circuit by opening and closing contacts in another circuit. This relay opens and closes the sprayer through the mobile app trigger. The connection between the relay and the mobile app trigger is through the Bluetooth module.

3.4 BLUETOOTH MODULE

It enables connection between the mobile app trigger and the sprayer.usiing the signals from the app the sprayer sprays the larvicide on the areas.

3.4 MOBILE APP

In this the app used is arduino Bluetooth controller which enables connection to the sprayer usig the Bluetooth module connected with the aurduino and relay in the system.

SYSTEM SPECIFICATION

4. SYSTEM SPECIFICATION

4.1 HARDWARE REQUIREMENTS

The hardware requirements used in this project are

4.1.1 QUADCOPTER

There are different types of drones like fixed wing drones, single rotary drones, GPS drones, multi rotary drones, quad copter, tri copter, hexa copter etc. According to the application drones are used. Quad copter is used for mosquito control. A quad copter, also called a quad rotor helicopter or quad rotor, is a multi rotor helicopter that is lifted and propelled by four rotors. Quad copters are classified as rotorcraft, as opposed to fixed-wing aircraft, because their lift is generated by a set of rotors (vertically oriented propellers).

A quad copter, or multi rotor, drone, or quad rotor, is a simple flying mechanical vehicle that has four arms, and in each arm there is a motor attached to a propeller. Multi copters with three, six or eight arms are also possible, but work on the same principal as a quad copter. Two of the rotors turn clockwise, while the other two turn counter clockwise. Quad copters are aerodynamically unstable, and require a flight computer to convert your input commands into commands that change the RPMs of the propellers to produce the desired motion. This is the short answer to the question of what is a quad copter. The remainder of this article provides further information to introduce you to the world of quad copters.

Quadcopters differ from a helicopter or a fixed wing aircraft in the way they generate lift and control forces. For an aircraft the lift is generated by the wings, but in a quad copter the lift is generated by the propellers. A helicopter uses its main rotor to generate lift, but also have the ability to vary the pitch of the rotor blades to generate control forces.

The quad copter concept is not new. Manned quad copter designs appeared in the 1920s and 1930s, but these early concepts had bad performance, a high level of instability, and required a lot of pilot inputs. The advancement of electronic technology in flight control computers, coreless or brushless motors, smaller microprocessors, batteries, accelerometers, cameras, and even GPS systems made it possible to design and fly quad copters. The simplicity of the quad copter has made it a very effective aerial photography and video platform.

A typical quad copter comes with a 4 channel controller that sends commands to the drone to affect its throttle, yaw, pitch and roll. The communications frequency used by most controllers is 2.4 GHz. This is also the typical frequency used for WiFi connections. Though it is unlikely that interference takes place, it may be something worth checking if you are experience communications lag or dropouts with your quad copter.

The controller also has 4 trim buttons to allow you to make minor corrections to your quad copter flight behaviour. If you notice that with no control input, the drone drifts in a particular direction, applying trim for that control input in the opposite direction will remove the drift.

It is important that you always remember that there are two sets of propellers, and two set of motors that rotate in the opposite direction. The pitches on the two sets of propellers are different. So if you switch a set of opposite propellers, they will be blowing air up instead of down. This results in a downward force on the quad copter leading to your drone flipping, and not flying. When you take the propellers are sure to mark which motors they belong to. If you look carefully, the propeller will have a mark indicating which set of motors that it belongs to.

Hovering takes place when the upward lift balances the downward force of gravity. To make the drone to higher, the lift force needs to be greater than the force of gravity. This is achieved by increasing RPM on all four propellers simultaneously, which produces more lift. To decrease the altitude, the RPM is decreased on all four propellers simultaneously. On your controller, this is accomplished by pushing the left stick up or down, which increases or decreases the rotor RPM, which makes your quadcopter go up or down.

In order to move the drone left, right, forwards, and backwards, we change the angle of the lifting force so it has a vertical and horizontal component. The vertical component still serves to keep the quad copter in the air, while the horizontal component allows produces control thrust.

To make quad copter move forward or backwards, we adjust the pitch using the right stick on the controller. What happens is that the front propellers decrease RPMs, while the back propellers increase RPMs. Now the lift force has a horizontal component which results in moving the quad copter forward. The opposite happens to make your quad copter go backwards. The front propeller RPM is increased, while the back propeller RPM is decreased.

To move your quad copter move sideways, a similar change in RPM takes place, but this time it is on the left and right propellers. To make your drone move to left, you need to point the lift force slightly to the left. This is done by decreasing the RPM on the left rotors,

and increasing the RPM on the right side. A similar change in the rotor RPMs takes place to move your quad copter to the left.

On a quad copter, if all four propellers rotated the same way, then the body of your quad copter would rotate the opposite way. But it obviously doesn't. This is because two of the propellers rotate clockwise, and the other two rotate anti-clockwise. So one set of propellers produces a torque in one direction, but the other two propellers are producing a torque in the opposite direction. These cancel out, so your quad copter does not rotate.

For a yaw maneuver, we do want the quad copter to rotate. This is done by reducing the RPM on one set of rotors, while increasing the RPM on the other set of rotors. Now there is a net torque in one direction, so your quad copter rotates. To produce the opposite rotation, the RPMs on the propellers is simply reversed. The direction of the yaw maneuver is controlled by the left stick on the controller.

Quad copter flies and moves by changing the RPMs of each propeller. So when you move a stick on your controller, this command needs to be converted into the proper commands each of the four motors on your quad copter. This is done by the flight control system. The purpose of the flight computer is to simplify the coordination of the control of all four propellers needed to make your drone fly. The flight control computer has the ability to connect to several other devices and sensors. The primary device that it connects to is the remote control receiver, which is linked to your remote transmitter.

In advanced quad copter, the flight controller also has a transmitter to communicate with your controller, to provide two way communications. Typical sensors come with more complex quad copters include GPS, gyro compass, and barometer. Quad copters are an economical way to introduce you to the world of remote control aerial vehicles. They are fun to fly, and with some practice and patience anyone can learn how to fly a quad copter.

4.1.2 ARDUINO CONTROLLER UNO

The Arduino controller Uno is a microcontroller board based on the ATmega328 (datasheet). It has 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a 16 MHz ceramic resonator, a USB connection, a power jack, an ICSP header, and a reset button. It contains everything needed to support the microcontroller; simply connect it to a computer with a USB cable or power it with an AC-to-DC adapter or battery to get started. The Uno differs from all preceding boards in that it does not use the FTDI USB-to-serial driver chip. Instead, it features the Atmega16U2 (Atmega8U2 up to version R2)

programmed as a USB-to-serial converter. Revision 2 of the Uno board has a resistor pulling the 8U2 HWB line to ground, making it easier to put into DFU mode. The board has the following new features:

- pinout: added SDA and SCL pins that are near to the AREF pin and two other new pins placed near to the RESET pin, the IOREF that allow the shields to adapt to the voltage provided from the board. In future, shields will be compatible both with the board that use the AVR, which operate with 5V and with the Arduino controller Due that operate with 3.3V. The second one is a not connected pin that is reserved for future purposes.
- Stronger RESET circuit.
- Atmega 16U2 replace the 8U2.

"Uno" means one in Italian and is named to mark the upcoming release of Arduino controller 1.0. The Uno and version 1.0 will be the reference versions of Arduino controller, moving forward. The Uno is the latest in a series of USB Arduino controller boards, and the reference model for the Arduino controller platform; for a comparison with previous versions, see the index of Arduino controller boards.

4.1.2 SCHEMATIC & REFERENCE DESIGN

EAGLE files: arduino controller-uno-Rev3-reference-design.zip (NOTE: works with Eagle 6.0 and newer) Schematic: arduino controller-uno-Rev3-schematic.pdf. The Arduino controller reference design can use an Atmega8, 168, or 328, Current models use an ATmega328, but an Atmega8 is shown in the schematic for reference. The pin configuration is identical on all three processors.

4.1.3 POWER

The Arduino controller Uno can be powered via the USB connection or with an external power supply. The power source is selected automatically. External (non-USB) power can come either from an AC-to-DC adapter (wall-wart) or battery. The adapter can be connected by plugging a 2.1mm center-positive plug into the board's power jack. Leads from a battery can be inserted in the Gnd and Vin pin headers of the POWER connector. The board can operate on an external supply of 6 to 20 volts. If supplied with less than 7V, however, the 5V pin may supply less than five volts and the board may be unstable. If using more than

12V, the voltage regulator may overheat and damage the board. The recommended range is 7 to 12 volts. The power pins are as follows:

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

4.1.4 MEMORY

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

4.1.5 INPUT AND OUTPUT

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

- **Serial: 0 (RX) and 1 (TX).** Used to receive (RX) and transmit (TX) TTL serial data. These pins are connected to the corresponding pins of the ATmega8U2 USB-to-TTL Serial chip.

- **External Interrupts: 2 and 3.** These pins can be configured to trigger an interrupt on a low value, a rising or falling edge, or a change in value. See the attachInterrupt() function for details.
- **PWM: 3, 5, 6, 9, 10, and 11.** Provide 8-bit PWM output with the analogWrite() function.
- **SPI: 10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK).** These pins support SPI communication using the SPI library.
- **LED: 13.** There is a built-in LED connected to digital pin 13. When the pin is HIGH value, the LED is on, when the pin is LOW, it's off.

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

TWI: A4 or SDA pin and A5 or SCL pin. Support TWI communication using the Wire library. There are a couple of other pins on the board:

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

See also the mapping between Arduino controller pins and ATmega328 ports. The mapping for the Atmega8, 168, and 328 is identical.

4.1.6 COMMUNICATION

The Arduino controller Uno has a number of facilities for communicating with a computer, another Arduino controller, or other microcontrollers. The ATmega328 provides UART TTL (5V) serial communication, which is available on digital pins 0 (RX) and 1 (TX). An ATmega16U2 on the board channels this serial communication over USB and appears as a virtual com port to software on the computer. The '16U2 firmware uses the standard USB COM drivers, and no external driver is needed. However, on Windows, a .inf file is required. The Arduino controller software includes a serial monitor which allows simple textual data to be sent to and from the Arduino controller board. The RX and TX LEDs on the board will

flash when data is being transmitted via the USB-to-serial chip and USB connection to the computer (but not for serial communication on pins 0 and 1).

A Software Serial library allows for serial communication on any of the Uno's digital pins.The ATmega328 also supports I2C (TWI) and SPI communication. The Arduino controller software includes a Wire library to simplify use of the I2C bus; see the documentation for details. For SPI communication, use the SPI library.

4.1.7 PROGRAMMING

The Arduino controller Uno can be programmed with the Arduino controller software (download). Select "Arduino controller Uno from the **Tools > Board** menu (according to the microcontroller on your board). For details, see the reference and tutorials. The ATmega328 on the Arduino controller Uno comes preburned with a bootloader that allows you to upload new code to it without the use of an external hardware programmer. It communicates using the original STK500 protocol (reference, C header files).You can also bypass the bootloader and program the microcontroller through the ICSP (In-Circuit Serial Programming) header; see these instructions for details.

The ATmega16U2 (or 8U2 in the rev1 and rev2 boards) firmware source code is available. The ATmega16U2/8U2 is loaded with a DFU bootloader, which can be activated by:

- On Rev1 boards: connecting the solder jumper on the back of the board (near the map of Italy) and then resetting the 8U2.
- On Rev2 or later boards: there is a resistor that pulling the 8U2/16U2 HWB line to ground, making it easier to put into DFU mode.

You can then use Atmel's FLIP software (Windows) or the DFU programmer (Mac OS X and Linux) to load a new firmware. Or you can use the ISP header with an external programmer (overwriting the DFU bootloader). See this user-contributed tutorial for more information.

4.1.8 AUTOMATIC (SOFTWARE) RESET

Rather than requiring a physical press of the reset button before an upload, the Arduino controller Uno is designed in a way that allows it to be reset by software running on a connected computer. One of the hardware flow control lines (DTR) of the

ATmega8U2/16U2 is connected to the reset line of the ATmega328 via a 100 nanofarad capacitor. When this line is asserted (taken low), the reset line drops long enough to reset the chip. The Arduino controller software uses this capability to allow you to upload code by simply pressing the upload button in the Arduino controller environment. This means that the bootloader can have a shorter timeout, as the lowering of DTR can be well-coordinated with the start of the upload. This setup has other implications. When the Uno is connected to either a computer running Mac OS X or Linux, it resets each time a connection is made to it from software (via USB). For the following half-second or so, the bootloader is running on the Uno. While it is programmed to ignore malformed data (i.e. anything besides an upload of new code), it will intercept the first few bytes of data sent to the board after a connection is opened. If a sketch running on the board receives one-time configuration or other data when it first starts, make sure that the software with which it communicates waits a second after opening the connection and before sending this data.The Uno contains a trace that can be cut to disable the auto-reset. The pads on either side of the trace can be soldered together to re-enable it. It's labeled "RESET-EN". You may also be able to disable the auto-reset by connecting a 110 ohm resistor from 5V to the reset line; see this forum thread for details.

4.1.9 USB OVERCURRENT PROTECTION

The Arduino controller Uno has a resettable polyfuse that protects your computer's USB ports from shorts and overcurrent. Although most computers provide their own internal protection, the fuse provides an extra layer of protection. If more than 500 mA is applied to the USB port, the fuse will automatically break the connection until the short or overload is removed.

4.1.10 PHYSICAL CHARACTERISTICS

The maximum length and width of the Uno PCB are 2.7 and 2.1 inches respectively, with the USB connector and power jack extending beyond the former dimension. Four screw holes allow the board to be attached to a surface or case. Note that the distance between digital pins 7 and 8 is 160 mil (0.16"), not an even multiple of the 100 mil spacing of the other pins.

4.2 RELAY

A relay is an electrically operated or electromechanical switch composed of an electromagnet, an armature, a spring and a set of electrical contacts. The electromagnetic

switch is operated by a small electric current that turns a larger current on or off by either releasing or retracting the armature contact, thereby cutting or completing the circuit. Relays are necessary when there must be electrical isolation between controlled and control circuits, or when multiple circuits need to be controlled by a single signal.

4.3 EMBEDDED C

Embedded C is one of the most popular and most commonly used Programming Languages in the development of Embedded Systems. So, in this article, we will see some of the Basics of Embedded C Program and the Programming Structure of Embedded C

As mentioned earlier, Embedded Systems consists of both Hardware and Software. If we consider a simple Embedded System, the main Hardware Module is the Processor. The Processor is the heart of the Embedded System and it can be anything like a Microprocessor, Microcontroller, DSP, CPLD (Complex Programmable Logic Device) and FPGA (Field Programmable Gated Array).All these devices have one thing in common: they are programmable i.e. we can write a program (which is the software part of the Embedded System) to define how the device actually works.

Embedded Software or Program allows Hardware to monitor external events (Inputs) and control external devices (Outputs) accordingly. During this process, the program for an Embedded System may have to directly manipulate the internal architecture of the Embedded Hardware (usually the processor) such as Timers, Serial Communications Interface, Interrupt Handling, and I/O Ports etc.

From the above statement, it is clear that the Software part of an Embedded System is equally important to the Hardware part. There is no point in having advanced Hardware Components with poorly written programs (Software).

There are many programming languages that are used for Embedded Systems like Assembly (low-level Programming Language), C, C++, JAVA (high-level programming languages), Visual Basic, JAVA Script (Application level Programming Languages), etc.

In the process of making a better embedded system, the programming of the system plays a vital role and hence, the selection of the Programming Language is very important.

The following are few factors that are to be considered while selecting the Programming Language for the development of Embedded Systems.

- Size: The memory that the program occupies is very important as Embedded Processors like Microcontrollers have a very limited amount of ROM.
- Speed: The programs must be very fast i.e. they must run as fast as possible. The hardware should not be slowed down due to slow running software.
- Portability**:** The same program can be compiled for different processors.
- Ease of Implementation
- Ease of Maintenance
- Readability

Earlier Embedded Systems were developed mainly using Assembly Language. Even though Assembly Language is closest to the actual machine code instructions, the lack of portability and high amount of resources spent on developing the code, made the Assembly Language difficult to work with.

There are other high-level programming languages that offered the above mentioned features but none were close to C Programming Language.

4.3.1 INTRODUCTION TO EMBEDDED C PROGRAMMING LANGUAGE

Before going in to the details of Embedded C Programming Language and basics of Embedded C Program, we will first talk about the C Programming Language. The C Programming Language, developed by Dennis Ritchie in the late 60's and early 70's, is the most popular and widely used programming language. The C Programming Language provided low level memory access using an uncomplicated compiler (a software that converts programs to machine code) and achieved efficient mapping to machine instructions. The C Programming Language became so popular that it is used in a wide range of applications ranging from Embedded Systems to Super Computers. Embedded C Programming Language, which is widely used in the development of Embedded Systems, is an extension of C Program Language. The Embedded C Programming Language uses the same syntax and semantics of the C Programming Language like main function, declaration of data types, defining variables, loops, functions, statements, etc.The extension in Embedded

C from standard C Programming Language include I/O Hardware Addressing, fixed point arithmetic operations, accessing address spaces, etc.

4.3.2 DIFFERENCE BETWEEN C AND EMBEDDED C

There is actually not much difference between C and Embedded C apart from few extensions and the operating environment. Both C and Embedded C are ISO Standards that have almost same syntax, data types, functions, etc. Embedded C is basically an extension to the Standard C Programming Language with additional features like Addressing I/O, multiple memory addressing and fixed-point arithmetic, etc. C Programming Language is generally used for developing desktop applications whereas Embedded C is used in the development of Microcontroller based applications.

4.3.3 BASICS OF EMBEDDED C PROGRAM

Now that we have seen a little bit about Embedded Systems and Programming Languages, we will dive in to the basics of Embedded C Program. We will start with two of the basic features of the Embedded C Program: Keywords and Data types.

4.3.4 Keywords in Embedded C

A Keyword is a special word with a special meaning to the compiler (a C Compiler for example, is software that is used to convert program written in C to Machine Code). For example, if we take the Keil's Cx51 Compiler (a popular C Compiler for 8051 based Microcontrollers) the following are some of the keywords:

- bit
- sbit
- sfr
- small
- large

These are few of the many keywords associated with the Cx51 C Compiler along with the standard C Keywords.

4.3.5 DATA TYPES IN EMBEDDED C

Data Types in C Programming Language (or any programming language for that matter) help us declaring variables in the program. There are many data types in C Programming Language like signed int, unsigned int, signed char, unsigned char, float, double, etc. In addition to these there few more data types in Embedded C.The following are the extra data types in Embedded C associated with the Keil's Cx51 Compiler.

- bit
- sbit
- sfr
- sfr16

4.3.6 DIFFERENT COMPONENTS OF AN EMBEDDED C PROGRAM

- **Comments:** Comments are readable text that are written to help us (the reader) understand the code easily. They are ignored by the compiler and do not take up any memory in the final code (after compilation).There are two ways you can write comments: one is the single line comments denoted by // and the other is multiline comments denoted by /*….*/.

- **Pre-processor Directive:** A Pre-processor Directive in Embedded C is an indication to the compiler that it must look in to this file for symbols that are not defined in the program.In C Programming Language (also in Embedded C), Preprocessor Directives are usually represented using #include… or #define….In Embedded C Programming, we usually use the preprocessor directive to indicate a header file specific to the microcontroller, which contains all the SFRs and the bits in those SFRs.In case of 8051, Keil Compiler has the file "reg51.h", which must be written at the beginning of every Embedded C Program.

- **Global Variables:** Global Variables, as the name suggests, are Global to the program i.e. they can be accessed anywhere in the program.

- **Local Variables:** Local Variables, in contrast to Global Variables, are confined to their respective function.

CONCLUSION

5. CONCLUSION

UAV are incredible technology in just a few years. UAV are crucial in mosquito control. Many countries, now, have the best mosquito surveillance and control technology. It's true to say that mosquito control has become technically advanced in many places. This project is a prototype model for adaptability of quadcopter UAV in mosquito control which can be developed as a real time system in near future. It is really very useful for providing a solution for the environmental issue caused by the mosquitoes. Perhaps, one day these UAV will become fully automated. They detect the places through image processing and destroy the mosquitoes.

SCOPE FOR FUTURE ENHANCEMENT

6. SCOPE FOR FUTURE ENCHANCEMENT

In today's scenario UAV have become very popular. In the proposed system the quad copter control mosquitoes only with short distance range. As a future enhancement more technically efficient UAV can be used. Instead of quad copter GPS UAV can be used. The GPS UAV is connected with satellites. It covers a large geographical area. Also tri copters, hexa copters etc can be used. Since the mosquitoes spreading diseases is increasing every day, new inventions are made to control and destroy them at the early stage. Comparing with other technologies UAV is easy to use and available in affordable cost.

BIBLIOGRAPHY

7. BIBLIOGRAPHY

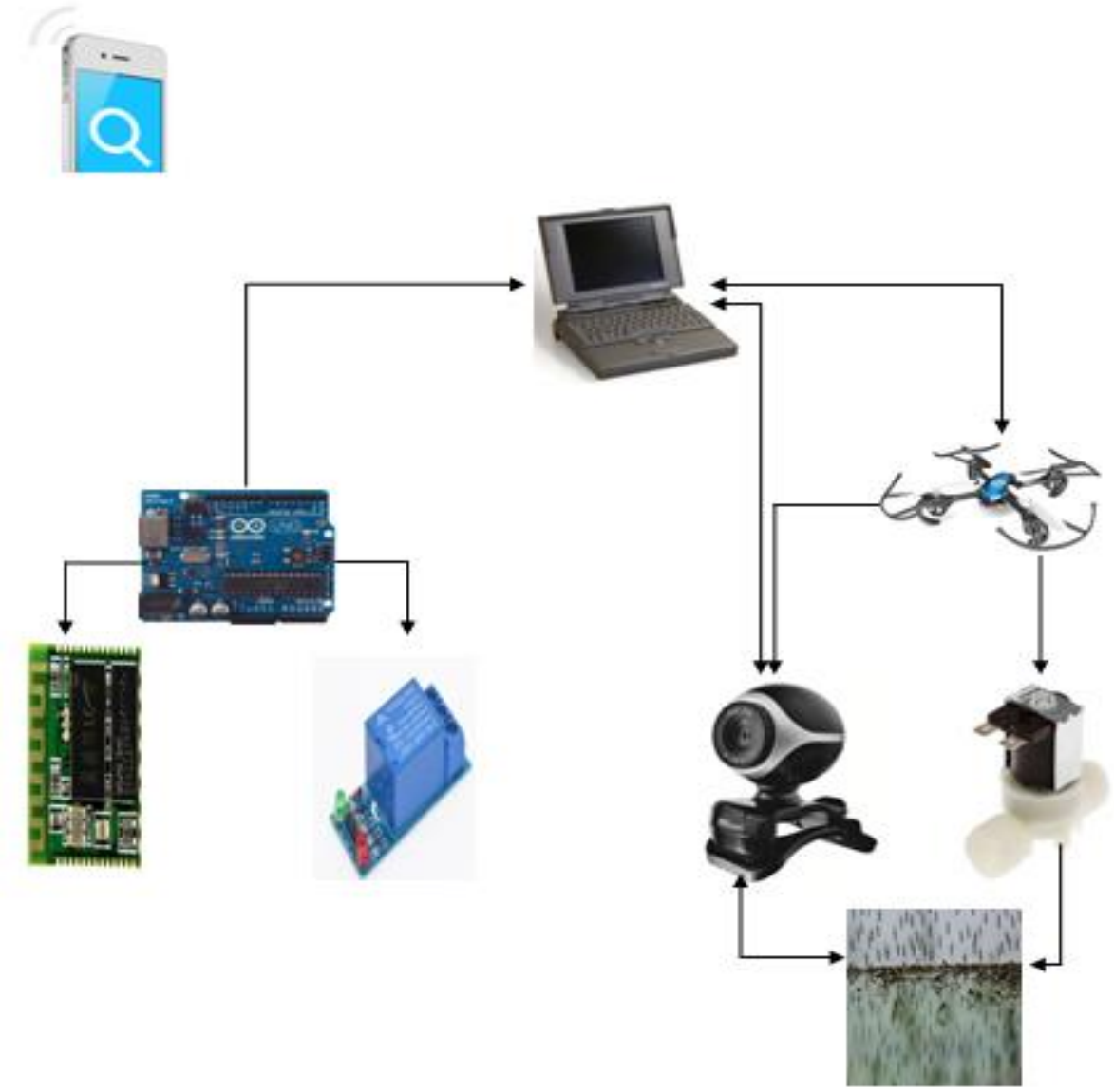
7.1 REFERENCE

- Duncan Still, third edition, 16 February 2005, How to build a quadcopter drone: A complete guide to building a radio controlled quadcopter.
- Jorge Dias, Lakme sanivertane, 2 november 2014,A survey of unmanned aerial vehicles: Recent development and trends.
- Adam Rothstein, third edition, 8 may 2001,The UAV book.
- Alex Elliott, first edition, 4 January 2004, Build Your Own Drone Manual:
- Craig S Issod, second edition 10 June 2013, Getting Started with Hobby Quadcopters and UAV: Learn about, buy and fly these amazing aerial vehicles.
- Lan cinnamon, Romi kadri, Fitz tepper, first edition, 25 November 2016, DIY UAV for the Evil Genius: Design, Build, and Customize Your Own UAV.
- Sarah kurney, first edition, 20 august 2003, UAV and targeted killings.

APPENDIX

APPENDIX

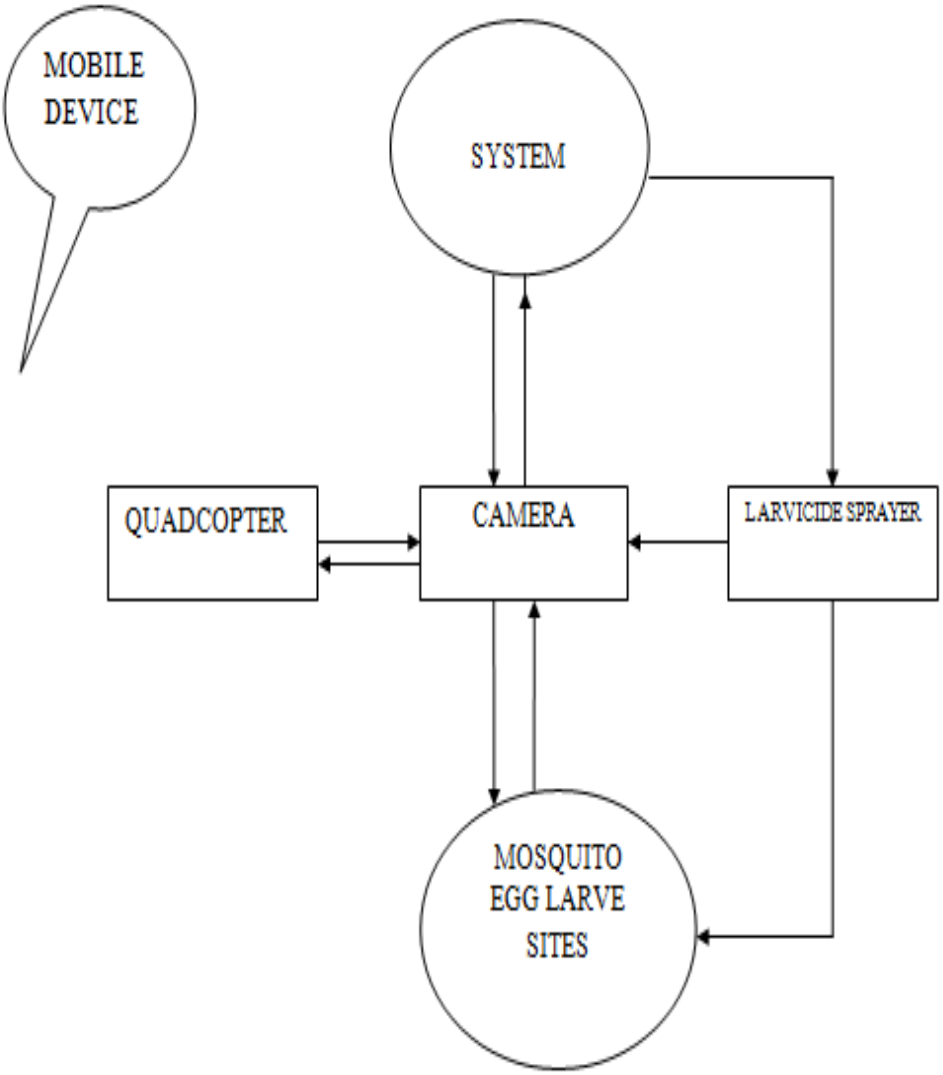
8.1 BLOCK DIAGRAM



DESCRIPTION

The components used in this project are quadcopter, camera, sprayer, aurdino controller, relay and Bluetooth. The quadcopter is fixed with the camera connected to the system. The aurduino controller is connected with the Bluetooth and relay which enables the sprayer to spray the larvicide through the mobile app trigger.

8.2 FLOW DIAGRAM



8.3 SCREENSHOTS

Figure1: Quadcopter

FIGURE 2: QUADCOPTER FIXED WITH CAMERA

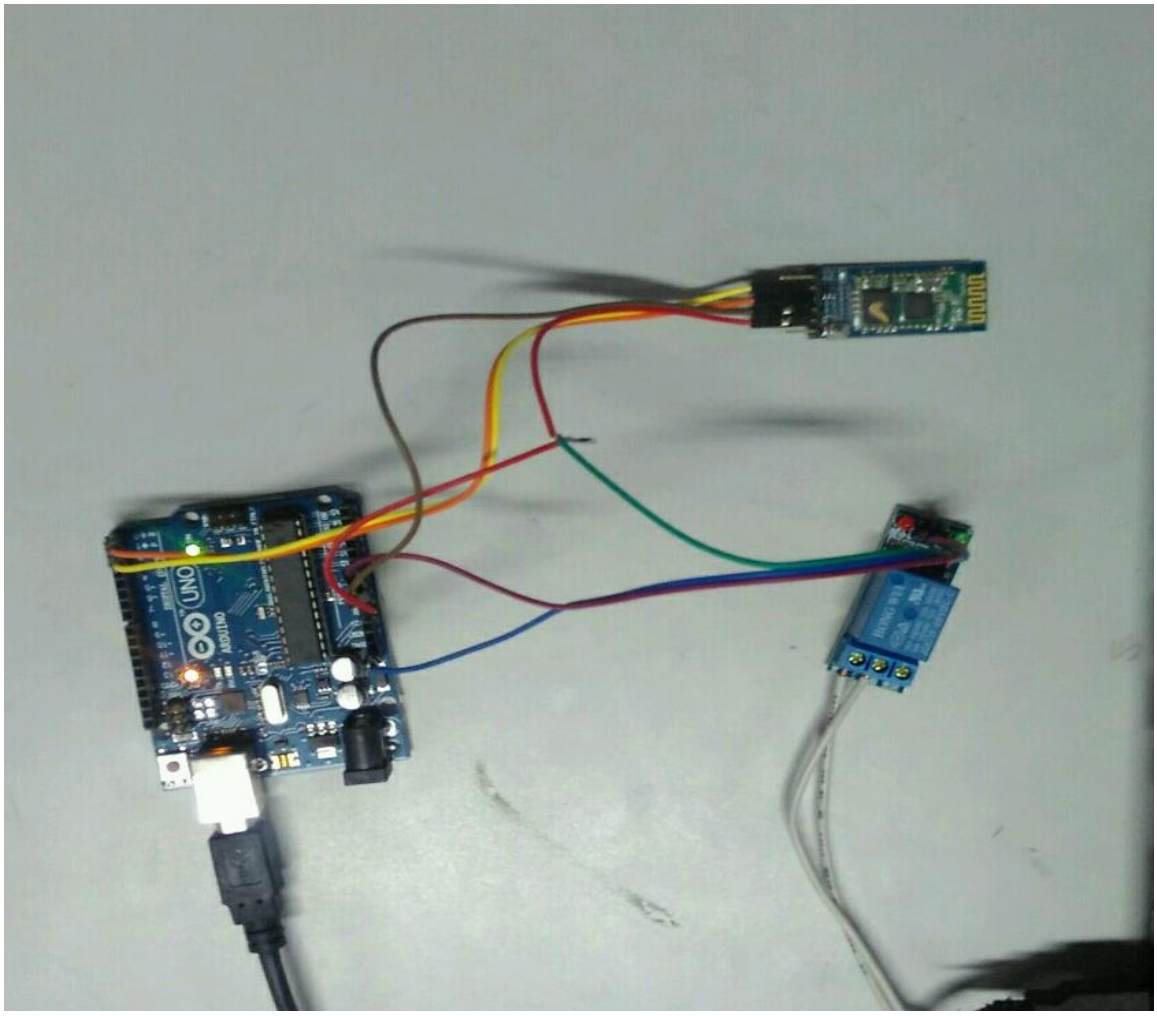


FIGURE 3: ***A*URDINO CONTROLLER UNO , BLUETOOTH AND RELAY CONNECTION**

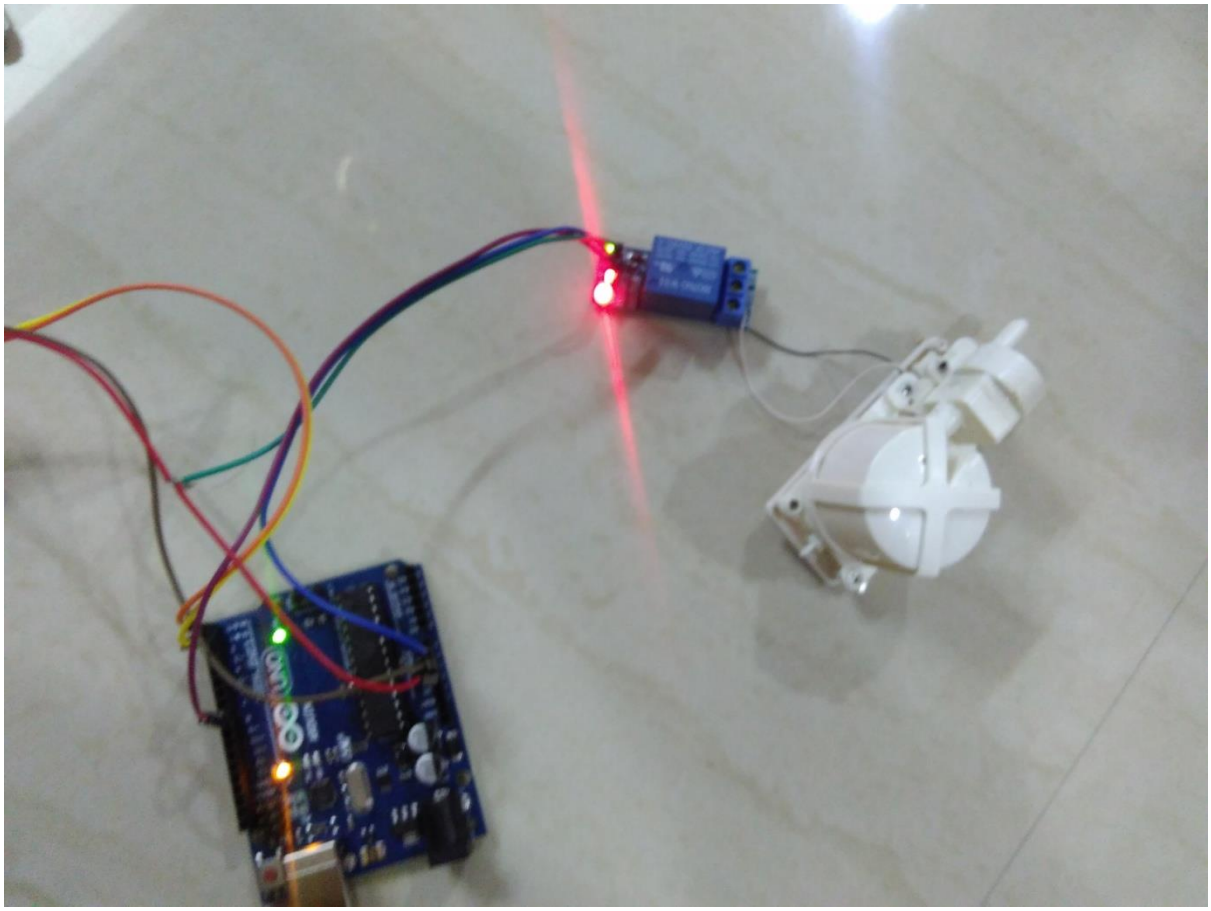


FIGURE 3: ***A*URDINO CONTROLLER UNO , BLUETOOTH AND RELAY CONNECTION**

SAMPLE CODING

```
#include "Configuration.h"
#include <Math.h>
#include <PID_v1.h>
#include <PinChangeInt.h>
#include <Servo.h>
#include <Wire.h>
// Angles
float angleX,angleY,angleZ = 0.0;
// RX Signals
int throttle=THROTTLE_RMIN;
volatile int rx_values[4]; // ROLL, PITCH, THROTTLE, YAW
// PID variables
double pid_roll_in,   pid_roll_out,   pid_roll_setpoint = 0;
double pid_pitch_in,  pid_pitch_out,  pid_pitch_setpoint = 0;
double pid_yaw_in,    pid_yaw_out,    pid_yaw_setpoint = 0;
  // Motors
int m0, m1, m2, m3; // Front, Right, Back, Left
// Track loop time.
unsigned long prev_time = 0;
void setup()
{
 #ifdef DEBUG_OUTPUT
 Serial.begin(115200);
 while(!Serial);
 Serial.println("Debug Output ON");
```

```
 #endif
 motors_initialize();
 leds_initialize();
 rx_initialize();
 pid_initialize();
 motors_arm();
 }
void loop()
{
imu_update();
control_update();
#ifdef DEBUG_OUTPUT
debug_process();
 #endif
 prev_time = micros();
}
```

COFIGURATION

```
#define SAFE
//-------PID Config----------
#define ROLL_PID_KP  0.250
#define ROLL_PID_KI  0.950
#define ROLL_PID_KD  0.011
#define ROLL_PID_MIN  -200.0
#define ROLL_PID_MAX  200.0
#define PITCH_PID_KP  0.250
#define PITCH_PID_KI  0.950
#define PITCH_PID_KD  0.011
```

```
#define PITCH_PID_MIN  -200.0
#define PITCH_PID_MAX  200.0
#define YAW_PID_KP  0.680
#define YAW_PID_KI  0.500
#define YAW_PID_KD  0.0001
#define YAW_PID_MIN  100.0
#define YAW_PID_MAX  100.0
//-------RX Config----------
#define THROTTLE_RMIN  1000
#define THROTTLE_SAFE_SHUTOFF 1120
#define THROTTLE_RMAX  1900
#define THROTTLE_RMID  1470
#define ROLL_RMIN  THROTTLE_RMIN
#define ROLL_RMAX  THROTTLE_RMAX
#define ROLL_WMIN  -30
#define ROLL_WMAX  30
#define PITCH_RMIN  THROTTLE_RMIN
#define PITCH_RMAX  THROTTLE_RMAX
#define PITCH_WMIN  -30
#define PITCH_WMAX  30
#define YAW_RMIN  THROTTLE_RMIN
#define YAW_RMAX  THROTTLE_RMAX
#define YAW_WMIN  -30
#define YAW_WMAX  30
//-------IMU Config-----------
#define ADDR_SLAVE_I2C 2
#define PACKET_SIZE 12
```

```
//-------Debug Config---------
#define DEBUG_OUTPUT
#define DEBUG_ANGLES
#define DEBUG_PID
#define DEBUG_RX
#define DEBUG_MOTORS
#define DEBUG_LOOP_TIME
//----------------------------
//-------Motor PWM Levels
#define MOTOR_ZERO_LEVEL  1000
#define MOTOR_ARM_START  1500
#define MOTOR_MAX_LEVEL  2000
//-------RX PINS-------------
#define RX_PINS_OFFSET 2
#define PIN_RX_ROLL 2
#define PIN_RX_PITCH 3
#define PIN_RX_THROTTLE 4
#define PIN_RX_YAW 5
//-------MOTOR PINS-----------
#define PIN_MOTOR0  6
#define PIN_MOTOR1  9
#define PIN_MOTOR2  10
#define PIN_MOTOR3  11
//-------LED PINS-------------
#define PIN_LED 13
```

MOTORS

```
Servo motor0;
```

```
Servo motor1;
Servo motor2;
Servo motor3;
void motors_initialize(){
  motor0.attach(PIN_MOTOR0);
  motor1.attach(PIN_MOTOR1);
  motor2.attach(PIN_MOTOR2);
  motor3.attach(PIN_MOTOR3);
  motor0.writeMicroseconds(MOTOR_ZERO_LEVEL);
  motor1.writeMicroseconds(MOTOR_ZERO_LEVEL);
  motor2.writeMicroseconds(MOTOR_ZERO_LEVEL);
  motor3.writeMicroseconds(MOTOR_ZERO_LEVEL);
}
void motors_arm(){
  motor0.writeMicroseconds(MOTOR_ZERO_LEVEL);
  motor1.writeMicroseconds(MOTOR_ZERO_LEVEL);
  motor2.writeMicroseconds(MOTOR_ZERO_LEVEL);
  motor3.writeMicroseconds(MOTOR_ZERO_LEVEL);
}
void update_motors(int m0, int m1, int m2, int m3)
{
  motor0.writeMicroseconds(m0);
  motor1.writeMicroseconds(m1);
  motor2.writeMicroseconds(m2);
  motor3.writeMicroseconds(m3);
}
```

LEDS

```
void leds_initialize(){
 pinMode(PIN_LED, OUTPUT);
 digitalWrite(PIN_LED, LOW);
}
void leds_status(byte stat){
 digitalWrite(PIN_LED, stat);
}
```

RECEIVER

```
volatile int pwm_start_time[4];
uint8_t latest_interrupted_pin;
void rx_initialize() {
 pinMode(PIN_RX_ROLL, INPUT); digitalWrite(PIN_RX_ROLL, HIGH);
 PCintPort::attachInterrupt(PIN_RX_ROLL, &rising, RISING);
 pinMode(PIN_RX_PITCH, INPUT); digitalWrite(PIN_RX_PITCH, HIGH);
 PCintPort::attachInterrupt(PIN_RX_PITCH, &rising, RISING);
pinMode(PIN_RX_THROTTLE, INPUT); digitalWrite(PIN_RX_THROTTLE, HIGH);
PCintPort::attachInterrupt(PIN_RX_THROTTLE, &rising, RISING);
pinMode(PIN_RX_YAW, INPUT); digitalWrite(PIN_RX_YAW, HIGH);
PCintPort::attachInterrupt(PIN_RX_YAW, &rising, RISING);
}
void rising()
{
 latest_interrupted_pin=PCintPort::arduino controllerPin;
 PCintPort::attachInterrupt(latest_interrupted_pin, &falling, FALLING);
 pwm_start_time[latest_interrupted_pin-RX_PINS_OFFSET] = micros();
```

```
}

void falling() {

latest_interrupted_pin=PCintPort::arduino controllerPin;

PCintPort::attachInterrupt(latest_interrupted_pin, &rising, RISING);

rx_values[latest_interrupted_pin-RX_PINS_OFFSET] = micros()-
pwm_start_time[latest_interrupted_pin-RX_PINS_OFFSET];
```