Reconnaissance Using UMV (Unmanned Marine Vehicle)

A Project Report

Submitted By

Ashutosh Bhatt

Nikunj Jadawala

Priyal Joshi

Maharsh Shah

In fulfillment for the award of the degree

of

BACHELOR OF ENGINEERING

in

Electronics And Communication Engineering

Indus Institute Of Technology And Engineering, Ahmedabad

Gujarat Technological University, Ahmedabad

May 2013

Acknowledgement

We have taken efforts in this project. However, it would not have been possible without the kind support and help of many individuals and organizations. We would like to extend my sincere thanks to all of them.

We are highly indebted to “Mr. Urvish Nawab” for their guidance and constant supervision as well as for providing necessary information regarding the project & also for their support in completing the project.

We would like to express my gratitude towards members of “Endeavour Engineering” for their kind co-operation and encouragement which helped us in completion of this project. We would like to express our special gratitude and thanks to industry persons for giving me such attention and time.

We would specifically like to thank “Prof. Hardik Prajapati” for his continuous support and guidance over the course of this project. We would also like to thank “Prof. R. N. Mutagi” for providing his valuable suggestions on the transmission of data and video. We would like to thank “Prof. Subhash Patel” for providing suggestion regarding power supply circuitry. We would also like to thank “Tom Carden” for providing Modest Maps Library, which helped us create our GUI for Maps.

Finally, an honorable mention goes to our families and friends for their understandings and supports on us in completing this project. Without helps of the particular that mentioned above, we would have faced many difficulties while doing this project.

- Ashutosh Bhatt

- Priyal Joshi

- Nikunj Jadawala

- Maharsh Shah

Abstract

Seas and Oceans are vital to world stability and economy. Maritime activities, such as fishing, Oil & Gas extraction and tourism provide a major source of income for coastal nations and are the source of livelihood for millions of people. Littoral as well as blue sea waters are fragile and susceptible to external events. This stability requires a permanent attendance at sea for research, survey and real-time situation awareness, which is provided by perimeter patrolling.

Perimeter patrol enhances the utility of unmanned marine vehicles (UMVs) by enabling many security and scientific missions, including harbor protection, water sampling, and geological survey. We present a novel approach to perimeter patrol that uses two sensors: wireless embedded camera for visual monitoring and the heading information from a GPS. Both these sensors will be mounted on the UMV and will be used for real-time perimeter patrolling.

With its versatility and persistence the Unmanned Marine Vehicle (UMV) will provide key contributions in sea operations by improving effectiveness of tasks like:

· Blue and coastal waters Reconnaissance and surveillance,

· Homeland security, illegal activity awareness, search and rescue,

· Littoral and inshore hydrographic operations,

· Environmental resources protection and survey,

· Industrial assets and value shipping protection,

· Polluted area detection, localization and inspection,

· Fishery support and regulation,

· Scientific monitoring and research.

LIST OF TABLES

LIST OF FIGURES

TABLE OF CONTENTS

Chapter 1

Introduction

1.1 Project Overview

The last decade has witnessed a tremendous progress in the development of marine technologies that provide scientists with advanced equipment and methods for ocean exploration and exploitation. Recent advances in marine robotics, sensors, computers, communications and information systems are being applied to develop sophisticated technologies that will lead to safer, faster and far more efficient ways of exploring the ocean frontier, especially in hazardous conditions. As part of this trend, there has been a surge of interest worldwide in the development of autonomous marine robots capable of roaming the oceans freely, collecting data at the surface of the ocean and underwater on an unprecedented scale. Representative examples are autonomous surface craft (ASC) and autonomous underwater vehicles (AUVs). The mission scenarios envisioned a call for the control of single or multiple AUVs acting in cooperation to execute challenging tasks without close supervision of human operators. Furthermore, it should be possible for users who are not necessarily familiar with the technical details of marine robot development to do mission programming and mission execution tasks. Thus there is a need to push the development of methods for reliable vehicle and mission control of single and multiple autonomous marine robots.

1.2 Motive

The idea behind selecting this project was to improve the security of littoral as well as blue sea water. It has been recently implemented by many developed countries like USA, Czech Republic etc.

Numerous marine emergency and military applications are of time sensitive nature and require rapid response. Often such marine events, such as perimeter patrolling is dangerous

due to foreign attacks. Therefore in such maritime disasters, current methodology is often insufficient because the human responder cannot be jeopardized by being placed in potentially lethal conditions which could result in the loss of life. For example, a human responder may be put in danger due to rough seas, high winds, fire, toxic fumes, poor visibility, or hostile weapons fire in military operations.

Current response equipment is often insufficient to meet the critical time requirements to effectively deal with such emergencies. Often the distance from the response equipment, weather conditions, or other dangerous conditions hinder, and sometimes prevent, response efforts.

Therefore, there is a continuing unaddressed need for a marine vehicle capable of modular adaptability for perimeter patrolling, emergency, and military applications.

1.3 Innovation

The uniqueness about this model is that it will be a remote controlled marine vehicle that can be used in littoral areas everywhere for effective patrolling and increase in security, and also to reduce lethal accidents of coastal guards.

1.4 Requirements

During our background research, we came across many restrictions and limitations related to this project. This is a result of the nature of the project; since the project requires certain components which are not commercially available in India and they are required to be ordered from certain foreign countries.

1.4.1 Mechanical Requirements

1) Adaptation: it should be based on a versatile platform, where parts can be added and modified at any time.

2) Should be able to withstand use in both estuaries and ocean. Therefore resist water corrosion etc.

3) Payload of maximum 5 kg.

4) Length: The overall length should run from a minimum of 50 cm to a maximum of 100 cm.

5) Minimum turning radius.

6) Reversible thrust. Motors should be able to operate in both directions

7) Pan/Tilt Mechanism: The mechanism must provide a free movement of camera so that full 360° video coverage can be made possible

1.4.2 Electronics Requirements

1) Battery life: The UMV is expected to have a battery life of 6 hours of use. During these 6 hours most of the time will be spent for data transmission rather than moving.

2) Use of an Arduino board.

3) Radio range should be greater than 1 km over water bodies. Due to the fact that water bodies tend to decrease the effective distance of radio module.

4) Data transfer rate for the radio should exceed 1Hz. Therefore send information packets back to the station at a rate of at least one packet per second.

5) Camera must have pins so that it can be interfaced with Arduino Board.

6) GPS unit should update the position at a rate of at least 1Hz.

1.4.3 Software Requirements

1) Well-documented code structure

2) Thoroughly tested and functional code

1.4.4 General Requirements

1) Cost: UMV should cost less than Rs. 35000. That includes all of the materials including structure, motors and electronics. Construction hours are not calculated for the moment.

2) Fast set up time: The UMV should take less than 5 minutes to be set up and ready for deployment, by an experienced user.

3) Waterproofing for over-turning: Even if the UMV overturns, electrical components should stay secure and dry.

Chapter 2

Theoretical Background

2.1 Brief History

World War II saw the first experimentation with USVs. Canadians developed the COMOX torpedo concept in 1944 as a pre-Normandy invasion USV designed to lay smoke during the invasion - as a substitute for aircraft. COMOX was designated a torpedo because it could only be programmed to traverse a fixed course. Meanwhile, the US Navy developed and demonstrated several types of "Demolition Rocket Craft" intended for mine and obstacle clearance in the surf one. Unmanned operation was part of the concept, although it is unknown which, if any, of these vehicles were demonstrated as USVs.

Post-war applications of USVs expanded, with the USN using drone boats to collect radioactive water samples after atomic bomb blasts Able and Baker on Bikini Atoll in 1946. The 1950s-era US Navy Mine Defense Laboratory's project DRONE constructed and tested a remotely operated minesweeping boat in 1954. By the 1960s, the Navy was using target drone boats based on remote-controlled "aviation rescue" boats for missile firing practice, and the Ryan Fire fish target drone boat was used for destroyer gunnery training.

Interest in USVs as minesweeping drones and for other dangerous missions continued to grow after the 1950s for obvious reasons, and further US Navy development included the small "Drone Boat" – a 15ft USV for unmanned munitions deployment - that was quickly developed and deployed to the fleet in ten vehicle kits in 1965 during the Vietnam War. Larger Minesweeping Drone (MSD) USVs were also developed and deployed in Vietnam in the late 1960s.

Navy interests in USVs for reconnaissance and surveillance missions emerged in the late 1990s, with the development of the Autonomous Search and Hydrographic Vehicle (ASH),

later called the Owl, and the Roboski. The Roboski, initially developed as Shipboard Deployed Surface Target (SDST) as a jet-ski type target for ship self-defense training, now also serves as a reconnaissance vehicle test-bed. By the early 2000s, several concepts for stealthy USV sensor platforms have been proposed and are under consideration by the surface fleet.

Navtec Inc developed in the late 1990s a USV for the Office of Naval Research (ONR) under the

name Owl MK II. The Owl is a Jet Ski chassis equipped with a modified low-profile hull for increased stealth and payload capability. One version with side scan sonar and a video camera has been in operational use in the Persian Gulf. Science Application International Corporation (SAIC) offer also a small USV for port security. This Unmanned Harbor Security Vehicle (UHSV) is an advanced version of the Owl MK II.

Fig 1.1 Owl MK II, 1998

Fig 1.2 Roboski

Fig 1.3 Remotely Piloted Surface Vehicle (RPSV)

Fig 1.4 Roboraider

Besides the USA, several other countries employ and develop USVs. In Japan, Yamaha developed two USVs, the Unmanned Marine Vehicle High-Speed UMV-H and the Unmanned Marine Vehicle Ocean type UMV-O, *Enderle et al. (2004).* The UMV-H is a deep-V mass-produced hull, equipped with 90 kW to go 40 knots using water-jet propulsion. The boat can be used either manned or unmanned. At a length of 4.44 m, the craft is small enough to be loaded on a small cutter, but large enough to accommodate all necessary equipment and instruments such as under-water cameras (ROV) and sonar equipment. The UMV-O is an ocean-going USV with displacement hull. It is used primarily in applications involving monitoring of bio-geo-chemical, physical parameters of the oceans and atmosphere that put the long-distance capabilities of the vehicle to effective use. The first UMV-O "Kan-Chan" was delivered in 2003 to the Japan Science and Technology Agency.

Fig 1.5 Yamaha UMV-O

Fig 1.6 Canadian Barracuda

2.2 Block Diagram

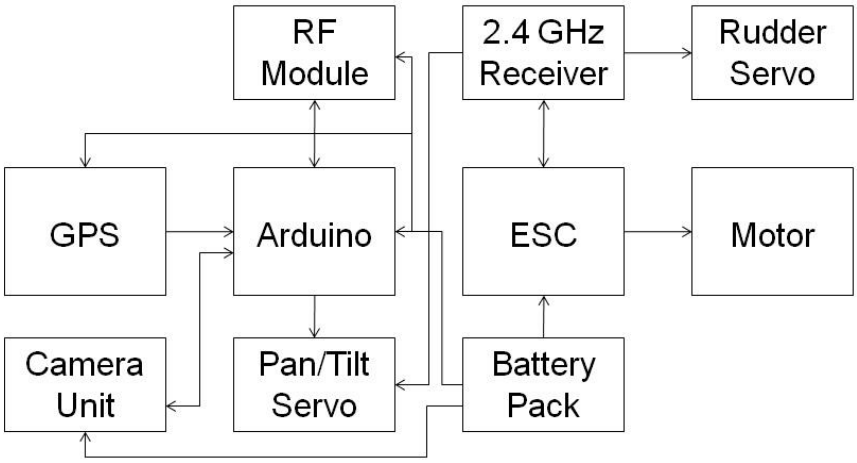


Fig 2.1 Block Diagram of UMV

The UMV contains various components which need to be controlled via micro-controller or user. Arduino is attached with GPS from which it will receive GPS co-ordinates and will transfer this data to Control Station using RF Module. 2.4GHz Receiver is connected to Rudder Servo and ESC thus controlling the maneuvering of boat by RF Remote control. It is also connected with Pan/Tilt Servos which will provide a 360° rotation to Camera. Battery Pack contains two different type of batteries that will power the system.

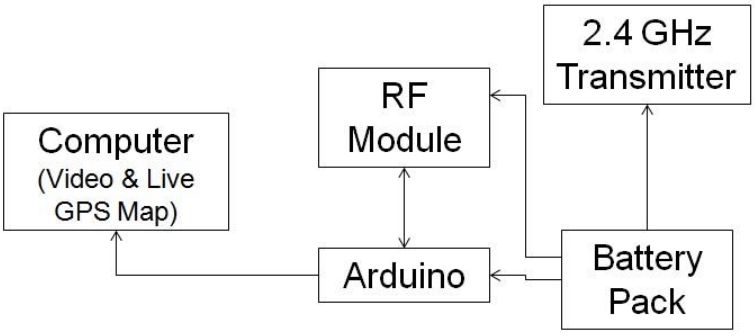


Fig 2.2 Block Diagram (Control Room)

On the other end at the Control Station, RF Module will receive the data containing GPS co-ordinates which will be passed to Computers Serial Port. This serial incoming data will be used by a Processing program which will process this data to display it in the programmed Graphic User Interface (GUI). User will use a 2.4 GHz transmitter that will help him/her to send commands to receiver and control the maneuvering of Boat.

Chapter 3

Mechanical Systems

The following sections describe the various aspects of the mechanical design, including the design itself, the materials selection, and the testing processes.

3.1 Design Concept

UMV is designed for the Surveillance purpose, so it should be of such a shape and color that enemy ships could not easily figure out that there's a small boat that is moving around them and spying them. So the most important part of design is to provide it stealth like shape, so that it can easily camouflage in Ocean. For such type of design it is very important to construct each and every part of boat with careful and proper considerations. The first step in boat design is to finalize a hull which is the watertight body of a ship or boat. Then by proper calculation of payloads it'll carry the height of waterline is decided (the line where the hull meets the water surface is called the waterline.) The structure of the hull varies depending on the vessel type. Since UMV's is developed for reconnaissance purpose, the area of the hull above the waterline needs to be as much small as possible so that it cannot be easily pointed out by the enemy ships.

During the course of the project, the team went through many design iterations before arriving on a final chassis design. The construction of boat structure is explained in the next topic.

3.2 Construction

3.2.1 Getting Started

Scratch building a model ship is not as difficult as it appears. The only additional skills required are interpreting the lines on the plans, selecting the materials to use in

Fig 3.1 Getting Started

the building process, and developing the method of construction.

Firstly the hull type is finalized and the required design specifications are found out and been researched on. It takes much more time to build from scratch than it does to build a kit, so we needed to be prepared to devote at least 500 hours to a scratch-built project, probably much more. We find our self spending much time developing new skills that were required for the entire construction of the vehicle. We made the decisions as to what wood or other material to use for a certain part on the boat.

3.2.1.1 Choosing the Plans

If we attempted a more difficult boat on our first try, it may be beyond our skills and lead to eventual frustration. However, we choose one within our own abilities, and are more than happy with the results. There are many sources of model boat plans. In choosing a set of plans for scratch build, we made sure they contain, as a minimum, three required views: a body plan, a frame structure, and a testing sustainability plan. Without these, it is not possible to build an accurate model.

Another consideration in choosing plans is the scale, which is the ratio of the size of the model to the size of the real ship. This is mainly a practical consideration. A typical ship-of-the-line was nearly 300 feet long in real life, which would make the finished model 37” on a scale of 1/8” = 1’. The same model in a scale of ¼” = 1’ would be nearly 75” long. Any plans could be purchased and could be enlarged or decreased in size. The scale of the model also determines how much detail will be included in the final model. The larger the scale is, the greater the detail that can be included.

3.2.1.2 Selecting the Materials

In order to build a model from scratch, we would need a wide variety of materials. The list seems daunting, but if we’ve already built kits, we probably would have used many of the materials listed. The materials discussed below will represent the basic requirements for the beginning scratch builder.

Wood: Many modelers use many different kinds of wood in building a single model. For us it was best to rely on the easiest kinds of wood to use and then expand later.

Basswood: By far, basswood is the most versatile wood for model builders. It is light, almost white, in color. It is very fine-grained, making it easy to saw, sand, drill, carve, stain, and paint. It also bends easily, which makes it ideal for planking curved bows on ships. It can be turned on a lathe as long as it's an eighth of an inch round or larger. After a sanding sealer is applied, it can be sanded to a smooth finish that eliminates the fuzziness associated with unsealed basswood. It can be made to simulate many other kinds of wood by the application of stains. Its only drawback is its softness.

Foam Sheet: This is a versatile hard sheet, whitish in color. It can be used for making small frame structures and provide bendability along with ductility. Paper-thin pieces can be sliced off on a band saw. It is especially useful in making parts that will have a great deal of stress, such as mast caps, because it does not split or break easily.

Glue: There are dozens of different glues on the market, but there are two kinds that are indispensable to the scratch builder. The first is white glue, such as Elmer's, which is ideal for basswood, a highly porous wood. For other types of wood it would probably be better to use epoxy. The second is cyano-acrylic glue, commonly called CA or super-glue, which provides an indestructible bond in a matter of seconds. It can glue any type of material to anything else.

Metal: We used copper and/or brass metals for various fittings, such as chain plates. Fine black metal wires were used for making eyes and stropping blocks. Sometimes, we used paper clips or staples or pieces of house wiring, or even the insulation around it.

Tape: There are several varieties of tape in the market, including self-adhesive, double-sided, and specialty 3 tapes. It's used for painting waterlines or any straight-line area. Sometimes, you might use it as a clamp.

Paper and cardboard: We used paper and cardboard in myriads of places, including window frames, anchor stock rings, paneling, edgings, etc. Bristol board, index cards, the backs of note tablets, typewriter paper etc.

Other material: The heads of pins were just the right size for rivets. Round-head pins made the rivets used for stiff flexibility purpose. Simple red and green stickers could be used to mark up the direction on the boat.

3.2.1.3 Developing a Method of Construction

We developed our own structural design for our specific purpose. The first decision we made was what type of hull should be constructed. There are several styles of hull construction worthy of consideration, including half-hull models, waterlines models, and whole hull models. This narrative will address only the whole hull models, of which there are three types:

Solid: This type of hull is carved from a solid piece of wood or from built-up laminations. It's usually started by cutting out the basic shape on a band saw, and is then finished with chisels and spoke shaves. A series of templates are made and used to check the shape of the hull at various points while the carving is in progress.

Plank-on-bulkhead: This type of hull is most familiar to kit builders. It consists of a length of wood that represents the backbone of the ship. It usually runs from stem to stern, with the bottom edge representing the keel and the top edge representing the upper decks. It is notched at regular intervals so that it can receive correspondingly notched bulkheads. After the bulkheads have been installed, bow and stern blocks are glued in place so that planking can be accommodated. Then the planking is installed directly onto the bulkhead edges. The plank-on-bulkhead hull, when completely planked, looks exactly like the real boat in every way. Only the unseen interior is not built like the real boat.

Plank-on-frame: Consisting of laying a keel, building individual frames, and planking the hull -- all exactly like the original boat. The plank-on-frame hull is like the real boat. Once we decided on what type of hull to construct, we studied the plans intensely having

understood the plans. Even after the hull is built, the next steps and procedures and Plan ahead were decided. Thinking through every step of the project, one at a time we made sure that we don't get in a position where a certain item cannot be installed because we've boxed items in a corner and can't get at that location.

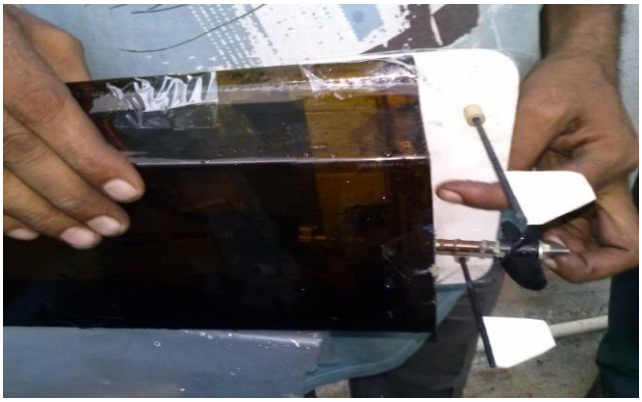


Fig 3.2 Prototype Boat Construction

Fig 3.3 Propeller with shaft

3.2.2 Interfacing The Plans

The Three Plans:

Body Plan: The body plan, sometimes called section plan, is divided in half and shows two separate views of the ship. The left-hand side of the plan represents the view from the stern of the ship looking forward. The right hand side of the plan is just the opposite, from the bow looking toward the stern. Note that the lines on the plans are comprised of four types: waterlines, section lines, buttock lines, and diagonal lines, which will be explained shortly.

Sheer Plan: The sheer plan, sometimes called the elevation plan, is a view of the ship from its side. It, too, contains waterlines, section lines, buttock lines, and diagonal lines. It also contains many other pieces of information, including the location of gun ports, wales, casts, chain plates, and deadeyes.

Half Breadth Plan: The half breadth plan, sometimes called simply the plan view, is a view of only half the ship looking from the top down. Only one half is necessary because the other half will be a mirror image of the former. Like the other two plans, the half breadth plan also contains waterlines, section lines, buttock lines, and diagonal lines.

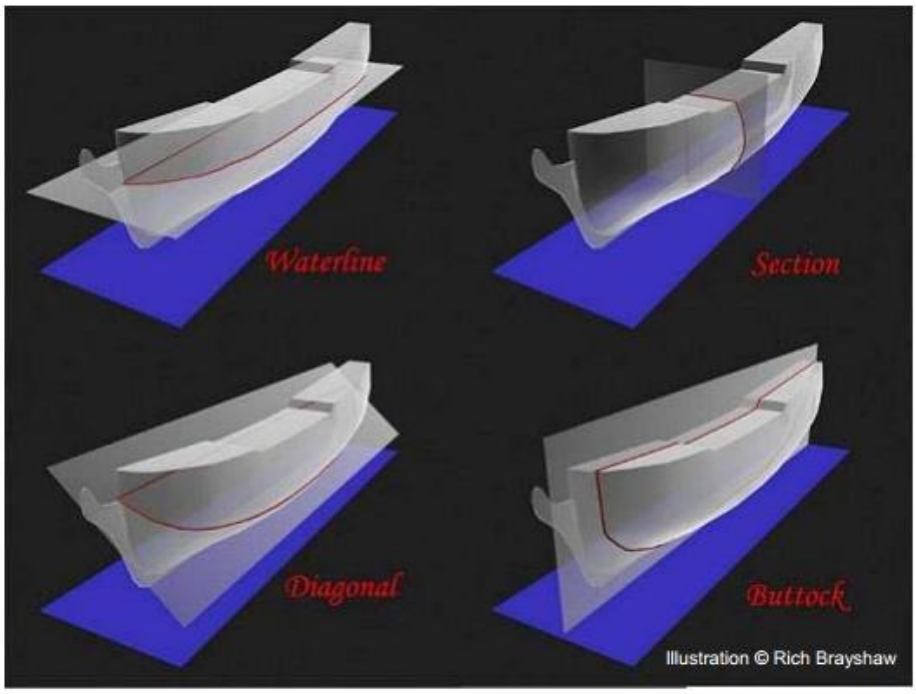


Fig 3.4 The Four Main Kinds Of Lines

The Four Main Kinds of Lines:

Waterlines: The waterlines are horizontal lines that pass through the hull at each area shown. It's as if you have placed a ship model in water at successively lower and lower lines. On many plans these lines are numbered from the keel upward starting with 1, with every plan showing the same number for each corresponding waterline. As you can visualize, the waterline near the midsection of the hull will be wider and slightly longer than the waterline below it.

Section Lines: These lines pass perpendicularly in a vertical plane through the hull. They are very important lines because they define the shape of the hull more graphically than other lines. These are particularly important for building frames for a plank-on-frame model. Of course, not every section line may be the exact placement for a frame, and precise locations

of additional frames must be determined. Usually, section lines on plans start with a centerline somewhere near the midpoint of the ship. Moving away from the centerline toward the bow, they are identified with letters A, B, C, and so forth. From the centerline toward the stern, they are identified numerically. Again, the body plan, sheer plan, and half breadth plan will all have the same identifying letters and numbers.

Buttock Lines: The buttock lines are lines that pass through the hull in a position that is parallel to the centerline. The sheer plan shows their true shapes. Note that on the body plan the buttock lines appear as vertical straight lines, and on the half breadth plan they appear as horizontal straight lines. Although the buttock lines are rarely used in constructing the model itself, they are quite useful in verifying section shapes.

Diagonal Lines: These lines pass through the hull at an angle to the vertical plane of the centerline. They're usually not used in the actual construction process, but they serve the purpose of checking the accuracy of other lines.

Baseline: This line, which appears on the body plan and the sheer plan, is usually shown as a dotted line drawn just above the keel, as shown on the Hassan Bashaw plans. On this ship, the baseline is parallel to the keel, but this is not always true.

Perpendiculars: These lines also appear on the body plan and the sheer plan. They are drawn at both ends of the baseline, frequently at the ends of the rabbet lines. These lines are always perpendicular to the baseline, which means they are not necessarily perpendicular to the keel**.**

3.2.3 Modifying plans to suit the needs of on-frame construction

3.2.3.1 Drawing the Frame Lines Directly on the Plans

We were planning to build a plank-on-frame model, but our plans probably do not show the location of all the frames, their width, or the spacing between the frames. Of course, there are plans available that are drawn specifically for plank-on-frame construction in mind, but most plans are not. Therefore, we needed to determine these measurements ourself.

If we decide to build a model on a scale of ¼” = 1’, it is usually best to make all frames ¼” wide, which means they would be one foot wide in real life. A smaller-scale model would require thinner frames. In most warships, the spacing between each frame was equal to the width of the frame itself. This is called room-and-space. Thus, a ¼” frame would have a ¼” space next to it, followed by another ¼” frame, and so forth. On the other hand, a merchant vessel did not require such close spacing between frames, mainly because they were not expected to be damaged by cannonball fire.

Fig 3.5 Horizontal Wooden Planes

Using the room-and-space rule, measure and draw the preliminary frame lines, starting with the centerline already shown on the plans, directly onto your plans, including both the sheer plan and the half breadth plan, which are usually directly above one another. Proceed from the centerline forward, and then stop at the point where the cant frames will begin. Then proceed from the centerline to the aftermost position where cant frames will start.

(Note that it is sometimes not necessary to build cant frames at the stern, especially if the stern does not have extreme curvature. Thus, if we decided no frames were necessary at the stern, we could continue drawing parallel lines all the way to the stern. We could not draw lines beyond the cant frame positions, because cant frames are not at right angles to the centerline. Due to the curvature of the bow, and perhaps the stern, cant frames are set at an angle so that the planking can rest on a consistent angle to the frames.

Another consideration for the placement of the frames is the location of ports. Most of the preliminary frames should lie adjacent to the port openings, not directly within those openings. If too many seem to lie within the gun port openings, it is best to adjust the frame locations starting at a slightly different point, but still using the room-and-space rule. We had access to the actual plans of the vessel that showed the frame locations, many of your own determinations will be purely arbitrary. Some of the lines we drew far would correspond with some of the section lines already given on the plans, but most will not. Since we would end up with a series of parallel lines exactly ¼" apart from each other, it is difficult to see which are the frames and which are the spaces, so, using a colored pencil, lightly shade in the 8 frames and leaving the spaces plain. We made sure that all of this was done with precision, because the accuracy of the finished model depends on the drawings we make now.

Fig 3.6 Base Frame View 1

Fig 3.7 Base Frame View 2

3.2.3.2 Creating a Master Grid

Now we are ready to draw the shapes of individual frames, which will be used as patterns for the actual construction of each frame. First, it is necessary to draw a master grid that will be used for every one of the frames. Using an ordinary sheet of typewriter paper, we drew a horizontal centerline from top to bottom. About an inch from the bottom of the page, a line perfectly perpendicular to the centerline – this line represents the lowest waterline, which is immediately above the keel. Now draw the remaining waterlines, as shown on the plans, and then marked them 1, 2, 3, etc., on the grid just as they are shown on the plans. Now drawing the buttock lines, which are perpendicular to the waterlines, again measuring from the plans.

The buttock lines are customarily marked to the left of the centerline as A, B, C, etc., and similarly to the right of the centerline.

Now measuring the width of the keel and also the length it extends below the base waterline. We drew our findings on your master grid. When we finished drawing all the required lines, the grid should look similar to the photo Fig 3.6. Now counting the total number of frames we would be building on our model, and then making at least that number of copies of our master grid, because we would need one grid as a pattern to cut out each of our frames. A word of caution on the use of copiers: Some copiers do not reproduce exact copies; they make slightly larger or slightly smaller copies than the original. Even a tiny variance cannot be tolerated, so we tested out the copier you are using before making all those copies.

3.2.3.3 Lofting the Outer Frame Lines on the Grid

Next we must make a pattern of each frame by drawing the shape of each frame on a separate grid. This is a tedious and time-consuming task, but it was necessary for the successful construction of a plank-on-frame model. Several of the frames shapes would already be known from the plans, but on one side only we must determine and draw the shapes of both sides of each frame. Of course, those near the center of the hull will be exactly or nearly exactly the same on both sides because there is little or no curvature in the shape of the hull at these points. However, the further we went after, the more the variance would be seen because the curvature of the hull changes more and more. These points would become more apparent as we drew the patterns. Note that the following procedure would describe how to draw frame on the grid. It is only necessary to draw the frame on one side of the centerline of your grid. This will be true for every grid you draw. The other half will take care of itself.

Fig 3.8 Frame Construction

Fig 3.9 Applying Mechanical Tensile Force

3.2.3.4 Finishing the Grids

We had completed only half of the drawings for all the required frame grids. The best way to complete the other half is to use the following procedure, which ensures that all of our frames will be perfectly symmetrical. Finish one frame grid at a time.

1. Using a steel ruler and sharp-pointed X-Acto knife, gently score a line directly down the centerline of the grid. This scoring line must be precise, because the goal is to make both halves perfectly symmetrical. Do not cut all the way through the grid, only enough to be able to fold it in half easily.

2. Crease the grid carefully down this scored line, with the drawn lines on the inside of the fold.

3. Hold the grid over a light source and one could see the frame lines through the paper. Taking a pencil, trace heavily over the frame lines one could see. We made sure we traced over all of them.

4. Open the grid and one can see faint lines on the other half of the grid.

5. Now fill in these faint lines with a pencil to make them as visible as the other half. We repeated this procedure for each of our grids.

When we have finished all of them, we were ready to use them as patterns to construct our frame.

3.2.4 COMPLETING THE HULL:

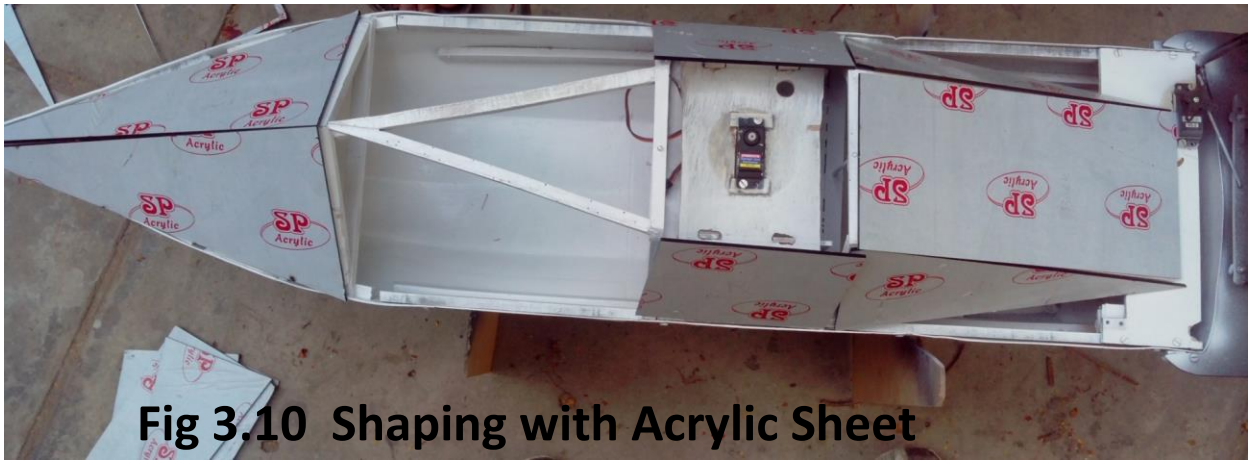


Fig 3.10 Shaping with Acrylic Sheet

3.3 Propulsion and Rudder

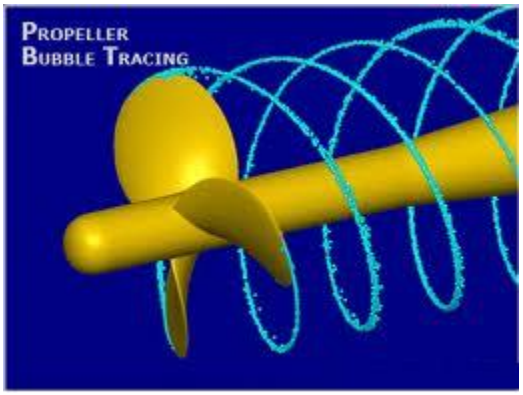


Fig 3.11 Propeller

The UMV navigates using a network of a motor driven propeller, and two directional rudders. The motor used to drive the propeller is a special kind of brushless motor which provides minimum resistance during the operation and provides faster motor revolutions. The motor is fixed inside the vessel and is connected to propeller using a shaft. The shaft is made up of steel rod and special grooves are made such that it can be directly attached to motor shaft on one side and propeller on another. The shaft is fixed at a declined angle such that the propeller always resides inside the water. The motor is having bidirectional rotation which will provide two way propeller rotations. This configuration provides a key benefit i.e. the ability to precisely

hold a position (station keeping), allows for accurate surveying of an area. Both two rudders are centrally connected thus allowing equal degree of motion to both the rudders at the same time. The rudder mechanism is controlled using a servo motor. Both brushless motor and servo motor are attached to receiver for ease of controlling and maneuvering operation of UMV.

Fig 3.12 Rudder

3.4 Camera Pan / Tilt Mechanism

The Pan/Tilt Mechanism employs two servo motors, each having a DOF (Degree Of Freedom) of 180°. These servos are connected to receiver. The mechanism provides two axis rotations to the camera: 1) Yaw Rotation and 2) Pitch Rotation. Yaw rotation will help the camera to perform a pan movement and Pitch rotation will help it to perform a tilt movement. Both servos Both servos together will provide a 360°

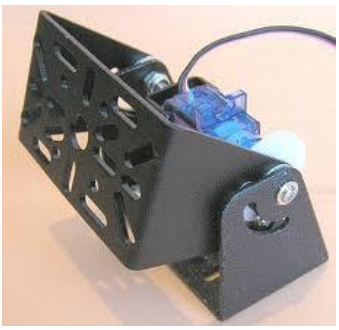


Fig 3.13 Camera Pan/Tilt Mechanism

Chapter 4

Electronic Systems

4.1 Brushless Motor

A brushless DC motor (BLDC) is a synchronous electric motor which is powered by direct-current electricity (DC) and which has an electronically controlled commutation system, instead of a mechanical commutation system based on brushes. In such motors, current and torque, voltage and rpm are linearly related.

In a conventional (brushed) DC motor, the brushes make mechanical contact with a set of electrical contacts on the rotor (called the commutator), forming an electrical circuit between the DC electrical source and the armature coil-windings. As the armature rotates on axis, the stationary brushes come into contact with different sections of the rotating commutator. The commutator and brush system form a set of electrical switches, each firing in sequence, such that electrical-power always flows through the armature coil closest to the stationary stator. In a BLDC motor, the electromagnets do not move; instead, the permanent magnets rotate and the armature remains static. This gets around the problem of how to transfer current to a moving armature. In order to do this the brush-system/commutator assembly is replaced by an electronic controller. The controller performs the same power distribution found in a brushed DC motor, but using a solid-state circuit rather than a commutator/brush system.

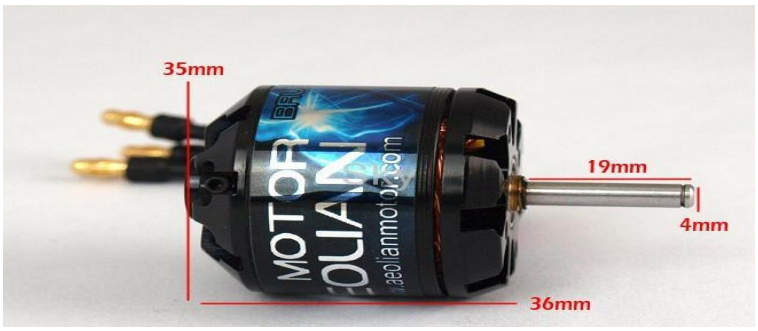


Fig 4.1 Brushless DC Motor

4.2 Electronic Speed Controller

An **electronic speed control** or **ESC** is an electronic circuit with the purpose to vary an electric motor's speed, its direction and possibly also to act as a dynamic brake. ESCs are often used on electrically powered radio controlled models, with the variety most often used for brushless motors essentially providing an electronically-generated three phase electric power low voltage source of energy for the motor.

An ESC can be a stand-alone unit which plugs into the receiver's throttle control channel or incorporated into the receiver itself, as is the case in most toy-grade R/C vehicles. Regardless of the type used, an ESC interprets control information not as mechanical motion as would be the case of a servo, but rather in a way that varies the switching rate of a network of field effect transistors, or FETs. The rapid switching of the transistors is what causes the motor itself to emit its characteristic high-pitched whine, especially noticeable at lower speeds. It also allows much smoother and more precise variation of motor speed in a far more efficient manner than the mechanical type with a resistive coil and moving arm once in common use.

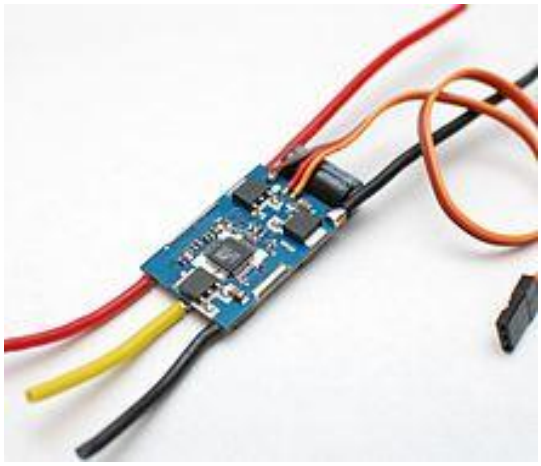


Fig 4.2 ESC with BEC

Most modern ESCs incorporate a battery eliminator circuit (or BEC) to regulate voltage for the receiver, removing the need for receiver batteries. BECs are usually either linear or switched mode voltage regulators.

DC ESCs in the broader sense are PWM controllers for electric motors. The ESC generally accepts a nominal 50 Hz PWM servo input signal whose pulse width varies from 1 ms to 2 ms. When supplied with a 1 ms width pulse at 50 Hz, the ESC responds by turning off the DC motor attached to its output. A 1.5 ms pulse-width input signal results in a 50% duty cycle output signal that drives the motor at approximately half-speed. When presented with 2.0 ms input signal, the motor runs at full speed due to the 100% duty cycle (on constantly) output.

Brushless ESC systems basically drive tri-phase Brushless motors by sending sequence of signals for rotation. Brushless motors otherwise called outrunners or inrunners have become very popular with radio controlled airplane hobbyists because of their efficiency, power, longevity and light weight in comparison to traditional brushed motors. However, brushless DC motor controllers are much more complicated than brushed motor controllers.

The correct phase varies with the motor rotation, which is to be taken into account by the ESC: Usually, back EMF from the motor is used to detect this rotation, but variations exist that use magnetic (Hall Effect) or optical detectors. Computer-programmable speed controls generally have a user-specified option which allows setting low voltage cut-off limits, timing, acceleration, braking and direction of rotation. Reversing the motor's direction may also be accomplished by switching any two of the three leads from the ESC to the motor.

4.3 Power Source

Two types of batteries have been used as power source which are as follows:

4.3.1 Lithium Polymer

The electrical system of the UMV begins with the choice of a power source. The UMV is powered by Lithium Polymer (Li-Po) batteries with an operating voltage of 11.1 volts. It has a continuous discharge rate of 25 C and has a capacity of 1350 mAh thereby providing an average running time of 2-3 hours. Li-Poly battery is selected due to its high power density, low weight, high draw rating, and charge speed. There are two power systems which are independent and segregated by purpose. One battery powers the propulsion drive and the rudder servos, while the other powers the controller and the sensors. This segregation prevents power draw spikes (by the motors) from affecting the controller and sensors, which are typically sensitive to voltage fluctuations.

4.3.2 Lithium Ion

The Video transmitter, receiver and camera are powered by Lithium Ion (Li-Ion) battery with an operating voltage of 11.1 volts. It has continuous discharge rate of 2C and has a capacity of 1500mAh thereby providing running time of 10-12 hours.

4.4 The Pilot of the System: FlySky CT6B RC Transmitter and Receiver

A Flysky transmitter and receiver are used to communicate with the UMV. Signals through specific channels are transmitted through the transmitter and thus received at the receiver end.

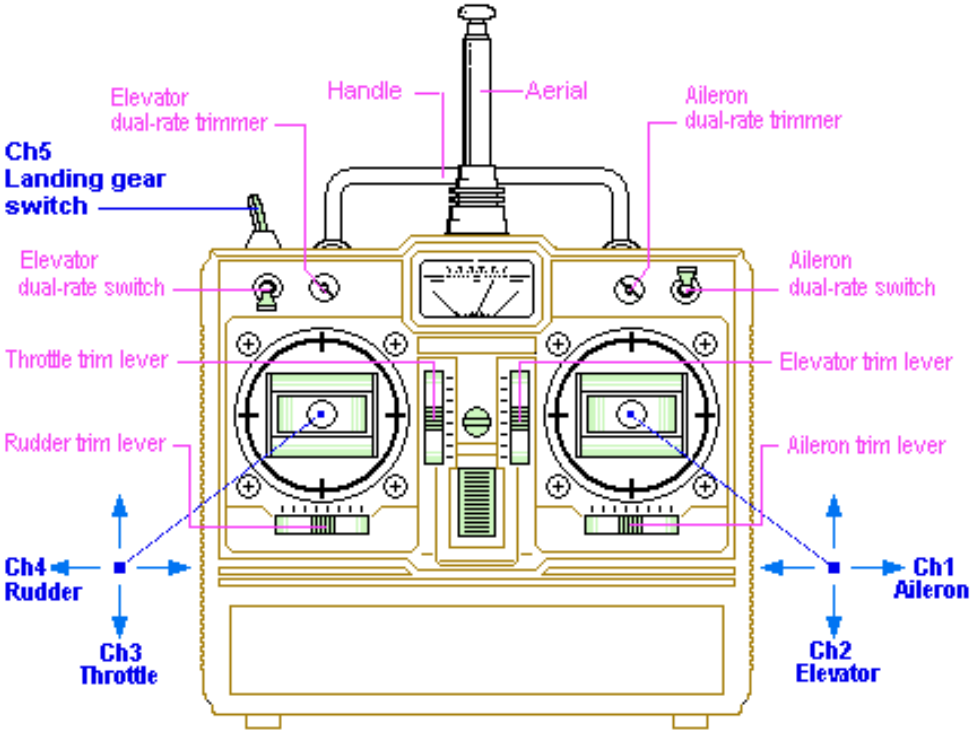


Fig 4.3 RC Transmitter

4.4.1 Transmitter Parameters:

Channels	:	6
Frequency Band	:	2.4GHz
Program Type	:	GFSK
Modulation Type	:	FM
RF Power	:	19 dB
Antenna Length	:	26mm

4.4.2 Receiver Parameters:

Channels	:	6
Frequency Band	:	2.4GHz
Program Type	:	GFSK
Modulation Type	:	FM
RF Receiver Sensitivity	:	-76 dB

4.4.3 Getting Started:

There are four sections to the main screen.

- At the top is a graphical signal chart that represents the six signals this radio provides.
- Next section is the Software options (System Option)
- Next section is the Signal configuration section (System Setting)
- Last section is the Switch configuration settings (Switch Program)

Each of these sections and individual options will be described in detail.

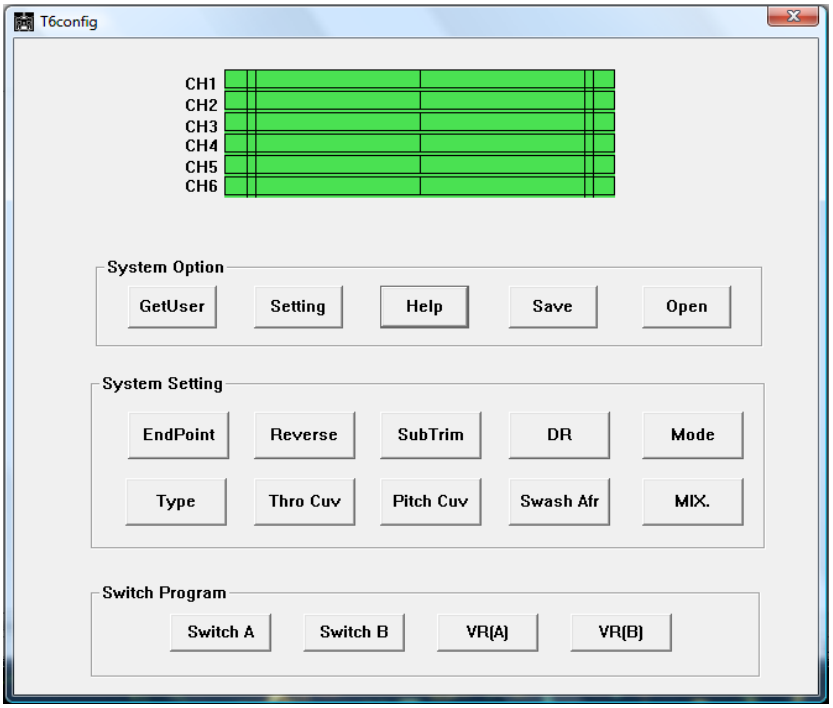


Fig 4.4 T6Config

4.4.4 Basic Setup:

1. **MODE** - Set the Mode to Model 2. In Mode 2, the Left stick controls the throttle and rudder, and the Right stick controls the swash plate through the Aileron, Pitch and Elevation servos. European (Mode 1) settings cause the right stick to control the throttle & rail rudder, and the left stick to control the swashplate.
2. **TYPE** – This setting tells the radio about your type of swash plate. The Servo linkages are connected to the swash plate at 120 degrees apart.
3. **ENDPOINT** – There are two columns of settings. The left column is the UP direction limitation setting, and the right column is the down limitation in percentage. They have a center, and can move 30 degrees Clockwise (CW) and 30 degrees Counter-Clockwise (CCW). At this stage in the setup, set all values to 100%. Typically one will not have to revisit this setting unless you have a mechanical interference or some other limiting factor that you need to prevent the servo from traveling too far.
4. **DR** – Dual Rate setup. There are ON and OFF columns. This setting is only relevant if your radio is later configured to utilize dual rates by associating this function with one of the switches. Since we recommend you have one of the switches set for Throttle Kill, and one will eventually probably want to use Idle-Up/3D mode with the other switch, there are no additional switches to provide this functionality. Dual rate can be thought of as selecting between two "sensitivities" of the rudder and cyclic controls. Reacting too abruptly to minor control movements can be difficult for a beginner or pilot who needs very precise control. Thus, these fields set what percentage of full-rate is desired in each switch position ("on" and "off" are again misleading misnomers, since this is really an "A" or "B" choice when either position could be the more sensitive one). At this stage in the setup, set all the fields of at least one selection to 100.
5. **SUBTRIM** – these settings are used to center the servos. At this stage in the setup, set them all to zero (0). You will revisit this option many times as you fine tune the vehicle.
6. **MIX** – The T6Config program and FS-CT6A model radio support 3 mixes. A "mix" is when one channel affects another automatically, or to associate a physical

transmitter control like a switch or dial to a certain radio channel. We will delve more into this option later in the setup, however for this initial setup stage, change the SWITCH setting for all three MIX settings to OFF. To do this, select the MIX 1 from the drop down list, and then select the OFF setting from the SWITCH drop down list. Now select the MIX 2 from the list, and set the SWITCH setting to OFF, and finally select the MIX 3 setting and set its SWITCH setting to OFF.

7. **THRO CUV** – The throttle curve setting has two modes. Normal (NOR) and Idle Up/3D mode (ID). At this stage in the setup, select the NOR option from the drop down list and set the values to a linear curve from 0 to 100 (0-25-50-75-100). We will return to this option later to tune the throttle to your motor and flying needs. While one is here, you can set your Idle Up throttle curve as shown below.

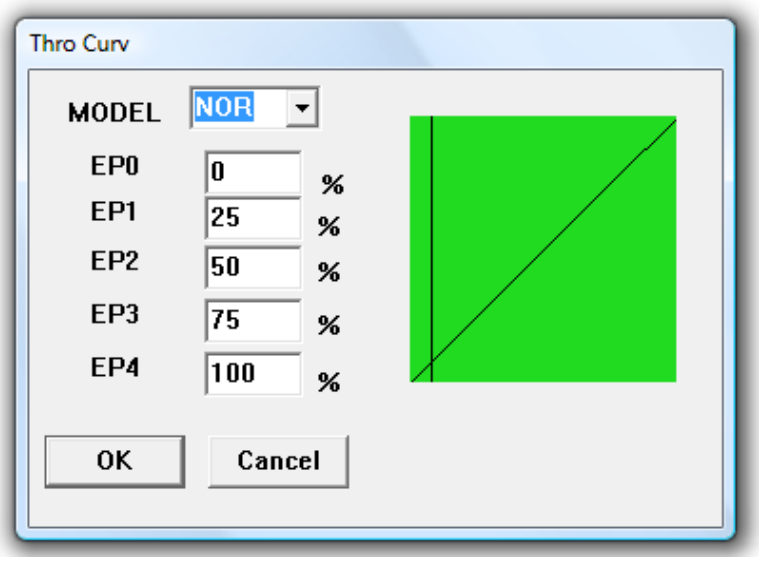


Fig 4.5 Throttle Curve (Normal)

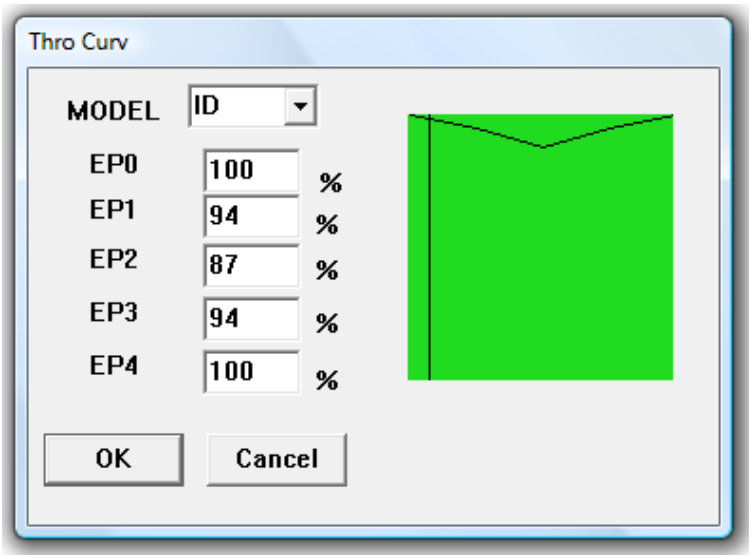


Fig 4.6 Throttle Curve (Idle)

9. **PITCH CUV** – Pitch Curve settings. There are two modes for pitch settings. Normal (NOR) and Idle Up/3D mode (ID). At this stage in the setup, select the ID option from the drop down list and set the values to a linear curve from 0 to 100.
10. **SWITCH A** – The A switch on this radio is the top right On/Off toggle switch. Set the switch to the THROCUT setting. When the switch is in the ON position, it will turn off the signal to the ESC (Electronic Sensor Control) that control the motor. The motor will not turn when this switch is on, providing another level of safety while working on the helicopter or positioning it setting. When the switch is in the ON position, it will turn off the signal to the ESC (Electronic Sensor Control) that control the motor. The motor will not turn when this switch is on, providing another level of safety while working on the helicopter or positioning it before flight. The ON position for this radio is the forward toggle position (towards the user).
11. **SWITCH B** – The B switch on this radio is the top left On/Off toggle switch. Set this switch to the NOR/ID setting. When the switch is in the ON position, it will use the ID throttle curve and ID pitch curve. In the OFF position it uses the NOR throttle curve and NOR pitch curve.

4.4.5 Binding the transmitter to the receiver:

1. Install the battery to the 2.4GHz transmitter, and turn it off.
2. Insert the binding plug into the BAT port of the receiver.
3. Connect the ESC to Channel 3, or an external battery (5vdc) to any one of the other channels observing the +/- polarity. Power up the receiver using the external battery, or connecting the main battery to the ESC. The LEDs should start to flash.
4. Press and hold the lower left button on the transmitter, and then switch on the transmitter's power switch.
5. Observe the LED lights on the receivers (main and satellite). Once the LEDs stop flashing, the receiver is bound to the transmitter. It will take about 10 seconds (or less) for the binding process to complete.
6. Release the match button on the transmitter. Remove power from the receiver, and turn off the transmitter.
7. Remove the receiver binding plug. Connect your servos and other channels as described in Fig 6.
8. Test by turning on the receiver without pressing the match/bind switch. Power the receiver, and the LEDs should light steady meaning it is bound to the transmitter.

Fig 4.7 Bind Button of RC Transmitter

4.4.6 Typical settings for a Boat:

The servo that will be controlled by the right stick for Left and Right directions should be plugged into channel 1..

Channel 2 is for the left stick (up & down), Channel 1 is on the right stick (Left & Right)

1. Set Mode to Model 1 (we used Mode 3* for our purpose)
2. Plug the servo that will control forward and backward into **channel 2**
3. Plug the servo that will control left and right into **channel 1**
4. Set the "Pitch cuv" [Pitch Curve] NOR to:

    1. EP0 = 0
    2. EP1 = 25
    3. EP3 = 50
    4. EP4 = 75
    5. EP5 = 100

1. EndPoint settings should be the default of 100% for everything.
2. Reverse, we were only concerned with channels 1 and 2. If a servo does not move in the direction we need, reverse that servo using this editor.
3. SubTrim adjusts the centering for each servo; again, we only need to adjust channels 1 and 2 to trim up those servos.
4. DR settings should be the default of 100% for everything.
5. Type setting to Acro
6. Thro Cuv does not come into play unless we also have a motor on our sail boat.
7. Mix, there are three mixes, each with a switch selection. Set all switch selections to OFF.

    *Mode 3- Left side is equipped with channel 1 and 2 while right side is equipped with channel 3 and 4. We used channel 1 and 2 for boa3 navigation and 3 and 4 for camera pan tilt mechanism.

4.5 Servo Motor:

Servo Motor is a type of motor which has a high torque and rotates to specific degrees as the pulse provided.

The servo has an electronic circuit that compares the incoming control pulse with a local generated one whose width corresponds to the servo arm's actual position.

The servo's internal pulse width is determined by its feedback potentiometer whose slider moves together with the servo's arm. When the width of the incoming control pulse is different from the local generated, the servo motor will rotate until the both pulses' width are equal.

The direction of rotation depends on whether the incoming pulse is wider or shorter than the local pulse.

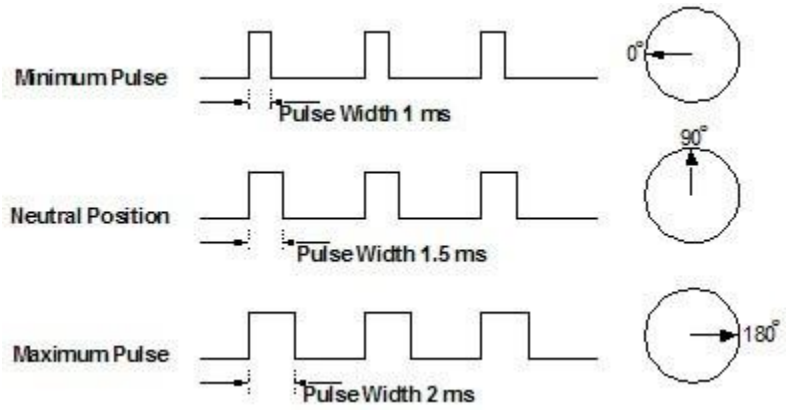


Fig 4.8 PWM of Servo Motor

Fig 4.9 Servo Motor

There are two operating concepts: the conventional servo and the digital servo. The conventional servo circuit uses a pulse stretcher to widen the pulse difference between the incoming pulse and the locally generated. Thereby a 1% pulse difference produces a 50% duty cycle for motor drive.

A continuous drive signal will be obtained when the pulse difference is over 10%. Also a small dead band is provided to prevent the servo being in continuous state of motion when significant pulse differences occur. The difference between the conventional and the digital servo is that the pulse drive to the motor occurs every 20mS with the conventional, whereas with the digital occurs (for example) every 3.3mS, which means that the digital servo sends pulses to the motor at a much higher frequency. Digital servo incorporates a microprocessor, which receives the input pulse signal and generates power pulses to the servomotor based on preset values. Some brands offer the possibility to program certain parameters such as Dead-Band Width, Direction of Rotation, Neutral Point, Servo Arm Throw and End Point. The digital servo is supposed to have constant torque throughout the servo travel, faster control response and more accurate positioning, but at the expense of greater power consumption.

4.6 The Heart of the System: Arduino Microcontroller

Arduino is an open-source single-board microcontroller, descendant of the open-source Wiring platform, designed to make the process of using electronics in multidisciplinary projects more accessible. The hardware consists of a simple open hardware design for the Arduino board with an Atmel AVR processor and on-board input/output support. The software consists of a standard programming language compiler and the boot loader that runs on the board.

Arduino hardware is programmed using a Wiring-based language (syntax and libraries), similar to C++ with some slight simplifications and modifications, and a Processing-based integrated development environment.

The Arduino microcontroller uses what is essentially C++, however it does not follow many of the normal conventions that actual C++ uses. The reason behind this change in convention has to do with who the Arduino is marketed to. The Arduino is designed to be easy to use by hobbyists that have minimal programming experience, allowing the user to quickly implement code to fit their project's needs without worrying about some of the more complex programming problems. The biggest example of this is the number of libraries that are included with the Arduino. Many library APIs like I2C, SPI, and many others are already

included, whereas if the user were to develop their project with another MCU, they may have to develop those libraries themselves. This becomes a matter of function over form, as it is simple to write “quick and dirty” code that runs without delving into the inner workings of the Arduino framework.

Arduino’s simplicity does not end in its code styling. The Arduino IDE is very minimalist, allowing users to get straight into coding without spending much time worrying about setting up the environment. This simplicity is not without its disadvantages. While the Arduino IDE allows for fast coding and setup, it lacks many of the extensive features that other IDEs have, such as Eclipse and Visual Studio. Things like a robust debugging environment built in revision control are just a few examples of features that the Arduino IDE is missing.

4.6.1 Arduino UNO Board

The Arduino Uno is a microcontroller board based on the ATmega328. 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 a 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.

"Uno" means one in Italian and is named to mark the upcoming release of Arduino 1.0. The Uno and version 1.0 will be the reference versions of Arduino, moving forward. The Uno is the latest in a series of USB Arduino boards.

4.6.2 Arduino Specifications

Microcontroller	ATmega328
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limits)	6-20V
Digital I/O Pins	14 (of which 6 provide PWM output)

Analog Input Pins	6
DC Current per I/O Pin	40 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	32 KB (ATmega328) of which 0.5 KB used by bootloader
SRAM	2 KB (ATmega328)
EEPROM	1 KB (ATmega328)
Clock Speed	16 MHz

4.6.3 Power

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

- **3V3.** A 3.3 volt supply generated by the on-board regulator. Maximum current draw is 50 mA.
- **GND.** Ground pins.
- **IOREF.** This pin on the Arduino board provides the voltage reference with which the microcontroller operates. A properly configured shield can read the IOREF pin voltage and select the appropriate power source or enable voltage translators on the outputs for working with the 5V or 3.3V.

4.6.4 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.

4.6.5 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 40mA and has an internal pull-up resistor (disconnected by default) of 20-50 kΩs. 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.
- **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 analogReference().
- **Reset.** Bring this line LOW to reset the microcontroller. Typically used to add a reset button to shields which block the one on the board.

4.6.6 Communication

The Arduino Uno has a number of facilities for communicating with a computer, another Arduino, 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 software includes a serial monitor which allows simple textual data to be sent to and from the Arduino 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 SoftwareSerial library allows for serial communication on any of the Uno's digital pins.

4.6.7 USB Overcurrent Protection

The Arduino 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.

4.6.8 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.

4.7 RF Module

Our project employs a pair of Xbee Modules for achieving the objective of Wireless Communication. With these modules a full duplex RF communication can be established. The detailed descriptions of these modules are as follows:

4.7.1 Xbee Transceiver

Fig 4.10 Xbee Module

XBee is a wireless communication module that Digi built to the 802.15.4/ZigBee standard. The ZigBee wireless standard can form self-healing mesh networks. XBee and XBee-PRO 802.15.4 OEM RF modules are embedded solutions providing wireless end-point connectivity to devices. These modules use the IEEE 802.15.4 networking protocol for fast point-to-multipoint or peer-to-peer networking. They are designed for high-throughput applications requiring low latency and predictable communication timing. XBee modules are ideal for low-power, low-cost applications. XBee-PRO modules are power-amplified versions of XBee modules for extended-range applications.

4.7.2 Specifications

Specification	XBee	XBee-PRO (S2)	XBee-PRO (S2B)
Performance			
Indoor/Urban Range	up to 133 ft. (40 m)	Up to 300 ft. (90 m), up to 200 ft (60 m) international variant	Up to 300 ft. (90 m), up to 200 ft (60 m) international variant
Outdoor RF line-of-sight Range	up to 400 ft. (120 m)	Up to 2 miles (3200 m), up to 5000 ft (1500 m) international variant	Up to 2 miles (3200 m), up to 5000 ft (1500 m) international variant

Transmit Power Output	2mW (+3dBm), boost mode enabled 1.25mW (+1dBm), boost mode	50mW (+17 dBm) 10mW (+10 dBm) for International	63mW (+18 dBm) 10mW (+10 dBm) for International variant
RF Data Rate	250,000 bps	250,000 bps	250,000 bps
Data Throughput	up to 35000 bps (see chapter 4)	up to 35000 bps (see chapter 4)	up to 35000 bps (see chapter 4)
Serial Interface Data Rate (software selectable)	1200 bps - 1 Mbps (non-standard baud rates also	1200 bps - 1 Mbps (non-standard baud rates also	1200 bps - 1 Mbps (non-standard baud rates also supported)
Receiver Sensitivity	-96 dBm, boost mode enabled -95 dBm, boost mode disabled	-102 dBm	-102 dBm
Power Requirements			
Supply Voltage	2.1 - 3.6 V	3.0 - 3.4 V	2.7 - 3.6 V
Operating Current (Transmit, max output	40mA (@ 3.3 V, boost mode enabled) 35mA (@ 3.3 V, boost mode	295mA (@3.3 V) 170mA (@3.3 V) international variant	205mA, up to 220 mA with programmable variant (@3.3 V) 217mA, up to 232 mA with programmable
Operating Current (Receive))	40mA (@ 3.3 V, boost mode enabled) 38mA (@ 3.3 V, boost mode	45 mA (@3.3 V)	47 mA, up to 62 mA with programmable variant (@3.3 V)
Idle Current (Receiver off)	15mA	15mA	15mA
Power-down Current	< 1 uA @ 25ºC	3.5 µA typical @ 25ºC	3.5 µA typical @ 25ºC
General			
Operating Frequency Band	ISM 2.4 GHz	ISM 2.4 GHz	ISM 2.4 GHz
Dimensions	0.960" x 1.087" (2.438cm x 2.761cm)	0.960 x 1.297 (2.438cm x 3.294cm)	0.960 x 1.297 (2.438cm x 3.294cm)
Operating Temperature	-40 to 85º C (industrial)	-40 to 85º C (industrial)	-40 to 85º C (industrial)
Antenna Options	Integrated Whip, Chip, RPSMA, or U.FL Connector	Integrated Whip, Chip, RPSMA, or U.FL Connector	Integrated Whip,PCB Embedded Trace, RPSMA, or U.FL Connector
Networking & Security			
Supported Network Topologies	Point-to-point, Point-to-multipoint, Peer-to-peer, and Mesh	Point-to-point, Point-to-multipoint, Peer-to-peer, and Mesh	Point-to-point, Point-to-multipoint, Peer-to-peer, and Mesh
Number of Channels	16 Direct Sequence Channels	14 Direct Sequence Channels	15 Direct Sequence Channels
Channels	11 to 26	11 to 24	11 to 25
Addressing Options	PAN ID and Addresses, Cluster IDs and Endpoints (optional)	PAN ID and Addresses, Cluster IDs and Endpoints (optional)	PAN ID and Addresses, Cluster IDs and Endpoints (optional)
Agency Approvals			
United States (FCC Part 15.247)	FCC ID: OUR-XBEE2	FCC ID: MCQ-XBEEPRO2	FCC ID: MCQ-PROS2B
Industry Canada (IC)	IC: 4214A-XBEE2	IC: 1846A-XBEEPRO2	IC: 1846A-PROS2B
Europe (CE)	ETSI	ETSI	ETSI

4.7.3 Hardware Specifications

The following specifications need to be added to the current measurement of the previous

table if the module has the programmable secondary processor. For example, if the secondary processor is running and constantly collecting DIO samples at a rate while having the RF portion of the XBEE sleeping the new current will be I total = I_{r2} + I_s. Where I_{r2} is the runtime current of the secondary processor and I_s is the sleep current of the RF portion of the module of the XBEE-PRO (S2B) listed in the table below.

Optional Secondary Processor Specification	These numbers add to S2B specifications (Add to RX, TX, and sleep currents depending on mode of operation)
Runtime current for 32k running at 20MHz	+14mA
Runtime current for 32k running at 1MHz	+1mA
Sleep current	+0.5uA typical
For additional specifications see Freescale Datasheet and Manual	MC9SO8QE32
Minimum Reset low pulse time for EM250	+50 nS (additional resistor increases minimum time)
VREF Range	1.8VDC to VCC

4.7.4 Pin Description

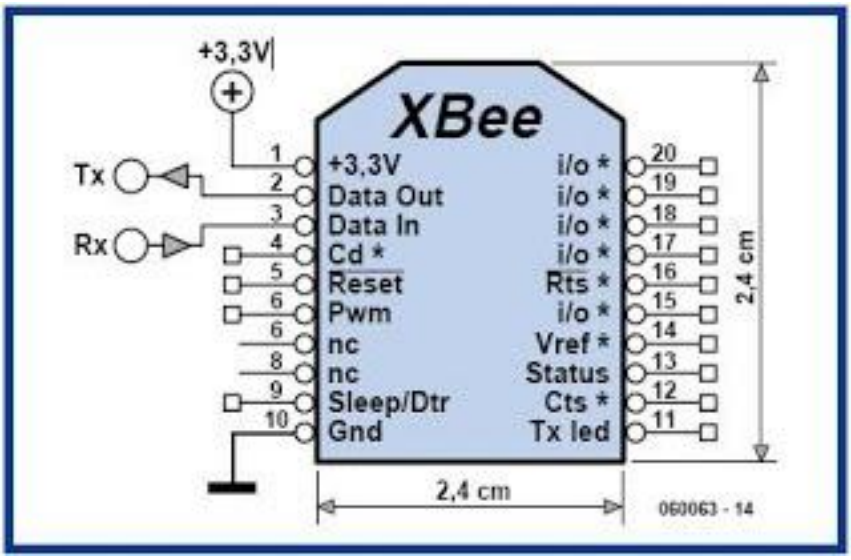


Fig 4.11 Xbee Pin Diagram

Pin Assignments for the XBee-PRO Modules

(Low-asserted signals are distinguished with a horizontal line above signal name.)

Pin #	Name	Direction	Default State	Description
1	VCC	-	-	Power supply
2	DOUT	Output	Output	UART Data Out
3	DIN / **CONFIG**	Input	Input	UART Data In
4	DIO12	Both	Disabled	Digital I/O 12
5	**RESET**	Both	Open-Collector with pull-up	Module Reset (reset pulse must be at least 200 ns)
6	RSSI PWM / DIO10	Both	Output	RX Signal Strength Indicator / Digital IO
7	DIO11	Both	Input	Digital I/O 11
8	[reserved]	-	Disabled	Do not connect
9	DTR / SLEEP_RQ/ DIO8	Both	Input	Pin Sleep Control Line or Digital IO 8
10	GND	-	-	Ground
11	DIO4	Both	Disabled	Digital I/O 4
12	**CTS** / DIO7	Both	Output	Clear-to-Send Flow Control or Digital I/O 7. CTS, if enabled, is an output.
13	ON / **SLEEP**	Output	Output	Module Status Indicator or Digital I/O 9
14	VREF	Input	-	Not used for EM250. Used for programmable secondary processor. For compatibility with other XBEE modules, we recommend connecting this pin voltage reference if Analog sampling is desired. Otherwise, connect to GND.
15	Associate / DIO5	Both	Output	Associated Indicator, Digital I/O 5
16	**RTS** / DIO6	Both	Input	Request-to-Send Flow Control, Digital I/O 6. RTS, if enabled, is an input.
17	AD3 / DIO3	Both	Disabled	Analog Input 3 or Digital I/O 3
18	AD2 / DIO2	Both	Disabled	Analog Input 2 or Digital I/O 2
19	AD1 / DIO1	Both	Disabled	Analog Input 1 or Digital I/O 1
20	AD0 / DIO0 / Commissioning Button	Both	Disabled	Analog Input 0, Digital IO 0, or Commissioning Button

4.7.5 Module Operation

4.7.5.1 Serial Communications

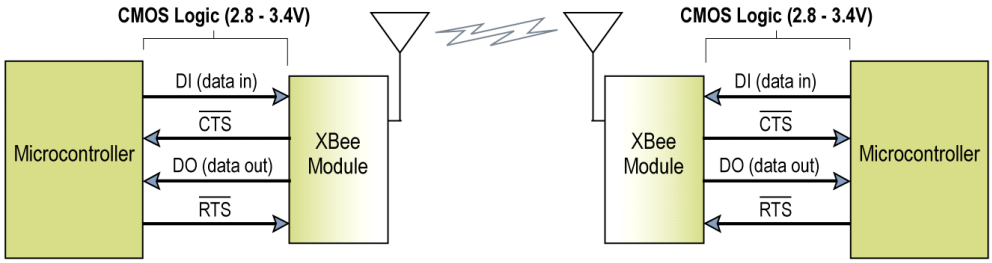
The XBee RF Modules interface to a host device through a logic-level asynchronous serial port. Through its serial port, the module can communicate with any logic and voltage compatible UART; or through a level translator to any serial device (for example: through a RS-232 or USB interface board).

4.7.5.2 UART Data Flow

Devices that have a UART interface can connect directly to the pins of the RF module as shown in the figure below.

Fig 4.12 System Data Flow Diagram in a UART-interfaced environment

(Low-asserted signals distinguished with horizontal line over signal name.)



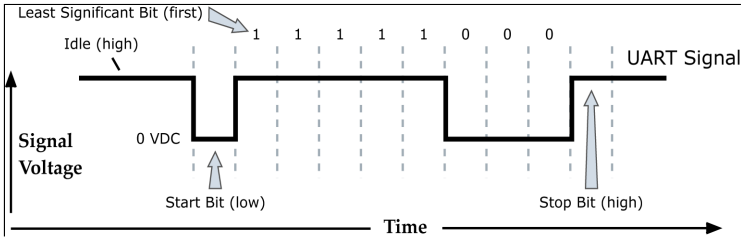
4.7.5.3 Serial Data

Data enters the module UART through the DIN (pin 3) as an asynchronous serial signal. The signal should idle high when no data is being transmitted.

Each data byte consists of a start bit (low), 8 data bits (least significant bit first) and a stop bit (high). The following figure illustrates the serial bit pattern of data passing through the module.

Fig 4.13 UART data packet 0x1F (decimal number "31") as transmitted through the RF module

Example Data Format is 8-N-1 (bits - parity - # of stop bits)



Serial communications depend on the two UARTs (the microcontroller's and the RF module's) to be configured with compatible settings (baud rate, parity, start bits, stop bits, data bits).

The UART baud rate, parity, and stop bits settings on the XBee module can be configured with the BD, NB, and SB commands respectively.

4.7.5.4 Serial Interface Protocols

The XBee modules support both transparent and API (Application Programming Interface) serial interfaces.

Transparent Operation: When operating in transparent mode, the modules act as a serial line replacement. All UART data received through the DIN pin is queued up for RF transmission. When RF data is received, the data is sent out through the DOUT pin. The module configuration parameters are configured using the AT command mode interface.

Data is buffered in the serial receive buffer until one of the following causes the data to be packetized and transmitted:

- No serial characters are received for the amount of time determined by the RO (Packetization Time- out) parameter. If RO = 0, packetization begins when a character is received.
- The Command Mode Sequence (GT + CC + GT) is received. Any character buffered in the serial receive buffer before the sequence is transmitted.
- The maximum number of characters that will fit in an RF packet is received.

API Operation: API operation is an alternative to transparent operation. The frame-based API extends the level to which a host application can interact with the networking capabilities of the module. When in API mode, all data entering and leaving the module is contained in frames that define operations or events within the module.

Transmit Data Frames (received through the DIN pin (pin 3)) include:

- RF Transmit Data Frame
- Command Frame (equivalent to AT commands)

Receive Data Frames (sent out the DOUT pin (pin 2)) include:

- RF-received data frame
- Command response
- Event notifications such as reset, associate, disassociate, etc.

The API provides alternative means of configuring modules and routing data at the host application layer. A host application can send data frames to the module that contain address and payload information instead of using command mode to modify addresses. The module will send data frames to the application containing status packets; as well as source, and payload information from received data packets.

The API operation option facilitates many operations such as the examples cited below:

-> Transmitting data to multiple destinations without entering Command Mode

-> Receive success/failure status of each transmitted RF packet

-> Identify the source address of each received packet

4.7.5 Xbee Zigbee Network

4.7.5.1 Introduction to Zigbee

ZigBee is an open global standard built on the IEEE 802.15.4 MAC/PHY. ZigBee defines a network layer above the 802.15.4 layers to support advanced mesh routing capabilities. The ZigBee specification is developed by a growing consortium of companies that make up the ZigBee Alliance. The Alliance is made up of over 300 members, including semiconductor, module, stack, and software developers.

4.7.5.2 Zigbee Stack Layers

The ZigBee stack consists of several layers including the PHY, MAC, Network, Application Support Sublayer (APS), and ZigBee Device Objects (ZDO) layers. Technically, an Application Framework (AF) layer also exists, but will be grouped with the APS layer in remaining discussions. The ZigBee layers are shown in the figure below.

A description of each layer appears in the following table:

ZigBee Layer	Description
PHY	Defines the physical operation of the ZigBee device including receive sensitivity, channel rejection, output power, number of channels, chip modulation, and transmission rate specifications. Most ZigBee applications operate on the 2.4 GHz ISM band at a

MAC	Manages RF data transactions between neighboring devices (point to point). The MAC includes services such as transmission retry and acknowledgment management, and collision avoidance techniques
Network	Adds routing capabilities that allows RF data packets to traverse multiple devices (multiple "hops") to route data from source to destination (peer to peer)
APS (AF)	Application layer that defines various addressing objects including profiles, clusters, and endpoints.
ZDO	Application layer that provides device and service discovery features and advanced network management capabilities

4.7.6. Networking Concepts

4.7.6.1 Device Types:

ZigBee defines three different device types: coordinator, router, and end device. Node Types / Sample of a Basic ZigBee Network Topology

A **coordinator** has the following characteristics: it

- •Selects a channel and PAN ID (both 64-bit and 16-bit) to start the network
- •Can allow routers and end devices to join the network
- •Can assist in routing data
- •Cannot sleep--should be mains powered
- •Can buffer RF data packets for sleeping end device children.

A **router** has the following characteristics: it

- •Must join a ZigBee PAN before it can transmit, receive, or route data
- •After joining, can allow routers and end devices to join the network
- •After joining, can assist in routing data
- •Cannot sleep--should be mains powered.
- •Can buffer RF data packets for sleeping end device children.

An **end device** has the following characteristics: it

•Must join a ZigBee PAN before it can transmit or receive data

•Cannot allow devices to join the network

•Must always transmit and receive RF data through its parent.

•Cannot route data.

•Can enter low power modes to conserve power and can be battery-powered.

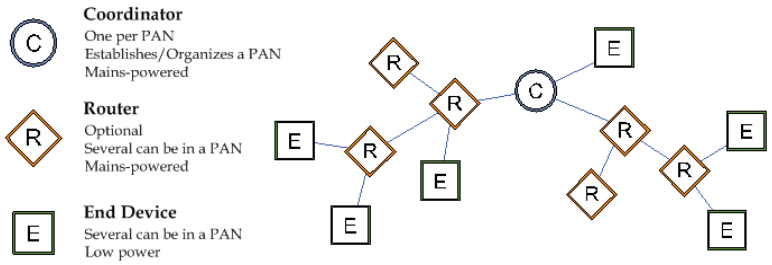


Fig 4.14 Example of Network

In ZigBee networks, the coordinator must select a PAN ID (64-bit and 16-bit) and channel to start a network. After that, it behaves essentially like a router. The coordinator and routers can allow other devices to join the network and can route data.

After an end device joins a router or coordinator, it must be able to transmit or receive RF data through that router or coordinator. The router or coordinator that allowed an end device to join becomes the "parent" of the end device. Since the end device can sleep, the parent must be able to buffer or retain incoming data packets
destined for the end device until the end device is able to wake and receive the data.

4.7.6.2 PAN ID

ZigBee networks are called personal area networks or PANs. Each network is defined with a unique PAN identifier (PAN ID). This identifier is common among all devices of the same network. ZigBee devices are either preconfigured with a PAN ID to join, or they can discovery nearby networks and select a PAN ID to join.

ZigBee supports both a 64-bit and a 16-bit PAN ID. Both PAN IDs are used to uniquely identify a network. Devices on the same ZigBee network must share the same 64-bit and 16-bit PAN IDs. If multiple ZigBee networks are operating within range of each other, each

should have unique PAN IDs.

The 16-bit PAN ID is used as a MAC layer addressing field in all RF data transmissions between devices in a network. However, due to the limited addressing space of the 16-bit PAN ID (65,535 possibilities), there is a possibility that multiple ZigBee networks (within range of each other) could use the same 16-bit PAN ID. To resolve potential 16-bit PAN ID conflicts, the ZigBee Alliance created a 64-bit PAN ID.

The 64-bit PAN ID (also called the extended PAN ID), is intended to be a unique, non-duplicated value. When a coordinator starts a network, it can either start a network on a preconfigured 64-bit PAN ID, or it can select a random 64-bit PAN ID. The 64-bit PAN ID is used during joining; if a device has a preconfigured 64-bit PAN ID, it will only join a network with the same 64-bit PAN ID. Otherwise, a device could join any detected PAN and inherit the PAN ID from the network when it joins. The 64-bit PAN ID is included in all ZigBee beacons and is used in 16-bit PAN ID conflict resolution.

Routers and end devices are typically configured to join a network with any 16-bit PAN ID as long as the 64- bit PAN ID is valid. Coordinators typically select a random 16-bit PAN ID for their network.

Since the 16-bit PAN ID only allows up to 65,535 unique values, and since the 16-bit PAN ID is randomly selected, provisions exist in ZigBee to detect if two networks (with different 64-bit PAN IDs) are operating on the same 16-bit PAN ID. If such a conflict is detected, the ZigBee stack can perform PAN ID conflict resolution to change the 16-bit PAN ID of the network in order to resolve the conflict. See the ZigBee specification for details.

To summarize, ZigBee routers and end devices should be configured with the 64-bit PAN ID of the network they want to join. They typically acquire the 16-bit PAN ID when they join a network.

4.7.6.3 Operating Channel

ZigBee utilizes direct-sequence spread spectrum modulation and operates on a fixed channel. The 802.15.4 PHY defines 16 operating channels in the 2.4 GHz frequency band. XBee modules support all 16 channels and XBee- PRO modules support 14 of the 16 channels.

4.7.7 Interfacing Xbee with Arduino

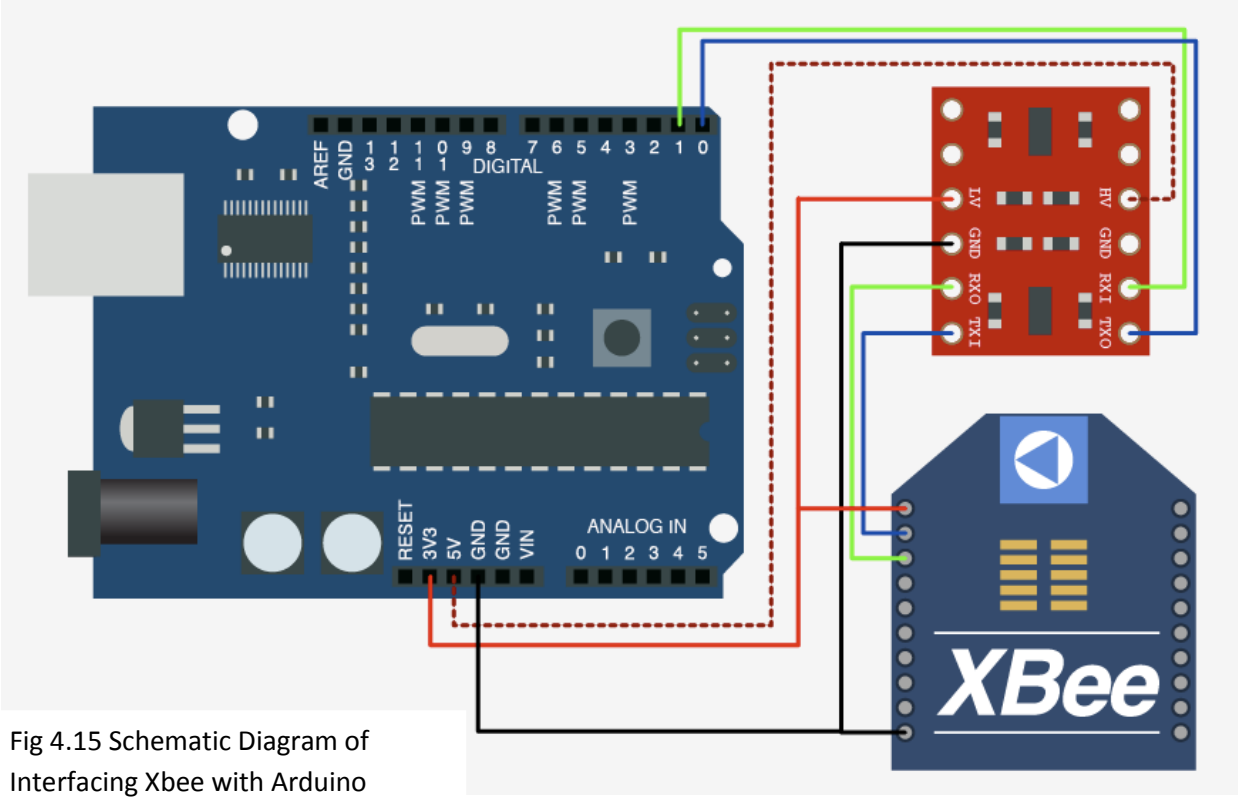


Fig 4.15 Schematic Diagram of Interfacing Xbee with Arduino

The adjacent figure shows connection of Arduino with Xbee Module.

A Xbee Shield is used to connect Xbee with Arduino. Xbee Shield is a specific board which can be easily mounted on Arduino board and can help users to access the pins of Arduino which is not in use by Xbee Modules for further interfaces.

Fig 4.16 Working Demo of Arduino & Xbee

4.8 GPS (Global Positioning System)

GPS is a satellite-based navigation system originally developed for military purposes and is maintained and controlled by the United States Department of Defense. GPS permits land, sea, and airborne users to determine their three-dimensional position, velocity, and time. It can be used by anyone with a receiver anywhere on the planet, at any time of day or night, in any type of weather.

It also called NAVSTAR (NAVigation System, Timing And Ranging)

4.8.1 GPS Segments

GPS uses radio transmissions. The satellites transmit timing information and satellite location information. The system can be separated into three parts:

- Space Segment
- Control Segment
- User Segment
-

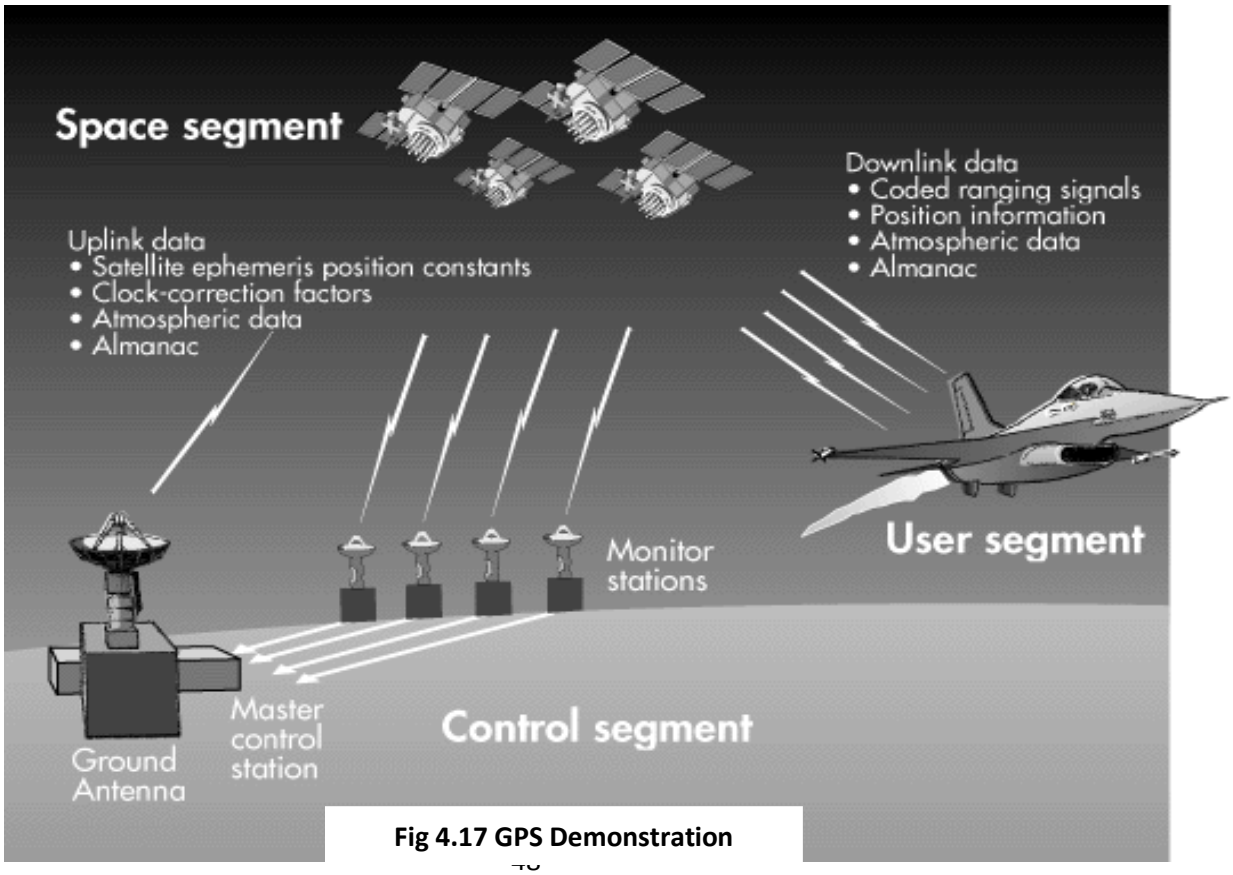


Fig 4.17 GPS Demonstration

4.8.2 Mathematical Basis

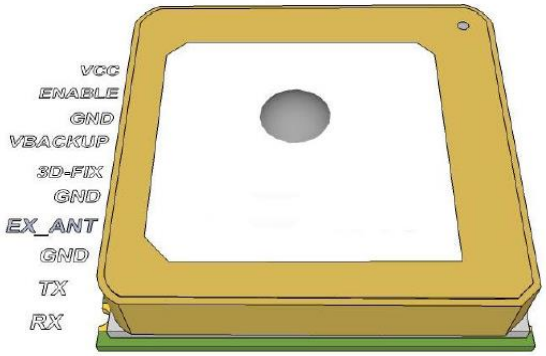
Each of the GPS satellites transmits radio signals. GPS receivers pick up these signals and measure the distance to a satellite by multiplying the speed of the signal by the time it takes the signal to get there. The speed of the signal is the speed of light and the time is encoded within the signal. The satellites also send information on their exact location.

In order to find longitude, latitude, and altitude, four satellites are needed. If a measurement is taken using just one satellite, then all that is known is that the receiver is on the surface of a sphere with radius equal to the distance to the satellite. If two satellites are used, then the receiver must be on the surface of both spheres which is the intersection of the two spheres or the perimeter of a circle. If a third satellite is used, then the location of the user is narrowed down to the two points where the three spheres intersect. Three measurements are enough for land receivers since the lower of the two points would be selected. But when in the air or space, four satellites are needed: the intersection of all four spheres will be the receiver's location. When more than four satellites are used, greater accuracy can be achieved.

4.8.3 GPS Specifications

Fig 4.18 GPS Module

Fig 4.19 Patch Antenna



- Dimension： 16mm x 16mm x 6mm (Full circuit 80mm x 23mm x 12mm)
- Patch Antenna Size：15mm x 15mm x 4mm
- Low Power Consumption：55mA @ acquisition, 40mA @ trackingL1 Frequency, C/A code, 51-channel

- High Sensitivity：Up to -158 dBm tracking, superior urban performances1
- Position Accuracy：Without aid: 3m 2D-RMS
- DGPS(RTM,SBAS(WAAS,EGNOS,MASA)):2.5m 2D-RMS
- Cold Start is Under 36 seconds (Typical)
- Warm Start is Under 34 seconds (Typical)
- Hot Start is Under 1 second (Typical)
- Data output Baud rate：9600 bps

Pin Definition

Pin	Name	I/O	Description
1	VCC	PI	Main DC power input
2	ENABLE	I	High active, or keep floating for normal working
3	GND	P	Ground
4	VBACKUP	PI	Backup power input
5	3D-FIX	O	3D-fix indicator
6	GND	P	Ground
7	EX_ANT	I/O	External Antenna 3.0V input and output for external antenna. The maximum consumption current for the GPS antenna is limited to 30mA. When a 3mA or higher current is detected, the IC will acknowledge the external antenna as being present and uses external antenna for reception. In the event of short circuit occurring at external antenna, the module will limit the drawn current to a safe level.
8	GND	P	Ground
9	TX	O	Serial data output of NMEA
10	RX	I	Serial data input for firmware update

4.8.4 Understanding of GPS code

Every GPS Module follows a specific protocol called '**NMEA'** protocol.

The following table shows a list of possible output codes

NMEA Output Sentence	**Table-1**
Option	**Description**
GGA	Time, position and fix type data.
GSA	GPS receiver operating mode, active satellites used in the position solution, and DOP values.
GSV	The number of GPS satellites in view satellite ID numbers, elevation, azimuth, and SNR values.
RMC	Time, date, position, course and speed data. Recommended Minimum Navigation Information.
VTG	Course and speed information relative to the ground.

Code (1)

GGA: Global Positioning System Fixed Data.

Time, Position and Fix related data for a GPS receiver.

Ex:

$GPGGA,064951.000,2307.1256,N,12016.4438,E,1,8,0.95,39.9,M,17.8,M,,*65

GGA Data Format **Table-2**

Name	Example	Units	Description
Message ID	$GPGGA		GGA protocol header
UTC Time	064951.000		hhmmss.sss
Latitude	2307.1256		ddmm.mmmm
N/S Indicator	N		N=north or S=south
Longitude	12016.4438		dddmm.mmmm
E/W Indicator	E		E=east or W=west
Position Fix Indicator	1		See **Table-3**
Satellites Used	8		Range 0 to 14
HDOP	0.95		Horizontal Dilution of Precision
MSL Altitude	39.9	meters	Antenna Altitude above/below mean-sae-level
Units	M	meters	Units of antenna altitude
Geoidal Separation	17.8	meters	
Units	M	meters	Units of geoidal separation
Age of Diff. Corr.		second	Null fields when DGPS is not used
Checksum	*65		
<CR> <LF>			End of message termination

Position Fix Indicator **Table-3**

Value	Description
0	Fix not available
1	GPS fix
2	Differential GPS fix

Code(2)

GSA: GPS receiver operating mode, Active Satellites used in the position solution & DOP values

Ex:

$GPGSA,A,3,29,21,26,15,18,09,06,10,,,,,2.32,0.95,2.11*00

GSA Data Format			Table-4
Name	**Example**	**Units**	**Description**
Message ID	$GPGSA		GSA protocol header
Mode 1	A		See **Table-5**
Mode 2	3		See **Table-6**
Satellite Used	29		SV on Channel 1
Satellite Used	21		SV on Channel 2
....	....	....	....
Satellite Used			SV on Channel 12
PDOP	2.32		Position Dilution of Precision
HDOP	0.95		Horizontal Dilution of Precision
VDOP	2.11		Vertical Dilution of Precision
Checksum	*00		
<CR> <LF>			End of message termination

Mode 1	Table-5
Value	**Description**
M	Manual—forced to operate in 2D or 3D mode
A	2D Automatic—allowed to automatically switch 2D/3D

Mode 2	Table-6
Value	**Description**
1	Fix not available
2	2D (< 4 SVs used)
3	3D (≧4 SVs used)

Code(3)

GSV: The number of Satellites in view, Satellite ID numbers, Elevation, Azimuth and SNR value

Ex:

$GPRMC,064951.000,A,2307.1256,N,12016.4438,E,0.03,165.48,260406, 3.05,W,A*55

GSV Data Format			**Table-7**
Name	**Example**	**Units**	**Description**
Message ID	$GPGSV		GSV protocol header
Number of Messages	3		Range 1 to 3 *(Depending on the number of satellites tracked, multiple messages of GSV data may be required.)*
Message Number1	1		Range 1 to 3
Satellites in View	09		
Satellite ID	29		Channel 1 (Range 1 to 32)
Elevation	36	degrees	Channel 1 (Maximum 90)
Azimuth	029	degrees	Channel 1 (True, Range 0 to 359)
SNR (C/No)	42	dBHz	Range 0 to 99, (null when not tracking)
....	....	....	....
Satellite ID	15		Channel 4 (Range 1 to 32)
Elevation	21	degrees	Channel 4 (Maximum 90)
Azimuth	321	degrees	Channel 4 (True, Range 0 to 359)
SNR (C/No)	39	dBHz	Range 0 to 99, (null when not tracking)
Checksum	*7D		
<CR> <LF>			End of message termination

Code(4)

RMC: Recommended Minimum Navigation Information

Ex:

$GPGSV,3,1,09,29,36,029,42,21,46,314,43,26,44,020,43,15,21,321,39*7D

$GPGSV,3,2,09,18,26,314,40,09,57,170,44,06,20,229,37,10,26,084,37*77

$GPGSV,3,3,09,07,,,26*73

RMC Data Format			Table-8
Name	**Example**	**Units**	**Description**
Message ID	$GPRMC		RMC protocol header
UTC Time	064951.000		hhmmss.sss
Status	A		A=data valid or V=data not valid
Latitude	2307.1256		ddmm.mmmm
N/S Indicator	N		N=north or S=south
Longitude	12016.4438		dddmm.mmmm
E/W Indicator	E		E=east or W=west
Speed Over Ground	0.03	knots	
Course Over Ground	165.48	degrees	True
Date	260406		ddmmyy
Magnetic Variation	3.05,W	degrees	E=east or W=west
Mode	A		A= Autonomous mode D= Differential mode E= Estimated mode
Checksum	*55		
<CR> <LF>			End of message termination

Code(5)

VTG: Course and speed information relative to the ground.

Ex:

$GPVTG,165.48,T,,M,0.03,N,0.06,K,A*37

VTG Data Format			Table-9
Name	**Example**	**Units**	**Description**
Message ID	$GPVTG		VTG protocol header
Course	165.48	degrees	Measured heading
Reference	T		True
Course		degrees	Measured heading
Reference	M		Magnetic
Speed	0.03	knots	Measured horizontal speed
Units	N		Knots
Speed	0.06	km/hr	Measured horizontal speed
Units	K		Kilometers per hour
Mode	A		A= Autonomous mode D= Differential mode E= Estimated mode
Checksum	*06		
<CR> <LF>			End of message termination

4.8.5 Interfacing GPS with ARDUINO

Pin Connection:

GPS Module		Arduino
Rx	➔	Tx
Tx	➔	Rx
Vcc	➔	5V
GND	➔	GND

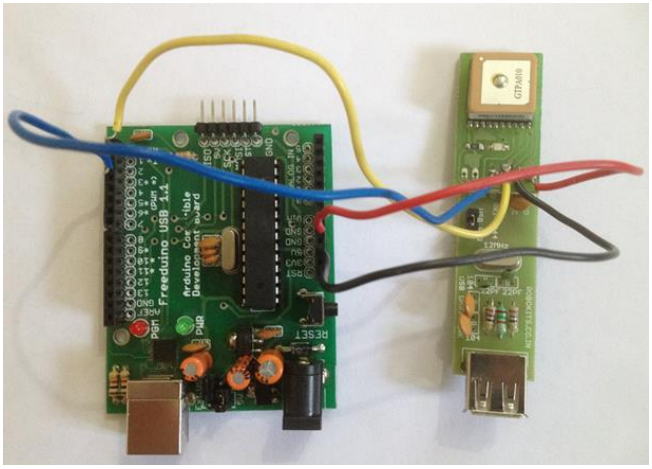


Fig 4.20 Connection of Arduino with GPS

4.9 Processing IDE

Processing is an open source programming language and environment for people who want to create images, animations, and interactions. Initially developed to serve as a software sketchbook and to teach fundamentals of computer programming within a visual context, Processing also has evolved into a tool for generating finished professional work. Today, there are tens of thousands of students, artists, designers, researchers, and hobbyists who use Processing for learning, prototyping, and production.

Processing is a great development tool for writing GUI programs. Useful when you want computers to "talk" with an Arduino, for instance to display or save some data collected by the Arduino.

Fig 4.21 Processing IDE

Arduino sends GPS data serially via Xbee to control room, where another Xbee receives that data. Now this data is in the form of geographic coordinates i.e. latitudes and longitudes. But

this data will be useless unless it is utilized in a proper fashion. There comes the use of Processing IDE. Processing helps users to display GUI by writing proper codes in Processing Language.

Processing Language is quite easy to understand & is quite similar to C & C++ language. Having a basic C programming knowledge anybody can understand the codes written in Processing Language. With the help of Processing, a GUI is developed for UMV, which helps user to track the current GPS location of UMV in a Map as well as can obtain a track record of the path that UMV has travelled. By proper coding Google Maps API is inserted in Processing GUI for tracking purpose. Since Google Maps API has a limit on users accessing its maps. The whole programming of Processing is revised.

A new program was written which employs Microsoft's Maps, which removes the previous limitation of Google API's. Also Microsoft provides three different views in Map viz Road view, Aerial view & Hybrid view. Thus users can track in three different views as per his/her wish.

Since Arduino sends data serially to Xbee at the control room. The Processing program communicates with this Xbee in the control room serially to get those GPS co-ordinates. Having obtained these GPS co-ordinates, it'll display these co-ordinates on Microsoft's maps. As GPS data is continuously coming at a rate of 1 Hz, the program continuously monitors this new GPS data and plots new co-ordinates on the Maps thereby creating a track record of the currently deployed UMV.

Displaying UMV's location on map is not enough; user might require this track record later for some other purpose. So for every route that UMV travels a GPS Track Record is generated in .txt format which can be easily opened and viewed in Notepad. Every track record is numbered so that next time when the program runs, it should not overwrite the previous data files. Some additional features that are added in programming for the ease of operator are:

➔ Saving current GPS Map as Image file

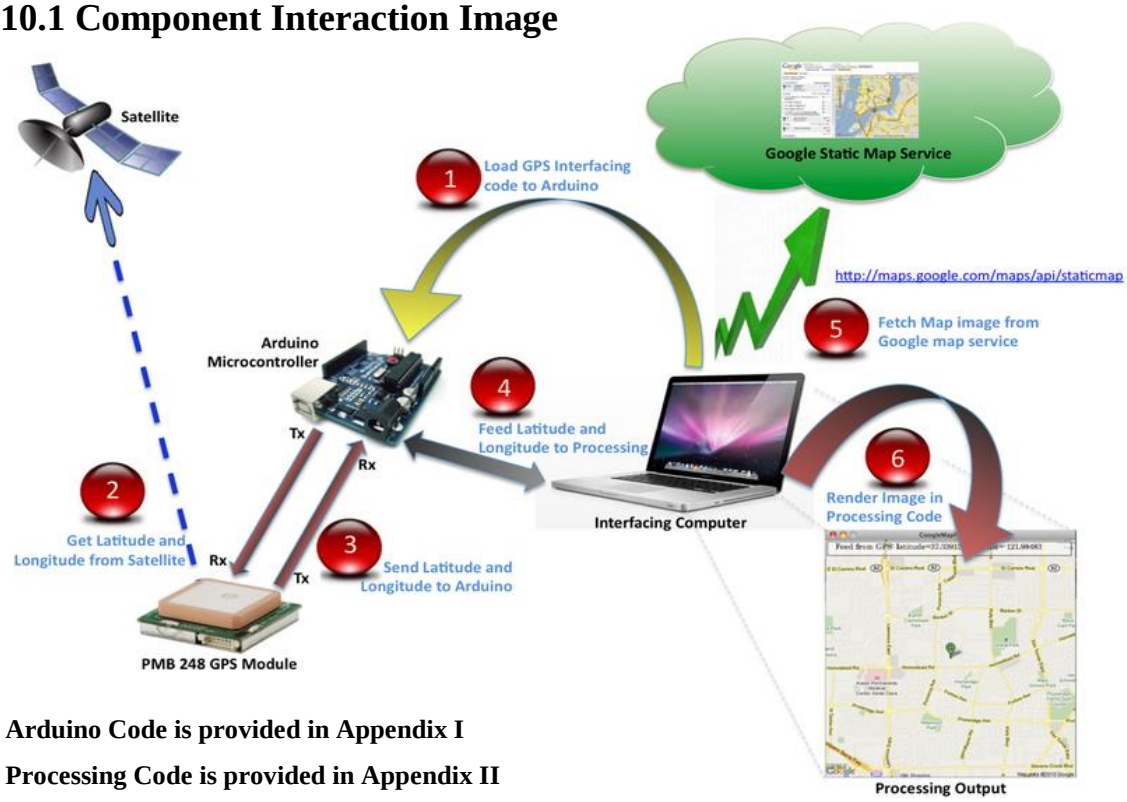
➔ Zoom in & Zoom out Buttons as well Zoom with Mouse Scroll

➔ Use of Pan/Tilt buttons or Arrow keys for changing map position.

➔ Use of ‘C’ button to center the map at current GPS location, etc

4.10 Functional System Overview

4.10.1 Component Interaction Image



**** Arduino Code is provided in Appendix I**

**** Processing Code is provided in Appendix II**

4.10.2 On-Board Functional Electronic System:

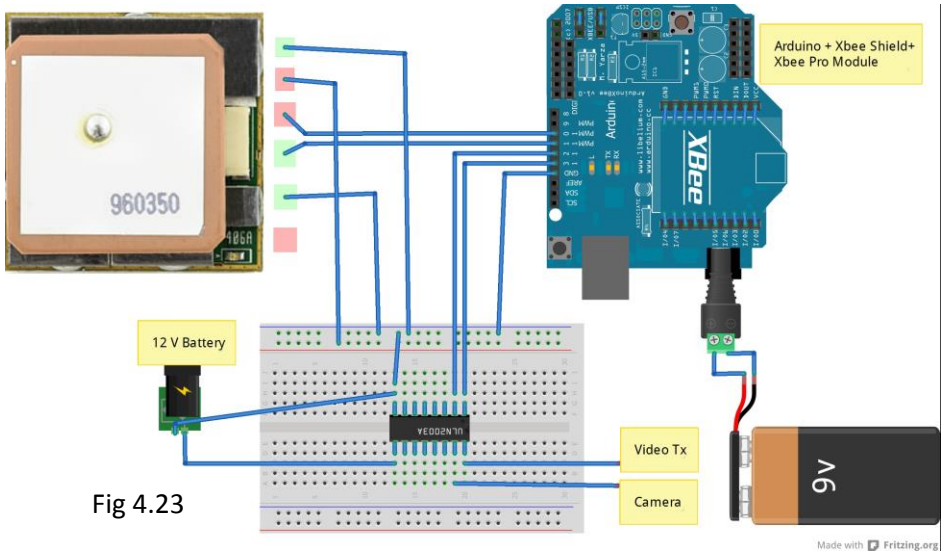


Fig 4.23

4.11 Video Transmission Via Camera

Fig 4.24

To obtain a wireless video link between the marine vehicle and the control room, the following components are used :

4.11.1 Camera

Our UMV system employs a CCTV camera in order to achieve the objective of Video Surveillance. This Camera provides an RCA output which has an IR Vision which helps the user to monitor at night. It is powered with a 2.1mm Power Plug. It also has an inbuilt microphone which provides the audio capturing capabilities to the marine vessel.

4.11.2 Video Transmitter

It is a light weight, portable and small video transmitting device that transmits the video output from the camera thereby providing a wireless video transmission link to the UMV.

Fig 4.25

The specifications of transmitter are:

Voltage	12V DC
Current	260mA
Output Power	700mW
Dimension	65 * 50 * 19.5
Frequency	0.9GHz to 1.3GHz
Operating Temperature	-20 to 60 °C
Video Output Format	NTSC, PAL
Video Output	1 RCA Composite, 1 S-Video

4.11.3 Video Receiver

It is a light weight, portable and small video receiving device that receives the video output from the video transmitter wirelessly, thereby completing a wireless video transmission link to the UMV.

The specifications of receiver are:

Voltage	12V DC
Current	350mA
Receiver Sensitivity	1000mW
Dimension	65 * 50 * 19.5
Frequency	0.9GHz to 1.3GHz
Operating Temperature	-20 to 60 °C
Video Output Format	NTSC, PAL
Video Output	1 RCA Composite, 1 S-Video
Number of Channels	15

Future Scope

- UMV can be equipped with SONAR for littoral scanning purposes. Attaching a sonar to the surface of the UMV, data regarding the depths of the uncharted oceans could be wirelessly obtained and thus surveying could be carried out.
- It can have weaponry systems for targeting enemies. Having attached the weaponry systems, the UMV could be used for warfare purposes where the major advantage is the loss of life is minimum. Here the UMV is controlled from the control room so the lives endangered are nil.
- Toeing system can be implemented for rescue purpose. Attaching a lifesaving Ring or any other lifesaving equipment, this UMV could be used for rescue purposes near the coastal areas.
- It can employ thermal cameras for better night vision capabilities. Equipping the UMV with the thermal camera, quality surveillance could be carried out during the night times when the coast is devoid of a light source. Simple cameras cold not fulfill the purpose of surveillance during the night time and so equipping the UMV with a thermal camera, quality surveillance could be carried out.
- It can use different sensors for hydrographic survey purpose. Mounting the UMV with hydraulic sensors, the temperature of water in various regions of the oceans could be found out. This temperature survey could be extremely useful for studying various marine life and also prove to be helpful in the transfer/migration of such species.

References

1) Daniel Shiffman (2008) 'LEARNING PROCESSING' , Morgan Kaufmann Publishers

2) Chris Ellix (1999) 'Radio Controlled Models', Chartwell Books NJ

3) Tiny GPS Library- "http://arduiniana.org/libraries/tinygps"

4) Modest Maps Library- " http://www.tom-carden.co.uk/2008/02/18/modest-maps-vs-processing "

5) Arduino GPS Integration- "http://it-architect.wikispaces.com/Arduino+GPS+Integration"

6) Xbee Shield- "http://arduino.cc/en/Main/ArduinoXbeeShield"

7) T6 Config- "http://www.mycoolheli.com/t6config.html"

8) GPS Datasheet- "http://www.robokits.co.in/downloads/Robokits_GPS_02.pdf"

9) RC Boat Hull- "http://www.building-model-boats.com/rc-boat-hull.html"

Appendix I

Arduino Code:

```
#include <SoftwareSerial.h>
#include <TinyGPS.h>
#define camPin 12
#define vidTxPin 13
#define RXPIN 10
#define TXPIN 11
#define GPSBAUD 9600

TinyGPS gps;
SoftwareSerial uart_gps(RXPIN, TXPIN);

void getgps(TinyGPS &gps);

void setup() {

  Serial.begin(9600);
  pinMode(camPin,OUTPUT);
  pinMode(vidTxPin,OUTPUT);
  uart_gps.begin(GPSBAUD);

  Serial.println("");
  Serial.println("GPS");
  Serial.println("       ...waiting for lock...           ");
  Serial.println("");

}
```

```
void loop() {
   while(uart_gps.available())  {
     int c = uart_gps.read();
     if(gps.encode(c)) {
       getgps(gps);
     }
   }
   digitalWrite(camPin,HIGH);
   digitalWrite(vidTxPin,HIGH);
}
void getgps(TinyGPS &gps) {
  float latitude, longitude;
  gps.f_get_position(&latitude, &longitude);
  Serial.print(latitude,5);
  Serial.print(",");
  Serial.println(longitude,5);
}
```

Appendix II

Processing Code:

```
import com.modestmaps.*;
import com.modestmaps.core.*;
import com.modestmaps.geo.*;
import com.modestmaps.providers.*;
import java.util.List;

//
// This is a test of the interactive Modest Maps library for Processing
// New Features added like Numbering GPS Images instead of one single image.
// Serial Event function
//
// Following is the list of switches and their functions:
// s = save current GPS map in an image
// g = turn on/off GUI
// space key = see whole world
// z = view perfectly loaded map in case of zoom in/out
// c = center map to current GPS location
// + = zoom in
// - = zoom out
// left arrow = move left in map
// right arrow = move right in map
// up arrow = move up in map
// down arrow = move down in map
//

import java.io.File;
```

```
import processing.serial.*;
String path ;
PrintWriter output;
float lat, lng ;
float static_lat =23.064696, static_lng= 72.43961;
Serial myPort;
int imgcount=0;
boolean imgexist = true;
// this is the only bit that's needed to show a map:
InteractiveMap map;
float plat, plng;// previous lattitude and longitudes

final static List<PVector> dump = new ArrayList();      // dump all the cordinates in an array
PImage img;

// buttons take x,y and width,height:
ZoomButton out = new ZoomButton(5, 5, 14, 14, false);
ZoomButton in = new ZoomButton(22, 5, 14, 14, true);
PanButton up = new PanButton(14, 25, 14, 14, UP);
PanButton down = new PanButton(14, 57, 14, 14, DOWN);
PanButton left = new PanButton(5, 41, 14, 14, LEFT);
PanButton right = new PanButton(22, 41, 14, 14, RIGHT);

boolean isGPSActive = false;

// all the buttons in one place, for looping:
Button[] buttons = {
  in, out, up, down, left, right
};

PFont font;
```

```
boolean gui = true;

void setup() {

  println("Processing Program For UMV");
  println();

  size(1280, 800);
  smooth();

  setupSerialPort();

  dump.add( new PVector(static_lat,static_lng,0));
  println(dump);

  output = createWriter("/data/GPS Co-ordinates.txt");
  // create a new map, optionally specify a provider
  map = new InteractiveMap(this, new Microsoft.RoadProvider());
  // others would be "new Microsoft.HybridProvider()" or "new Microsoft.AerialProvider()"
  // the Google ones get blocked after a few hundred tiles
  // the Yahoo ones look terrible because they're not 256px squares :)

  // set the initial location and zoom level to a predifined location:
  Location static_location = new Location(static_lat, static_lng);
  map.setCenterZoom(static_location, 13);

  // zoom 0 is the whole world, 19 is street level
  // (try some out, or use getlatlon.com to search for more)

  // set a default font for labels
  font = createFont("Helvetica", 12);
```

```
  // enable the mouse wheel, for zooming
  addMouseWheelListener(new java.awt.event.MouseWheelListener() {
    public void mouseWheelMoved(java.awt.event.MouseWheelEvent evt) {
      mouseWheel(evt.getWheelRotation());
    }
  }
  );

  // Saving Images
  path= sketchPath("/gps images");    // Get the path for gps images folder
  while (imgexist) {
    imgexist= new File(path + "/GPS" + imgcount + ".png").exists(); // **Use image path for
continuous image counting
    imgcount++;
  }
  imgcount--;

}

void draw() {
  background(0);
  fill(255);
  rect(0, 0, 1400, 800);

  // draw the map:
  map.draw();
  // (that's it! really... everything else is interactions now)

  //get the latest coordinate from GPS Device
  getLatestCoordinates();
```

```
smooth();

// draw all the buttons and check for mouse-over
boolean hand = false;
if (gui) {
  for (int i = 0; i < buttons.length; i++) {
    buttons[i].draw();
    hand = hand || buttons[i].mouseOver();
  }
}

// if we're over a button, use the finger pointer
// otherwise use the cross

cursor(hand ? HAND : CROSS);

// see if the arrow keys or +/- keys are pressed:
// (also check space and z, to reset or round zoom levels)

if (gui) {
  textFont(font, 12);

  // grab the lat/lon location under the mouse point:
  Location location = map.pointLocation(mouseX, mouseY);

  // draw the mouse location, bottom left:
  fill(0);
  noStroke();
  rect(5, height-5-g.textSize, textWidth("mouse: " + location), g.textSize+textDescent());
  fill(255, 255, 0);
  textAlign(LEFT, BOTTOM);
```

```
text("mouse: " + location, 5, height-5);

  // grab the center
location = map.pointLocation(width/2, height/2);

// draw the center location, bottom right:
fill(0);
noStroke();
float rw = textWidth("map: " + location);
rect(width-5-rw, height-5-g.textSize, rw, g.textSize+textDescent());
fill(255, 255, 0);
textAlign(RIGHT, BOTTOM);
text("map: " + location, width-5, height-5);

if (isGPSActive) {
  img=loadImage("marker.png");
  location = new Location(lat, lng);
  output.println("Latitude: " + lat + " Longitude: " + lng );
  Point2f p = map.locationPoint(location);
  image(img, p.x-10, p.y-34);
  dump.add( new PVector(lat,lng,0));
   plot();
}
else if (!isGPSActive) {
  // show marker at point of interest
  img=loadImage("marker.png");
  location = new Location(static_lat, static_lng);
  output.println("Latitude: " + static_lat + " Longitude: " + static_lng );
  Point2f p = map.locationPoint(location);

  //fill(0,255,128);
```

```
    // stroke(255,255,0);
     //ellipse(p.x, p.y, 10, 10);
    image(img, p.x-10, p.y-34);

     dump.add( new PVector(static_lat,static_lng,0));
      plot();
   }
  }

  // println((float)map.sc);
  // println((float)map.tx + " " + (float)map.ty);
  // println();
}

void keyReleased() {
  if (key == 'g' || key == 'G') {
   gui = !gui;
  }
  else if (key == 's' || key == 'S') {
   // Check for the existing image numbers and increment till last image number is found
   /*  while (imgexist) {
    imgexist= new File("C:/Users/Ashu/Desktop/interactive_maps/gps images/GPS" +
imgcount + ".png").exists();
    imgcount++;
    }
    imgcount--;*/
   save(path + "/GPS" + imgcount + ".png");
   imgcount++;
  }
  else if (key == 'z' || key == 'Z') {
   map.sc = pow(2, map.getZoom());
```

```
  }
  else if (key == ' ') {
    map.sc = 2.0;
    map.tx = -128;
    map.ty = -128;
  }
}

// see if we're over any buttons, otherwise tell the map to drag
void mouseDragged() {
  boolean hand = false;
  if (gui) {
    for (int i = 0; i < buttons.length; i++) {
      hand = hand || buttons[i].mouseOver();
      if (hand) break;
    }
  }
  if (!hand) {
    map.mouseDragged();
  }
}

// zoom in or out:
void mouseWheel(int delta) {
  if (delta > 0) {
    map.sc *= 1.05;
  }
  else if (delta < 0) {
    map.sc *= 1.0/1.05;
  }
}
```

```
// see if we're over any buttons, and respond accordingly:
void mouseClicked() {
  if (in.mouseOver()) {
    map.zoomIn();
  }
  else if (out.mouseOver()) {
    map.zoomOut();
  }
  else if (up.mouseOver()) {
    map.panUp();
  }
  else if (down.mouseOver()) {
    map.panDown();
  }
  else if (left.mouseOver()) {
    map.panLeft();
  }
  else if (right.mouseOver()) {
    map.panRight();
  }
}

// see if the arrow keys or +/- keys are pressed:
// (also check space and z, to reset or round zoom levels)
void keyPressed() {
  if (key == CODED) {
    if (keyCode == LEFT) {
      map.tx += 5.0/map.sc;
    }
    else if (keyCode == RIGHT) {
```

```
    map.tx -= 5.0/map.sc;
   }
   else if (keyCode == UP) {
    map.ty += 5.0/map.sc;
   }
   else if (keyCode == DOWN) {
    map.ty -= 5.0/map.sc;
   }
  }
  else if (key == 'c' || key == 'C') {
   map.setCenterZoom(new Location(lat, lng), 13);
  }
  else if (key == '+' || key == '=') {
   map.sc *= 1.05;
  }
  else if (key == '_' || key == '-' && map.sc > 2) {
   map.sc *= 1.0/1.05;
  }
}

void plot() {
  noFill();
  stroke(255,0,0);
  beginShape();
  for (PVector p : dump) {
   Location location = new Location (p.x,p.y);
   Point2f pos = map.locationPoint(location);
   ellipse(pos.x, pos.y, 10, 10);
   vertex(pos.x,pos.y);
   }
  endShape();
```

```
  stroke(0);
}

void exit() {
  output.flush();
  super.exit();
}

//-------------------------------------Buttons----------------------------------------//

class Button {

  float x, y, w, h;
  Button(float x, float y, float w, float h) {
    this.x = x;
    this.y = y;
    this.w = w;
    this.h = h;
  }

  boolean mouseOver() {
    return (mouseX > x && mouseX < x + w && mouseY > y && mouseY < y + h);
  }

  void draw() {
    stroke(80);
    fill(mouseOver() ? 255 : 220);
    rect(x,y,w,h);
  }
}
```

```
class ZoomButton extends Button {

  boolean in = false;

  ZoomButton(float x, float y, float w, float h, boolean in) {
    super(x, y, w, h);
    this.in = in;
  }

  void draw() {
    super.draw();
    stroke(0);
    line(x+3,y+h/2,x+w-3,y+h/2);
    if (in) {
      line(x+w/2,y+3,x+w/2,y+h-3);
    }
  }
}

class PanButton extends Button {

  int dir = UP;
  PanButton(float x, float y, float w, float h, int dir) {
    super(x, y, w, h);
    this.dir = dir;
  }

  void draw() {
    super.draw();
    stroke(0);
    switch(dir) {
```

```
      case UP:
        line(x+w/2,y+3,x+w/2,y+h-3);
        line(x-3+w/2,y+6,x+w/2,y+3);
        line(x+3+w/2,y+6,x+w/2,y+3);
        break;
      case DOWN:
        line(x+w/2,y+3,x+w/2,y+h-3);
        line(x-3+w/2,y+h-6,x+w/2,y+h-3);
        line(x+3+w/2,y+h-6,x+w/2,y+h-3);
        break;
      case LEFT:
        line(x+3,y+h/2,x+w-3,y+h/2);
        line(x+3,y+h/2,x+6,y-3+h/2);
        line(x+3,y+h/2,x+6,y+3+h/2);
        break;
      case RIGHT:
        line(x+3,y+h/2,x+w-3,y+h/2);
        line(x+w-3,y+h/2,x+w-6,y-3+h/2);
        line(x+w-3,y+h/2,x+w-6,y+3+h/2);
        break;
    }
  }
}

//---------------------------------Serial Event---------------------------------------//
/*
 * Setup the serial communication with Arduino
 */

void setupSerialPort() {
  // List all the available serial ports:
```

```
  println(Serial.list());
  myPort = new Serial(this, Serial.list()[0], 9600);
}

void getLatestCoordinates() {
  while (myPort.available() > 0) {
    String inBuffer = myPort.readString();
    String[] op = split(inBuffer, ',');
    //only when we get valid data, split it to get coordinates
    if(op.length==3) {
        String lat1 = op[1];  //get latitude
        String[] temp=splitTokens(lat1);
        lat = float(temp[1]);
        lng = float(op[2]);  //get longitude
        isGPSActive = true;
    }
    delay(200);
  }
}
```