ISLAMIC UNIVERSITY OF TECHNOLOGY (IUT)

ORGANISATION OF ISLAMIC CONFERENCE (OIC)

DESIGN, CONSTRUCTION AND MANUFACTURING OF THE CONTROL SYSTEM OF AN UNDERWATER ROBOT

A thesis submitted to the department of Mechanical and Chemical Engineering (MCE), Islamic University of Technology (IUT), in the partial fulfillment of the requirement for the degree of Bachelor in Science in Mechanical Engineering.

Prepared By

MD. SHAHRIAR ISLAM (091407)

Supervised by

Dr. Professor Md. Nurul Absar Chowdhury

Department of Mechanical and Chemical Engineering

Islamic University of Technology (IUT)

The Organization of Islamic Cooperation (OIC)

Declaration:

This is to certify that the work presented in the thesis is an outcome of the experiment and research carried out by the author under the supervision of

Dr. Professor Md. Nurul Absar Chowdhury.

Signature of the Candidate

____

Md. Shahriar Islam (Author)

Date:

Signature of the Supervisor

____

Dr. Professor Md. Nurul Absar Chowdhury

(Project Supervisor)

Date:

ACKNOWLEDGEMENT

All praises to Almighty ALLAH (Sbw) Who blessed me with potential and patience to complete this project successfully overcoming the obstacles in the way.

It is not sufficient to only give thanks to some people who helped and guided me when needed and without them the thesis could never come into being successfully in time. My gratitude and respect to my supervisor **Dr. Professor Md. Nurul Absar Chowdhury** , MCE Department, Islamic University of Technology (IUT) for his encouragement, erudite advices and extreme patience throughout the course for establishing such a creative project. His care, concern and continuous moral support for me are appreciated.

I am also grateful to the Head of Mechanical and Chemical Engineering Department Dr. Md. Abdur Razzaq Akhanda for providing us with necessary funding and supports.

Although I tried my best to complete this thesis flawlessly, I seek apology if there is any mistake found in this report**.**

Abstract

In our time we have seen a lot of development in the design and development in the unmanned underwater vehicle for civil and military purposes. A great array of vehicle types and applications has been produced along with wide range of innovative approaches for enhancing the performance of underwater robot. Key technology advances in the relevant area include battery technology, fuel cells, underwater communications, and propulsion system and sensor fusion. These recent advances enable the extension of underwater robot’s flight envelop comparable to that of manned vehicles. For undertaking longer missions, therefore more advanced control and navigation will be required to maintain an accurate position over larger operational envelop particularly when a close proximity to obstacles(such as manned vehicles, pipelines and underwater structure) is involved. In this case a sufficient good model is prerequisite of control system design. The thesis is focused on discussion on advanced cheap design from modeling, control and guidance perspectives. Lessons learned from recent achievements.

TABLE OF CONTENTS

01. WHY USE UNMANNED UNDERWATER ROBOT

Underwater robots are useful because they can accomplish tasks that are dangerous, uncomfortable, or tedious for humans to perform. The underwater environment is intrinsically unfriendly to human life, with a lack of breathable oxygen, high pressure not encountered in the atmosphere, and at times in tolerably cold temperature such diving remains hazardous occupation where air supply and decompression must be carefully monitored, with potentially lethal consequences. Submarines shield the occupant from the surrounding environment but require their own ventilation systems and buoyancy control, on which human life depends. Aside from the intelligence that the operator provides, and perhaps a certain sense of adventure, there is no need to include human on board.

Danger of a manned system aside, life support systems on a submarine greatly complicate the system design, since they must control the human temperature, have a large pressure vessel and a breathable gas composition. The safety margin of the design must also be much larger, given the consequence of a failure.

Another consideration is the overall effort in completing an underwater task. For a few exploratory dives, it may not be a burden to send a trained crew. However, keeping a near constant underwater presence, or operating a fleet of vehicles, would require a large number of skilled operators. Some laborious task such as fixed survey patterns, are better done by never tiring computer than humans.by removing the human presence and giving the vehicles a degree of autonomy, the effectiveness of the vehicles may be greatly increased, and the tedium to human reduced.

02. INTRODUCTION

Underwater vehicles are all types of underwater robots which are operated with minimum or without intervention of human operator. The phrase is used to describe both a remotely operated vehicle (ROV) and an autonomous underwater vehicle (AUVs).remotely operated vehicles are tele-operated robots that are deployed primarily for underwater installation, inspection and repair tasks. They have been used extensively in offshore industries due to their advantages over human driver s in term s of higher safety, greater depths, longer endurance and less demand for support equipment.in its operation, the ROV receives operation on board a surface ship or other mooring platform through tethered cable or acoustic link. AUVs on the other hand operate without the need of the constant monitoring and supervision from a human operator. As such the vehicle do not have the limiting factor in its operating range from the cable typically associated with ROVs this enables AUVs to be used for certain types of mission such as a long range oceanographic data collection where the use of ROVs deemed impractical.

Ura t.,AUV ‘r2d4’, its operation and roadmap for AUV development, in :Advances in Unmanned Marine Vehicles, edited by G.N.Robert&r. Sutton(IEE control series69)

Proposed classification of AUVs area of application into3 different categories starting from the basic to more advanced missions:

a) Operations at a safe distance from the sea floor including observation of the sea floor using sonar, examination of water composition, sampling of floating creatures.

b) Inspection in close proximity to the sea floor and manmade structure such as inspection of hydrothermal activity, creatures on the seafloor and underwater structures.

C) Interactions with the sea floor and manmade structures, i.e. sampling of substance on the sea floor and drilling.

Various control techniques have been proposed for UUVs BOTH IN SIMULATION ENVIRONMENT and actual in water experiments from the year 1990 onwards.

03. Background: science and technology

a. History of unmanned underwater development

The conceptual design for submarine was dated back as early 1578.the first modern UUV was constructed in the form of a self-propelled torpedo in 1868.During the year 1958 us navy instigated the cable control underwater vehicle program as the precursor of ROV.THE use of commercial UUV was recognized owing to primarily the onset of the offshore oil and gas major operation. The use of AUVs in the meantime only gradually gains acceptance both for naval and commercial sectors due to more stringent operational requirements. The rapid development in underwater sensors, battery and other supporting technologies, the development of AUV HAVE GAINED acceleration in recent decades. There were more than 46 AUV models in 1999 and according to a survey in 2004, about 240 AUVs, ranging from 10kg to 10tons in weight and several meters to 6000 meter in operational depth, were in operation at different sea locations in the world.

The offshore survey industry uses AUVs for detailed mapping of the sea floor, allowing oil companies to carry out construction and maintenance of underwater structures in the most effective manner and with mini mum disruption to the environment. The maintenance mission typically requires a combination of sub bottom profilers, visual sensors, and extensive onboard processing .military application for an AUV includes the mapping of an area for mine detection purposes and undersea resupply of foodstuff, fuel, and ammunition. Scientists deploy AUVs to study ocean and ocean floor using INS, side scan sonar, multi beam echo sounders, magnetometers, thermistors, and other underwater sensors including AD©PS AND WATER QUALITY SENSORS. Contemporary AUVs with their corresponding maximum operational depth and speed are depicted in fig 1.

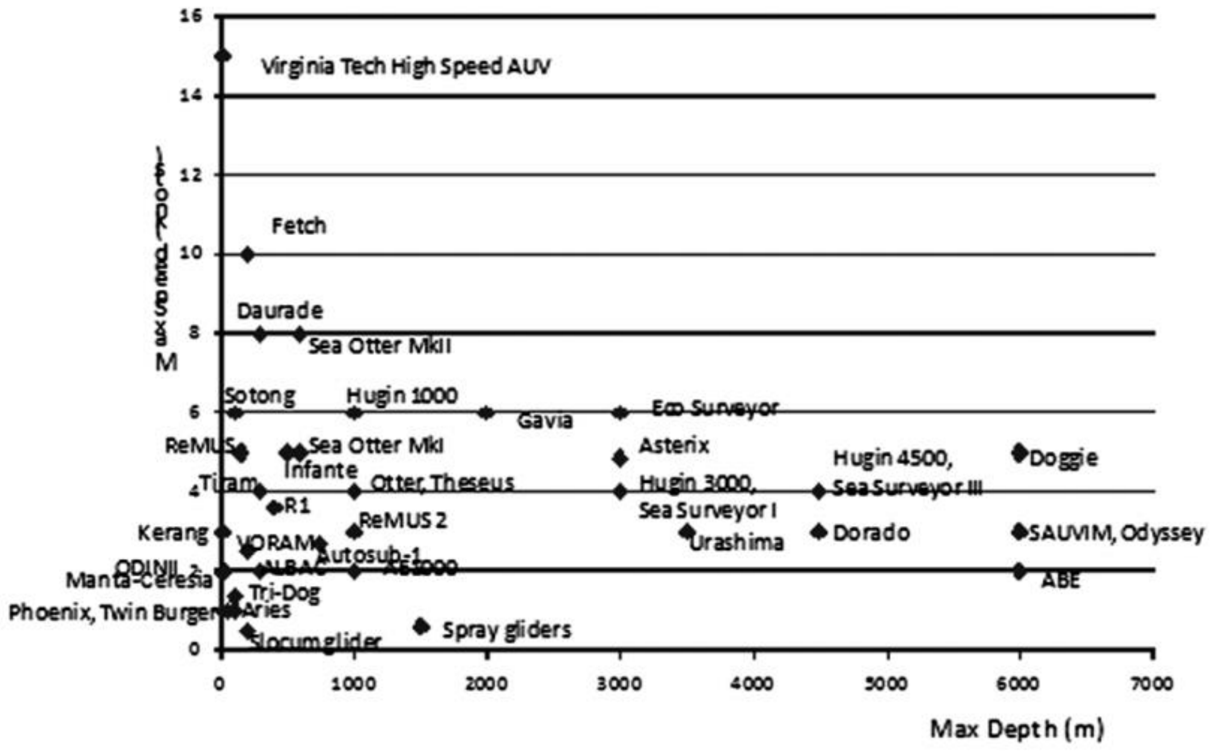


Fig. 1—Representative AUVs with their maximum operational depth and speed [22-34]

b. DESIGN PHILOSOPHY:

For any underwater robot to be adopted into practical use, it must be more favorable to use than comparable, existing methods of accomplishing the same goal. Depending on the mission, these methods might be using human divers, a sensor towed by a ship, a manned submarine or other UUVs.

One measurement of advantage is cost effectiveness. A technological improvement only has relevance if it contributes to a more efficient way of completing a goal, excepting circumstances where nothing else can complete that goal. A UUV that can work twice the work of a human diver is a difficult sell to end users if the total cost of operation is ten times that of a diver. To be cost effective, the design must use mass-produced, commercially available components as much as possible, and any custom parts must be optimized for manufacturing efficiency.

The total cost of operation involves not just the vehicle, but the support equipment used to operate it. AUUV might itself be relatively small and inexpensive, but if it require a large support vessel with a crane for deployment, the operation cost will high. Consumable elements that must be replaced after every mission, such as primary batteries, are also unfavorable for this reason. Ideally, a UUV could be lifted and transported by hand, and deployed by setting it off the side of a dock or ship, without special equipment.

Simplicity is an important virtue in almost any design.it reduces cost if there are few parts to be made and fewer features for each part. It also makes assembly and maintenance simpler if there are fewer steps in each procedure. Perhaps most importantly, the fragility of a system increases as the number of independent parts increases, and the space of potential failure grows. Thoughtful minimization at the outset of the design saves much cost, production, assembly, and repair throughout the rest of the project.

At a minimum it should be possible to add a payload, both on exterior of the vehicle and inside a dry pressure housing, without undue reconfiguration of the vehicle. To meet this need, the design should have substantial dry payload space, extra penetrators to pass wiring through the pressure housing, and a frame that can be fitted with clamps or brackets for external payloads.

Finally, operating the vehicle should be made as simple as possible. If turning on and deploying the vehicle requires a team of engineers and programmers, the number of practical application is quite limited. However, if the interface is simplified to be more like that of a remote control toy, any technician with basic computer skills can be trained to run it within a few hours. This makes widespread, cost effective adoption of the UUV possible.

04. EXISTING UUV TECHNOLOGIES AND DESIGN CONSTRAINTS

a. UUV COMMUNICATION

For many applications, accurate navigation information is essential. For example, measurement of water quality is not useful to a scientist without the locations of the samples. Searching or mapping an area is very difficult without being able to navigate. While solutions to underwater navigation exists, they constitute a large part of the expense of the vehicle.

At the water's surface, it is possible to use GPS for inexpensive, highly accurate position data. Commercial GPS receivers are presently in the range of costs, and provide absolute position, worldwide to within 1m or so. For most purposes, this accuracy in navigation is more than adequate. However, since GPS radio frequency signals do not carry through water, submerged navigation becomes a separate and challenging problem.

Other submerged navigation system recreate a system like GPS under the water, using acoustic rather than electromagnetic waves. Transducers may be permanently mounted to know locations the sea floor, or deployed on buoys or ships with GPS, to act position references. With a transducer mounted on the vehicle, many navigation schemes are possible, such as exploiting time delays from multiple sources, as long base line (LBL) navigation. Alternately, it can be done using delay and phase information, as ultra-short base line (USBL) navigation systems. These systems provide an absolute reference, and remove the drift in measurements from dead reckoning. The accuracy of acoustic navigation system can be as good as meter over a range of kilometers, under ideal conditions. However, the cost is high. Also, they may require a support ship to stay nearby, or the set up and removal of beacons, inconveniencing operation. Acoustic navigation systems are also strongly affected by the characteristics of the environment.

ROV navigation is often as informal as the operator navigating by visual landmarks, or rough dead reckoning using a compass and estimated vehicle speed. Navigation typical of AUVs has also been done with ROVs, including DVL and acoustic based systems.

Because the state of art underwater navigation systems are expensive, and have inherent accuracy limitations, they suggest a possibility for an improved system design.

b. UNMANNED UNDERWATER ROBOT COMMUNICATION

Communication between the operator and the UUV is necessary for control of the vehicle, initiating preprogrammed missions or behaviors, monitoring the vehicles' progress and relaying accurate data. The challenges posed by communication are a significant part of the cost and complexity of the UUV system design and may impose limitation on how the UUV may be operated.

ROV use a tethered to enable live, high bandwidth communication. Sending high definition video is a common capability of electronic or fiber optic tethers. However, a tether between the operator and vehicle adds operational difficulty. The vehicle must operate close to shore, or a support vessel must be moved with ROVs the drag force on the tether becomes prohibitive at some point.

A common untethered approach is to use radio communication when the UUV is at the surface to retrieve data from the last mission, and queue up the next one. Radio communication saves the difficulty of retrieving and deploying the vehicle but does not work when submerged. Thus it cannot be used to change the course of the vehicle during a mission, or to give feedback on the mission's progress.

The state of the art in untethered UUV communication is through acoustic modems. With transducers at the operator and vehicle sides, data may be acoustically transmitted during the mission. While acoustic modems are common and proven to be useful, and can have a range of kilometers, they have limitations. First, the cost is high for an inexpensive UUV, typically at around $15000 FOR A PAIR OF MODEMS. Also the bandwidth is low compared to radio or tethered communication, limited by the distance that high frequency sound can travel in water. A typical data rate might be in the hundreds or thousands of bits per second, which is enough to send high level command or small amount of sensory data, but not video, streaming sonar images, or other high bandwidth applications.

05. THE INTENDED APPLICATION OF OUR UNDERWATER ROBOT

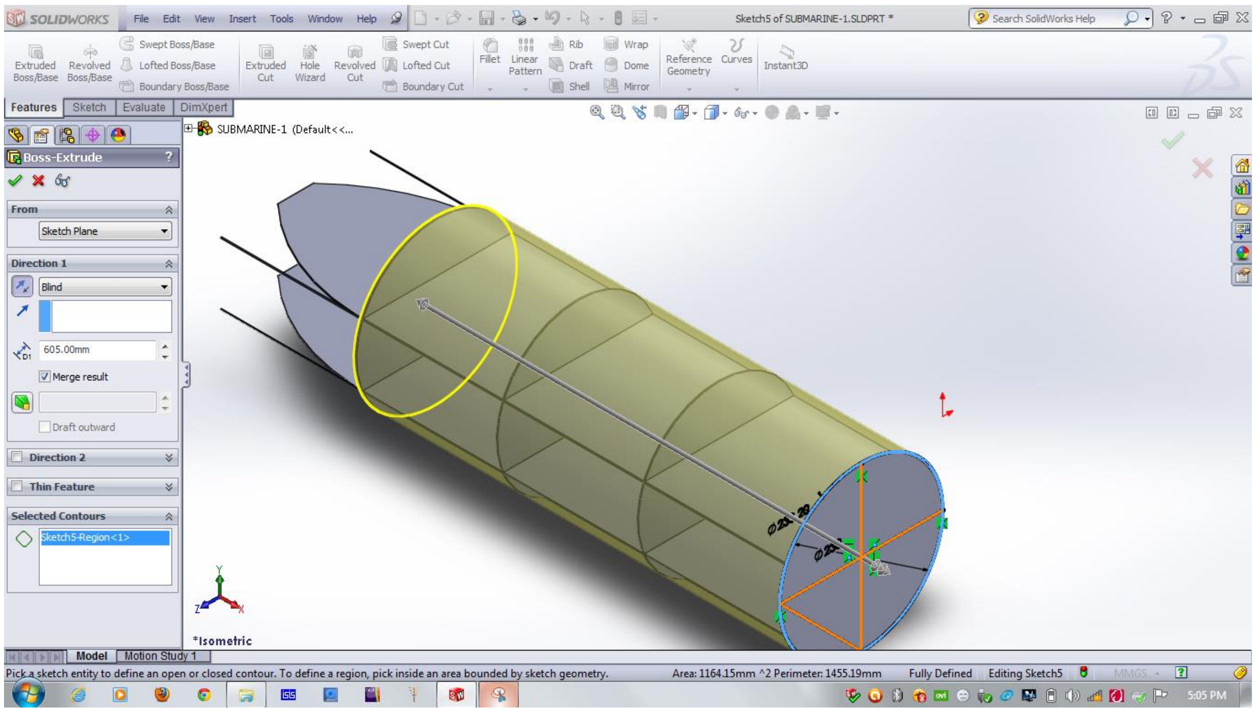
- To Collect sample water from lake, pond, river to study the mixture and combination of water deep under water.
- To transport necessary liquid under water for certain application.
- To help develop the navigation under water.

Besides educational purposes, a UUV with video recording and space for water quality sensors has many scientific uses. Presently many surveys of bottom type, biological data, and water quality are done by tedious means. Reef biology researchers describe taking a boat to various locations by GPS free-diving once anchored, and then logging observations on a waterproof clipboard.

The robot can have harbor security applications. Inspecting the sides of ships hulls is a well-suited tasks to its vertically tethered configuration. The operator can be indoor on shore, not chasing the robot on boat, and can move between multiple ships searching the bottom of harbor is also a well suited task, as is necessary when vessels engaged in smuggling ditch their cargo from the underside when confronted.

Search in shallow water is another suitable application. Presently, searching a pond or lake is a time consuming task, employing skilled divers, or towed side scan sonar. If the water is murky, divers may only see a small region at a time, and have difficulty navigating. Without knowing what has been searched, it is hard to know when to stop searching. In cold water, a search entails much discomfort. While sonar can cover a greater area, and can be easy to deploy, it cannot image all objects, and can show many artifacts.

06. DESIGN OF THE UNDERWATER ROBOT:


Sketch5 of SUBMARINE-1.SLDPRT *

Boss-Extrude

From: Sketch Plane

Direction 1: Blind

605.00mm

☑ Merge result

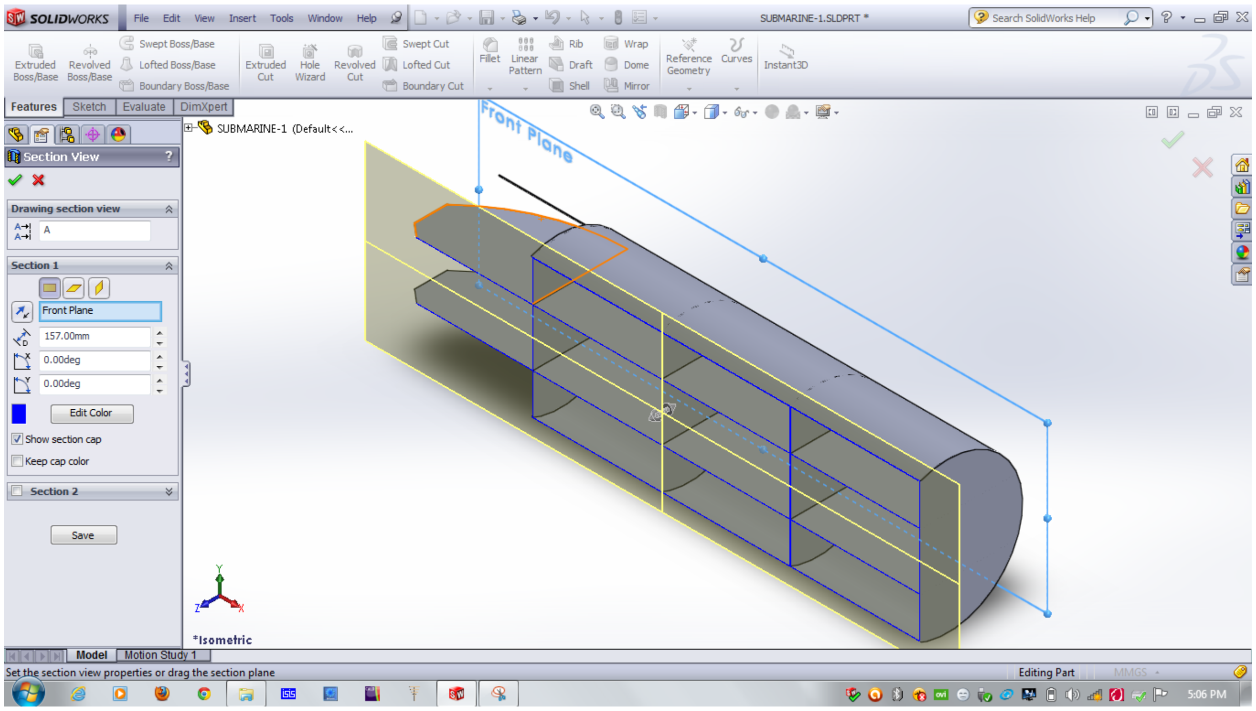
☐ Draft outward

Selected Contours: Sketch5-Region<1>

Pick a sketch entity to define an open or closed contour. To define a region, pick inside an area bounded by sketch geometry.

Area: 1164.15mm ^2 Perimeter: 1455.19mm Fully Defined Editing Sketch5

Figure : underwater robot's main body with internal structure 1


SUBMARINE-1.SLDPRT *

Section View

Drawing section view: A

Section 1: Front Plane

157.00mm

0.00deg

0.00deg

☑ Show section cap

☐ Keep cap color

Set the section view properties or drag the section plane

Editing Part

SOLIDWORKS
File Edit View Insert Tools Window Help
Features Sketch Evaluate DimXpert
SUBMARINE-1 (Default<<...
Front Plane
Edit Color
Section 2
Save
*Isometric
Model Motion Study 1
5:06 PM

Figure : underwater robot's main body with internal structure 2

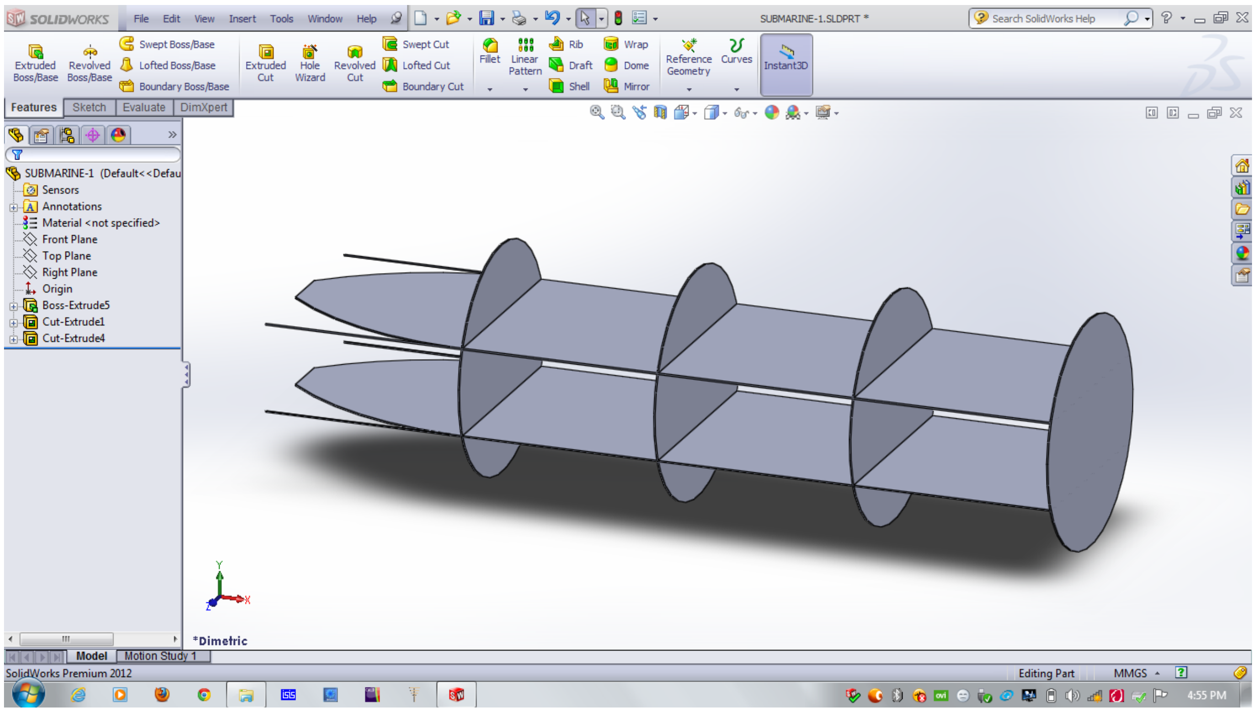


Figure : underwater robot's main body with internal structure 3

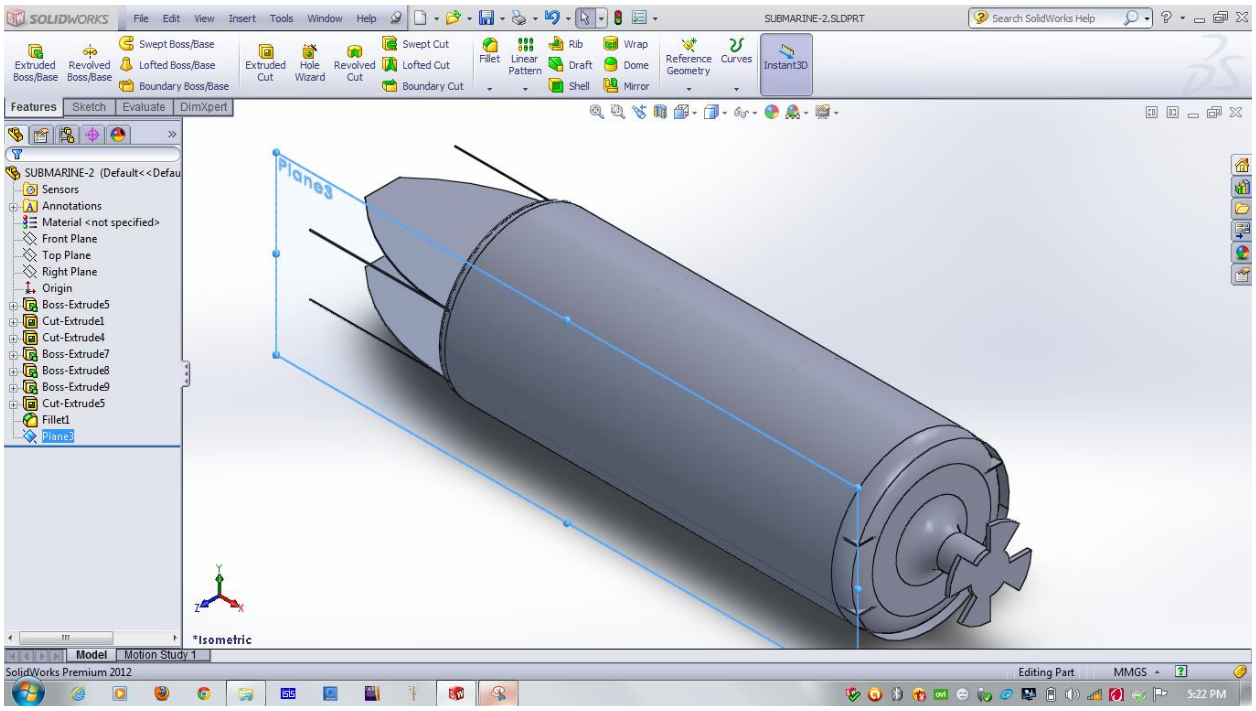


Figure: underwater robot's main body with driving propeller

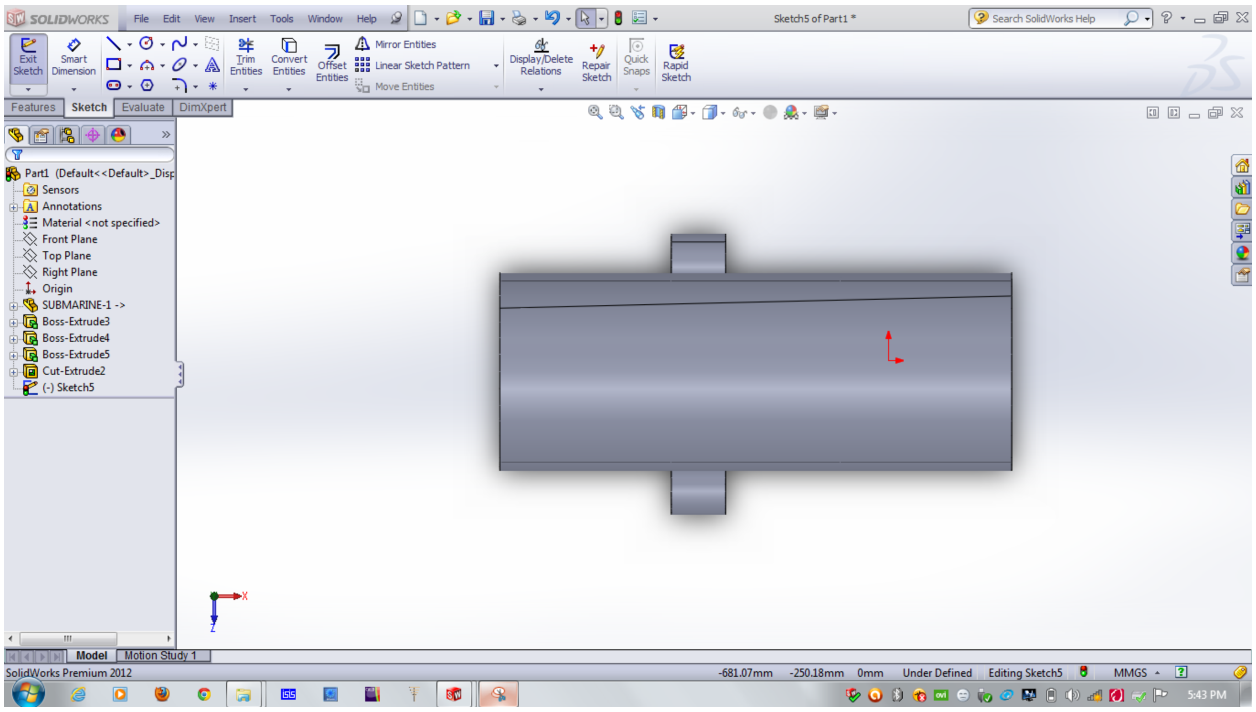


Figure: underwater robot's main body with two directional propeller

Background of the design:

The design idea of the robot came through studying different operational submarines around the world among them USS VIRGINIA, USS FLORIDA put real impression on this project.

07. CONTROL SYSTEM OF THE ROBOT

- 4 MOTORS

  1 DRIVING MOTOR

  2 DIRECTION CONTROL MOTOR

  1 MOTOR TO MOVE UP

- 2 PUMP

  1 PUMP TO COLLECT WATER IN ONE COMPARTMENT

  1 PUMP TO HELP DIVING

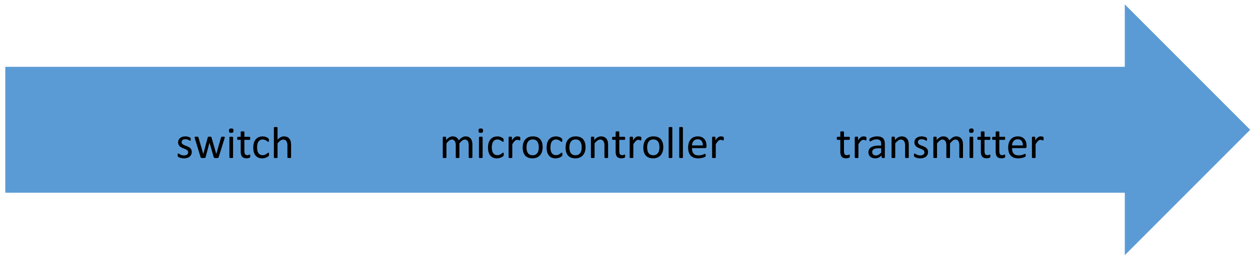
- **2 SECTIONS** of the control system

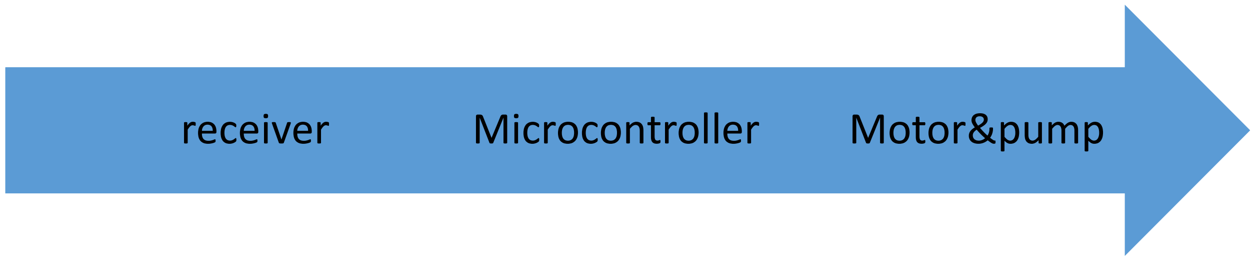
1. Command center
2. Command receive and control section.

   2 MICROCONTROLLER

1 microcontroller to process and send the command through a wireless transfer module and another microcontroller will receive the command by a receiver module and will process the command to control accordingly.

a. System continuity

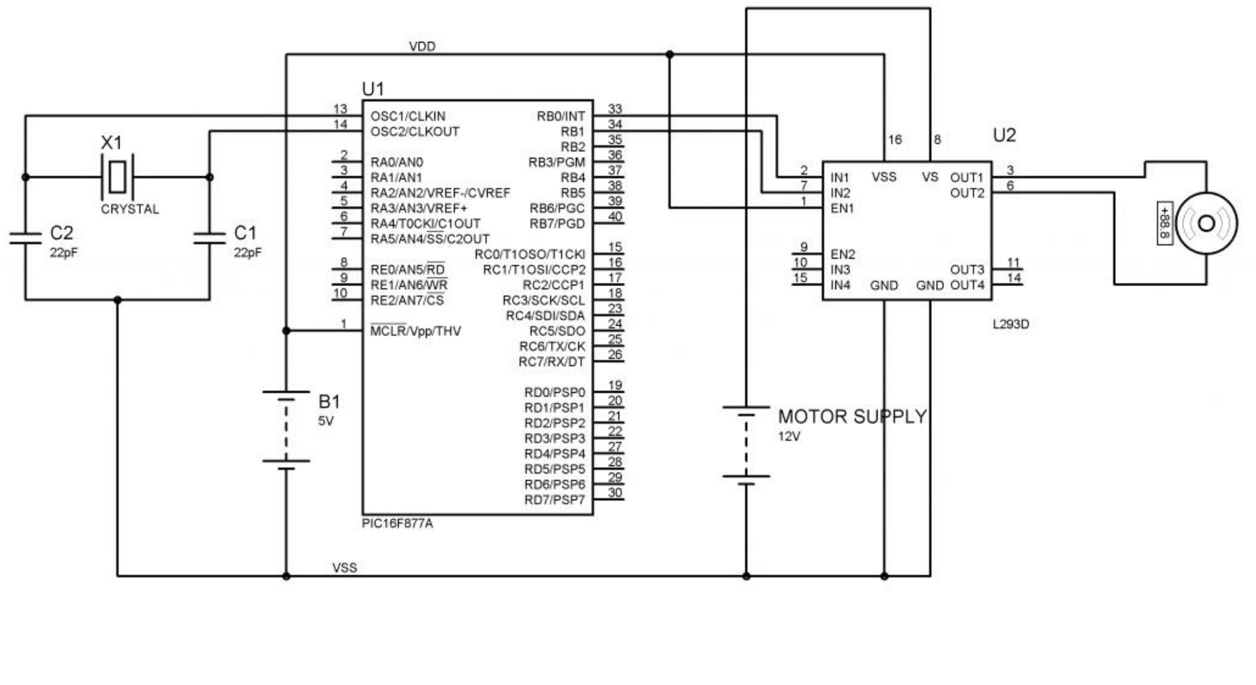




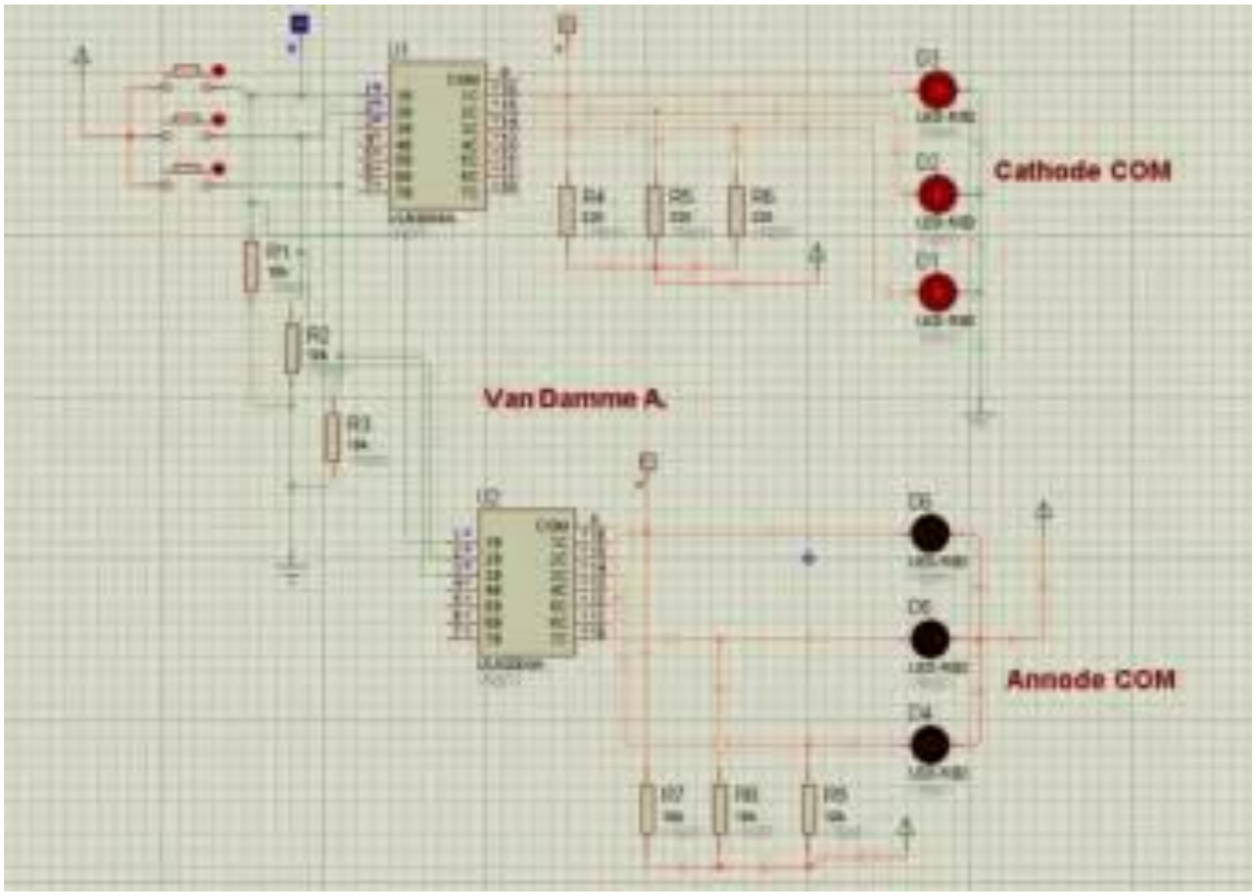
b. COMPONENTS

- 2 ARDUINO ATMEGA 2560
- 3 MOTOR SHIELD
- SPEED CONTRLO AND ON-OFF SWITCH
- Nrf MODULE (transfer and receiver)
- LEPO BATTERY (lithium ion battery)
- 4 MOTORS
- 2 PUMPS

Previously used system



c. Proteus simulation



Overview

The main theme of the Proteus 8 release is integration. Development has therefore been focused on taking the various discrete parts of an electronic design and coupling them together to achieve a better workflow. In order to achieve this, three major architectural changes were necessary; a unified application framework, a common database and a live netlist.

Common Database & Live Netlisting

The common database and live netlisting features provide system wide access to the properties of the parts and the connectivity between them. Features like pinswap, gateswap and annotation are both automatic and bi-directional between schematic and PCB and connectivity changes on the schematic can be automatically reflected in any other module (BOM, Design Explorer, ARES).

These features also lay the foundation for a number of development projects such as design snippets which we plan to bring forth during the lifetime of Proteus 8.

Proteus 8 stores the design (DSN), layout (LYT) and common database in a single project file (PDSPRJ).

d. ARDUINO ATMEGA 2560

Arduino is an open-source electronics prototyping platform based on flexible, easy-to-use hardware and software. It's intended for artists, designers, hobbyists, and anyone interested in creating interactive objects or environments.

Specifications

Operating Voltage	5V
Input Voltage (recommended)	7-12 V
Digital I/O Pins	54 (of which 15 provide PWM output)
Analog Input Pins	16
Flash Memory	256 KB
SRAM8 KB	
EEPROM	4 KB
Clock Speed	16 MHz

Overview

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

The Mega 2560 is an update to the Arduino Mega, which it replaces.

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

Power

The Arduino Mega can be powered via the USB connection or with an external power supply. The power source is selected automatically.

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

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

The power pins are as follows:

- **VIN.** The input voltage to the Arduino board when it's using an external power source (as opposed to 5 volts from the USB connection or other regulated power source). You can supply voltage through this pin, or, if supplying voltage via the power jack, access it through this pin.

• 5V. This pin outputs a regulated 5V from the regulator on the board. The board can be supplied with power either from the DC power jack (7 - 12V), the USB connector (5V), or the VIN pin of the board (7-12V). Supplying voltage via the 5V or 3.3V pins bypasses the regulator, and can damage your board. We don't advise it.

• 3V3. A 3.3 volt supply generated by the on-board regulator. Maximum current draw is 50 mA.

• GND. Ground pins.

• IOREF. This pin on the Arduino board provides the voltage reference with which the microcontroller operates. A properly configured shield can read the

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

Memory

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

Input and Output

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

- **Serial: 0 (RX) and 1 (TX); Serial 1: 19 (RX) and 18 (TX); Serial 2: 17 (RX) and 16 (TX); Serial 3: 15 (RX) and 14 (TX).** Used to receive (RX) and transmit (TX) TTL serial data. Pins 0 and 1 are also connected to the corresponding pins of the ATmega16U2 USB-to-TTL Serial chip.
- **External Interrupts: 2 (interrupt 0), 3 (interrupt 1), 18 (interrupt 5), 19 (interrupt 4), 20 (interrupt 3), and 21 (interrupt 2).** These pins can be configured to trigger an interrupt on a low value, a rising or falling edge, or a change in value.
- **PWM: 2 to 13 and 44 to 46.** Provide 8-bit PWM output with the analogWrite() function.
- **SPI: 50 (MISO), 51 (MOSI), 52 (SCK), 53 (SS).** These pins support SPI communication using the SPI library. The SPI pins are also broken out on the ICSP header, which is physically compatible with the Uno, Duemilanove and Diecimila.

- **LED: 13.** There is a built-in LED connected to digital pin 13. When the pin is HIGH value, the LED is on, when the pin is LOW, it's off.
- **TWI: 20 (SDA) and 21 (SCL).** Support TWI communication using the Wire library. Note that these pins are not in the same location as the TWI pins on the Duemilanove or Diecimila.

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

There are a couple of other pins on the board:

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

Communication

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

A SoftwareSerial library allows for serial communication on any of the Mega2560's digital pins.

The ATmega2560 also supports TWI and SPI communication. The Arduino software includes a Wire library to simplify use of the TWI bus.

Programming

The Arduino Mega can be programmed with the Arduino software.

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

You can also bypass the bootloader and program the microcontroller through the ICSP (In-Circuit Serial Programming) header.

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

- On Rev1 boards: connecting the solder jumper on the back of the board (near the map of Italy) and then resetting the 8U2.
- On Rev2 or later boards: there is a resistor that pulling the 8U2/16U2 HWB line to ground, making it easier to put into DFU mode. You can then use Atmel's FLIP software (Windows) or the DFU programmer (Mac OS X and Linux) to load a new firmware. Or you can use the ISP header with an external programmer (overwriting the DFU bootloader).

Automatic (Software) Reset

Rather than requiring a physical press of the reset button before an upload, the Arduino Mega2560 is designed in a way that allows it to be reset by software running on a connected computer. One of the hardware flow control lines (DTR) of the ATmega8U2 is connected to the reset line of the ATmega2560 via a 100 nanofarad capacitor. When this line is asserted (taken low), the reset line drops long enough to reset the chip. The Arduino software uses this capability to allow you to upload code by simply pressing the upload button in the Arduino environment. This means that the bootloader can have a shorter timeout, as the lowering of DTR can be well-coordinated with the start of the upload.

This setup has other implications. When the Mega2560 is connected to either a computer running Mac OS X or Linux, it resets each time a connection is made to it from software (via USB). For the following half-second or so, the bootloader is running on the Mega2560. While it is programmed to ignore malformed data (i.e. anything besides an upload of new code), it will intercept the first few bytes of data sent to the board after a connection is opened. If a sketch running on the board receives one-time configuration or other data when it first starts, make sure that the software with which it communicates waits a second after opening the connection and before sending this data.

The Mega2560 contains a trace that can be cut to disable the auto-reset. The pads on either side of the trace can be soldered together to re-enable it. It's labeled "RESET-EN". You may also be able to disable the auto-reset by connecting a 110 ohm resistor from 5V to the reset line.

USB Overcurrent Protection

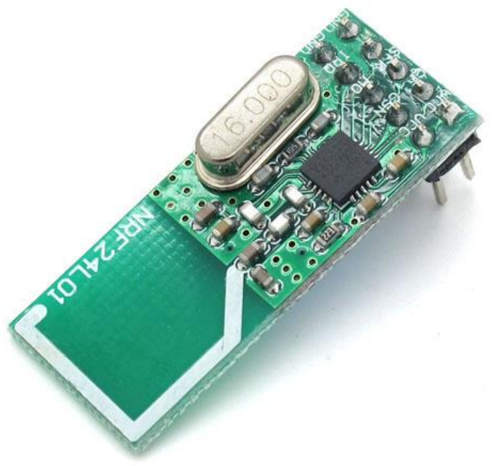
The Arduino Mega2560 has a resettable polyfuse that protects your computer's USB ports from shorts and overcurrent. Although most computers provide their own internal protection, the fuse provides an extra layer of protection. If more than 500 mA is applied to the USB port, the fuse will automatically break the connection until the short or overload is removed.

Physical Characteristics and Shield Compatibility

The maximum length and width of the Mega2560 PCB are 4 and 2.1 inches respectively, with the USB connector and power jack extending beyond the former dimension. Three 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.

The Mega2560 is designed to be compatible with most shields designed for the Uno, Diecimila or Duemilanove. Digital pins 0 to 13 (and the adjacent AREF and GND pins), analog inputs 0 to 5, the power header, and ICSP header are all in equivalent locations. Further the main UART (serial port) is located on the same pins (0 and 1), as are external interrupts 0 and 1 (pins 2 and 3 respectively). SPI is available through the ICSP header on both the Mega2560 and Duemilanove / Diecimila.

e. NRF24L01:



Features

Features of the nRF24L01 include:

· **Radio**

>worldwide 2.4GHz ISM band operation

>126 RF channels

>Common RX and TX pins

>GFSK modulation

>1 and 2Mbps air data rate

>1MHz non-overlapping channel spacing at 1Mbps

>2MHz non-overlapping channel spacing at 2Mbps

·**Transmitter**

＞Programmable output power: 0, -6, -12 or -18dBm

＞11.3mA at 0dBm output power

·**Receiver**

＞Integrated channel filters

＞12.3mA at 2Mbps

＞82dBm sensitivity at 2Mbps

＞-85dBm sensitivity at 1Mbps

＞Programmable LNA gain

·**RF Synthesizer**

＞Fully integrated synthesizer

＞No external loop filer, VCO varactor diode or resonator

＞Accepts low cost ±60ppm 16MHz crystal

·**Enhanced ShockBurst™**

＞1 to 32 bytes dynamic payload length

＞Automatic packet handling

＞Auto packet transaction handling

＞6 data pipe MultiCeiver™ for 1:6 star networks

·**Power Management**

>Integrated voltage regulator

>1.9 to 3.6V supply range

>Idle modes with fast start-up times for advanced power management

>22uA Standby-I mode, 900nA power down mode

>Max 1.5ms start-up from power down mode

>Max 130us start-up from standby-I mode

·**Host Interface**

>4-pin hardware SPI

>Max 8Mbps

>3 separate 32 bytes TX and RX FIFOs

>5V tolerant inputs

·**Compact 20-pin 4x4mm QFN package**

Introduction of nRF24L01

The nRF24L01 is a single chip 2.4GHz transceiver with an embedded baseband protocol engine (Enhanced ShockBurst™), designed for ultra-low power wireless applications. The nRF24L01 is designed for operation in the world wide ISM frequency band at 2.400 - 2.4835GHz. An MCU (microcontroller) and very few external passive components are needed to design a radio system with the nRF24L01.The nRF24L01 is configured and operated through a Serial Peripheral Interface (SPI.) Through this inter-face the register map is available. The register map contains all configuration registers in the nRF24L01 and is accessible in all operation modes of the chip.

The embedded baseband protocol engine (Enhanced ShockBurst™) is based on packet communication and supports various modes from manual operation to advanced autonomous protocol operation. Internal FIFOs ensure a smooth data flow between the radio front end and the system’ s MCU. Enhanced Shock-Burst ™ reduces system cost by handling all the high-speed link layer operations.

The radio front end uses GFSK modulation. It has user configurable parameters like frequency channel, output power and air data rate.

The air data rate supported by the nRF24L01 is configurable to 2Mbps. The high air data rate combined with two power saving modes makes the nRF24L01 very suitable for ultra-low power designs. Internal voltage regulators ensure a high Power Supply Rejection Ratio (PSRR) and a wide power supply range.

Radio Control

This chapter describes the different modes the nRF24L01 radio transceiver can operate in and the param-eters used to control the radio.The nRF24L01 has a built-in state machine that controls the transitions between the different operating modes of the chip. The state machine takes input from user defined register values and internal signals.

Operational Modes

The nRF24L01 can be configured in four main modes of operation. This section describes these modes.

1 State diagram

The nRF24L01 is undefined until the **VDD** becomes 1.9V or higher. When this happens nRF24L01 enters the Power on reset state where it remains in reset until it enters the Power Down mode. Even when the nRF24L01 enters Power Down mode the MCU can control the chip through the SPI and the Chip Enable (**CE**) pin. Three types of states are used in the state diagram. “Recommended operating mode” is a state that is used during normal operation. “Possible operating mode” is a state that is allowed to use, but it is not used during normal operation. “Transition state” is a time limited state used during startup of the oscillator and settling of the PLL.

2 Power down Mode

In power down mode nRF24L01 is disabled with minimal current consumption. In power down mode all the register values available from the SPI are maintained and the SPI can be activated.

3 Standby Modes

By setting the PWR_UP bit in the CONFIG register to 1, the device enters standby-I mode. Standby-I mode is used to minimize average current consumption while maintaining short start up times. In this mode part of the crystal oscillator is active. This is the mode the nRF24L01 returns to from TX or RX mode when **CE** is set low.

In standby-II mode extra clock buffers are active compared to standby-I mode and much more current is used compared to standby-I mode. Standby-II occurs when **CE** is held high on a PTX device with empty TX FIFO. If a new packet is uploaded to the TX FIFO, the PLL starts and the packet is transmitted

The register values are maintained during standby modes and the SPI may be activated. For startup time

3 RX mode

The RX mode is an active mode where the nRF24L01 radio is a receiver. To enter this mode, the

nRF24L01 must have the PWR_UP bit set high, PRIM_RX bit set high and the **CE** pin set high.In this mode the receiver demodulates the signals from the RF channel, constantly presenting the demod-ulated data to the baseband protocol engine. The baseband protocol engine constantly searches for a valid packet. If a valid packet is found (by a matching address and a valid CRC) the payload of the packet is presented in a vacant slot in the RX FIFO. If the RX FIFO is full, the received packet is discarded.

The nRF24L01 remains in RX mode until the MCU configures it to standby-I mode or power down mode. If the automatic protocol features (Enhanced ShockBurst^{TM}) in the baseband protocol engine are enabled, the nRF24L01 can enter other modes in order to execute the protocol.

In RX mode a carrier detect signal is avaliable. The carrier detect is a signal that is set high when a RF sig-nal is detected inside the receiving frequency channel. The signal must be FSK modulated for a secure detection. Other signals can also be detected. The Carrier Detect (CD) is set high when an RF signal is detected in RX mode, otherwise CD is low. The internal CD signal is filtered before presented to CD register.

5 TX mode

The TX mode is an active mode where the nRF24L01 transmits a packet. To enter this mode, the nRF24L01 must have the PWR_UP bit set high, PRIM_RX bit set low, a payload in the TX FIFO and, a high pulse on the **CE** for more than 10µs.

The nRF24L01 stays in TX mode until it finishes transmitting a current packet.

If **CE**= 0 nRF24L01 returns to standby-I mode.

If **CE**= 1, the next action is determined by the status of the TX FIFO. If the TX FIFO is not empty the nRF24L01 remains in TX mode, transmitting the next packet. If the TX FIFO is empty the nRF24L01 goes into standby-II mode.The nRF24L01 transmitter PLL operates in open loop when in TX mode. It is important to never keep the nRF24L01 in TX mode for more than 4ms at a time. If the auto retransmit is enabled, the nRF24L01 is never in TX mode long enough to disobey this rule.

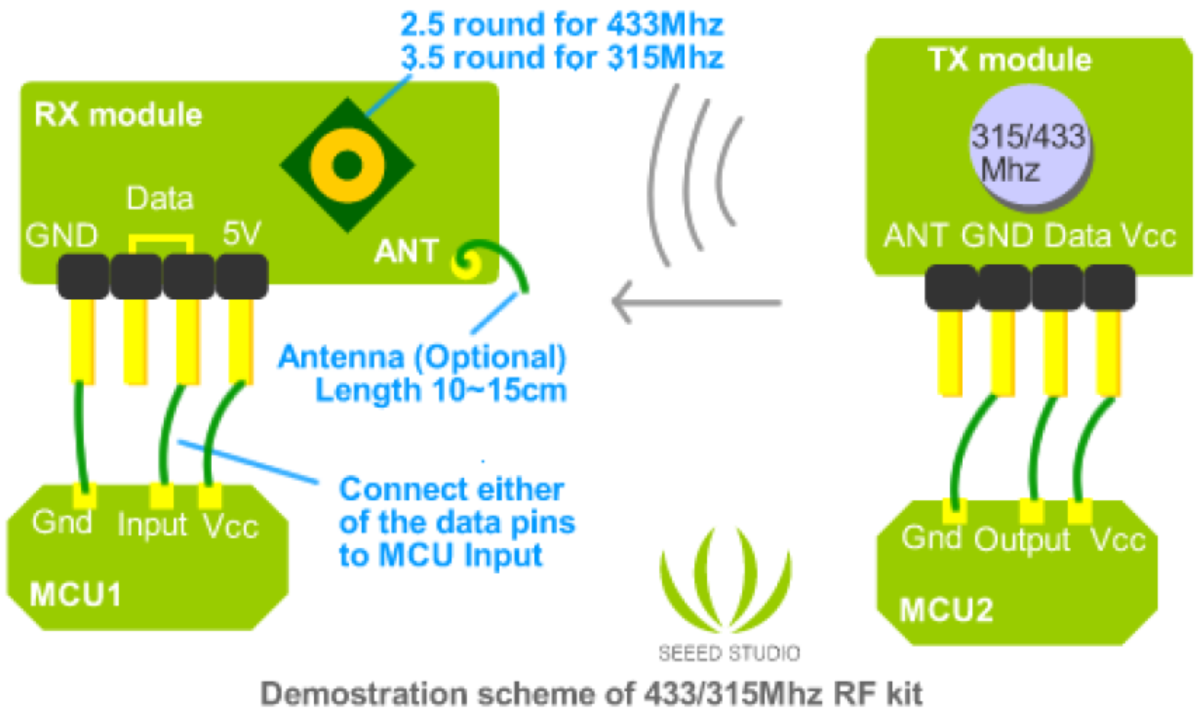


Figure :the example of tx rx combination

This simplified figure explains the combination of tx rx module, and how interact.

DATA TRANSFER

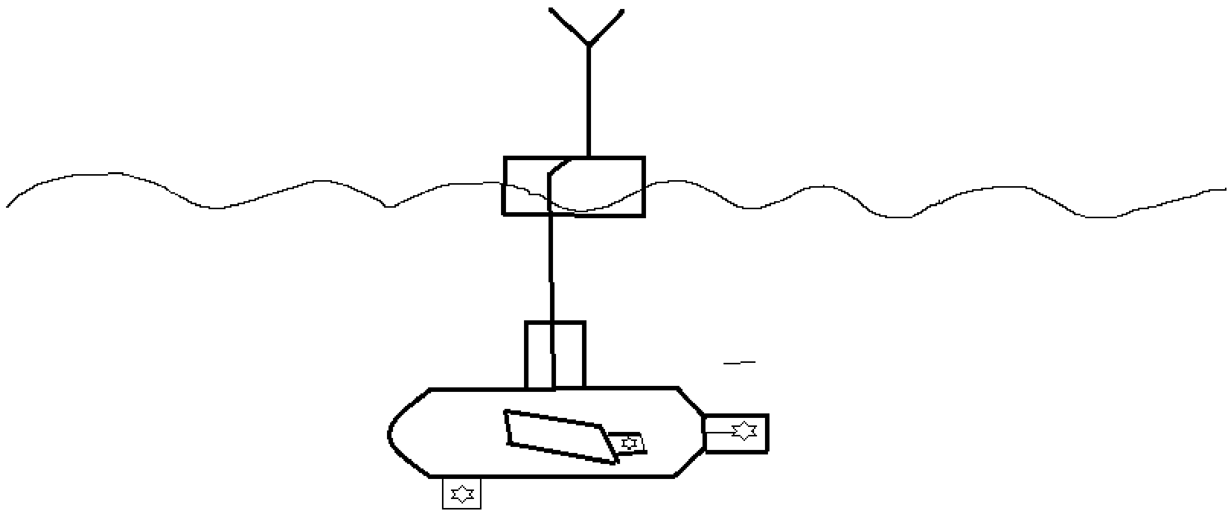


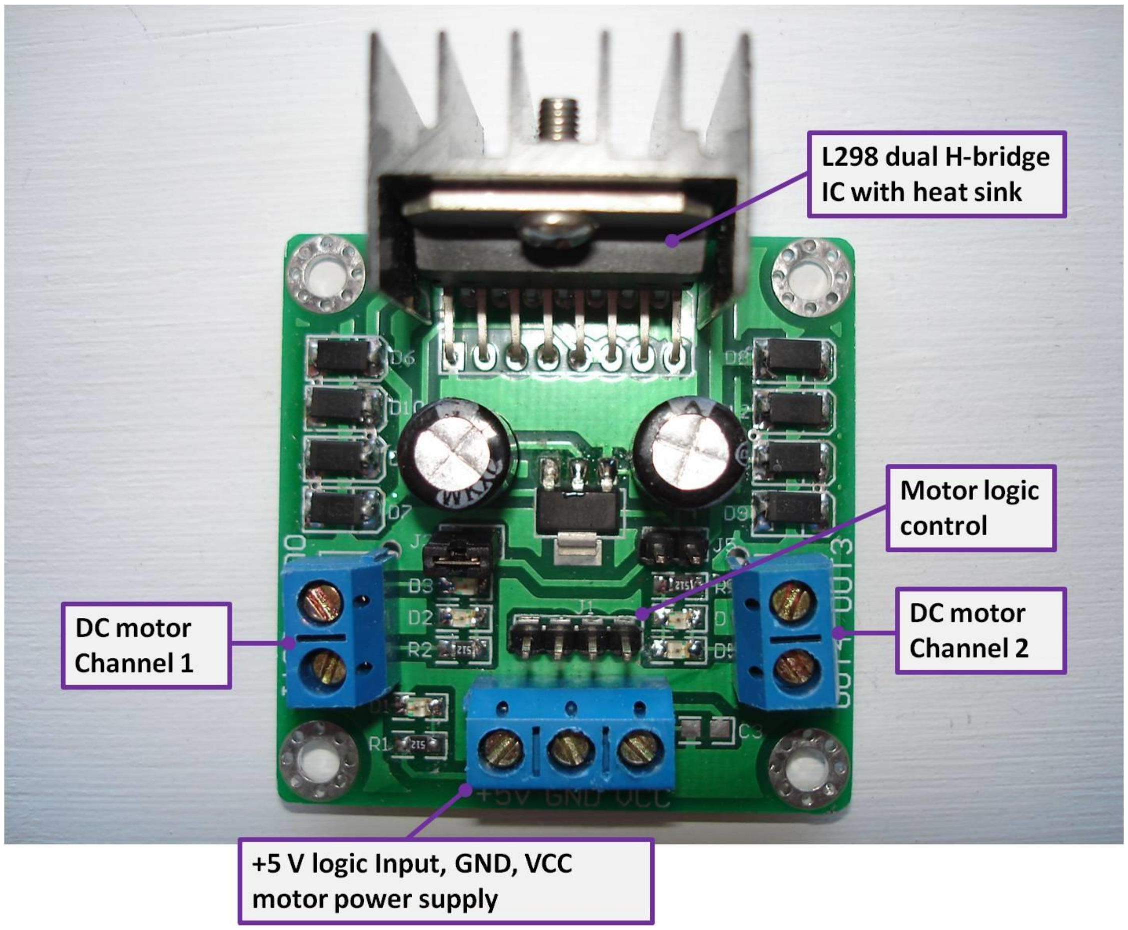
Figure: a way of showing the use of float for the robot

FLOAT:

The float provides buoyancy to maintain tension on the cable, a dry housing for the radio, and mounts the antenna, GPS, and on/off switch. It will made of 6’’ PVC sewage pipe, which is inexpensive, pressure resistant and can survive considerable impact.

The bottom of the pipe will be sealed with a machined PVC sealed cap which is permanently jointed with the solvent. It has a threaded feature to mount the tether whip, a stainless steel welded part which connects to the tether. It also passes the tether conductors through a penetrator, and into the float housing. On the inside, it has tapped holes to mount a plate which attaches the radio.

f. MOTOR SHIELD:


L298 dual H-bridge IC with heat sink
Motor logic control
DC motor Channel 1
DC motor Channel 2
+5 V logic Input, GND, VCC motor power supply

The L298 Motor Driver Kit allows the user to safely interface two DC Motors to a host microcontroller using only 4 control lines. The motor driver isolates the host and controls continuous currents of up to 2 Amps per motor (4 Amps total). Short peaks (spikes) up to 3 Amps per motor can be tolerated without damage. While ideally suited for use with a microcontroller, the kit may also be used with just about any form of 0-5 Volt signal (i.e. manual switches, TTL logic gâtes, relayes etc.).

Assembling the Board

Assembly of the kit is straight forward; the silkscreen (white printing on the circuit board) shows where each component belongs and its orientation. Make sure you install each diode so that the white ring on the diode is aligned with the white bar on the diode symbol on the silkscreen. The diodes are sometimes a tight fit in their holes and you might find a pair of small pliers useful in installing them.

This kit now comes with a 78L05 +5 Volt voltage regulator. The regulator takes power from the Motor Battery terminals and steps it down to +5 Volts for the logic circuits of the L298N chip (this assumes that your Motor Battery voltage is at least 6 Volts). If you are using a Motor Battery that is less than 6 Volts, the 78L05 will not be able to supply the necessary +5 volts to the L298. In this case, you will need to supply this voltage to the Logic Battery terminals yourself (if you do this, do not install the 78L05 and remove it if it is already installed). If you have doubts, measure the voltage across the Logic Battery terminals while both motors are running -there should be a minimum of +4.5 Volts. If there is less, the L298N may behave erratically.

The 0.01uF tantalum capacitors (marked “103”) are used to suppress transients (spikes).

The kit includes 2 black terminal blocks. These terminal blocks provide an easy and convenient way to connect to the L298 kit; simply ‘poke’ a solid wire into the terminal holes to make a connection. If you want a more permanent connection, you can choose not to install the terminal blocks and solder your control lines directly to the board.

About Motor Current

Motors are inductive devices; they draw much more current at startup than when they are running at a steady speed. Before connecting any motor to the L298 you should know a few things about the motor:

- What voltage it is designed to work at

- How much current it draws when running (unloaded)

- How much current it draws at stall.

The “stall current” is the current the motor draws when you stop (stall) the output shaft (if you can). Stalling a motor is very hard on the motor and can burn open the motor windings and ruin the motor. If you want to test for stall current, grab the output shaft with your hand while measuring the current drawn. As the motor approaches stall, the current will climb.

The L298N can safely handle 2 Amps of continuous current for each motor. Short surges up to 3 Amps as the motor starts can be tolerated. A heat sink would be a good idea in situations that see the current surge above 2 Amps.

Connecting the Motors

A DC motor has 2 terminals on it. If you take the positive and negative leads from a power source (battery, power supply etc.) and connect them to the terminals of the motor, the motor will spin in one direction. If you swap the connections, the motor will spin in the opposite direction.

You will want to wire your motors to the L298 board in such a way that the motor spins in the direction you call ‘forward’ when the Fwd line is activated and ‘reverse’ when the Back line activated.

When connecting the motors to the circuit board, use as thick a wire as is practical. The thicker the wire, the less the voltage drop and the more power is delivered to the motor. We recommend a minimum of 18 gauge stranded wire. Solid wire is fine, but will break if flexed too often.

Solder the Positive and negative motor leads for the ‘left’ motor to Motor Left + and - solder pads. Repeat for the ‘right’ motor.

Connect the + and - leads from your motor battery to the Motor Battery + and - solder pads -again, use as thick a wire as it practical.

A Note on ‘Motor Battery’ Voltages: There is a 1.4V drop associated with the L298. This means that if your motor runs on 12 VDC, you should use a 13.4 VDC supply in order to get full power to the motor.

Operation

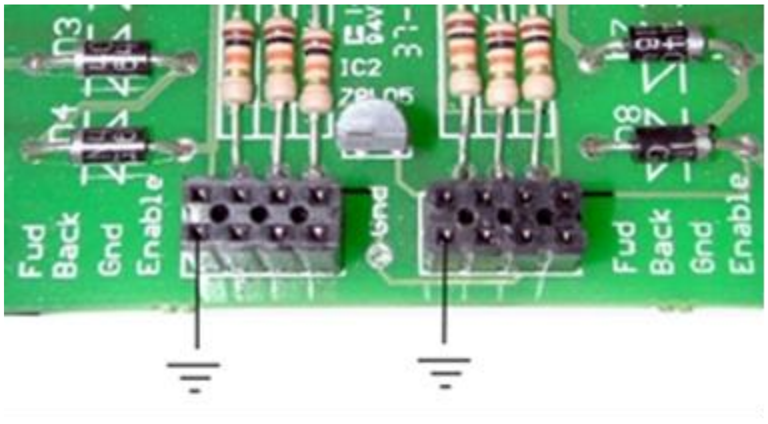
Control is accomplished by grounding the control pin(s) for the desired function/motor. This is usually done by putting a logic LOW (i.e. 0V or ground) on an output pin of your host microcontroller, which is in turn connected to the appropriate control connection (Fwd, Back, Enable).

Fwd

To make a motor move forward, ground the ‘Fwd’ connection on the

appropriate connector (left / right). To stop, un-ground the connection.

The photo below shows how both motors would be made to go ‘forward’.



Back (Reverse)

To drive the motor in reverse, ground the appropriate Back connection.

To stop, unground the connection.

Gnd

Gnd is a ground connection. The L298 Controller board and your control circuit must share the same gound.

Enable

The Enable connection is an active LOW connection that is pulled HIGH for you on the circuit board. Ground this connection to disable a motor. Note: While disabled, commands from the host microcontroller (grounding/un-grounding the control connections) will have no effect on that motor. Disabling a motor shuts-down the L298’ s internal circuits, putting it into a low current consumption mode (for that channel). When both channels are disabled, the motor controller will consume approx. 10 mA

Pulse-Width Modulation (PWM)

Pulse-Width Modulation is a method of controlling the speed of a motor by turning the power on and off at varying speeds. If you have a 12 Volt motor and turn it on 50% of the time and turn it off 50% of the time (switching it at several KHz) then the effective voltage you are applying to the motor is 6 Volts. As the voltage to the motor varies, so does its' speed.

It should be noted that the L298 Kit was not intended for high-speed PWM operation. The diodes in the kits are general purpose rectifier diodes intended for 60 Hz operation. If you want to experiment with PWM and you don't get the kind of results you want, try exchanging the diodes for 'fast recovery Schottky' models.

Heat Sinks

The L298 has internal thermal protection circuitry that shuts down the chip if it becomes too hot (when you try to draw too much current). If you find this happening, you should add some kind of heat sink to the L298. A simple heat sink can be made from a piece of scrap aluminum by cutting as big a piece as you have room for and drilling a 1/8" hole into it (for a bolt to hold it to the L298). If available, a thin smear of thermal compound (white, greasy stuff, available at Radio Shack) on the back of the L298's metal tab (i.e. between it and the heat sink) will maximize heat dissipation. Of course, you should always make sure that air can circulate freely around the L298 and it's heatsink.

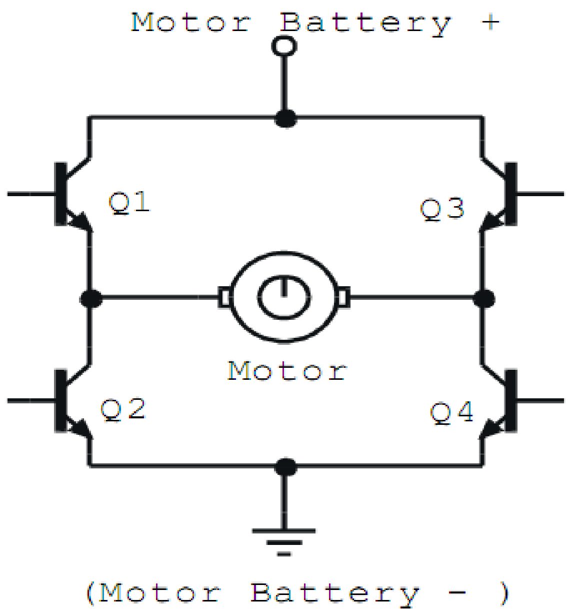
NOTE: If a motor behaves erratically i.e. turning on and off rapidly, it is likely the L298 senses that it is being overloaded. This usually means that you are drawing too much current. Either add a heat sink to the L298 and/or reduce the current being drawn.

Caution: The L298's heat sink (the metal tab) is at ground potential. Do not allow any ground-referenced voltage source to touch it or any heat sink connected to it, or you will cause a short.

H-Bridge Theory

Figure 1 shows the basic schematic for a typical H-Bridge along with it's truth table. In order to make a motor turn, we need to apply a voltage to it. We do this by turning certain NPN transistors on. By looking at the truth table, we can see that in order to make a motor go forward

(NOTE: 'Forward and 'Reverse' are arbitrary directions for purposes of illustration. In your application, forward and reverse will be determined by how the motors are mounted with respect to each other and the polarity of the voltage) we must turn on Q1 and Q4. This puts the Motor Battery Positive on the left side of the motor (through Q1) and grounds the other side of the motor (through Q4).



Truth Table			
	FWD	REV	STOP
Q1	1	0	0
Q2	0	1	0
Q3	0	1	0
Q4	1	0	0

Figure 1.

To go in the opposite direction, we must turn off these transistors and turn on Q2 and Q3. Now, the Motor Battery Positive will be on the right side of the motor (through Q3) and ground is on the left (through Q2). You have now reversed the polarity of the motor's supply voltage and the motor will spin in the opposite direction.

You will notice that each time a motor is turned on, current passes through 2 NPN transistors. Each transistor has (approximately) 0.7 Volt drop across it, so the motor will see about 1.4 Volts LESS than the Motor Battery Voltage across it's terminals. This means that if you have a 12 Volt motor, and you want it to receive maximum power, you should use a 13.4 Volt battery.

Also notice that if transistors Q1 and Q2 (or Q3 and Q4) were turned on, that you would make a short circuit across the battery. For this reason, the L298N has internal logic that prevents this from happening.

Specifications

Max supply voltage: 46 V

Max current (per channel): 2 A (DC); Non-repetitive (t=100 uS): 3A;

Repetitive (80% on, 20% off, ton=10 ms): 2.5 A

Total Power Dissipation: 25W

Controls 2 DC Motors up to 2 Amps each

Forward, Reverse, Stop

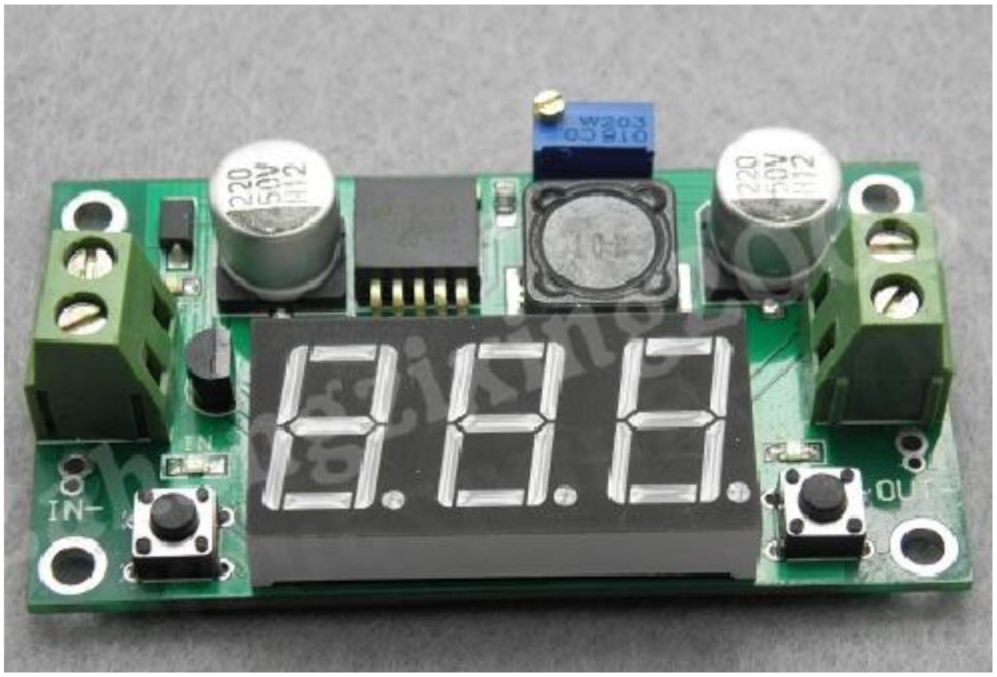
Runs on 6 to 35 Volts DC

Fully Isolates Motor Electricals from Microcontroller

g. POWER:

Li-Poly
11.1V
Rechargeable Battery
2200mAh
Caution:
Please read charging instruction before use.
Improper use can cause explosion or fire.
Use Specified charger only
Do not disassemble, or put the batteries in the fire.

h. Power supply board



FEATURES

• 3.3V, 5V, 12V, and Adjustable Output Versions

• Adjustable Version Output Voltage Range,

1.2V to 37V ±4% Max Over Line and Load Conditions

• Available in TO-220 and TO-263 Packages

• Ensured 3A Output Load Current

• Input Voltage Range Up to 40V

• Requires Only 4 External Components

• Excellent Line and Load Regulation

Specifications

• 150 kHz Fixed Frequency Internal Oscillator

• TTL Shutdown Capability

• Low Power Standby Mode, IQ Typically 80 μA

• High Efficiency

• Uses Readily Available Standard Inductors

• Thermal Shutdown and Current Limit Protection

APPLICATIONS

• Simple High-Efficiency Step-Down (Buck) Regulator

• On-Card Switching Regulators

• Positive to Negative Converter

DESCRIPTION

The LM2596 series of regulators are monolithic integrated circuits that provide all the active functions for a step-down (buck) switching regulator, capable of driving a 3A load with excellent line and load regulation. These devices are available in fixed output voltages of 3.3V, 5V, 12V, and an adjustable output version.

Requiring a minimum number of external components, these regulators are simple to use and include internal frequency compensation , and a fixed-frequency oscillator.

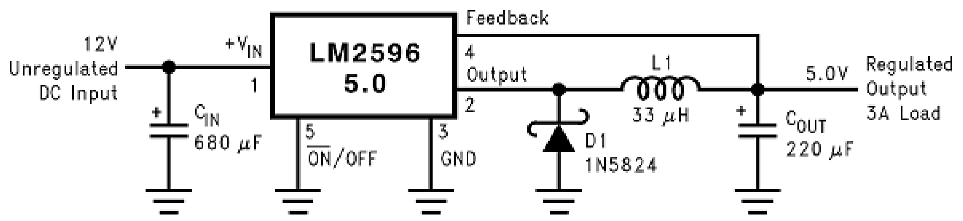
The LM2596 series operates at a switching frequency of 150 kHz thus allowing smaller sized filter components than what would be needed with lower frequency switching regulators. Available in a standard 5-lead TO-220 package with several different lead bend options, and a 5-lead TO-263 surface mount package.

A standard series of inductors are available from several different manufacturers optimized for use with the LM2596 series. This feature greatly simplifies the design of switch-mode power supplies.

Other features include an ensured ±4% tolerance on output voltage under specified input voltage and output load conditions, and ±15% on the oscillator frequency. External shutdown is included, featuring typically 80 µA standby current. Self-protection features include a two stage frequency reducing current limit for the output switch and an over temperature shutdown for complete protection under fault conditions.

Typical Application

(Fixed Output Voltage Versions)



08. COMMAND CENTER

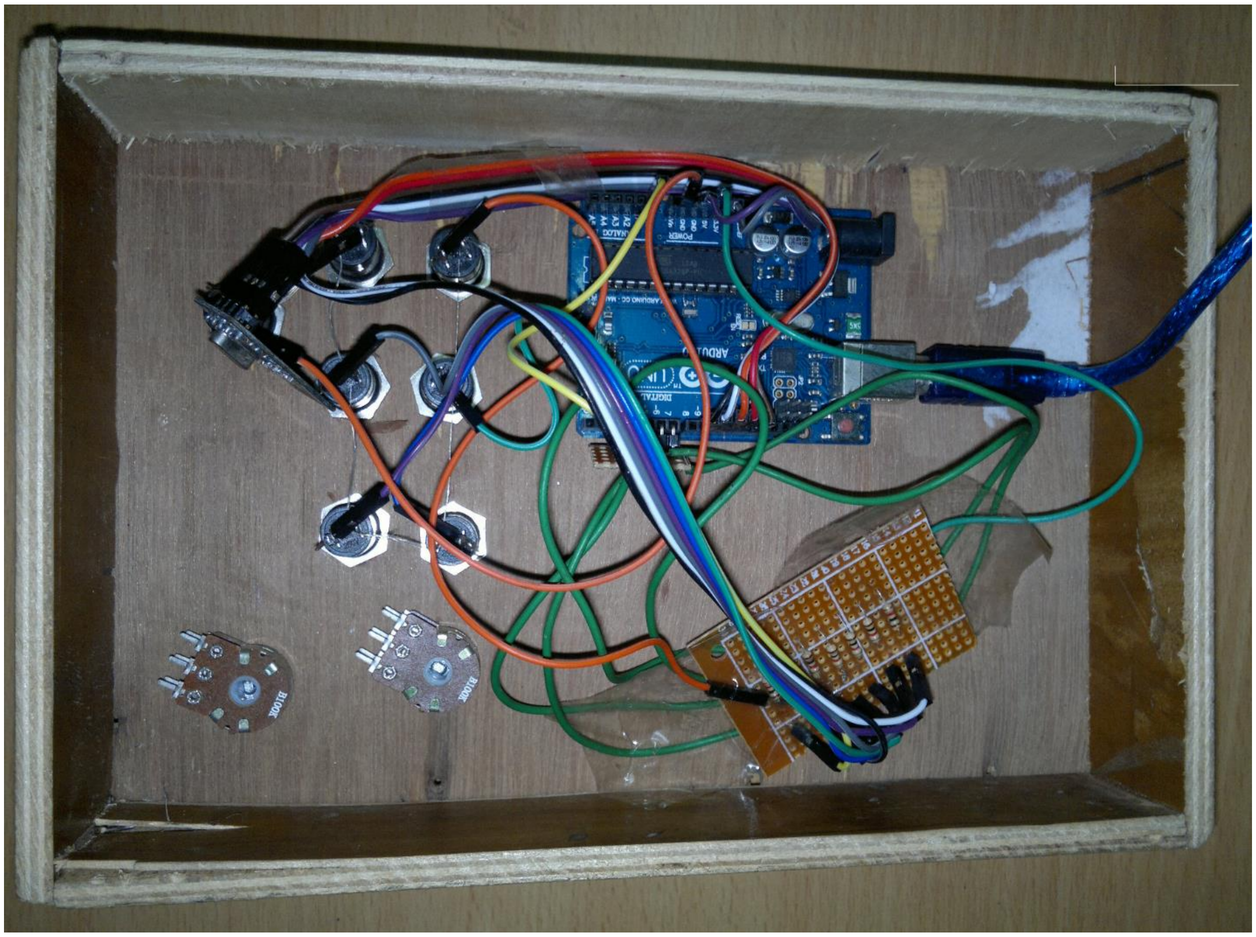


Figure: the command section assembled

As explained earlier the command section consists of several switches, arduino and the transmitter.

09. CONTROL CENTER

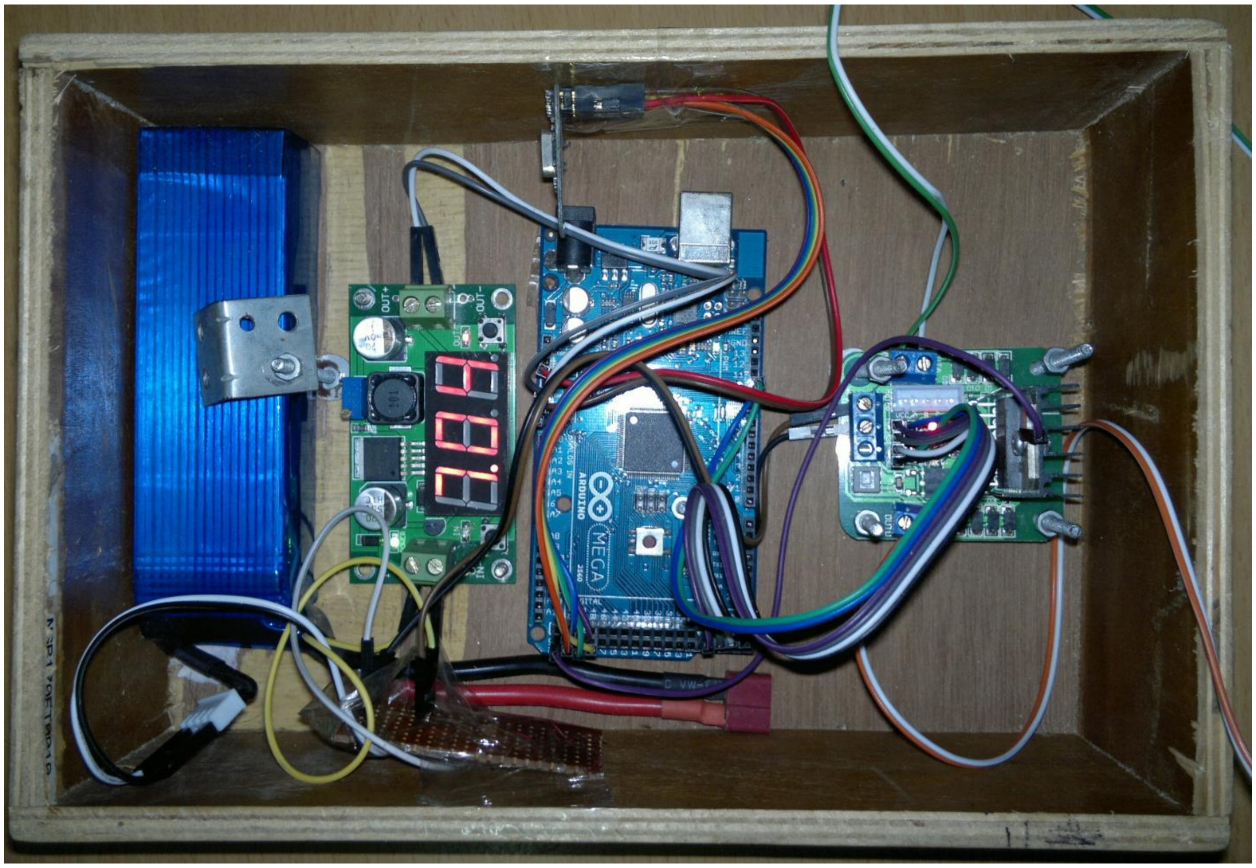


Figure: control section assembled

The control section contains power source, power converter, arduino, receiver module and one motor controller for the simplified view.

10. COMPUTER PROGRAM

tx

- #include <SPI.h>
- #include "nRF24L01.h"
- #include "RF24.h"
-
- // Hardware configuration
- // Set up nRF24L01 radio on SPI bus plus pins 9 & 10
- RF24 radio(9,10);
- ///its pipes to be opened for talk to each other
-
-
-
-
-   boolean state[7]={1,1,1,1,1,1,1};
-   int pins[7]={2,3,4,5,6,7,A0};
-   boolean pins_state[7]={1,1,1,1,1,1,1};
-
-
- void setup()
- {
-
- Serial.begin(57600);

```
•  radio.begin();
•     radio.setRetries(15,15);
•       radio.openWritingPipe(0xF0F0F0F022LL);
•         radio.openReadingPipe(1,0xF0F0F0F011LL);
•
•  // This code will only run once, after each powerup or reset of the board
•  for(int i=0;i<7;i++)
• {         pinMode(pins[i],INPUT);
•              digitalWrite(pins[i],HIGH);//internal pullup resistors on....
•
• }
•
• }
• void button_read()
• {
•  for(int i=0;i<7;i++)
• {
•           digitalRead(pins[i]);
•              if (digitalRead(pins[i])==LOW)
•                   { pins_state[i]=0;
•                                    }
•  else
•  pins_state[i]=1;
•
```

```
//  Serial.print(pins_state[i]);
 // Serial.print("");

}


//Serial.println();
}

void loop()
{  radio.stopListening();///stop receiving while transmitting
        byte dataToBeSent=0;
  boolean diff=false;
      button_read();
   for(int i=0;i<7;i++)
       { if (state[i]!=pins_state[i])
             diff=true;
                state[i]=pins_state[i];
  }

 if (diff)
 {

             byte data=221;
```

- if (state[0]==1 && state[1]==1 && state[2]==1 && state[3]==1 && state[4]==1 && state[5]==1&& state[6]==1){ dataToBeSent=221; radio.write("",8);radio.write(&data ,8);radio.write(&dataToBeSent,8);Serial.println("button pushed out");}//all off
- if (state[0]==0 && state[1]==1 && state[2]==1 && state[3]==1 && state[4]==1 && state[5]==1&& state[6]==1){ dataToBeSent=222; radio.write("",8);radio.write(&data ,8);radio.write(&dataToBeSent,8); Serial.println("button 2 pushed");}///front
- if (state[1]==0 && state[0]==1 && state[2]==1 && state[3]==1 && state[4]==1 && state[5]==1&& state[6]==1){ dataToBeSent=223; radio.write("",8);radio.write(&data ,8);radio.write(&dataToBeSent,8);Serial.println("button 3 pushed");}///back
- if (state[2]==0 && state[0]==1 && state[1]==1 && state[3]==1 && state[4]==1 && state[5]==1&& state[6]==1){ dataToBeSent=224;radio.write("",8);radio.write(&data ,8); radio.write(&dataToBeSent,8);Serial.println("button 4 pushed");}//left
- if (state[3]==0 && state[1]==1 && state[2]==1 && state[0]==1 && state[4]==1 && state[5]==1&& state[6]==1){ dataToBeSent=225;radio.write("",8);radio.write(&data ,8); radio.write(&dataToBeSent,8);Serial.println("button 5 pushed");}//right
- if (state[4]==0 && state[1]==1 && state[2]==1 && state[3]==1 && state[0]==1 && state[5]==1&& state[6]==1){dataToBeSent=226; radio.write("",8);radio.write(&data ,8);radio.write(&dataToBeSent,8); Serial.println("button 6 pushed");}//pump1
- if (state[5]==0 && state[1]==1 && state[2]==1 && state[3]==1 && state[4]==1 && state[0]==1&& state[6]==1){ dataToBeSent=227; radio.write("",8);radio.write(&data ,8);radio.write(&dataToBeSent,8);Serial.println("button 7 pushed");}//pump2
- if (state[6]==0 && state[1]==1 && state[2]==1 && state[3]==1 && state[4]==1 && state[5]==1&& state[0]==1){dataToBeSent=228;

radio.write("",8);radio.write(&data
,8);radio.write(&dataToBeSent,8);Serial.println("button a0
pushed");}//diving

- ///////////// 2 sswitches activation
- ///f plus r
- if (state[0]==0 && state[3]==0 && state[1]==1 && state[2]==1 && state[4]==1 && state[5]==1&& state[6]==1){ dataToBeSent=229; radio.write("",8);radio.write(&dataToBeSent,8); Serial.println("");}
- //f +p1
- if (state[0]==0 && state[4]==0 && state[2]==1 && state[3]==1 && state[1]==1 && state[5]==1&& state[6]==1){ dataToBeSent=230; radio.write("",8);radio.write(&dataToBeSent,8); Serial.println("");}
- //f +p2
- if (state[0]==0 && state[5]==0 && state[2]==1 && state[3]==1 && state[4]==1 && state[1]==1&& state[6]==1){ dataToBeSent=231; radio.write("",8);radio.write(&dataToBeSent,8); Serial.println("");}///front
- //f + bottom
- if (state[0]==0 && state[6]==0 && state[2]==1 && state[3]==1 && state[4]==1 && state[5]==1&& state[1]==1){ dataToBeSent=232; radio.write("",8);radio.write(&dataToBeSent,8); Serial.println("");}///front
- //f + l
- if (state[0]==0 && state[2]==0 && state[1]==1 && state[3]==1 && state[4]==1 && state[5]==1&& state[6]==1){ dataToBeSent=233; radio.write("",8);radio.write(&dataToBeSent,8); Serial.println("");}///front
-
-
- //rr+ r

- if (state[1]==0 && state[3]==0 && state[2]==1 && state[0]==1 && state[4]==1 && state[5]==1&& state[6]==1){ dataToBeSent=234; radio.write("",8);radio.write(&dataToBeSent,8); Serial.println("");}
- /////rr+ p1
- if (state[1]==0 && state[4]==0 && state[2]==1 && state[3]==1 && state[0]==1 && state[5]==1&& state[6]==1){ dataToBeSent=235; radio.write("",8);radio.write(&dataToBeSent,8); Serial.println("");}
- ///r +p2
- if (state[1]==0 && state[5]==0 && state[2]==1 && state[3]==1 && state[4]==1 && state[0]==1&& state[6]==1){ dataToBeSent=236; radio.write("",8);radio.write(&dataToBeSent,8); Serial.println("");}
- ///r + bottom
- if (state[1]==0 && state[6]==0 && state[2]==1 && state[3]==1 && state[4]==1 && state[5]==1&& state[0]==1){ dataToBeSent=237; radio.write("",8);radio.write(&dataToBeSent,8); Serial.println("");}
- ///r+l
- if (state[1]==0 && state[2]==0 && state[0]==1 && state[3]==1 && state[4]==1 && state[5]==1&& state[6]==1){ dataToBeSent=238; radio.write("",8);radio.write(&dataToBeSent,8); Serial.println("");}
-
-
- //l+ r
- if (state[2]==0 && state[3]==0 && state[0]==1 && state[1]==1 && state[4]==1 && state[5]==1&& state[6]==1){ dataToBeSent=239; radio.write("",8);radio.write(&dataToBeSent,8); Serial.println("");}
- ///l +p1
- if (state[2]==0 && state[4]==0 && state[0]==1 && state[3]==1 && state[1]==1 && state[5]==1&& state[6]==1){ dataToBeSent=240; radio.write("",8);radio.write(&dataToBeSent,8); Serial.println("");}

- ///l +p2
- if (state[2]==0 && state[5]==0 && state[0]==1 && state[3]==1 && state[4]==1 && state[1]==1&& state[6]==1){ dataToBeSent=241; radio.write("",8);radio.write(&dataToBeSent,8); Serial.println("");}
- ///l+ bottom
- if (state[2]==0 && state[6]==0 && state[0]==1 && state[3]==1 && state[4]==1 && state[5]==1&& state[1]==1){ dataToBeSent=242; radio.write("",8);radio.write(&dataToBeSent,8); Serial.println("");}
-
-
-
- // r+p1
- if (state[3]==0 && state[4]==0 && state[2]==1 && state[0]==1 && state[1]==1 && state[5]==1&& state[6]==1){ dataToBeSent=243; radio.write("",8);radio.write(&dataToBeSent,8); Serial.println("");}
- //R+ p2
- if (state[3]==0 && state[5]==0 && state[2]==1 && state[0]==1 && state[4]==1 && state[1]==1&& state[6]==1){ dataToBeSent=244; radio.write("",8);radio.write(&dataToBeSent,8); Serial.println("");}
- // r+bottom
- if (state[3]==0 && state[6]==0 && state[2]==1 && state[0]==1 && state[4]==1 && state[5]==1&& state[1]==1){ dataToBeSent=245; radio.write("",8);radio.write(&dataToBeSent,8); Serial.println("");}
-
-
- //p1+p2

- if (state[4]==0 && state[5]==0 && state[2]==1 && state[3]==1 && state[0]==1 && state[1]==1&& state[6]==1){ dataToBeSent=246; radio.write("",8);radio.write(&dataToBeSent,8); Serial.println("");}
- //p1+bottom
- if (state[4]==0 && state[6]==0 && state[2]==1 && state[3]==1 && state[0]==1 && state[5]==1&& state[1]==1){ dataToBeSent=247; radio.write("",8);radio.write(&dataToBeSent,8); Serial.println("");}
-
-
- //p2+ bottom
- if (state[5]==0 && state[6]==0 && state[2]==1 && state[3]==1 && state[4]==1 && state[0]==1&& state[1]==1){ dataToBeSent=248; radio.write("",8);radio.write(&dataToBeSent,8); Serial.println("");}
-
-
- ///////////////////////////////for three button push////
-
- ///Reverse+l+r8
- if (state[1]==0 && state[2]==0 && state[3]==0 && state[0]==1 && state[4]==1 && state[5]==1&& state[6]==1){ dataToBeSent=249; radio.write("",8);radio.write(&dataToBeSent,8); Serial.println("");}
-
-
- ///l+r+bottom
- if (state[2]==0 && state[3]==0 && state[6]==0 && state[0]==1 && state[4]==1 && state[5]==1&& state[1]==1){ dataToBeSent=250; radio.write("",8);radio.write(&dataToBeSent,8); Serial.println("");}

-
-
- ///r8+bottom+pump1
- if (state[3]==0 && state[4]==0 && state[6]==0 && state[2]==1 && state[5]==1 && state[1]==1&& state[0]==1){ dataToBeSent=251; radio.write("",8); radio.write("",8);radio.write(&dataToBeSent,8); Serial.println("");}
-
- ///r8+bottom+pump2
- if (state[3]==0 && state[6]==0 && state[5]==0 && state[2]==1 && state[4]==1 && state[0]==1&& state[1]==1){ dataToBeSent=252; radio.write("",8);radio.write(&dataToBeSent,16); Serial.println("");}
-
- ///bottom+pump1+pump2
- if (state[6]==0 && state[4]==0 && state[5]==0 && state[2]==1 && state[0]==1 && state[1]==1&& state[3]==1){ dataToBeSent=253; radio.write("",8);radio.write(&dataToBeSent,8); Serial.println("");}
-
- ///forward+left+r8
- if (state[0]==0 && state[2]==0 && state[3]==0 && state[1]==1 && state[4]==1 && state[5]==1&& state[6]==1){ dataToBeSent=254; radio.write("",8);radio.write(&dataToBeSent,8); Serial.println("");}
- ///left bottom pump 2 and pump 1 code has not been coded;
-
- //////////////////////////////////////for four buttons
-
- ///rev+lft +r8+bottom

- if (state[1]==0 && state[2]==0 && state[3]==0 && state[6]==0 && state[4]==1 && state[5]==1&& state[0]==1){ dataToBeSent=255; radio.write("",8);radio.write(&dataToBeSent,8); Serial.println("");}
-
-
- ///left+r8+bottom+pump1
- if (state[2]==0 && state[3]==0 && state[6]==0 && state[4]==0 && state[0]==1 && state[1]==1&& state[5]==1){ dataToBeSent=256; radio.write("",8);radio.write(&dataToBeSent,8); Serial.println("");}
-
-
- /////1 r8+bottom+pump1+pump2
- if (state[3]==0 && state[6]==0 && state[4]==0 && state[5]==0 && state[1]==1 && state[2]==1&& state[0]==1){ dataToBeSent=257; radio.write("",16);radio.write(&dataToBeSent,8); Serial.println("");}
-
- ///2 front+left+right+bottom
- if (state[0]==0 && state[2]==0 && state[3]==0 && state[6]==0 && state[1]==1 && state[4]==1&& state[5]==1){ dataToBeSent=258; radio.write("",8);radio.write(&dataToBeSent,8); Serial.println("");}
-
- ///3 left+r8+bottom+pump2
- if (state[2]==0 && state[3]==0 && state[6]==0 && state[5]==0 && state[0]==1 && state[1]==1&& state[4]==1){ dataToBeSent=259; radio.write("",8);radio.write(&dataToBeSent,8); Serial.println("");}
-
-

- ///4 left+bottom+p1+p2
- if (state[2]==0 && state[6]==0 && state[4]==0 && state[5]==0 && state[0]==1 && state[1]==1&& state[3]==1){ dataToBeSent=260; radio.write("",8);radio.write(&dataToBeSent,8); Serial.println("");}
-
-
- ////////////////////////////////for five buttons
-
- /// 5 rev+left+R8+bottom+p1
- if (state[1]==0 && state[2]==0 && state[3]==0 && state[6]==0 && state[4]==0 && state[0]==1&& state[5]==1){ dataToBeSent=261; radio.write("",8);radio.write(&dataToBeSent,8); Serial.println("");}
-
-
- ///forard+left+r8+bottom+p1
- if (state[0]==0 && state[2]==0 && state[3]==0 && state[6]==0 && state[4]==0 && state[1]==1&& state[5]==1){ dataToBeSent=262; radio.write("",8);radio.write(&dataToBeSent,8); Serial.println("");}
-
-
- ///rev+left+r8+bottom+p2
- if (state[1]==0 && state[2]==0 && state[3]==0 && state[6]==0 && state[5]==0 && state[0]==1&& state[4]==1){ dataToBeSent=263; radio.write("",8);radio.write(&dataToBeSent,8); Serial.println("");}
-
- ///forward+left+r8+bottom+p2

- if (state[0]==0 && state[2]==0 && state[3]==0 && state[6]==0 && state[5]==0 && state[1]==1&& state[4]==1){ dataToBeSent=264; radio.write("",8);radio.write(&dataToBeSent,8); Serial.println("");}
-
-
- ///left+r8+bottom+p1+p2
- if (state[2]==0 && state[3]==0 && state[6]==0 && state[4]==0 && state[5]==0 && state[0]==1&& state[1]==1){ dataToBeSent=265; radio.write("",8);radio.write(&dataToBeSent,8); Serial.println("");}
-
- ///////////////////////////////////for six buttons
-
- ///forward+left+r8+bottom+p1+p2
- if (state[0]==0 && state[2]==0 && state[3]==0 && state[6]==0 && state[4]==0 && state[5]==0&& state[1]==1){ dataToBeSent=266; radio.write("",8);radio.write(&dataToBeSent,8); Serial.println("");}
-
-
- ///reverse+left+r8+bottom+p1+p2
- if (state[1]==0 && state[2]==0 && state[3]==0 && state[6]==0 && state[4]==0 && state[5]==0&& state[0]==1){ dataToBeSent=267; radio.write("",8);radio.write(&dataToBeSent,8); Serial.println("");}
-
-
-
- for(int i=0;i<7;i++){
-   // Serial.println("button pusshed");

-   Serial.print(state[i]);
-   Serial.print("");
-    }
- Serial.println();
- }
-
- delay(50);
- }

- Rx

- #include <SPI.h>
- #include "nRF24L01.h"
- #include "RF24.h"
-
- // Hardware configuration
- // Set up nRF24L01 radio on SPI bus plus pins 9 & 10
- RF24 radio(49,53);
-
- void setup()
- {
-   // This code will only run once, after each powerup or reset of the board

- radio.begin();
- radio.setRetries(15,15);
-
- radio.openWritingPipe(0xF0F0F0F011LL);
- radio.openReadingPipe(1,0xF0F0F0F022LL);
-
-
- radio.startListening();
-
- pinMode(13,OUTPUT);
- ////////////////////////////////////motor controller 1
- Serial.begin(57600);
- pinMode(22,OUTPUT);//ind
- pinMode(24,OUTPUT);//inc
- pinMode(26,OUTPUT);//inb
- pinMode(28,OUTPUT); //ina
- pinMode(3,OUTPUT); ////for motor1
- pinMode(4,OUTPUT);   ///for motor2
- pinMode(A0,INPUT);////input speed from pot1
- pinMode(A1,INPUT);////input speed from pot2
-
- /////////////////////////////////motor controller 2
- pinMode(30,OUTPUT);//ind
- pinMode(32,OUTPUT);//inc

- pinMode(34,OUTPUT);//inb
- pinMode(36,OUTPUT); //ina
- pinMode(6,OUTPUT); ////for motor1 ena
- pinMode(5,OUTPUT);   ///for motor2 enb
-
-
- ///////////////////////////////motor controller 2
- pinMode(31,OUTPUT);//ind
- pinMode(33,OUTPUT);//inc
- pinMode(35,OUTPUT);//inb
- pinMode(37,OUTPUT); //ina
- pinMode(7,OUTPUT); ////for motor1 ena
- pinMode(8,OUTPUT);   ///for motor2 enb
-
-
- }
-
- void pot1(int speedmotor1)
- {
-
-
- }
- void pot2(int speedmotor2)
- {

```


}
                              ///////////////////////motor1 and moto2

void motor1(int INc,int INd,int speed){
        digitalWrite(24,INc);
        digitalWrite(22,INd);
        analogWrite(3,speed);

};
void motor2(int INa,int INb,int speed){
        digitalWrite(28,INa);
        digitalWrite(26,INb);
        analogWrite(4,speed);

};

                                    ////////////brake motor1 and 2
void brakeMotor1(int INc,int INd,int speed)
{
        digitalWrite(24,INc);
        digitalWrite(22,INd);
        analogWrite(3,speed);
```

```
•           }
• void brakeMotor2(int INa,int INb,int speed)
• {
•           digitalWrite(28,INa);
•           digitalWrite(26,INb);
•           analogWrite(4,speed);
•           }
• ////////////////////////////////////////////////////motor3
• void motor3(int INa,int INb,int speed){
•           digitalWrite(36,INa);
•           digitalWrite(34,INb);
•           analogWrite(6,speed);
•
• };
•
• void brakeMotor3(int INc,int INd,int speed)
• {
•           digitalWrite(36,INc);
•           digitalWrite(34,INd);
•           analogWrite(6,speed);
•           }
•  ///////////////////////////////////////////motor4
• void motor4(int INc,int INd,int speed){
•           digitalWrite(32,INc);
•           digitalWrite(30,INd);
•           analogWrite(5,speed);
•
•   };
•
•   void brakeMotor4(int INc,int INd,int speed)
•   {
•           digitalWrite(34,INc);
•           digitalWrite(30,INd);
•           analogWrite(5,speed);
•           }
•
•
•
•   ///////////////////////motor5 and motor 6
•
•   void motor5(int INc,int INd,int speed){
•           digitalWrite(33,INc);
•           digitalWrite(31,INd);
•           analogWrite(8,speed);
•
•   };
•   void motor6(int INa,int INb,int speed){
•           digitalWrite(37,INa);
•           digitalWrite(35,INb);
•           analogWrite(7,speed);
•
• };
•
•                                       /////////////motor5  and motor 6
• void brakeMotor5(int INc,int INd,int speed)
• {
•           digitalWrite(33,INc);
•           digitalWrite(31,INd);
•           analogWrite(8,speed);
•           }
• void brakeMotor6(int INa,int INb,int speed)
• {
•           digitalWrite(37,INa);
•           digitalWrite(35,INb);
•           analogWrite(7,speed);
•           }
•  void blinky()
•  {
•  for (int f=0;f<10;f++)
•  {digitalWrite(13,HIGH); delay(20);digitalWrite(13,LOW);
•  }
•  }
• void loop()
• {
•
•            byte receivedData ;
•    int value;
•     bool dataAvailable = radio.available();
•
•  if(dataAvailable==true)
•  {
•
•  bool done = false;
•
•
•
•  while (!done) //Loop until the read buffer is being complitely read
•  {     //
•   radio.read( &receivedData,16);
•                        if (receivedData>0){  value=receivedData;
  Serial.println(value); } // blinky; }
• switch(value){
•  case 221:
•             motor1(0,0,0);
•             motor2(0,0,0);
•             motor3(0,0,0);
•             motor4(0,0,0);
```

- motor5(0,0,0);
- motor6(0,0,0);
- Serial.println(".all brake");
- delay(200);
- break;
- case 222://front
- motor1(1,0,255);
- Serial.println("front");
- delay(200);
- break;
- case 223://back
- motor1(0,1,255);
- Serial.println("back");
- delay(200);
- break;
- case 224://left
- motor2(1,0,255);
- Serial.println("left");
- delay(200);
- break;
- case 225://right
- motor3(1,0,255);
- Serial.println("r8");
- delay(200);

```
          break;

case 226://p1
            motor5(1,0,255);
           Serial.println("pump1");
          delay(200);
          break;

case 227://p2
           motor6(1,0,255);

           Serial.println("pump2");
          delay(200);
          break;

case 228://bottom
  motor4(1,0,255);
           Serial.println("bottom");
          delay(200);
          break;


                         ///////////////////////////////two button press

```

- case 229: //f +r
- motor1(1,0,255);
-   motor3(1,0,255);
-          Serial.println("f +r");
-         delay(200);
-         break;
-
- case 230:///f +p1
- motor1(1,0,255);
-   motor5(1,0,255);
-          Serial.println("f +p1");
-         delay(200);
-         break;
-
- case 231://f+p2
- motor1(1,0,255);
-   motor6(1,0,255);
-          Serial.println("f+p2");
-         delay(200);
-         break;
-
- case 232://f+bottom
- motor1(1,0,255);
-   motor4(1,0,255);

- Serial.println("f+bottom");
- delay(200);
- break;
-
- case 233://f + l
- motor1(1,0,255);
- motor2(1,0,255);
- Serial.println("f + l");
- delay(200);
- break;
-
- case 234://rr+r
- motor1(0,1,255);
- motor3(1,0,255);
- Serial.println("//rr+r1");
- delay(200);
- break;
-
- case 235://rr+p1
- motor1(0,1,255);
- motor5(1,0,255);
- Serial.println(".//rr+p1");
- delay(200);
- break;

```
•
• case 236://rr+P2
•  motor1(0,1,255);
•   motor6(1,0,255);
•          Serial.println("//rr+P2");
•         delay(200);
•         break;
•
• case 237://RR+ bottom
•  motor1(0,1,255);
•   motor4(1,0,255);
•          Serial.println("//RR+ bottom");
•         delay(200);
•         break;
•
• case 238://r+l
•  motor1(0,1,255);
•   motor2(1,0,255);
•          Serial.println("//r+l1");
•         delay(200);
•         break;
•
• case 239://l+r
•  motor2(1,0,255);
•   motor3(1,0,255);
•          Serial.println("//l+r");
•         delay(200);
•         break;
•
•  case 240://l+p1
•   motor2(1,0,255);
•    motor5(1,0,255);
•          Serial.println(".../l+p1");
•         delay(200);
•         break;
•
•  case 241://l +p2
•   motor2(1,0,255);
•    motor6(1,0,255);
•          Serial.println("//l +p2");
•         delay(200);
•         break;
•
• case 242://l+bottom
•   motor2(1,0,255);
•    motor4(1,0,255);
•          Serial.println("../l+bottom");
•         delay(200);
•          break;
•
• case 243://r+p1
•   motor3(1,0,255);
•    motor5(1,0,255);
•           Serial.println("r+p1");
•          delay(200);
•          break;
•
• case 244://l+bottom
•   motor3(1,0,255);
•    motor6(1,0,255);
•           //Serial.println("l+bottom");
•          delay(200);
•          break;
•
• case 245://r+bottom
•   motor3(1,0,255);
•    motor4(1,0,255);
•           Serial.println("./r+bottom");
•          delay(200);
•          break;
•
• case 246://p1+p2
```

- motor5(1,0,255);
- motor6(1,0,255);
- Serial.println(".p1+p2");
- delay(200);
- break;
-
- case 247://p1+bottom
- motor5(1,0,255);
- motor4(1,0,255);
- Serial.println("./p1+bottom");
- delay(200);
- break;
-
- case 248://p2+bottom
- motor6(1,0,255);
- motor4(1,0,255);
- Serial.println("p2+bottom");
- delay(200);
- break;
-
- //////////////three buttons push
- case 249://///Reverse+l+r8
- motor1(0,1,255);
- motor2(1,0,255);

- motor3(1,0,255);
-   Serial.println("Reverse+l+r8");
-   delay(200);
-   break;
-
- case 250://///l+r8+bottom
-
- motor2(1,0,255);
- motor3(1,0,255);
- motor4(1,0,255);
-   Serial.println("l+r8+bottom");
-   delay(200);
-   break;
-
- case 251://///r8+bottom+pump1
-
-
- motor3(1,0,255);
- motor4(1,0,255);
-   motor5(1,0,255);
-
-   Serial.println(".r8+bottom+pump1");
-   delay(200);
-   break;

```
•
• case 252://///r8+bottom+pump2
•
•
•   motor3(1,0,255);
•    motor4(1,0,255);
•           motor6(1,0,255);
•
•           Serial.println("r8+bottom+pump2");
•          delay(200);
•          break;
•
• case 253://///bottom+p1+p2
•
•
•
•    motor4(1,0,255);
•           motor5(1,0,255);
•           motor6(1,0,255);
•           Serial.println("bottom+p1+p2");
•          delay(200);
•          break;
•
• case 254://///f+l+r8
```

- motor1(1,0,255);
- motor2(1,0,255);
- motor3(1,0,255);
- Serial.println("f+l+r8");
- delay(200);
- break;
-
-
- //////////////////////////////////for four buttons
-
- case 255://reverse+left+r8+bottom
- motor1(0,1,255);
- motor2(1,0,255);
- motor3(1,0,255);
- motor4(1,0,255);
- Serial.println("Rreverse+left+r8+bottom");
- delay(200);
-
- break;
-
-
- case 256://left+r8+bottom+pump1
-
- motor2(1,0,255);

```
•           motor3(1,0,255);
•           motor4(1,0,255);
•   motor5(1,0,255);
•   Serial.println("left+r8+bottom+pump1");
•           delay(200);
•           break;
•
•   case 1://r8+bottom+pump1+pump2
•
•
•           motor3(1,0,255);
•           motor4(1,0,255);
•           motor5(1,0,255);
•           motor6(1,0,255);
•   Serial.println("r8+bottom+pump1+pump2");
•           delay(200);
•           break;
•
•   case 2://front+left+r8+bottom
•           motor1(1,0,255);
•           motor2(1,0,255);
•           motor3(1,0,255);
•           motor4(1,0,255);
•           Serial.println("lfront+left+r8+bottom");
•           delay(200);
•           break;
•
• case 3://left+r8+bottom+pump2
•
•           motor2(1,0,255);
•           motor3(1,0,255);
•           motor4(1,0,255);
•           motor6(1,0,255);
•           Serial.println("left+r8+bottom+pump2");
•           delay(200);
•           break;
•
• case 4://left+bottom+pump1+pump2
•
•           motor2(1,0,255);
•           motor4(1,0,255);
•           motor5(1,0,255);
•           motor6(1,0,255);
•           Serial.println("left+bottom+pump1+pump2");
•           delay(200);
•           break;
•
•
```

```
•
•
•
•
• ///////////////////////////////////for five buttons
• case 5://reverse+left+r8+bottom+p1
•           motor1(0,1,255);
•           motor2(1,0,255);
•           motor3(1,0,255);
•           motor4(1,0,255);
•           motor5(1,0,255);
• Serial.println("reverse+left+r8+bottom+p1");
•           delay(200);
•           break;
•
•           case 6://forward+left+r8+bottom+p1
•           motor1(1,0,255);
•           motor2(1,0,255);
•           motor3(1,0,255);
•           motor4(1,0,255);
•           motor5(1,0,255);
• Serial.println("forward+left+r8+bottom+p1");
•           delay(200);
•           break;
```

-
- 	case 7://reverse+left+r8+bottom+p2
- 	motor1(0,1,255);
- 	motor2(1,0,255);
- 	motor3(1,0,255);
- 	motor4(1,0,255);
- 	motor6(1,0,255);
- Serial.println("reverse+left+r8+bottom+p2");
- 	delay(200);
- 	break;
-
- 	case 8://forward+left+r8+bottom+p2
- 	motor1(1,0,255);
- 	motor2(1,0,255);
- 	motor3(1,0,255);
- 	motor4(1,0,255);
- 	motor6(1,0,255);
- Serial.println("forward+left+r8+bottom+p2");
- 	delay(200);
- 	break;
-
- 	case 9://left+r8+bottom+p1+p2
-
- 	motor2(1,0,255);

- motor3(1,0,255);
- motor4(1,0,255);
- motor5(1,0,255);
- motor6(1,0,255);
- Serial.println("left+r8+bottom+p1+p2");
- delay(200);
- break;
-
- ///for six buttons
-
- case 10://forward+left+r8+bottom+p1+p2
- motor1(1,0,255);
- motor2(1,0,255);
- motor3(1,0,255);
- motor4(1,0,255);
- motor5(1,0,255);
- motor6(1,0,255);
- Serial.println("forward+left+r8+bottom+p1+p2");
- delay(200);
- break;
-
- case 11://reverse+left+r8+bottom+p1+p2
- motor1(0,1,255);
- motor2(1,0,255);

- motor3(1,0,255);
- motor4(1,0,255);
- motor5(1,0,255);
- motor6(1,0,255);
- Serial.println("reverse+left+r8+bottom+p1+p2");
- delay(200);
- break;
-
-
- default:
- brakeMotor1(0,0,255);
- brakeMotor2(0,0,255);
- brakeMotor3(0,0,255);
- brakeMotor4(0,0,255);
- brakeMotor5(0,0,255);
- brakeMotor6(0,0,255);
- Serial.println("nothing ");
-
- }
-
- delay(20);
-
- }
-

- }
-
-
-
-
-
-
- //default:
-
-
- }

CONCLUSION:

Next dream:

We started research and development of a demonstration long range vehicle to cruise. It is important to improve many elemental technologies such as navigation system and so on to achieve this aim figure shows a concept image of concentrating these elemental technologies to next generation vehicle.

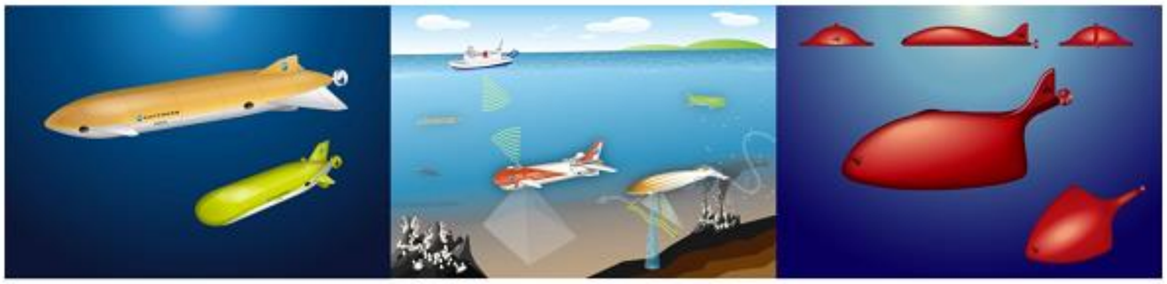


Figure: The Next Generation Autonomous Underwater Vehicles

Summary:

The present study confers recent progress in the technology for unmanned underwater vehicle from the modeling, control and guidance perspectives. The survey of the contemporary AUVs is briefly presented. Dynamics of an unmanned underwater vehicle or robot is important of modeling in the control synthesis. A model based low level controller is presented. The major trends in underwater robotics are discussed: autonomous system, robotics, and multi UUV system. Future challenges for advancing underwater robotics technology will be pivoted on finding accurate, robust yet affordable navigation technology for longer mission, exploration of biomimetic principles for viable products and development of formal model and analysis tool to synthesize collaborative underwater robotics behavior.

12. BIBLIOGRAPHY :

1. Tadahiro Hyakudome Japan Agency for Marine-Earth Science and Technology (JAMSTEC), Japan '*Design of Autonomous Underwater Vehicle'*

2. Kristensen, J. et al. (1998). "Hugin an untethered underwater vehicle for seabed surveying,"OCEANS '98, C onf. Proc. Vol.1 pp.118-123.

3. Mellingham, J. G. et al. (1993). "Demonstration of a high-performance, low-cost autonomous underwater vehicle," MITSG 93-28.

4. Bradley, A. M. et al. (2000). "Extending the Endurance of an Operational S cientific AUV using Lithium-ion Batteries," Proc. of Unmanned Underwater Vehicle Showcase (UUV S) 2000.

5. Stokey, R. P. et al. (2005). "Development of the REMUS 600 autonomous underwater vehicle," OCEANS 2005 Proc. of MTS/IEEE, Vol.2 pp. 1301-1304.

6. Ura t.,AUV 'r2d4', its operation and roadmap for AUV development, in :Advances in Unmanned Marine Vehicles, edited by G.N.Robert&r. Sutton(IEE control series69)

7. Dylan Owen. 'design ,construction and operation of an unmanned underwater vehicle'

8. www.arduino.com