République **A**lgérienne **D**émocratique et **P**opulaire
Ministère de l'**E**nseignement **S**upérieur et de la **R**echerche **S**cientifique

UNIVERSITE **M**OULOUD **M**AMMERI DE **T**IZI-**O**UZOU

FACULTE DU **G**ENIE **E**LECTRIQUE ET D' **I**NFORMATIQUE
DEPARTEMENT D'AUTOMATIQUE

Mémoire de Fin d'Etudes de MASTER ACADEMIQUE

Domaine : **Sciences et Technologies**
Filière : **Génie électrique**
Spécialité : **Commande des systèmes**

Présenté par
Achour SILEM

Thème

Commande à distance d'un socle de caméra

Mémoire soutenu publiquement le 01/07/ 2015 devant le jury composé de :

MR.DJENNOUNE Saïd
Professeur, à l'UMMTO, Président

MR.MELLAH Rabah
Maitre de conférence (A), à l'UMMTO, Encadreur

MR.GUERMAH Saïd
Maitre de conférence (A), à l'UMMTO, Examinateur

MR.TOUAT Mohand Ouachour
Maitre de conférence (B), à l'UMMTO, Examinateur

Ce travail a été conçu et réalisé au laboratoire L2CSP de l'UMMTO

PEOPLE'S DEMOCRATIC REPUBLIC OF ALGERIA

MINISTRY OF HIGHER EDUCATION AND SCIENTIFIC RESEARCH

MOULOUD MAMMERI UNIVERSITY OF TIZI-OUZOU

ELECTRICAL ENGINEERING AND COMPUTER SCIENCE FACULTY

AUTOMATIC DEPARTMENT

Project Report presented in Partial Fulfillment of the requirements of the Degree of MASTER

Domain: **Science and Technology**

Path: **Electrical Engineering**

Option: **Systems Control**

Presented by

Achour SILEM

Title:

Remote Controlled Plinth

Memory publicly supported on 07/01/ 2015 in front of the following jury:

MR. Saïd DJENNOUNE.

Professor, at. MMUOT, President

MR Rabah MELLAH.

M.C.C.A, at. MMUOT, Supervisor

MR .Saïd GUERMAH.

M.C.C.A, at. MMUOT, Examiner

MR.TOUAT Mohand Ouachour.

M.C.C.B at MMUOT, Examiner

This work was conceived and implemented in L2CSP laboratory of MMUOT

DEDICATIONS

Dedications:

I want dedicate this work to:

-My family especially my Mom.

-My Sweetie.

-To those who have encouraged me, believed in me and challenged me to do better.

<<Be the change that you want to see in the word>>

ACKNOWLEDGEMENT

Acknowledgement:

First of all, from the bottom of my heart I want to thank my Shepherded ***LORD*** for his help, for having been with me every single day, I want to thank you for your love, wisdom and spiritual intelligence. (***ALSHADAY*** you are).

Then I would like to express my sincere appreciation for my supervisor ***MR.R.MELLAH***, and a special thank for ***MR.K.MOULOUDJ*** for being here when I needed help, for having oriented me, and giving his best in my project. Without forgetting the ***L2CSP*** laboratory team for their generosity especially my friend ***NADIA*** for inviting to be part of ***L2CSP***, thank you to ***MR.A.ARDJAL*** and ***MR.A.TRIKI*** for their advices and encouragements.

I would like to record my warmest gratitude to my parents and brothers, sisters for their love and endless support that have made it possible for me to excel in my studies.

My kind regards to my friends with whom I have learned and experimented all the moments that we had in college.

The best for last, I want to thank my loving ***MOM*** for her love, and a warm thank for my adorable, lovely ***SWEETIE***.

ABSTRACT

Abstract

The aim of this project is to control a plinth remotely in two axes, i.e "x-axes" for left/right and "y-axes" for up/down. The plinth is known as supporting block conceived to hold something for example a camera for motion. The term "remote control" means controlling at some distance, thus we have the main part referred as "master site" that sends control signals to that plinth referred as "slave site", then the subsystem ensuring communication between them is called "communication channel", these three main parts form the teleoperation control system

In our case, the master consists in thumb joystick and an electronic based board, the slave (plinth) consists in another electronic based board and two servomotors to be moved in the two mentioned axes. The motors can either be stepper or servomotors. We opted to use servomotors in this work for its precise control of angular position. And the electronic based board used is an Arduino since it's easy to program and well documented. In order to maintain the two sites connected, we can use either wired or wireless transmission.

Nowadays, modern technology is growing in means of communications, thus the most used protocol in wireless communication is WIFI, after submission done by IEEE in 2001, Zigbee alliance is created to develop the IEEE 802.15.4 norm approved on Mai 2003 known as a wireless language (Xbee). In other words Zigbee is the norm using the standard IEEE 802.15.4 and Xbee is the module based on that norm.

This mode is known for its reliability and low power consumption (100 µW), in addition this module changes its operation mode in less time (300 µs) comparing to other WPAN as Bluetooth.

In other words there is the operator that manipulates the joystick to send control signals to be processed by the master main controller then transmission/reception by the Xbee modules. After reception these control signals are uploaded to the slave main controller board to move the plinth to the desired position.

Finally the whole system is an autonomous position-visual feedback unilateral teleoperation control system using wireless transmission commands

Keywords: *Unilateral control, PWM control, Wireless transmission, position, plinth.*

Résumé

Les objectifs entrepris au cours de ce mémoire concernent la commande d'un socle en position dans les deux axes à distance, "l'axe x" pour les directions "Gauche/droite", et l'axe "y" pour les directions "Haut/bas". Le socle est connu comme étant un support permettant de tenir des objets pour les faire bouger dans des directions souhaitées comme une camera. Le terme "commande à distance" nous fait comprendre qu'il ya deux sites, un site situé au contrôle principale connu comme étant le maitre, et l'autre qui exécute ces commandes est connu sous le nom de l'esclave, et le sous système assurant la communication entre ces deux derniers est appelé "canal de transmission". Ces trois parties essentielles mentionnées dans ce paragraphe forment un système de teleopération.

Dans notre cas, le maitre est composé d'une simple manette (Joystick en Anglais), et une carte électronique, par la suite on a l'esclave représentant le socle est constitué d'une autre carte électronique et deux moteurs qui le font bouger dans les directions indiquées au premier paragraphe. Cependant les moteurs qui seront utilisés peuvent être soit moteurs pas à pas, ou bien servomoteurs. On a utilisé les servomoteurs connus par leur précision et rapidité, notamment pour leur régulateur qui assure la loi de commande interne sans se soucier des paramètres qui ont tendance à refléter la complexité d'identification de ces derniers. En suite la carte électronique utilisée est Arduino-UNO, on l'a choisie vu qu'elle contient tous les composants dont on a besoins dans un seul chip.

Pour assurer la liaison entre ces deux sites, on peut soit utiliser la transmission filaire ou sans fils, cependant le choix revient aux circonstances rencontrées au cours de l'opération de la transmission de données. Il est souvent conseillé d'utiliser le sans fils quand la distance s'élève à des dizaines de mètres, de plus pour éviter tout le placement de fils, c'est pour ces raisons qu'on a opté pour l'un des protocoles de communication sans fils connu sous le nom de Xbee de la norme Zigbee.

De nos jours la technologie ne cesse d'augmenter quant à la transmission de données, cependant le protocole le plus répandu est le WIFI, après une soumission à l'IEEE 2001, la ZigBee Alliance a été créée pour développer et promouvoir la norme IEEE 802.15.4 ratifiée en mai 2003 dite Xbee. En d'autre terme Zigbee est la norme de ce Protocol, et Xbee est un module basé sur cette dernière. Ce mode permet à une entité communicante ZigBee de consommer très peu d'énergie (100 µW) tout en permettant de passer en mode opérationnel en très peu de temps (300 µs), contrairement à d'autres WPAN comme Bluetooth par exemple.

En d'autres termes, on a l'opérateur qui manipule le joystick pour envoyer des signaux de contrôle passant par le contrôleur du site maitre pour la transmission de données sans fils. Une fois que la réception est réalisée, ces signaux vont être chargés dans le contrôleur du site esclave afin de faire bouger le socle dans la direction indiquée par l'operateur. Et enfin notre système est un système de teleoperation unilatéral non autonome dont les caractéristiques sont position-retour visuel.

Mots-clés: *Commande unilatérale, commande PWM, Transmission de données sans fils, position, socle.*

NOTATIONS & ABBREVIATIONS

Notations and Abbreviations

- ***Notations:***

Parameter	Unit	Description
$F_{\bullet}$*	N	Exogenous input force
F_h	N	Operator force exerted on the master
F_e	N	Environment force exerted on the slave
F_m and F_s		Controller forces
$Y_{\bullet}$	m	Position
$Z_{\bullet}$	$N.m^{-1}$	Impedance
C_s and C_m		Local position controllers
C_5 and C_6		Local force feedback controllers
C_1 to C_4		Position or force transfer controllers
$M_{\bullet}$	kg	Mass
$B_{\bullet}$	$N.s.m^{-1}$	Damping
$K_{\bullet}$	$N.m^{-1}$	Stiffness
h_{ij}		Hybrid parameters
H		Hybrid matrix

MY: Stands for Source Address.

DH&DL: Stands for Destination address.

xp: The "x" axes potentiometer value.

Yp: The "y" axes potentiometer value.

x_pos: stands for the position of servomotor moving with respect to "x" axes.

y_pos: Stands for the position of servomotor moving with respect to "y" axes.

COM5: Arduino Transmitter via Xbee 1.

COM6: Arduino Receiver via Xbee 2.

- ***Abbreviations:***

ANL: Argonne National Laboratory.

DOF: Degree Of Freedom.

HSI: Human Service Interface.

CH: Channel.

PEB: Position Error Based.

FEB: Force Error Based.

DFR: Direct Force Reflection.

PP: Position-Position.

FF: Force-Force.

PF: Position-Force.

FP: Force-Position.

EBB: Electronic Based Board.

μC: Microcontroller.

CPU: Central Processing Unit.

I/O: Input/Output.

ALU: Arithmetic Logic Unit.

CU: Control Unit.

RAM: Random Access Memory

SRAM: Static Random Access Memory.

DRAM: Dynamic Random Access Memory.

ROM: Read Only Memory.

PROM: Programmable Read Only Memory.

EPROM: Erasable Programmable Read Only Memory.

EEPROM: Electrically Erasable Programmable Read Only Memory.

LED: Light Emitting Diode.

ADC: Analog to Digital Converter.

ISR: Interrupt Service Routine.

UART: Universal Asynchronous Receiver Transmitter.

TxD: Data Transmission.

TxD: Data Reception.

PWM: Pulse Width Modulation.

SPI: Serial Peripheral Interface.

I^2C: Inter Integrated Circuit.

USB: Universal Serial Bus.

PDA: Personal Digital Assistant.

LAN: Local Area Network.

IEEE: Institute of Electrical and Electronic Engineering.

MAC: Medium Access Control.

WPAN: Wireless Personal Area Network.

IP: Internet Protocol.

FHSP: Frequency Hoping spread Spectrum.

GND: Ground.

UHF: Ultra High Frequency.

WNIC: Wireless Network Interface controller.

PAN ID: PAN IDentifier.

LR-WPAN: Low Rate WPAN.

RF: Radio Frequency.

PCB: Printed Circuit Board.

IDE: Integrated Development Environment.

API: Address Programmable Interface.

.

TABLE OF CONTENTS

Table of Contents

Chapter III: The Conception

Chapter IV: The Implementation

INTRODUCTION

General Introduction:

With the fast development of modern technology, human is nowadays being able to explore the extremely dangerous environment such like aerospace, seafloor or high-radiation area and etc. by the support of teleoperation. The teleoperation system enables the human operator to control the remote robot without facing the harsh working environment directly, which guarantees the safety of human beings.

Thus, teleoperators, or the act of teleoperation, extends the manipulative capabilities of the human arm and hand to remote, physically hostile, or dangerous environments. In this sense, teleoperation conquers space barriers in performing manipulative mechanical actions at remote sites, like telecommunication conquers space barriers in transmitting information to distant places; however this introduces time delay and interaction constraints that most teleoperation systems suffer from.

In a more modem point of view, teleoperators are specialized robots, called telerobots, performing manipulative mechanical work remotely where humans cannot go or do not want to go. Teleoperator robots serve to extend, through mechanical, sensing, and computational techniques, the human manipulative, perceptive, and cognitive abilities into an environment that is either hostile to or remote from the human operator. Teleoperator robots or, in today's terminology, telerobots typically perform non-repetitive or singular, servicing, maintenance or repair work under a variety of environmental conditions ranging from structured to unstructured conditions. Telerobotic control is characterized by a direct involvement of the human operator in the control since, by definition of task requirements, teleoperator systems extend human manipulative, perceptual skills to remote places.

Telerobotics has given birth to teleguidance, where there is no presence of contact force, i.e the operator system deals with position remote control that is widely used in tele-surveillance.

The outline of this thesis is described as following:

Chapter I: gives an introduction of teleoperation in modern control, including its essential elements and different application, followed by the types of teleoperation control by giving the linear modeling of different components such as human operator, environment, controllers, and master and slave robots. As a typical example for bilateral control, the four-channel architecture – a generalization of many existing approaches – is first presented. Next, performance of other control schemes, including the two-channel. Stability and transparency are the major goals in every bilateral control design that are also briefly reviewed at the end of this chapter.

Chapter II: provides an overview to the system to be controlled, including and describing its internal components and talked about different devices and components existing to build the system.

Chapter III: focuses on the conception by choosing the adequate devices described in chapter II to complete our system building circuitry.

Chapter IV: is aimed to implement the whole system by relating the components described in chapter II, and the conception made in chapter III.

Chapter I: *Generalities about Teleoperation*

1.1) ***Introduction:***

Teleoperator is a machine that enables a human operator to move about, sense and mechanically manipulate objects at a distance (Controling over distance). Most generally any tool, which extends a person's mechanical action beyond her reach, is a teleoperator.

It's known as the oldest domains of robotics and one of the most challenging areas until now. It enables human operator to interact with remote environment. A general teleoperation system consists of human operator, local master device, controller, a remotely located slave robot and environment. In one side, human operator handles the master device, and in another side slave robot performs some tasks to the environment according to human operator's command (Master-Slave scheme). These two sides communicate with each other through controllers which exchange information either in the forms of position or force.

Since its early use in the remote manipulation of radioactive materials, teleoperation has expanded its scope to many areas such as space technologies, underwater explorations, military, and even include manipulation at different scales. Scaled teleoperation offers a solution where the environments dimensions do not match with human workspaces as in the case when we are dealing with macro-micro world.

We usually find these types in teleoperation theory:

- ***In bilateral teleoperation***: force feedback is provided to the operator, enabling him to feel the contacts with the environment.
- ***Modern bilateral teleoperation:***

It was shown that if master and slave exchange power variables (***such as force and velocity***), the communications channel is not passive in the presence of time delays and may destabilize the whole system.

The proposed solution was reformulated as the transmission of a pair of wave variables, which preserves passivity of the communications channel for constant time delays. Position error may arise due to different reasons (initial mismatch, contacts with the environment, numerical errors, . . .), and, since only the master velocity is transmitted – not its position – there is no way to recover from this error. Several improvements have been proposed to overcome this problem, for more information see:

- Niemeyer and Slotine (1997b)
- Yokokohji, Tsujioka, and Yoshikawa (2002).
- Chopra, Spong, Ortega, and Barabanov (2006)
- Tavakoli, Patel, and Moallem (2008).....and so on.

1.2) ***Brief history:***

In about 1945 the first modern master-slave system teleoperator was developed by ***Goertz*** at Argonne National Laboratory (***ANL***). This system, which was a mechanical pantograph mechanic system, allowed a human operator to manipulate radioactive materials in a "hot cell" from outside. By using a master handle, the human operator could move the slave tong located inside the hot cell and receive force reflection. Soon (in 1954) electrical servomechanisms replaced the direct mechanical master-slave system and able linkages. Closed circuit television was introduced so that the operator could stay at an arbitrary distant place. In addition, the contact force between the slave and its environment was returned to the master arm. In 1964, Mosher developed an impressive work, which was a handy-man containing ***electrohydraulic*** arms.

The problem of remote control of robotic systems has been the subject of much research in recent years. Remote control of robotic systems has been applied in manufacturing, underwater manipulation, storage tank inspection, nuclear power plant maintenance, space exploration, etc.

1.3) ***Applications of Teleoperation:***

Motivation:

The need of human skill and intelligence in the areas where he/she cannot be present has been the basic idea for formation of teleoperation systems. Over the past three decades, the use of teleoperation technology has been steadily growing in a wide range of applications. These applications include space operation, underwater exploration, mining, nuclear material handling, toxic material handling, the entertainment industry, and more recently health care.

1.3.1) ***Space:***

Teleoperation has been used in space applications very frequently. Most of the deep space probes have been *telerobots*, which had relatively simple, straightforward, but reliable controls, having *lowbandwidth* capability to receive pictures and other sensory data, and the ability to be reprogrammed in space. In 1993 the German space agency DLR successfully demonstrated the first space *telerobot*, the "*Rotex"* experiment, on the NASA Space Shuttle.

This experiment showed that the ability of a computer to control a *telerobot* in space, and also showed that space *telemanipulation* can be controlled from earth through a time delay. Japan has made noteworthy progress on *teleoperator* development. The Japan Experiment Module (JEM) is an integral part of Space Station Freedom. JEM includes a long *teleoperator* operating from a "porch." Figure-1.1- shows this *teleoperator* that was on the Japanese Experiment Module for Space Station Freedom.

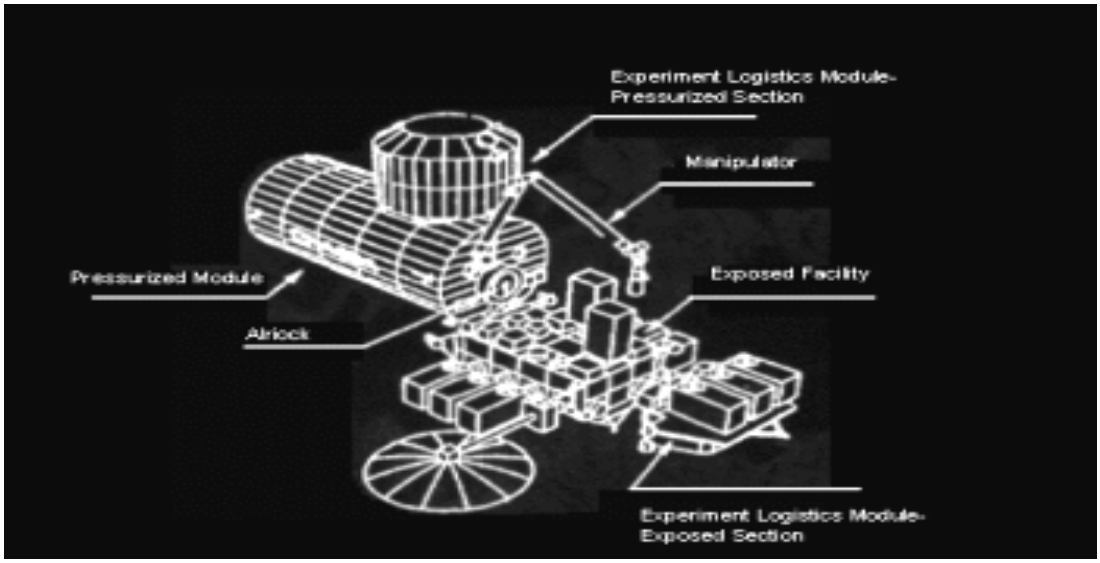


Figure-1.1-: The Japanese teleoperator

Planetary surface telerobots have also been developed with mixed success. After the first American Surveyor mission in 1967, others such as the Russian Lunakhods in 1969 and 1980, and the American Viking missions to Mars in 1976 have been attempted. The Mars Pathfinder space probe sent back the panoramic color images of Mars' surface in 1997. NASA is planning to send another probe to Mars to explore that planet. Also, the Space Shuttle manipulator on the shuttle has been controlled by two three-degree-of-freedom joysticks from within the shuttle.

Figure 1.2: shows an illustration of telerbotic in the space domain, on the left "***Mars pathfinder***" with Sorjourner robot in 1997, and on the right ***«Mars ExplorationRovers»*** with two twin robots, ***spirit*** and ***opportunity*** in 2004.

Figure-1.2-: Telerobotic in space

1.3.2) ***Undersea:***

In the area of undersea applications, *teleoperation* is used in offshore oil exploration, inspection and maintenance on drill-heads, oil platforms and pipelines, geological surveys, and classified navy tasks. *Americans*, British, *Japanese* have led in this area.

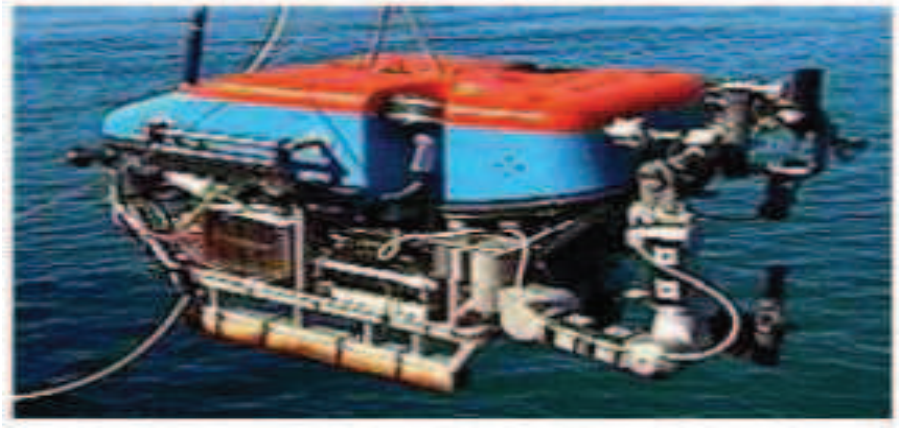


Figure-1.3-: Undersea vehicle

Figure 1.3 shows an undersea vehicle Jason that was controlled remotely to locate Titanic. This system was developed as *Argo-Jason* project at the *Woods Hole* Oceanographic Institute and named after Jason and his *Argonauts of Greek mythology*.

1.3.3) ***Toxic cleanup:***

In toxic waste cleanup, *teleoperation* has the *irreplaceable* function, especially in nuclear plants. Presence of radioactive materials and leakage make the environment dangerous for humans, and all tasks in this environment lend itself to *teleoperation*. Although the main concept of remote control and *teleoperation* remains the same, size of the system may vary considerably depending on the size of the task. Remotely controlled systems may vary from pipe crawlers to trucks with hydraulically operated manipulators that can carry several thousand pounds of payload.

1.3.4) ***Teleoperation in nuclear:***

Inspection and maintenance is essential in the nuclear industry. It's not easy to carry out such maintenance tasks since the environments are usually highly radioactive and are unsafe for human workers.

Figure-1.4- gives illustration of some nuclear applications

Figure-1.4-: Teleoperation in nuclear applications

Issues:

- Suffer from low pay load capacity.
- Relatively large end point detections.
- Installation and the storage of these long manipulators could be costly.

1.3.5) ***Military missions:***

Clearly military missions can be very unsafe for a human, so the use of a robot to do this could be a solution (*peace is one of them*) by operating it from distance.

This's the Army robot soldier for Iraq, January 24, 2005 shown just below in figure-1.5-:

Figure-1.5-: Robot army soldier

1.3.6) ***Other applications:***

Today teleoperators find their way into more and different applications; they become accepted as the normal and ordinary way of doing tasks. In medicine, CAT and MRI imaging,

Teleopetation is now being used to guide robotic devices in order to machine the heads of femurs and other bone structures for much tighter and surer fitting of prosthetic joint implants than was possible with manual drilling.

The modern and advanced *teleoperators* and *telerobotic* devices are applied in a dangerous environments including those that are for humans too (*nuclear reactors*), those where humans adversely affect the environment (*clean rooms*), and those which are impossible for humans to be situated in (deep space exploration). We believe that *teleoperation* systems will be used in more and more different environments with the development of new and effective technologies.

1.4) ***Essential elements of teleoperation system:***

The general conception of a teleoperation control is: see [1]

- Human Service Interface (HSI).
- Master (System Interface).
- Communication network.
- Slave (Teleoperator)
- Remote Environment.

Figure-1.6- shows the relation between these main components:

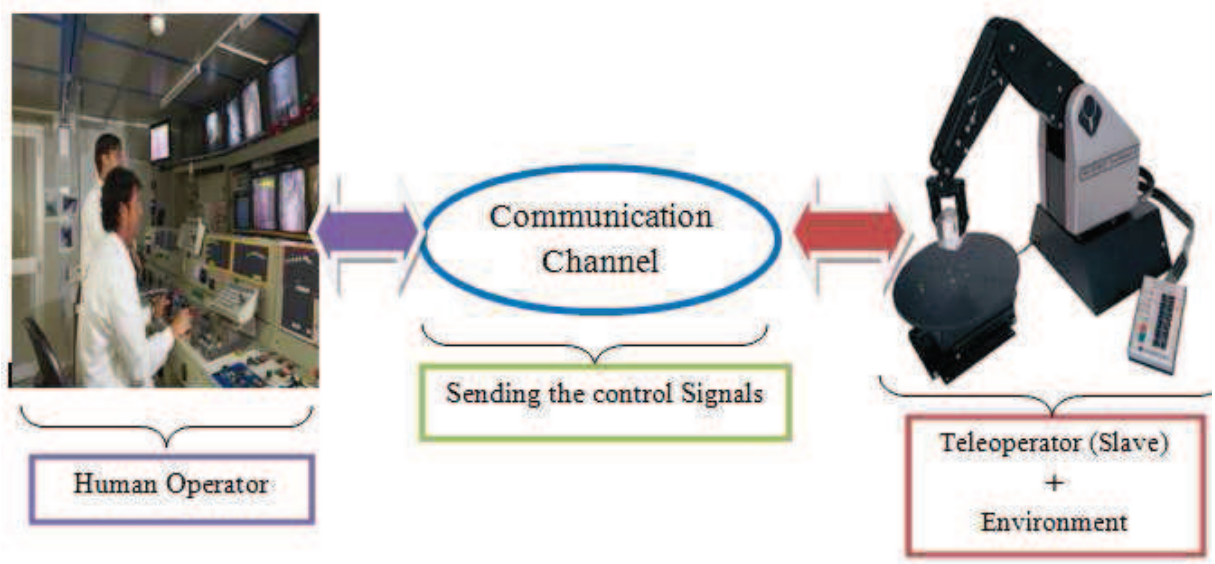


***Figure-1.6a-**: The basic elements of teleoperation Control System.*

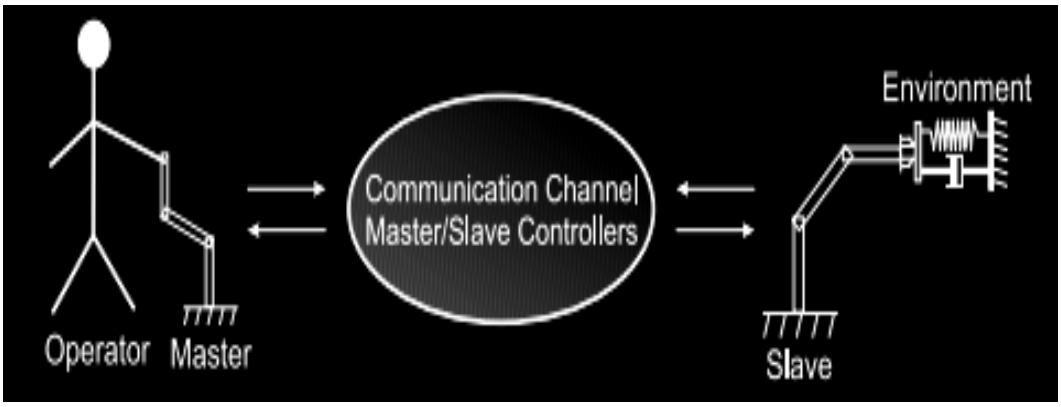


***Figure-1.6b-:** Illustrates the five elements of teleoperation*

Where:

- ***Human operator (Controller):*** is the person who monitors the operated machine and makes the needed control actions.
- ***Communication Channel :*** represents the way or the method used to send the control signal from master site to slave site.

- ***Teleoperator (Slave):*** is a sophisticated teleoperator machine that executes the tasks required by the Operator on the Environment.
- ***The Environment***: is the place or the space on which the operator manipulates tasks via the slave.

1.5) ***Classification of teleoperation control:***

Teleoperation can be classified in two categories:

1.5.1) ***Unilateral Teleoperation:***

Unilateral teleoperators transmit *position and/or force* commands to the slave site and relay visual sensory information (*figure-1.7b*-) from the task back to the master site. See [2].

In the unilateral case:

- The system is designed similar to an open loop controls system, i.e. there is no feedback available to the user.
- This system is generally not very difficult to build, but the applications are limited.

One scenario where the unilateral system may be a good choice might be when the master and slave are in the same room and have a direct hardwire link between them. An operator that is in the same room as the slave robot can already see what is happening and may not need another type of feedback.

Figure-1.7- : shows a bilateral teleoperation consisting of the operator who exerts a force to make the slave moving, with no feedback efforts.

In this case (Figure-1.7a-) there's no need to get back the sensory information , since the operator does.

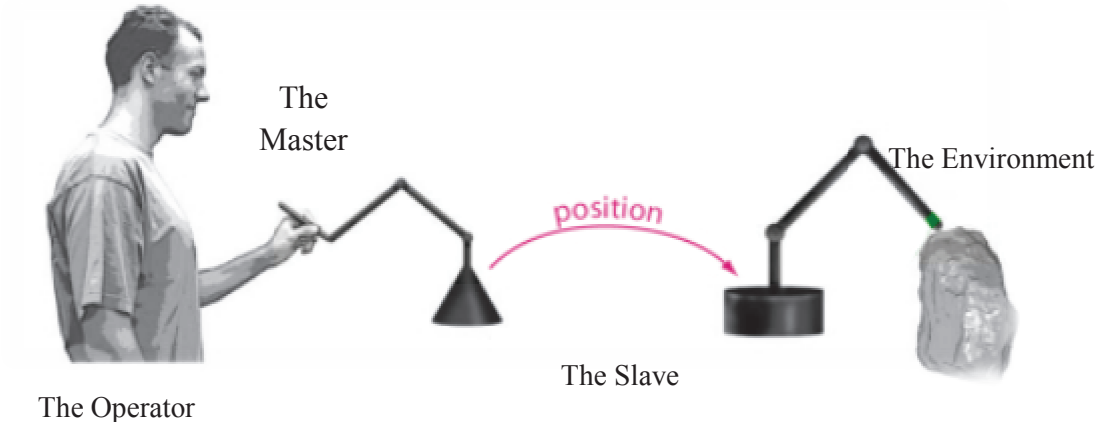


***Figure-1.7a-*: Unilateral Teleoperation**

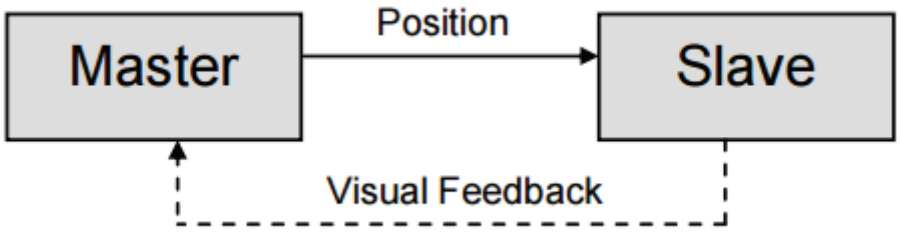


Figure-1.7b-: one way dissipating energy from Master to slave where there is visual feedback

1.5.2) ***Bilateral Teleoperation:***

A bilateral teleoperation system is comprised of a slave robot that interacts with a usually unknown environment, the slave robot is connected to the master robot and the latter is controlled by a human operator that closes the loop, the two systems (*master and slave*) exchange signals like position, velocity and force.

In bilateral teleoperation, force feedback allows the user to have a better feel for the remote environment through the master (figure-1.8a-), providing more information to the operator such as kinesthetic and *haptic* feedback. See [3].

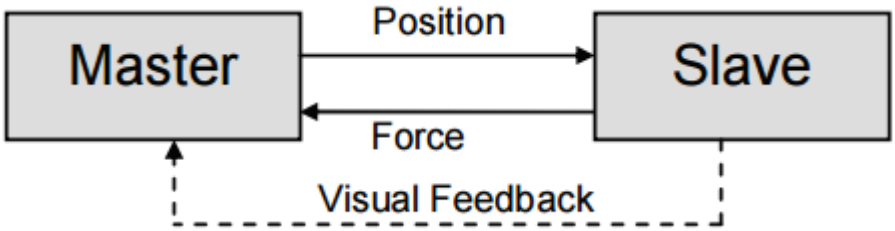


Figure-1.8a-: Bilateral teleoperation system

Forms of feedback:

- Live video from video cameras
- Haptic (touch, such as a vibration)
- Auditory (human ear range)
- Temperature
- Contact sensors.
- Sonar images.

However, the performance of a bilateral teleoperator might be degraded, frequently to a point of instability, when there is a significant amount of time delay in the propagation of signals through the communication line.

According to definition, teleoperation is extension of human sense and ability to work with manipulator in a remote position. A general bilateral teleoperation model is shown in ***Figure-1.8b-***

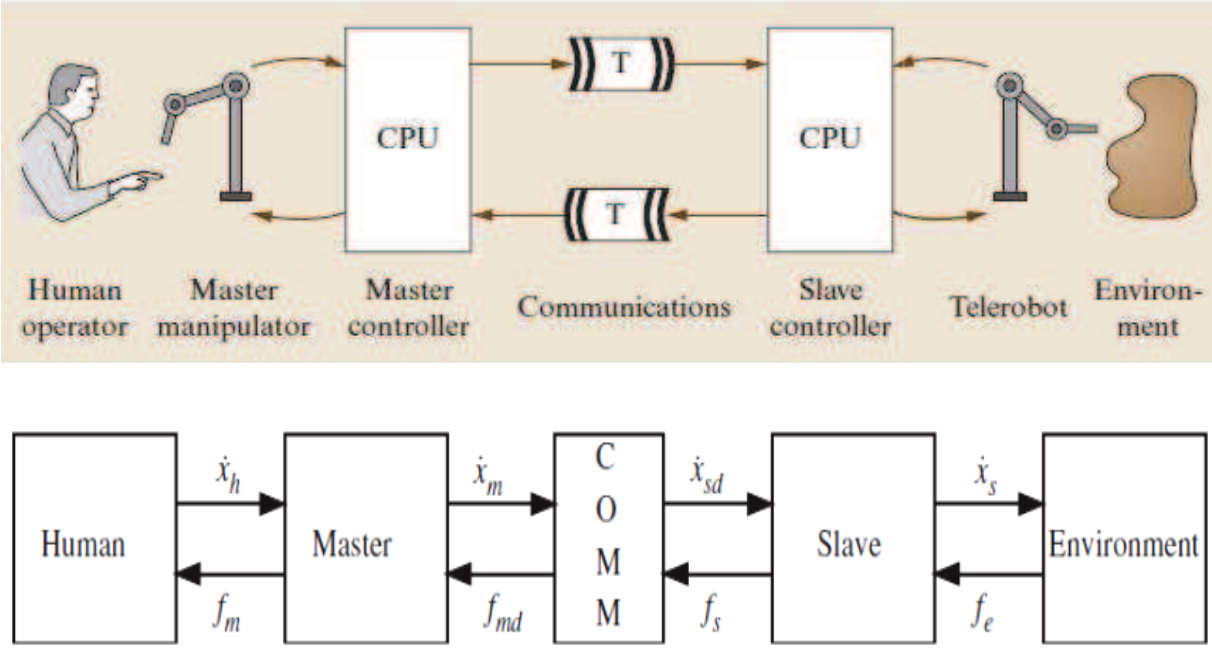


Figure-1.8b-: *Block diagram of bilateral teleoperation system*

In this The teleoperation system:

- operator applies a force ***Fm*** on the master device, and this latte moves at some velocity ***vm***, in response.
- The scenario is equivalent to the case in which the operator gives a velocity command ***vm*** to the master. The real intention of the operator is to move the device, not to merely apply forces on it.
- When being moved, the master device sends a response force ***Fm*** back to the operator.
- The velocity information of the master is transmitted, through the communication channel, to the slave. The transmitted velocity becomes the reference velocity ***Vsd*** for the slave.
- Upon receiving the master velocity, the slave controller calculates the corresponding force ***Fs***, to drive the slave manipulator to follow the received reference velocity.
- The calculated driving force, called coordinating force, is also sent back to the master through the communication channel, and becomes the desired master force ***Fmd***.
- Based on this desired force value, the master actuators generate the same amount of force and the operator can then feel it.
- When the slave robot contacts with the environment, the contact force ***Fe*** will be reflected to the operator through the coordinating force ***Fs***.
- But since only at steady state, ***Fs*** has the same value as ***Fe***, the force sensed by the operator includes the dynamic effect of the slave, especially when there is no contact, ***Fe*** is zero, while ***Fs***, is usually not.

1.6) ***Teleoperation System Building blocks:***

We recall that teleoperator is an interface (*master-controller-slave*) that communicates force and movement between the *human operator* and the *remote environment*. As it can be seen in *Figure-1.9-*, the operator and the environment are not part of the teleoperator itself but mechanically connected to the teleoperator. Due to this mechanical coupling, they influence the dynamics of the teleoperation system, mainly regarding stability (see section 1.8.8).

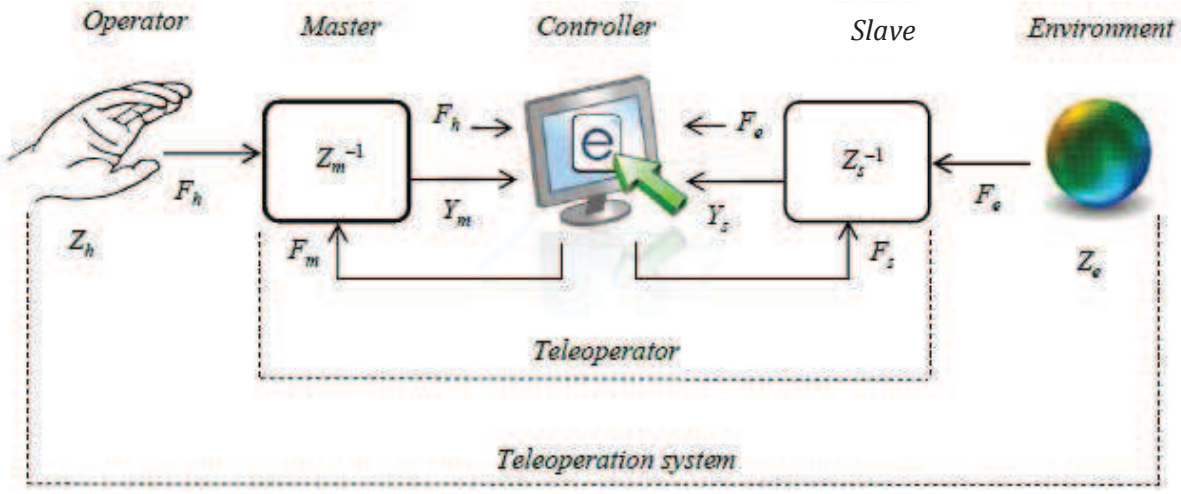


Figure-1.9-*:* "Master-Controller-Slave" is in contact with operator and remote environment.

Referring to Figure-1.9-:

- ***Zm and Zs****:* denote impedances of master and slave manipulators' linearized dynamics, which are generally approximated by simple mass-spring-damper systems.
- **Zh and Ze** denote impedances, the dynamic characteristics of the human operator's hand and the environment.
- ***Ym and Ys*** are the master and slave positions.
- ***Fm and Fs****:* are the (force) control signals for the master and slave manipulators.
- ***Fh and Fe:*** are respectively the operator force exerted on the master and the environment force exerted on the slave.

See [3], [4], [5].

The signals used in the teleoperation are also called "***power variables***" effort (force) and flow (velocity). This leads to a definition of impedance as force divided by velocity ($Z(s) = F(s)/Y(s)$) . Some researchers instead define the impedance as force over position [6]. It seems more natural to use position, as most real teleoperation systems do measure position.

Furthermore, ensuring ***velocity tracking*** between the master and the slave might cause small offsets between the master and slave positions (*i.e.*, ***steady-state errors in position tracking***).

Generally, when the delay in the communication channel is negligible, the use of position controllers or velocity controllers does not affect the stability of the teleoperation system, thus we opt to use position controllers.

Figure-1.10- shows a general approximation of teleopration system where we can see that the whole system acting like "***Mass Spring Damper System***". See section (1.6.1).

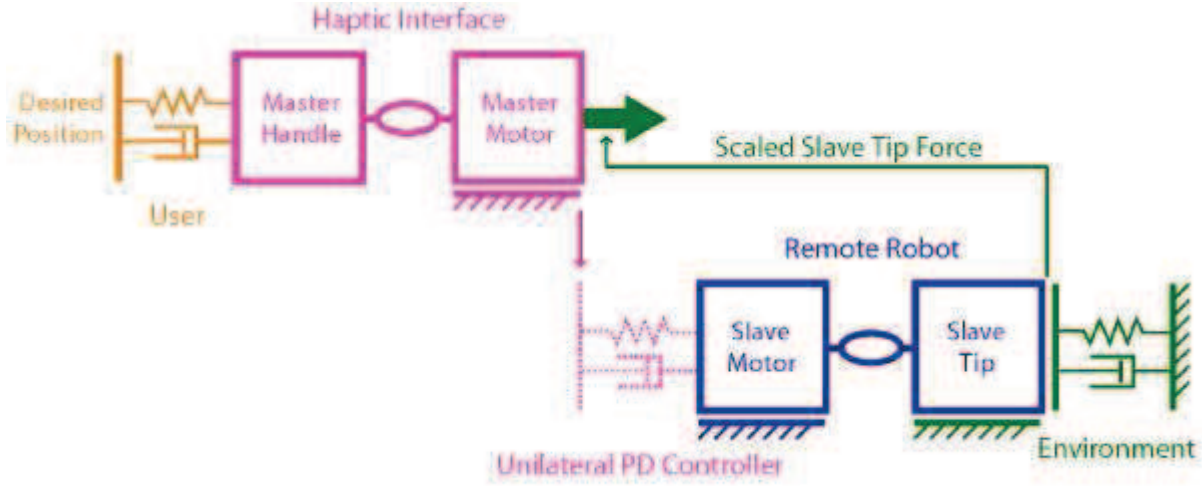


Figure-1.10-: General approximation of a teleoperation system

1.6.1) ***Human Operator Model:***

The human operator is usually modeled as impedance. This choice has a rather large impact on the teleoperation system, since it implies that the human can be assumed passive. For stability, this makes passivity analysis possible.

However, a human is not passive at all, since he/she can apply an active force to the master to change the system behavior. These lead us to two classes of models can be identified as [3]:

- ***Passive operator***: the human operator cannot increase total energy in the system that means he/she does not perform actions that will make the teleoperation system unstable.
- ***Active operator:*** the human operator provides input signals based on his goal and perceives information about the state of the system.

In our case, the operator is assumed to be passive and its impedance can be approximated by a low-order model with parameters in certain range. For linear analysis, it's generally chosen as ***"mass-spring-damper"*** model [7] for the operator impedance as:

$$Z_h(s) = M_h s^2 + B_h s + K_h \tag{1.1}$$

Where: M_h, B_h, and K_h are assumed to be positive corresponding to the mass, damping and spring (stiffness) coefficients of the operator.

To understand the equation (1.1), here is a mass spring damper system characteristic.

- ***Mass Spring Damper System:*** *(Deriving the motion)*

A general mass spring system is shown in ***figure-1.11-***, where:

- ***C***: is the damping constant.
- ***k***: is the spring constant.

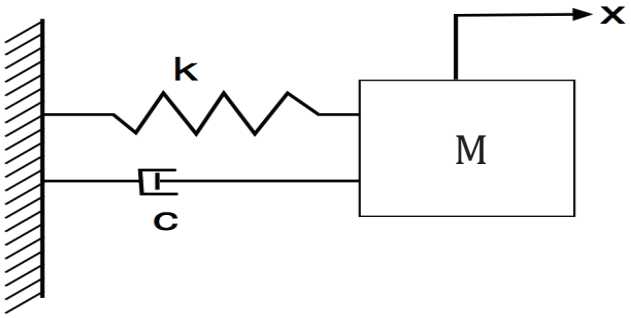


Figure-1.11-: General Mass spring damper System

The arrow shows the positive direction of "x" (motion).

Mass (***m***) is attached to Hook's law spring. The spring force is given by $F_s = -kx$, and the damping force is given by $F_d = c\frac{dx}{dt}$.

Then from Newton's second law we have:

$$\left.\begin{aligned} \sum F_{ext} &= Ma \\ M\ddot{x} &= -c\dot{x} - kx \end{aligned}\right\} \quad (1.2)$$

Rearranging:

$$M\ddot{x} + c\dot{x} + kx = 0 \quad (1.3)$$

Dividing by m:

$$\frac{d^2x}{dt^2} + \frac{c}{M}x + \frac{k}{M}x = 0 \quad (1.4)$$

If we let $\omega_0 = \sqrt{\frac{k}{M}}$, be the natural frequency of the system and $\xi = \frac{c}{2\sqrt{kM}}$, be the damping ratio, then we get:

$$\ddot{x} + 2\delta\omega_0\dot{x} + \omega_0^2 = 0 \qquad (1.5)$$

"A differential equation is a mathematical equation for an unknown function of one or several variables that relates the values of the function itself and of its derivatives of various orders."

Applying Laplace Transform:

$$x(s^2 + 2\xi\omega_0 s + \omega_0^2) = 0 \qquad (1.6)$$

To explain the result found in equation (***1.1***), let's take:

$$Z_h = s^2 + 2\xi\omega_0 s + \omega_0^2 \qquad (1.7)$$

In this case, we see by identification that:

- $M_h = 1$.
- $B_h = 2\xi\omega_0$.
- $K_h = \omega_0^2$.

1.6.2) ***Environment Model:***

The environment is generally the most uncertain component in the teleoperation system. The variation in environment impedance can be large (*i.e.*, ***from zero to infinity***), especially when the slave moves in free space and suddenly comes in contact with a stiff object. Often, the environment can be approximated as a LTI mass-spring-damper system:

$$Z_e(s) = M_e s^2 + B_e s + K_e \qquad (1.8)$$

Where M_e, B_e, and K_e are assumed to be positive corresponding to the mass, damping and spring coefficients of the environment.

Furthermore, an environment model is often used to quantify the performance of the teleoperation system. The transparency expresses how well the system can reproduce a certain remote environment to the operator. The performance of the teleoperation system will be elaborated in "***section (1.8)***".

1.6.3) ***Controller Model:***

By definition, the controller is a model of all the components (*sensors, amplifiers, transmission line, and controller hardware*) between the master and slave devices.

Assuming that the four signals of forces and positions (*Fh, Ym, Fe, Ys*) are measured, the aim is to choose the controller transfer functions *Kij* to optimize the performance (i.e.*transparency and stability*)[8] of the teleoperation system:

$$\left.\begin{aligned} F_m &= K_{11}F_h + K_{12}Y_m + K_{13}F_e + K_{14}Y_s \\ F_s &= K_{21}F_h + K_{22}Y_m + K_{23}F_e + K_{24}Y_s \end{aligned}\right\} \quad (1.9)$$

Or we can express this as:

$$\begin{bmatrix} F_m \\ F_s \end{bmatrix} = \underbrace{\begin{bmatrix} K_{11} & K_{12} & K_{13} & K_{14} \\ K_{21} & K_{22} & K_{23} & K_{24} \end{bmatrix}}_{K} \begin{bmatrix} F_h \\ Y_m \\ F_e \\ Y_s \end{bmatrix} \quad (1.10)$$

K: is the controller transfer function chosen with respect to the desired performances.

In the literature, there are several teleoperation control architectures (see section 1.7), which present certain choices of the controller parameters.

1.6.4) ***Master/ Salve model:***

Most of literature describes the master and slave manipulators as linear models. This implicates that all nonlinearities and disturbances are cancelled by using compensation techniques.

Generally, the ***Master/ Slave*** manipulators can be categorized as admittance or impedance type, depending on whether they behave like velocity or force sources, respectively. This behavior is determined by the structural design and actuation employed by the manipulator.

By definition:

- ***An impedance device*** receives a force input and applies force to the environment in response to the measured position.
- ***However; admittance device*** receives a ***velocity/position*** input and applies a ***velocity/position*** to the environment in response to the measured contact force.

Most of the proposed control architectures are adapted to haptic (sens of touch) teleoperation systems with impedance types of master and slave manipulators.

The impedance of the master and slave device can be expressed as either a simple mass system [3&9] or a mass-damper system [10] as given by:

$$Z_m(s) = M_m s^2 + b_m s = \frac{F_h + F_m}{Y_m} \quad \textbf{and} \quad Z_s(s) = M_s s^2 + b_s s = \frac{F_s - F_e}{Y_s} \quad (1.11)$$

To explain equation (1.11), we can describe with a linear model the master and slave system separately as follows:

$$\left.\begin{aligned} M_m \frac{d^2Y_m}{dt^2} + b_m \frac{dY_m}{dt} &= F_m + F_h \\ M_s \frac{d^2Y_s}{dt^2} + b_s \frac{dY_s}{dt} &= F_s + F_e \end{aligned}\right\} \quad (1.12)$$

Where:

- $\boldsymbol{b_m}$ ***and*** $\boldsymbol{b_s}$: are the damping matrix.
- $\boldsymbol{M_m}$***and*** $\boldsymbol{M_s}$: are the masses of the master and slave manipulators.

By applying Laplace transform and introducing the impedance notion to the equations (1.11) we get:

$$\left.\begin{aligned} Z_m Y_m &= F_m + F_h \\ Z_s Y_s &= F_s + F_e \end{aligned}\right\} \quad (1.13)$$

With: $Z_m(s) = M_m s^2 + b_m s = \frac{F_h + F_m}{Y_m}$ ***and*** $Z_s(s) = M_s s^2 + b_s s = \frac{F_s - F_e}{Y_s}$

$Z_m(s)$: is the master impedance.

$Z_s(s)$: is the slave impedance.

See Figure-1.12-

In the illustration of ***Figure-1.12-***, the environment force ***Fe*** is defined as the contact force pushing on the slave device, opposite direction of the controller force ***Fs***. It is in contrast to the definition of the human force ***Fh*** in the model of the master device where ***Fh*** and ***Fm*** are the same direction.

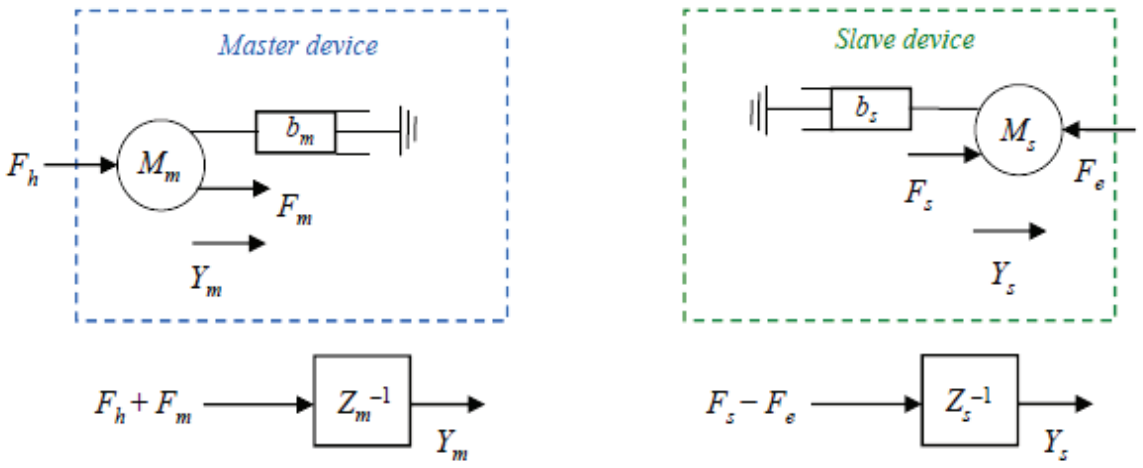


Figure-1.12-: Example of Master/ Slave device model

1.6.5) ***Transmission Line:***

In the bilateral teleoperation, the force and position signals are transmitted from the master to the slave and vice-versa. These signals pass through the communication line that, in general, cause time delays in the transmission.

It has been recognized that the presence of time delay is one of the most important barriers in teleoperation systems. This problem is mainly due to the distance separating the master from the slave site. To overcome this problem, many concepts, such as "***network theory***", passivity and scattering theory have been used [11]. The idea is to analyse mechanisms responsible for the loss of stability and derive a time delay compensation scheme to guarantee stability. See Section (1.8)

1.6.6) ***Formulation using a two-port presentation:***

We have already seen how we can represent the master and the slave subsystems using traditional differential equations with inertia, damping and forces involved, now we'll see how we can represent the whole teleoperation system as a 2-port network (*see figure-1.13-*), allowing for an easier analysis of its properties.

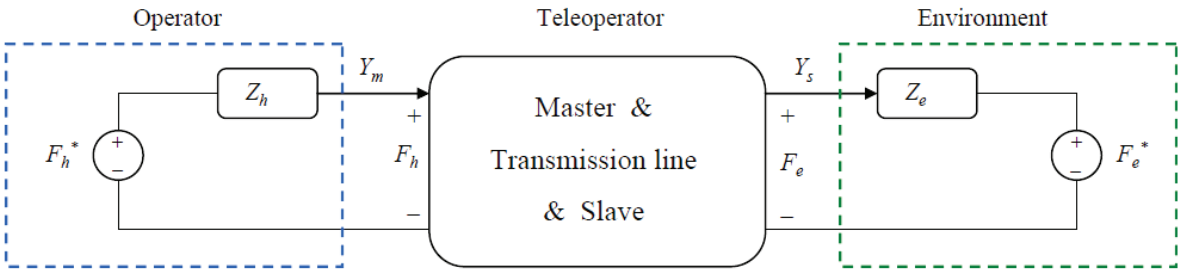


Figure-1.13-: Two-port network model of a teleoperation

The dynamics of the operator and the environment can be modeled by linear-time-invariant impedances ***Zh*** and ***Ze*** according to:

$$\left.\begin{aligned} F_h &= F_h^* - Z_h Y_m \\ F_e &= F_e^* - Z_e Y_s \end{aligned}\right\} \qquad (1.14)$$

Where:

F_h^* and **F_e^*** are the operator's and the environment's exogenous input forces, respectively, and independent of teleoperation system behavior. In most of cases, the operator as well as the environment are supposed to be passive (**$F_e^* = 0$).**

We can consider this as a ***mechanical/electrical*** analogy with the external signals being efforts and flows; for a mechanical system the efforts are the forces applied by the operator and the flows are for example the velocities, for an electrical system the efforts are the voltages and the flows are the currents.

A sign convention is also generally chosen as follows:

- The power will be positive if flowing for example from master to slave.
- The power negative in the opposite direction.

When the slave is in contact with the environment its forces and positions are related by the impedance that characterizes the environment:

$$F_s = Z_{te} Y_s \qquad (1.15)$$

$\boldsymbol{Z_{te}}$: Is the impedance transmitted to the environment.

A similar relation should link the forces and positions of the master:

$$F_m = Z_{to} Y_m \qquad (1.16)$$

$\boldsymbol{Z_{to}}$: is the impedance transmitted to operator.

To evaluate the transparency of teleoperation, the hybrid representation of the two-port network model of a ***master-slave system*** is most suitable [10]. In this representation, master position $\boldsymbol{Y_m}$ and slave force $\boldsymbol{F_e}$ are chosen as input [12]. (Note that we can choose velocity rather than position).

Then the relation between the inputs, outputs and hybrid matrix is given as follows:

$$\underbrace{\begin{bmatrix} F_h \\ -Y_s \end{bmatrix}}_{\text{Output Vector}} = \underbrace{\begin{bmatrix} h_{11} & h_{12} \\ h_{21} & h_{22} \end{bmatrix}}_{\text{H}} \underbrace{\begin{bmatrix} Y_m \\ F_e \end{bmatrix}}_{\text{Input Vector}} \qquad (1.17)$$

The ***H-matrix parameters"hij"*** are rational transfer functions, containing all the information about the device models and the controller and characterizing completely the teleoperation system (*Master/Slave dynamics controllers*).

By combining (1.9) and (1.11), $\boldsymbol{h_{ij}}$ can be calculated. Furthermore they can be obtained through two simple experiments where $\mathbf{F_e = 0}$ (in free space) and $\mathbf{Y_m = 0}$ (in hard contact).

Note that the tradition of choosing "**–** $\boldsymbol{Y_s}$ "as the output signal comes from the electrical linear representation, where positive currents are going into the network.

The essential desire is to provide a faithful transmission of signals (***positions, velocities, forces***) between master and slave to couple the operator as closely as possible to the remote task.

Ideally, the teleoperation system would be completely transparent, so operators feel that they are directly interacting with the remote task. This means that ***the master and the slave*** positions and forces will match regardless of the operator and environment dynamics:

$$Y_m = Y_s \; and \; F_h = F_e \tag{1.18}$$

$$Z_h = Z_{te} \;\; and \;\; Z_e = Z_{to} \tag{1.19}$$

Then transparency can be defined as:

$$Z_{to} = Z_{te} \tag{1.20}$$

By supposing that our system is passive we can write these following equations from figure-1.13-:

$$\left.\begin{aligned} Z_{to} &= \frac{F_h}{Y_m}|F_e^* = 0 \\ Z_{te} &= \frac{F_e}{Y_s}|F_e^* = 0 \end{aligned}\right\} \tag{1.21}$$

From the modal representation (1.17) we can write these equations:

$$\left.\begin{aligned} F_h &= h_{11}Y_m + h_{12}F_e \\ -Y_s &= h_{21}Y_m + h_{22}F_e \end{aligned}\right\} \tag{1.22}$$

From equation (1.21), we can rewrite this as follows:

$$F_h = h_{11}Y_m + h_{12}Z_eY_s \tag{1.23}$$

$$-Y_s = h_{21}Y_m + h_{22}Z_eY_s \tag{1.24}$$

Then equation (1.24) can be rewritten as:

$$Y_s = -\frac{h_{21}}{1+h_{22}z_e}Y_m \tag{1.25}$$

Then:

$$Z_{to} = \frac{F_h}{Y_m} = \frac{h_{11}Y_m + h_{12}F_e}{Y_m} = h_{11} + \frac{h_{12}Z_eY_s}{Y_m} \tag{1.26}$$

Replacing equation (1.25) in (1.26) leads to:

$$Z_{to} = h_{11} - \frac{h_{12}h_{21}Z_e}{1+h_{22}Z_e} \tag{1.27}$$

To achieve full transparency (equation 1.20), the hybrid matrix parameters must be as follows:

Referring to equation (1.27):

$$h_{11} = h_{22} = 0 \; and \; h_{12} = 1 \; and \; h_{21} = -1 \tag{1.28}$$

Which means that the ***H-matrix*** has the following form:

$$H(s) = \begin{pmatrix} Z_{in} & Force\ Scaling \\ Velocity\ Scoling & Z_{out}^{-1} \end{pmatrix}, \quad H_{ideal}(s) = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix} \tag{1.29}$$

Each element of the ***H-matrix*** has a physical meaning [16].

- $h_{11} = \frac{F_h}{Y_m}|F_e = 0$: is the input impedance in free-motion condition. Nonzero values for $\boldsymbol{h_{11}}$ mean that even when the slave is in free space, the user will receive some force feedback, thus providing a sticky feel of free-motion movements.
- $h_{12} = \frac{F_h}{F_e}|Y_m = 0$: is a measure of force tracking when the master is locked in motion (perfect force tracking when $\boldsymbol{h_{12} = 1}$).
- $h_{21} = \frac{-Y_s}{Y_m}|F_e = 0$: is a measure of position tracking performance when the slave is in free space (perfect position/velocity tracking when $\mathbf{h_{21} = -1}$).
- $h_{22} = \frac{-Y_s}{F_e}|Y_m = 0$: is the output admittance when the master is locked in motion. Nonzero values for $\boldsymbol{h_{22}}$ indicate that even when the master is locked in place, the slave will move in response to slave/environment contacts.

1.7) ***Basic control architectures:***

The teleoperator system under consideration is primarily an *impedance–impedance (see section 1.6.4)* teleoperator system where the master and slave are impedance devices. In a generic bilateral teleopeartor system, both force and position are communicated bilaterally between the master and slave devices.

For achieving the ideal response (equations (1.19&1.20)), various teleoperation control architectures are proposed in the literature. A general classification is based on the number of communication channels (***two, three or four channels***) that are required for transmitting position and force signals from the master to the slave and vice versa. The aim of this section is to provide some basic concepts of the different teleoperation architectures as well as an initial choice of their controller parameters.

1.7.1) ***Four channel (4CH) diagram:***

Figure-1.14-depicts the general ***4CH*** bilateral teleoperation architecture proposed by (Lawrence 1993) [5]. Here:

- ***Zh:*** operator impedance.
- ***Ze***: environment impedance.

- F_h^*: operator exogenous force input.
- F_e^* : environment exogenous force input.
- ***Cm***: master local position controller.
- ***Zm***: master impedance.
- ***Zs***: slave impedance.
- ***Cs***: slave local position controller.
- ***C1***: master coordinating force feedforward controller.
- ***C2***: slave force feedforward controller.
- ***C3***: master force feedforward controller.
- ***C4***: slave coordinating force feedforward controller.
- ***Vh***: master manipulator velocity.
- ***Ve***: slave manipulator velocity.

This architecture involves four types of data transmission between the master and the slave: ***force*** and ***position*** (***or velocity***) from the master to the slave and vice versa. (Assuring full transparency).

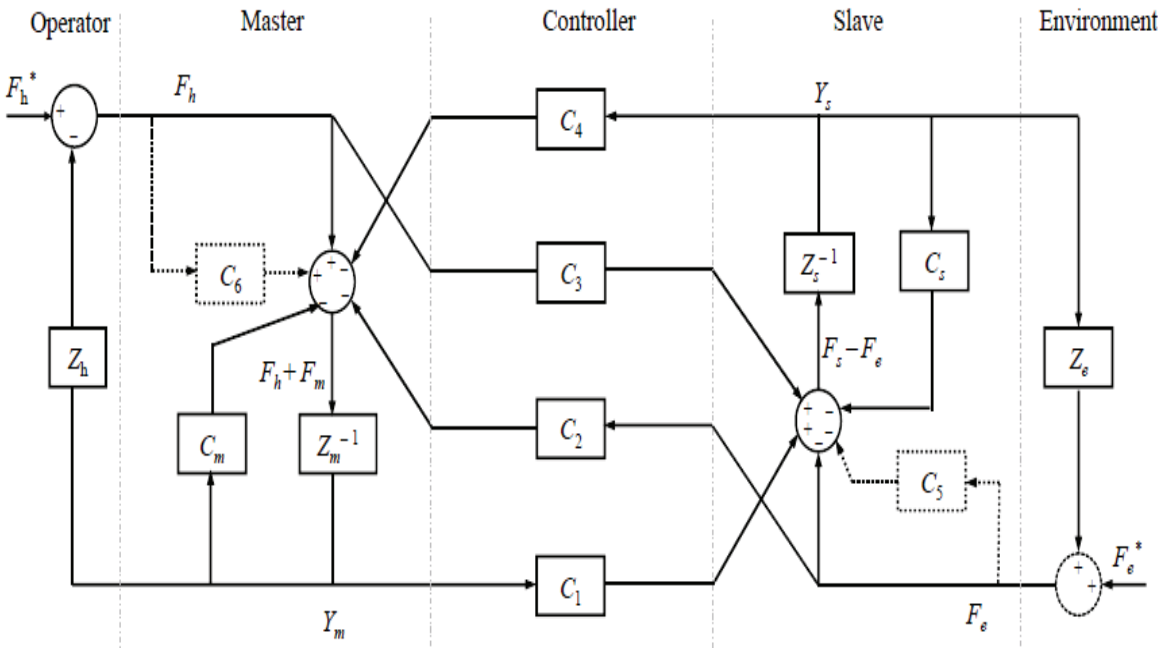


Figure-1.14-: 4CH bilateral controller (Lawrence 1993).

The controller forces F_m and F_s of the master and slave manipulations in Figure-1.14- are given by:

$$\left.\begin{aligned} F_m + F_h &= C_6F_h + F_h - C_mY_m - C_2F_e - C_4Y_s \\ F_s - F_e &= C_3F_h + C_1Y_m - F_e - C_5F_e - C_sY_s \end{aligned}\right\} \quad (1.30)$$

Simplifying:

$$\left.\begin{aligned} F_m &= C_6F_h + -C_mY_m - C_2F_e - C_4Y_s \\ F_s &= C_3F_h + C_1Y_m - C_5F_e - C_sY_s \end{aligned}\right\} \quad (1.31)$$

The h_{ij} parameters can be computed using (1.11) & (1.22), by combining these two equations and using the parameters of h_{ij} given in section (1.6.6) we can be lead to this result [3]:

$$\left.\begin{aligned} h_{11} &= \frac{Z_{ts}Z_{tm} + C_1C_4}{D} \\ h_{12} &= \frac{[Z_{ts}C_2 - (1 + C_5)C_4]}{D} \\ h_{21} &= -\frac{[Z_{ts}C_3 + (1 + C_6)C_1]}{D} \\ h_{22} &= -\frac{[C_2C_3 - (1 + C_5)(1 + C_6)]}{D} \end{aligned}\right\} \quad (1.32)$$

Where: $D = -C_3C_4 + Z_{ts}(1 + C_6)$

And: $Z_{ts} = Z_s + C_s, \quad Z_{tm} = Z_m + C_m$.

The elements *hij* depend both on the device hardware and the controller parameters chosen. As mentioned in (Lawrence 1993), a sufficient number of control parameters in the ***4CH*** architecture allows to achieve ideal transparency. In fact, from equation (1.32), by selecting ***C1*** through ***C6*** according to:

$$C_1 = Z_{ts}\,, C_4 = -Z_m, \quad C_6 + 1 = C_2, \quad C_5 + 1 = C_3 \qquad (1.33)$$

1.7.2) ***Two channel (2CH) diagram:***

The ***2CH*** diagrams can be represented by the ***4CH*** diagram through the appropriate selection of the controllers ***C1*** to ***C6***. This architecture involves two types of data transmission between the master and the slave: Position/velocity (or force) from the master to the slave and vice versa.

The 2CH architectures are usually classified as:

- *Position–force* (*i.e.*, position control at the master side and force control at the slave side)
- *Force–position (*i.e. force control at master side and position control at slave side*)*
- *Position–position (*i.e. position control at both master and slave side*)*
- *Force–force (*i.e. position control at both master and slave side*)*

Such architectures have been reported in a number of papers because they require fewer sensors and are less complicated to implement. In addition, due to the simplifications provided by eliminating two out of four data transmission channels, the analytical study of the 2CH systems can be easily achieved.

1.7.2.1) ***Position error Based (PEB):***

A position-error-based, also called ***position–position*** *(see figure-1.15b-)*, system involves the simplest bilateral controller in which no force sensors are required. It is usually the most cost effective solution to implement as well, because the only hardware requirement of the architecture is the position sensing on both manipulators.

As can be seen in ***Figure-1.15a***-, the aim of this architecture is to minimize the difference between the master and the slave positions.

In this simplest case, both robots are instructed to track each other. Both sites implement a tracking controller, often a proportional derivative (PD) controller.

This is a proportional-derivative controller, which attempts to make the slave follow the master's position *and* velocity:

In the PEB control architecture, which was further developed and analyzed in [6][8][13], the direct force feed-forward terms are set to zero (*i.e.*, $\mathbf{C_2 = C_3 = C_5 = C_6 = 0}$).

Then from equation (1.31) the control law becomes:

$$\left.\begin{aligned} F_m &= -C_m Y_m - C_4 Y_s \\ F_s &= C_1 Y_m - C_s Y_s \end{aligned}\right\} \qquad (1.34a)$$

For transparency the following parameters are chosen as following [5]:

$$C_4 = -C_m \; and \; C_1 = C_s \qquad (1.34b)$$

Where:

F_m:Master actuator force F_s:Slave actuator force

Y_m: Master's position Y_s: Slave's position

Then equations (1.31a and 1.31b) lead us to write:

$$\left.\begin{aligned} F_m &= C_m(Y_s - Y_m) \\ F_s &= C_1(Y_m - Y_s) \end{aligned}\right\} \qquad (1.34c)$$

Finally the equivalent hybrid matrix is obtained from equation (1.32) as follows:

$$H_{PEB} = \begin{bmatrix} Z_{tm} + \dfrac{C_1 C_4}{Z_{ts}} & \dfrac{C_m}{Z_{ts}} \\ \dfrac{-C_s}{Z_{ts}} & \dfrac{1}{Z_{ts}} \end{bmatrix}$$

These parameters depend on the dynamic of the system.

Every time the master's position is recorded, the slave robot attempts to follow the master using this control law (equation 1.34) and vice versa.

The PD controller is also used to compensate the modeling errors between the master and the slave

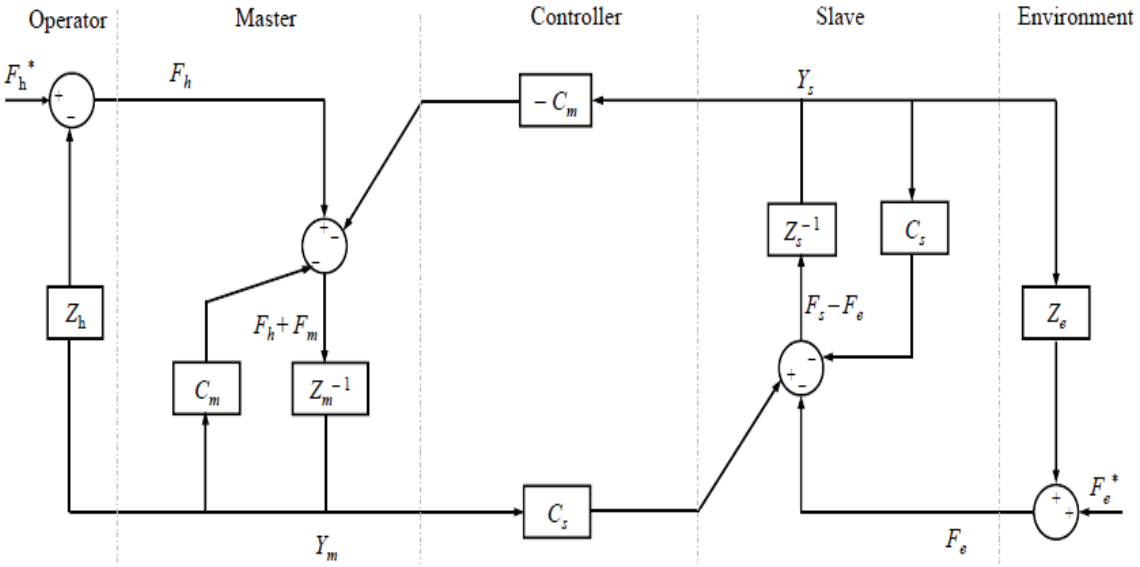


Figure-1.15a-: PEB bilateral control architecture

Blocks of the diagram denote different components affecting the control system.

By recalling that:

- ***Cm and Cs***: are the position controllers for both the master and the slave.
- ***Zm:*** is the master manipulator.
- ***Zh*** : is the impedance of the operator's hand.
- ***Zs:*** is the impedance of the slave manipulator.
- ***Ze:*** is the working environment of the manipulator.

The controller modeling of Cs, C1, Cm, C4 are given as follows:

$$\left.\begin{aligned} C_s = C_1 = B_s + \frac{K_s}{s} \\ c_m = -C_4 = B_m + \frac{K_m}{s} \end{aligned}\right\} \quad (1.35)$$

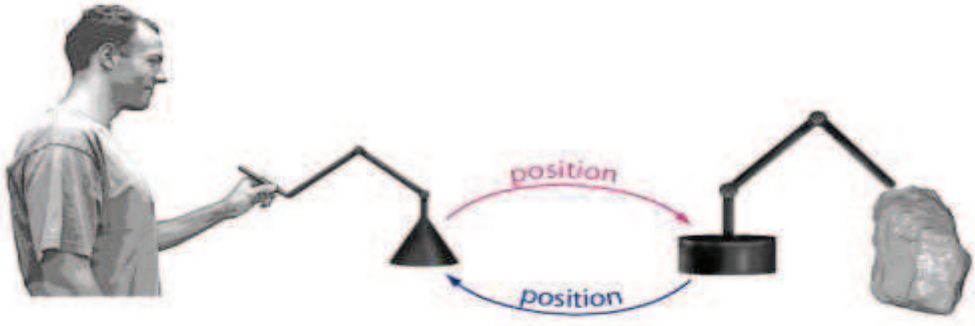


Figure-1.15b-: PEB bilateral control architecture

1.7.2.2) ***Force Error Based (FEB):***

A force-error-based: also called *force–force*, teleoperation architecture is shown in Figure-1.16-. The difference between this system and the PEB system is that the position sensors are replaced by the force sensors.

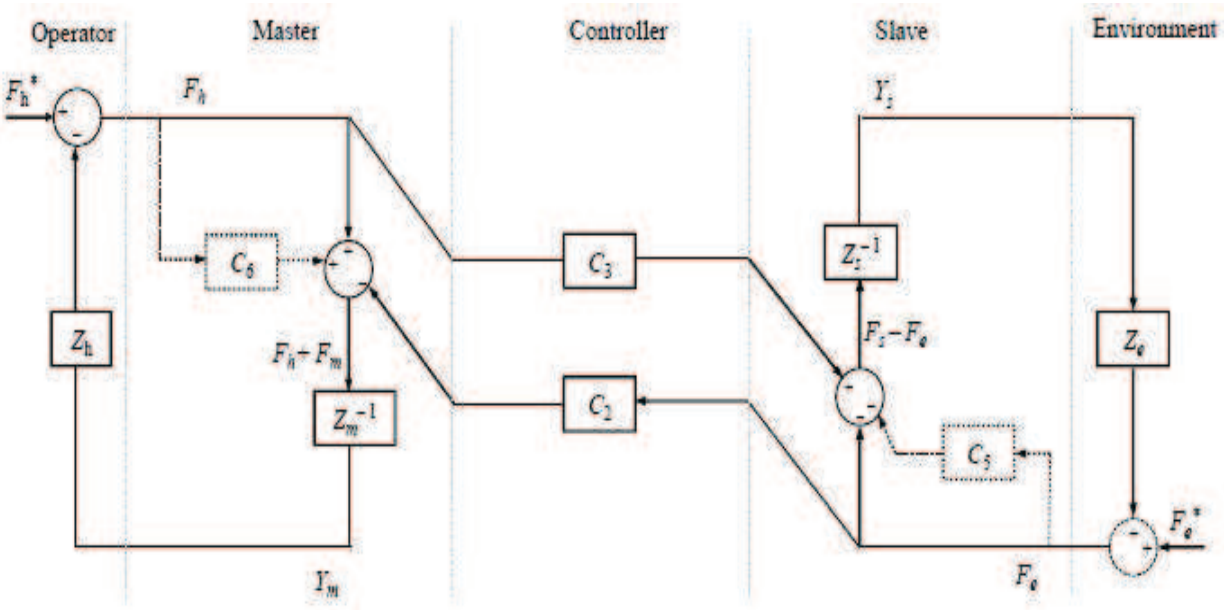


Figure-1.16-*:* FEB bilateral control architecture

The ***FEB*** architecture shows a good force tracking performance ($\mathbf{h_{12} = 1}$) in hard-contract motion ($\mathbf{Y_m = 0}$). Furthermore, an ideal value of the output admittance ($\mathbf{h_{22} = 0}$), in this architectures shows that when the master is locked in motion, the slave's movement in response to external force disturbances quickly converges to zero. Therefore an excellent stiffness is obtained for the slave manipulator.

The literature does not show much interest in the ***FEB*** controller since two force sensors are required with no significant performance improvement.

1.7.2.3) ***Direct Force Reflection:***

Direct-force-reflection, also called ***force–position***, teleoperation architecture is shown in Figure1.17. The DFR system has been developed, implemented, and analyzed by many researchers for the past three decades [14].

In this architecture, we get the controller as $\mathbf{C_3 = C_4 = 0}$, $\mathbf{C_2 = 1}$ and $\mathbf{C_1 = C_5}$.

Consequently, although the perception of free motion is still less than ideal ($\boldsymbol{h_{11} \neq 0}$), a perfect force tracking is attained ($\boldsymbol{h_{12} = 1}$). Nonetheless, compared to the ***PEB*** method, $\boldsymbol{h_{11}}$ is much closer to zero in the ***DFR*** method. Although the ***DFR*** method tends to be better than the ***PEB*** method, both

methods suffer from the non-ideal h_{22} value with respect to the ***FEB*** method, which results in poor slave stiffness

Finally, the ***DFR*** architecture takes advantage of both ***PEB*** and ***FEB*** architectures in terms of position and force tracking. Nonetheless, it is not suitable for applications with very hard contact [15], or requiring high bandwidth where the master and the slave display fast movements [16].

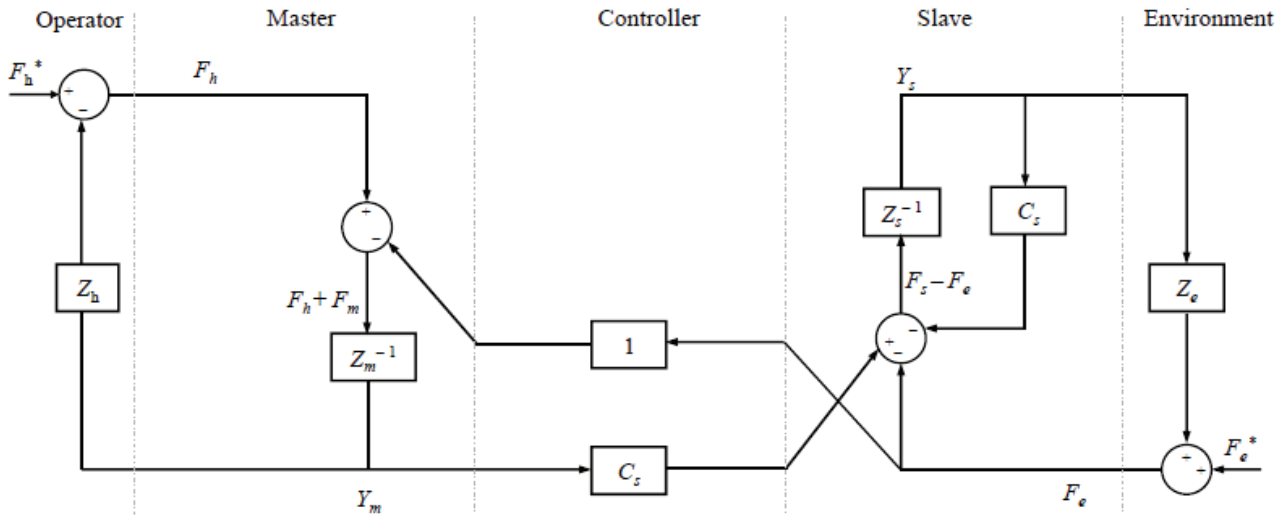


***Figure-1.17-*:** DFR bilateral control architecture.

1.7.2.4) ***Position-Force Architecture:***

The position-force (P-F) architecture (***traditional force-feedback)*** is the most intuitive one of the teleoperation architectures. The principle of the architecture is that the slave manipulator accurately follows movements of the master manipulator, and the master manipulator accurately repeats the forces sensed by the slave manipulator.

In this case, the slave manipulator has to be equipped with a force sensor that senses the forces and torques that are reflected from the teleoperation environment.

Implementing a teleoperation system with this architecture is generally more expensive than the ***position-position*** case, because force sensors are rather expensive and usually have to be installed specially for the bilateral teleoperation needs.

Figure-1.18- illustrates the concept of the ***position-force*** architecture with a block diagram, and the chosen controllers.

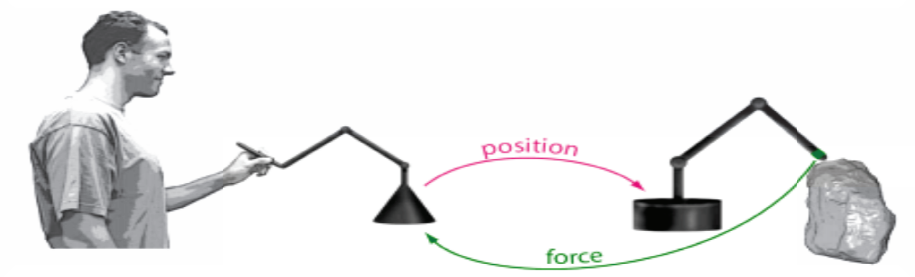
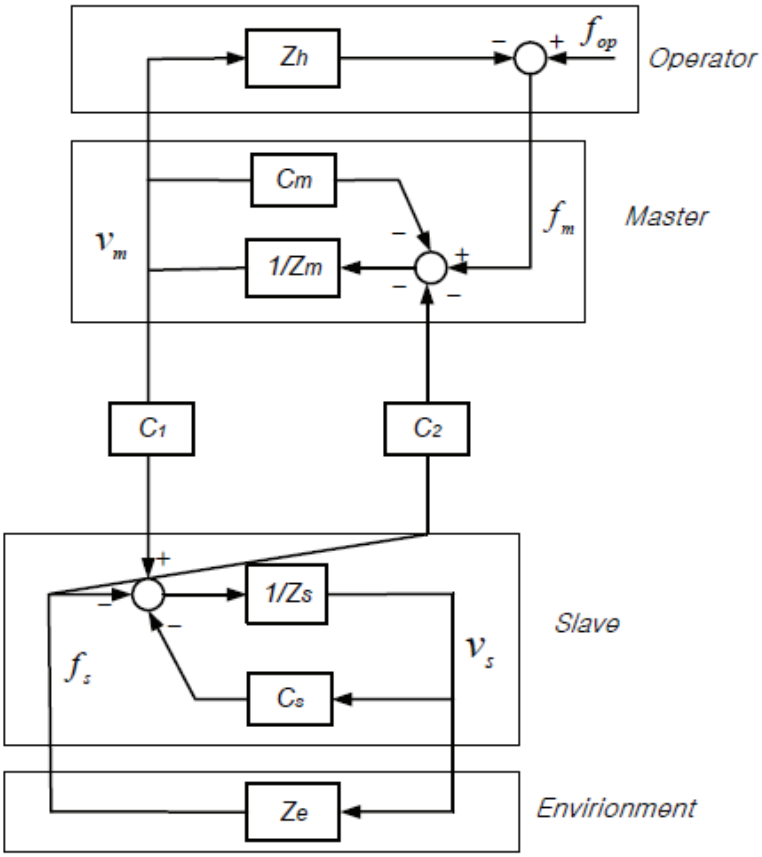


***Figure-1.18a-**:* external view of position-force exchanges.



$$\left.\begin{aligned} C_s = C_1 = B_s + \frac{K_s}{s} \\ C_2 = 1 \\ C_m = B_m \end{aligned}\right\} \quad (1.37)$$

***Figure-1.18b-**:* Block diagram presentation of the position-force teleoperation architecture, including external forces.

Position-Force architectures place a force sensor at the tip of the slave robot and feedback the force from there. That is the system is controlled by:

$$f_{am}(t) = f_e \quad (1.37)$$

Where:

$f_{am}(t)$: is the master actuator force

f_e: is the measured environment force

Important note: *in this section, we didn't take into consideration the time delay when representing the block architectures (see section (1.8) for time delay).*

1.8) ***Performances of teleoperation system:***

1.8.1) ***Tracking Error:***

The most straightforward way to see how well a teleoperator works consists in comparing the movements and forces of the master and the slave. The position and force tracking errors were also the first measures used to quantify teleoperator performance [9]. Position tracking is calculated when the slave robot is in free motion ($F_e = 0$), and force tracking when the slave robot is in contact with a hard object ($Y_m = 0$).

1.8.2) ***Bandwidth:***

Bandwidth is related to the information transfer between the operator and the remote environment. The requirements for information transfer in the two directions differs considerably [3]. The human operator gives force and movement commands with a relatively low frequency content, in the range of ***0-10 Hz***, whereas the contact information at the slave side often contains frequencies up to ***1KHz*** for stiff environments.

It is usually stated that the bandwidth should be as high as possible [17]. Each of the four transfer functions $h_{ij}(s)$ between forces and positions has its own bandwidth.

- This corresponds to position bandwidths from the master to slave ($\omega y,m \rightarrow s$), slave to master ($\omega y,s \rightarrow m$)
- Force bandwidths with similar notations (ωf,m→s and ωf,s→m).

1.8.3) ***Scaling Product:***

Scaling defines how forces and positions are magnified between the master and the slave. The force scaling and the position scaling are calculated from the ***H-matrix*** (1.17), and their product is called the ***scaling product.***

1.8.4) ***Transparency:***

The term "transparency" means the degree of telepresence associated to a teleoperation system. Therefore, in a system with a perfect transparency the operator of the system should feel as if he was manipulating the task directly, without the manipulators between him and the task. Transparency is the most common performance criterion used in literature. As mentioned in section (***1.6.6***), perfect transparency means that the master and the slave positions and forces will match regardless of the operator and environment dynamics equations see equations (1.18&1.19&1.20).
The transparency is defined as the quotient between the transmitted impedance and the remote impedance according to [3]:

$$T_z = \frac{Z_{to}}{Z_e} = \frac{h_{11}Z_e^{-1} + h_{11}h_{22} - h_{12}h_{21}}{1 + h_{22}Z_e} \qquad (1.38)$$

Perfect transparency implies Tz= 1 to ensure equation (1.20)

The transparency would ideally be unity for all frequencies. As we see in (1.29), transparency depends both on the teleoperator properties (***H-parameters***) and the remote impedance (Z_e) which is unknown.

1.8.5) ***Time Delay:***

Communication time delay between master and slave is very crucial in teleoperation.

It affects not only transparency, because the operator actions and feedback are delayed, but also stability.

So when dealing with teleoperation systems, we have to avoid the time delay because it influences the system performances.

1.8.6) ***Passivity:***

The general theory of passivity, when dealing with a bilateral teleoperation system, ensures that the energy being sent from the master to the remote system is always greater than or equal to the energy being sent back from the remote system to the master.

In a system where force and velocity are transmitted, passivity requires that the product of the force and velocity at the master side, or power in, minus the same combination at the slave site, or power out, must be a positive value. For this to be true, the communication link along with the slave system would have to be lossless or dissipative.

By definition, referring to the two port network in figure-1.19-, with initial energy storage E(0) = 0 is passive if and only if:

$$\int_0^t (f_h(\tau)v_m(\tau) - f_e(\tau)v_e(\tau))d\tau \geq 0, \forall t \geq 0. \qquad (1.39)$$

Holds for admissible forces ($\boldsymbol{f}$) and velocities ($\boldsymbol{v}$), where their product is defined to be positive when power enters the system port. Equation (1.30) states that the energy supplied to a passive network must be positive for all time.

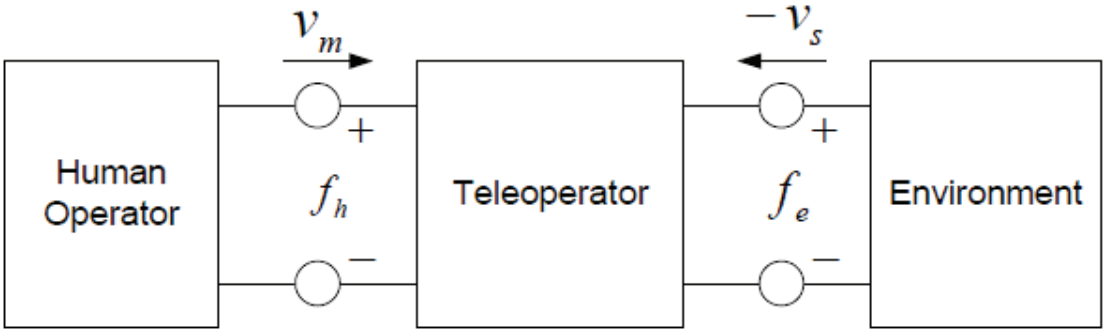


Figure-1.19-: One-port network system representing components

1.8.7) ***Wave Variables:***

When significant time delays occur in the teleoperated system, wave variables can be used to ensure stability [18]. This is a way of encoding the information to be sent in the communication channel of a teleoperated system in means of power and energy.

The wave variables are applicable to nonlinear systems and can handle large uncertainties which make it suitable for interaction with real physical environments.

Its structure is shown in figure-1.20-:

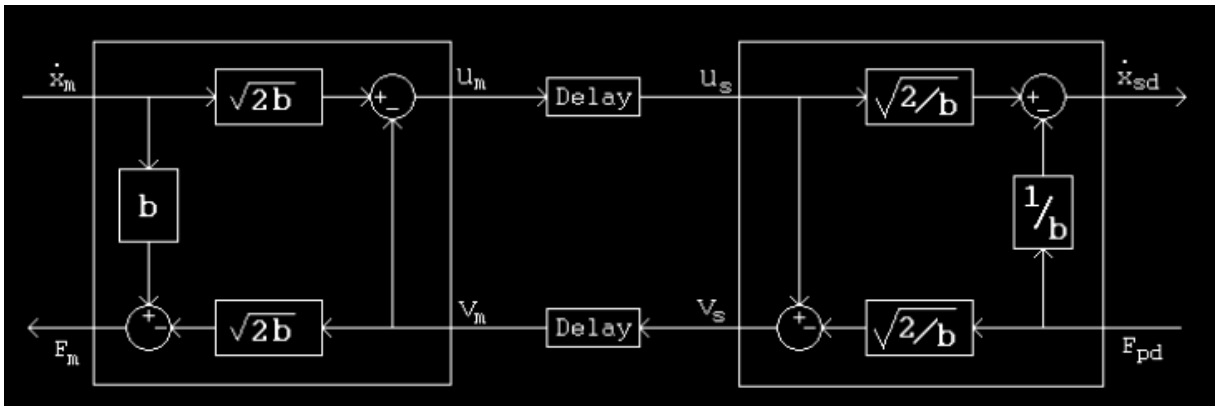


Figure-1.20-*:* Block diagram of a Wave Variable Communication

We recall that wave variables provide an alternative means for encoding information. The transforms that are required are very simple and still preserve all of the original information. The nature of these transforms, in combination with time delays, introduces damping elements that allow the system to perform within the levels defined by the delay limitations.

In the configuration shown in Figure-1.20-, velocity and force are transformed into wave variables before they are transmitted across the line. Once the wave variables are received the velocity and force information is extracted. Therefore, the equations that govern the transmission process are:

$$\left.\begin{aligned} U_s(t) &= U_m(t-T) \\ V_m(t) &= V_s(t-T) \end{aligned}\right\} \qquad (1.40)$$

Referring to figure-1.20-, the wave transformation equations for the left wave junction are:

$$\left.\begin{aligned} U_m(t) &= \frac{b\dot{x}_m(t) + F_m(t)}{\sqrt{2b}} \\ V_m(t) &= \frac{b\dot{x}_m(t) - F_m(t)}{\sqrt{2b}} \end{aligned}\right\} \qquad (1.41)$$

And the equations for the right wave junction are given by:

$$\left.\begin{aligned} U_s(t) &= \frac{b\dot{x}_{sd}(t) + F_{pd}(t)}{\sqrt{2b}} \\ V_s(t) &= \frac{b\dot{x}_{sd}(t) - F_{pd}(t)}{\sqrt{2b}} \end{aligned}\right\} \qquad (1.42)$$

Where $\boldsymbol{b}$ is the damping factor for the transformations

Some of the motivation for using wave variables is the guarantee of stability, which is also what makes this technique very conservative. As the performance of the system is a compromise between transparency and stability, the transparency when using wave variables are not the best. The stability is preserved by the passivity of wave variables, which is ensured by the condition that more energy has to be put into a system than that may come out of the system.

1.8.8) ***Stability:***

Stability is a prerequisite for performance. In essence, stability means that energy does not increase in any part of the system. Typically, instability looks like oscillations with increasing amplitude or that one part moves away uncontrollably in a certain direction. A teleoperation system can be often stable in some configurations (*in contact with a certain environment and a certain operator, in a certain point of the workspace*) while it is unstable for some other configurations.

To analyse the stability of the teleoperation systems, different definitions of stability have been proposed. The first concept is *bounded input-bounded output (BIBO*) stability. A system is said to be BIBO if every bounded input results in bounded outputs regardless of the system state [19].

Another concept of stability, from ***Lyapunov***, is that the output and all the internal variables never become unbounded when time tends to infinity for sufficiently small initial conditions. This is called ***asymptotic internal stability.***

Let's take the case with known operator and environment models:
In this section, only ***Lyapunov*** method is presented. Indeed, compared to other methods (Nyquist, Bode ....so no), the Lyapunov offers a possibility to deal with a nonlinear system.

Lyapunov stability is based on a state space model (1.23), where *x* is a state vector:

$$\dot{x} = f(x) \qquad (1.43)$$

It is assumed that the equations have been rewritten in such a way that $x = 0$ is an equilibrium point, *i.e.* $f(0) = 0$. The Lyapunov function $V(\mathrm{x})$ has to satisfy these following properties:

1. $V(0) = 0$

2. $V(x) > 0. \|x\| \neq 0.$

3. $V \text{ and its derivative are continuouswith respect to all components of } x.$

$\dot{V}(x) \leq 0.$

The theorem affirms that if there exists a Lyapunov function for the state equations and if, in addition $V(\dot{x}) < 0$, then the stability is asymptotic.

In the case of a linear system, the last condition mentioned can only hold if all the poles of the system are in the left half-plane.

<u>The Lyapunov stability criterion has the following advantages:</u>

- The criterion is applicable to nonlinear systems;
- The stability conditions are not conservative.

<u>Concerning the disadvantages:</u>

- The criterion cannot address the infinite-dimensional models generated by communication delays.
- Stability margins are not available.

1.9) ***Conclusion:***

This chapter has provided a brief overview of the teleoperation [20] system, in which some relevant points have been raised.

- The first point described a definition of teleoperation system by including a brief history and its modern applications, also the essential elements of teleoperation system which are fundamental to understand teleoperation control.

- The second point was about the classification of teleoperation control, where we have seen that the unilateral has no feedback effort (open loop system), however; the bilateral teleoperation [3] has feedback effort (closed loop system) as seen throughout section (1.5)

- The third point described a general linear model of the different blocks (i.e***., operator, environment, controller [21], transmission line, master, and slave)*** section (1.6) in a teleoperation system, allowing to explain how forces and movements are bilaterally transmitted from the operator to the environment and vice versa. For an easier performance analysis and controller design of the teleoperator, a representation of the two-port network was investigated so that the force and position relationship of the master and slave could be expressed through the ***H-matrix*** parameters (**h11, h12, h21, h22**).

- Then we have seen the transparent analysis and comparison of the most commonly used LTI teleoperation architectures (i.e. 2CH, or 4CH) where its classification is given by the number of communication channels. It has been established in the literature that the ***4-CH*** control architecture is the most successful in terms of ensuring full transparency [3][8].

On the other hand, none of other methods (*2CH*) such as *position error based (PEB)*, *force error based (FEB),* and *direct force reflection (DFR)* can achieve ideal transparency.

- In the last point, performance measurements and stability formulae that based on the linear model of hybrid matrix notation were investigated. It has been seen that there exist several performance criteria such as ***tracking error, bandwidths, scaling product, time delay, transparency***, etc. that allow to assess the above control architectures. It is noteworthy that stability in bilateral teleoperation commonly depends on the characteristics of both operator and environment.

REFERENCES

References:

[1]Ben-Dov, D., & Salcudean, S. E. A force-controlled pneumatic actuator. *IEEE Transactions on Robotics and Automation (ICRA)*, 1995 **11**, 906–911.

[2]http://www.ms.sapientia.ro/~martonl/Research_TE_Documents/Passive_Teleoperation.pdf

[3] D. Lawrence, "Stability and transparency in bilateral teleoperation," *IEEE Trans. Robot. Automat.*, vol. 9, pp. 624–637, October 1993.

[4]Tavakoli, M., Patel, R. V., & Moallem, M. *Haptics for Teleoperated Surgical Robotic Systems.* World Scientific, 2008, p.180.

[5]Zhu, W. H., & Salcudean, S. E. Stability guaranteed teleoperation: An adaptive motion/force control approach. *IEEE Transactions on Automatic Control*, 2000, **45**, 1951–1969.

[6]Aliaga, I., Rubio, A., & Sanchez, E. Experimental quantitative comparison of different control architectures for master-slave teleoperation. *IEEE Transactions on Control Systems Technology*, 2004, **12**, 2–11.

[7] L'Institut National des Sciences Appliquées de Lyon, Thése de doctorat "DEVELOPMENT OF BILATERAL CONTROL FOR PNEUMATIC ACTUATED TELEOPERATION SYSTEM".

[8]Fite, K. B., Speich, J. E., & Goldfarb, M. 2001. Transparency and stability robustness in two-channel bilateral telemanipulation. *Journal of Dynamic Systems, Measurement, and Control*, **123**, 400–407

[9]Yokokohji, Y., & Yoshikawa, T. Bilateral control of master-slave manipulators for ideal kinesthetic coupling-formulation and experiment. *IEEE Transactions on Robotics and Automation*, 1994, **10**, 605–620.

[10]Hannaford, B. A design framework for teleoperators with kinesthetic feedback. *IEEE transactions on Robotics and Automation*, 1989, **5**, 426–434

[11]Aziminejad, A., Tavakoli, M., Patel, R. V., & Moallem, M. Transparent Time-Delayed Bilateral Teleoperation Using Wave Variables. *IEEE Transactions on Control Systems Technology*, 2008 **16**, 548–555.

[12] Analysis of Control Architectures for Teleoperation Systems with Impedance/ Admittance Master and Slave Manipulators." http://web.stanford.edu/class/me327/readings/6-HashtrudiZaad01-IJRR-Teleop.pdf"

[13]Anderson, R. J., & Spong, M. W. Bilateral control of teleoperators with time delay. *IEEE Transactions on Automatic Control*, 1989, **34**, 494–501.

[14]Hannaford, B., & Anderson, R. 1988 April Philadelphia. Experimental and simulation studies of hard contact in force reflecting teleoperation. *IEEE International Conference on Robotics and Automation*, 1988, **1**, 584–589.

[15]Tavakoli, M., Aziminejad, A., Patel, R. V., & Moallem, M. 2007 July New York. Enhanced Transparency in Haptics-Based Master-Slave Systems. *American Control Conference (ACC '07)*, 2007, 1455–1460.

[16]Klomp, F. M. Haptic feedback control designs in teleoperation systems for minimal invasive surgery DCT Report No. 2004.117

[17]Fischer, P., Daniel, R., & Siva, K. V. 1990 May Cincinnati. Specification and design of input devices for teleoperation. *IEEE International Conference on Robotics and*

[18]Niemeyer, G. and Slotine, J. J. E. (2004). Telemanipulation with time delays. *The International Journal of Robotics Research*, 23.*Automation*, 1990, **1**, 540–545.

[19]Franklin, G. F., Powell, J. D., Emami-Naeini, A., & Powell, J. D. *Feedback control of dynamic systems.* Addison-Wesley Reading, 1994, p. 910.

[20] http://en.wikipedia.org/wiki/Teleoperation

[21] S. Salcudean, "Control for teleoperation and haptic interfaces," *Control Problems in Robotics and Automation LNCIS230, B. Siciliano and K.P. Valavanis (Eds.). Springer*, pp. 51–66, 1998.

Chapter II: Description of the plinth (the System)

2.1***) Introduction:***

This chapter focuses on the study of the system globally by describing the plinth which is the main part of our system and the related parts to it. Hence we will be focusing on how it can be controlled.

Well; our system main part represents the plinth which is the remote site to be controlled (the slave) via the master site consisting of the operator and a joystick (the master) to send control signals through transmission/reception line.

Throughout this chapter, we are about to see the teleoperation control system in which we are interested to, by defining in precise manner the functionality of each component and its role in the system.

2.2) ***Plinth definition:***

A plinth is known as supporting blocks used to hold objects for motion, in our case we define the plinth in control (see ***figure-2.1-)*** as a motorized support (mobile) that carry things upon such as a camera.

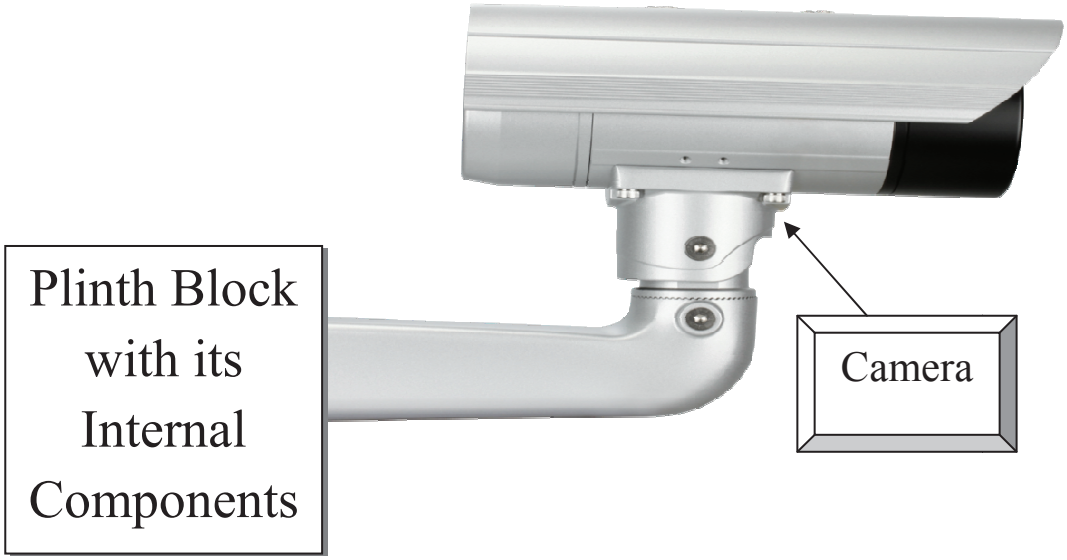


Figure-2.1-: Shows a motorized plinth holding a camera

2.3) ***Applications:***

Their applications can be found generally in control systems such as holding and manipulating objects such as cameras for public security known as tele-surveillance, they are widely used in remote control (space, underwater, ...so on) , and telerobototics for manipulating hard task,

2.4) ***The Plinth Components:***

The components of plinths differ from a constructor (such as: BOCH, SAMSUNG, SONY, HEDEN...so on) to another; each one uses its proper internal circuitry.

The standard internal structure of a plinth [1] (figure-2.1-) can be represented by the following figure-2.2-:

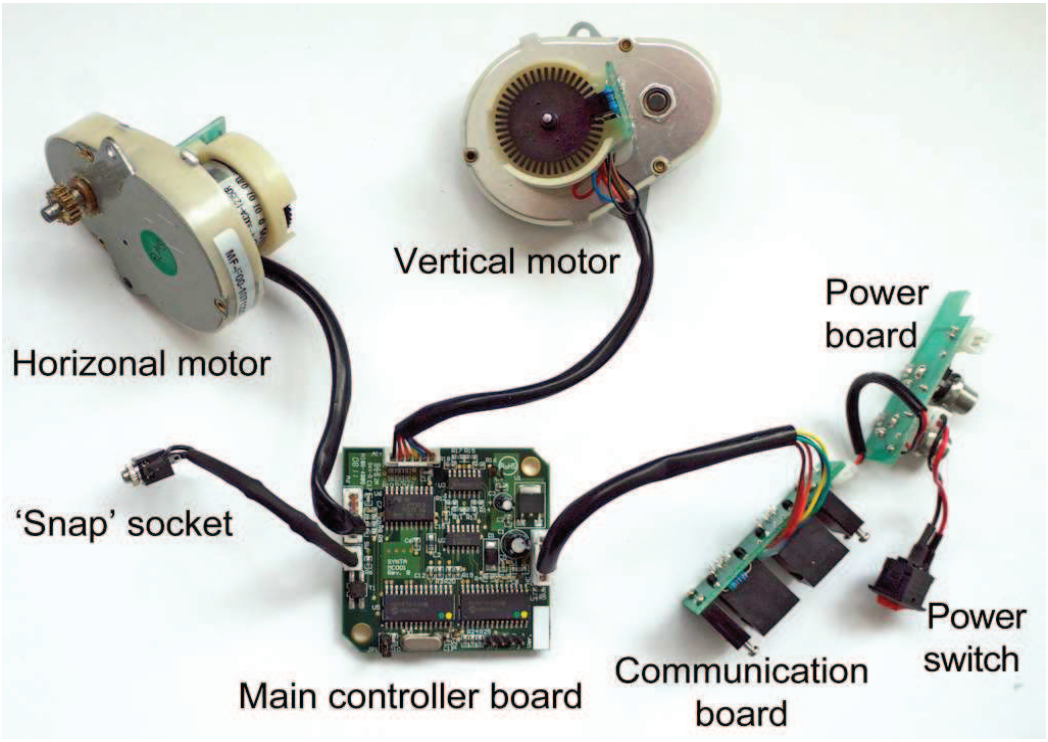


Figure-2.2-: The internal structure of a basic standard plinth

"It's important to note that the picture illustrated by figure-2.2- is given to show the model of the internal components of a standard motorized plinth of course by using specific components, hence; it stays a general example for us".

Referring to figure-2.2-, the main components of a plinth are:

- Main controller board known as "Electronic Based Board (EBB)."
- Motors. (Vertical and horizontal).
- Communication board.

And the power and switch board to keep the plinth either "On" or "Off".

2.4.1) ***The Electronic Based Board:***

In this section we are about to see the main components of an electronic based board which has an important role in our control system.

The essential components of an EBB are:

- Microcontroller.
- Input/Output ports
- Connection interface (serial ports).....so on

2.4.1.1) ***Microcontroller:***

2.4.1.1.1) ***Definition:***

A microcontroller [2] (sometimes abbreviated µC, uC) is a small *computer* on a single *integrated circuit* containing a processor core, memory, and programmable *input/output* peripherals. Another term for microcontroller is embedded controller, since most of the microcontrollers are built into (or embedded in) the devices they control.

Micro suggests that the device is small, and ***controller*** suggests that it is used in control applications.

2.4.1.1.2) ***Applications:***

Microcontrollers are used in automatically controlled products and devices, such as:

- Automobile engine
- Control systems,
- Remote control,
- Office machines and other ***embedded systems***.

By reducing the size and cost compared to a design that uses a separate microprocessor, memory, and input/output devices, microcontrollers make it economical to digital control even more devices and processes.

2.4.1.1.3) ***µC types:***

Microcontrollers architecture differs from one family to another, the most known families are: ***ARM, ATMEL, INTEL,***...and so on.

2.4.1.1.4) ***µC components:***

Microcontroller is used to describe a system that includes at minimum a [2] (see figure-2.3-):

- Microprocessor (Central Processing Unit)
- Program memory, data memory.
- Input-output (I/O) device.
- Interconnection structure (Buses).

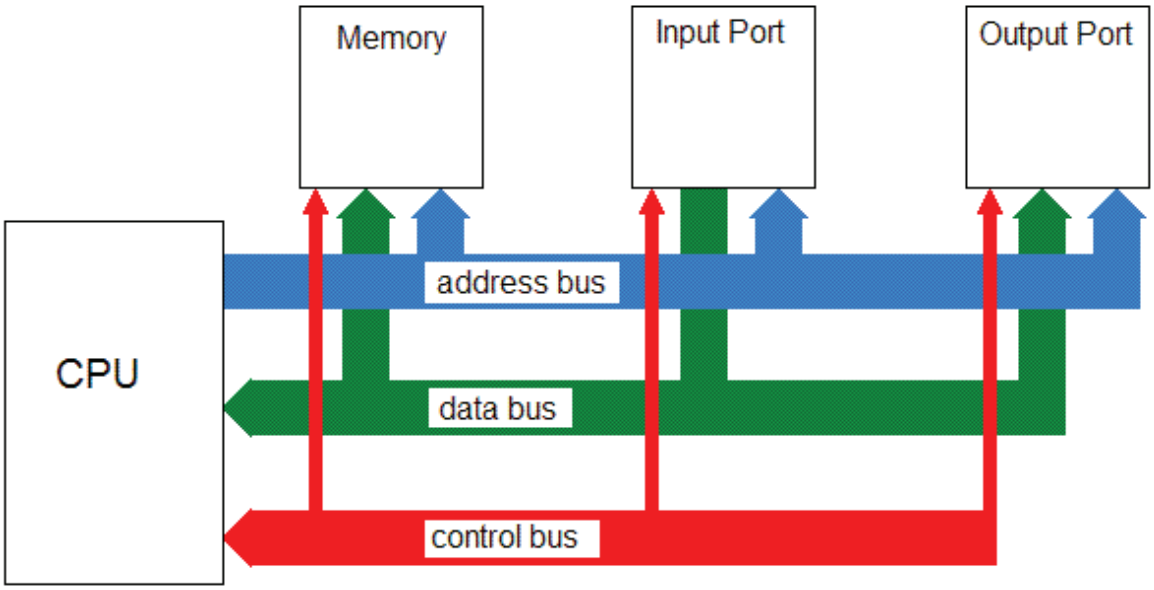


Figure-2.3-: Microcontroller block diagram

2.4.1.1.4.1)) ***Microprocessor (CPU)***

- ***Basic Definition:***

The ***µP*** is a semiconductor device capable of executing a program and making decisions to change the flow of execution. It is composed of [3] (see figure-2.4-):

- Control Unit,
- Arithmetic Logic Unit (ALU),
- Register Array and Internal Bus.

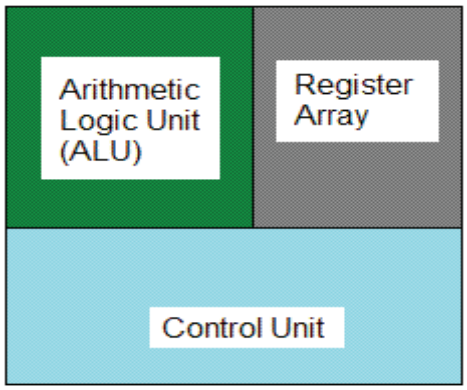


Figure-2.4-: uP main

A Program: is a sequence of steps which are called Instructions. For each step, an arithmetic or logical operation is done (e.g. ADD, MOVE). For each operation, a different set of control signals are needed. These instructions are stored in memory.

The ***µP*** Controls the operation of the computer and performs its data processing functions. When there is only one processor, it is often referred to as the central processing unit (CPU).

A microprocessor differs from a microcontroller in a number of ways. The main distinction is that:

- Microprocessor requires several other components for its operation, such as program memory and data memory, input-output devices, and an external clock circuit.
- A microcontroller, on the other hand, has all the support chips incorporated inside its single chip. All microcontrollers operate on a set of instructions (or the user program) stored in their memory.

A microcontroller fetches the instructions from its program memory one by one, decodes these instructions, and then carries out the required operations.

Figure-2.5-: shows the internal structure of a general processor core:

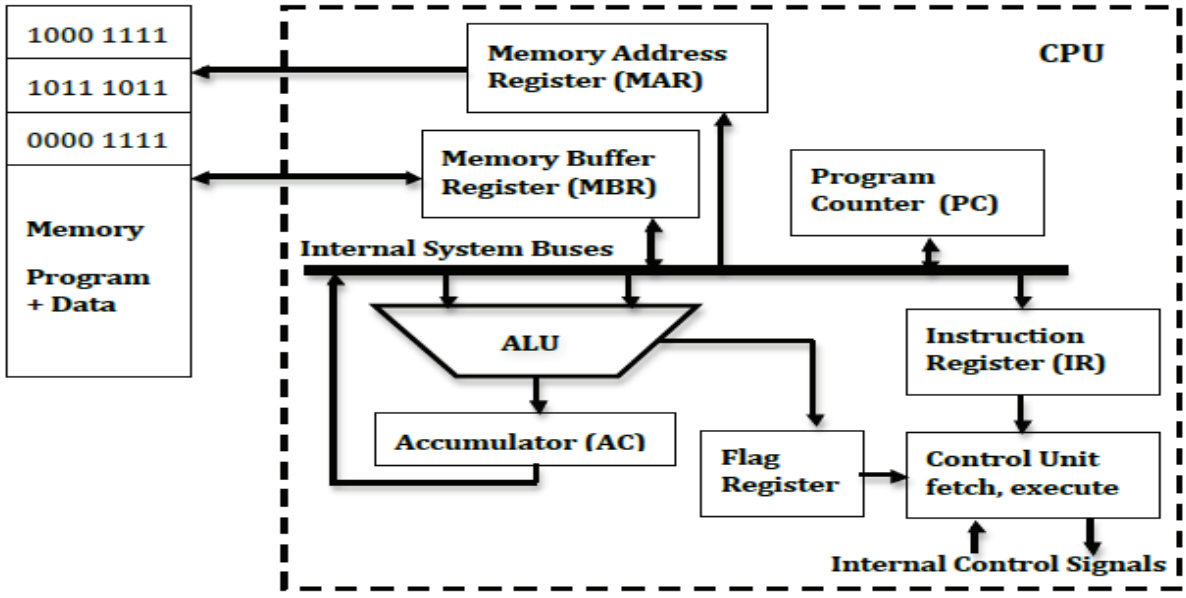


Figure-2.5-: The internal structure of a CPU

- ***CPU Components:***

The CPU is the brain of the microcomputer in general manner [3]; this is where all the arithmetic and logic operations are performed.

The ***µP***'s main components are (see figure-2.4):

a. ***The Control Unit (CU):***

The ***CU*** controls the internal operations of the microprocessor and sends signals to other parts of the microcomputer to carry out the required instructions.

Thus the CU fetches program instructions from memory (***Opcode Fetch Cycle***) and stores them temporarily in the Instructions Register (IR). These instructions are decoded by the operation decoder (***Decode Cycle***), which then asserts the internal control signals to the relevant parts of the μP via the Control Bus. These control signals causes the internal parts of the ***µP*** (***ALU)*** to carry out the required operation (Execute Cycle). The timing of these control signals is determined by the clock.

In other hand for each instruction takes 3 steps to be executed by μP:

- Fetch.
- Decode.
- Execute.

The major types of operations controlled by the control signals are:

- All computer functions which are: Data processing, Data storage, Data movement, Control
- Moving data from one part of the μP to another (***read, write cycles***).

- ***Input/output*** Operations ***(I/O cycles***).
- ***ALU*** calculations (+, -, or, nand, ...).
- Jumping to other instructions during program execution.
- Start/Stop/Reset ***µP*** operations..... And so on

b. Arithmetic Logic Unit (ALU):

This unit carries out the actual operations on data. These usually consist of arithmetic operation (+, -) and logical ones (and, or). These operations are performed as follows:

- Store data fetched from memory of ***I/O*** devices into the registers.
- Fetches this data as needed from the registers and/or from the accumulator.
- Perform the instructions using the arithmetic or logical circuitry.
- Send results to the accumulator, then to the memory of the ***I/O*** interfaces.

c. Register Array:

These registers are used to temporarily store data, memory addresses, or program codes until being sent to the ***ALU, Control Unit, or memory***. (See figure-2.5-).

d. ***Internal System Buses:***

The three components described above (CU, ALU, Registers) are connected together using three buses. Data in transferred internally and externally to the μP through these buses.

- ***Address Bus***: a unidirectional used by the μP to send addresses to the memory or I/O devices. The width of this bus (number of bits) specifies how much memory locations that can be addressed. A 16-bit address bus is capable of addressing 65536 (64K) addresses.
- ***Data Bus***: a bidirectional used to transfer data and/or instructions into the μP, or transfer back the results of operations to memory and/or I/O interfaces. The width of Data Bus varies from one μP to another.
- ***Control Bus***: used by the μP to send/receive timing and control signals to manage its operations and communicate with other devices such as memory and I/O.

2.4.1.1.4.2) ***Memory:***

Memory is a semiconductor device that stores binary information as instruction and data, which is an important part of a uC system. It can be classified into two types [2]:

- ***Program memory***: stores the program written by the programmer and is usually nonvolatile (***i.e., data is not lost after the power is turned off).***

- ***Data memory***: stores the temporary data used in a program and is usually volatile (***i.e., data is lost after the power is turned off***).

Figure-2.6- shows memory hierarchy classifying the types of memories

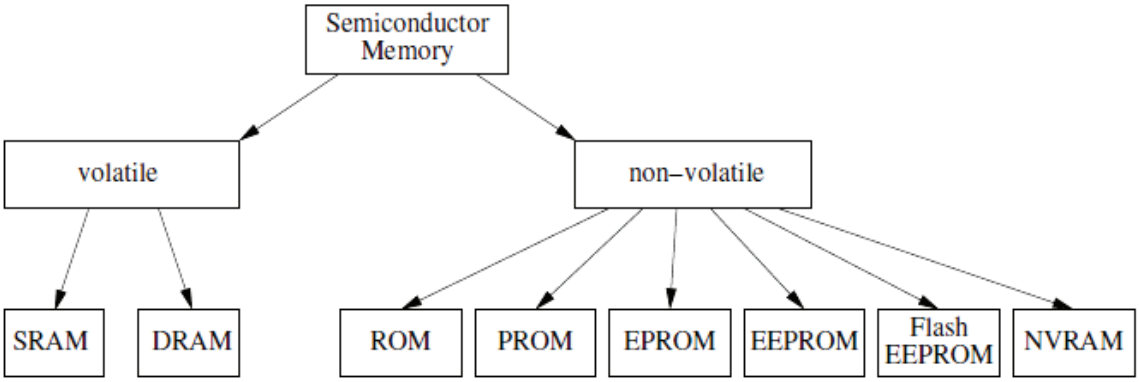


Figure-2.6-: Memory Hierarchy

There are basically six types of memories, summarized as follows:

- ***RAM (Read Only Memory):***

RAM, random access memory, is a general purpose memory that usually stores the user data in a program. ***RAM memory*** is volatile in the sense that it cannot retain data in the absence of power (***i.e. data is lost after the power is turned off***).

Two types are known:

- ***Static RAM (SRAM)***: The information remains stored as long as power is applied to the chip, unless the same location is written again.
- ***Dynamic RAM (DRAM***): The data stored at each location must be periodically refreshed by reading it and writing it back again, else it disappears.

- ***ROM (Read Only Memory):***

ROM, read only memory, usually holds program or fixed user data. ***ROM*** is nonvolatile. If power is removed from ***ROM*** and then reapplied, the original data will still be there. ***ROM*** memory is programmed during the manufacturing process, and the user cannot change its contents.

- ***PROM (Programmable Read Only Memory):***

PROM, programmable read only memory, is a type of *ROM* that can be programmed in the field, often by the end user, using a device called a *PROM* programmer. Once a *PROM* has been programmed, its contents cannot be changed.

PROMs are usually used in low production applications where only a few such memories are required.

- ***EPROM:***

EPROM, erasable programmable read only memory, is similar *to ROM*, *but EPROM* can be programmed using a suitable programming device. An *EPROM* memory has a small clear glass window on top of the chip where the data can be erased under strong ultraviolet light.

Once the memory is programmed, the window can be covered with dark tape to prevent accidental erasure of the data. An *EPROM* memory must be erased before it can be reprogrammed.

- ***EEPROM:***

EEPROM, electrically erasable programmable read only memory (see figure-2.12-), is a nonvolatile memory that can be erased and reprogrammed using a suitable programming device.

EEPROMs are used to save configuration information, maximum and minimum values, identification data, etc.

- ***Flash EEPROM:***

Flash ***EEPROM***, a version of ***EEPROM*** memory, has become popular in microcomputers' applications and is used to store the user program. ***Flash EEPROM*** is nonvolatile and usually very fast. The data can be erased and then reprogrammed using a suitable programming device.

Flash memories known as external hard drive, since they can stock data with power off, we can find them in flash disks and as already said, external hard drives.....so on

2.4.1.1.4.3) ***uC Input/ Output:***

The input section transfers data from the outside world to the microcontroller. It includes devices like: ***keyboard, switches, and ADC.***

The output section transfers data back from the *CPU* to other devices like ***LED***s, printer.....etc

2.4.1.1.4.4) ***System Buses:***

This is a communication path between the *μP* and the other peripheral in the microcontroller system [3]. It can be divided into *data, address*, and *control buses*.

Generally, all peripherals share the same buses, with only one device using the bus at any time. Devices are connected to the bus through tri-state buffers, as shown in figure-2.7:

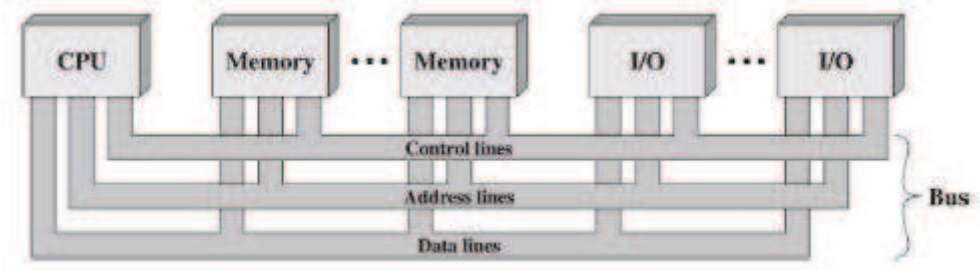


Figure-2.7-: System Buses of a general Microcomputer

2.4.1.1.4.5) ***Other Features:***

Microcontroller may include some other features as timers, UARTs, ADC, Reset, Interrupts serial ports....etc (see [2]).

Some of them are shown in figure-2.8- just below:

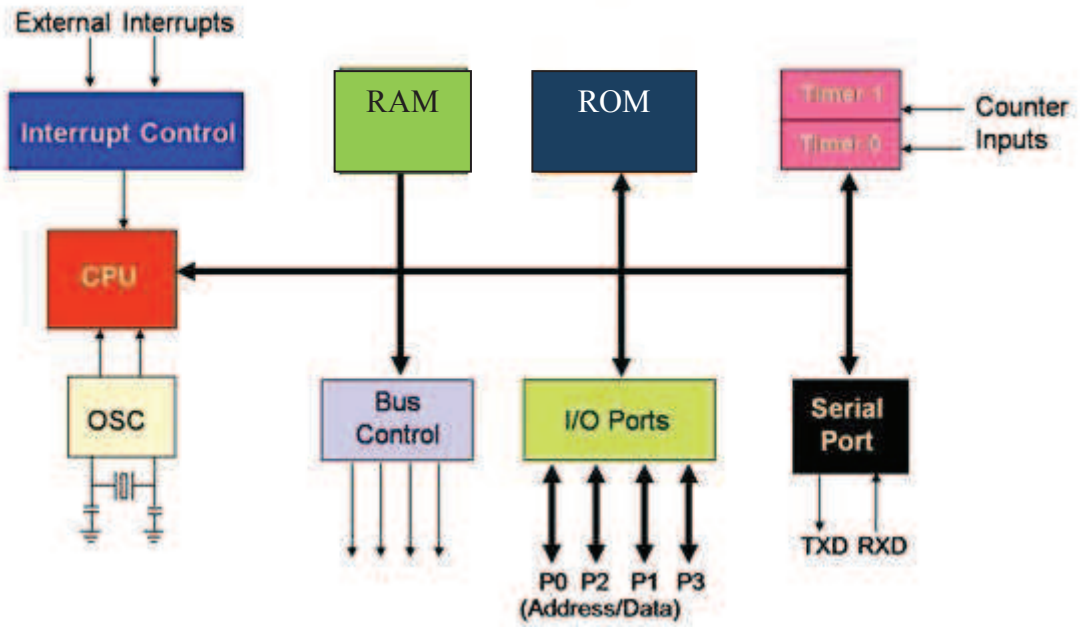


Figure-2.8-: Other features of a standard microcontroller

- ***The Clock:***

All microcontrollers require a clock (***or an oscillator***) to operate, usually provided by external timing devices connected to the EBB. In most cases, these external timing devices are a crystal plus two small capacitors. In some cases they are resonators or an external resistor-capacitor pair.

An instruction is executed by fetching it from the memory and then decoding it. This usually takes several clock cycles and is known as the ***instruction cycle.***

- ***Timers:***

Timers are important parts of any microcontroller. A timer is basically a counter which is driven from either an external clock pulse or the microcontroller's internal oscillator.

A timer can be ***8 bits*** or ***16 bits*** wide. Data can be loaded into a timer under program control, and the timer can be stopped or started by program control. Most timers can be configured to generate an interrupt when they reach a certain count (***usually when they overflow***). The user program can use an interrupt to carry out accurate timing-related operations inside the uC. The main functions of a timer are:

- Measuring the larger of a signal.

- Events Counting.
- Generating periodic signals.
- Providing base time.

- ***Reset Input:***

A reset input is used to reset the uC externally. Resetting puts this late into a known state such that the program execution starts from address 0 of the program memory. An external reset action is usually achieved by connecting a push-button switch to the reset input. When the switch is pressed, the whole device is reset.

- ***Interrupts:***

Interrupts are an important concept in microcontrollers. An interrupt causes the Microcomputer to respond to external and internal (***e.g. a timer***) events very quickly.

When an interrupt occurs, the microcomputer leaves its normal flow of program execution and jumps to a special part of the program known as the interrupt service routine (***ISR***). The program code inside the ***ISR*** is executed, and upon return from the ***ISR*** the program resumes its normal flow of execution.

- ***Analog to Digital Converter (A/D):***

An analog-to-digital converter (A/D) (***see figure-2.9-)*** is used to convert an analog signal, such as voltage, to digital form so a uC can read and process it.

Of course the characteristics of A/D differ from uC to another.

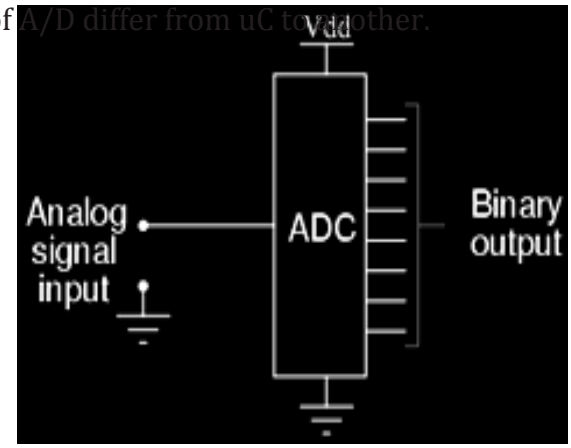


Figure-2.9-*:* Example of *A/D* convertor

- ***Serial ports:***

Serial ports allow microcontrollers to communicate serially (sending data bit by bit). It is a point at which an external device (peripheral) attaches to the computer system (microcontroller in our case). Ports allow data to be sent/retrieved from the external devices.

To accomplish the necessary translation between bytes and bits, another piece of hardware is required – the UART.

- ***UART (Universal Asynchronous Receiver Transmitter):***

A UART (***Universal Asynchronous Receiver Transmitter***) is a device allowing the reception and transmission of information, in a serial and asynchronous way [4-5].

A UART allows the communication between a computer and several kinds of devices (***printer, modem, etc***), interconnected via an ***RS-232*** cable. (see section for RS-232).

UART receives 8-bit data from the processor then performs parallel to series conversion to send it bit by bit via ***TxD*** line.

Also it receives via the ***RxD*** line serial data to perform the inverse operation to provide these to processor via the bus. As shown in ***figure-2.10***- just below:

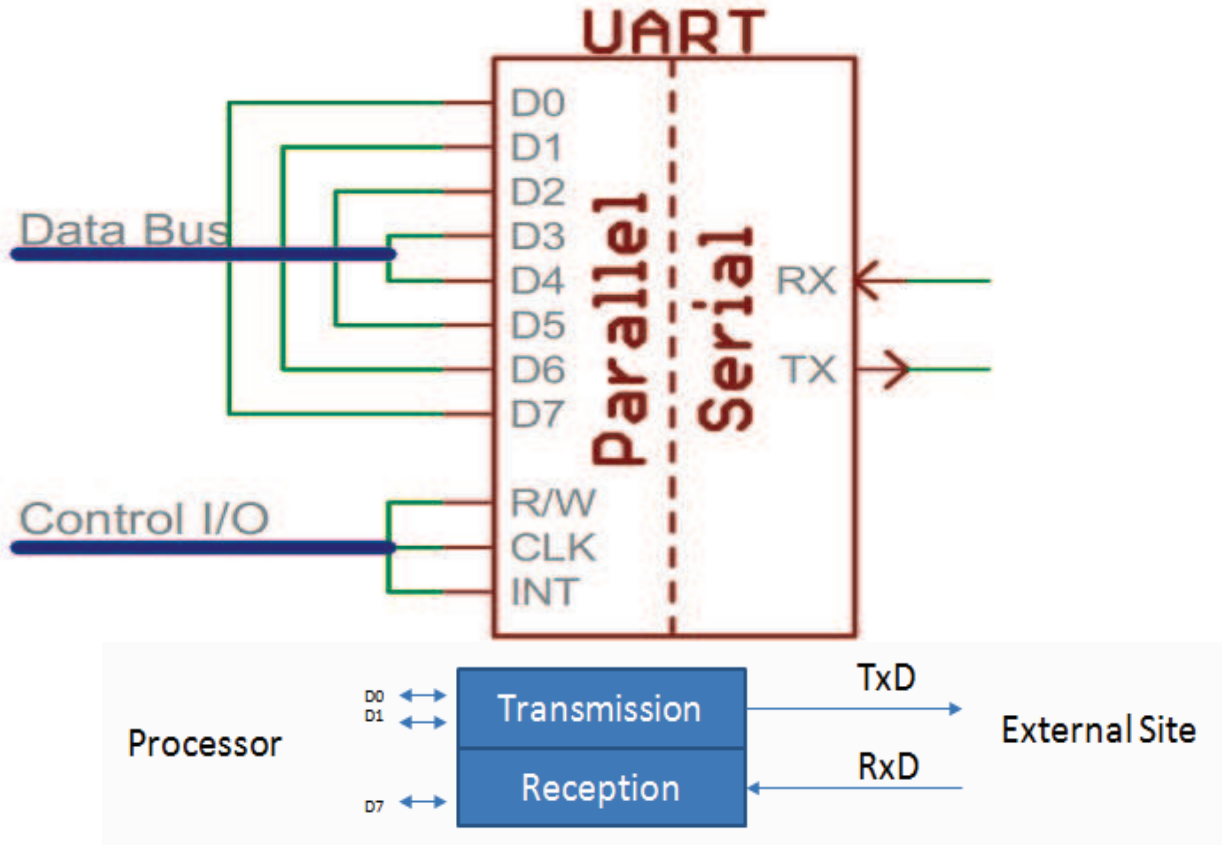


Figure-2.10-: UART Structure

The transmission is asynchronous (see section), so the transmitter and receiver internal clocks have to be in the same cadence

The most transmission speeds used are:

✓300 bps ✓2 400 bps ✓19 200 bps ✓57 600 bps
✓1 200 bps ✓9 600 bps ✓38 400 bps ✓115 200 bps

The most used is 9600.

Data transmission is made by the ***UART*** in a serial way, by ***11-bit*** blocks (see figure-2.11-):

- A"0" bit marks the starting point of the block
- Eight bits for data
- One parity bit
- A 1 bit marking the end of the block.

The transmission and reception lines should hold a 1 when no data is transmitted.

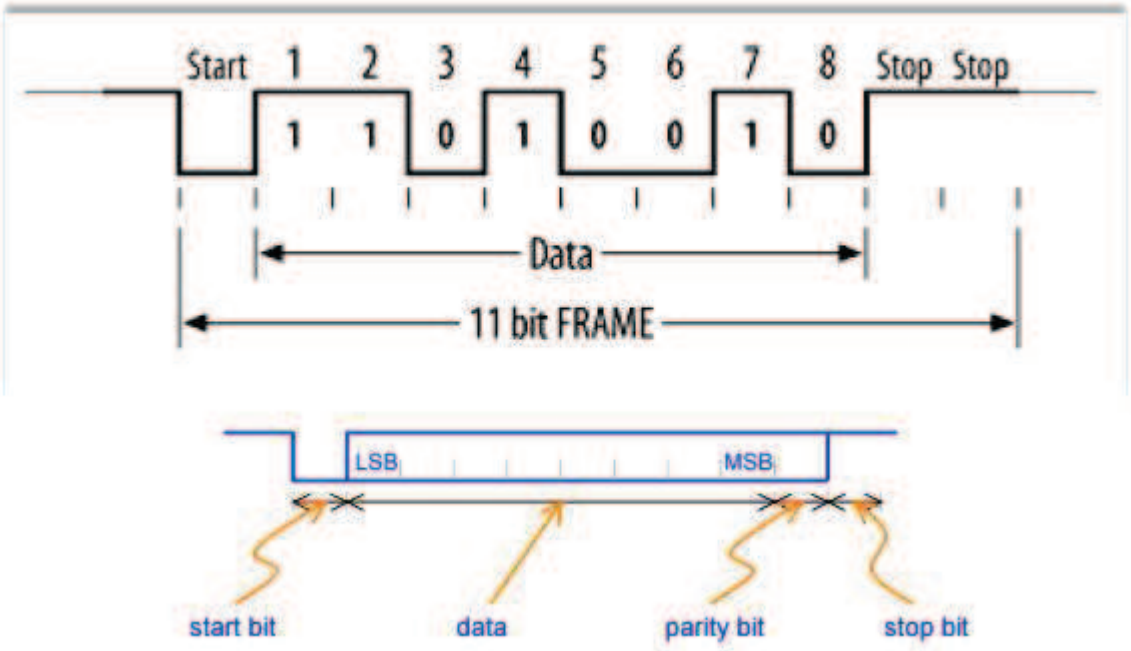


Figure-2.11-: Transmission characteristics of a ***UART***

2.4.1.2) ***Input/output ports of the main controller board:***

As uC, the EBB has input/output ports to be connected with other devices as shown in figue-2.2- where the EBB is connected to both motors and communication board to ensure connection between them and data exchanging.

The number of Is/Os can be either analog or digital; this differs from a manufacturer to another.

2.4.2) ***Communication Board:***

The communication board is incorporated inside the plinth letting this latte especially the EBB to receive and transmit data from/to external devices that can be subsystems. Internally this board is a part of the EBB since it is used to ensure communication between the main controller board and other devices for data exchanging. The different methods used for communication between EBBs or systems are described in section (2.5.2).

2.4.3) ***Motors:***

This part contains the motors that are controlled by the operator (Master Site) to make move the plinth in the two axes (***x*** (***Left-Right***) and ***y*** (***Up-Down***)).

So for each axe we have a motor (vertically and horizontally) see figure-2.2-.

We can either use "*Stepper Motors*" or "*Servomotors*".

2.4.3.1) ***Stepper Motor:***

2.4.3.1.1) ***Definition:***

A ***stepper motor*** *(see figure-2.12-)* is an electromechanical device which converts electrical pulses into discrete mechanical movements [6]. The ***shaft or spindle*** of a stepper motor rotates in discrete step increments when electrical command pulses are applied to it in the proper sequence.

The motors rotation has several direct relationships to these applied input pulses. The sequence of the applied pulses is directly related to the direction of motor shafts rotation. The speed of the motor shafts rotation is directly related to the frequency of the input pulses and the length of rotation is directly related to the number of input pulses applied.

Figure-2.12-: External view of a Stepper Motor

2.4.3.1.2) ***Applications:***

Stepper motors were used as early as the ***1920s***; their use has risen suddenly to a high level with the advent of digital computer. Thus they are the most widely used control motor today.

Their application domains are:

- Industrial.
- Military.
- Medical
- Automation....so on.

2.4.3.1.3) ***Advantages and disadvantages***:

- ***Advantages:***

- The rotation angle of the motor is proportional to the input pulse.
- Precise positioning and repeatability of movement since good stepper motors have an accuracy of 3 – 5% of a step and this error is non cumulative from one step to the next.
- Excellent response to starting/ stopping/reversing.
- Very reliable since there is no contact of brushes in the motor. Therefore the life of the motor is simply dependant on the life of the bearing.

- The motors response to digital input pulses provides open-loop control, making the motor simpler and less costly to control.
- It is possible to achieve very low speed synchronous rotation with a load that is directly coupled to the shaft.
- No serious stability problems exist.
  - ***Disadvantages:***
- Resonances (oscillations at a specific frequency) can occur if not properly controlled.
- Not easy to operate at extremely high speeds.

2.4.3.1.4) ***Basic Wiring diagram:***

The stepper motor can be controlled using adequate EBB such as (microcontroller). Stepper motors can be classified according to their characteristics thus; they can be unipolar or bipolar, single-phase, multi-phase and so on. In this section we will focus on the functionality of the standard two-phase unipolar stepper motor which is used widely in automation control.

To work with the unipolar stepper motor, the common points (see figure-2.13-) are connected to either Ground or Vcc and the end points of both the phases (Blue and Red wires) are usually connected through the port pins of a microcontroller [31]. In figure-2.13- shown below the green wires are common. The end points receive the control signals as per the controller's output in a particular sequence to drive the motor.

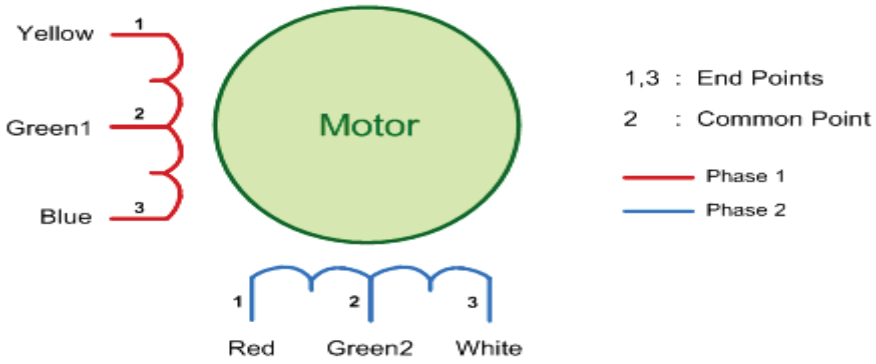


Figure-2.13-: *Lead unipolar stepper motor*

Since the coils related to each phase are arranged in alternate manner, the end points of two phases are energized in alternate fashion to rotate the motor. This means that the voltage signal should be applied to first end point of Phase1 and then to the first end point of the Phase2 and so on.

2.4.3.1.5) ***Step Sequencing:*** *(Concerning unipolar stepper motor)*

There are three modes of operation available for all types of stepper motor. The mode of operation is determined by the step sequence applied. The three step-sequences are [7 (see description)-8]:

- Wave
- Full ***H = HIGH = +V***
- Half Stepping ***L = LOW = 0V***

Wave drive Stepping mode:

The direction of rotation can be clockwise or anti clockwise depending upon the selection of end points.

The wave stepping sequence is shown in ***figure-2.14-*** just below.

Signal sequence for Wave Drive Stepping Mode				
Step	**Yellow lead** (End point 1 of Phase1)	**Blue lead** (End point 2 of Phase1)	**Red lead** (End point 1 of Phase2)	**White lead** (End point 2 of Phase2)
1	**1**	0	0	0
2	0	0	**1**	0
3	0	**1**	0	0
4	0	0	0	**1**

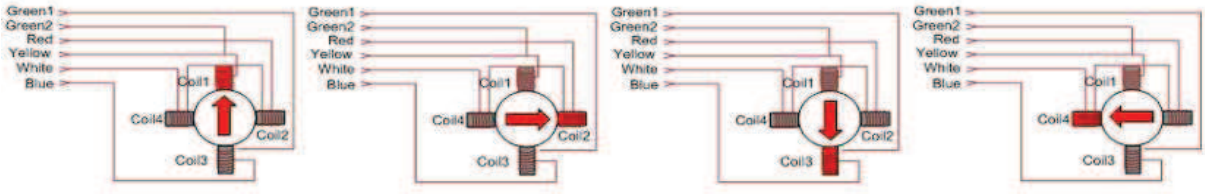


Figure-2.14-: the basic operation of wave stepping

Wave stepping has less torque then full stepping. It is the least stable at higher speeds and has low power consumption.

Full drive Stepping mode:

The Full Drive Stepping can be achieved by energizing two endpoints of different phases simultaneously. The full stepping sequence is shown in ***figure-2.15-*** below.

Signal sequence for Full Drive Stepping Mode				
Step	**Yellow lead** (End point 1 of Phase1)	**Blue lead** (End point 2 of Phase1)	**Red lead** (End point 1 of Phase2)	**White lead** (End point 2 of Phase2)
1	**1**	0	**1**	0
2	0	**1**	**1**	0
3	0	**1**	0	**1**
4	**1**	0	0	**1**

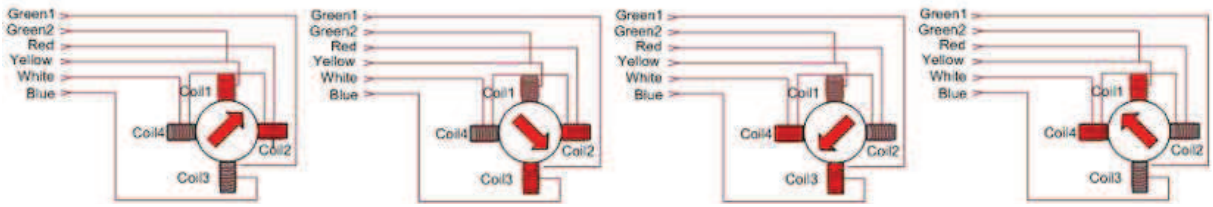


Figure-2.15-: Full Stepping basic operation

Full stepping has the lowest resolution and is the strongest at holding its position. Clock-wise and counter clockwise rotation is accomplished by reversing the step sequence.

- ***HALF-STEPPING – A COMBINATION OF WAVE AND FULL STEPPING:***

The Half Drive Stepping is achieved by combining the steps of Wave and Full Drive Stepping Modes. This divides the stepping angle by half.

The half-step sequence is in ***figure-2.16-*** shown below.

Signal sequence for Half Drive Stepping Mode				
Step	**Yellow lead** (End point 1 of Phase1)	**Blue lead** (End point 2 of Phase1)	**Red lead** (End point 1 of Phase2)	**White lead** (End point 2 of Phase2)
1	**1**	0	0	0
2	**1**	0	**1**	0
3	0	0	**1**	0
4	0	**1**	**1**	0
5	0	**1**	0	0
6	0	**1**	0	**1**
7	0	0	0	**1**
8	**1**	0	0	**1**

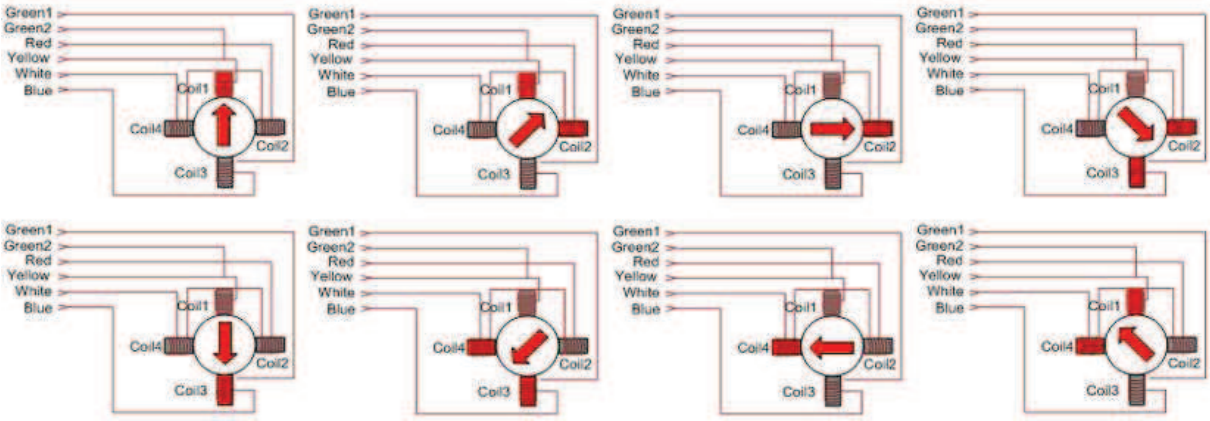


Figure-2.16-: Half-Step basic operation

The ***half-step sequence*** has the most torque and is the most stable at higher speeds. It also has the highest resolution of the main stepping methods.

2.4.3.2) ***Servomotor:***

2.4.3.2.1) ***Definition:***

A servomotor (see figure-2.17-) is a *rotary actuator* that allows for precise control of angular position (*generally it rotates from 0 to 180 degrees*), velocity and acceleration, it's also known as a brushless dc motor combined with a position sensing device (e.g. *a digital decoder*)[9].

It consists of a suitable motor coupled to a sensor for position feedback. It also requires a relatively sophisticated controller; often a dedicated module designed specifically for use with servomotors, CNC (Computer Numerical Control) machinery or automated manufacturing.

Figure-2.17-: shows Parallax standard Servo motor

2.4.3.2.2) ***Main components:***

A DC servo motor incorporates a [10-11]: (see figure-2.18-)

- DC motor.
- Gear reduction unit known as a toothed part that transmits.motion.
- Potentiometer for position feedback.
- Integrated circuit for position control.

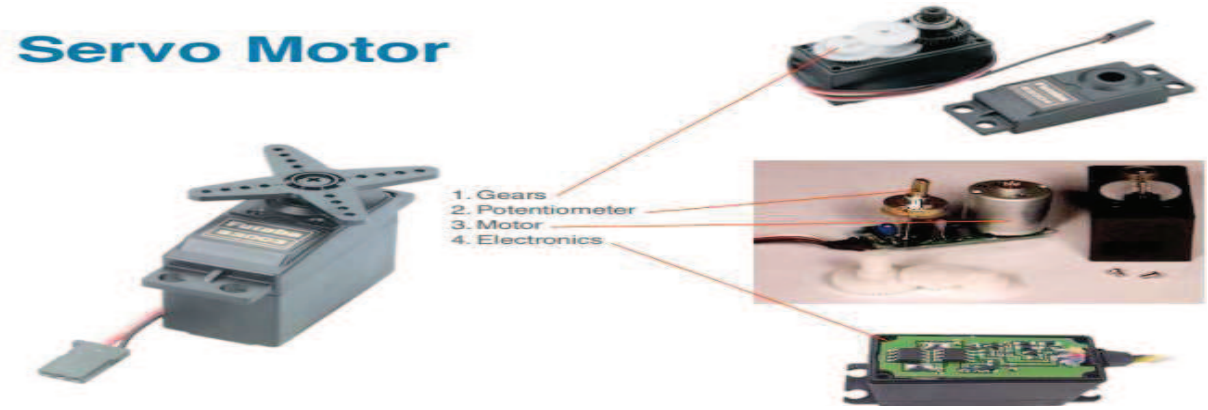


Figure-2.18-: shows the main components of a standard Servo motor

The motor is attached by gears to the control wheel. As the motor rotates, the potentiometer's resistance changes, so the control circuit can precisely regulate how much movement there is and in which direction.

A standard servo motor has three wires [11] that are (see figure-2.19-):

- Black for the ground
- Red for power
- Yellow for control

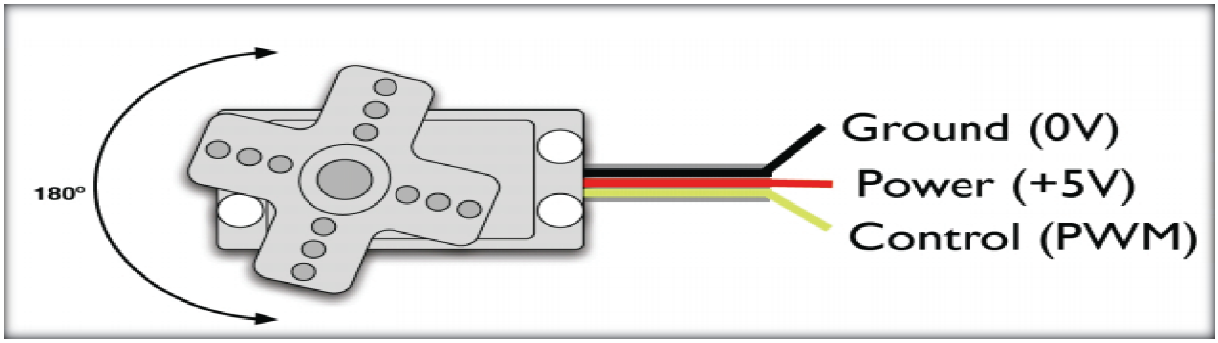


Figure-2.19-: Three-wire DC servo motor

An Example of wiring a servomotor using *microcontroller* is shown in figure-2.20:

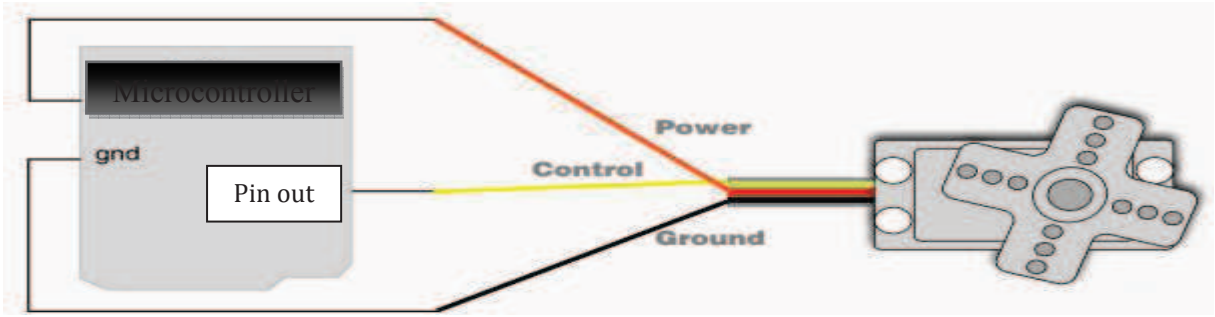


Figure-2.20-: shows an example of a wired servomotor with microcontroller

2.4.3.2.3) ***PWM (Pulse Width Modulation):***

Pulse-width modulation (***PWM***), as it applies to motor control, is a way of delivering energy through a succession of pulses rather than a continuously varying (analog) signal. By increasing or decreasing pulse width, the controller regulates energy flow to the motor shaft.

Figure-2.21- shows the principal working of a ***PWM*** signal. [12].

The input red signal to the minus polarity of the comparator known as sawtooth signal, this latter is compared to the blue sinusoidal signal as follow:

- If blue greater than red than the output to comparator is +5V or "1" logic.
- Else the output is zero.

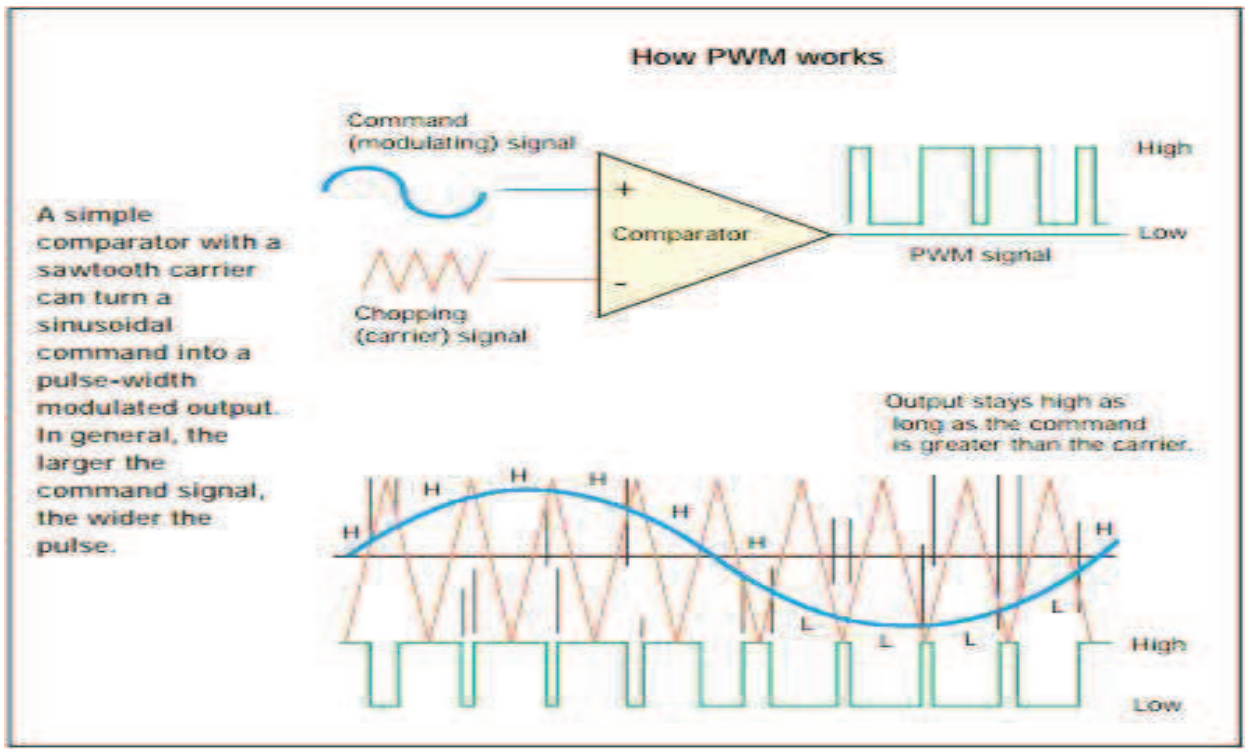


Figure-2.21-: shows how ***PWM*** works

2.4.3.2.4) ***Servo motor Control:***

Two basic controls available [13] are shown in figure-2.22-:

- Open Loop.
- Closed Loop.

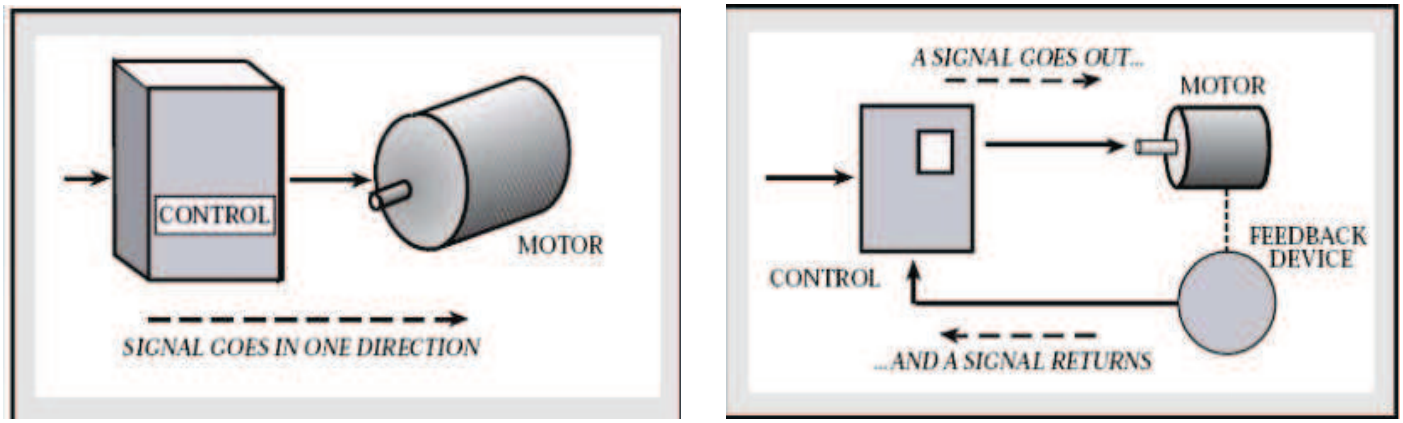


Figure-2.22-: From left to right; Open Loop Control and closed loop control.

The standard control works as follows [9]:

- The servo responds to a ***1 to 2ms*** pulse/signal ***20ms*** apart.
- An external controller tells the servo which direction and how fast to move. This is referred to as Pulse Width Modulation (***PWM***).
- A control wire communicates the desired angular movement. The angle is determined by the duration of the pulse applied to the control wire.
- The servo expects to see a pulse every *20 milliseconds* (*0.02 seconds*). The length of the pulse will determine how far the motor turns.
  - ✓ If the pulse is shorter than ***1.5 ms***, then the motor will turn ***clockwise***.
  - ✓ If the pulse is longer than ***1.5ms***, the shaft turns ***counter-clockwise***.
  - ✓ If the pulse equals to ***1.5 ms***, the shaft stays in the middle (neutral position)

A normal servo is used to control an angular motion of between 0 and 180 degrees as shown in figure-2.23a-:

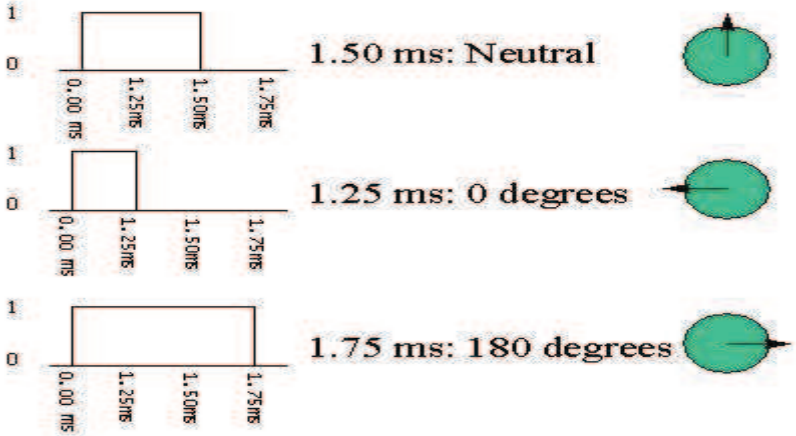


Figure-2.23a-: shows a PWM controlled servomotor

The internal control system block is given relating its components and including the PWM signal [11] is shown in figure-2.23b-:

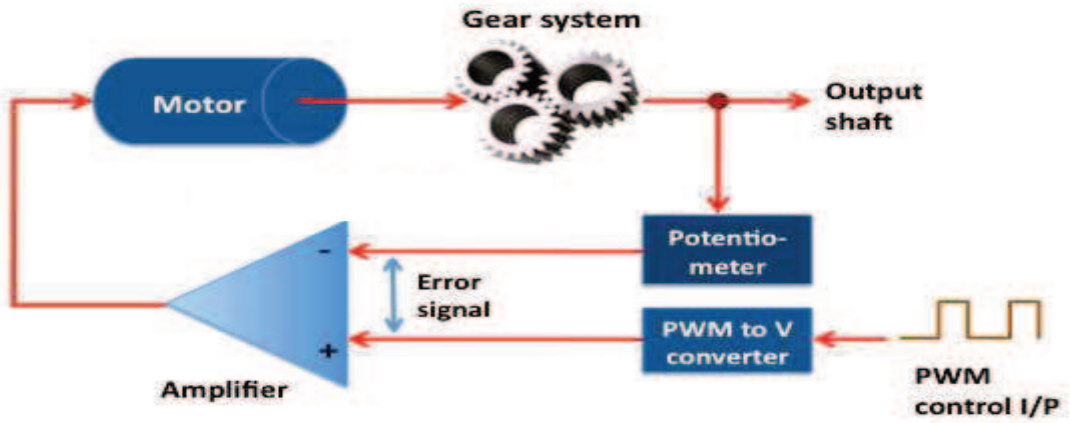


Figure-2.23b: servo motor control system block

In order to accomplish a servo function, an instantaneous positioning information of the output shaft is fed back to the control circuit using a transducer. A simplest way of doing this is by attaching a potentiometer to the output shaft or somewhere in the gear train. The control electronics compares the feedback signal (which contains the current position of the shaft) from the potentiometer to the control input signal (which contains information of the desired position of the shaft), and any difference between the actual and desired values (known as an error signal) is amplified and used to drive the DC motor in a direction necessary to reduce or eliminate the error. The error is zero when the output shaft gets to the desired position.

2.5) ***The plinth control:***

The plinth site represents the slave to be controlled by operator who sends control signals via communication channel to the remote side.

Thus in this part we need a joystick, another electronic based board, and of course the communication channel.

2.5.1) ***Joystick:***

2.5.1.1) ***Definition:***

Joysticks are manually actuated control devices for installation in control and front panels as well as in portable control equipment. They are used wherever motion sequences analogous to the actuation direction are controlled by hand. They are ideal for rising, lowering and triggering movements to the right and left.

Figure-2.24-: shows a simple

2.5.1.2) ***Applications:***

Joysticks are often used to control video games, and usually have one or more push-buttons whose state can also be read by the computer. Joysticks are also used for controlling machines such as:

- Cranes.
- Trucks.
- Underwater unmanned vehicles.
- Wheelchairs.
- Surveillance cameras.

2.5.1.3) ***Its components:***

The main components of an electronic joystick are [14-15]: two potentiometers that vary with respect to the input position, and a thumb bottom to set the desired position, and the pins.

Potentiometers are variable resistors and, in a way, they act as sensors providing us with a variable voltage depending on the rotation of the device around its shaft.

Generally joysticks have 5 pins that are shown in figure-2.25-:

- ***Vcc pin***: for powering labeled by "+"
- ***The ground(GND)***: pin labeled by "-"
- **X and Y**: position pins.
- **B pin**: this is the output from the pushbutton.

The two positions X and Y work as follows:

- Left or Right motion in x axes
- ***Up and Down*** movement in the ***Y*** axis.

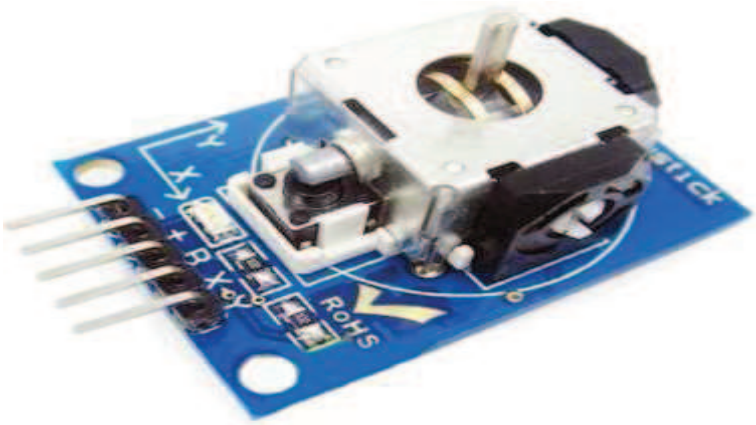


Figure-2.25-: shows the components of a joystick

The two potentiometers are shown by the black boxes in figure-2.25- below

2.5.2) ***Communication channel (Data transmission/Reception):***

The transmission is ensured by another electronic based board having the same characteristics as the one described in section (2.4.1), and the data will be received by the second EBB which is located in the remote site representing the main part of the plinth.

In this section we are going to describe the methods used for data transmission/reception, or simply the communication between two EBBs.

This part permits us to send control signals from the master site to the slave site, by using either wired or wireless communication. Although communication interfaces modules are often integrated into the controller and can therefore be seen as controller components. Still, microcontrollers generally contain several communication interfaces and sometimes even multiple instances of a particular interface, like two ***UART*** modules.

Communication is essential in electronics system. It can be in the form of wired or wireless, serial or parallel [16-17-18]. ***In parallel mode***, multiple bits are sent with each clock pulse. ***In serial mode***, one bit is sent with each clock pulse. While there is only one way to send parallel data, there are two subclasses of serial transmission; synchronous and asynchronous. See ***Figure-2.27a-.***

The main idea is to transfer information from one system to another system; communication in one direction is called a simplex communication system. Half duplex means that communication is taking place in both direction but only one direction communication is taking place at any one time, and full duplex means communication is in both directions at the same time as shown in figure-2.26b-:

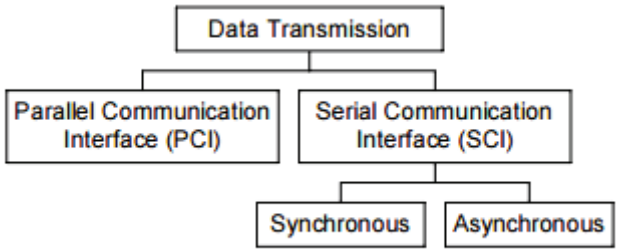


Figure-2.26a-: Data transmission structure

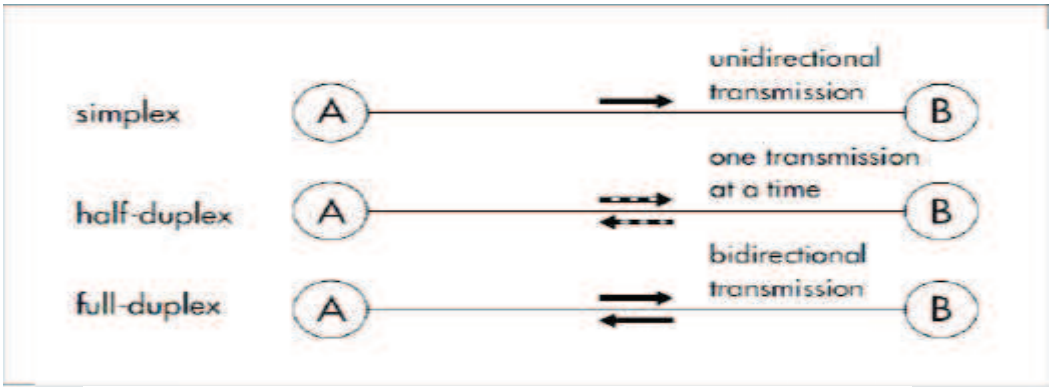


Figure-2.26b-: depicts the three main methods for data T/R

2.5.2.1) ***Serial Data Transmission:***

In serial transmission one bit follows another (see figure-2.27b-), so we need only one communication channel rather than ***n*** to transmit data between two communicating devices. The advantage of serial over parallel transmission is that with only one communication channel, serial transmission reduces the cost of transmission over parallel by roughly a factor of *n.*

A serial bus consists of just two wires (see figure-2.27a-); one for sending data and another for receiving. As such, serial devices should have two serial pins: the receiver, **RX**, and the transmitter, **TX**. [17]

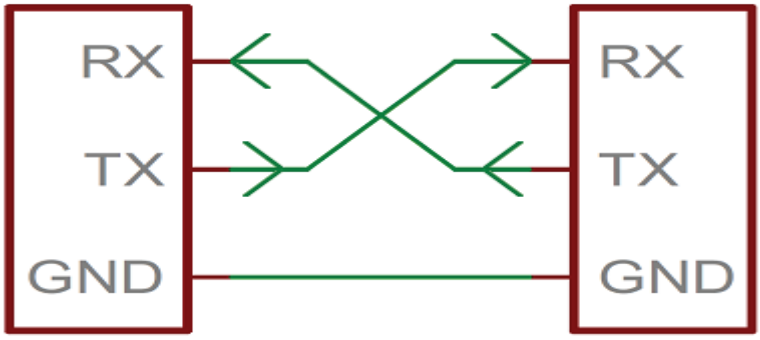


Figure-2.27a-: Serial transmission

Since communication within devices is parallel, conversion devices are required at the interface between the sender and the line (***parallel-to-serial***) and between the line and the receiver (***serial-to-parallel***). As shown in figure-2.27b-.

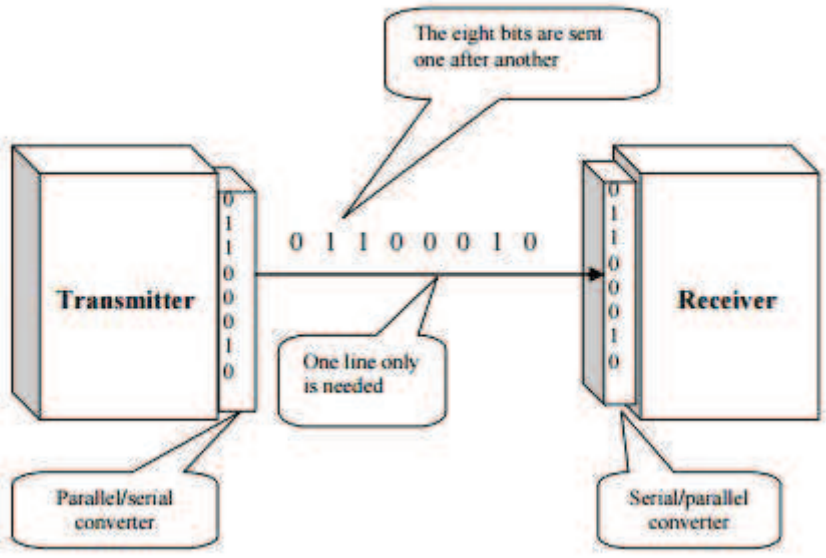


Figure-2.27b-: Serial Transmission

Serial transmission occurs in one of two ways; asynchronous or synchronous. [17-18]

2.5.2.1.1) ***Asynchronous Transmission:***

Asynchronous transmission is so named because the timing of a signal is unimportant. Instead, information is received and translated by agreed-upon patterns. Patterns are based on grouping the bit stream into bytes. Each group, usually eight bits, is sent along the link as a unit.

To alert the receiver to the arrival of a new group, an extra bit is added to the beginning of each byte. This bit, usually a 0, is called the ***start bit.***

To let the receiver know that the byte is finished, one or more additional bits are appended to the end of the byte. These bits, usually ***1s***, are called ***stop bits.***

By this method, each ***byte*** is increased in size to at least ***10 bits***, of which ***8*** are information and ***2*** or more are signals to the receiver. In addition, the transmission of each byte may then be followed by a ***gap*** of varying duration. This ***gap*** can be represented either by an ***idle channel*** or by a *stream* of *additional stop* bits (see figure-2.28-).

When the receiver detects a start bit, it sets a timer and begins counting bits as they come in. After ***n*** bits, the receiver looks for a stop bit. As soon as it detects the stop bit it ignores any received pulses until it detects the next start bit.

The following Figure-2.28- is a schematic illustration of asynchronous transmission. In this example, the start bits are ***0s***, the stop bits are ***1s***.

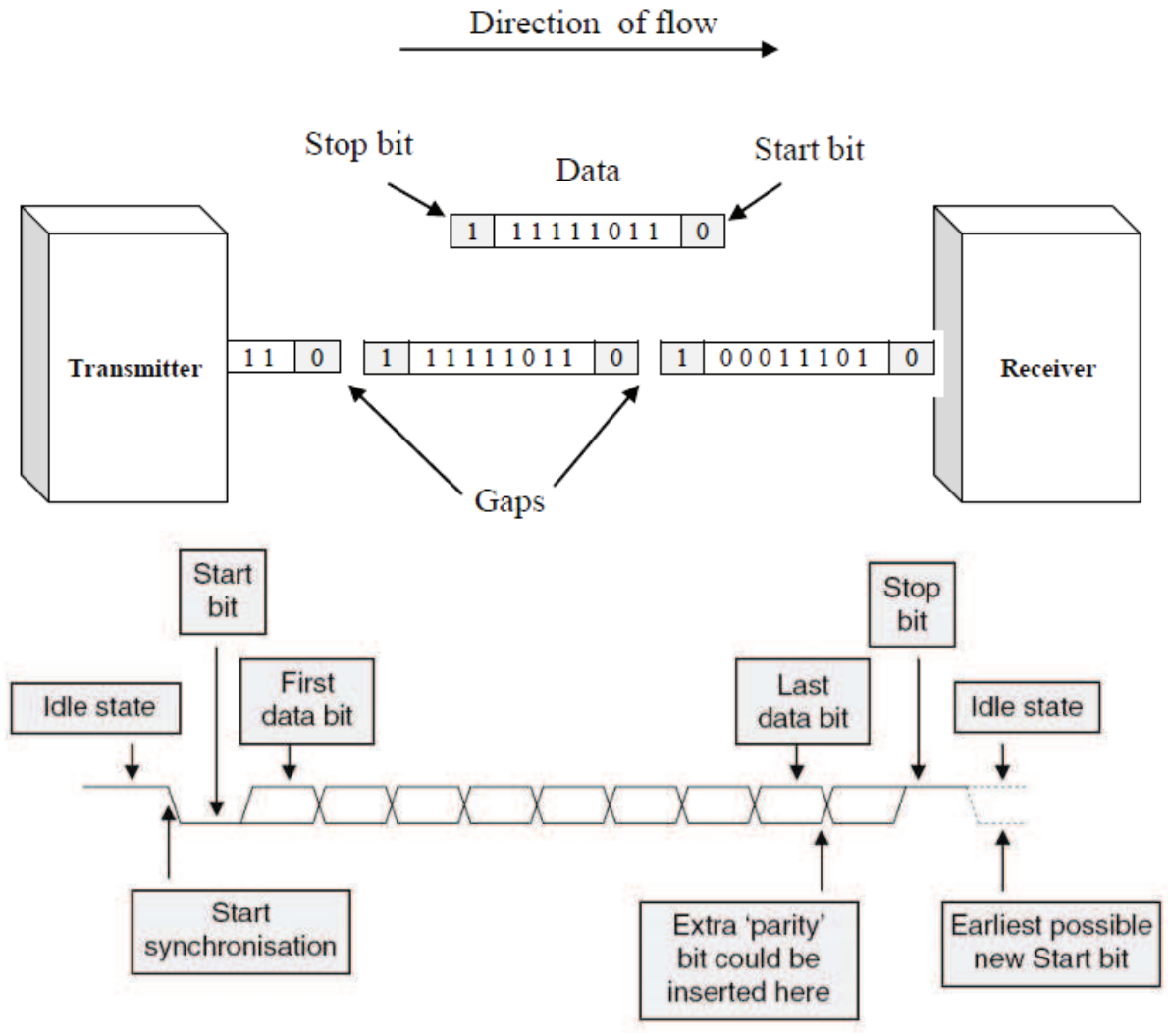


Figure-2.28-: Asynchronous Transmission

Asynchronous transmission is slower than other forms of transmission because of the addition of control information. But it is cheap and effective, two advantages that make it an attractive choice for situations like low-speed communication. For example, the connection of a terminal to a computer is a natural application for asynchronous transmission. A user types only one character at a time, types extremely slowly in data processing terms, and leaves unpredictable gaps of time between each character.

2.5.2.1.2) ***Synchronous Transmission:***

In synchronous transmission**,** the bit stream is combined into longer "frames," which may contain multiple bytes. Each byte, however, is introduced onto the transmission link without a gap between it and the next one. It is left to the receiver to separate the bit stream into bytes for decoding purposes (see figure-2.29b***-).*** In other words, data are transmitted as an unbroken string of ***1s*** and ***0s***, and the receiver separates that string into the bytes, or characters, it needs to reconstruct the information.

In synchronous transmission, we send bits one after another without ***start/stop bits*** or ***gaps***. It is the responsibility of the receiver to group the bits.

The following ***Figure-2.29a***- gives a schematic illustration of synchronous transmission. We have drawn divisions between bytes. In reality, those divisions do not exist; the sender puts its data onto the line as one long string. The receiver counts the bits as they arrive and groups them in ***eight-bit*** units.

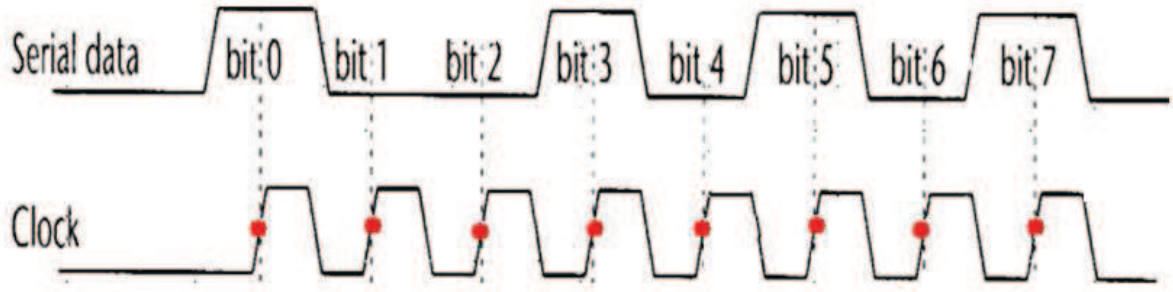


Figure-2.29a-: Asynchronous Transmission as an unbroken string

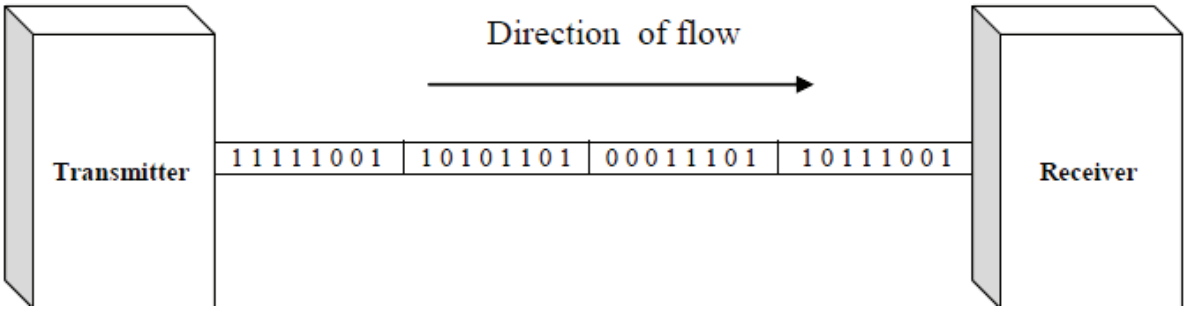


Figure-2.29b-: Asynchronous Transmission

The advantage of synchronous transmission is speed. With no extra bits or gaps, synchronous transmission is faster than asynchronous transmission. For this reason, it is more useful for high-speed applications like the transmission of data from one microcontroller to another

2.5.2.2) ***Parallel Data Transmission:***

Binary data, consisting of ***1s*** and ***0s***, may be organized into groups of *n* bits each. By grouping, we can send data *n* bits at a time instead of one. This is called **parallel transmission.**

We use ***n*** wires to send ***n*** bits at one time. That way each bit has its own wire, and all ***n*** bits of one group can be transmitted with each clock pulse from one device to another. The *Figure-2.30*- bellow shows how parallel transmission works for *n* = 8. Typically, the eight wires are bundled in a cable with a connector at each end.

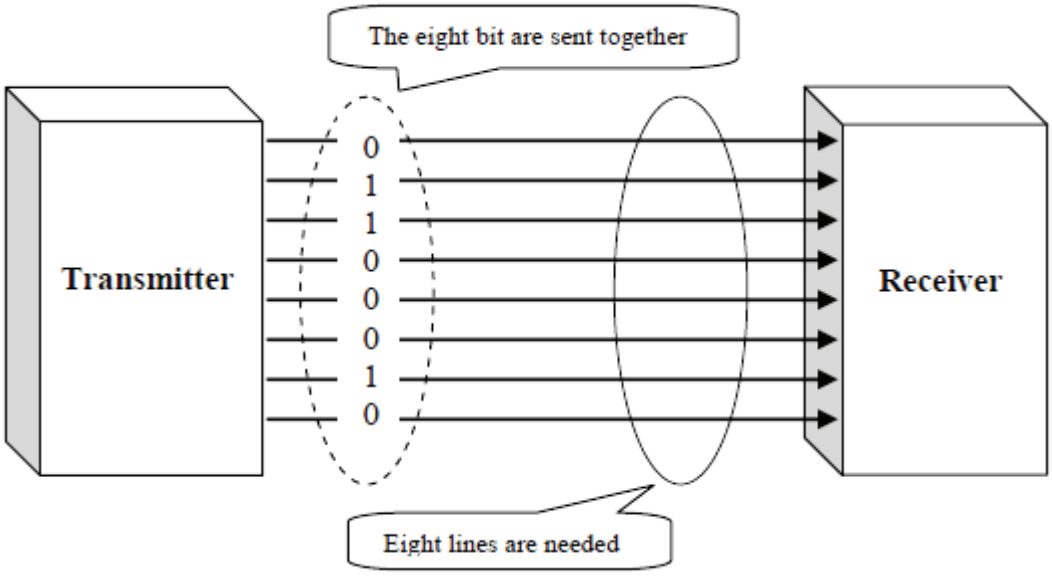


Figure-2.30-: Parallel Data Transmission

The advantage of parallel transmission is speed; it can increase the transfer speed by a factor of ***n*** over serial transmission. A significant disadvantage of parallel transmission is cost. Parallel transmission requires ***n*** communication lines (wires in the example) just to transmit the data stream. Because this is expensive, parallel transmission is usually limited to short distances.

2.5.2.3) ***Wired and Wireless communication protocols:***

The communication between two systems uses the communication protocol to stay connected, can be either wired or wireless communication.

Two primary methods of providing a network connection exist; wired (***via cable***) and wireless. Wired networking has traditionally been deployed for stationary computers and machines which do not require mobility. Wireless networking allows the user to roam wire free where the wireless

network exists while never dropping the connection to the network. Traditionally, wired connections have been the primary means of access to the network, with wireless connectivity offering a secondary means of connection for mobile devices.

2.5.2.3.1) ***Wired communication:***

We refer to wired communication as a two systems that can be either computers or microcontrollers related by a wire. C**ommunication protocol** is a system of rules for data exchange within or between for example computers.

Wired networking connections provide the foundation of the Local Area Network. Incoming connections. The primary benefit to a wired connection is that the wire provides a standard level of service (performance, security, reliability) which can be relied upon in all situations.

- ***Advantages:***
  - Standard level of service guaranteed to each user/device.
  - High bandwidth capable (1GB/10GB).
  - Low cost of support.
  - Higher level of security.
- ***Disadvantages:***
  - High cost of initial installation.
  - Difficult to install in some locations.
  - Number of connections limited by number of cables installed.

The most used wired communication protocols: RS-232, USB, SPI, I2C...etc

2.5.2.3.1.1) ***RS-232:***

RS-232 is a standard for serial communication transmission of data. It formally defines the signals connecting between a *DTE* (*data terminal equipment*) such as a computer terminal, and a *DCE* (*data circuit-terminating equipment*, originally defined as *data communication equipment*), such as a modem. The RS-232 standard is commonly used in computer serial ports.

RS-232 cables are commonly available with either 9 or 25-pin wiring [19]. The *25-pin* cable connects every pin (see figure-2.31a-), the *9-pin* cables do not include many of the uncommonly used connections (see *figure-2.31b-):*

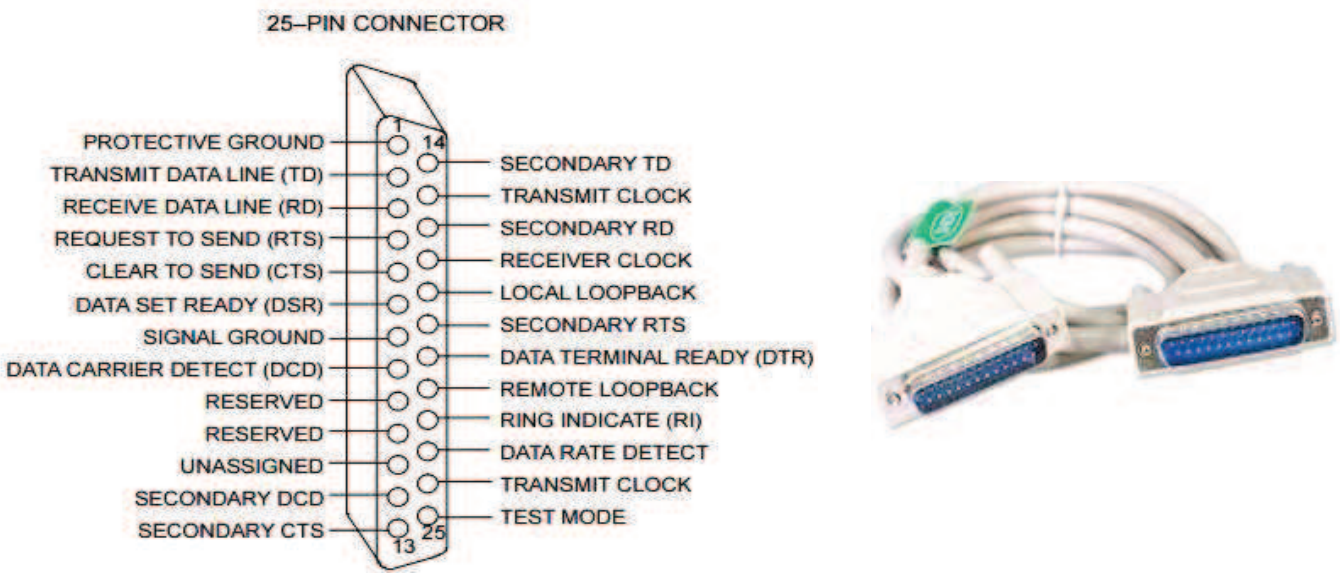


Figure-2.31a-: RS-232 with 25-pin connector

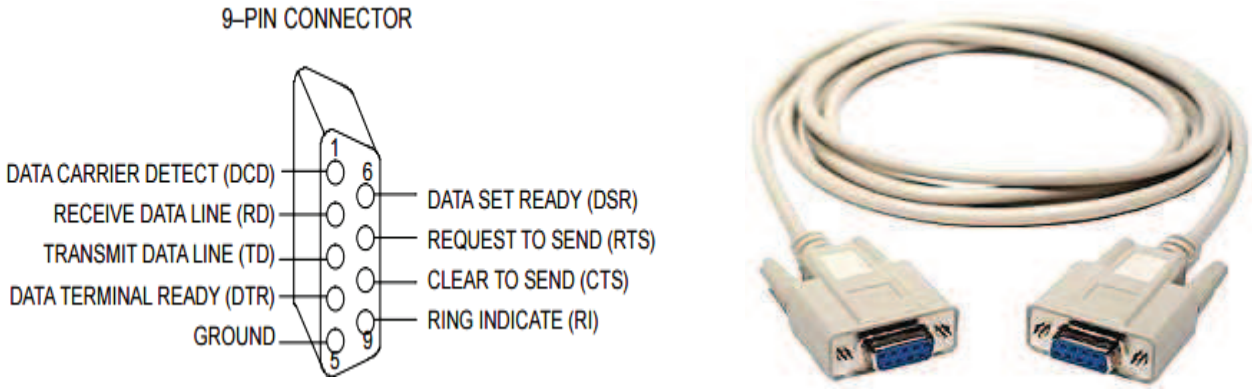


Figure-2.31b-: RS-232 with 9-pin connector

2.5.2.3.1.2) ***Serial Peripheral Interface (SPI):***

- ***Definition:***

The *Serial Peripheral Interface (SPI*) [20] bus is a synchronous serial communication interface specification used for short distance communication, primarily in embedded systems . The interface was developed by Motorola. Typical applications include sensors, Secure Digital cards.

SPI devices communicate in full duplex mode using a master-slave architecture with a single maste. The master device originates the frame for reading and writing. Multiple slave devices are supported through selection with individual slave select (SS) lines as shown in figure-2.32b-.

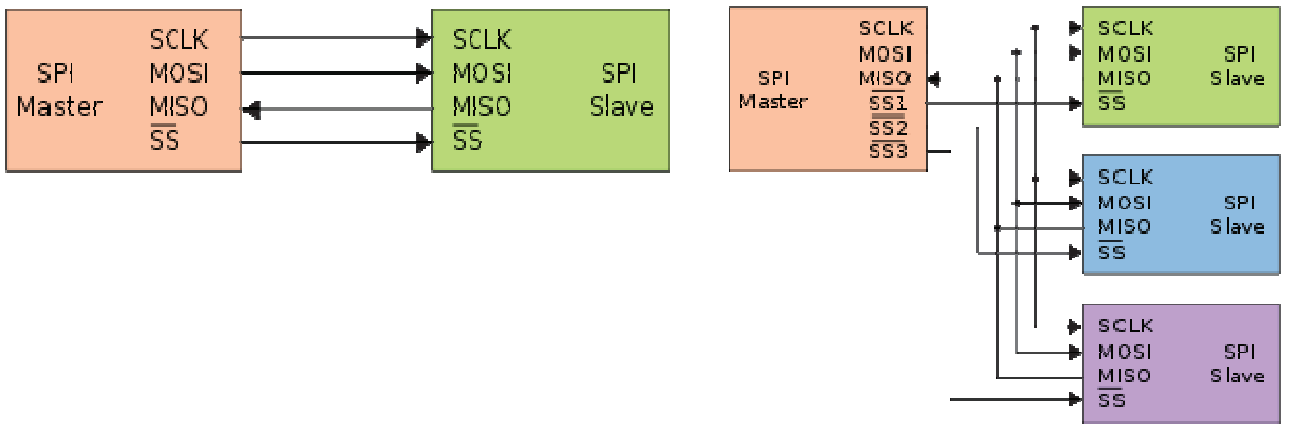


Figure-2.32-: From left to right; "single master and single slave"," single master multiple slave" SPI bus

The SPI bus specifies four logic signals:

- ***SCLK*** : Serial Clock (output from master).
- ***MOSI*** : Master Output, Slave Input (output from master).
- ***MISO*** : Master Input, Slave Output (output from slave).
- ***SS*** : Slave Select (active low, output from master).

- ***Operation:***

An SPI transmission is a simultaneous communication between a master and a slave:

- The master generates the clock and selects the desired slave to communicate by using the SS pin.
- The slave answers to the master requests.

At each occurring clock there is one bit exchanged between the master and the slave, after 8 clocks 8 bits are exchanged between them. The clock speed is fixed with respect to the peripherals characteristics.

2.5.2.3.1.3) ***I^2C:***

- ***Definition:***

I^2C (Inter-Integrated Circuit) [21] is a short distance serial interface that requires only 2 bus lines for data transfer. It was invented by Philips in 1980's, originally to provide easy on-board communications between a CPU and various peripheral chips in a TV set. Today, it is widely used in varieties of embedded systems to connect low speed peripherals (external EEPROMs, digital sensors, LCD drivers, etc) to the main controller.

- ***Operation:***

Data on the***I^2C*** bus can be transferred at a rate up to 100 Kbps (in standard mode), 400 Kbps (in fast mode), or up to 3.4 Mbps (in high-speed mode)

I2C bus has two lines: a **serial data** line (SDA) and a **serial clock** line (SCL). Any data sent from one device to another goes through the SDA line, whereas the SCL line provides the necessary synchronization clock for the data transfer. The devices on an I2C bus are either **Masters** or **Slaves**. Only a Master can initiate a data transfer and Slaves respond to the Master. It is possible to have multiple Masters on a common bus, but only one could be active at a time. The SCL clock line is always driven by the master. Figure-2.33- below shows an I2C bus with a single master and three slaves. Slaves can never initiate a data transfer but they can transfer data over the I2C bus, and that is always controlled by the Master.

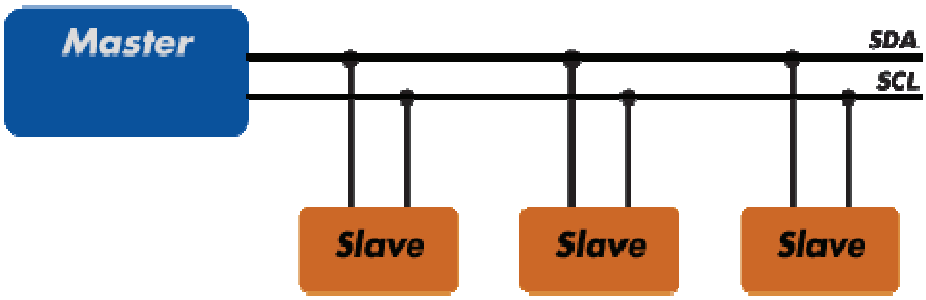


Figure-2.33-: Multiple devices on common I2C

2.5.2.3.1.4) ***USB (Universal Serial Bus):***

Universal Serial Bus (USB) [22-23] is an *industry standard* developed in the mid-1990s that defines the cables, connectors and ***communications protocols*** used in a ***bus*** for connection, communication, and power supply between ***computers*** and electronic devices.

Modern PCs support several trees of USB devices, usually:

- USB 3.0 about 10 time faster then 2.0 (4.68Gb)
- USB 2.0 (480 Mbit/sec each)
- USB 1.1 trees (12 Mbit/sec each)....and so on.

2.5.2.3.2) ***Wireless Communication:***

Wireless communications is a fast-growing technology that enables people to access networks and services without cables. Deployment can be envisaged in various scenarios: different devices belonging to a single user, such as a mobile telephone, a portable computer, a personal digital assistant (PDA) and others, which need to interact in order to share documents; a user who receives emails on the PDA; a shopping mall where customers display special offers on their PDA; car drivers loading maps and other tourist information while driving on the motorway. All of these scenarios have become a reality from a technological point of view and successful experiments are being carried out around the world.

The wireless approach shows many advantages but also has some disadvantages with respect to cabled networks [3]. Mobility is clearly one of the major advantages of wireless with respect to cabled devices, which require plugging. Another advantage lies in the way new wireless users can

dynamically join or leave the network. Wireless networks are simple to deploy, and in some cases, they cost less than wired LANs. Nevertheless, the technological challenges involved in wireless networks are not trivial, leading to disadvantages with respect to cabled networks, such as lower reliability due to interference, higher power consumption, data security threats and lower data rates.

Currently the wireless scene is held by two standards, namely the *Bluetooth* and the *IEEE 802.11 (wifi)* protocols, which define the physical layer and the medium access control (MAC) for wireless communications over a short action range (from a few up to several hundred meters) and with low power consumption (from less than 1 mW up to hundreds of mW).

The wireless scene is also held by the modern wireless technology namely zigbee which is based on 802.15.4 protocol. Thus; there is three main wireless protocols:

- WIFI (IEEE 802.11)
- Bluetooth.
- Zigbee.

2.5.2.3.2.1) ***Bluetooth:***

- ***Definition:***

Bluetooth [24-25-26] is a standard for wireless communications based on a radio system designed for short-range (from 2.4 to 2.485 GHz), cheap communications devices suitable for substituting cables for printers, faxes, joysticks, mice, keyboards, etc. The devices could also be used for communications between portable computers, act as bridges between other networks. This range of applications is known as WPAN (*Wireless Personal Area Network*).

Figure-2.34- shows the Bluetooth symbol:

Figure-2.34-: The Bluetooth symbol

- ***Operation:***

Any Bluetooth device can be a master or a slave, depending on the application scenario. Bluetooth employs Frequency Hopping Spread Spectrum (FHSS) to communicate. So in order for multiple Bluetooth devices to communicate, they must all synchronize to the same hopping sequence. The master sets the hopping sequence, and the slaves synchronize to the Master.

The following figure-2.35- shows a Bluetooth module having 4 pins to be connected with an EBB.

- TX: for transmission.
- RX: for reception.
- +Vcc: powering.
- GND: for the ground

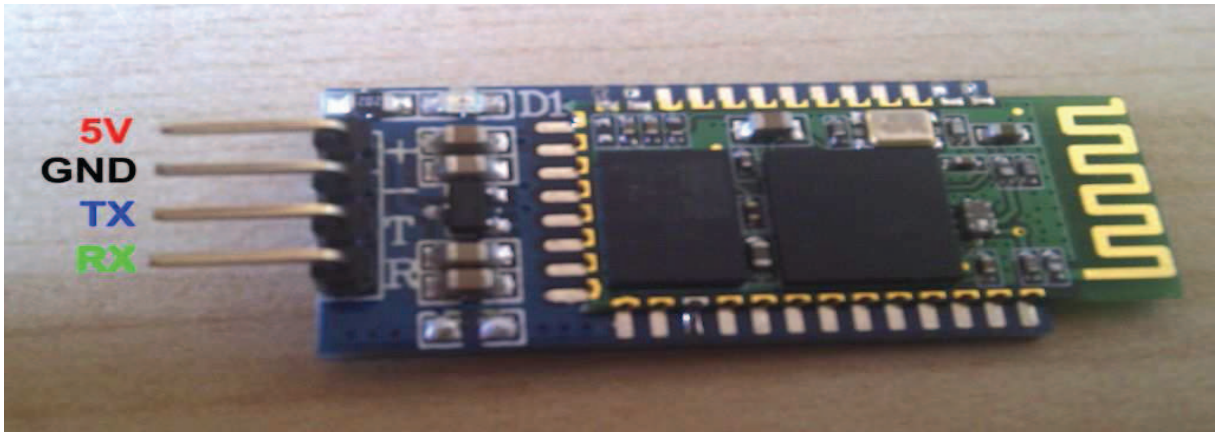


Figure-2.35-: An example of bluetooth

2.5.2.3.2.2) ***WIFI:***

- ***Definition:***

Wi-Fi [27-28](or **WiFi**) is a local area wireless technology that allows an electronic device to participate in computer networking using 2.4 GHz UHF (Ultra high Frequency) and 5 GHz SHF (Super High Frequency)

The Wi-Fi Alliance defines Wi-Fi as any "wireless local area network" (WLAN) product based on the Institute of Electrical and Electronics Engineers' (IEEE) 802.11 standards". However, the term "Wi-Fi" is used in general English as a synonym for "WLAN" since most modern WLANs are based on these standards.

- ***Uses:***

To connect to a Wi-Fi LAN, a computer has to be equipped with a ***wireless network interface controller***. The combination of computer and interface controller is called a *station*. All stations share a single radio frequency communication channel. Transmissions on this channel are received by all stations within range. The hardware does not signal the user that the transmission was delivered and is therefore called a best-effort delivery mechanism. A carrier wave is used to transmit the data in packets, referred to as "Ethernet frames". Each station is constantly tuned in on the radio frequency communication channel to pick up available transmissions.

A **wireless network interface controller** [29] (WNIC) is a network interface controller which connects to a radio-based computer network rather than a wire-based network. A WNIC is an essential component for wireless desktop computer. This card uses an antenna to communicate through microwaves.

2.5.2.3.2.3) ***Zigbee:***

- ***Definition:***

ZigBee [30] is a *specification* for a suite of high-level communication protocols used to create *personal area networks* built from small, low-power *digital radios*.

It is known as a radio frequency (*RF*) communications standard based on *IEEE 802.15.4,* 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 *ZigBee* standard can communicate with 2*50kbps* data rate, but 4*0 kbps* can meet the requirements of most control systems. They can transmit data over long distances by passing data through a ***mesh network*** of intermediate devices to reach more distant ones so it is sufficient for controlling most home automation devices. *Figure-2.36-,* depicts the general block diagram of a *Zigbee* based home automation system [31],

Figure-2.36-: Block diagram of a Zigbee based home automation

- ***Applications***

ZigBee is typically used in low data rate applications that require long battery life and secure networking.

This device network has the characteristics of electric power-saving, reliability, low cost, large capacity and security, and it can be widely used in various fields of automatic control. The target application domains are aimed at industry such as:

- *home automation*
- *telemetry and remote control*
- *vehicle automation,*
- *agriculture automation,*
- *medical care*
- *lighting control automation*
- *wireless data acquisition and monitoring sensor.....so on*

- ***Some related points to Zigbee:***
  - ***Protocol***: *Communication protocol* is a system of digital rules for data exchange within or between computers.
  - ***IEEE 802.15.4 Standard:***

IEEE 802.15.4[28] is a standard which specifies the *physical layer* and *media access control* for low-rate wireless *personal area networks* (*LR-WPANs*). It is maintained by the *IEEE 802.15* working group, which has defined it in 2003. It is the basis for the *ZigBee.*

  - ***Mesh Network:***

A mesh network (see figure-2.37-) is a *network topology* in which each *node* relays data for the network. All nodes cooperate in the distribution of data in the network. Every node in a mesh network is called a *mesh node.*

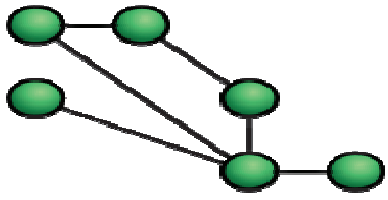


Figure-2.37-: Mesh Network

The green balls are the nodes which are connected by wires to show a network, and the whole represents the mesh network.

  - ***RF communication:***

Wireless systems (radios) transmit data and voice information using a specific radio frequency (RF) to other radios tuned to the same frequency. Common radio messages are transmitted over the RF band between 0.05 MHz and 900 MHz. Most public safety communications radios (portable, mobile,

base station and repeaters) transmit frequencies between 30 MHz and 900 MHz which are dedicated to public service use. Cell phones and systems, such as global positioning receivers, call boxes, electronic signs, and mobile command units, that transmit information from remote locations, transmit in the microwave band between 1 GHz and 20 GHz.

- ***Xbee Module:***

- ***Description:***

The Xbee [32] module (see figure-2.38-) is a wireless transmission device using IEEE-802.15.4 protocol based on Zigbee norm. This module is actually used in wireless network sensors, data transfer between two related devices...and so on.

Xbee is provided as a PCB (Printed Circuit Board), containing a chip that implements the 802.15.4 protocol and an antenna. This module exists in different versions such as [33-34]:

- ***Series 1***
- ***Series 2***
- ***Xbee pro***

Figure-2.38-: Xbee module

A. ***XBee Series 1 (also called XBee 802.15.4) :***

These are the easiest to work with, they don't need to be configured, the ***802.15.4*** style of XBee allows ***point-to-point*** networking and point-to-multipoint (one node to all nodes) networking. All nodes on the network use the same firmware version though settings on various nodes may vary.
Data can be transmitted up to 30m (indoor). This is of course depending on the used module.

B. ***Xbee series 2:***

Series 2 modules must be configured before they can be used. They can run in a transparent mode or work with API commands, but this all depends on the configuration. These modules are in no way compatible with the Series 1.
Series 2 modules can sent data up to 40m (indoor) this a little bit more performer than Series 1.
The differences between XBee Series 1 and 2 are more evident in the firmware features. Listed below are some of the more defining differences in the firmware:

- **802.15.4:** XBee Series 1 comes standard with 802.15.4 firmware for point-point or star topology. This mature firmware offers ADC (analog-to-digital conversion) inputs, and digital and analog I/O line passing. The 802.15.4 XBee is significantly faster than ZigBee.

- **ZigBee:** XBee Series 2 does not offer any 802.15.4-only firmware; it is always running the ZigBee mesh firmware. The infrastructure of a ZigBee network is more complex and requires more configurations to fully implement.

C. ***XBee-Pro:***

The Pros are a bit longer similar to the two series but differs in few things when comparing it to the others for example; it uses more power and cost more money. The greater power means longer range (1.6 Km instead of 30m). It's usually recommended for far remote control.

- ***Application domains:***

Xbee allows implementing a ***PAN*** (Personal Area Network) network, for large application domains; in particular it can implement the following functions:

- It can be used as a sensor node, where it can sensor a digital or analog value (such as: temperature, voltage, current .....so on) then send it via wireless link.
- It can be configured as a node actuator. In this case for implementing a remote control.
- It can also simply used as a communicating module, in this case transferring received data via wireless link.

- ***Standard usage mode:***

The role of the ***Xbee module*** is to be as transparent as possible. It should be as if a wire was connecting the input to the output and connectivity is never lost. While this sounds simple, it's actually a very hard task to accomplish.

Figure-2.39- depicts the Xbee module pin-out [35]:

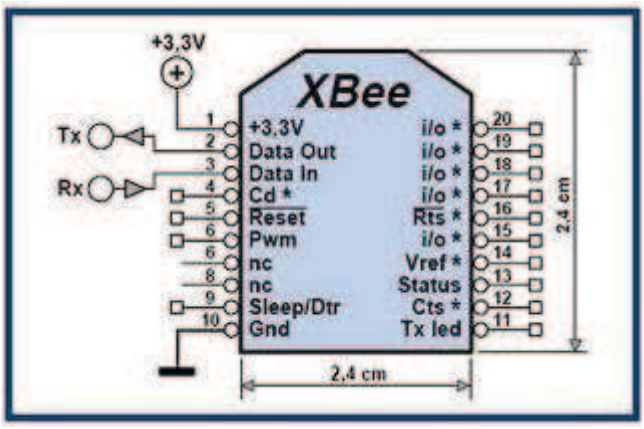


Figure-2.39-: Xbee module pinout

For transmitting data from one EBB side to another by using two Xbee modules, let us illustrate this by figure-2.40- :

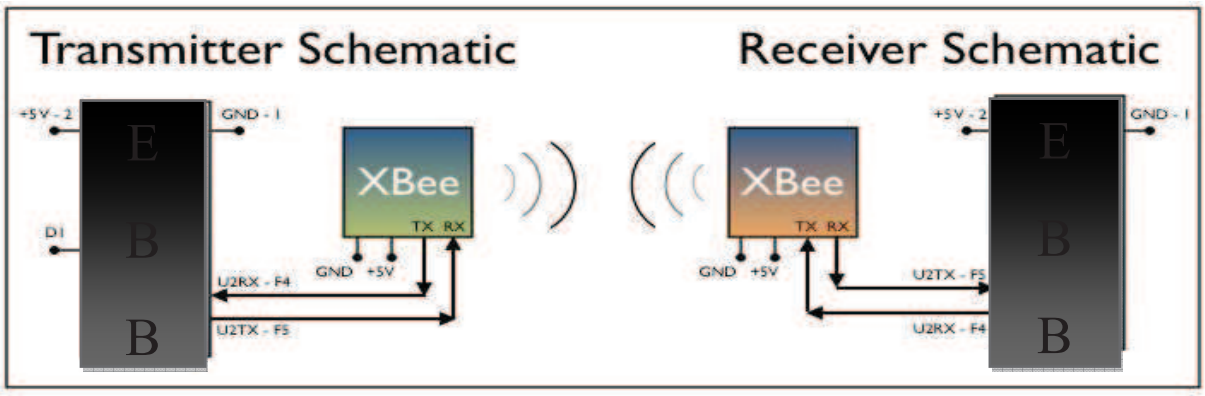


Figure-2.40-: Standard usage of transmitting data using two Xbee

Referring to ***figure-2.38-,*** each Xbee module is related to the EBB for interchanging data via (***TxD*** and ***RxD***) ports, and then the data is sent wirelessly between the two Xbee modules.

2.6) ***Conclusion:***

In this chapter we have described our system globally, especially the main part which represents the plinth by defining the main components in general manner and the way that it can be controlled as mentioned in the introduction.

By focusing on a standard motorized plinth, we have found that it contains:

- Electronic based board known as main controller board which plays an important role of data processing. The main part of an EBB is the microcontroller which is an incorporated computer in one single chip.
- Two motors that can be either stepper or servomotors that move the plinth to a desired position.
- Communication board to be connected with external devices.

In the third point we have defined the plinth control by including the needed devices as:

- Joystick which is composed of two potentiometers that vary with respect to the desired position in the two axes (X, Y)
- Communication channel (protocol) [36] contains the characteristics for data transmission/reception. In this section we have also seen two ways of sending data (Serial and Parallel), and two types of communication (***Wired and Wireless***.) Thus;
  - For wired communication we have described the most known buses such as : RS-232, SPI, USB, I^2C [37] ...so on.
  - For wireless we have seen WIFI, Bluetooth, Zigbee protocols.

These two components (joystick and communication protocol) are added to the plinth to form the teleoperation control system.

The global system scheme will be given in chapter III.

REFERENCES

References to chapter II:

[1] http://twiki.cis.rit.edu/twiki/bin/view/MVRL/BioGigapan?rev=42 (for EBB)

[2] Dogan Ibrahim "advanced PIC microcontroller projects in C" from USB to Zigbee.

[3] Wiiliam Stalling"Computer Organization and Architecture" Designing for performance.

[4] https://learn.sparkfun.com/tutorials/serial-communication

[5] http://lslwww.epfl.ch/pages/teaching/cours_lsl/ca_es/UART.pdf

[6] http://www.solarbotics.net/library/pdflib/pdf/motorbas.pdf

[7] http://www.engineersgarage.com/microcontroller/8051projects/stepper-motor-interfacing-with-8051-microcontroller-circuit

[8]http://hibp.ecse.rpi.edu/~connor/education/IEE/lectures/Lecture_8_Stepper_motors.pdf

[9]http://mechatronics.mech.northwestern.edu/design_ref/actuators/servo_motor_intro.html

[10] http://lizarum.com/assignments/physical_computing/2008/servo.html

[11] http://engineeringagenda.com/agenda/2014/05/servo-motors-control-arduino/

[12] https://www.tumblr.com/search/pulse%20width%20modulation

[13] http://www.baldor.com/Shared/manuals/1205-394.pdf

[14] http://icmasteronline.com/ps2-thumb-joystick-module---arduino--252808944

[15] http://www.baboon.co.in/eshop/sensors/65-dual-axis-xy-joystick-biaxial-button-joystick-ps2-game-joystick-sensor-joystick-module-ardunio.html

[16] http://www.siongboon.com/projects/2006-03-06_serial_communication/

[17] https://learn.sparkfun.com/tutorials/serial-communication

[18] http://students.iitk.ac.in/eclub/assets/lectures/embedded/Embedded-Old-4.pdf

[19] http://www.intea.hr/downloads/introduction_to_serial_communication.pdf

[20] http://en.wikipedia.org/wiki/Serial_Peripheral_Interface_Bus

[21] http://embedded-lab.com/blog/?p=2583 (I2C)

[22] http://en.wikipedia.org/wiki/USB

[23] http://en.wikipedia.org/wiki/USB#USB_3.0

[24] http://en.wikipedia.org/wiki/Bluetooth

[25]http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.1.1709&rep=rep1&type=pdf

[26]http://webuser.hs-furtwangen.de/~heindl/ebte-08ss-bluetooth-Ingo-Puy-Crespo.pdf

[27] http://en.wikipedia.org/wiki/Wi-Fi

[28] Hung-Yu Wei, Poland Sudhir Dixit Hewlett "Basic Communication Protocols and Application Areas" 2013.

[29] http://en.wikipedia.org/wiki/Wireless_network_interface_controller

[30] http://en.wikipedia.org/wiki/ZigBee

[31] http://www.caesjournals.org/splуploads/IJCAES-CSE-2013-NCIC20.pdf

[32] http://en.wikipedia.org/wiki/XBee

[33]http://urrg.eng.usm.my/index.php?option=com_content&view=article&id=281:xbee-&catid=31:articles&Itemid=70

[34] http://www.eecs.umich.edu/eecs/courses/eecs373/Lec/F13Student/XBee.pdf

[35] http://hades.mech.northwestern.edu/index.php/PIC32MX:_XBee_Wireless_Round-trip_Latency

[36] http://en.wikipedia.org/wiki/Communications_protocol

[37] http://en.wikipedia.org/wiki/I%C2%B2C

[39] http://multimechatronics.com/images/uploads/mech_n/Step_Motors.pdf

Chapter III: The Conception

3.1) ***Introduction:***

After describing teleoperation and talked about the whole system generally and globally in chapter I and Chapter II, we give our system block and choose the adequate components that satisfy our needs.

The two first chapters are meant to be the theoretical part of this paper, thus; in this chapter we will focus on the conception of our system by interfacing and studying deeply the different chosen components.

3.2) ***The System block scheme and functionality:***

In this section we built our system in blocks to give it a shape to be meaningful in order to help the reader to understand the utility of the two first chapters.

Figure-3.1- depicts the system block scheme:

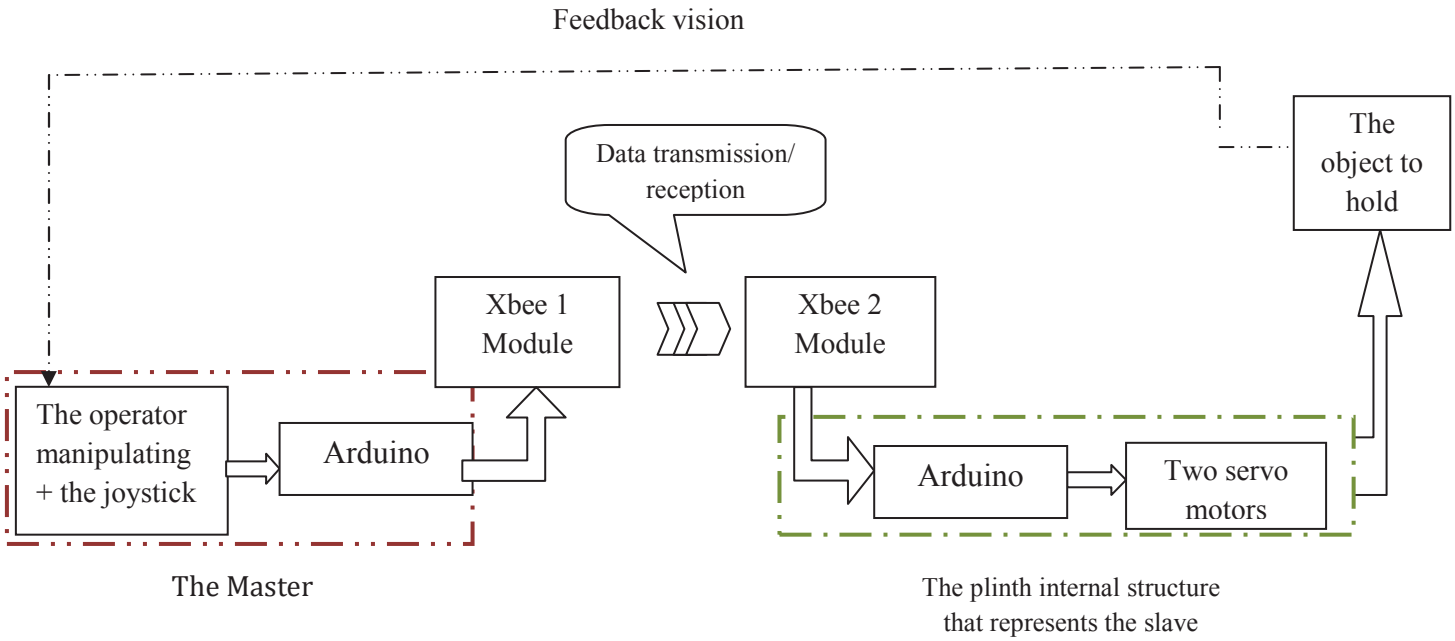


Figure-3.1a-*:* System block scheme

Referring to figure-3.1a-, from left to right:

The operator exerts the desired position by manipulating the joystick in four position i.e two axis (X for left/right, Y for up/down), the joystick is connected to the main controller Arduino, the Arduino is programmed in such manner to move the plinth in the desired position. The subsystem consisting of the joystick and the Arduino is the master site.

The Arduino is related to the Xbee module which is also configured in such a way to transmit the control signal (desired position) to the slave site. Then the transferred data is received by the second Xbee and loaded to the second Arduino to make the plinth (servomotors) moving. As we can see, the plinth represents the slave site that can carry objects such as camera, manipulating arm ...etc

Finally the environment is the space on where the task is executed, for example if we place a camera on the plinth, then the output signal is the pictures captured by the camera which would represent the environment.

Basing on chapter I, our system is a simple unilateral position teleoperation control system, the unilateral stands for the fact that there is no feedback position to master site. If both the master and slaves sites are located in the same place, the visual feedback is ensured by the operator. However, if not the camera held by the plinth should be an "IP camera". The advantage of an "IP camera" is that we can join it to the network relating both master and slave so that the operator can see exactly the position of the slave. The position flow is given in figure-3.1b-:

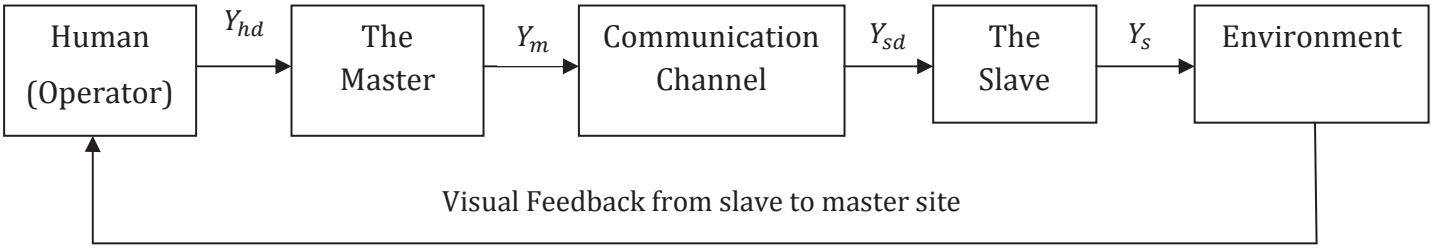


Figure-3.1b: Unilateral position-visual feedback control system

Where:

Y_{hd}: The operator desired position. Y_{sd}: The slave desired position.

Y_m: The master actual position. Y_s: The slave actual position

Referring to figure-3.1b, the operator desired position (Y_{hd}) becomes the master actual position (Y_m), then this latter becomes the slave desired position (Y_{sd}) which meant to be the actual position to the slave (Y_s).

3.3) ***The System interfacing and description:***

3.3.1) ***The joystick:***

The joystick that we use has exactly the same characteristics as the one described in section (2.5.1). Its main role stands for converting the mechanical movement to electrical signals.

This module combines two potentiometers and a pushbutton switch into a solid mechanical package which is perfect for controlling motors, servos, etc. The potentiometers track the position of the joystick in two dimensions and the button responds to downward pressure. Springs pull the joystick back to the center position. The one that we are going to use is shown in figure-3.2-

Figure-3.2-: The joystick

- ***Features:***

  - 2 x 5K Potentiometers for reading the X and Y position
  - 1 x Normally Open Momentary Switch (Closes with downward pressure on the joystick)
  - Simple 5 pin interface with clearly marked pins on standard 0.1" centers.

- ***Connection:***

The joystick can be connected to the Arduino (see section-3.2-) using the following hardware wiring shown in figure-3.3a, and the interfacing is shown in figure-3.3b.

The joystick is powered and grounded by the Arduino, the two axe-positions are connected to two of the six analog pins of the Arduino as shown in figure-3.3a-.

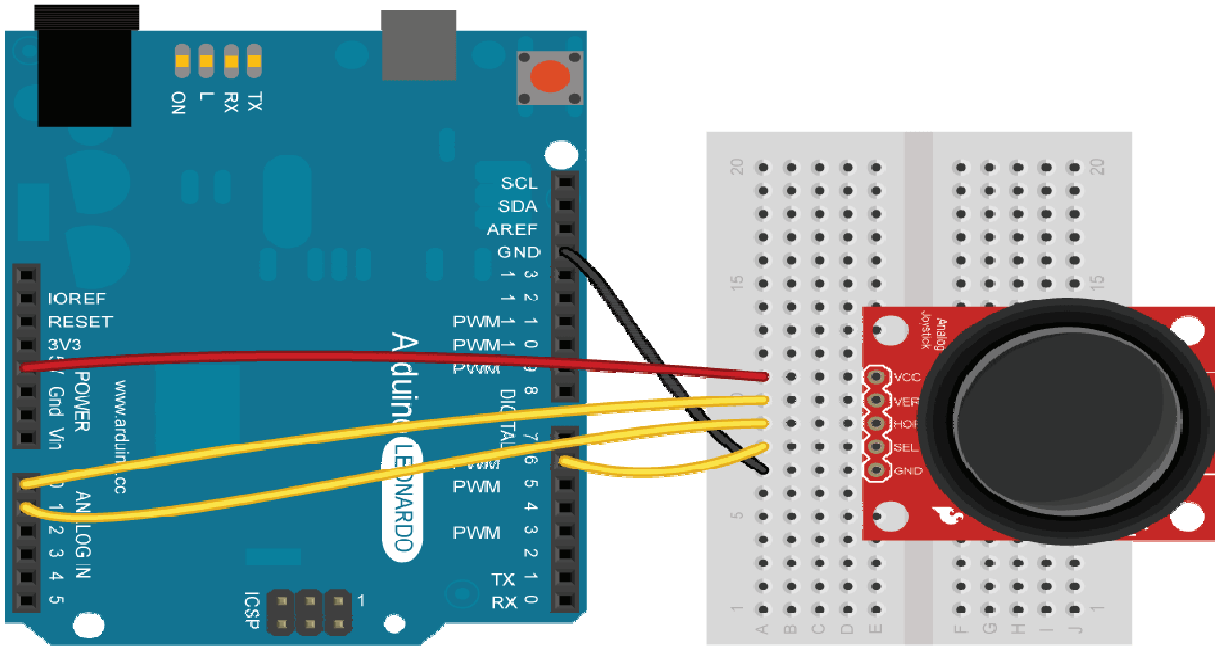


Figure-3.3a*:* Joystick to Arduino hardware connection

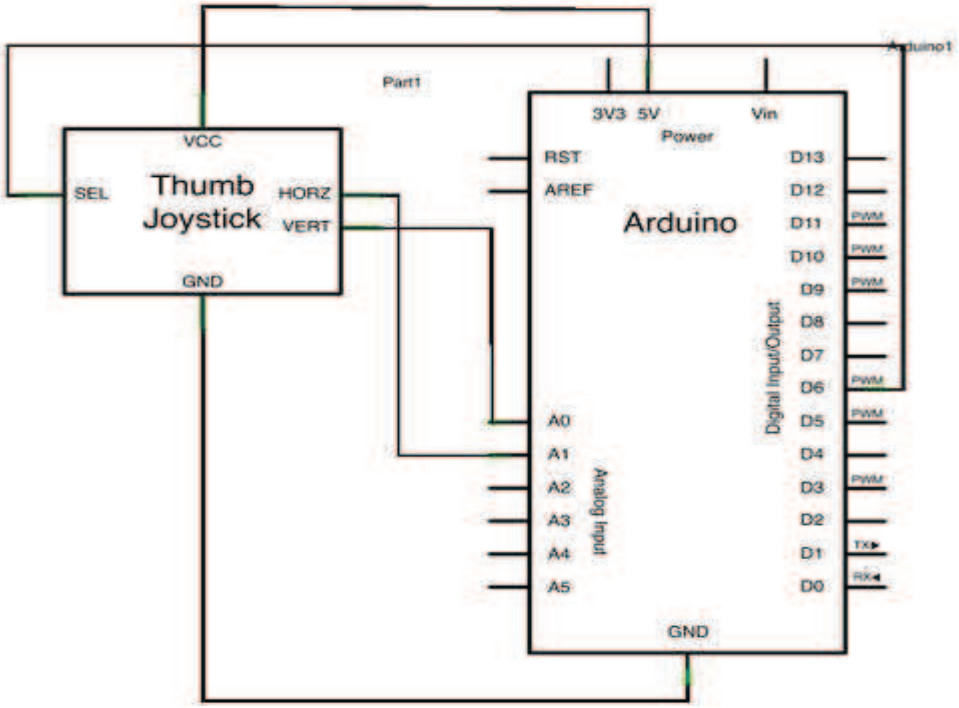


Figure-3.3b-: Joystick to Arduino interfacing connection

3.3.2) ***The Arduino:***

3.3.2.1) ***Definition:***

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. [1-2].

Arduino can sense the environment by receiving input from a variety of sensors and can affect its surroundings by controlling lights, motors, and other actuators. Arduino is a single-board microcontroller designed to make the process of using electronics in multidisciplinary projects more accessible. The hardware consists of a simple open source hardware board designed around an 8-bit Atmel AVR microcontroller, though a new model has been designed around a 32-bit Atmel ARM. The software consists of a standard programming language compiler and a boot loader that executes on the microcontroller.

There are three reasons that motivated us to use an Arduino rather that a microcontroller:

- It's relatively cheap and small.
- It's well documented.
- And easy to program.

3.3.2.2) ***Hardware Design:***

An Arduino board consists of an Atmel 8-bit AVR microcontroller with complementary components to facilitate programming and incorporation into other circuits.

An important aspect of the Arduino is the standard way that connectors are exposed, allowing the CPU board to be connected to a variety of interchangeable add-on modules known as shields.

Some shields communicate with the Arduino board directly over various pins, but many shields are individually addressable via an I^2C serial bus, allowing many shields to be stacked and used in parallel. Official Arduinos have used the mega AVR series of chips, specifically the ATmega8, ATmega168, ATmega328, ATmega1280, and ATmega2560. A handful of other processors have been used by Arduino compatibles. Most boards include a 5 volt linear regulator and a 16 MHz crystal oscillator. An Arduino's microcontroller is also pre-programmed with a boot loader that simplifies uploading of programs to the on-chip flash memory, compared with other devices that typically need an external programmer.

The Arduino board exposes most of the microcontroller's I/O pins for use by other circuits. The Diecimila, Duemilanove, and current Uno provide 14 digital I/O pins, six of which can produce pulse-width modulated signals, and six analog inputs. These pins are on the top of the board, via female 0.1 inch headers. Several plug-in application shields are also commercially available.The Arduino Nano, and Arduino-compatible Bare Bones Board and Boarduino boards may provide male header pins on the underside of the board to be plugged into solderless breadboards. [2]

3.3.2.3) ***Software:***

3.3.2.3.1) ***Background:***

The Arduino integrated development environment (IDE) is a cross-platform application written in Java, and is derived from the IDE for the Processing programming language and the Wiring projects [3]. It is designed to introduce programming to artists and other newcomers unfamiliar with software development. It includes a code editor with features such as syntax highlighting, brace matching, and automatic indentation, and is also capable of compiling and uploading programs to the board with a single click. There is typically no need to edit "makefiles" or run programs on a command-line interface.

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

- ***setup():*** a function run once at the start of a program that can initialize settings
- ***loop():*** a function called repeatedly until the board powers off

3.3.2.3.2) ***Installing the Arduino IDE:***

The Arduino IDE runs on all the latest versions of Microsoft Windows, such as Windows XP, Windows Vista, and Windows 7 [1]. Installing the software is easy, because it comes as a self-contained ZIP archive, so we don't even need an installer. Downloading [4] the archive, and extract it to a location of our choice.

Before starting the IDE, the Arduino's USB port must be installed. This process depends on the Arduino board that being used and on flavor Windows.

To start the driver installation process the Arduino must be plugged into a USB port.

3.3.2.4) ***The Arduino UNO board:***

In this project, we are making use of Arduino-uno since it's known as the most “standard” Arduino board currently on the market, and is probably the best choice for beginners just getting started with the platform. The board is compatible with more shields (add-on boards) and has a number of facilities for communication than other models [3]. Additionally, the ATmega328 microcontroller can be removed from its socket and replaced in case for breakdown. The Arduino Uno has a number of facilities for communicating with a computer, than another Arduino, or other microcontrollers

The Arduino Uno is a microcontroller board, based on the ATmega328 (datasheet). "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, and the reference model for the Arduino platform.

Figure-3.4- shows the external structure of the Arduino Uno board.

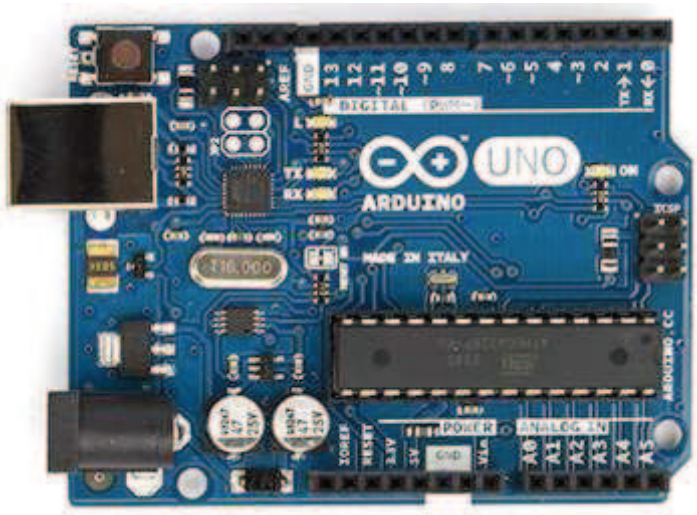


Figure-3.4-: The Arduino UNO

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.

3.3.2.4.1) ***Arduino UNO features and characteristics:***

The Arduino UNO based board includes the following characteristics [2]: (see figure-3.5-)

- 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
- Length
  - 68.6 mm
- Width
  - 53.4 mm
- Weight
  - 25 g

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 or battery.

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

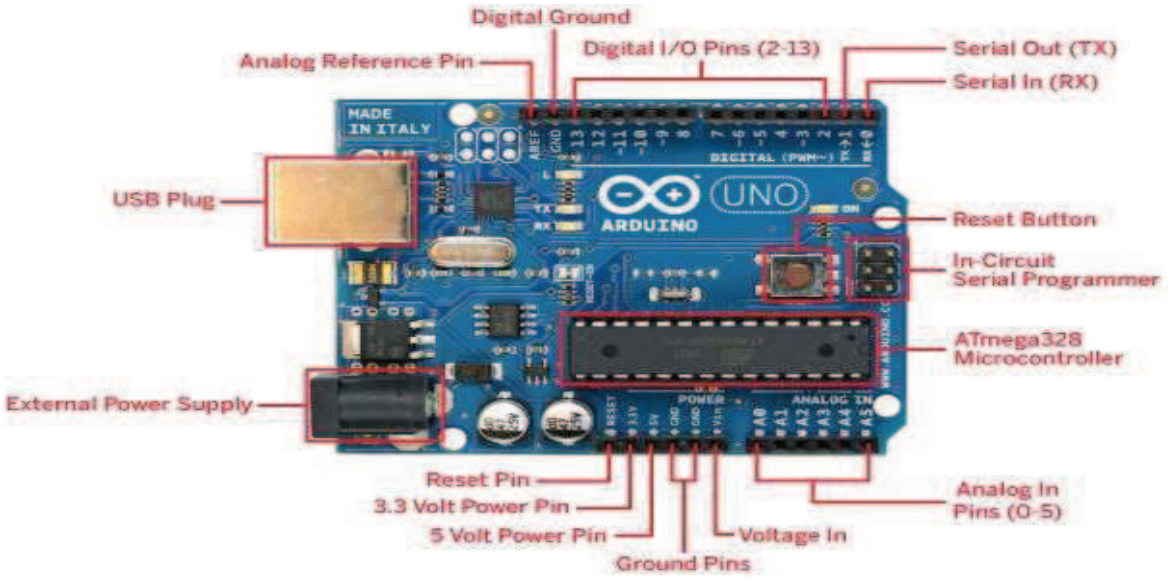


Figure-3.5-: The features and pins name of Arduino Uno

3.3.2.4.1.1) ***POWER PINS***

The power pins are as follows (see figure-3.6- for power pins location):

- ***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). One 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 the board.
- ***3.3V***: 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.

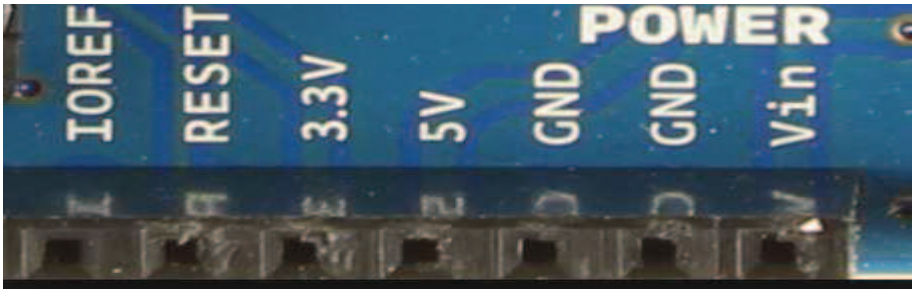


Figure-3.6-: *The Arduino UNO power pins*

3.3.2.4.1.2) ***Memory:***

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

3.3.2.4.1.3) ***Input/output:***

Each of the 14 digital pins on the Uno can be used as an input or output, using pinMode(), digitalWrite(), and digitalRead()functions [2-5]. They operate at 5 volts. Each pin can provide or receive a maximum of 40 mA and has an internal pull-up resistor (disconnected by default) of 20-50 kOhms. In addition, some pins have specialized functions:

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

- ***External Interrupts: 2 and 3***: These pins can be configured to trigger an interrupt on a low value, a rising or falling edge, or a change in value. See the attachInterrupt() function for details.
- ***PWM: 3, 5, 6, 9, 10, and 11***: Provide 8-bit PWM output with the analogWrite() [2-5] function.
- ***SPI: 10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK):*** These pins support SPI communication using the SPI library [2].
- ***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.

3.3.2.4.1.4) ***Analog Input:***

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 analog Reference () function [2]. Additionally, some pins have specialized functionality:

- ***I2C:*** A4 or SDA pin and A5 or SCL pin. Support I2C 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.

3.3.2.4.1.5) ***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 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 Software Serial library allows for serial communication on any of the Uno's digital pins. The ATmega328 also supports I2C and SPI communication. The Arduino software includes a Wire library to simplify use of the I2C bus.

3.3.2.4.1.6) ***The ATmega328 microcontroller:***

The Atmel ATmega328 [6] is a 32K 8-bit microcontroller based on the AVR architecture. Many instructions are executed in a single clock cycle providing a throughput of almost 20 MIPS (Million Instruction Per Second) at 20MHz. The ATMEGA328-PU comes in 28 pin package. (see figure-3.5)

The following figure-3.7- gives the different pins [7] of the microcontroller:

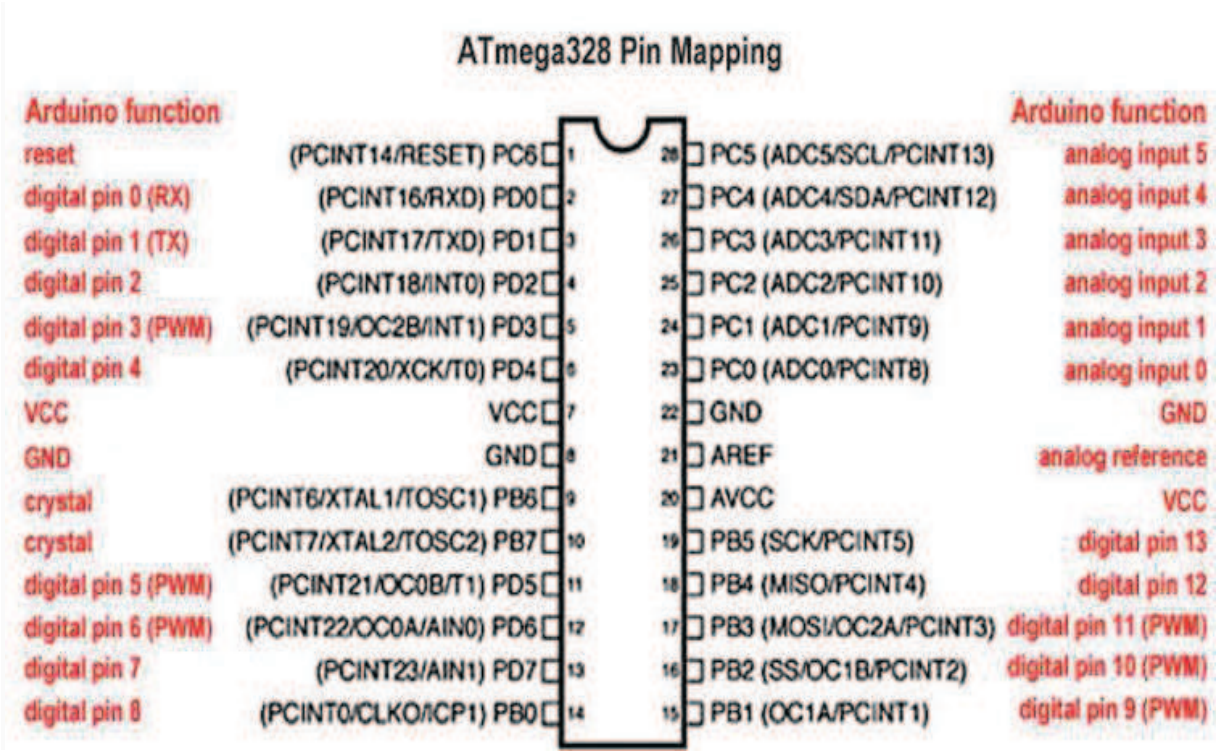


Figure-3.7-: ATmega328 pin mapping

Its characteristics are given by the following table:

Features	Flash	EEPROM	SRAM	MAX I/O pin	Frequency Max	VCC	10-bits A/D channels	Analog comparator	16-bit Timers
Characteristics	32KB	1KB	2KB	23	20KHZ	1.8-5.5	6	Yes	1

8-bit Timer	External Interrupts	Interrupts	ISP	PWM channels	SPI	TWI	UART	Watchdog	Package
2	2	26	Yes	6	1	Yes	1	Yes	PDIP 28

Table-3.1-: shows the characteristics of ATmega328

3.3.3) ***Xbee module:***

In order to control various devices using remote i.e. for wireless transmission, a wireless RF module known as Xbee is used. We recall that **Xbee** is a transceiver module which is used for wireless communication that Digi built based on the 802.14.5/zigbee protocol [8]. It converts digital input given to it into electromagnetic waves of radio frequency which travels through some distance and vice versa [3].

"XBee and XBee-PRO 802.15.4 RF modules (referred as Series1) 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. XBee modules are ideal for low-power, low-cost applications. XBee-PRO modules are power-amplified versions of XBee modules for extended-range applications. While series 2 are used in wide complex range communication, based on ZNET 2.5 zigbee protocol using mesh network.

In our application we are interested in simple point to point communication, so the adequate modules are series 1 which are designed for peer to peer (see section-3.2.3.4-) simple communication between two devices.

Note that we can use the series 2, but series 1 are simpler to use in our case, in addition they are easy to configure then series 2. Using series 2 for simple peer to peer communication maybe complicated since it requires advanced configuration.

We are making use of Xbee S1 pro XPB 24, the main difference of standard Xbee and pro is the range.

The following figure-3.8- shows the XPB24 pro antenna series-1 Xbee module:

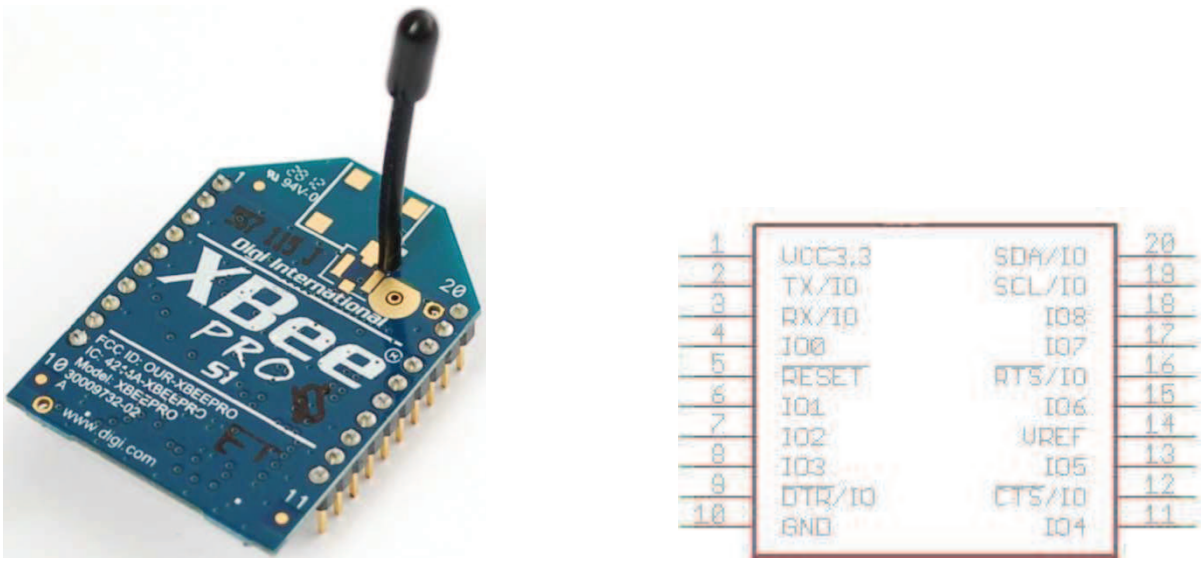


Figure-3.8-: 60mW wire antenna series1 Xbee pro module

The mechanical drawing (top view) is given by figure-3.9- shown below [9]:

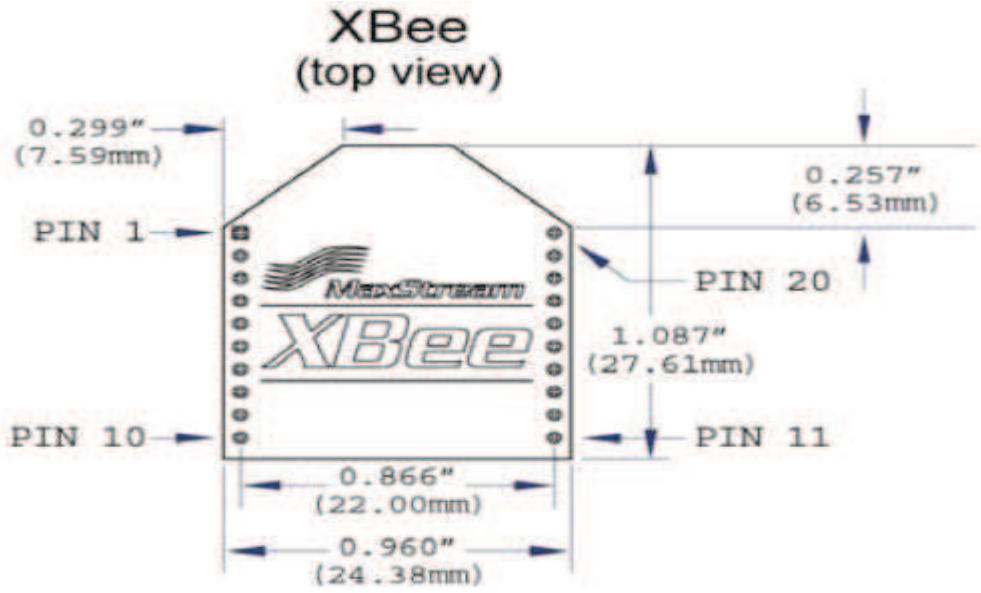


Figure-3.9-: Xbee top view

The XBee has pins spaced 2 mm apart which helps keep the component small, but won't allow us to use it directly on a standard 0.1″ breadboard. In order to connect the Xbee to the Arduino the "Arduino Xbee shield" is used for direct connection without even wires, or we can use a "breakout board" to connect it to a bread board then to Arduino by wires as shown in figure-3.10-.

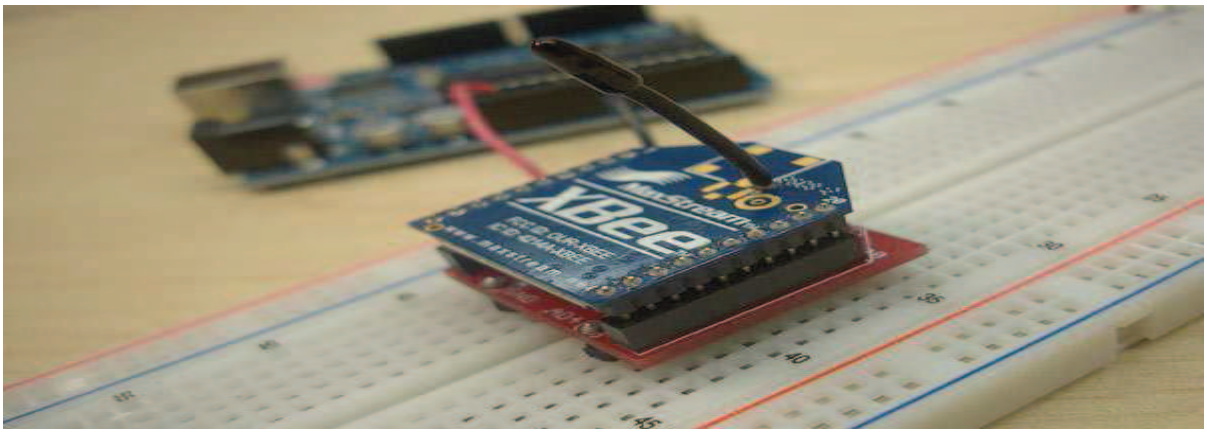


Figure-3.10-: Shows an Xbee connected to a Red breakout board to be linked with Arduino

The module can be configured once connected to RF shield related to a computer via its serial port. The configuration can be done with the software designed for. (See Appendix A)

3.3.3.1) ***Pin assignments:***

The pins assignments for the Xbee series-1 pro module are given in table-3.2- just below [9]:

Pin #	Name	Direction	Description
1	VCC	-	Power supply
2	DOUT	Output	UART data out
3	DIN / $\overline{\text{CONFIG}}$	Input	UART data In
4	DO8*	Either	Digital output 8
5	$\overline{\text{RESET}}$	Input	Module reset (reset pulse must be at least 200 ns)
6	PWM0 / RSSI	Either	PWM output 0 / RX signal strength indicator
7	PWM1	Either	PWM output 1
8	[reserved]	-	Do not connect
9	$\overline{\text{DTR}}$ / SLEEP_RQ/ DI8	Either	Pin sleep control line or digital input 8
10	GND	-	Ground
11	AD4 / DIO4	Either	Analog input 4 or digital I/O 4
12	$\overline{\text{CTS}}$ / DIO7	Either	Clear-to-send flow control or digital I/O 7
13	ON / $\overline{\text{SLEEP}}$	Output	Module status indicator
14	VREF	Input	Voltage reference for A/D inputs
15	Associate / AD5 / DIO5	Either	Associated indicator, analog input 5 or digital I/O 5
16	$\overline{\text{RTS}}$ / DIO6	Either	Request-to-send flow control, or digital I/O 6
17	AD3 / DIO3	Either	Analog input 3 or digital I/O 3
18	AD2 / DIO2	Either	Analog input 2 or digital I/O 2
19	AD1 / DIO1	Either	Analog input 1 or digital I/O 1
20	AD0 / DIO0	Either	Analog input 0, digital IO 0

* Function is not supported at the time of this release

Table-3.2-: The Xbee series 2 pin assignments

3.3.3.2) ***Modes of operation:***

There are five modes of operation [8] as shown in figure-3.11-:

3.3.3.2.1) ***Idle mode:***

When not receiving or transmitting data, the RF module is in Idle Mode. The module shifts into the other modes of operation under the following conditions:

- Transmit Mode (Serial data in the serial receive buffer is ready to be packetized)
- Receive Mode (Valid RF data is received through the antenna)
- Sleep Mode (End Devices only)
- Command Mode (Command Mode Sequence is issued)

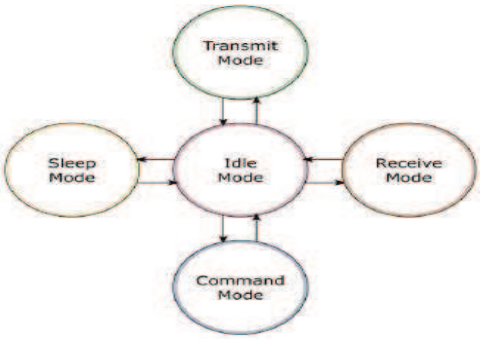


Figure-3.11-: The Xbee operation modes

3.3.3.2.2) ***Transmit mode:***

When serial data is received and is ready for packetization, the RF module will exit Idle Mode and attempt to transmit the data. The destination address determines which node(s) will receive the data. Prior to transmitting the data, the module ensures that a 16-bit network address and route to the destination node have been established. If the destination 16-bit network address is not known, network address discovery will take place. If a route is not known, route discovery will take place for the purpose of establishing a route to the destination node. If a module with a matching network address is not discovered, the packet is discarded. The data will be transmitted once a route is established. If route discovery fails to establish a route, the packet will be discarded as shown in figure-3.12-:

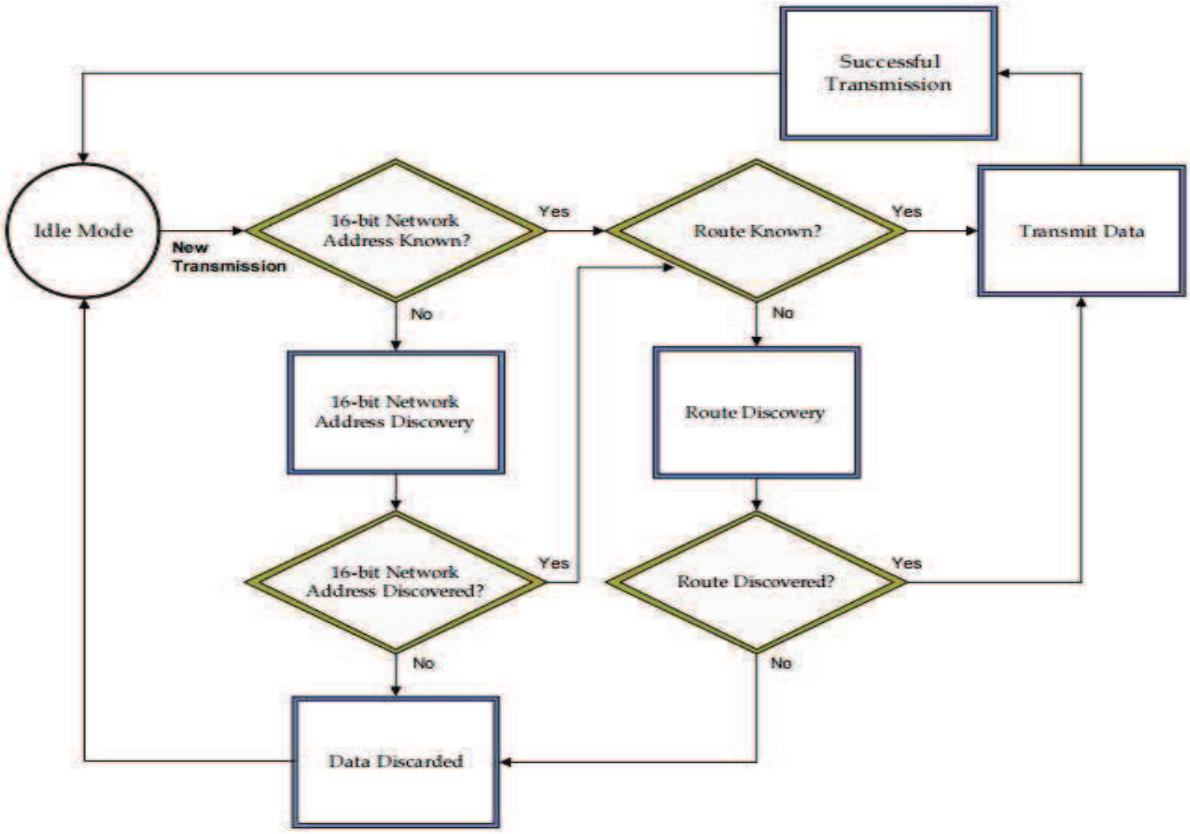


Figure-3.12-: The transmit mode flowchart

3.3.3.2.3) ***Received mode:***

If a valid RF packet is received, the data is transferred to the serial transmit buffer.

3.3.3.2.4) ***Command mode:***

To modify or read RF Module parameters, the module must first enter into Command Mode - a state in which incoming serial characters are interpreted as commands referred as "AT commands"

3.3.3.2.5) ***Sleep mode:***

Sleep modes allow the RF module to enter states of low power consumption when not in use. Sleep modes are supported on end devices only. Coordinator devices participate in routing data

packets and are intended to be mains powered. End devices must be joined to a parent (router or coordinator) before they can participate on a ZigBee network. The parent device does not track when an end device is awake or asleep. Instead, the end device must inform the parent when it is able to receive data. The parent must be able to buffer incoming data packets destined for the end device until the end device can awake and receive the data. When an end device is able to receive data, it sends a poll command (permission) to the parent. When the parent router or coordinator receives the poll command, it will transmit any buffered data packets for the end device.

3.3.3.3) ***Networking concepts:***

ZigBee defines two different device types [8]: coordinator, end device.

3.3.3.3.1) ***Coordinator:***

The most capable device, the coordinator forms the root of the network tree and might bridge to other networks. There must be one ZigBee coordinator in each network since it is the device that starts the network originally. The coordinator initiates a Personal Area Network (PAN) by selecting a RF channel and PAN ID. Coordinator also allows End-devices to join the PAN. It is able to store information about the network, including acting as the Trust Center & repository for security keys.

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

- Selects a channel and PAN to start the network
- Can allow end devices to join the network
- Cannot sleep--should be mains powered.

3.3.3.3.2) ***End Device:***

Contains just enough functionality to talk to the parent node (the coordinator); it cannot relay data from other devices. This relationship allows the node to be asleep a significant amount of the time thereby giving long battery life. An End Device must join PAN before sending any senor data.

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.

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 can allow other devices to join the network and can route data. After an End device joins a coordinator, it must be able to transmit or receive RF data through that coordinator. The 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.

This networking can be configured either: AT (transparent) or API operation.

3.3.3.3.3) ***Transparent operation:***

AT mode is synonymous with "Transparent" mode. In AT mode, any data sent to the XBee module is immediately sent to the remote module identified by the Destination Address in memory. When the module is in AT mode, it can be configured by the user or a host microcontroller by first placing the module in Command mode and then sending predefined AT commands through the UART port. This mode is useful when we don't need to change destination addresses very often, or we have a very simple network, or simple point to point communication.

When using the Xbee in AT mode we are limited to point to point communication, it means simple transmission/reception within the whole network as shown in figure-3.13-:

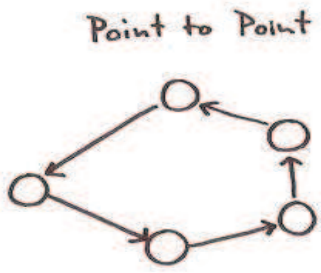


Figure-3.13-: Point to point wireless communication

Two devices in figure-3.13- represent the point to point communication it means, one acts as transmitter while the other receiver. This configuration is also known as peer to peer where there is no coordinator, the devices are all "End devices" related by the same PAN ID. The coordinator is used when we have a lot of devices that need routing. We use API for this type of networking.

3.3.3.3.4) ***API (Address Programming Interface) operation:***

For larger networks that involve nodes talking to multiple targets, API mode is more useful. In API mode, rather than sending AT commands serially, data packets are assembled that include the Destination Address. API mode allows to change destination address much more quickly because Command Mode doesn't need to be entered. API mode is also useful if the user needs to change the configuration of a remote module. This project focuses on AT mode operation.

In API mode, we can trivially send and receive from both the COORDINATOR and many other XBees connected to same network.

In a point-to-multipoint wireless configuration (see figure-3.14-), multiple nodes pictured in at the terminals, send to and receive from a central coordinator (central node).

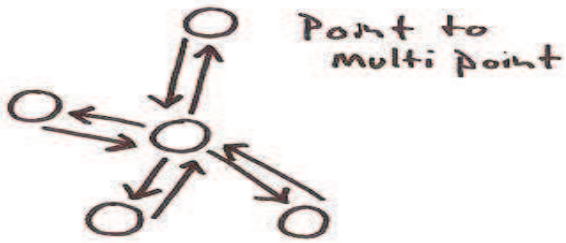


Figure-3.14-: Point to multipoint wireless communication

When dealing with point to multipoint communication, a coordinator is needed for routing between the devices. This is known as the complex communication.

3.3.3.4) ***Basic connection:***

The Xbee module can be connected to Arduino using the following wiring given by figure-3.15-:

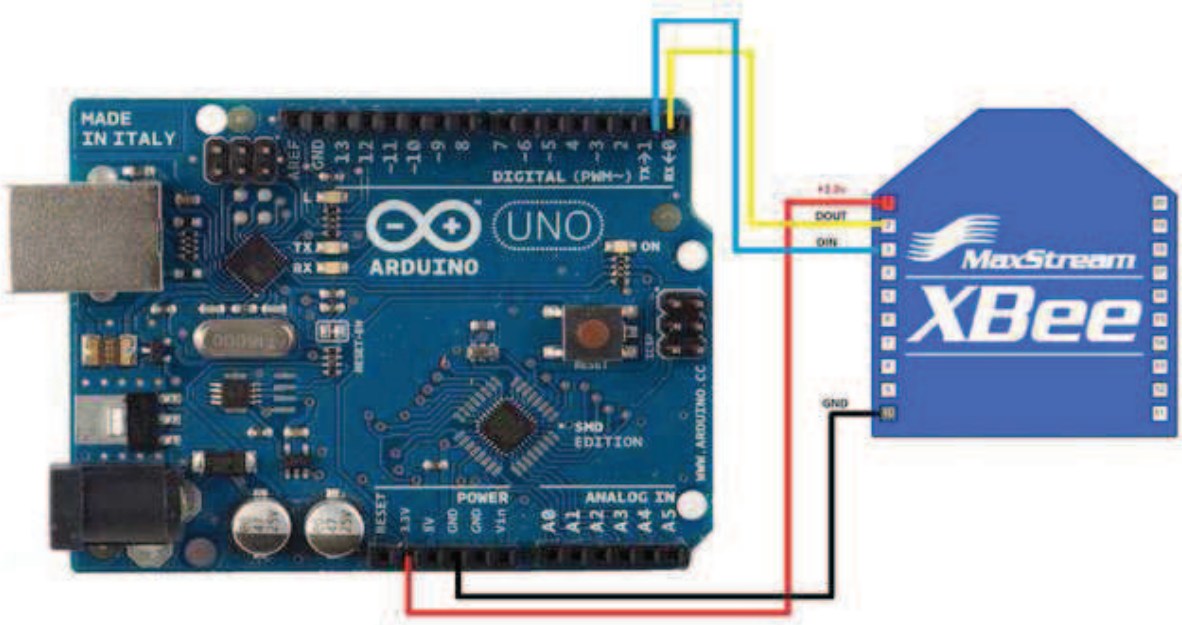


Figure-3.15-: Xbee to Arduino connection

Where the connection is done as follows:

- Arduino 3V3 - XBee Pin 1 (VCC)
- Arduino Ground - XBee Pin 10 (Ground).
- Arduino Tx - XBee Pin 3 (DIN).
- Arduino Rx - XBee Pin 2 (DOUT).

Figure-3.15- shows just how to relate an Arduino to Xbee. But when connecting them for serial communications, we must allow for the fact that the Xbee runs on 3.3V and the Arduino runs on 5V [3]. Below (figure-3.16-) is a simple interface that recommend as a minimum requirement.

In figure-3.16- the 3.3V output from the Xbee's TX pin will be accepted by the Arduino's 5V logic without any problem. The series resistor simply protects the Xbee TX pin in case our program

converts the RX input into a digital output. The 5V output of the Arduino's TX pin is divided by the 10K and 15K resistors to just over 3V.

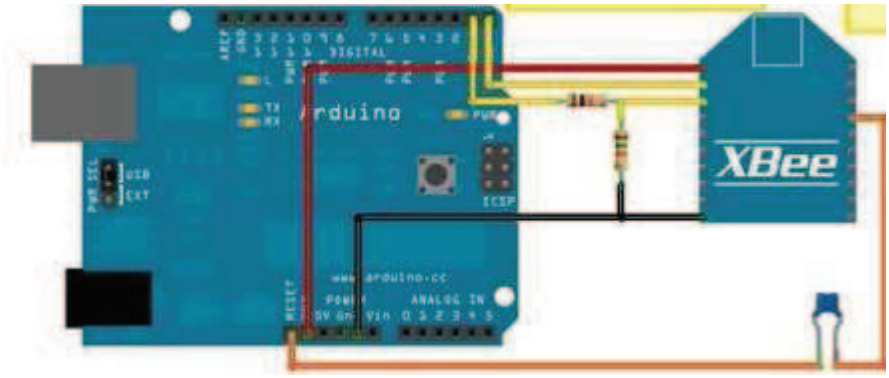


Figure-3.16a-: Arduino to Xbee hardware circuitry

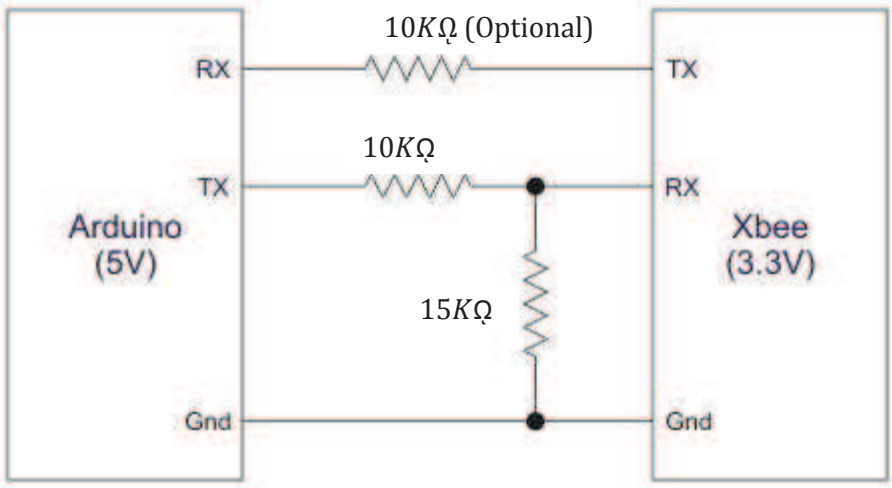


Figure-3.16b-: Arduino to Xbee power interfacing security

Then the voltage across the RX pin of the xbee and the ground is :

$$V_{RX} = \frac{15K}{10K + 15K} * 5V = \frac{75}{25}V = 3V$$

For interfacing one Arduino with another (see figure-3.17-), two Xbee are placed between two Arduino kits. One Arduino kit acts as a transmitter whereas the other acts as a receiver. RF shield provides a wireless medium to carry signals from transmitter to receiver. When the input is given to joystick (see figure-3.1-), it is transmitted to first arduino (transmitter) and then receiver arduino receives the signals through RF shield. These digital signals are then further processed using different types of circuits (Two servomotors in our case).

Figure-3.17-: shows two Arduinos communicating using Xbee

3.3.4) ***The Servomotors:***

We are interested to use servomotors than stepper motors for three reasons, they are:

- Quiet.
- Available in all sizes
- Precise control for angular position.

The servomotors that we are going to use are from "***HITEC HS-65HB*** *servomotor***"** [10] (see figure-3.18-), having the same basic components as described in section (2.4.3.2.2).

3.3.4.1) ***Specifications***

- Control system: +PULSE WIDTH CONTROL 1500usec NEUTRAL
- Weight: 11.2 g
- Dimension: 22.2 x 11.8 x 31 mm approx.
- Operating speed: 0.16 s/60 degree at no load
- Operating voltage: 4.8 V (~5V).
- Temperature range: -20 ºC --+ 55 ºC

Figure-3.18-: *The HS-65HB servomotor.*

As we have seen in chapter II, it has 3 wires:

- Red for powering.
- Black for Ground.
- Yellow for control.

In our case the servomotor will be controlled by the Arduino UNO by sending control signals through its PWM pins (see section 2.4.3.2.3) in chapter II)

3.3.4.2) ***Basic Connection:***

The following figure-3.19- shows a general concept of how connecting a servomotor to an Arduino [2]. Of course this is simple, we just need to power and ground the servo by 5V and GND Arduino pins, and connect the yellow wire for control to the one of the PWM Arduino's signals.

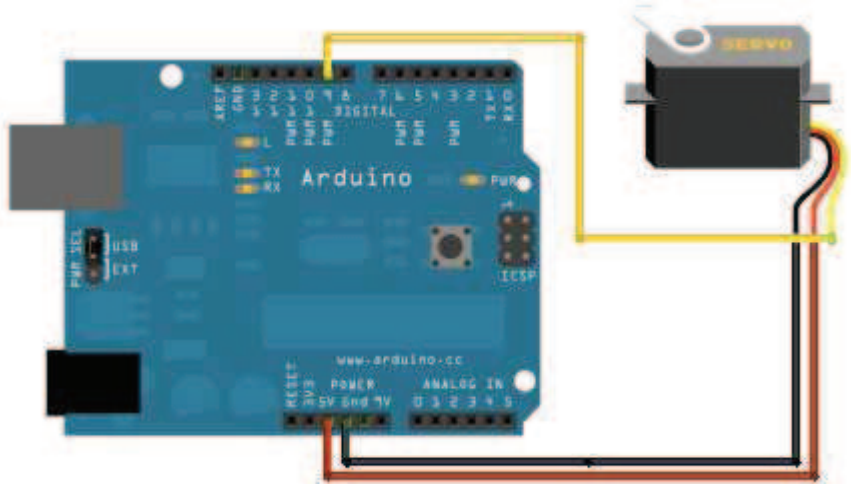


Figure-3.19-: Basic connection of a servomotor to an Arduino.

3.4) ***Final system block scheme:***

The block scheme given in figure-3.1b- can be detailed by figure-3.20- shown below. Well; this figure shows that the operator exerts some force (Fh) to move the joystick to the desired position, then the joystick converts the mechanical movement to electrical signals (Xm, Ym) with respect to the desired position. These signals are then uploaded to the main controller board (Arduino1) that sends them via communication channel (xbees transceivers).

Once the signals are sent, this take some time delay to reach the destination (slave site), so this is expressed by the two terms (X(t-T), and Y(t-T)) where "T" shows the time delay.

After reception the signals are once again uploaded to the slave main controller board (Arduino 2) which become the slave desired position (Xsd, Ysd). Then these two lattes are input under PWM signals to control the specified servomotor.

It would be interesting to get the system mathematical model by deriving the state space representation for more advanced studies, but we don't have the parameters to extract the physical equations to do so. This leads us to explain why we are making use of servomotors. Well, servomotors are equipped with an internal regulator that ensures the servo-system regulation, so we don't need to do it by ourselves which means that we are not obliged to find the parameters that can take a lot of time. The servomotors loops shown in figure-3.20-, Xs and Ys try to attempt Xsd and Ysd using the servomotors internal control law (closed loop control) as shown in figure-2.21-

Note that the servomotor feedback loops representation is not unity, this is given just to show that they act as closed loop control system.

In unilateral teleoperation, the information flow is in one direction or unidirectional. Master system that is driven by the human operator sends the necessary inputs (e.g., position, and/or velocity) through the communications line to drive the slave system. There is no feedback

information sent to the master system or the human operator during this type of manipulation. Instead, in most of the cases, the slave system has a local closed-loop control system as shown in figure-3.21-, in this case, the operator cannot be involved in the loop control which means that the operator is passive cannot cause instability issues. Hence the operator can just send control signals which the slave interprets with respect its interactions with the environment.

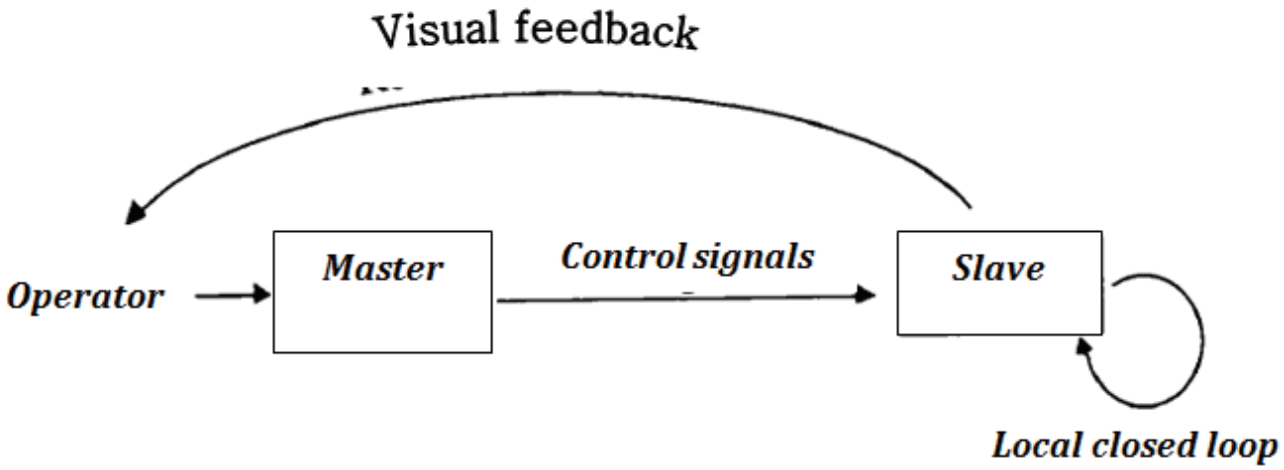


Figure-3.21-: Unilateral information flow

3.5) ***Conclusion:***

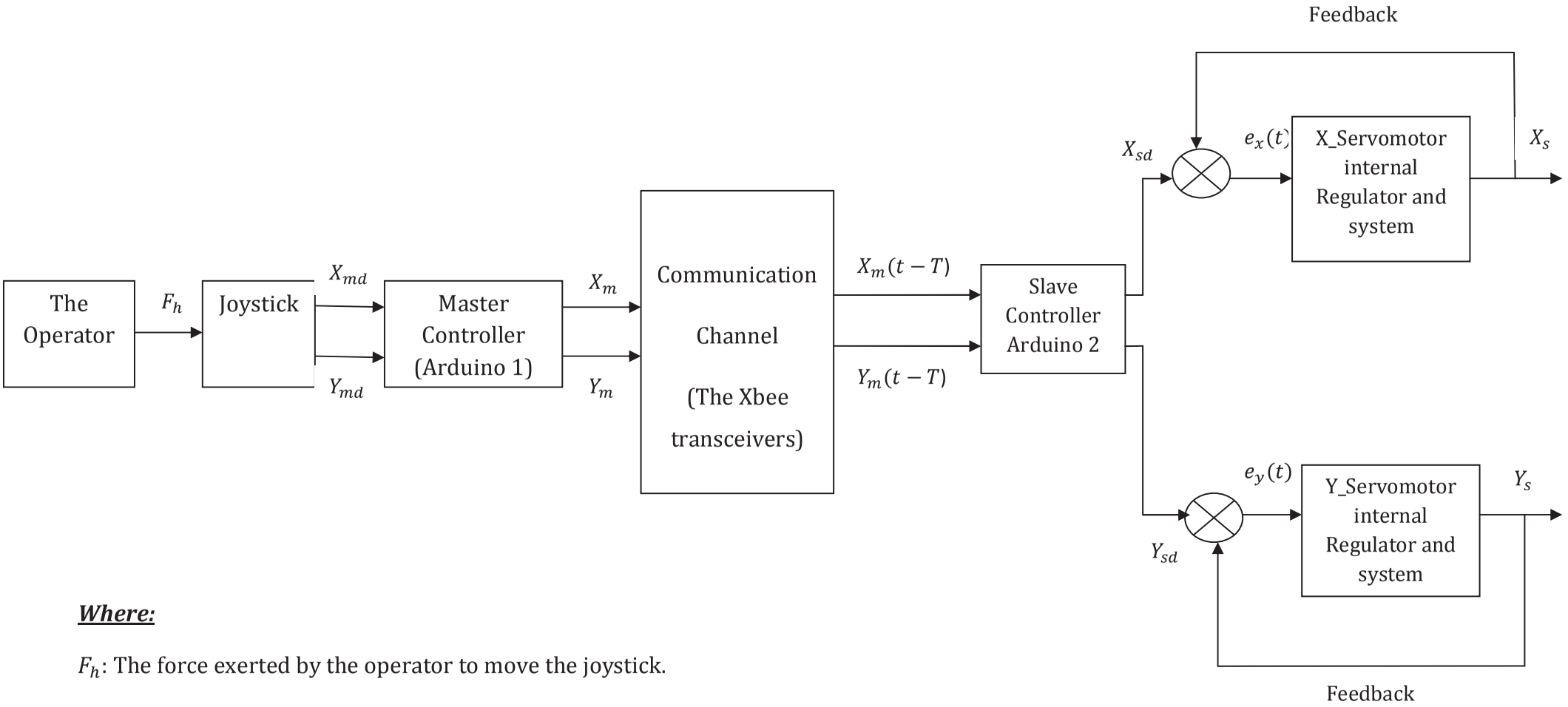
Throughout this chapter we have discussed the components chosen to build our system as illustrated by figure-3.1-. Well; the first block represents the standard thumb joystick which is widely used in remote control that we have connected to Arduino UNO that is the most standard conceived for beginners and easy to program, consisting of the group of the needed features such as analog and digital I/O, converters, timers, interrupts, reset....etc, to manipulate and control tasks.

The Arduino is then connected to Xbee series 1 which the standard wireless transmission module based on the Zigbee norm that is used in modern wireless communication known for its reliability.

The transmission and reception is ensured by the two Xbee modules, where the second is connected to the second Arduino (located in the remote site) that controls the two servomotors in the four directions depending on the operator desired position. The servomotors are equipped with an internal regulator ensuring the local closed loop in order to make sure that Xs and Ys attempt Xsd and Ysd as shown in figure-3.20-

The system described in this paper is position-visual feedback unilateral control system, Since every teleoperation system either unilateral or bilateral has a visual feedback. In order to do so an "IP camera" is necessary to let the operator know the plinth's position.

Finally we have finished by giving the more detailed block scheme in figure-3.20-, with respect to figure-3.1b-.



Where:

F_h: The force exerted by the operator to move the joystick.

X_{md}, Y_{md}: The master desired X and Y servomotor positions respectively.

X_m, Y_m: The master X and Y actual positions.

$X_m(t-T)$, $Y_m(t-T)$: The master actual positions delayed by "T".

X_{sd}, Y_{sd}: The slave desired X and Y servomotor positions respectively.

X_m, Y_m: The slave X and Y actual positions.

Figure-3.20-: Final System Block Scheme

REFERENCES

References:

[1] https://learn.sparkfun.com/tutorials/arduino-comparison-guide

[2] www.Arduino.cc

[3] *APPLIANCE REMOTE CONTROL USING ARDUINO (http://ijltet.org/wp-content/uploads/2013/07/5.pdf)*

[4] http://www.arduino.cc/en/Main/Software

[5] Arduino a quick user guide by Maik Schmidt

[6] https://www.sparkfun.com/datasheets/Components/SMD/ATMega328.pdf

[7] http://www.hangar42.nl/article.php?title=diyduino

[8]Xbee/xbee pro RF module (http://ftp1.digi.com/support/documentation/90000982_S.pdf)

[9] Xbee/Xbee pro RF module (http://site.gravitech.us/MicroResearch/Wireless/XBee-SER1-C/XBee_Manual.pdf)

[10] GENERAL SPECIFICATION OF the SH-65HB servomotor datasheet "http://hitecrcd.co.jp/products/hitec/servo/pdf/33065.pdf"

Chapter IV: The Implementation

4.1) ***Introduction:***

In this chapter we are about to implement our system by showing both hardware and software aspect. Thus; we focus on how the main parts are related to each other to design the system, and describing the hardware and software materials and programs used to do so.

In software look we are going to talk about the Arduino Software which used to program the Arduino, and the XCTU to configure the Xbee modules. In hardware look we will talk about the whole system connection and circuitry.

4.2) ***Xbee configuration:***

Before connecting the Xbee to Arduino , we first have to configure the Xbee modules to communicate between each other, which means that one module is the transmitter while the other one is the receiver.

In order to connect or relate two devices when dealing with communication we have to connect them to the same network (same PAN ID). To do so, we use the ***XCTU*** software provided by Digi (the Xbee manufacturer) to setup the modules, and makes sure that these two lattes are related by the same network for data exchanging.

Since we are interested by simple peer to peer communication using two modules (xbee1 for transmission, xbee2 for reception), we use them as End devices for data exchanging. Well, we have just to join them to same network (PAN ID). (See the next section)

The steps depicting how to configure the modules with the software are described in details in appendix "B".

4.2.1) ***Network Configuration:***

The three most important Xbee settings are:

- Channel.
- PAN ID (see section-2.5.2.3.2.3-).
- MY address (Source address).
- Destination Address (DH and DL).

4.2.1.1) ***Channel:***

Channel controls the frequency band that the XBee communicates over. Most XBee's operate on the 2.4GHz 802.15.4 band, and the channel further calibrates the operating frequency within that band. The channel is set to its standard ("C") value before any changes the XBP24 Xbee is ranged from 0x0C to 0x17 (12 possible hex values).

4.2.1.2) ***PAN ID:***

PAN ID stands for personal area network ID (see section 2.5.2.3.2.3). The network ID is some hexadecimal value between 0 and 0xFFFF (16 bit PAN ID). XBees can only communicate with each other if they have the same network ID. There being 65536 (2^{16}) possible ID's, there's a very small chance that our neighbor will be operating on the same network (as long as we change it from the default).

4.2.1.3) ***MY Address:***

Each XBee in a network should be assigned a 16-bit address (again between 0 and 0xFFFF), which is referred to as **MY address**, or the "source" address.

4.2.1.4) ***Destination Address:***

Destination address determines which source address an XBee can send data to. For one XBee to be able to send data to another, it must have the same destination address as the other XBee's source. It has 64 bit length divided in two parts; Destination Low labeled "DL" (0-0xFFFFFFFF), and Destination High labeled "DH" (0-0xFFFFFFFF)

For example, if XBee 1 has a MY address of "0x1234", and XBee 2 has an equivalent destination address of "0x1234", then XBee 2 can send data to XBee 1 (see table-4.1-).

But if XBee 2 has a Destination address of "0x5201", and XBee 1 has a MY address of "0x5200", then XBee 2 cannot send data to XBee 1. In this case, only one-way communication is enabled between the two XBee's (only XBee 1 can send data to XBee 2). For our case the second example is equivalent.

Setting	**Acronym**	**XBee 1**	**XBee 2**
Channel	**CH**	C	C
PAN ID	**ID**	C345	C345
Destination Address High	**DH**	0	0
Destination Address Low	**DL**	1240	1234
16-bit Source Address	**MY**	1234	1240

Table-4.1*-:* data communication in two ways configuration

This table shows the required connection between two modules for data exchanging it means; the TX of Xbee 1 is connected to Rx of Xbee 2 and Rx of Xbee 1 to Tx of Xbee 2, in this case the two modules can communicate between each other.

Setting	Acronym	XBee 1	XBee 2
Channel	**CH**	C	C
PAN ID	**ID**	9907	9907
Destination Address High	**DH**	0	0
Destination Address Low	**DL**	7000	3001
16-bit Source Address	**MY**	3000	7000

Table-4.2-: one way data transfer configuration

In this case only Xbee 1 can send data to Xbee 2, hence the destination address xbee1 must be the same as the one of xbee2 source address as shown in table-4.2-. Whereas we can choose any MY for xbee1 and DL for xbee2. (see figure-4.7c- and figure-4.7d).

After getting finished with the Xbees configuration, the Arduino programming is the next step.

4.3) ***The Arduino programming:***

The system has two Arduinos; one for the master site, the other one for the slave site. So; the first one is programmed as transmitter while the second one is programmed as a receiver.

The Arduino master site is programmed in such a way that:

- The Arduino reads the two potentiometers' variation of the joystick using two of its six analog inputs. (See figure-3.3a- for wiring).
- These variations are the control signals that Arduino master site loads to its serial communication (TX).

- The Arduino TX is connected to the Xbee DIN for data loading (see figure-3.15-), i.e; the control signals are loaded into the Xbee module via its DIN pin, then this latter sends them wirelessly after being configured as a transmitter (See Appendix B).

The Arduino slave site is programmed in such a way that:

- It receives the controls signals from the Xbee 2 Dout to its RX.
- Then; controlling the two servomotors (the plinth). The servomotor to be controlled depends on potentiometers' variation. (Desired position).

Now, we can represent the system flowchart functionality.

4.4) ***The System Flowchart:***

Before giving the system flowchart, let's give some explanations about the program flow. The two analog input from joystick to Arduino are labeled "xp" and "yp";

- xp: is the x potentiometer value (x axes).
- yp: is the y potentiometer value (y axes).

These two positions are respectively connected to "A0" and "A1" Arduino analog input. We have also used two other variables labeled "x" and "y" to read the "xp" and "yp" values.

The Arduino has a 10bit ADC, so the analog input once printed are ranged from $0\ to\ 2^{10}-1$, i.e from $0\ to$ 1023. Where "xp" and "yp" are in the middle about "510" at rest. (see figure-4.2.b-).

When we press the joystick, the "xp" and "yp" positions change meaning that controls signals are available. Then we can read them and interpreting the following algorithmic sketch.

4.4.1) ***The Algorithmic sketch:***

The servomotors are initialized to 90 degrees i.e in the middle. Its positions can be ranged from 0 to 180 degrees.

4.4.1.1) ***The algorithmic sketch at master site:***

When the "x,y" potentiometers axes change, control signals are sent to control the x_servomotor and y_servomotor as follows:

- Declare all variables
- Read data from joystick ("xp" and "yp").

Subprogram to x_servomotor:

- If "xp" is greater "750" and if "x_pos" is smaller or equal to "180" degrees, in this case we increment the x_pos.
- Else if "xp" is smaller "400" and if "x_pos" is greater or equal then "0" degrees, in this case we decrement the x_pos.
- Send the desired control signal wirelessly.

Subprogram to y_servomotor:

- Also: If "yp" is greater "750" and if "y_pos" is smaller or equal to "180" degrees, in this case we increment the y_pos.
- Else if "yp" is smaller "400" and if "y_pos" is greater or equal then "0" degrees, in this case we decrement the y_pos.
- Send the desired control signal wirelessly.

Where:

x_pos: stands for the position of x_servomotor.

y_pos: stands for the position of y_servomotor.

Note that the chosen values "750 and 400" are arbitrary.

4.4.1.2) ***The Algorithm sketch at slave site:***

- Receive the data sent from master site.
- Read the control signals.
- Applying the required task either by moving the x_servomotor or y_servomotor.

Finally the system flowchart is given just below in figure-4.1-. Where:

- The first decision box "***if data is available***" means if the joystick is pressed to read the potentiometers' variation.
- Another instruction that must be explained is "***if x_pos(new)!=x_pos(old)***" shown in the last decision box. Well; this instruction compares the previous value of x_pos to the new one, and it's obvious that if there's no equality, then the x_pos is changed. In this case the joystick is pressed in "x" axes to move the x_servomotor. Else if this's not true, this means that the joystick is pressed in "y" axes to move the y_servomotor. Somehow it's an algorithm to let the slave know which servomotor to move.

4.5) ***Hardware Implementation and results:***

4.5.1) ***The joystick variation:***

The connection of joystick to Arduino is shown in figure-4.2a- exactly as the circuitry given in figure-3.3a.

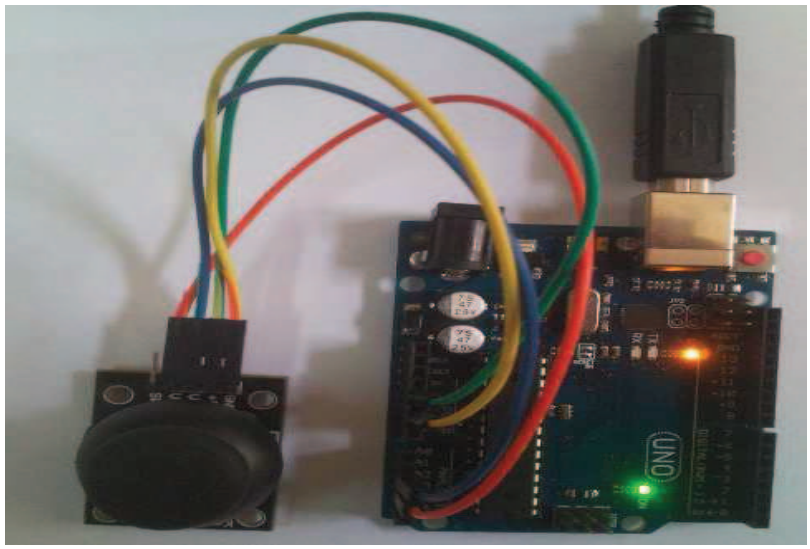


Figure-4.2a-: Joystick to Arduino connection

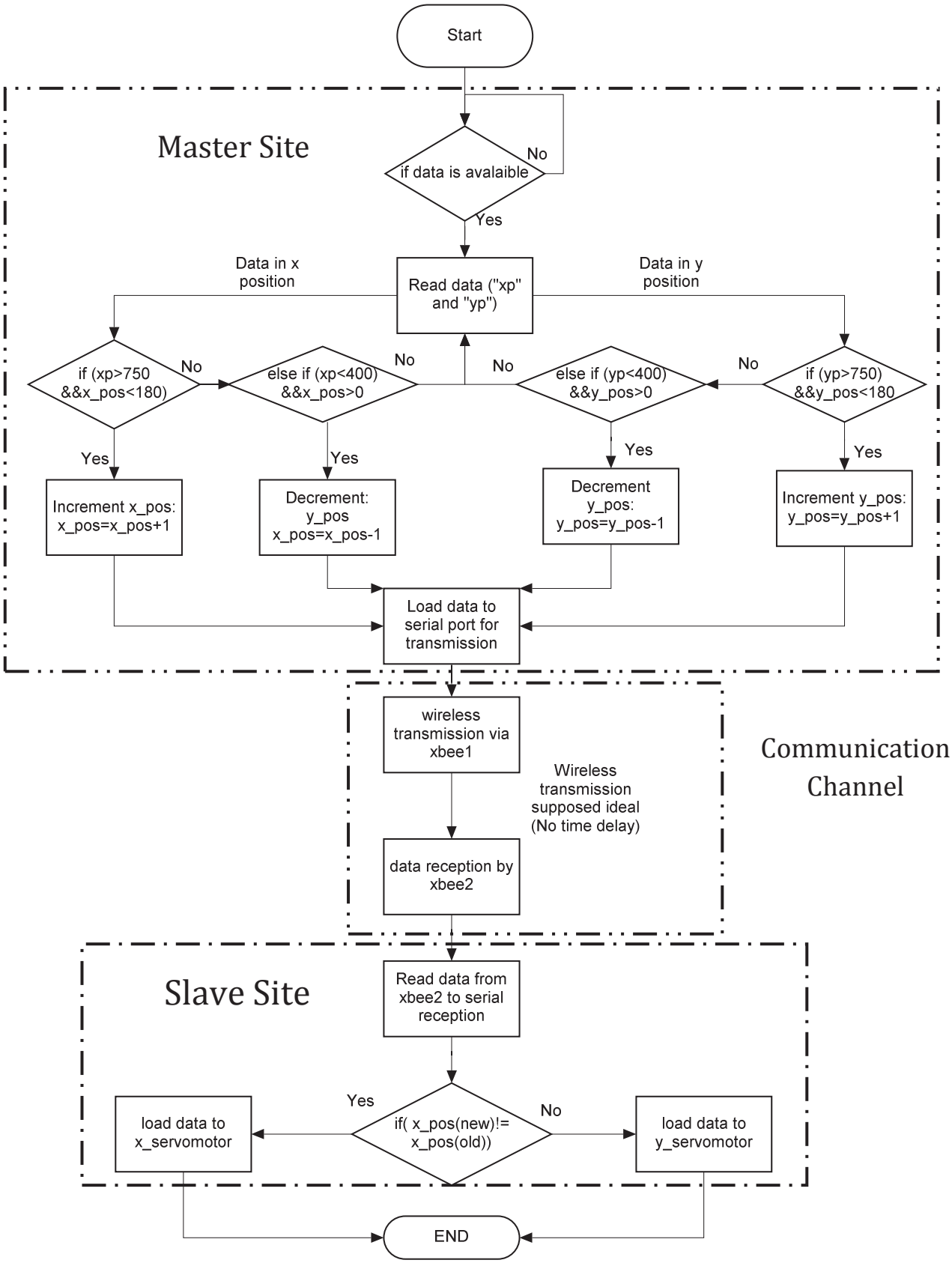


Figure-4.1-: System Flowchart

The potentiometers' variation is shown in figure-4.2b-, we have taken the case where there's no press, and in the two axes (x and y).

- No variation:

```
X:510   Y:510
X:510   Y:510
```

- Varying "x" from min to max while "y" is in the middle:

```
X:1023   Y:510
X:1023   Y:510
```

```
X:0   Y:510
X:0   Y:510
```

- Varying "y" from min to max while "x" is in the middle:

```
X:512   Y:0
X:510   Y:0
```

```
X:510   Y:1023
X:510   Y:1023
```

Figure-4.2b: Different values taken by joystick's potentiometers

4.5.2) ***Servomotors' positions:***

Initially the two servomotors are in the middle "90" degrees. Once data given by figure-4.2b- is read and following the flowchart given by table-4.1-, here is the variation of the servomotor shown in figure-4.4-:

- The servomotors at rest:

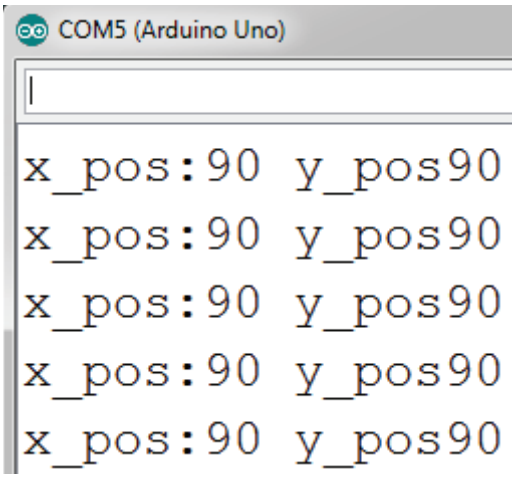


Figure-4.4-: The servomotors position before receiving any control signal

- Changing the x_servomotor position by selecting two cases (decreasing to 0 and increasing to 180):

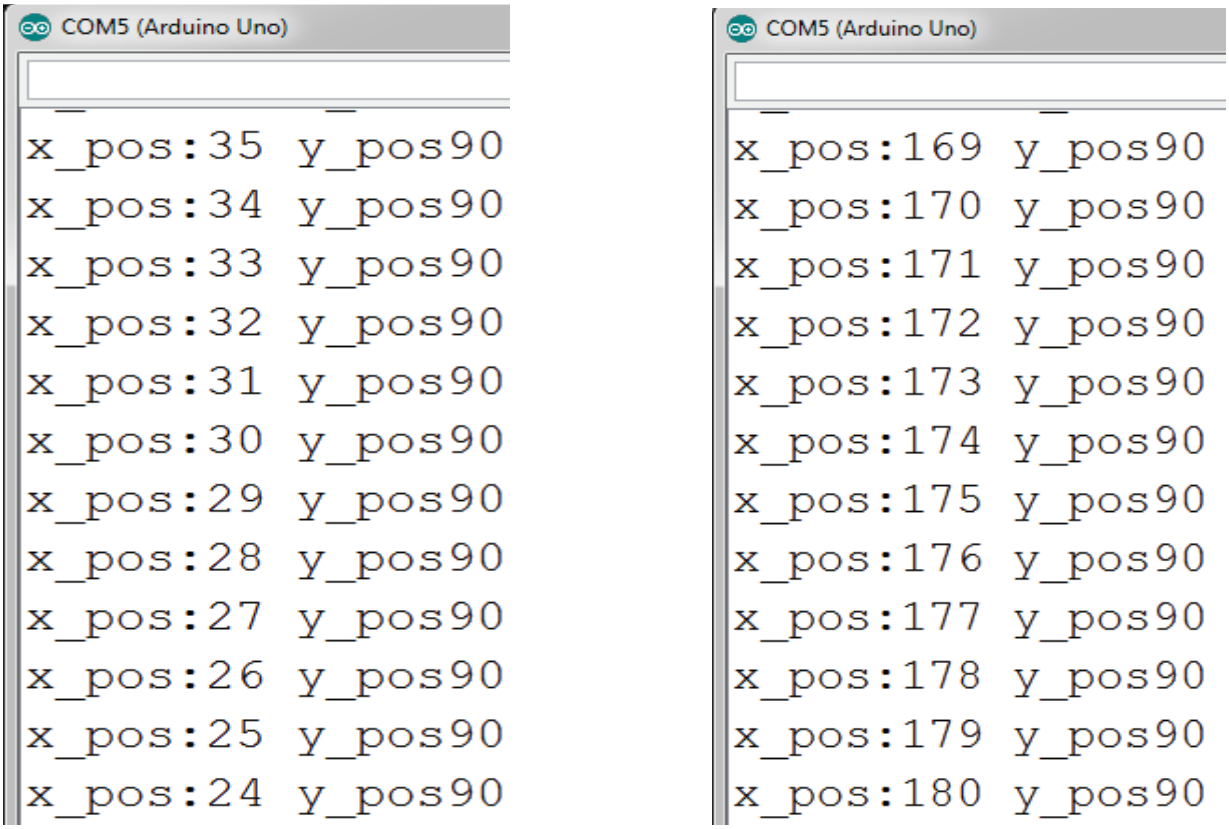


***Figure-4.5**-:* from left to right respectively: the decreasing and increasing of the x_servomotor position.

- Changing the y_servomotor position by selecting two cases (decreasing to 0 and increasing to 180):

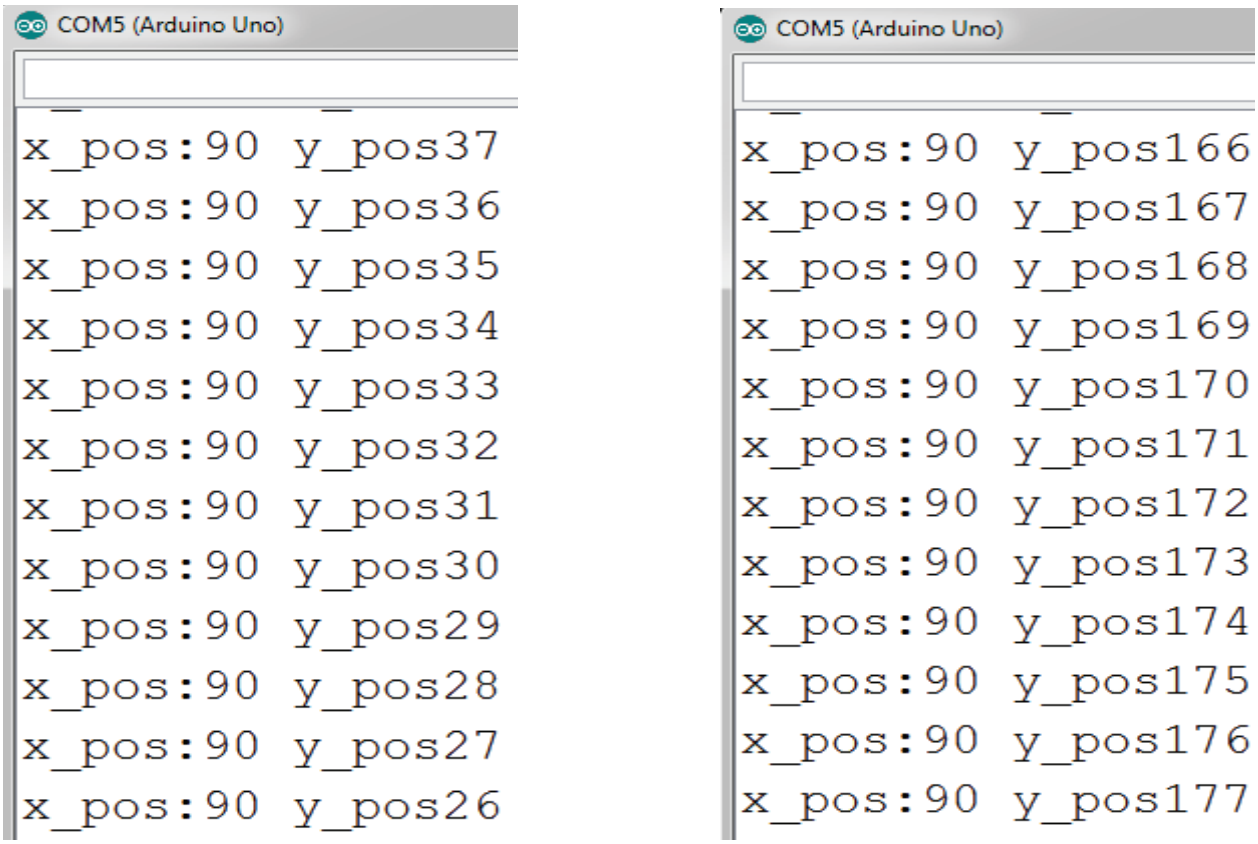


Figure-4.6-: from left to right respectively: the decreasing and increasing of the y_servomotor position.

Then, those values are sent wirelessly with the following xbees configuration parameters shown in table-4.3-:

Setting	**Acronym**	**XBee 1**	**XBee 2**
Channel	**CH**	C	C
PAN ID	**ID**	9907	9907
Destination Address High	**DH**	0	0
Destination Address Low	**DL**	3000	1000
16-bit Source Address	**MY**	1000	3000

Table-4.3-: Xbee parameters configuration

Note that this configuration is a two-way configuration which means that the two xbee modules are configured as transceivers, however; we need just one way (see table-4.2-). But this not a problem at all, since it works with this configuration.

Most teleoperation control systems suffer from time delay, but in our case with these modules they are so reliable that there is absolutely no time delay ($T \approx 0$), which is the main reason of using them.

Figure-4.7- depicts the xbee parameters in the software as the ones shown in table-4.3-:

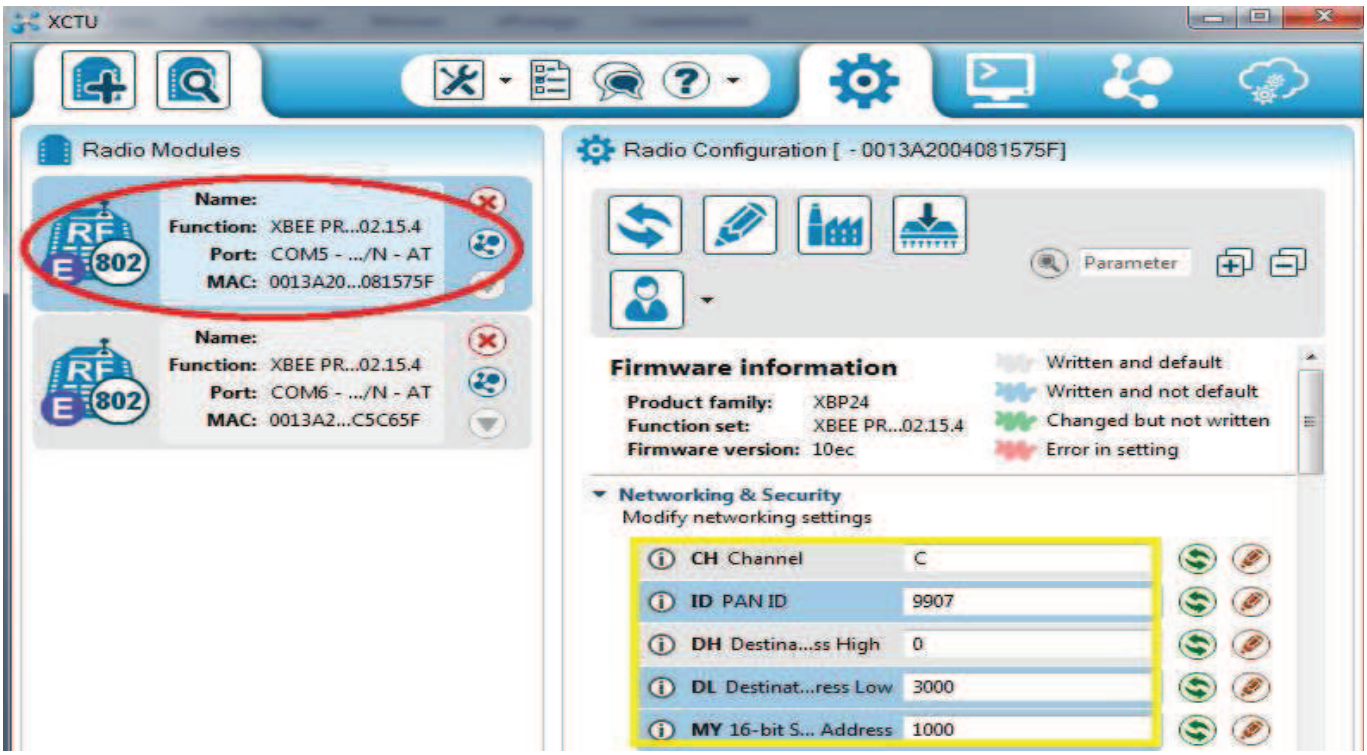


Figure-4.7a-: The xbee1 configuration

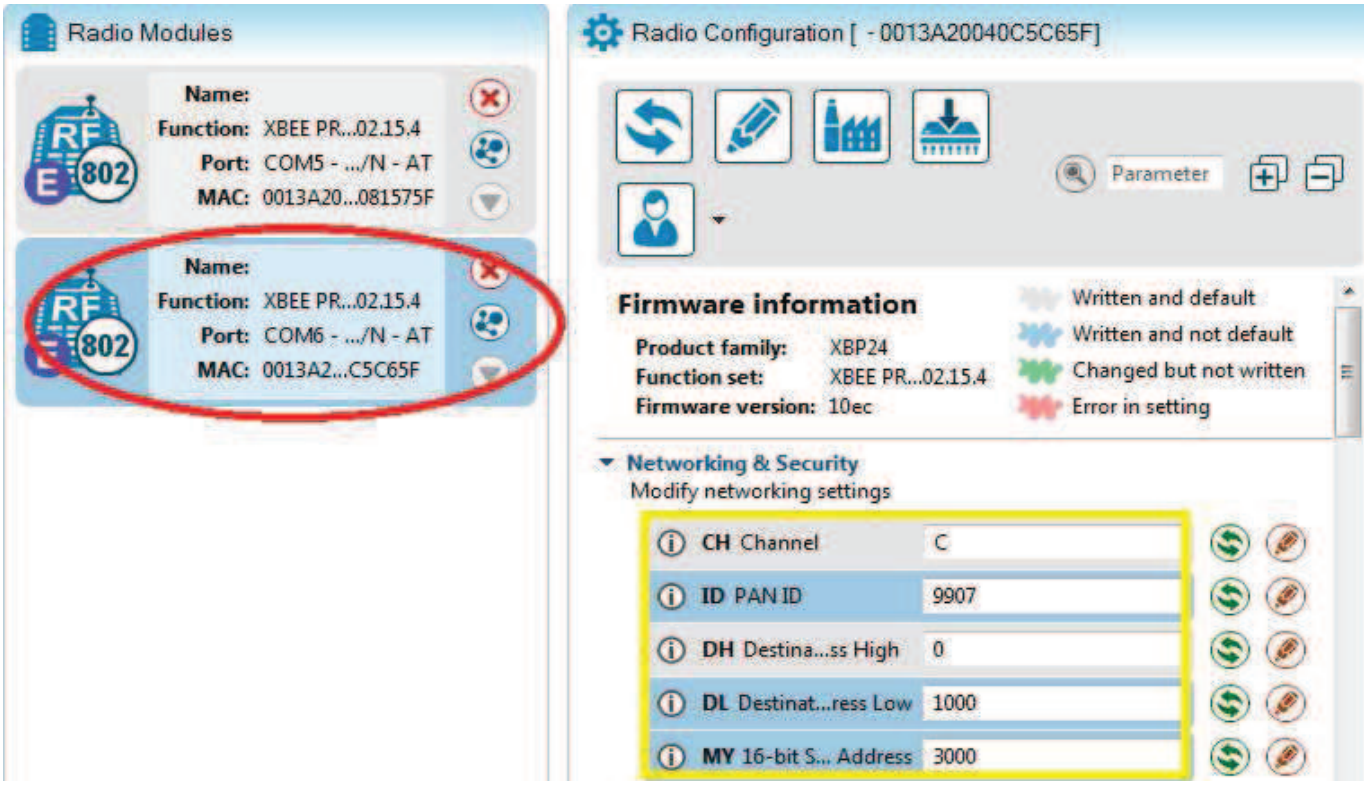


Figure-4.7b-: The xbee2 configuration

We show by the red circle the selected xbee and its appropriate configurations by the yellow box. The two xbees are configured as transceivers, but we use just one way to control i.e the xbee at com5 is the transmitter and the one at com6 is the receiver.

The configurations given in table-4.2- are shown in figures (4.7c and 4.7d) :

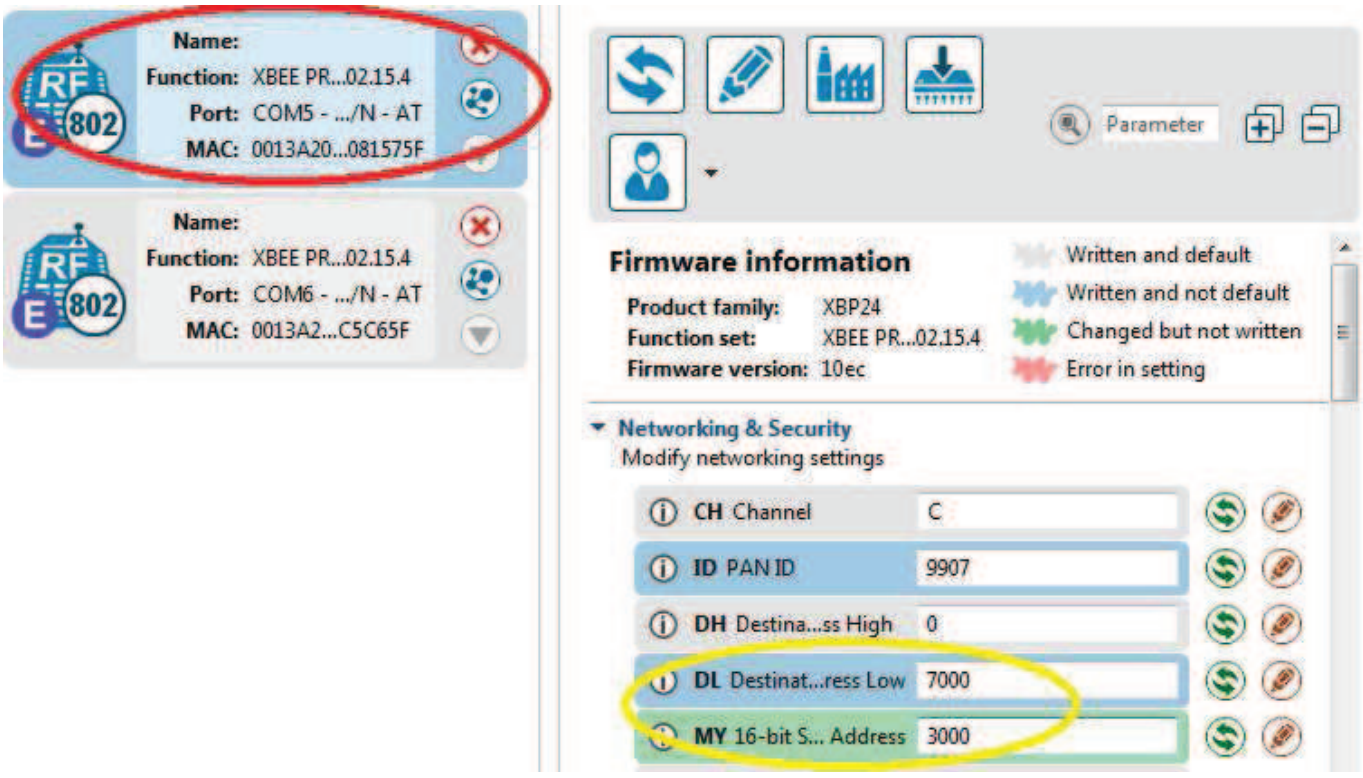


Figure-4.7c-: The xbee1 as one way transmitter.

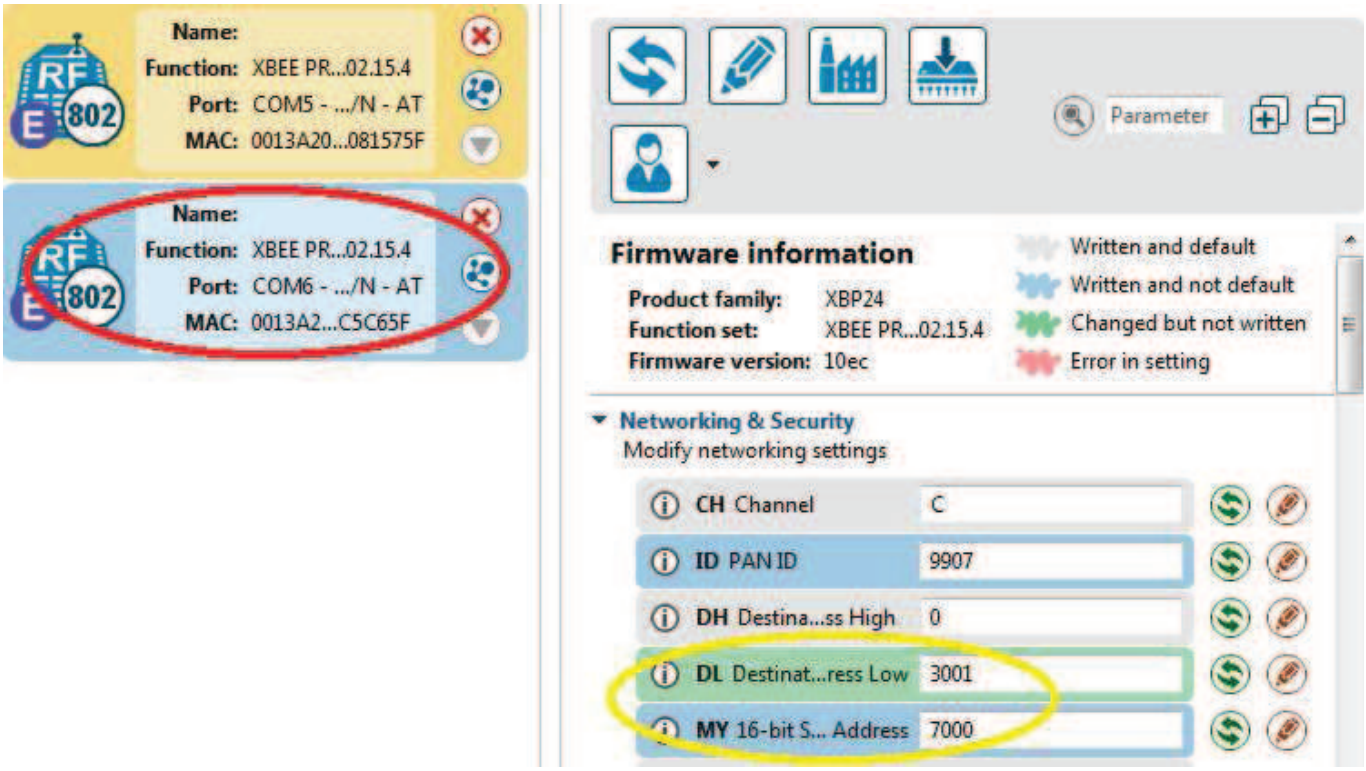


Figure-4.7d-: The xbee2 as one way receiver.

4.5.4) ***System building circuitry:***

- The master site is shown in figure-4.8- including the joystick, Arduino and the Xbee1:

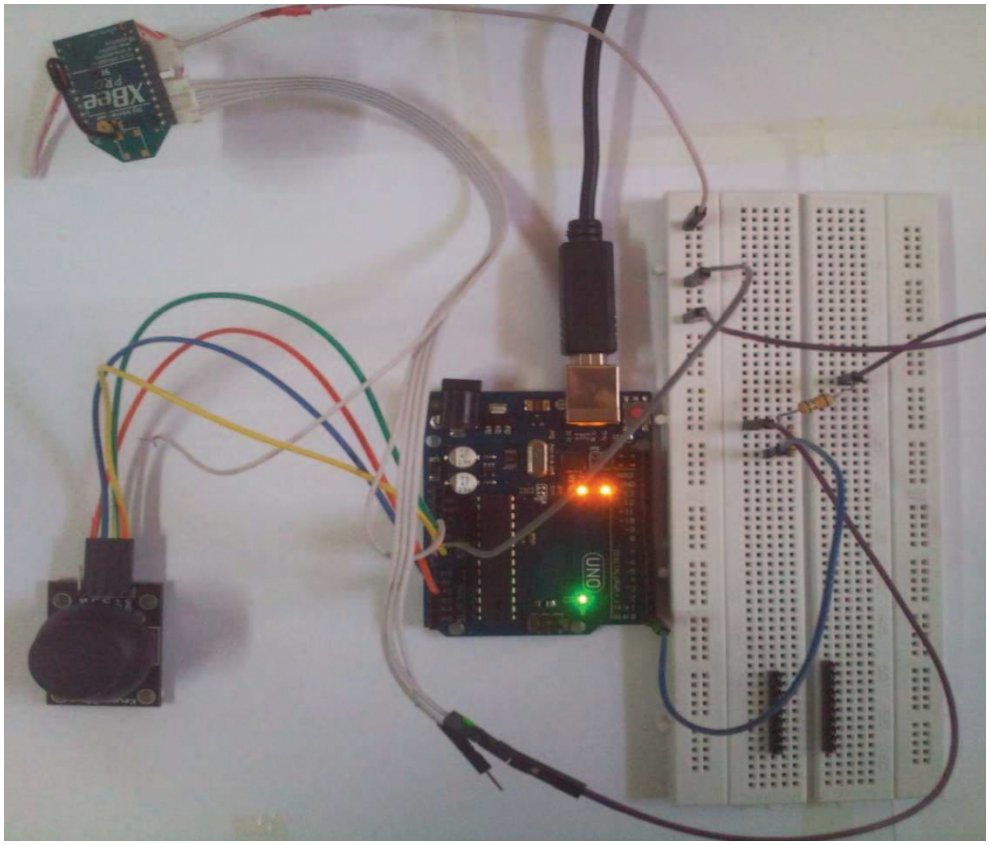


Figure-4.8-: The master site composed by Joystick+Arduino+xbee1

Figure-4.8- is the combination of the two figures (3.3a and 3.16)

- The slave site is shown in figure-4.9- including the Xbee2, Arduino and the two servomotors. In this picture we show the internal components of the plinth (the slave) before building the mechanical support.

It's important to note that the two xbee modules are not included in both master and slave, hence; they represent the way of communicating between the two sites. However their inclusion just with respect to the circuitry.

Figure-4.9- is given as follows:

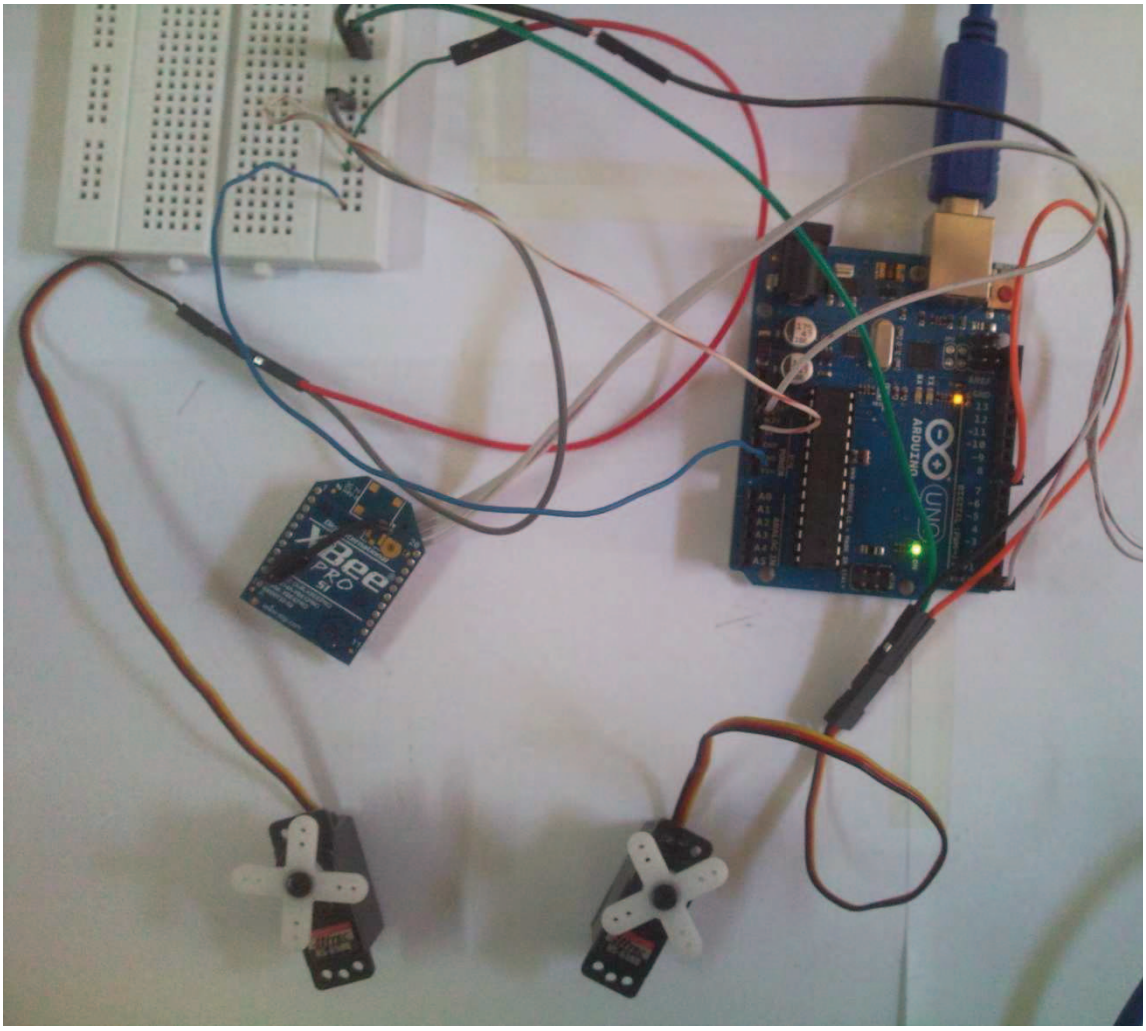


Figure-4.9-: The slave site composed by xbee2+Arduino+servomotors

Now the whole hardware system is the combination of the two figures (4.8 and 4.9)-

The plinth is holding a simple camera just to show its movements. The plinth is shown in figure-4.10- holding a camera:

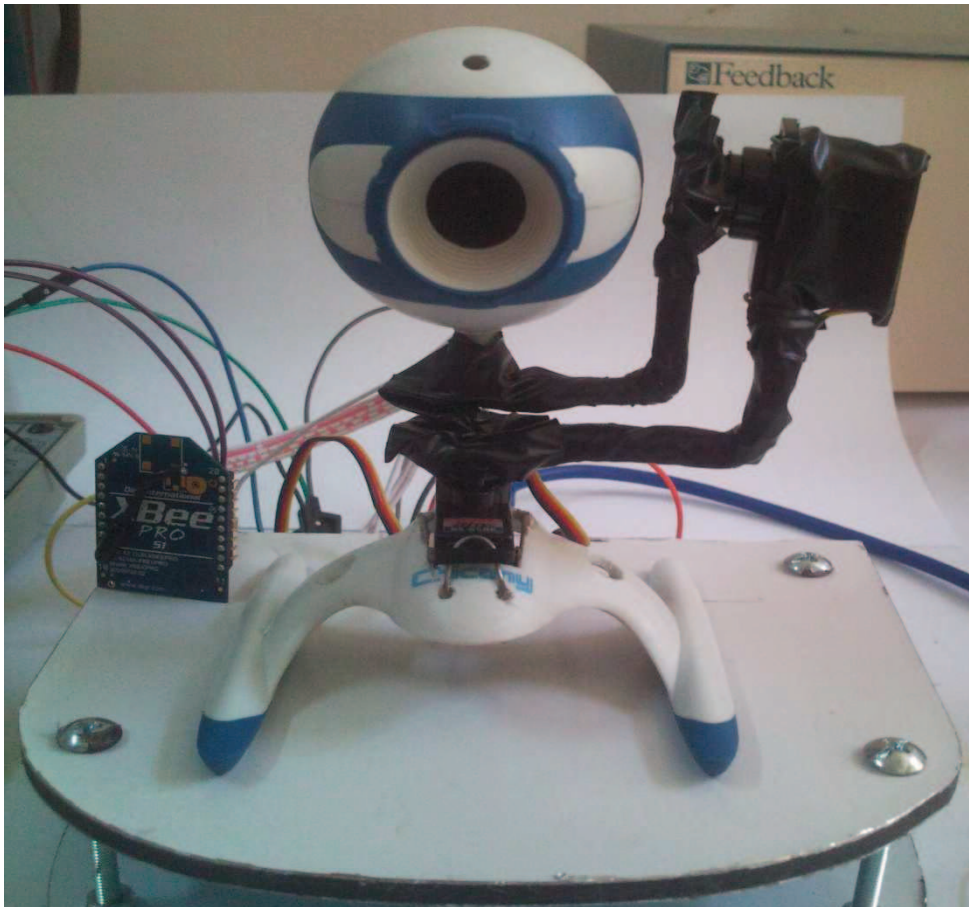


Figure-4.10-: The plinth holding a camera

- The controls signals from master to slave site are shown in figure-4.11-:

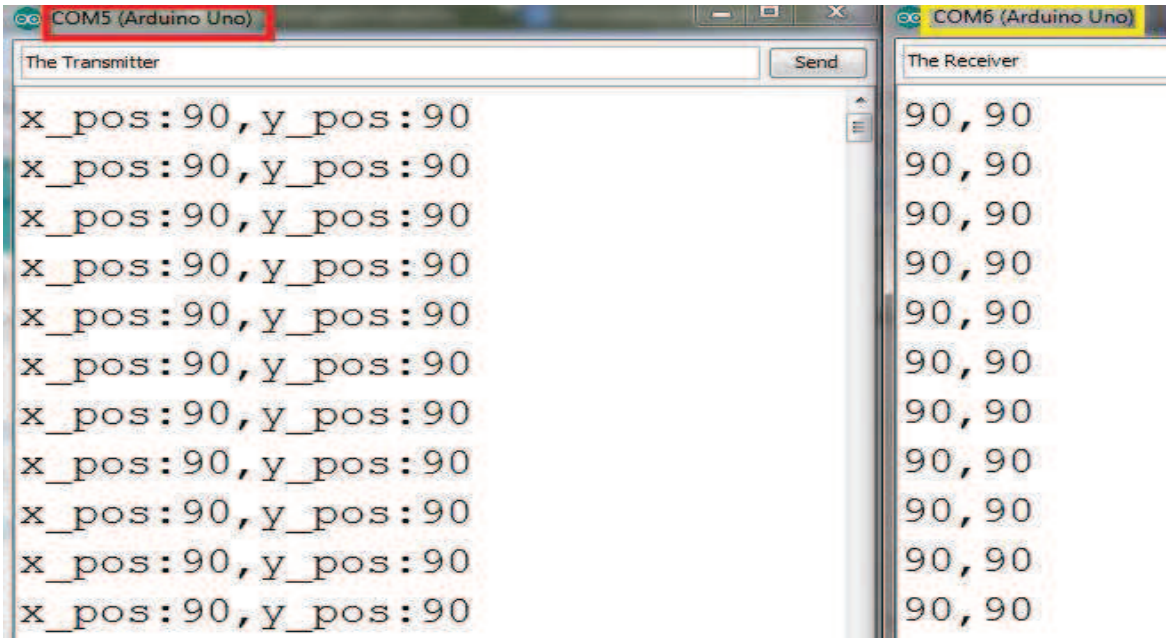


Figure-4.11: From left to right; the transmitter (COM5) send controls signals to Receiver (COM6)

Figure-4.11-, shows the plinth in the middle position.

- Here is another case where the signals changes given by figure-4.12 shown below:

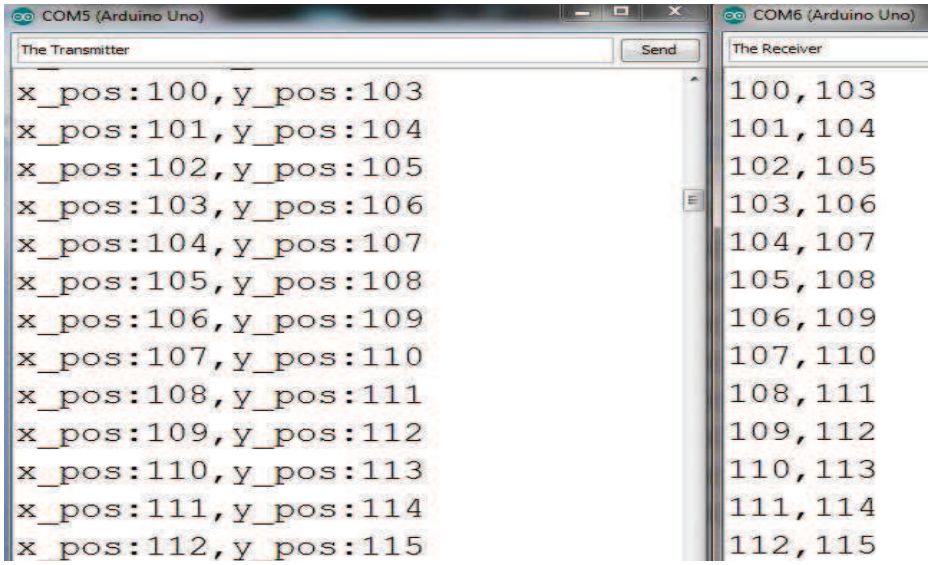


Figure-4.12-: the case where the signals change

These signals can be interpreted by the following figure-4.13- shown in the next page:

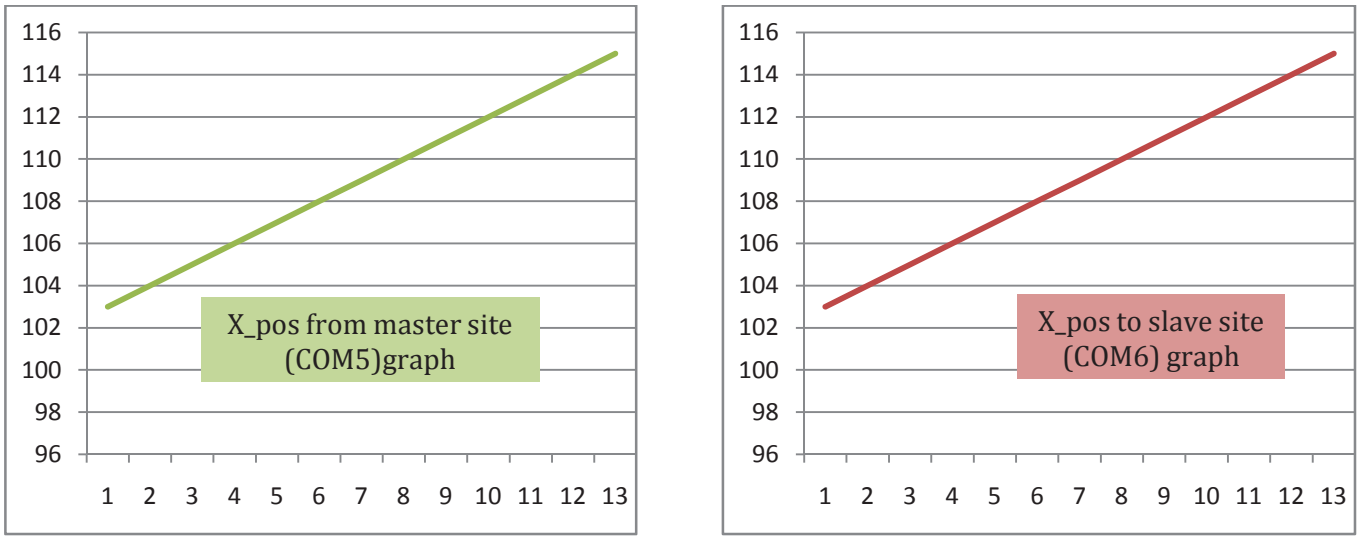


Figure-4.13a-: from left to right; controls signals of x_servomotor graph from master site (Green) to slave site (brown) with respect to figure-4.12

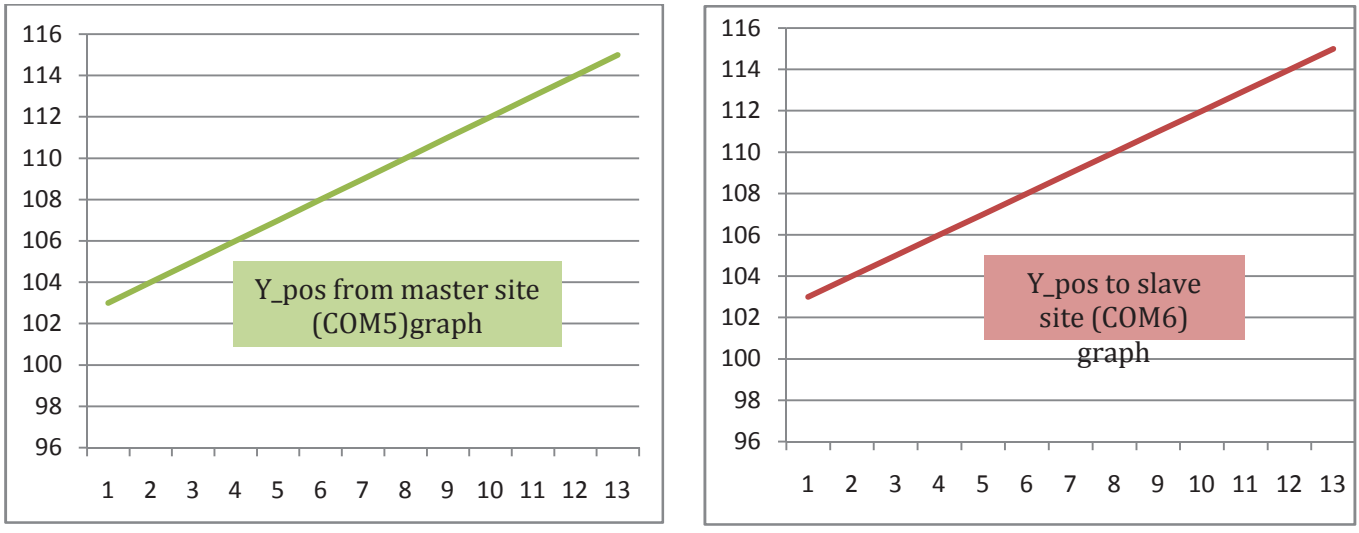


Figure-4.13b-: from left to right; controls signals of y_servomotor graph from master site (Green) to slave site (brown) with respect to figure-4.12

- Finally the following figure-4.14- shows the plinth position for some different variation given:

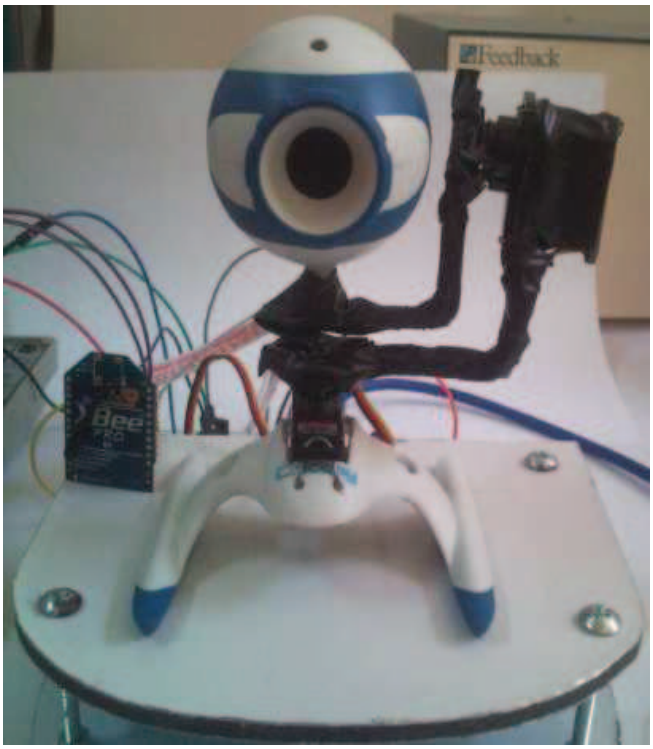


Figure-4.14a-

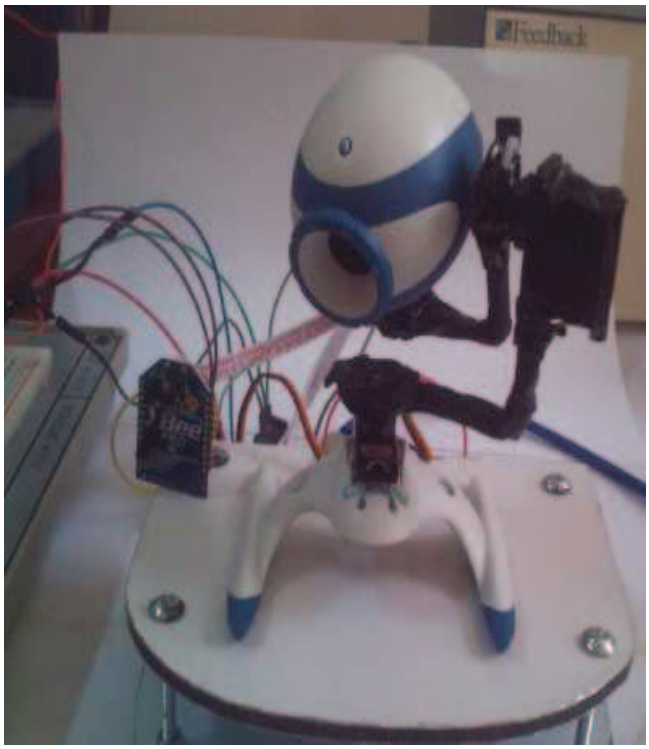


Figure-4.14b-

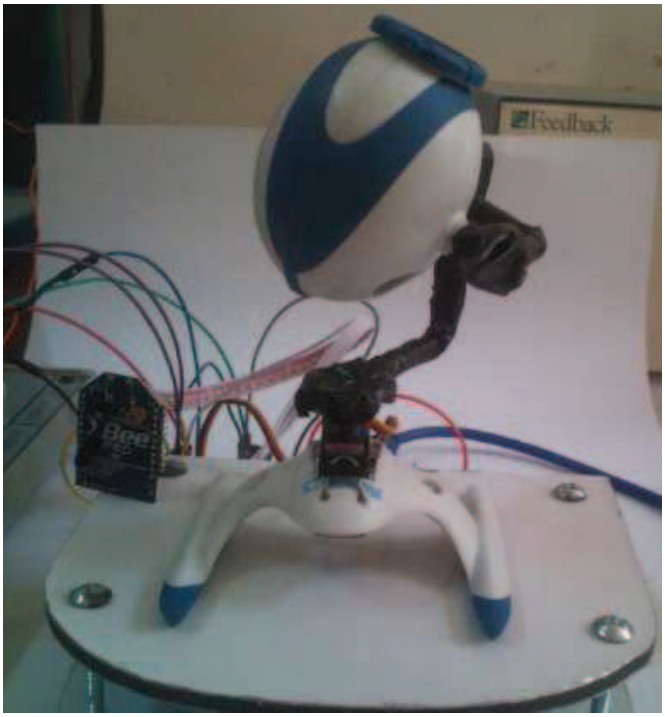


Figure-4.14c-:

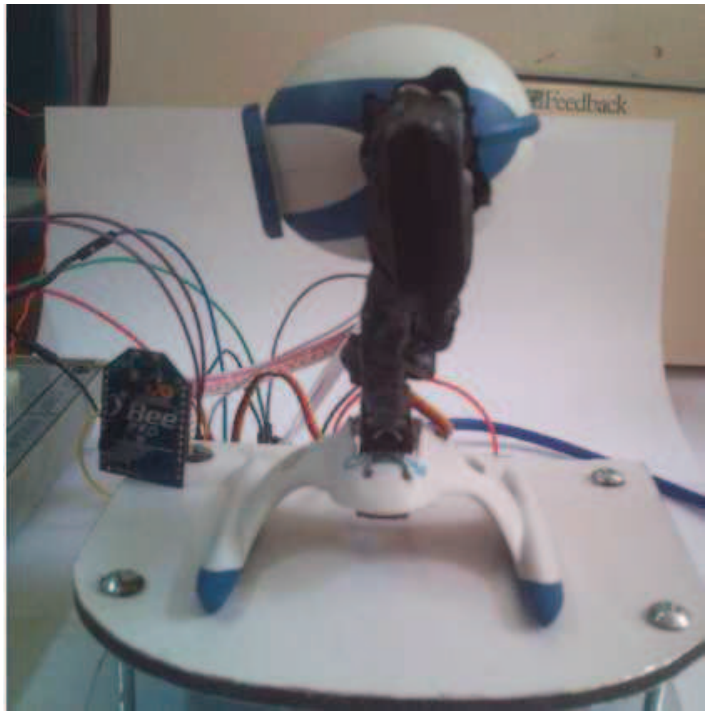


Figure-4.14d-:

4.6) ***Conclusion:***

Throughout this chapter, we have joined the conception made in chapter III and implemented our system by focalizing on the theoretical studies about the functionality of each component.

Well; we have started by configuring the xbee modules as transmitter/receiver as END devices, to ensure the wireless communication between master/slave sites. Note that before any configuration the modules are configured by default, it means as end devices. We have also concluded since we are dealing with simple peer to peer communication, we use the modules under AT commands, in this case we don't need a coordinator since it's not a complex networking [8]

Then we have programmed the two Arduinos in such manner that the one located in master site sends the joystick signals which represent the operator desired position via xbee1. And the second one is programmed in such a way to retrieve this signals and applying them to the servomotors.

Our goal in this paper is to control the plinth wirelessly, and as mentioned in chapter II, the plinth is a supporting block made to hold something to be manipulated. In order to see exactly how it moves, we have installed a simple camera on it.

We have also concluded that within the range permitted by the modules, we don't meet any time delay.

REFERENCES

Reference:

[1] Arduino programming note book by Brian.W Evans.

[2] Introduction to Arduino by Alen.G Smith.

[3] http://www.digi.com/support/productdetail?pid=3352&type=utilities

[4]http://coderdojoic.org/sites/default/files/MEGA2560%20Stater%20Kit%20Tutoral.pdf

CONCLUSION & FUTURE WORK

Conclusion:

The main objective of our work is to investigate the control of a plinth remotely in teleoperation control system. We leant on the practical aspect, which would be interesting if we have given the mathematical model of the whole system, and making and advanced study about its theoretical functionality. All of these are part of the perspectives that will be mentioned at the end of the conclusion.

After giving a general overview of teleoperation, "chapter I" has begun to present the essential part of a standard teleoperation system and the relation that exists between each subsystem, then giving the two types of commands which are unilateral and bilateral control. We have seen in unilateral the system acts as an open loop with no feedback; in this case there's no information available to the operator except the visual feedback if there is any, however bilateral teleoperation there is feedback information such as haptic feedback where the operator can feel what the slave is manipulating. It's important to note that **chapter I** is based on the study of the generalities of teleoperation, that's why we have given all the possible cases that exist for example we have illustrated the teleoperation system building block where we have seen that the operator is a simple mass spring damper system by deriving its equivalent equation and so on. After that we have seen an important point which stands for the hybrid matrix given by equation (1.17) permitting us to study the performances such as stability, transparency, passivity....etc as seen in section-1.8-.

Lawrence 1993 [5] has given the standard four channel architecture of a general teleoperation control system, where we can derive from the two-channel architecture by modifying the internal controllers (C1.....C6). The two channel architecture is the most used; the number of channels stands for the number of controllers relating the whole systems. This architecture contains four classifications: PP-FF-FP-PF, for each used classification, the corresponding controllers is implemented to ensure the adequate control.

After getting finished with generalities we have talked about the deferent excising components and devices that could be used in our system since we are interested for implementation. Thus **chapter II** has given an overview of all possible cases we can have and devices we can use starting by defining the plinth (slave) representing the main part to be controlled by giving its standard components (motors, EBB) and explaining its internal circuitry, this is referred as the slave site.

There is no slave without master, so the master is consisted of an electronic joystick nicknamed "thumb joystick" used for simple control having two potentiometers that vary with respect the position and 4 main pins: Vcc, GND, X, Y. And an EBB having the same characteristics as the slave's one, then for communication between the two sites we have mentioned the two possible ways that we can have i.e wired and wireless communication.

After defining the global blocks to be used, **chapter III** depicts the conception by choosing the adequate components to build up our system, well; the joystick used is a standard manufacturer used to send control signals, then the EBB used is an Arduino-UNO known as the most standard microcontroller designed for beginners and easy to program, next for communication line we have used the wireless communication protocol (WPAN) known as xbee for its reliability and less power consumption. Finally we have the system main part (the plinth) having two servomotors and another Arduino-UNO.

Chapter IV deals with the implementation and system circuitry, thus we have started by configuring the communication line as transceivers, the xbee modules are configured as END devices (AT mode) by default, i.e if we have two modules for a simple peer to peer communication as in our case, it's just necessary to connect them to the same PAN ID and same Channel. Or we can associate a coordinator that plays a role of router in this case, for more complex networking. We have also seen throughout this chapter that a standard xbee requires at minimum five configurations (CH,PAN ID, MY, DH, DL) for a complete functionality, then seen that it cannot be connected directly to a breadboard because of its pins small distanced, so we use either a shield or we can configure it using the Arduino as serial port (see Appendix-B).

After finishing with communication line, we have programmed the two Arduinos in such a way that the joystick loads the potentiometers' variation made by the operator, and uploads these signals to its serial transmission (TX), to be sent via the wireless line. The xbee works with 3.3 V, so if we connect directly the Arduino TX pin to xbee "Din" pin, it will burn up the xbee, that's why we have made a voltage divider to decrease the 5V to 3 V for protection. Once the signals are uploaded to xbee "Din" pin, they will be sent by xbee1 and received by xbee2.

After reception, the xbee2 downloads these control signals to the plinth main controller to control the servomotors in the desired position. Finally different figures are given to show the result of reading and sending the control signals in real time, also we have concluded that the modules have a negligible time delay with respect to the experiences made. In order to complete the formulation of the unilateral teleoperation system, a visual feedback is necessary ensured by an "IP camera". The IP camera is used when the master and slave are located far away from each other; however in our case we limit the distance so that the operator can do it by himself. Thus we have used a simple camera just to show the plinth position and motion.

Future Work

As perspectives it will be interesting to derive the mathematical model of the system to make an advanced study about its internal block scheme functionality by identifying the parameters of each single part to make a coherent relationship of all parts. In our case we have used servomotors having their own regulator ensuring the control law, thus; we have concluded that it's not necessary to find the control law by identifying the parameters which would take a lot of time that we don't have.

For completing the system, the camera held by the plinth is supposed "IP" ensuring the visual feedback to the operator.

LIST OF FIGURES

- ***Chapter I***:

- ***Chapter II:***

- ***Appendix A:***

- ***Appendix B:***

LIST OF TABLES

- ***Chapter III:***

- ***Chapter IV:***

APPENDICES

A) ***About Arduino:***

A.1) ***Arduino guide in few steps:***

Getting Started with Arduino on Windows

This document explains how to connect your Arduino board to the computer and upload your first sketch.

1 | Get an Arduino board and USB cable

2 | Download the Arduino environment

3 | Connect the board

4 | Install the drivers

5 | Launch the Arduino application

6 | Open the blink example

7 | Select your board

8 | Select your serial port

9 | Upload the program

1 | ***Get an Arduino board and USB cable:***

A standard USB cable (A plug to B plug) is needed (the kind we would connect to a USB printer, for example). See figure-A.1-

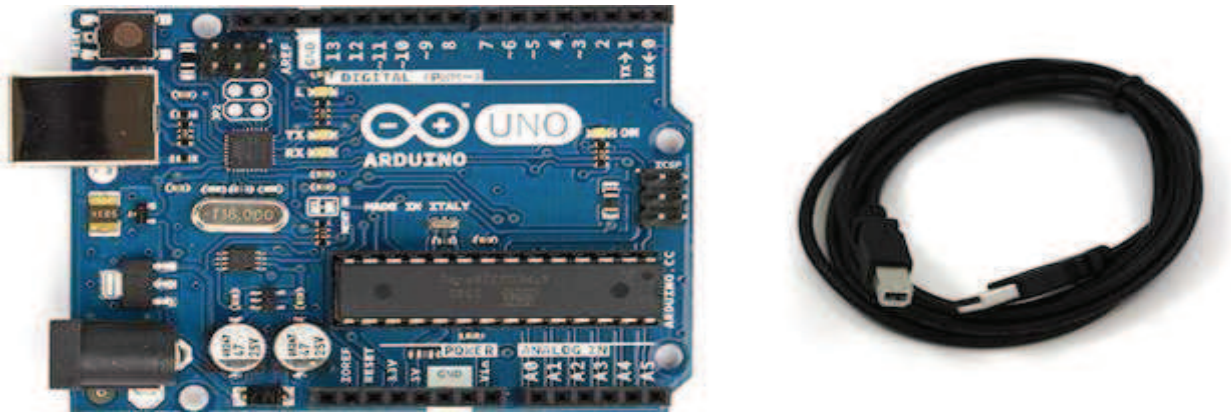


Figure-A.1-: Arduino board and USB

2 | Download the Arduino environment

Get the latest version from the official site [4 of chapter III].

When the download finishes, unzip the downloaded file. Make sure to preserve the folder structure. Double-click the folder to open it. There should be a few files and sub-folders inside.

3 | Connect the board

The Arduino Uno, Mega, Duemilanove and Arduino Nano automatically draw power from either the USB connection to the computer or an external power supply.

Connect the Arduino board to your computer using the USB cable. The green power LED (labelled PWR) should go on.

4 | Install the drivers

Installing drivers for the Arduino Uno with Windows 7, Vista, or XP:

Plug in your board and wait for Windows to begin it's driver installation process. After a few moments, the process will fail, despite its best efforts

Click on the Start Menu, and open up the Control Panel.

While in the Control Panel, navigate to System and Security. Next, click on System. Once the System window is up, open the Device Manager.

Look under Ports (COM & LPT). You should see an open port named "Arduino UNO (COMxx)". If there is no COM & LPT section, look under "Other Devices" for "Unknown Device".

Right click on the "Arduino UNO (COmxx)" port and choose the "Update Driver Software" option.

Next, choose the "Browse my computer for Driver software" option.

Finally, navigate to and select the driver file named "arduino.inf", located in the "Drivers" folder of the Arduino Software download (not the "FTDI USB Drivers" sub-directory). If you are using an old version of the IDE (1.0.3 or older), choose the Uno driver file named "Arduino UNO.inf"

Windows will finish up the driver installation from there.

5 | Launch the Arduino application

Double-click the Arduino application icon on desktop. Then a window should open as follows (see figure-A.2-):

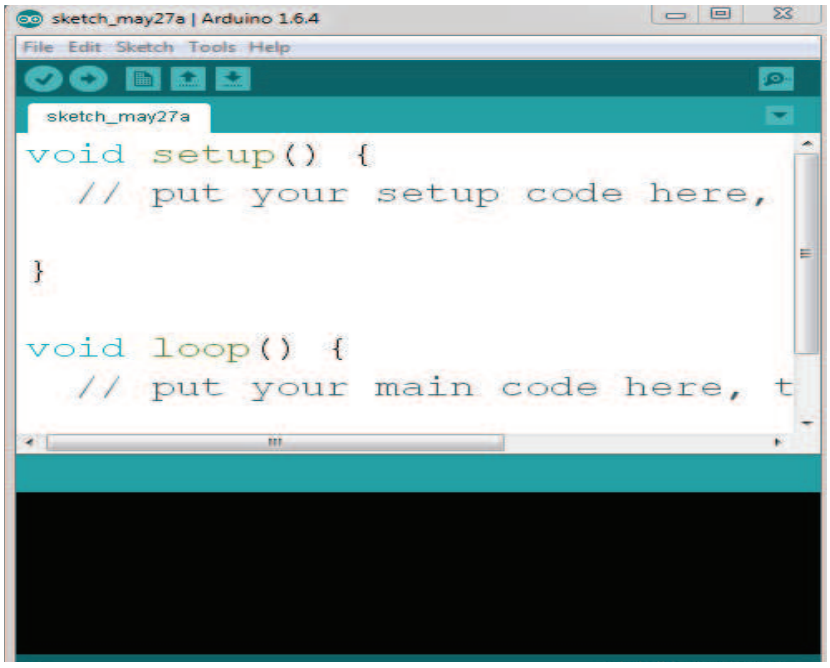


Figure-A.2-: Arduino main window

6 | ***Open the blink example***

Open the LED blink example sketch: File > Examples > 1.Basics > Blink. As shown in Figure-A.3-:

Figure-A.3-: Openeing the blink example

7 | ***Select your board:***

You'll need to select the entry in the Tools > Board menu that corresponds to your Arduino. (Arduino Uno in our case). (see Figure-A.4-):

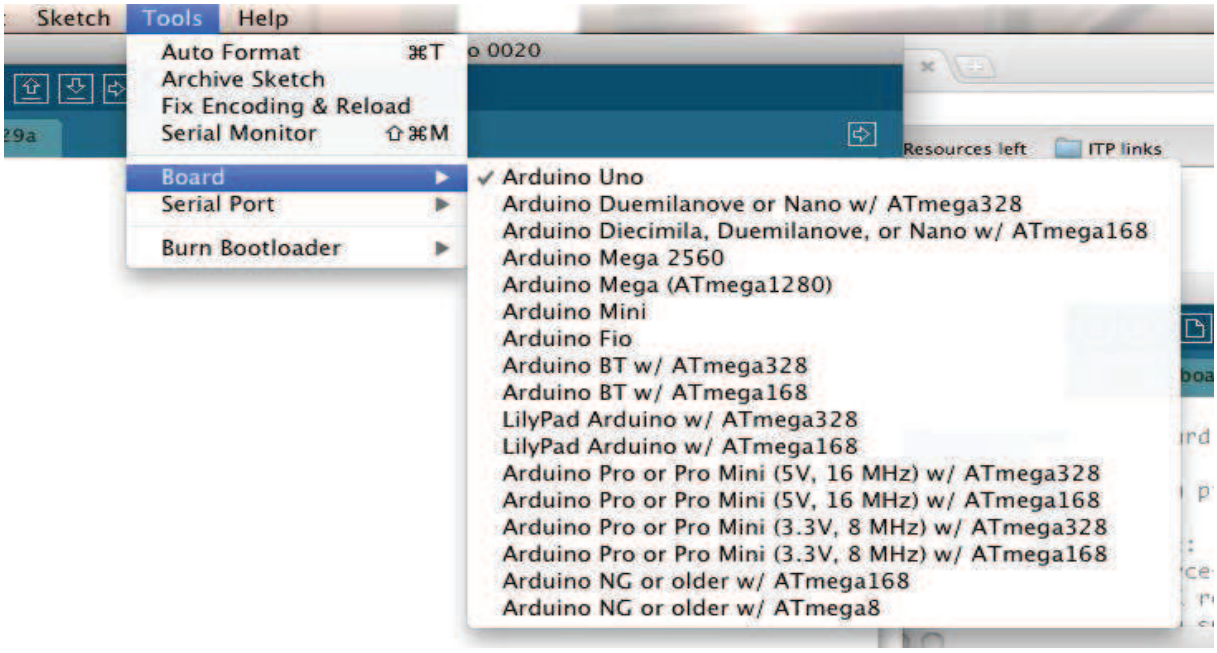


Figure-A.4-: Selecting the board

Then Selecting an Arduino Uno

8 | ***Select your serial port:***

Select the serial device of the Arduino board from the Tools | Serial Port menu. This is likely to be COM3 or higher (COM1and COM2 are usually reserved for hardware serial ports). To find out, you can disconnect your Arduino board and re-open the menu; the entry that disappears should be the Arduino board. Reconnect the board and select that serial port.

9 | ***Upload the program:***

Now, simply click the "Upload" button in the environment circled by the red circle shown in figure-A.5-. Wait a few seconds - you should see the RX and TX leds on the board flashing. If the upload is successful, the message "Done uploading." will appear in the status bar.

Figure-A.5-: uploading the program

A few seconds after the upload finishes, you should see the pin 13 (L) LED on the board start to blink (in orange). If it does, congratulations! You've gotten Arduino up-and-running.

For further reading about the Arduino functions and examples see [1&2 of chapter IV].

A.2) ***The Atmega microcontroller:***

The ATmega internal architecture and CPU Core are shown below in Figure-A.6-, figure-:

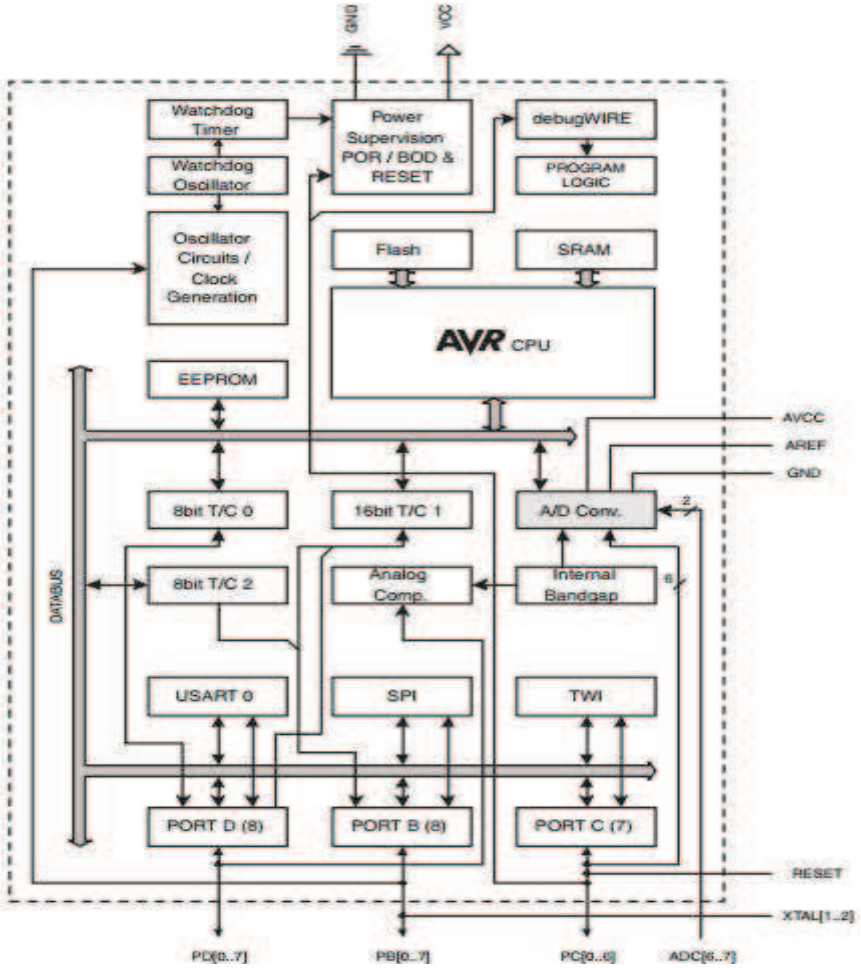


Figure-A.6a-: Internal architecture of ATmega328 microcontroller

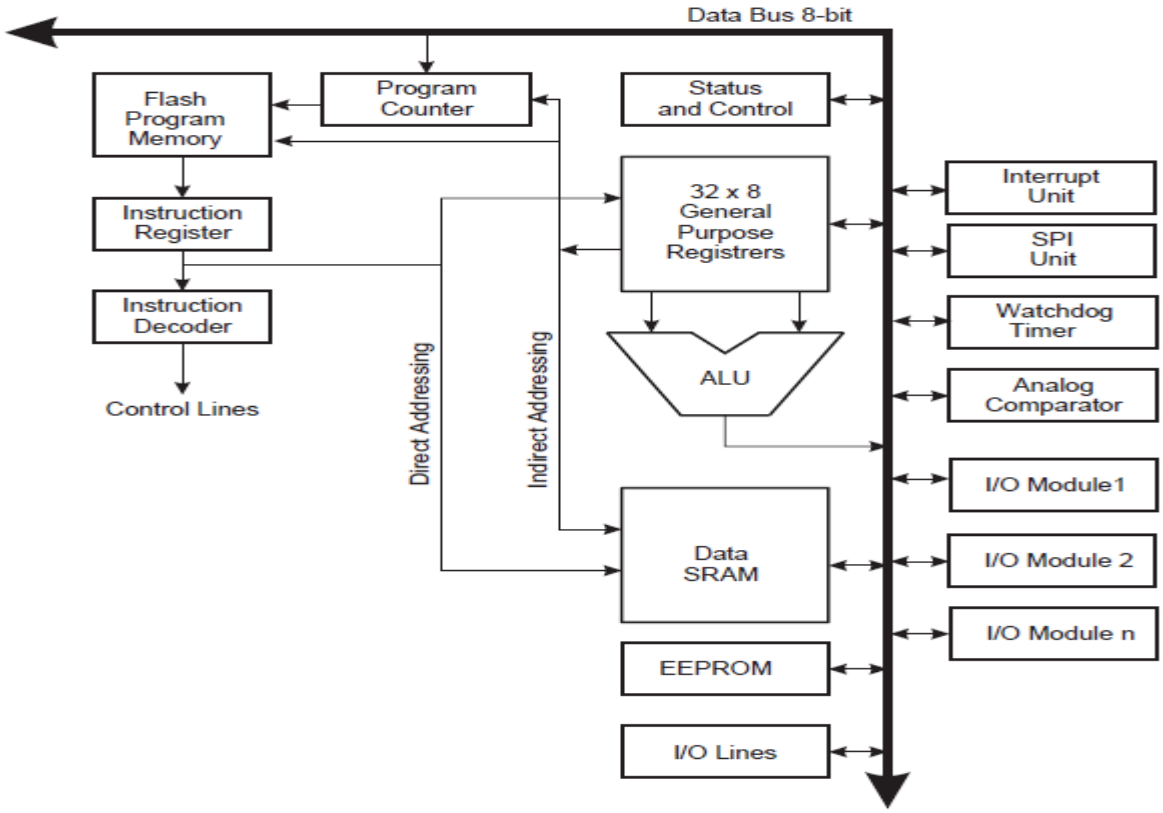


Figure-A.6b-: The ATmega CPU core architecture

B) ***About The Xbee:***

B.1) ***The Xbee configuration methods:***

We can configure a module either by using XCTU or Arduino UNO.

B.1.1) ***Using XCTU:***

B.1.1.1) ***Connecting an xbee to computer:***

In order to connect an xbee to computer we have to use either:

- Xbee shield.
- Xbee expolerer.
- Or Arduino as serial port.

We have used the third case (Arduino as serial port) see figure-B.1-.

The steps to connect an Xbee using arduino to computer are as follows:

1. First plug the Arduino to computer alone.
2. Then bypass the microcontroller, this can be done either:
    - By Removing the microcontroller from Arduino Uno.
    - Or short-circuit the Arduino by connecting a wire between RST and GND.
    - Or flashing the Arduino by uploading an empty program (just the two basic functions).

We recommend to use the third suggestion, i.e flashing the Arduino board bay an empty program.

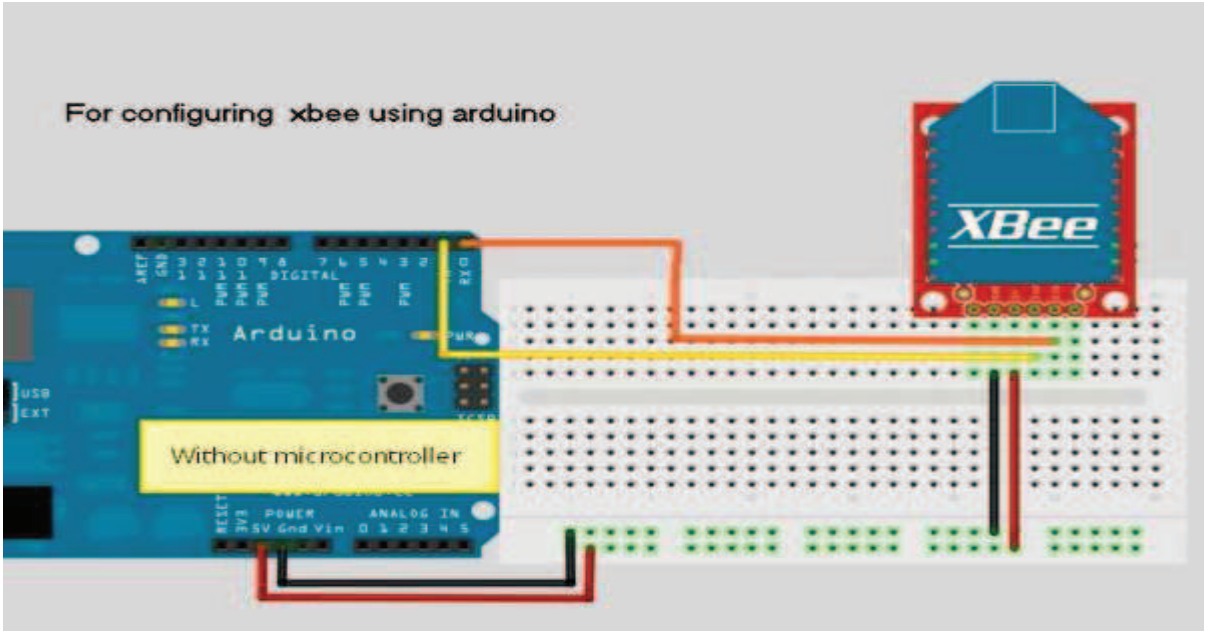


Figure-B.1: Xbee to computer connection using Arduino

3. Next disconnect the Arduino from computer and connect the xbee to it as follows (see Fig-C.7):

- Xbee pin1 (3.3) to arduino 3.3
- Xbee pin10 (GND) to arduino GND.
- Xbee pin2 (Dout) to Arduino TX.
- Xbee pin3 (Din) to Arduino Rx.

This pin assignments is available just when connecing an xbee to computer using arduino, ones finished with configuration they have to connected as shown in Figure-3.16a-

4. Finally plug the Arduino into the computer. The next step is to read the xbee using XCTU.

B.1.1.2) ***Exploring The XCTU:***

a) ***Starting With X-CTU:***

First, download the software at [3 of chapteIV], ones it's done launch the application, a window shown just below in figure-B.2- should appear.

b) Adding Xbee modules:

Before continuing on, make sure you've connected an XBee (correctly) to your computer by following the steps given in (section B.1.1.1).

To add your XBee, **click the "Add device" icon** –  – in the upper-left part of the window. That will prompt the screen shown in figure-B.3-:

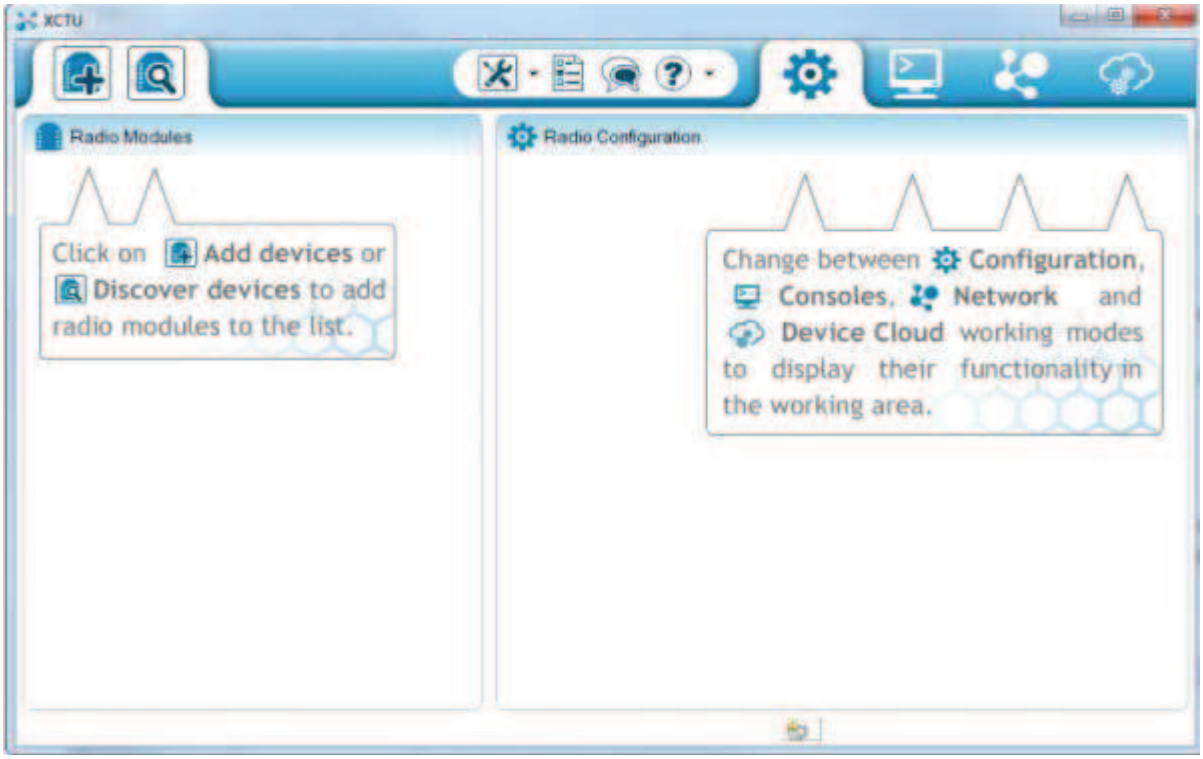


Figure-B.2-: The XCTU main page

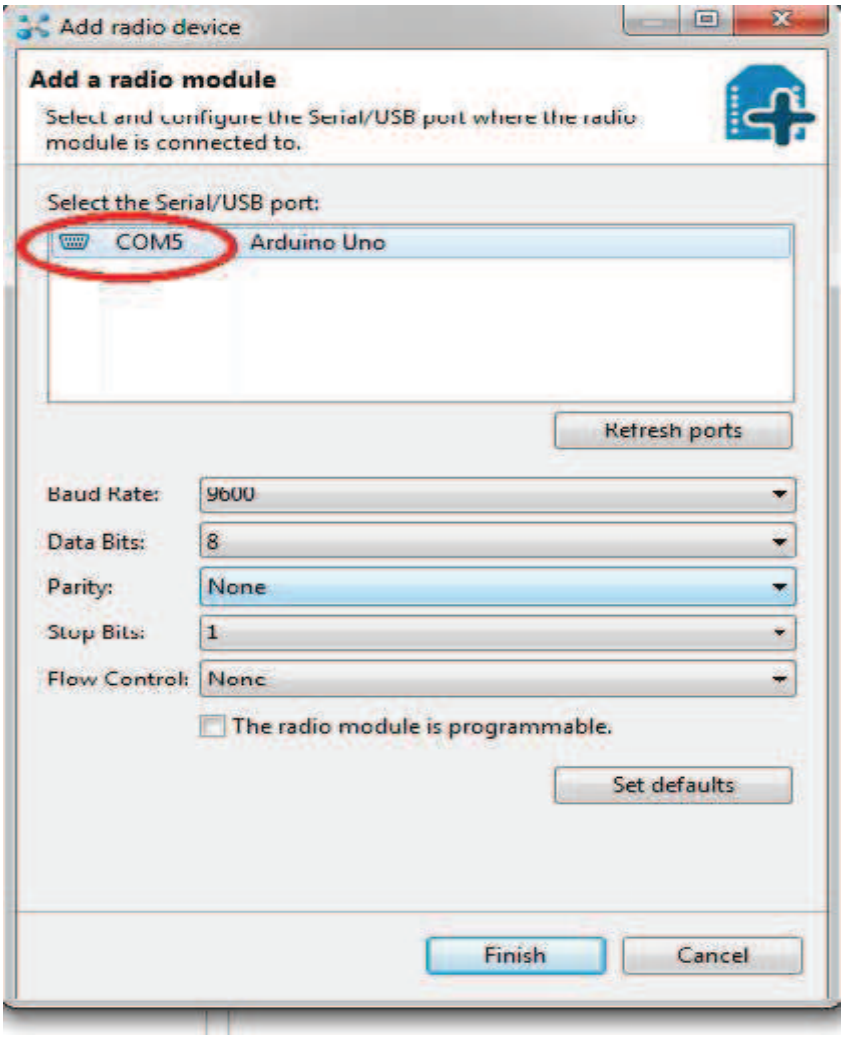


Figure-B.3-: Add radio device configuration

We can see that the first xbee is in COM5 circled in red.

Select the communication **port for configuration (COM5)**. This window also allows us to specify more specific serial characteristics like baud rate, data bits, and stop bits. Verify that the baud and parity settings of the Com Port match those of the RF module. Then **click Finish**

A "**Discovering radio modules**..." window shown with the green box will appear, click on that module then wait few seconds that XCTU reads its configurations. The default settings are shown by the yellow circle in figure-B.4-. The red one indicates that it's at COM5.

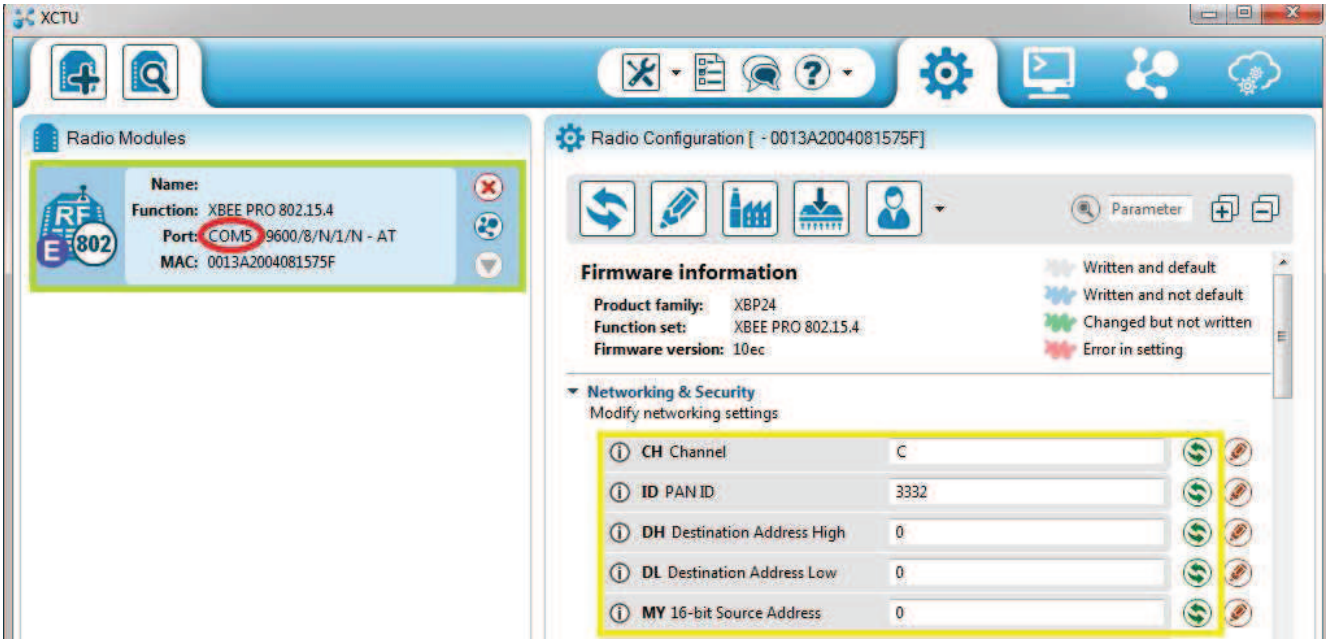


Figure-B.4-: Add and read a module

As you can see by scrolling down the right half, there are *a lot* of configuration settings available. For our case we are interested (channel=C, PAN ID=3332, DH=0, DL=0, MY=0). Which is the standard configuration.

c) Do it again:

To test communication between the two XBees, you'll need to connect your second XBee to a computer as well. That means doing steps (a,b) one more time.

If you have another computer available, you can install X-CTU on that as well and perform the same set up. You can certainly perform this test with both XBees connected to the same computer as well, just make sure you select the correct port number when you're adding the second XBee as shown in figure-B.5-.

If you add a second XBee to the same computer, a second entry will be added to the "**Radio Modules**" list. Selecting either of those entries will show the configuration settings for that, specific XBee.

Ones you have added another module, you'll be printed by the same figure as "B.3" with two ports (COM5 and COM6) (see figure-B.5-). The COM6 belongs to the second that you have added.

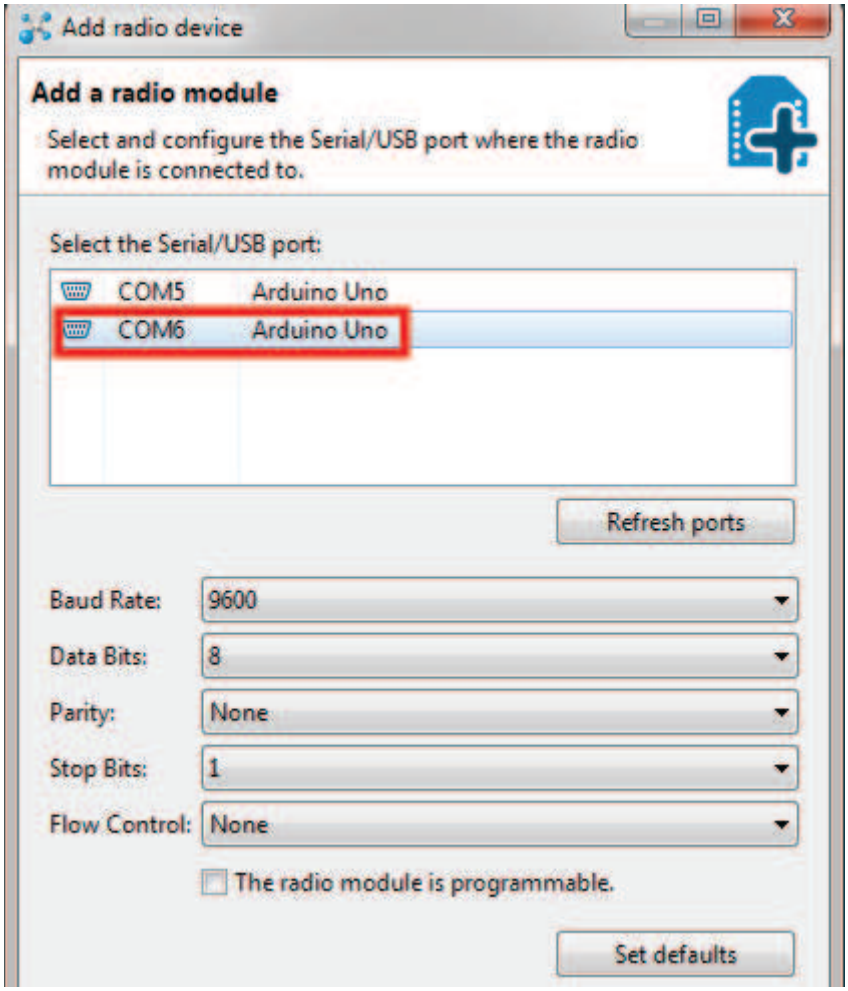


Figure-B.5-: Adding a second module

The red bex shown in figure-C.11- indicates that we are dealing with COM6 of a second module which is connected to computer via Arduino Uno.

After clicking on that new module the screen given by figure-B.6 will be printed:

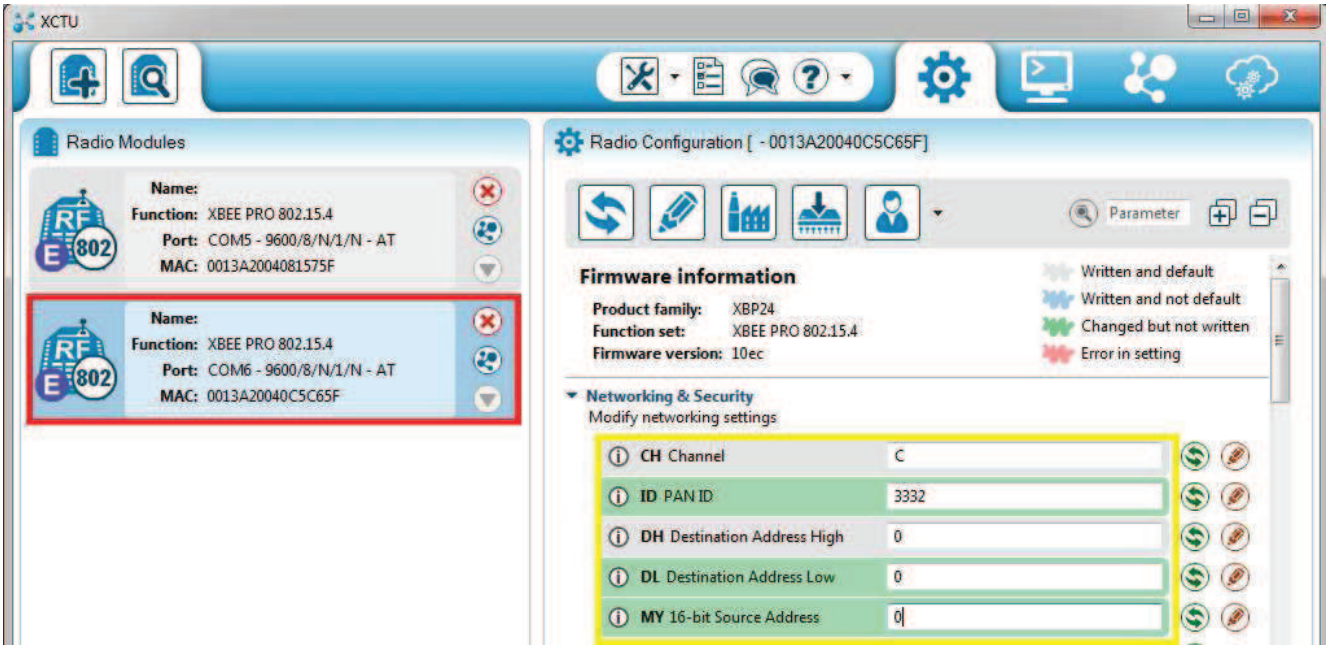


Figure-B.6-: Reading the second module

d) Quick test:

Click the "Switch to Consoles" icon –  – in the upper-right part of the window. This will switch from the configuration tab to the console. We can use the console to send characters to an XBee, which will route that character over-the-air to any other XBee connected to.

If you have two XBees connected to your computer, you can switch between each radio's console by selecting the device on the left.

First, **open a serial connection** on each device by clicking the connect icon –  . The icon will change, and the border of the console will turn green as shown in figure-B.6-.

Next, click into the **left half** of the console, and **type a letter or number**. You should notice that character echoed in a blue font (the hexadecimal digits on the right represent the ASCII value). As shown in figure-B.7-.

Now click into the other XBee's console. As long as it was open, you should see that same character, but ***red*** *(see figure-B.7-.)*. Try typing a different character into the second XBee's console, and you should see it work the other way. Then the modules are communicating between each other by default.

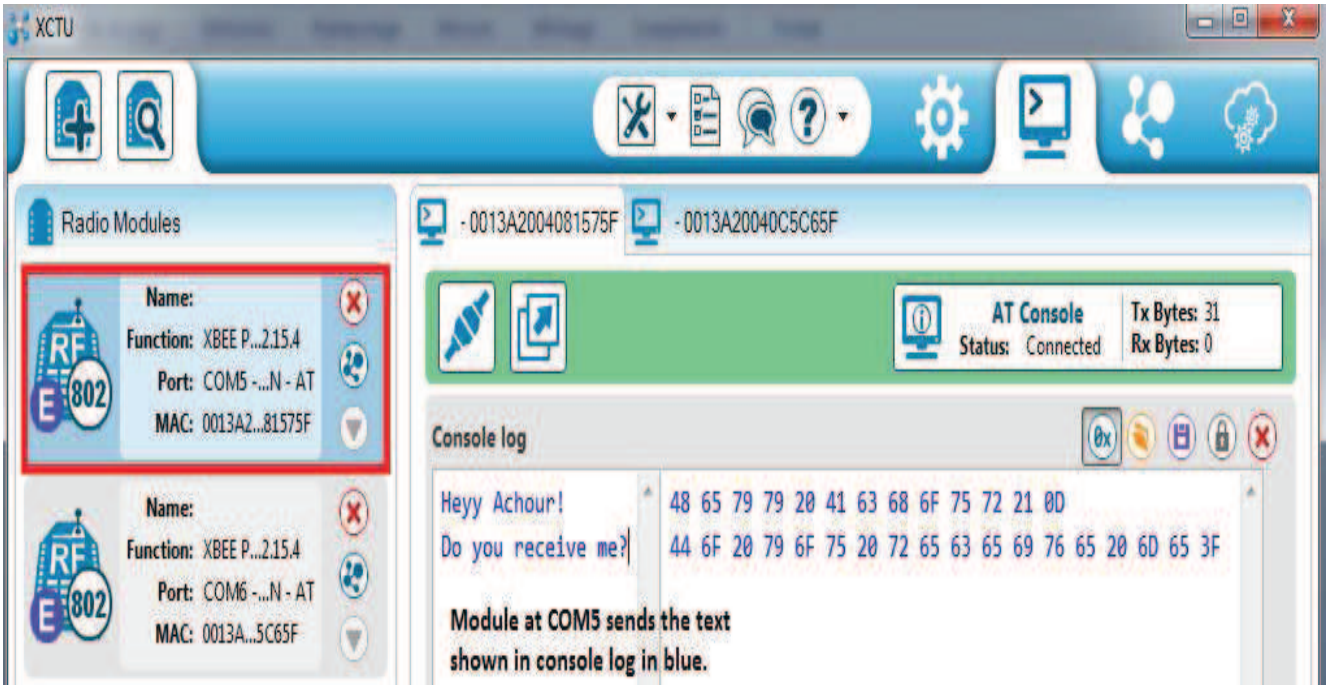


Figure-B.7a-: Module at COM5 sends the text in blue to COM6

Figure-B.7b-: Module at COM6 receives the text in Red from COM5

Note that the operation is vice versa, i.e we can send from COM6 to COM5. This is a two way communication by default.

Now, you can test the configurations given in tables (4.1-4.2-4.3) to see clearly the difference.

Note that for every change made you have to write the new parameters by clicking on the ic located at most right edge of the window.

See [8 of chapter III] for remote configurations commands that are another of configuring xbee modules. And for further configurations.

B.1.2) ***Using Arduino Uno:***

For addition the modules can be configured using Arduino, in this case we don't need XCTU. In order to so; there are special commands to talk with the xbee connected as seen in (section B1.1.1).

To get the module into configuration mode, you need to send it three plus signs: +++ via serial monitor and there needs to be at least one second before and after during which you send no other character to the module. Note that this includes newlines or carriage return characters. Thus, if you're trying to configure the module from the computer, you need to make sure your terminal software is configured to send characters as you type them, without waiting for you to press enter. Otherwise, it will send the plus signs immediately followed by a newline (i.e. you won't get the needed one second delay after the +++). If you successfully enter configuration mode, the module will send back the two characters 'OK', followed by a carriage return.

Send Command	Expected Response
+++	OK<CR>

Once in configuration mode, you can send AT commands to the module. Command strings have the form **ATxx** (where ***xx*** is the name of a setting). To read the current value of the setting, send the command string followed by a carriage return. To write a new value to the setting, send the command string, immediately followed by the new setting (with no spaces or newlines in-between), followed by a carriage return. For example, to read the network ID of the module (which determines which other Xbee modules it will communicate with), use the *'ATID* command:

Send Command	Expected Response
ATID<*enter*>	3332<*CR*>

To change the network ID of the module:

Send Command	Expected Response
ATID3331<*enter*>	OK<*CR*>

Now, check that the setting has taken effect:

Send Command	Expected Response
ATID<*enter*>	3331<*CR*>

Unless you tell the module to write the changes to non-volatile (long-term) memory, they will only be in effect until the module loses power. To save the changes permanently (until you explicitly modify them again), use the ATWR command:

Send Command	Expected Response
ATWR<*enter*>	OK<*CR*>

To reset the module to the factory settings, use the ATRE command:

Send Command	Expected Response
ATRE<*enter*>	OK<*CR*>

See ([8] of chapter III and ([4] to chapter IV) for more details.