```
A. RPC
#use for RPC communication to control station
from SerialM8 import SerialM8
from Cmps03 import Cmps03
import xmlrpclib
import SimpleXMLRPCServer
import string,socket
import time

left=1
right=1
accessList=('10.10.1.27','10.14.1.3')#ip address who can access UGV

class Server(SimpleXMLRPCServer.SimpleXMLRPCServer):
    def __init__(self,*args):
        SimpleXMLRPCServer.SimpleXMLRPCServer.__init__(self,(args[0],args[1]))

    def server_bind(self):
        self.socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
        SimpleXMLRPCServer.SimpleXMLRPCServer.server_bind(self)

    def verify_request(self,request, client_address):
        if client_address[0] in accessList:
            return 1
        else:
            return 0
class learningdata:
    def __init__(self):
        pass
    def manual(self,pwml,pwmr,left,right):
        Con=SerialM8()
        command="$master,%s,%s,%i,%i" %(pwml,pwmr,left,right)
        dataSerial=Con.SerialSend(command)
        print dataSerial
        self.Magnetometer=Cmps03()
        MagnetometerHeading=self.Magnetometer.data()
        return time.strftime("%Y:%m:%d:%H:%M:%S", time.gmtime()),MagnetometerH
eading,dataSerial


if __name__ == "__main__":
    server = Server('',8000)
    print "Listen port 8000"
    server.register_instance(learningdata())
    server.serve_forever()


B.Autonomous
'''
autonomous use for autonoumous mode
generate path planner and execute it use calss Navigation
class autonomous bind in RPCserver
'''
import re

from Navigation import Navigation

class Autonomous:
    def __init__(self,waypoint):
        self.waypoint=waypoint
        self.Nav=Navigation()
        self.wp=re.waypoint(",")
        self.pathplanning=[]
    def auto(self):
        AutonomousNavigation=Navigation()
        #add waypoint to execute in navigation class using fuzzy logic algorithm
        #self.wp(0)=waypoint start self.wp(1)=second waypoint self.wp(2)=final waypoint
        AutonomousNavigation.addwaypoint(self.wp(0),self.wp(1),self(2))#goto final waypoint and send data in stream over rpc



C.Navigation

'''
autonomous use for autonoumous mode
generate path planner and execute it use calss Navigation
class autonomous bind in RPCserver
'''
import re

from Navigation import Navigation

class Autonomous:
    def __init__(self,waypoint):
        self.waypoint=waypoint
        self.Nav=Navigation()
        self.wp=re.waypoint(",")
        self.pathplanning=[]
    def auto(self):
        AutonomousNavigation=Navigation()
        #add waypoint to execute in navigation class using fuzzy logic algorithm
        #self.wp(0)=waypoint start self.wp(1)=second waypoint self.wp(2)=final waypoint
        AutonomousNavigation.addwaypoint(self.wp(0),self.wp(1),self(2))#goto final waypoint and send data in stream over rpc



D.Controler

#/bin/python
'''
@author : estheim
Controler for mode manual and auto
make a command to send use SerialM8 module

'''
from SerialM8 import SerialM8
import time

class Controler:
    def __init__(self,mode):
        self.mode = mode
        self.master = SerialM8()
        self.dataSerial=[]
    def manual(self,pwml,pwmr,left,right):
        if self.mode =="manual":
            if pwml>255:
                pwml=255
            if pwmr>255:
                pwmr=255
            if pwml<200:
                pwml=200
            if pwmr<200:
                pwmr=200
            command = "$master,%s,%s,%i,%i" %(pwml,pwmr,left,right)
            self.dataSerial=self.master.SerialSend(command)
            #print time.strftime("%Y-%m-%d %H:%M:%S", time.gmtime()),self.data
Serial
            return self.dataSerial
        else:
            self.dataSerial=['','','']
            return self.dataSerial


if __name__=="__main__":
    pass

E.GPS

#! /usr/bin/python
# Written by Dan Mandle http://dan.mandle.me September 2012
# License: GPL 2.0
# edited by estheim telkom institute teknologi

import os
from gps import *
from time import *
import time
import threading

GpsData = None #seting the global variable

class GpsPoller(threading.Thread):
    def __init__(self):
        threading.Thread.__init__(self)
        global gpsd #bring it in scope
        gpsd = gps(mode=WATCH_ENABLE) #starting the stream of info
        self.current_value = None
        self.running = True #setting the thread running to true
    def run(self):
        global gpsd
        while self.running: #cek thread true
            gpsd.next()  #this will continue to loop
    def status(self):
        #return nilai
        if (len(gpsd.satellites) > 4):
            return True
        else:
            return False
    def data(self):
        #data GPS utc, latitude, longitude, altitude, speed, track, fix mode, len satellites
        return gpsd.utc,gpsd.fix.latitude,gpsd.fix.longitude,gpsd.fix.altitude, gpsd.fix.speed,gpsd.fix.track, gpsd.fix.mode, len(gpsd.satellites)

F. Incremental Encoder

import re

class SensorEncoder:
    def __init__(self):
        self.sumencoderL=0
        self.sumencoderR=0

    def data(self,dataserial):
        #print enc
        listcount=re.split(",",dataserial)
        if listcount[0]=="$counter":
            self.encoderL=listcount[1]
            self.encoderR=listcount[2].strip() #bersihkan data menghilangkan komponen \r\n dibagian akhir
            self.sumencoderL+=int(self.encoderL)
            self.sumencoderR+=int(self.encoderR)
            return [self.sumencoderL,self.sumencoderR]
        else:
            return [0,0]
    def status(self):
        pass
'''
if __name__=="__main__":
    dataserial=["$master,255,255,1,1","$slave,255,255","$counter,123,222"]
    Sen=SensorEncoder()
    data=Sen.data(dataserial)
    print data
    data=Sen.data(dataserial)
    print data
'''

G.SerialM8

#/bin/python
import serial
import re
import time

class SerialM8:
    def __init__(self):
        self.ser=serial.Serial('/dev/ttyAMA0',9600)
        self.lsdataserial=[]

    def SerialSend(self,command):
        self.ser=serial.Serial('/dev/ttyAMA0',9600)
        #print command
        command = ("%s\r\n") %(command)
        self.ser.write(command)
        #self.ser.flushInput()
        for i in range(3):#get 3 line serial
```

```
            #print "kirim %i" % (i)
            temp = self.ser.readline()
            #print temp
            self.lsdataserial.append(temp.strip()) #append tambah komponen list strip hilangkan \r\n di tiap akhir
        #print self.lsdataserial
        return self.lsdataserial
        self.ser.close() #stop serial
'''
if __name__=="__main__":
  while True:
    #print("begin")
    controler=SerialM8()
    #print "send serial"
    dataserial=controler.SerialSend("$master,123,123,1,1")
    print dataserial
    time.sleep(0.1)
'''

```