

```

1 A. RPC
2 #use for RPC communication to control station
3 from SerialM8 import SerialM8
4 from Cmps03 import Cmps03
5 import xmlrpclib
6 import SimpleXMLRPCServer
7 import string,socket
8 import time
9
10 left=1
11 right=1
12 accessList=('10.10.1.27','10.14.1.3')#ip address who can access UGV
13
14 class Server(SimpleXMLRPCServer.SimpleXMLRPCServer):
15     def __init__(self,*args):
16         SimpleXMLRPCServer.SimpleXMLRPCServer.__init__(self,(args[0],args[1]))
17
18     def server_bind(self):
19         self.socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
20         SimpleXMLRPCServer.SimpleXMLRPCServer.server_bind(self)
21
22     def verify_request(self,request, client_address):
23         if client_address[0] in accessList:
24             return 1
25         else:
26             return 0
27 class learningdata:
28     def __init__(self):
29         pass
30     def manual(self,pwml,pwmr,left,right):
31         Con=SerialM8()
32         command="$master,%s,%s,%i,%i" %(pwml,pwmr,left,right)
33         dataSerial=Con.SerialSend(command)
34         print dataSerial
35         self.Magnetometer=Cmps03()
36         MagnetometerHeading=self.Magnetometer.data()
37         return time.strftime("%Y:%m:%d:%H:%M:%S", time.gmtime()),MagnetometerH
38 eading,dataSerial
39
40 if __name__ == "__main__":
41     server = Server('',8000)
42     print "Listen port 8000"
43     server.register_instance(learningdata())
44     server.serve_forever()
45
46
47 B.Autonomous
48 '''
49 autonomous use for autonomous mode
50 generate path planner and execute it use calss Navigation
51 class autonomous bind in RPCserver
52 '''
53 import re
54
55 from Navigation import Navigation
56
57 class Autonomous:

```

```

58     def __init__(self, waypoint):
59         self.waypoint=waypoint
60         self.Nav=Navigation()
61         self.wp=re.waypoint(",")
62         self.pathplanning=[]
63     def auto(self):
64         AutonomousNavigation=Navigation()
65         #add waypoint to execute in navigation class using fuzzy logic algorit
hm
66         #self.wp(0)=waypoint start self.wp(1)=second waypoint self.wp(2)=final
        waypoint
67         AutonomousNavigation.addwaypoint(self.wp(0),self.wp(1),self.wp(2))#goto f
        inal waypoint and send data in stream over rpc
68
69
70
71 C.Navigation
72
73 '''
74 autonomous use for autonomouous mode
75 generate path planner and execute it use calss Navigation
76 class autonomous bind in RPCserver
77 '''
78 import re
79
80 from Navigation import Navigation
81
82 class Autonomous:
83     def __init__(self, waypoint):
84         self.waypoint=waypoint
85         self.Nav=Navigation()
86         self.wp=re.waypoint(",")
87         self.pathplanning=[]
88     def auto(self):
89         AutonomousNavigation=Navigation()
90         #add waypoint to execute in navigation class using fuzzy logic algorit
hm
91         #self.wp(0)=waypoint start self.wp(1)=second waypoint self.wp(2)=final
        waypoint
92         AutonomousNavigation.addwaypoint(self.wp(0),self.wp(1),self.wp(2))#goto f
        inal waypoint and send data in stream over rpc
93
94
95
96 D.Controler
97
98 #/bin/python
99 '''
100 @author : estheim
101 Controler for mode manual and auto
102 make a command to send use SerialM8 module
103
104 '''
105 from SerialM8 import SerialM8
106 import time
107
108 class Controler:
109     def __init__(self, mode):

```

```

110         self.mode = mode
111         self.master = SerialM8()
112         self.dataSerial=[]
113     def manual(self,pwml,pwmr,left,right):
114         if self.mode == "manual":
115             if pwml>255:
116                 pwml=255
117             if pwmr>255:
118                 pwmr=255
119             if pwml<200:
120                 pwml=200
121             if pwmr<200:
122                 pwmr=200
123             command = "$master,%s,%s,%i,%i" %(pwml,pwmr,left,right)
124             self.dataSerial=self.master.SerialSend(command)
125             #print time.strftime("%Y-%m-%d %H:%M:%S", time.gmtime()),self.data
Serial
126         return self.dataSerial
127     else:
128         self.dataSerial=['',' ',' ']
129         return self.dataSerial
130
131
132 if __name__=="__main__":
133     pass
134
135 E.GPS
136
137 #!/usr/bin/python
138 # Written by Dan Mandle http://dan.mandle.me September 2012
139 # License: GPL 2.0
140 # edited by estheim telkom institute teknologi
141
142 import os
143 from gps import *
144 from time import *
145 import time
146 import threading
147
148 GpsData = None #setting the global variable
149
150 class GpsPoller(threading.Thread):
151     def __init__(self):
152         threading.Thread.__init__(self)
153         global gpssd #bring it in scope
154         gpssd = gps(mode=WATCH_ENABLE) #starting the stream of info
155         self.current_value = None
156         self.running = True #setting the thread running to true
157     def run(self):
158         global gpssd
159         while self.running: #cek thread true
160             gpssd.next() #this will continue to loop
161     def status(self):
162         #return nilai
163         if (len(gpssd.satellites) > 4):
164             return True
165         else:
166             return False

```

```

167     def data(self):
168         #data GPS utc, latitude, longitude, altitude, speed, track, fix mode,
len satellites
169         return gpsd.utc,gpsd.fix.latitude,gpsd.fix.longitude,gpsd.fix.altitude
, gpsd.fix.speed,gpsd.fix.track, gpsd.fix.mode, len(gpsd.satellites)
170
171 F. Incremental Encoder
172
173 import re
174
175 class SensorEncoder:
176     def __init__(self):
177         self.sumencoderL=0
178         self.sumencoderR=0
179
180     def data(self,dataserial):
181         #print enc
182         listcount=re.split(",",dataserial)
183         if listcount[0]=="$counter":
184             self.encoderL=listcount[1]
185             self.encoderR=listcount[2].strip() #bersihkan data menghilangkan k
omponen \r\n dibagian akhir
186             self.sumencoderL+=int(self.encoderL)
187             self.sumencoderR+=int(self.encoderR)
188             return [self.sumencoderL,self.sumencoderR]
189         else:
190             return [0,0]
191     def status(self):
192         pass
193
194 if __name__=="__main__":
195     dataserial=["$master,255,255,1,1","$slave,255,255","$counter,123,222"]
196     Sen=SensorEncoder()
197     data=Sen.data(dataserial)
198     print data
199     data=Sen.data(dataserial)
200     print data
201
202
203 G.SerialM8
204
205 #/bin/python
206 import serial
207 import re
208 import time
209
210 class SerialM8:
211     def __init__(self):
212         self.ser=serial.Serial('/dev/ttyAMA0',9600)
213         self.lsdataserial=[]
214
215     def SerialSend(self,command):
216         self.ser=serial.Serial('/dev/ttyAMA0',9600)
217         #print command
218         command = ("%s\r\n") %(command)
219         self.ser.write(command)
220         #self.ser.flushInput()
221         for i in range(3):#get 3 line serial

```

```
222         #print "kirim %i" % (i)
223         temp = self.ser.readline()
224         #print temp
225         self.lsdataserial.append(temp.strip()) #append tambah komponen lis
226     t strip hilangkan \r\n di tiap akhir
227     #print self.lsdataserial
228     return self.lsdataserial
229     self.ser.close() #stop serial
230 '''
231 if __name__=="__main__":
232     while True:
233         #print("begin")
234         controler=SerialM8()
235         #print "send serial"
236         dataserial=controler.SerialSend("$master,123,123,1,1")
237         print dataserial
238         time.sleep(0.1)
239     '''
```