```
 1 from VideoStream import videoStream
 2
 3 from modbusTcp import startModbusDataThreads
 4 from modbusWriter import modbusClient
 5 from SerialReadGyro import SerialReadGyro
 6 import threading
 7 import serial
 8
 9
10 def main():
11     modbusIpadress = "192.168.0.112"
12     t = threading.Thread(target=videoStream,name="
   VidThread",args=("192.168.0.101",12345))
13     t.start()
14     #startModbusDataThreads(modbusIpadress)
15     modclient=modbusClient(modbusIpadress)
16     modclient.start()
17
18
19
20
21
22
23
24 if __name__ == '__main__':main()
```

```
 1 import gps
 2 from threading import Thread
 3 import queue
 4
 5 class gpsReader():
 6
 7     def __init__(self):
 8
 9
10         # Gps data variables
11         self.GpsTime= "asa"
12         self.Speed=0
13         self.Latitude=0
14         self.Heading=0
15         self.Longitude=0
16         self.NumberOfsatelites=0
17         #queue for sharing data betweem thrread
18         self.q = queue.LifoQueue()
19         #need to implement
20
21         # Listen on port 2947 (gpsd) of localhost
22         self.session = gps.gps("localhost", "2947")
23         self.session.stream(gps.WATCH_ENABLE | gps.
   WATCH_NEWSTYLE)
24
25     def start(self):
26
27         t = Thread(target=self.run, name="gpsReaderThread"
   , args=())
28         t.daemon = True
29         t.start()
30
31
32
33     # outQueue.put(0)
34     def run(self):
35         try:
36             while True:
37
38
39                 self.NumberOfsatelites=self.session.
   satellites_used
40                 report = self.session.next()
41                 #print("Number of satelites :" + str(self.
   NumberOfsatelites))
```

```
42
43
44                  # Wait for a 'TPV' report from gpsd
45                  # To see all report data, uncomment the
   line below
46                  # print(report)
47                  if report['class'] == 'TPV':
48                      if hasattr(report, 'time'):
49                          #print("time :" + report.time)
50                          self.GpsTime = report.time
51
52                      if hasattr(report, 'track'):
53                          #print("Heading :" + str(report.
   track))
54                          self.Heading = report.track
55                      if hasattr(report, 'lon'):
56                          #print("Langitude :" + str(report.
   lon))
57                          self.Longitude = report.lon
58
59                      if hasattr(report, 'lat'):
60                          #print("Latitude :" + str(report.
   lat))
61                          self.Latitude = report.lat
62                      if hasattr(report, 'speed'):
63                          self.Speed = (report.speed * gps.
   MPS_TO_KPH)
64
65                      gpsDataArray=[self.GpsTime,self.
   Heading,self.Longitude,
66                                    self.Latitude,self.Speed
   ,self.NumberOfsatelites]
67                      self.q.put(gpsDataArray)
68
69          except KeyError:
70              pass
71          except KeyboardInterrupt:
72              quit()
73          except StopIteration:
74              session = None
75              print("GPSD has terminated")
76
77      def getGpsData(self):
78
79          # Return the latest gpsdata:arrray[time,heading,
```

```
79 long,lat,speed]
80         if not self.q.empty():
81             newdata = self.q.get()
82 
83             while not self.q.empty():
84                 trashBin = self.q.get()
85 
86             return newdata
87         else:
88             # print("empty Queue in seiralRead")
89             noData = [None] * 6
90             return noData
91 
92     def queReady(self):
93         if not self.q.empty():
94             return True
95         else:
96             return False
```

```
 1
 2 import time
 3 import socket
 4 import cv2
 5
 6
 7 def videoStream(ipadress,port):
 8
 9
10     cam = cv2.VideoCapture(0)
11     cam.set(cv2.CAP_PROP_FPS,30)
12     UDP_IP = ipadress
13     UDP_PORT = port
14     sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM
   )
15
16
17
18     # warmup camera
19     time.sleep(0.1)
20     while True:
21
22         ret_val, img = cam.read(0)
23
24         img = cv2.flip(img, 1)
25         img = cv2.resize(img, (320, 240))
26
27         x = [int(cv2.IMWRITE_JPEG_QUALITY), 80]
28         _, compressed = cv2.imencode(".jpg", img, x)
29         sock.sendto(compressed, (UDP_IP, UDP_PORT))
30
31
32     rawCapture.truncate(0)
33
34
35
36
37 #if __name__ == "__main__": main()
38
39
40
41
```

```
 1 import serial
 2 import time
 3 from pymodbus.client.sync import ModbusTcpClient
 4 from pymodbus.constants import Endian
 5 from pymodbus.exceptions import ConnectionException
 6 from pymodbus.payload import BinaryPayloadBuilder
 7 from threading import Thread
 8 from GpsReader import gpsReader
 9 from SerialReadGyro import SerialReadGyro
10
11
12 #The modbusclient reads the value from the GPS and the
   Gyro, and sends the data to the Wago PLC.
13 class modbusClient():
14
15     def __init__(self,iAdress):
16         self.client = ModbusTcpClient(iAdress)
17         self.builder = BinaryPayloadBuilder(byteorder=
   Endian.Big,
18                                             wordorder=Endian.
   Little)
19         self.gpsreader = gpsReader()
20         self.gyroreader = SerialReadGyro(serialPort=serial
   .Serial('/dev/ttyACM1', 19200, timeout=5))
21
22         self.gpsTime0 = "asa"
23         self.timeOutCheckTime = 2
24         self.gpsTimeOutTime = time.time()
25         self.GpsEnabled = True
26         self.connCheckTimer = time.time()
27         self.connCheckVar=True
28
29         ##GPSDATA#####
30         self.GpsTime = "asa"
31         self.GpsHeading = 0
32         self.GpsLong = 0
33         self.GpsLat = 0
34         self.GpsSpeed = 0
35         self.GpsNumberOfSat = 0
36
37         ##GYRODATA##
38         self.GyroHeading = 0
39         self.GyroPitch = 0
40         self.GyroRoll = 0
41
```

```
42          ##cameraServoController###
43          self.CameraRovPos=False
44          self.camCoilNr=0
45
46      def start(self):
47          self.gpsreader.start()
48          self.gyroreader.start()
49          t = Thread(target=self.run, name="ModbusThread",
    args=())
50          t.daemon = True
51          t.start()
52
53      # runns when called thread.start. reads data from gps
    and gyro
54      def run(self):
55
56          while True:
57
58              #checking if there is connection to the GPS by
     looking for change in the clock, Checks status each 2 sec
59              if self.gpsTimeOutTime + self.timeOutCheckTime
     < time.time():
60                  if self.GpsTime == self.gpsTime0 or self.
    GpsTime == None:
61                      self.GpsEnabled = False
62                  else:
63                      self.GpsEnabled = True
64                      self.gpsTime0 = self.GpsTime
65                      self.gpsTimeOutTime = time.time()
66                      # Gps enable flag on coil 32912
67                  self.client.write_coil(32768, [self.
    GpsEnabled] * 8, unit=1)
68                  print("GPS ENABLED: "+str(self.GpsEnabled)
    )
69
70              # Switching a bit for so the plc can notice
    connection loss. switching 1 time a sec.
71              if self.connCheckTimer + 1 < time.time():
72                  if self.connCheckVar:
73                      self.connCheckVar = False
74                  else:
75                      self.connCheckVar = True
76
77                  self.client.write_coils(33168, [self.
    connCheckVar] * 8, unit=1)
```

```
 78                 self.connCheckTimer = time.time()
 79
 80
 81
 82
 83
 84             # if the gpsReader thread is ready with new
    values then send to Modbus
 85             if self.gpsreader.queReady():
 86
 87                 self.updateLocalGPsData(self.gpsreader.
    getGpsData())
 88
 89                 # write modbus Number of satelites on
    register 32008 on wago plc
 90                 self.client.write_registers(32001, self.
    build16BitMessage(self.GpsNumberOfSat), unit=1)
 91
 92                 # write modbus Speed on register 32004 on
     wago plc
 93
 94                 self.client.write_registers(32002, self.
    build32BitMessage(self.GpsSpeed), unit=1)
 95
 96                 # write modbus Latitude on register 32012
     on wago plc
 97
 98                 self.client.write_registers(32004, self.
    build64BitMessage(self.GpsLat), unit=1)
 99
100                 # write modbus Heading on register 320012
     on wago plc
101
102                 self.client.write_registers(32012, self.
    build64BitMessage(self.GpsHeading), unit=1)
103
104                 # write modbus Longitude on register
    32008 on wago plc
105
106                 self.client.write_registers(32008, self.
    build64BitMessage(self.GpsLong), unit=1)
107
108                 #Change CameraPos
109                 self.client.read_coils(self.camCoilNr, 1,
     unit=1)
```

```
110
111             #if Gyro thread is ready with new data then
    send gyro data to plc
112             if self.gyroreader.queReady():
113                 try:
114                     self.updateLocalGyroData(self.
    gyroreader.getSerialData())
115
116                     # write modbus GyroHeading on
    register 32004 on wago plc
117
118                     self.client.write_registers(32025,
    self.build32BitMessage(self.GyroHeading), unit=1)
119
120                     # write modbus GyroPitch on register
    32004 on wago plc
121
122                     self.client.write_registers(32016,
    self.build32BitMessage(self.GyroPitch), unit=1)
123
124                     # write modbus GyroRoll on register
    32004 on wago plc
125
126                     self.client.write_registers(32020,
    self.build32BitMessage(self.GyroRoll), unit=1)
127                 except ConnectionException as e:
128                     print ("exception write register
    gyrodata")
129                     continue
130
131
132
133
134
135     def updateLocalGPsData(self,gpsData):
136         self.GpsTime = gpsData[0]
137         self.GpsHeading = gpsData[1]
138         self.GpsLong = gpsData[2]
139         self.GpsLat = gpsData[3]
140         self.GpsSpeed = gpsData[4]
141         self.GpsNumberOfSat = gpsData[5]
142
143     def updateLocalGyroData(self,gyroData):
144         self.GyroHeading = gyroData[0]
145         print("gyroHeading: "+ str(self.GyroHeading))
```

```
146         self.GyroPitch = gyroData[2]
147         print("gyroPitch: " + str(self.GyroPitch))
148         self.GyroRoll = gyroData[1]
149         print("gyroRoll: " + str(self.GyroRoll))
150
151
152     def build16BitMessage(self,message):
153         #encode 16 bit integer message
154         self.builder.reset()
155         self.builder.add_16bit_int(message)
156         encoded16 = self.builder.to_registers()
157         return encoded16
158
159
160     def build32BitMessage(self, message):
161         # encode 32 bit float message
162         self.builder.reset()
163         self.builder.add_32bit_float(message)
164         encoded32 = self.builder.to_registers()
165         return encoded32
166
167     def build64BitMessage(self, message):
168         # encode 64 bit float message
169         self.builder.reset()
170         self.builder.add_64bit_float(message)
171         encoded64 = self.builder.to_registers()
172         return encoded64
173
174
```

```
 1 import queue
 2 import serial
 3 from threading import Thread
 4 import time
 5
 6
 7 # Reads serialcom from the Gyroscope on the port "
   serialPort"
 8 class SerialReadGyro():
 9     def __init__(self, serialPort):
10
11         self.q = queue.LifoQueue()
12
13         self.serialPort = serialPort
14
15     def start(self):
16
17         t = Thread(target=self.run, name="GyroThread",
   args=())
18         t.daemon = True
19         t.start()
20
21     # runns when called thread.start. reads data from
   arduino and setts the input to the arranged value
22
23     def run(self):
24         try:
25             while (True):
26
27                 # print("starting to read")
28                 data = str(self.serialPort.readline())
29
30
31                 ##removing b' flag at the beginning of the
    string
32                 cleanedString = data.split("b'")
33                 #print(cleanedString)
34
35                 # splitting the string where there is ;
36                 splitData = cleanedString[1].split(";")
37                 cleanLastPart = splitData[2].split("\\")
38
39                 # list type=string [0]=gyro heading(yaw) [
   1] = roll [2]= pitch
40                 splitData[2] = cleanLastPart[0]
```

```
41
42                 if splitData.__len__() == 3:
43                     self.q.put(self.convertStringToFloat(
   splitData))
44
45
46
47
48         except serial.SerialException as e:
49             print("SerialPortException i SerialRead")
50
51     def convertStringToFloat(self, l):
52         # create a list with the size of 3 (0...2)&
   converts the string to float values
53         list = [None] * 3
54         list[0] = float(l[0])
55         list[1] = float(l[1])
56         list[2] = float(l[2])
57         return list
58
59     def getSerialData(self):
60
61         # Return the latest serialData
62         if not self.q.empty():
63             newdata = self.q.get()
64
65             # while not self.q.empty():
66             # trashBin = self.q.get()
67
68             return newdata
69         else:
70             # print("empty Queue in seiralRead")
71             noData = [None] * 3
72             return noData
73
74     def queReady(self):
75         if not self.q.empty():
76             return True
77         else:
78             return False
79
```