Supplemental Material A: Python Programming Code

```
# -*- coding: utf-8 -*-

import math
import string,os
import time
import urllib.request
import xml.dom.minidom as minidom
from urllib.parse import quote


class LocaDiv:

    def __init__(self,region,gap):
        self.region = region
        self.gap=gap

    def lat_all(self):
        lat_sw = float(self.region.split(',')[1])
        lat_ne = float(self.region.split(',')[3])
        lat_list = []
        for i in range(0,math.ceil((lat_ne-lat_sw)/self.gap)):
            lat_list.append(lat_sw + self.gap * i)
        lat_list.append(lat_ne)
        return lat_list

    def lng_all(self):
        lng_sw = float(self.region.split(',')[0])
        lng_ne = float(self.region.split(',')[2])
        lng_list = []
        for i in range(0,math.ceil((lng_ne-lng_sw)/self.gap)):
            lng_list.append(lng_sw+self.gap*i)
        lng_list.append(lng_ne)
        return lng_list

    def ls_row(self):
        l1 = self.lat_all()
        l2 = self.lng_all()
        ls = []
        for i in range(len(l1)-1):
            for j in range(len(l2)-1):
                t=str(l2[j])+','+str(l1[i])+'|'+str(l2[j+1])+','+str(l1[i+1])
                ls.append(t)
        return ls

class GetPOI:
    def __init__(self,url,limit):
        self.url = url
        self.limit=limit
    #write logs
    def log2file(file_handle,text_info):
        # print("********输出")
        file_handle.write(text_info.decode('utf-8'))

    #phrase data from xml
    def parseXML(xmlFile):
        print("**"*30+"n"+"**"*30)
        #record number
        # total_rec=0
        try:
            with open(file_name,'a') as file_handle:
                dom = minidom.parse(xml_file)
                root = dom.getElementsByTagName("response")
                # print(root.length,'length')
                for node in root:
                    pois = node.getElementsByTagName("pois")
                    for poi in pois[0].getElementsByTagName('poi'):
                        name=poi.getElementsByTagName("name")[0].childNodes[0].nodeValue
                        Typecode=poi.getElementsByTagName("typecode")[0].childNodes[0].nodeValue
                        location=poi.getElementsByTagName("location")[0].childNodes[0].nodeValue
                        province=poi.getElementsByTagName("pname")[0].childNodes[0].nodeValue
```

```
                                city=poi.getElementsByTagName("cityname")[0].childNodes[0].nodeValue
                                county=poi.getElementsByTagName("adname")[0].childNodes[0].nodeValue
                                try:
                                    address=poi.getElementsByTagName("address")[0].childNodes[0].nodeValue
                                except Exception:
                                    address=' '
                                try:
                                    Type1=dic_POI.get(Typecode)[0]
                                    Type2=dic_POI.get(Typecode)[1]
                                    Type3=dic_POI.get(Typecode)[2]
                                except:
                                    Type1=''
                                    Type2=''
                                    Type3=''

text_info=''+name+','+Typecode+','+Type1+','+Type2+','+Type3+','+location+','+province+','+city+','+county+','+address+'\n'
                                print (text_info)
                                GetPOI.log2file(file_handle,text_info.encode('utf-8'))
            except IOError as err:
                print ("IO error: "+str(err)+" parseXML")

class batchGet:
    total_records_pre=0
    total_records_post=0

    def __init__(self,Types,region,gap,limit,keys,xmlFile):
        self.Types=Types
        self.polygons=region
        self.gap=gap
        self.limit=limit
        self.keys=keys
        self.xmlFile=xmlFile

    def quadtree(self,gap,power=2):
        quad_gap=gap/power
        return quad_gap

    def overLimit(self,polygon,url_amap,total_record,xmlFile):
        each_page_rec=20
        polygon=polygon
        if (total_record%each_page_rec)!=0:
            page_number=total_record//each_page_rec+1
        else:
            page_number=total_record//each_page_rec
        #retrive the other records
        for each_page in range(2,page_number+1):
            print ('parsing page '+str(each_page)+' ... ...')
            url_amap=url_amap.replace('page='+str(each_page-1),'page='+str(each_page))
            GetPOI.getHtml(url_amap)
            print("解析第%i页"%each_page)
            GetPOI.parseXML(xmlFile)
        return total_record

    def divid(self,gap,pos,key,types):
        for posID in range(len(pos)):
            polygon=pos[posID]

url_amap=r'http://restapi.amap.com/v3/place/polygon?key=%s&polygon=%s&keywords=&output=xml&types=%s&offset=20&page=1&extensions=base'%(key,polygon,types)
            url_amap=quote(url_amap, safe = string.printable)
            if GetPOI.getHtml(url_amap)==0:
                try:
                    with open(file_name,'a') as file_handle:
                        dom = minidom.parse(xml_file)
                        root = dom.getElementsByTagName("response")
                        for node in root:
                            # str-->int
                            try:
                                total_rec=int(node.getElementsByTagName('count')[0].childNodes[0].nodeValue)
                                print(total_rec)
                            except Exception:
                                total_rec=0
                except IOError as err:
                    print ("IO error: "+str(err)+" Error1")
            print("数量",total_rec)
            if(total_rec<limit):
                Grid_content=types+'\t'+format(polygon,'<70')+'\t'+format(str(total_rec),'>10')+'\n'
```

```
                    # GetPOI.getHtml()
                    GetPOI.parseXML(file_name)
                    with open(Grid_ID,'a') as GridFile_handle:
                        # log2file(GridFile_handle,Grid_content)
                        GridFile_handle.write(Grid_content)
                    self.overLimit(polygon,url_amap,total_rec,file_name)
                else:
                    pos1=polygon.replace('|',',')
                    gap1=self.quadtree(gap,power=2)
                    pos1=LocaDiv(pos1, gap1).ls_row()
                    self.divid(gap1,pos1,key,types)

    def calculate(self):
        pos=LocaDiv(region, gap).ls_row()
        key=self.keys[0]
        for keyID in range(len(Types)):
            types=Types[keyID]
            try:
                self.divid(gap,pos,key,types)
            except Exception:
                if(len(keys)>1):
                    print("change key",Exception)
                    self.keys.pop(0)
                    self.calculate()

if __name__=='__main__':
    file_name='resultPOI.txt'
    Grid_ID='Grid.txt'
    dic_POI=dict()

    path_POI_ID_table='POI parse.txt'
    if (os.path.exists(path_POI_ID_table)):
        path_ID = open(path_POI_ID_table, 'r')
        res = path_ID.readlines()
    else:
        path_ID = open(r'POI_ID_table.txt', 'a+')
        res = path_ID.readlines()
    for i in range(0,len(res)):
        f = res[i]
        f0 = f.split("\t")
        ID,type1,type2,type3=f0[0],f0[1],f0[2],f0[3][0:-1]
        dic_POI[ID]=[type1,type2,type3]
    f=open(file_name,'w+', encoding='utf-8')

line='%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s\n'%("Name",'Typecode','Type1','Type2','Type3','longitude','latitude','province','city','county','addre
ss')
    f.write(line)
    f.close()
    keys=['a6faabf7297ac9b598a172ba40827a58']
    region='116.9,31.7,117.5,32.0'
    gap=0.05
    limit=850
    xml_file='tmp.xml'

Types=["010000","020000","030000","040000","050000","060000","070000","080000","090000","100000","110000","120000","130000","1
40000","150000","160000","170000","180000","190000","200000"]
    batchGet(Types, region,gap,limit,keys,xml_file).calculate()
```

Supplemental Material B: Online map data crawl system for urban land use data with functional properties

```
import os
import sys
import time,datetime
import math,shutil,re
from selenium import webdriver
from selenium.webdriver.common.keys import Keys
from selenium.webdriver.chrome.options import Options
from PIL import Image
from win32api import GetSystemMetrics
import urllib.request, urllib.error
import threading
```

```
from osgeo import gdal
import cv2
import subprocess
import glob
import numpy as np
from tqdm import trange
import configparser


def PixelToInglat(level,coordinate_pix,dir_chrome):
    '''
    chrome_options = Options()
    chrome_options.add_argument('--headless')
    chrome_options.add_argument('--disable-extensions')
    chrome_options.add_argument('--no-sandbox')
    chrome_options.add_argument("disable-infobars")
    chrome_options.add_argument('--disable-gpu')
    chrome_options.binary_location = dir_chrome
    driver = webdriver.Chrome(chrome_options=chrome_options)
    dir_html = os.getcwd()
    html = "file:///%s\\InglatToPixel.html" % (dir_html)
    driver.get(html)
    # get coordinates
    lng = driver.execute_script("return new AMap.Map('container', {resizeEnable: true,zoom: %d}).pixelToLngLat(new AMap.Pixel%s).getLng()" % (level, tuple(coordinate_pix)))
    lat = driver.execute_script("return new AMap.Map('container', {resizeEnable: true,zoom: %d}).pixelToLngLat(new AMap.Pixel%s).getLat()" % (level, tuple(coordinate_pix)))
    # driver.close()
    driver.quit()
    return [lng, lat]


def InglatToPixel(level,coordinate,dir_chrome):
    chrome_options = Options()
    chrome_options.add_argument('--headless')
    chrome_options.add_argument('--disable-extensions')
    chrome_options.add_argument('--no-sandbox')
    chrome_options.add_argument("disable-infobars")
    chrome_options.add_argument('--disable-gpu')
    chrome_options.binary_location = dir_chrome
    driver = webdriver.Chrome(chrome_options=chrome_options)
    dir_html=os.getcwd()
    html="file:///%s\\InglatToPixel.html"%(dir_html)
    driver.get(html)
    X_pixel=round(driver.execute_script("return new AMap.Map('container', {resizeEnable: true,zoom: %d}).InglatToPixel(%s).getX()"%(level,coordinate)))
    Y_pixel=round(driver.execute_script("return new AMap.Map('container', {resizeEnable: true,zoom: %d}).InglatToPixel(%s).getY()"%(level,coordinate)))
    # driver.close()
    driver.quit()
    return [X_pixel,Y_pixel]
def
AmapDataTool(level_To,AmapCenter,start_Lng_Lat,width,height,rows,columns,dir_out,dir_chrome,AmapKey,level_default=12,res=100,change_diag=False):
    folders = [folder for folder in glob.glob(dir_out + "*") if os.path.isdir(folder)]
    if(len(folders)!=rows):
        AmapCenter_pix_defaultLevel = InglatToPixel(level_default, AmapCenter, dir_chrome)
        # AmapCenter_pix_LevelTo=InglatToPixel(level_To,AmapCenter,dir_chrome)
        start = InglatToPixel(level_default, start_Lng_Lat, dir_chrome)

        distance_move_H=math.ceil(start[0]-AmapCenter_pix_defaultLevel[0])
        distance_move_V=math.ceil(start[1]-AmapCenter_pix_defaultLevel[1])
        print("Distance to center",distance_move_H,distance_move_V)

        if(distance_move_H>=0):
            director_H=Keys.ARROW_RIGHT
        else:
            director_H=Keys.ARROW_LEFT
        if(distance_move_V>=0):
            director_V=Keys.ARROW_DOWN
        else:
            director_V=Keys.ARROW_UP
        chrome_options = Options()
        chrome_options.add_argument('--headless')
        chrome_options.add_argument('--disable-extensions')
        chrome_options.add_argument('--no-sandbox')
        chrome_options.add_argument("disable-infobars")
```

```
chrome_options.add_argument('--disable-gpu')
chrome_options.binary_location = dir_chrome
driver = webdriver.Chrome(chrome_options=chrome_options)
# driver = webdriver.Chrome()
url='http://lbs.amap.com/dev/mapstyle/mapshare/%s'%AmapKey
driver.get(url)
outerWidth=driver.execute_script("return window.outerWidth")
outerHeight=driver.execute_script("return window.outerHeight")
innerWidth=driver.execute_script("return window.innerWidth")
innerHeight=driver.execute_script("return window.innerHeight")


driver.set_window_size(width+outerWidth-innerWidth,height+outerHeight-innerHeight)
driver.implicitly_wait(10)
driver.execute_script("document.getElementsByClassName('amap-controlbar')[0].style.display = 'none';")
driver.execute_script("document.body.style.overflow='hidden'")
element = driver.find_element_by_tag_name('body')

for i in range(math.ceil(abs(distance_move_H)/res)):
    element.send_keys(director_H)
for i in range(math.ceil(abs(distance_move_V)/res)):
    element.send_keys(director_V)
#default zoom level =12, the max level=18
# driver.implicitly_wait(10)
for i in range(level_default,level_To):
    element.send_keys(Keys.ADD)

time.sleep(1)
# Imgs = os.listdir(dir_out)
Imgs = glob.glob(dir_out + "*.png")
len_Imgs = len(Imgs)
print("Have got %i images"%len_Imgs)
# folders = [folder for folder in glob.glob(path + "*") if os.path.isdir(folder)]
# and len(folders) != rows
if(len_Imgs!=rows * columns ):
    cur_row = math.floor(len_Imgs / columns)
    if (cur_row % 2 == 0):
        if (change_diag == False):
            cur_col = len_Imgs % columns
        else:
            cur_col = columns - len_Imgs % columns
    else:
        if (change_diag == False):
            cur_col = columns - len_Imgs % columns
        else:
            cur_col = len_Imgs % columns

    for i in range(int(cur_row * height / res)):
        element.send_keys(Keys.DOWN)
    for i in range(int(cur_col * width / res)):
        element.send_keys(Keys.RIGHT)
    print("start screenshot")
    len_name = int(rows * columns)
    for r in trange(cur_row, rows):
        if (r == cur_row):
            if (cur_row % 2 == 0):
                if (change_diag == False):
                    cur_col = cur_col
                else:
                    cur_col = columns - cur_col - 1
            else:
                if (change_diag == False):
                    cur_col = columns - cur_col - 1
                else:
                    cur_col = cur_col
        else:
            cur_col = 0
        for c in range(cur_col, columns):
            # timeGap(10,url)
            time.sleep(0.3)
            driver.get_screenshot_as_file(dir_out+'{0}{1}{2}'.format('%0',len(str(len_name)),'i.png')%(r*columns+c))
            if (r % 2 == 0):
                for i in range(int(width/res)):
                    if(change_diag==False):
                        element.send_keys(Keys.ARROW_RIGHT)
                    else:
                        element.send_keys(Keys.ARROW_LEFT)
```

```
                            else:
                                for i in range(int(width/res)):
                                    if (change_diag == False):
                                        element.send_keys(Keys.ARROW_LEFT)
                                    else:
                                        element.send_keys(Keys.ARROW_RIGHT)
                            for j in range(int(height / res)):
                                element.send_keys(Keys.ARROW_DOWN)
                            for j in range(int(width / res)):
                                if (r % 2 == 0):
                                    if (change_diag == False):
                                        element.send_keys(Keys.ARROW_LEFT)
                                    else:
                                        element.send_keys(Keys.ARROW_RIGHT)
                                else:
                                    if (change_diag == False):
                                        element.send_keys(Keys.ARROW_RIGHT)
                                    else:
                                        element.send_keys(Keys.ARROW_LEFT)
                    else:
                        print("Done the screenshot")
                    # driver.close()
                    driver.quit()
def pic_combain_PIL(Lng_Lat_pix1, Lng_Lat_pix2, dir_pngs, limit=45500,change_diag=False):
    folder = dir_pngs + '合成\\'
    if (os.path.exists(folder) == False):
        os.mkdir(folder)
    Imgs = os.listdir(dir_pngs)
    with Image.open(dir_pngs + Imgs[0]) as temp_img:
        image_size = temp_img.size
    width = image_size[0]
    height = image_size[1]
    limit_cols = math.floor(limit / width)
    limit_rows = math.floor(limit / height)
    rows = abs(math.ceil((Lng_Lat_pix2[1] - Lng_Lat_pix1[1]) /    limit_rows/ height))
    cols = abs(math.ceil((Lng_Lat_pix2[0] - Lng_Lat_pix1[0]) /limit_cols    / width))
    with open(folder+'log.txt','w+') as log:
        txt_log="%s%d%s%s%d"%("rows:",rows,'\n',"colums:",cols)
        print(txt_log)
        log.write(txt_log)
    print(cols,rows)
    dif_col = math.ceil(abs(Lng_Lat_pix2[0] - Lng_Lat_pix1[0]) /width ) -(cols - 1) * limit_cols
    dif_row = math.ceil(abs(Lng_Lat_pix2[1] - Lng_Lat_pix1[1]) / height) - (rows - 1) * limit_rows

    for r in range(rows):
        for c in range(cols):
            if (r < rows - 1 and c < cols - 1):
                new_img = Image.new('RGB', (limit_cols * width, limit_rows * height), 255)
                for lr in range(limit_rows):
                    for lc in range(limit_cols):
                        if ((r * limit_rows + lr) % 2 == 0):
                            if(change_diag==False):
                                index = r * ((cols - 1) * limit_cols + dif_col) * limit_rows + lr * (limit_cols * (cols - 1) + dif_col) + c *
limit_cols + lc
                            else:
                                index = r * ((cols - 1) * limit_cols + dif_col) * limit_rows + lr * (limit_cols * (cols - 1) + dif_col) + ((cols -
1) * limit_cols + dif_col - c * limit_cols - lc - 1)
                        else:
                            if (change_diag == False):
                                index = r * ((cols - 1) * limit_cols + dif_col) * limit_rows + lr * (limit_cols * (cols - 1) + dif_col) + ((cols -
1) * limit_cols + dif_col - c * limit_cols - lc - 1)
                            else:
                                index = r * ((cols - 1) * limit_cols + dif_col) * limit_rows + lr * (limit_cols * (cols - 1) + dif_col) + c *
limit_cols + lc
                        print(Imgs[index])
                        with Image.open(dir+ Imgs[index]) as tmp_img:
                            new_img.paste(tmp_img, (lc * width, lr * height, (lc + 1) * width, (lr + 1) * height))
            else:
                if (c == cols - 1 and r < rows - 1):
                    new_img = Image.new('RGB', (dif_col * width, limit_rows * height), 255)
                    for lr in range(limit_rows):
                        for lc in range(dif_col):
                            if ((r * limit_rows + lr) % 2 == 0):
                                if(change_diag == False):
                                    index = r * ((cols - 1) * limit_cols + dif_col) * limit_rows + lr * (limit_cols * (cols - 1) + dif_col) + c *
limit_cols + lc
                                else:
```

```
                                        index = r * ((cols - 1) * limit_cols + dif_col) * limit_rows + lr * (limit_cols * (cols - 1) + dif_col) +
((cols - 1) * limit_cols + dif_col - c * limit_cols - lc - 1)
                                    else:
                                        if (change_diag == False):
                                            index = r * ((cols - 1) * limit_cols + dif_col) * limit_rows + lr * (limit_cols * (cols - 1) + dif_col) +
((cols - 1) * limit_cols + dif_col - c * limit_cols - lc - 1)
                                        else:
                                            index = r * ((cols - 1) * limit_cols + dif_col) * limit_rows + lr * (limit_cols * (cols - 1) + dif_col) + c *
limit_cols + lc
                                    print(Imgs[index])
                                    tmp_img = Image.open(dir_pngs    + Imgs[index])
                                    new_img.paste(tmp_img, (lc * width, lr * height, (lc + 1) * width, (lr + 1) * height))
                        if (r == rows - 1 and c < cols - 1):
                            # print('else2')
                            new_img = Image.new('RGB', (limit_cols * width, dif_row * height), 255)
                            for lr in range(dif_row):
                                for lc in range(limit_cols):
                                    if ((r * limit_rows + lr) % 2 == 0):
                                        if(change_diag == False):
                                            index = r * ((cols - 1) * limit_cols + dif_col) * limit_rows + lr * (limit_cols * (cols - 1) + dif_col) + c *
limit_cols + lc
                                        else:
                                            index = r * ((cols - 1) * limit_cols + dif_col) * limit_rows + lr * (limit_cols * (cols - 1) + dif_col) +
((cols - 1) * limit_cols + dif_col - c * limit_cols - lc - 1)
                                    else:
                                        if (change_diag == False):
                                            index = r * ((cols - 1) * limit_cols + dif_col) * limit_rows + lr * (limit_cols * (cols - 1) + dif_col) +
((cols - 1) * limit_cols + dif_col - c * limit_cols - lc - 1)
                                        else:
                                            index = r * ((cols - 1) * limit_cols + dif_col) * limit_rows + lr * (limit_cols * (cols - 1) + dif_col) + c *
limit_cols + lc
                                    print(Imgs[index])
                                    tmp_img = Image.open(dir_pngs    + Imgs[index])
                                    new_img.paste(tmp_img, (lc * width, lr * height, (lc + 1) * width, (lr + 1) * height))
                        if (c == cols - 1 and r == rows - 1):
                            new_img = Image.new('RGB', (dif_col* width, dif_row * height), 255)
                            for lr in range(dif_row):
                                for lc in range(dif_col):
                                    if ((r * limit_rows + lr) % 2 == 0):
                                        if (change_diag == False):
                                            index = r * ((cols - 1) * limit_cols + dif_col) * limit_rows + lr * (limit_cols * (cols - 1) + dif_col) + c *
limit_cols + lc
                                        else:
                                            index = r * ((cols - 1) * limit_cols + dif_col) * limit_rows + lr * (limit_cols * (cols - 1) + dif_col) +
((cols - 1) * limit_cols + dif_col - c * limit_cols - lc - 1)

                                    if ((r * limit_rows + lr) % 2 != 0):
                                        if (change_diag == False):
                                            index = r * ((cols - 1) * limit_cols + dif_col) * limit_rows + lr * (limit_cols * (cols - 1) + dif_col) +
((cols - 1) * limit_cols + dif_col - c * limit_cols - lc - 1)
                                        else:
                                            index = r * ((cols - 1) * limit_cols + dif_col) * limit_rows + lr * (limit_cols * (cols - 1) + dif_col) + c *
limit_cols + lc
                                    print(Imgs[index])
                                    tmp_img = Image.open(dir_pngs    + Imgs[index])
                                    new_img.paste(tmp_img, (lc * width, lr * height, (lc + 1) * width, (lr + 1) * height))
                    # new_img.save(folder+'%s.jpg'%(r*cols+c))
                    new_img.save(folder+'{0}{1}{2}'.format('%0', len(str(cols*rows)), 'i.png') % (r*cols+c))

def doShift(LTLng_Lat_pix, RBLng_Lat_pix,LTLng_Lat_coordinates,RBLng_Lat_coordinates,rows,columns, dir_pngs,width,height):
    coordinates=LTLng_Lat_coordinates+RBLng_Lat_coordinates
    pngs=[]
    for i in range(rows*columns):
        png_path = dir_pngs + '{0}{1}{2}'.format('%0', len(str(int(rows * columns))), 'i.png') % (i)
        pngs.append(png_path)

    xml = glob.glob(dir_pngs + "*.xml")
    if(len(pngs)!=len(xml) and len(glob.glob(dir_pngs+"*.png"))>0):
        for r in trange(rows):
            threads = []
            for c in range(columns):
                if(r%2==0):
                    index=columns*r+c
                else:
                    index=(r+1)*columns-c-1
                lon_offset=width*c
                lat_offset=-height*r
```

```
                    if(not os.path.exists(pngs[index]+".aux.xml")):
                        t = threading.Thread(target=shift, args=(pngs[index],lon_offset,lat_offset))
                        threads.append(t)
                for t in threads:
                    t.start()
                for t in threads:
                    t.join()

def project(input_img,LTLng_Lat_pix, RBLng_Lat_pix,LTLng_Lat_coordinates,RBLng_Lat_coordinates,dir_gdal):

    x_pix = RBLng_Lat_pix[0] - LTLng_Lat_pix[0]
    y_pix = RBLng_Lat_pix[1] - LTLng_Lat_pix[1]
    coordinates = LTLng_Lat_coordinates + RBLng_Lat_coordinates
    gcp_TL = "%d %d %f %f %f" % (0, 0, coordinates[0], coordinates[1], 0.0,)
    gcp_TR = "%d %d %f %f %f" % (x_pix, 0, coordinates[2], coordinates[1], 0.0)
    gcp_BR = "%d %d %f %f %f" % (x_pix, y_pix, coordinates[2], coordinates[3], 0.0)
    gcp_BL = "%d %d %f %f %f" % (0, y_pix, coordinates[0], coordinates[3], 0.0)
    command = "%sgdal_translate -co COMPRESS=DEFLATE -a_srs EPSG:4326 -co COMPRESS=DEFLATE -gcp %s -gcp %s -gcp %s -gcp %s %s %s
" % (dir_gdal, gcp_TL, gcp_TR, gcp_BR, gcp_BL, input_img, input_img.replace(".tif", '_Geo.tif'))
    subprocess.call(command, shell=False)



def extractBand(input_img):
    outMerged_name = os.path.split(os.path.dirname(input_img))[1] + ".tif"
    outPath = os.path.split(os.path.dirname(input_img))[0] + "\\"
    if not os.path.exists(outPath):
        os.mkdir(outPath)

    outputPath_merged = outPath + outMerged_name
    merge_image = cv2.imdecode(np.fromfile(outputPath_merged, dtype=np.uint8), -1)
    red_image=merge_image[:,:,1]
    green_image = merge_image[:, :, 2]
    blue_image = merge_image[:, :, 3]

    red_image_path = outPath + outMerged_name.replace(".tif", '_red.tif')
    green_image_path = outPath + outMerged_name.replace(".tif", '_green.tif')
    blue_image_path = outPath + outMerged_name.replace(".tif", '_blue.tif')

    cv2.imencode('.png', red_image)[1].tofile(red_image_path)
    cv2.imencode('.png', green_image)[1].tofile(green_image_path)
    cv2.imencode('.png', blue_image)[1].tofile(blue_image_path)


def shift(filePath,x_offset,y_offset):
    # gdal_translate multi threads
    rast_src = gdal.Open(filePath)
    gt = rast_src.GetGeoTransform()
    gtl = list(gt)
    gtl[0] = x_offset
    gtl[1] = 1
    gtl[2] = 0
    gtl[3] = y_offset
    gtl[4] = 0
    gtl[5] = -1
    rast_src.SetGeoTransform(tuple(gtl))
    rast_src = None


def doMerge(path,rows,columns,format_suffix):
    print("sub folders merge...")
    if(len(glob.glob(path+"*.%s"%format_suffix))>0):
        for r in trange(rows):
            out_name = "{0}{1}{2}".format("%0", len(str(rows)), "d") % r
            newFolder = path + out_name
            if (not os.path.exists(newFolder)):
                os.mkdir(newFolder)
            for c in range(columns):
                tif = path + '{0}{1}{2}'.format('%0', len(str(rows*columns)), 'i.%s'%format_suffix) % (r * columns + c)
                if(os.path.exists(tif)):
                    xml = tif+ ".aux.xml"
                    newTifFile=newFolder+"\\\\"+os.path.split(tif)[1]
                    newXmlFile=newFolder+"\\\\"+os.path.split(xml)[1]
                    if(os.path.exists(newTifFile)):
                        os.remove(newTifFile)
                        shutil.move(tif, newFolder)
                    else:
```

```
                shutil.move(tif, newFolder)
            if(os.path.exists(newXmlFile)):
                os.remove(newXmlFile)
                shutil.move(xml, newFolder)
            else:
                shutil.move(xml, newFolder)

    path_up1 = os.path.dirname(path)
    out_path = os.path.split(path_up1)[0] + "\\" + os.path.split(path_up1)[1] + "_temp\\"
    if(not os.path.exists(out_path)):
        os.mkdir(out_path)
    threads = []
    folders=[folder for folder in glob.glob(path+"*") if os.path.isdir(folder)]
    for i in trange(1,len(folders)+1):
        folder=folders[i-1]
        out_vrt = folder + "\\%s.vrt" % os.path.split(folder)[1]
        imgs=folder+"\\*.%s"%format_suffix
        mergedTIF = out_path + "\\%s.tif" % os.path.split(folder)[1]
        if(not os.path.exists(mergedTIF)):
        # newTifs = os.path.split(tifs)[0] + "\\" + out_name + "\\" + os.path.split(tifs)[1]
            t = threading.Thread(target=merge, args=(imgs, out_vrt, mergedTIF))
            threads.append(t)
            if (i%10 == 0 or i==len(folders)):
                for t in threads:
                    t.start()
                for t in threads:
                    t.join()
                threads = []
            if os.path.exists(out_vrt):
                os.remove(out_vrt)

def merge(imgs,out_vrt,out_merge):
    if not os.path.exists(out_merge):
        if os.path.exists(out_vrt):
            os.remove(out_vrt)
        command = "C:\\Program Files (x86)\\GDAL\\gdalbuildvrt %s %s" % (out_vrt, imgs)
        subprocess.call(command, shell=False)
        command = "gdal_translate   --config GDAL_CACHE_MAX 1024 -co NUM_THREADS=ALL_CPUS -of GTiff -co COMPRESS=JPEG -co
PHOTOMETRIC=RGB -co TILED=YES %s %s" % (out_vrt, out_merge)
        subprocess.call(command, shell=False)
        if os.path.exists(out_vrt):
            os.remove(out_vrt)


def check_contain_chinese(check_str):
    for ch in check_str:
        if u'\u4e00' <= ch <= u'\u9fff':
            print("The path is not legal !  ")
            sys.exit()
            return True
    return False

def log_done(time,LT_pix,RB_pix,LT_coor,RBY_coor):
    with open("Had got data.txt","a") as log:
        # str(LT_pix), str(RB_pix), str(LT_coor), str(RBY_coor)
        txt_log=str(time)+":\t%s\t%s\t%s\t%s\n"%(LT_pix,RB_pix,LT_coor,RBY_coor)
        log.write(txt_log)

def config():
    cf = configparser.ConfigParser()
    cf.read("config.conf", encoding="utf-8-sig")
    Lng_Lat_1 = cf.get("config","The top-left coordinate")
    Lng_Lat_2 = cf.get("config"," The bottom-right coordinate ")
    start_Lng_Lat = Lng_Lat_1
    level_To = cf.get("config","zoom level")
    AmapKey = cf.get("config","yourself Amap key")
    dir_out = cf.get("config","outPath")
    dir_gdal = cf.get("config","GDAL path")
    dir_chrome = cf.get("config","chrome path")
    operate = cf.get("config"," operation ")
    format_suffix = cf.get("config"," suffix format")

    AmapCenter = [116.405285, 39.904989]

    Lng_Lat_1=[float(re.findall(r'-?\d+\.?\d*e?-?\d*?', Lng_Lat_1)[0]),float(re.findall(r'-?\d+\.?\d*e?-?\d*?', Lng_Lat_1)[1])]
    Lng_Lat_2 = [float(re.findall(r'-?\d+\.?\d*e?-?\d*?', Lng_Lat_2)[0]),float(re.findall(r'-?\d+\.?\d*e?-?\d*?', Lng_Lat_2)[1])]
    # start_Lng_Lat=[float(re.findall(r'-?\d+\.?\d*e?-?\d*?', start_Lng_Lat)[0]),float(re.findall(r'-?\d+\.?\d*e?-?\d*?', start_Lng_Lat)[1])]
```

```
    level_To=int(level_To)

    LTLng_Lat_pix = lnglatToPixel(level_To, Lng_Lat_1, dir_chrome)
    RBLng_Lat_pix = lnglatToPixel(level_To, Lng_Lat_2, dir_chrome)
    AmapCenter_Lng_LatPix_LevelTo = lnglatToPixel(level_To, AmapCenter, dir_chrome)
    check_contain_chinese(dir_out)
    # get screen resolution
    width = int((GetSystemMetrics(0) - 70) / 100) * 100
    height = int((GetSystemMetrics(1) - 90) / 100) * 100


    columns_LT = math.ceil((AmapCenter_Lng_LatPix_LevelTo[0] - LTLng_Lat_pix[0] - width / 2) / width)
    rows_LT = math.ceil((AmapCenter_Lng_LatPix_LevelTo[1] - LTLng_Lat_pix[1] - height / 2) / height)
    columns_RB = math.ceil((RBLng_Lat_pix[0] - AmapCenter_Lng_LatPix_LevelTo[0] - width / 2) / width)
    rows_RB = math.ceil((RBLng_Lat_pix[1] - AmapCenter_Lng_LatPix_LevelTo[1] - height / 2) / height)

    columns = columns_LT + columns_RB + 1
    rows = rows_LT + rows_RB + 1
    print("The dimension ", rows, columns)
    LTLng_Lat_pix = [AmapCenter_Lng_LatPix_LevelTo[0] - columns_LT * width - width / 2, AmapCenter_Lng_LatPix_LevelTo[1] - rows_LT *
height - height / 2]
    RBLng_Lat_pix = [AmapCenter_Lng_LatPix_LevelTo[0] + columns_RB * width + width / 2, AmapCenter_Lng_LatPix_LevelTo[1] + rows_RB
* height + height / 2]
    print("The real coordinates", LTLng_Lat_pix, RBLng_Lat_pix)
    LTLng_Lat_coor = PixelTolnglat(level_To, LTLng_Lat_pix, dir_chrome)
    RBlng_Lat_coor = PixelTolnglat(level_To, RBLng_Lat_pix, dir_chrome)
    time_log = datetime.datetime.now()
    with open("The control points.txt","w+") as log:
        LT="%f\t%f\t%f\t%f"%(0,0,LTLng_Lat_coor[0],LTLng_Lat_coor[1])
        RT="%f\t%f\t%f\t%f"%(columns*width,0,RBlng_Lat_coor[0],LTLng_Lat_coor[1])
        LB="%f\t%f\t%f\t%f"%(0,-rows*height,LTLng_Lat_coor[0],RBlng_Lat_coor[1])
        RB="%f\t%f\t%f\t%f"%(columns*width,-rows*height,RBlng_Lat_coor[0],RBlng_Lat_coor[1])
        txt_log="%s\n%s\n%s\n%s"%(LT,RT,LB,RB)
        log.write(txt_log)
        AmapDataTool(level_To,          AmapCenter,          start_Lng_Lat,width,height,rows,columns,          dir_out,          dir_chrome,
AmapKey,change_diag=False)
        doShift(LTLng_Lat_pix, RBLng_Lat_pix, LTLng_Lat_coor, RBlng_Lat_coor,rows,columns, dir_out, width, height)
        doMerge(dir_out, rows, columns, format_suffix="png")
        path_up1 = os.path.dirname(dir_out)
        out_path = os.path.split(path_up1)[0] + "\\" + os.path.split(path_up1)[1] + "_temp\\"
        imgs = out_path + "*.tif"
        out_vrt = out_path + os.path.split(path_up1)[1] + ".vrt"
        out_merge = os.path.dirname(path_up1) + "\\" + os.path.split(path_up1)[1] + "_merged.tif"
        merge(imgs, out_vrt, out_merge)
        project(out_merge, LTLng_Lat_pix, RBLng_Lat_pix,LTLng_Lat_coor, RBlng_Lat_coor, dir_gdal)
        # extractBand(out_merge)

if __name__ == '__main__':
    config()
```