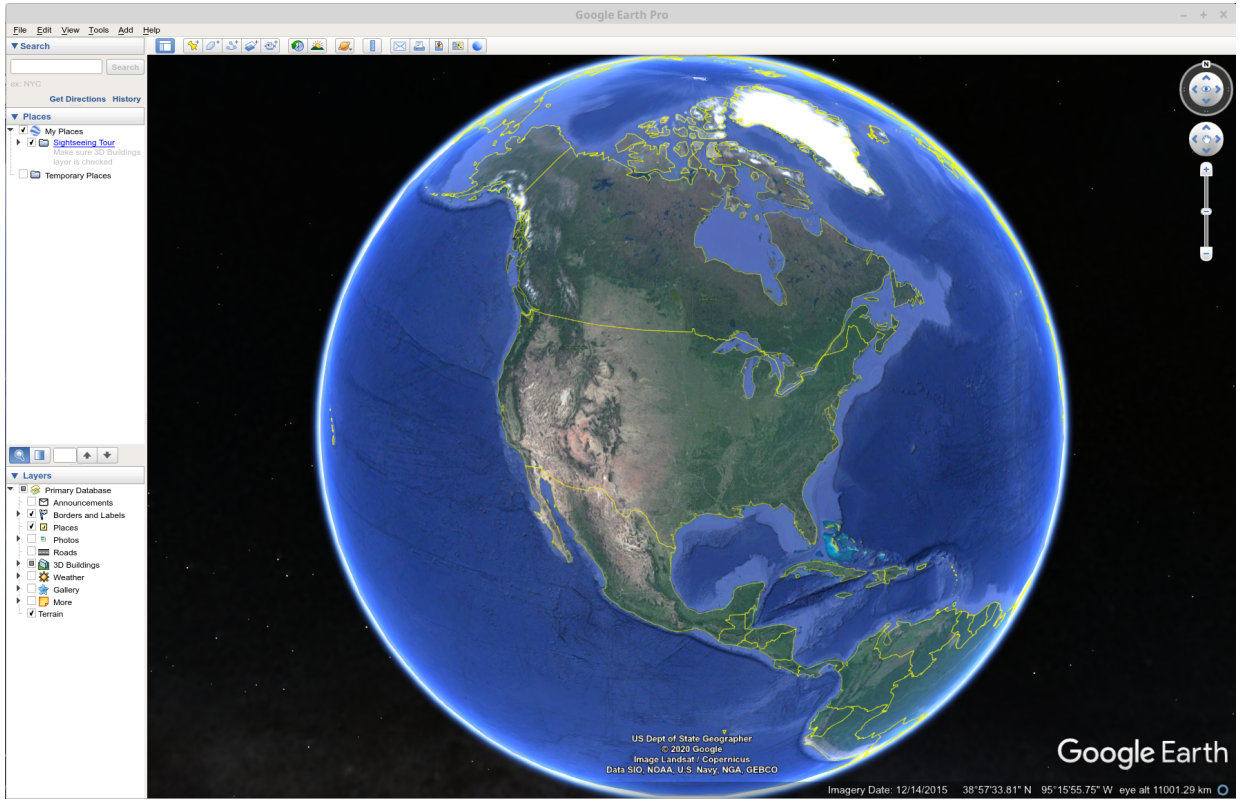
GPS Receiver Mapping

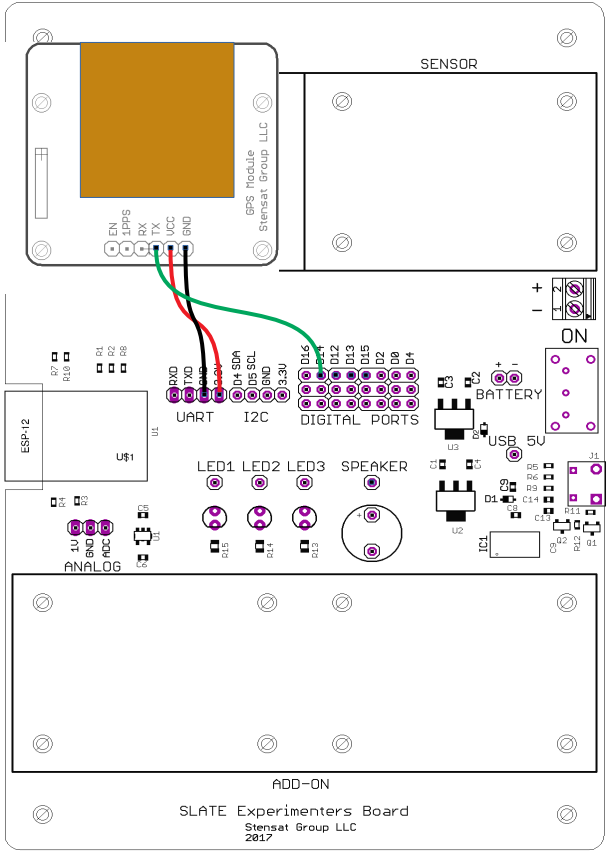


Mapping

- In this section, the GPS position data will be captured by a python program and a KML file will be generated to plot positions and paths in Google Earth.
- The python program will provide a GUI to let you capture and save the coordinates to any file.
- Open a terminal window or power shell and enter the following:
  - **pip3 install simplekml**
  - This will install the kml generating module.
- To download Google Earth, go to:
  - https://www.google.com/earth/download/gep/agree.html?hl=en-GB
  - Install the program.

GPS Receiver

- Install the GPS receiver module in one of the sensor positions. It can be stacked on top of another module using the included standoffs. The standoffs are aluminum so be careful to not over tighten the nuts on the screws of the standoff.
- Connect the GPS receiver GND to a GND on the SLATE.
- Connect the GPS receiver VCC to the UART 3V.
- Connect the GPS receiver TX pin to digital pin 14.



Mapping

- The Arduino program will be used again from the previous GPS lesson. The print statements will be replaced to support generating the KML file.
- Change the program to print only the floating point latitude and longitude with a comma separating them. This will be captured by the python program.
- Upload the code the SLATE and verify it works using the serial monitor.

```
void setup() {
  Serial.begin(115200);
  gps.begin(9600);
}

void loop() {
  char buf[128];
  struct gngga pos;
  if(gps.available() > 0) {
    bzero(buf,128);
    gps.readBytesUntil('\n',buf,128);
    if(!strncmp("$GNGGA",buf,6)) {
      getPosition(buf,&pos);
      Serial.print(pos.flatitude,5);
      Serial.print(“,”);
      Serial.println(pos.flongitude,5);
    }
  }
}
```

Function **getPosition()** is not shown. It is required.

Mapping

- Enter the program. It is spread over a few pages.
- A few new modules are required.
  - **filedialog** will let you open a dialog window to select a file to save. The **filedialog** module is imported and renamed **fd**.
  - **simplekml** will create the **KML** file.
  - threading will let the data be captured in parallel with the tkinter **mainloop()** running.

```
from tkinter import *
from tkinter import filedialog as fd
import serial
import simplekml
import threading

top = Tk()
s = serial.Serial('COM4',115200)

capture = 0

def startcapture():
    global capture
    capture = 1
    print("Start capture")

def stopcapture():
    global capture
    capture = 2
    print("Stop capture")
```

Mapping

- The tkinter window is created and the serial port is opened.
- variable capture is set to zero. This variable will be used to determine the state of operations.
  - zero is nothing is happening
  - one is data is being captured
  - two is data gets saved and the capture stops

```
from tkinter import *
from tkinter import filedialog as fd
import serial
import simplekml
import threading

top = Tk()
s = serial.Serial('COM4',115200)

capture = 0

def startcapture():
    global capture
    capture = 1
    print("Start capture")

def stopcapture():
    global capture
    capture = 2
    print("Stop capture")
```

Mapping

- The two functions are called by buttons that are declared later. The **startcapture()** function changes the the **capture** variable to one. Notice the statement **global capture** at the top of the function. This is required to use the **capture** that was declared earlier. If **capture** is not declared **global**, the function will create its own variable that is separate from the one declared above. The variable would be a local variable only available in the function and not accessible in any other part of the program.
- The function **stopcapture()** changes the global variable **capture** to two.

```
from tkinter import *
from tkinter import filedialog as fd
import serial
import simplekml
import threading

top = Tk()
s = serial.Serial('COM4',115200)

capture = 0

def startcapture():
    global capture
    capture = 1
    print("Start capture")

def stopcapture():
    global capture
    capture = 2
    print("Stop capture")
```

Mapping

- This function runs in parallel with tkinter **mainloop()** as a thread.
- The variable **capture** is declared as a global variable so it uses the one declared in the beginning of the program.
- variable **kml** is declared. This will hold the coordinates captured.
- List **position** is declared.

```
def capture_coords():
    global capture
    print("Starting thread")
    kml = simplekml.Kml()
    position = []
    while 1:
        a = s.readline()
        b = a.decode('utf-8','ignore')
        l1.config(text=b)
        if capture == 1:
            print(b)
            c = b.split(',')
            if len(c) == 2:
                position.append((float(c[1]),float(c[0])))
        if capture == 2:
            print("Saving kml")
            lin = kml.newlinestring(name='here',coords=position)
            filename = fd.asksaveasfilename()
            kml.save(filename)
            capture = 0
```

Mapping

- At the start of the loop, the coordinates are read from the SLATE and converted to a string using the decode function.
- Label **l1** is changed to display the coordinates received.

```
def capture_coords():
    global capture
    print("Starting thread")
    kml = simplekml.Kml()
    position = []
    while 1:
        a = s.readline()
        b = a.decode('utf-8','ignore')
        l1.config(text=b)
        if capture == 1:
            print(b)
            c = b.split(',')
            if len(c) == 2:
                position.append((float(c[1]),float(c[0])))
        if capture == 2:
            print("Saving kml")
            lin = kml.newlinestring(name='here',coords=position)
            filename = fd.asksaveasfilename()
            kml.save(filename)
            capture = 0
```

Mapping

- If capture is one, the coordinates are to be captured. The string **b** is split in two and converted to floating point values when appended to the list **position**.
- As the function loops, the coordinates are added to the **position** list.

```
def capture_coords():
    global capture
    print("Starting thread")
    kml = simplekml.Kml()
    position = []
    while 1:
        a = s.readline()
        b = a.decode('utf-8','ignore')
        l1.config(text=b)
        if capture == 1:
            print(b)
            c = b.split(',')
            if len(c) == 2:
                position.append((float(c[1]),float(c[0])))
        if capture == 2:
            print("Saving kml")
            lin = kml.newlinestring(name='here',coords=position)
            filename = fd.asksaveasfilename()
            kml.save(filename)
            capture = 0
```

Mapping

- When capture is two, the function saves the data captured.
- A path is created with kml.newlinestring() function.
- A file dialog window is opened with the fd.asksaveasfilename(). The filename and path is returned from the function and saved in variable filename.
- kml.save() saves the position data to the filename specified.
- capture is set to zero to go to the idle state.

```
def capture_coords():
    global capture
    print("Starting thread")
    kml = simplekml.Kml()
    position = []
    while 1:
        a = s.readline()
        b = a.decode('utf-8','ignore')
        l1.config(text=b)
        if capture == 1:
            print(b)
            c = b.split(',')
            if len(c) == 2:
                position.append((float(c[1]),float(c[0])))
        if capture == 2:
            print("Saving kml")
            lin = kml.newlinestring(name='here',coords=position)
            filename = fd.asksaveasfilename()
            kml.save(filename)
            capture = 0
```

Mapping

- The last part of the program creates the label object with the initial text.
- Two buttons are created, one to start the capture and the other to stop the capture.
- Next, the capture_coords() function is set up as a thread to execute in parallel. The x.start() function starts running the thread.
- Lastly, the GUI is created and mainloop() is executed.

```
l1 = Label(top,text="GPS coordinates")
b1 = Button(top,text="Start",command=startcapture,width=20)
b2 = Button(top,text="Stop",command=stopcapture,width=20)

x = threading.Thread(target=capture_coords)
x.start()
l1.pack()
b1.pack()
b2.pack()
mainloop()
```

Mapping

- Start the code on the SLATE. It will start putting out position data when the GPS receiver acquires enough satellites.
- Start the python program. Watch the top where the coordinates are displayed. When coordinates appear that look valid, click on the start button.
  - This is when you can start travelling by car or walking around while collecting position data.
- After some time, click on the stop button and specify the file to save the data. Use **.kml** at the end of the file name.
- Start Google Earth and in Google Earth, select The File menu and select Open. Select the file kml file saved from python. Google Earth should show the positions on the map.

Displaying a Path

- A path can be shown in Google Earth.
- The two highlighted lines set the color and the width of the line that will be drawn between all the coordinates captured.
- The color is in hexadecimal with two digits per color and transparency. The first two digits sets the red color, the second two the green color, the third pair, blue. The last two digits sets the transparency

```
def capture_coords():
    global capture
    print("Starting thread")
    kml = simplekml.Kml()
    position = []
    while 1:
        a = s.readline()
        b = a.decode('utf-8','ignore')
        l1.config(text=b)
        if capture == 1:
            print(b)
            c = b.split(',')
            if len(c) == 2:
                position.append((float(c[1]),float(c[0])))
        if capture == 2:
            print("Saving kml")
            lin = kml.newlinestring(name='here',coords=position)
            lin.style.linestyle.color = '#ff0000ff'
            lin.style.linestyle.width = 5
            filename = fd.asksaveasfilename()
            kml.save(filename)
            capture = 0
```

More details can be found at:
https://simplekml.readthedocs.io/en/latest/