# Applications of Space Science Satellite Navigation Assignment: Where Am I? Determining GPS Receiver Locations using Data from Four Satellites

Alexander Kusmirek [20203284]
*University College Dublin, School of Physics*
Submitted: Monday 7th December 2020

**Abstract**

This report details the determination of the latitude, longitude, and elevation of a Sat Nav receiver based on generated data from four satellites in geostationary orbits around Earth. Using MATLAB, a program was developed to find the receiver location and clock offset, found to be 53.4093°N, 6.2557°W, with a clock offset of 1518.15μs. Consequently, the receiver was located in the north of the city of Dublin in County Dublin, Ireland.

**Keywords:** Satellites, navigation, SatNav, GPS, triangulation, pseudo-trilateration

## 1.0 Introduction

Satellite navigation is the most common use of commercial space equipment in the early 21st century. It works by the triangulation of a satellite navigation receiver's position based on the received signals from multiple satellites in orbit around Earth, usually separated into ranges of low Earth orbits (LEO; below 2,000km altitude), medium Earth orbit (MEO; typically around 20,200km altitude) and high geosynchronous (GSO) and geostationary earth (GEO) orbits at altitudes of 35,000km above the planet's surface. In the determination of the Sat Nav receiver location carried out in this assignment, four MEO satellites at assumed identical altitudes of 20,200km are used to locate the receiver, using unique latitude, longitude, and flight time data generated by a MATLAB *Receiver.p* program provided by Dr Nam Tran at the University College Dublin School of Electrical and Electronic Engineering. A MATLAB script was developed to generate and convert satellite data into a Cartesian coordinate system to allow for the determination of the receiver's location through the Newton method of numerical analysis. Using the Newton method allowed for the Cartesian coordinates of the receiver's position ($x, y, z$) to be calculated alongside a receiver clock offset ($\Delta t$). Once these coordinates were calculated, the receiver location was converted into a spherical coordinate system again to allow for an output in terms of latitude and longitude. The purpose of this assignment was to demonstrate and understand practical applications of space science and the methods by which satellite navigation equipment is used to determine locations anywhere in the world.

Throughout this assignment, the following constants and assumptions were used:

- Speed of light $c$ = 299792458 m/s.
- Mean radius of Earth = 6371km.
- The Earth is perfectly spherical, with completely flat terrain (geocentric = geodetic).
- Satellite altitudes = 20200km.
- All satellites are perfectly synchronised with each other.

## 2.0 Satellite Data and Calculations

The satellite data used to triangulate the receiver's location was generated using the provided *Receiver.p* MATLAB program, which requires a user input of a unique student number to generate arrays of satellite information in MATLAB, creating a unique selection of latitudes, longitudes and measured flight times for the four MEO navigation satellites. Upon running the receiver program, satellite positions were obtained using a spherical coordinate system (Figure 1), with the calculated array of satellite data shown in Table 1 (the exact level of precision in the table values is not required, but it is included as these are the exact values used by the MATLAB script to calculate satellite and receiver positions):

**Table 1: *Receiver.p* Generated Satellite Data (input sn = 3284, output = SatData)**

| **Satellite** | **Latitude (°)** | **Longitude (°)** | **Measured Time of Flight (s)** |
|---|---|---|---|
| ***Satellite 1*** | 46.240026289304033 | 25.669755077194555 | 0.070832170298841 |
| ***Satellite 2*** | 23.058976285965223 | 16.015472549939567 | 0.073701646185311 |
| ***Satellite 3*** | 43.377303993440009 | -50.192755804298500 | 0.072613070078296 |
| ***Satellite 4*** | 85.224018219944185 | -20.643098810585780 | 0.073015654541655 |

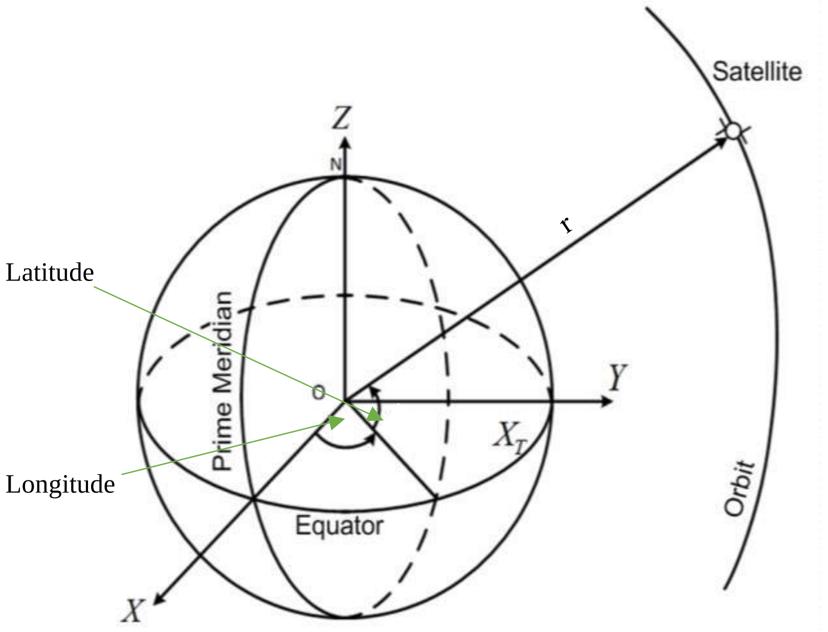


*Figure 1: Geocentric Spherical and Cartesian Coordinate Systems relative to an orbiting satellite, demonstrating the angles used for latitude and longitude (credit: Dubrovin, Baranov, and Tryastsin, 2018).*

In order to convert the satellite data into a Cartesian coordinate system suitable for determining a receiver position (Figure 2), the degree values of latitude and longitude were converted to radians and used to calculate satellite coordinates in terms of $x_k$ $y_k$, and $z_k$. This function outputs values for (x, y, z) for a satellite given its longitude, latitude, and satellite distance from (0, 0). Given that longitude is represented by the azimuth $\theta$ and latitude by the elevation $\phi$, and the distance from the origin to the satellite being $r$, the Cartesian values for $x$, $y$ and $z$ for each satellite are related by the following equations:

$$x = r.\cos\phi.\cos\theta$$

$y = r.\cos\phi.\sin\theta$

$z = r.\sin\phi$

Pseudoranges were calculated using the measured time of flight for each satellite, multiplied by the speed of light *c*, as per the equation *d = c*(tof)* where tof represents measured time of flight. The Cartesian coordinates and pseudoranges for each satellite are listed in Table 2.

**Table 2: Cartesian Coordinates and Pseudoranges of Satellites (array = SatPos)**

| **Satellite *k*** | **$x_k$ (*$10^7$)** | **$y_k$ (*$10^7$)** | **$z_k$ (*$10^7$)** | **$d_k$ (*$10^7$)** |
|---|---|---|---|---|
| ***Satellite 1*** | 1.656377848523558 | 0.796084222863797 | 1.919073407409892 | 2.123495043936409 |
| ***Satellite 2*** | 2.349913882179314 | 0.674513780497314 | 1.040728742881721 | 2.209519766854076 |
| ***Satellite 3*** | 1.236434392169033 | -1.483633176701439 | 1.824895339698250 | 2.176885076169856 |
| ***Satellite 4*** | 0.207026249607946 | -0.077993831832806 | 2.647874148098679 | 2.188954254752168 |

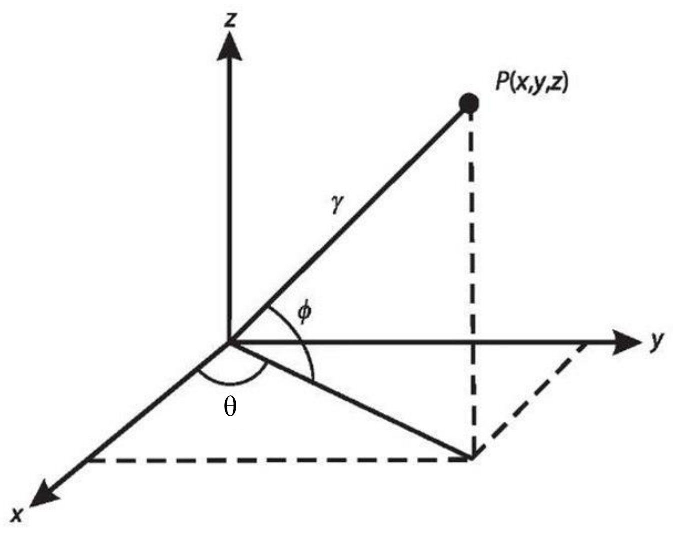


*Figure 2: A receiver position P in terms of (x, y, z), compared against spherical angles and a Cartesian x-y-z coordinate system. (credit: What-When-How, 2020).*

With the (x, y, z) satellite coordinates alongside the pseudoranges calculated, the script progressed to determining the GPS receiver position through numerical analysis.

## 3.0 Receiver Position Calculation

The numerical analysis used to calculate receiver position was the Newton method, as the satellite data generated produces a 4-variable nonlinear system. This method was used because the system is well-defined, meaning it contains an equal number of variables and unknowns: 4 of each, in this case, meaning that Newton's method is the most appropriate choice for optimisation, as opposed to the least known squares method which is used for over-defined systems. The fact that there are four variables and four unknowns to be determined (x, y, z, and b of the receiver) is the reason why four satellites are required for this form of pseudo-trilateration – each satellite correlates to being able to solve one variable when using a Jacobian matrix within Newton's method. This method is an iterative numerical method, typically represented in the form:

$U_{n+1} = U_n - J(\mu_n)\backslash F(\mu_n)$

mu = mu – J\F (expression for Newton's method as used in the MATLAB script)

where: *J()* = 4x4 Jacobian matrix of the vector of functions at step *n*, *F* = 1x4 vector of functions for step *n* (the Cost Function)

To compute the receiver location, the Jacobian *J(μn)* and the cost function *F(μn)* were calculated for each iteration of Newton's method. The initial estimated values for U and F were 4x1 vectors of zeros, with the initial Jacobian being a 4x4 array of zeros. The cost function was expressed as:

$f_k = (x - x_k)^2 + (y - y_k)^2 + (z - z_k)^2 - (d_k - b)^2$

where: k = 1,2,3,4 (representing the four satellites), $(x, y, z)_k$ = satellite Cartesian coordinates, $d_k$ = pseudorange

The Jacobian array produced is of the form:

$$\mathbf{J} = \begin{matrix} \frac{\partial f_1}{\partial x} & \frac{\partial f_1}{\partial y} & \frac{\partial f_1}{\partial z} & \frac{\partial f_1}{\partial b} \\ \frac{\partial f_2}{\partial x} & \frac{\partial f_2}{\partial y} & \frac{\partial f_2}{\partial z} & \frac{\partial f_2}{\partial b} \\ \frac{\partial f_3}{\partial x} & \frac{\partial f_3}{\partial y} & \frac{\partial f_3}{\partial z} & \frac{\partial f_3}{\partial b} \\ \frac{\partial f_4}{\partial x} & \frac{\partial f_4}{\partial y} & \frac{\partial f_4}{\partial z} & \frac{\partial f_4}{\partial b} \end{matrix}$$

When a defined tolerance is reached, the iteration will converge, at which point the script can stop running and output the required receiver location information. Additionally, receiver clock offset *Δt* was calculated using the final value of *b* calculated by Newton's method, whereby

$b = c.\Delta t$ where: *c* = speed of light (m/s)

As the results of the iteration are in Cartesian coordinates, they had to be converted back to a spherical coordinate system and then from radians to degrees, whereby:

latitude (rad) = atan2(y, x)

longitude (rad) = atan2(z, sqrt(x.^2 + y.^2))

elevation (m) = sqrt(x.^2 + y.^2 + z.^2)

And latitude and longitude are converted to degrees by multiplying each value by (180°/pi), whilst the height is equal to the elevation minus the mean Earth radius.

## 4.0 Code Results and Outputs

Figure 3 shows the output of the MATLAB script and the calculated receiver values based on an input student number of 20203284. Given the output shown in Figure 3, the satellite receiver location details are as follows:

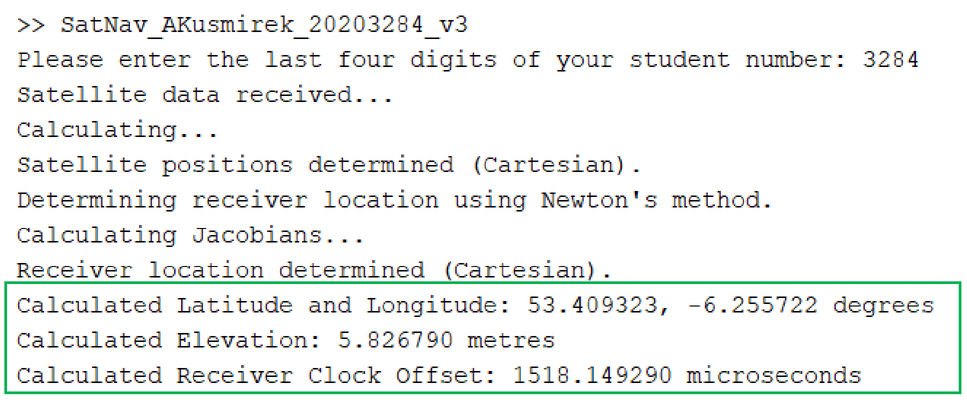
**Latitude:** 53.409323°

**Longitude:** -6.255722°

**Elevation:** 5.826790m

**Receiver Clock Offset:** 1518.149290μs

**Location:** Dublin



```
>> SatNav_AKusmirek_20203284_v3
Please enter the last four digits of your student number: 3284
Satellite data received...
Calculating...
Satellite positions determined (Cartesian).
Determining receiver location using Newton's method.
Calculating Jacobians...
Receiver location determined (Cartesian).
Calculated Latitude and Longitude: 53.409323, -6.255722 degrees
Calculated Elevation: 5.826790 metres
Calculated Receiver Clock Offset: 1518.149290 microseconds
```

*Figure 3: The Output of the MATLAB Script, giving values for the spherical coordinate system location of the GPS receiver based on data received from 4 satellites.*

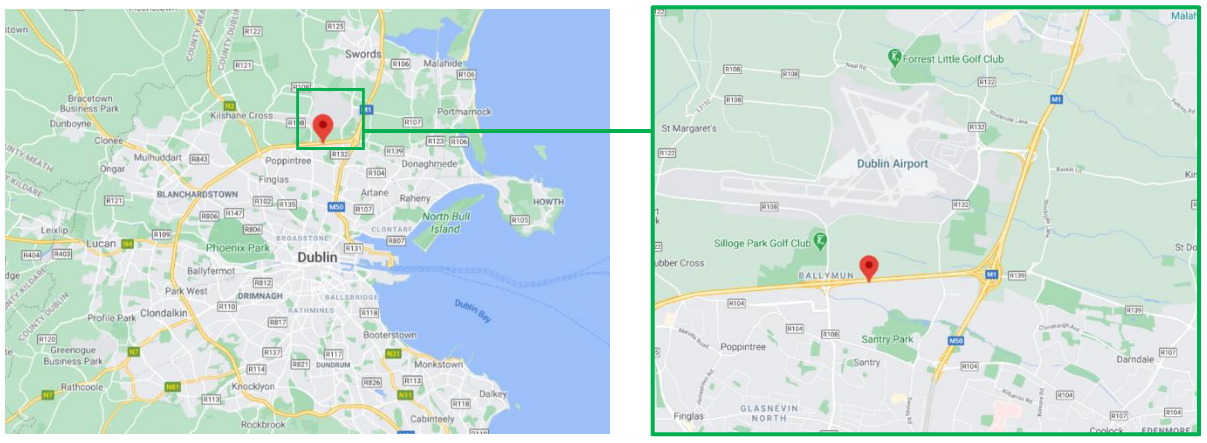


*Figure 4: The Receiver Location Determined by the MATLAB Program, in near the Dublin Airport north of the city. The left map demonstrates the location of the receiver within Dublin whilst the right map shows the local area, where the latitude and longitude of the receiver converged.*

Based on the code output the location of the receiver for *sn* = 3284 was determined as alongside the motorway just south of the airport in Dublin, Ireland (see the map in Figure 4). In order to validate that the MATLAB script was effective in determining these receiver values, it was run with a number of input 4-digit numbers (where *sn = 1234* etc.). All of the calculated receiver locations were distributed around Ireland, suggesting that the program functions as desired alongside the *Receiver.p* program which generates random receiver locations in Ireland only. Additional test locations and their associated input numbers are presented in Appendix A in Table 3.

When originally outputting receiver locations, it was noted that the latitude and longitude of the receiver locations quickly converged using Newton's method. However, the elevation (based on the height *r* in spherical coordinates) and the receiver clock offset did not converge at the same rate. This was due to a mistake in the partial derivation of the function used in the Jacobian, leading to a common pseudorange error that increased exponentially with each iteration. By correctly deriving the partial differentiation $\partial f_i/\partial b$, the correct receiver locations were defined, with all four unknowns (latitude, longitude, elevation, and clock offset) converging.

A solution can also be found more efficiently through less iterations of Newton's method by using localised initial values for input instead of arrays of zeros, although when applied to the MATLAB program developed to calculate receiver locations this does not offer any reasonable benefit over the initial estimates of U, F and J as consisting only of zeros, as the code outputs results almost instantaneously. A situation when more accurate initial estimates would be required would be for a continuous GPS tracking by constantly moving satellites, as aside to the static problem definition investigated in this report.

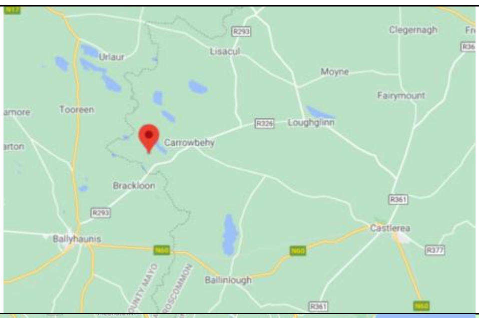
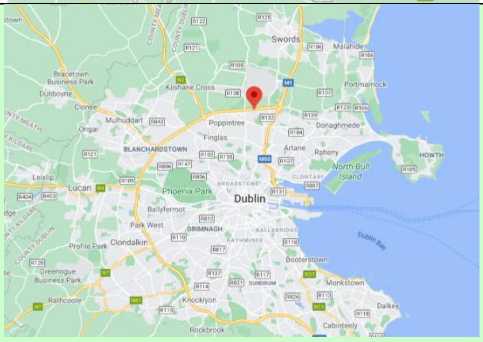
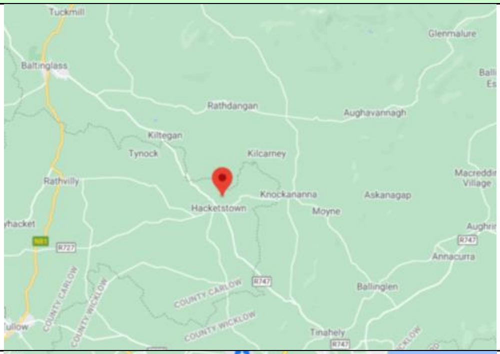
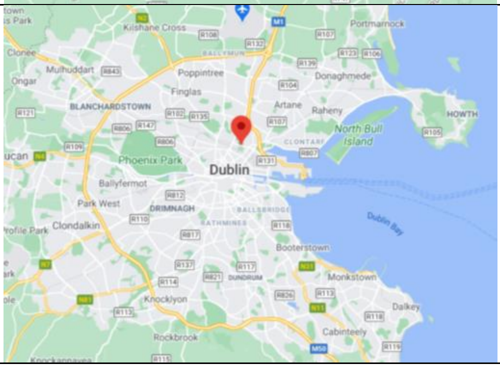
## 5.0 References

1. Ancillary Description Writer's Guide (2013). *Orbit: Definition*. National Aeronautics and Space Administration (NASA) Global Change Master Directory. [http://gcmd.nasa.gov/add/ancillaryguide/platforms/orbit].
2. Dubronov, A.G.; Baranov, Yu N.; Tryastsin, A.P. (2018). *The development of the model of satellite orbital motion*. IOP Conference Series: Materials Science and Engineering. 450, 022033.
3. What-When-How, (2020). *Geocentric Earth-Fixed Coordinate Systems (GPS)*. what-when-how: In Depth Tutorials and Information. [http://what-when-how.com/gps/geocentric-earth-fixed-coordinate-systems-gps/]

# 6.0 Appendices

## 6.1 Appendix A – Table of Receiver Results

**Table 3: Receiver Locations for Different *Receiver.p* Input Numbers**

| SN | Latitude (°) | Longitude (°) | Elevation (m) | Clock Offset (µs) | Receiver Location |
|---|---|---|---|---|---|
| 1234 (ref.) | 53.8072 | -8.7027 | 19.9852 | -1191.52 |  |
| 3284 (AK) | 53.4093 | -6.2557 | 5.8268 | 1518.15 |  |
| 1111 | 52.8716 | -6.5556 | 15.9154 | -1095.55 |  |
| 9999 | 53.3628 | -6.2509 | 10.5660 | 1823.39 |  |

## 6.2 Appendix B – Code Listing

```
%SatNav Assignment - AKusmirek [20203284]

%constants and input:
prompt = "Please enter the last four digits of your student number: ";
sn = input(prompt);
c = 299792458;
meanEarthradius = 6371E3;
satellitealtitudes = 20200E3;
r = meanEarthradius + satellitealtitudes;

%Satellite Data from Receiver:
format long
addpath('D:\1. Alexander\2. University College Dublin\6. Applications of
Space Science\Assignment 3 Satellite Navigation')
SatData = Receiver(sn);
disp("Satellite data received...")
disp("Calculating...")
%SatData now generated for the four satellites

%Converting SatData to Cartesian coordinate system:
%Change spherical degrees to radians
D = [SatData(1,1), SatData(1,2); SatData(2,1), SatData(2,2); SatData(3,1),
SatData(3,2); SatData(4,1), SatData(4,2)];
R = (pi/180).*D;
%Note: for larger problems, this can be done with a "for" loop
%Create array of Cartesian satellite coordinates
xyz1 = [r.*cos(R(1,1)).*cos(R(1,2)), r.*cos(R(1,1)).*sin(R(1,2)),
r.*sin(R(1,1))];
xyz2 = [r.*cos(R(2,1)).*cos(R(2,2)), r.*cos(R(2,1)).*sin(R(2,2)),
r.*sin(R(2,1))];
xyz3 = [r.*cos(R(3,1)).*cos(R(3,2)), r.*cos(R(3,1)).*sin(R(3,2)),
r.*sin(R(3,1))];
xyz4 = [r.*cos(R(4,1)).*cos(R(4,2)), r.*cos(R(4,1)).*sin(R(4,2)),
r.*sin(R(4,1))];
SatCart = [xyz1; xyz2; xyz3; xyz4];
disp("Satellite positions determined (Cartesian).")

%Calculating pseudoranges from times of flight:
tof = [SatData(1,3); SatData(2,3); SatData(3,3); SatData(4,3)]; %measured
time of flight (seconds)
pr = tof.*c; %pseudoranges (metres)

%Array of converted satellite position values [x, y, z, d]:
SatPos = [SatCart, pr];

%Newton's method:
mu = zeros(4,1); %initial vector of variables
F = zeros(4,1); %initial vector of functions
J = zeros(4,4); %initial Jacobian
tol = 1e-4;
disp("Determining receiver location using Newton's method.")
disp("Calculating Jacobians...")

for n = 1:20
    %Calculate function values and Jacobian for current iteration:
    for i = 1:4
        d = (mu(1)-SatPos(i,1)).^2 + (mu(2)-SatPos(i,2)).^2 + (mu(3)-
SatPos(i,3)).^2 - (SatPos(i,4)-mu(4)).^2;
```

```
        F(i) = d;
        J1 = [(2*mu(1)-2*SatPos(1,1)),(2*mu(2)-2*SatPos(1,2)),(2*mu(3)-
2*SatPos(1,3)),(-2*mu(4)+2*SatPos(1,4))];
        J2 = [(2*mu(1)-2*SatPos(2,1)),(2*mu(2)-2*SatPos(2,2)),(2*mu(3)-
2*SatPos(2,3)),(-2*mu(4)+2*SatPos(2,4))];
        J3 = [(2*mu(1)-2*SatPos(3,1)),(2*mu(2)-2*SatPos(3,2)),(2*mu(3)-
2*SatPos(3,3)),(-2*mu(4)+2*SatPos(3,4))];
        J4 = [(2*mu(1)-2*SatPos(4,1)),(2*mu(2)-2*SatPos(4,2)),(2*mu(3)-
2*SatPos(4,3)),(-2*mu(4)+2*SatPos(4,4))];
        J = [J1; J2; J3; J4];
    end

    %Iterate through Newton's method:
    mu = mu - J\F;
end

%Location of receiver in Cartesian coordinates
disp("Receiver location determined (Cartesian).")
recx = mu(1);
recy = mu(2);
recz = mu(3);
reccprerr = mu(4);
recclockoffset = (reccprerr/c)*10.^6; %microseconds

%Converting receiver location to spherical coordinates and degrees
longr = atan2(recy, recx);
latr = atan2(recz, sqrt(recx.^2 + recy.^2));
long = longr.*(180/pi);
lat = latr.*(180/pi);
rr = sqrt(recx.^2 + recy.^2 + recz.^2);
relev = rr - meanEarthradius;

%Code output:
fprintf('Calculated Latitude and Longitude: %f, %f degrees\n', lat, long)
fprintf('Calculated Elevation: %f metres\n', relev)
fprintf('Calculated Receiver Clock Offset: %f microseconds\n',
recclockoffset)
```