Advanced IOT Applications

Dr. T V Prabhakar
Department of Electronic System Engineering
Indian Institute of Science, Bangalore

Lecture – 3
Outdoor localization without GPS – II

So, what will do now is we will try and see the physical code which implements the localisation without GPS. You have seen the bicycle and where the sensors were placed; what were the sensors that were used and all that. So, just now let us summarise and look at the code.

But before you start looking at the code, it is important to understand the philosophy of trying to localise this cycle so that you would have to get very clear. Look at it this way as your driving the bicycle often at very low speed, you hold the handle because of at low speed your unstable. So, the handle keeps shifting all the time.

So, sometimes when you apply brake or your front wheel shifts or when you are moving very slowly, the front wheel sort of moves around perhaps not in a straight line; keep moving a little bit. So, the magnetometer will give you lot of fluctuating values which often mean nothing, you are not changed any direction, but magnetometer is reflecting all that. So, somehow you have to keep in mind that there can be lot of values for magnetometer, but I must have an algorithm, which will sort of eliminate all those values and sort of ensure that there is no change in direction. You would not have that much of a problem with hall sensor; a pulse will come only when there is a wheel rotation.

So, now you can actually start thinking about your algorithm, if there are no pulses coming, but magnetometer continuous to indicate values clearly someone is playing the fool with the handle. It is just moving and then creating a lot of data from the magnetometer which is not going to going to help you.

You can also have a situation where pulses are coming very slowly, but there is a lot of fluctuation in the magnetometer which is clearly an indication that you are at low speed and you may end up with a situation where large deflections can come and you are not even sure whether you have actually done, you have turned left you have turned right and so on.

So, all of this means your algorithm should be robust enough to take care of practical situations of problems of this nature. The code that is written and the code that I would like to share with you is just an overview which will work. But several modifications to this code is up to you; you may want to try and you may want to modify based on your local settings.

We mentioned that you will start with one assumption that you have the last well known GPS coordinate and soon after you will start losing GPS. Quickly you absorb that last lat, long and you will ask your algorithm which is called the localisation without GPS algorithm to kick in and say ‘Hey, I have the last GPS coordinate do something with it quickly’. What is this quickly? You take this last GPS coordinate and pass it through that UTM, routine which I mention to you.

So, you pass it through that you will end up with some X, Y essentially GPS will give you all the latitude and longitude and other height and other related information which often may not be useful. Because in the localisation paradigm that we are talking here we are purely looking at 2D from where ever we have lost to some reasonable distance where other parameters given by GPS may not change significantly. So, that is the assumption. So, then you get back you convert this latitude and longitude to some realistic x and y; keep it with you and say ‘Hey, this is the last X, Y I know for sure where I am from the GPS’.

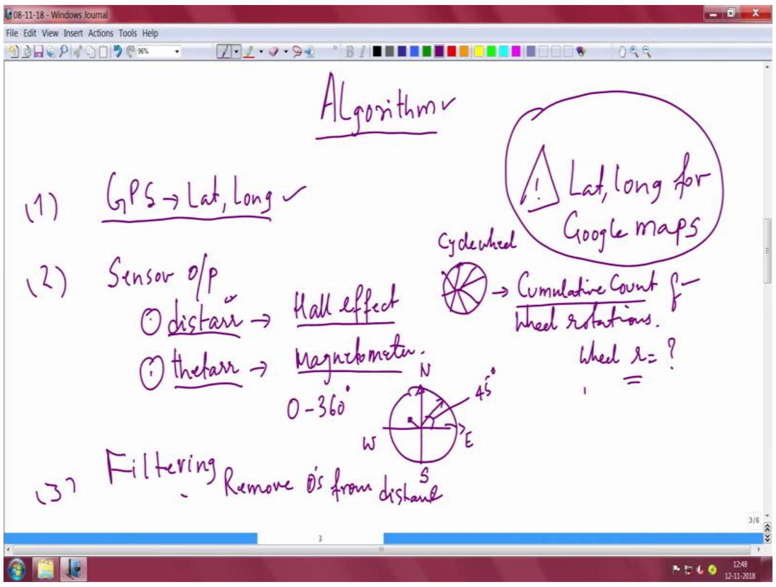
Of course, the map is there with you. The map definitely you have to that is your reference which you have with. It can be offline also. It is a map that is just stored on your computer and which you want to keep mapping back, because you do not have GPS now somehow you have to map it back and say ‘where am I?’, if you want to know you have to consult the map only. That is the only way that you are going to use this application.

So, somehow you have to hold on to that original that I would say x and y coordinates and keep updating this coordinate as and when you get this magnetometer information coupled with the hall sensor information. Hall sensor is perfectly going to tell you that you have done a rotation; that means, circular motion has been converted to some linear distance and that is when you get a pulse and then, you know the circumference and from there you find out linear distance and all that you can do, simple geometry stuff and then you arrive at some linear distance.

Be careful magnetometer is going to be very noisy and as I told you the handle shaking can give you a lot of disturbance. So, you need to do proper thresholding and once you do that you know yes, I have taken the left turn, I have taken a right turn or I have done nothing, I am proceeding straight and so on and so forth. Then, take that x y, now if you ask yourself where I am! What should you do? Take this x y; you have the original X, Y which you got from the UTM conversion of latitude and longitude, add it like an offset back to it and then, put it back on the map. Then, you know exactly where you are. So, this is the core idea that you have to follow if you want to do all of this. Let us see whether this code that we were talking of actually does all what I said as a in a summary.

Let me point you before showing you C code, let me just point you to what all you should look for in the C code because it is just going to give you some C code, some variables and some you know for loops and so on which is perhaps of not much consequence at the moment. So, you must get the high level picture of what is happening.

(Refer Slide Time: 07:52)


Algorithm ✓

(1) GPS → Lat, Long ✓

(2) Sensor o/p
- distarr → Hall effect
- thetarr → Magnetometer. 0 - 360°

(3) Filtering Remove 0's from distance

⚠ Lat, long for Google maps

Cyclewheel → Cumulative Count of Wheel rotations.

Wheel r = ?

08-11-18 - Windows Journal
File Edit View Insert Actions Tools Help
N
45°
E
W
S

So, let me point you above figure. Here is the algorithm of interest that is what we want to do.

First step is as I said get hold of the Lat, long of the last GPS coordinate and I have marked here that Lat, long is required for all Google maps. If you want to put back anything into Google map, you will have to do it with latitude and longitude and therefore, you will have to get hold of conversion which will come to later. So, you have the last latitude longitude step number 1, store it safely and pass it through UTM and store the original X, Y carefully somewhere.

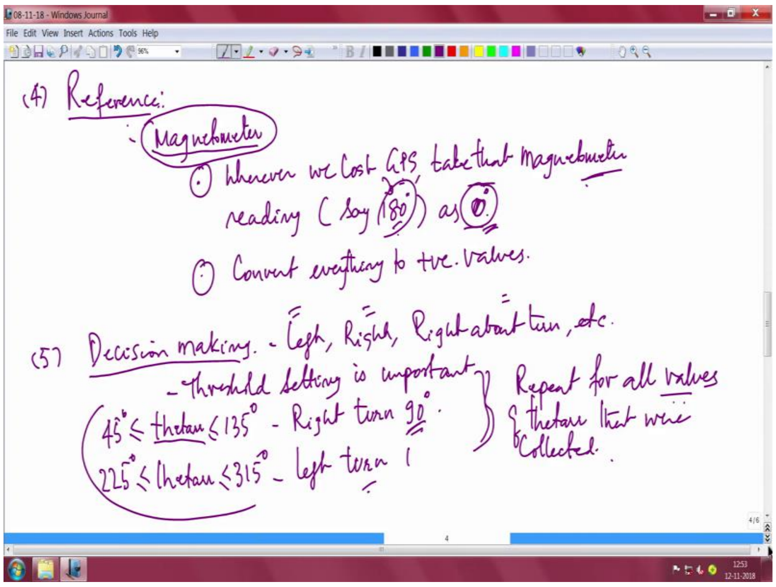
Next step Sensor output. What are the two sensors? You have the Hall Effect sensor and you have the Magnetometer sensor. Two variables you will find in the C-code; one is called the distarr, (distance array essentially) and then the magnetometer thetarr. Essentially the third step will be all. Zeros from distance will have to be removed as well.

Sometimes even the hall-effect sensor will give you some kind of numbers which are quite vague; not just filtering with respect to magnetometer, but also you have to remove the zeros coming from the Hall Effect. Of course, that number will depend on what is the circumference of the cycle and how often it repeats that and some sort of cumulative count which will have to come because of the wheel rotations.

So, I have shown (in above figure) you a cycle wheel and every time a 1 rotation happens you get a pulse. So, you start putting it into this distance array, you count that and then you put the next count which will be the first count plus the next count that is after 2 rotations. Then third one will be the first, second and third and so on and so forth. You go on this way you start cumulatively counting and you actually come to the linear distance that is the key objective. I have written you must know the radius of the wheel and therefore, I expect you to do this calculation and then, find out the linear distance.

Now for magnetometer. We mentioned do you already that it is with respect to earth’s magnetic north that it gives you a number which is also in some degrees. So, you will see that 45 degree means essentially the vehicle or bicycle is moving in the north east direction and so on and so forth.

(Refer Slide Time: 10:53)


(4) Reference:

- Magnetometer
  - Whenever we lost GPS, take that magnetometer reading (say 180°) as 0°.
  - Convert everything to +ve. values.

(5) Decision making. - Left, Right, Right about turn, etc.

- Threshold setting is important

$45^\circ \leq$ thetau $\leq 135^\circ$ - Right turn 90°.

$225^\circ \leq$ thetau $\leq 315^\circ$ - left turn

Repeat for all values of thetau that were collected.

Actually before you do any of that cleaning up process, you have to do this forth one which is where am I you want to know!. So, magnetometer is going to tell you something about where you are with respect to the existing sensor based system. So, you again rely on the magnetometer. You know where you have lost the GPS, exactly there what was the magnetometer reading you take. Supposing it read a value which is 180 in the experiment that you do, it can be anything; it can give you 270.Some value it will give you, take that as 0; then, you start converting convert everything to positive values because negative values does not make sense, you will see that in code and use that starting magnetometer value which you have made it has 0. Use this 180 degree as an offset and start subtracting this offset from the starting point.

So, somehow your magnetometer is keeping track of the direction essentially. So, this is when you say reference, your magnetometer is now calibrated with respect to the GPS and the particular direction in which you are proceeding. So, your starting by saying everything is 0 now and from here I will start tracking the magnetometer accordingly that is what you are saying in step number 4.

Now come to step number 5; left turn, right turn, right about turn all these are very good possibilities. Again you are to be dependent on the magnetometer to get to know where you actually turned. I told you about that problem you can be just moving your handle and creating a situation where the magnetometer is giving you out values after you have done this 0; after you done this 0 calibration its giving you some values. It is not going to make sense if you are not getting pulses, which means all those have to be discarded.

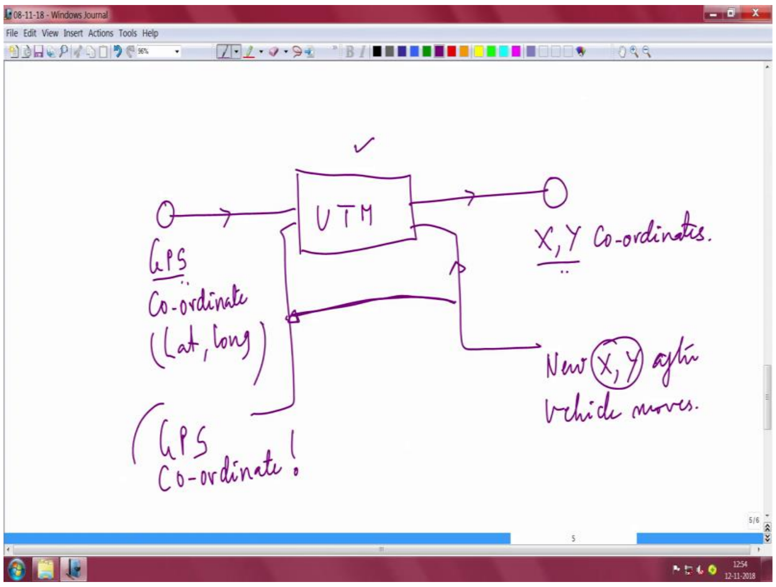
So, this decision making process is a very important thing. Here you have to do some kind of thresholding. In the example that I am going to show you thetarr is between 45 degrees and 135 degrees; then, you say you have done a 90 degree turn if it is between 225 degrees and 315 degrees I'm saying I have done a left turn .

Actually you do the experiment, you do this 0 calibration and then, you turn and see what actually happens and put those values into your code. Also this 180; in my case it is 180 when you started with the last known GPS value and you want your magnetometer to keep track of the direction, it may be something. It may be 40; it maybe 30; it maybe 20, whatever that you make it 0 and then you start from there. That is the point of doing this referencing part.

Now, you are getting an array of magnetometer values. So, you to repeat for all values of thetarr, that you collected except that the second value of the array of magnetometer values that you have will have to be with respect to the first one. So, you take a difference of the first one. The third value in the array will be a difference with respect to the second one. The fourth one will be with respect to the third one and so on and so forth. And therefore, because you have to keep track dynamically.

So, the first one you put thresholding and then thresholding anyway is what you are trying to say about your turns and all that and then remaining parts, you all have to keep doing this by taking the difference of the third, the second with the first and then applying this rule back. You apply this rule back after you take the difference for second values onward, and then, you will definitely arrive at some reasonable X, Y position.

(Refer Slide Time: 15:15)

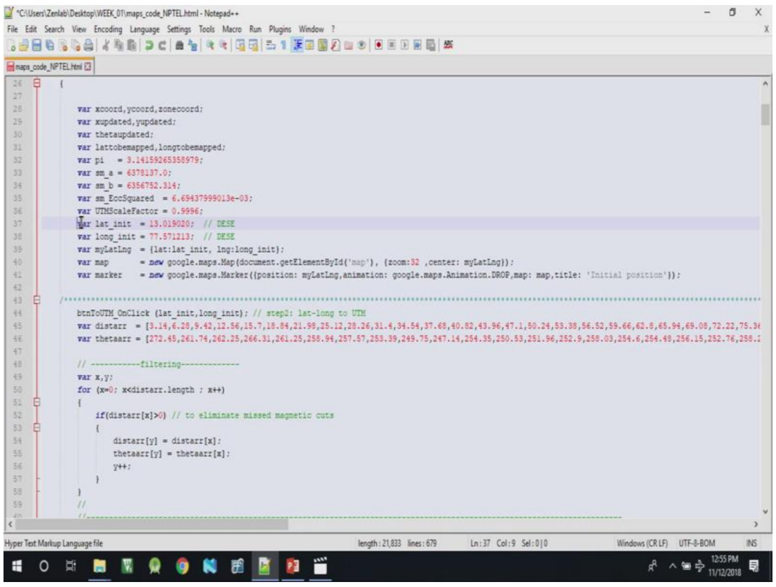

08-11-18 - Windows Journal
File Edit View Insert Actions Tools Help
UTM
GPS Co-ordinate (Lat, long)
X, Y Co-ordinates.
New X, Y after Vehicle moves.
(GPS Co-ordinate!

So, then what you do after that is pretty straight forward. Remember your last known GPS coordinate was the latitude longitude. You have now passes it through this UTM, then you got the original X, Y. I meant the last known X, Y as with respect to GPS and then you did all this wheel rotations with distarr and thetarr; you found out what is the linear distance ,angle and all that and then you arrived at the new X, Y. The new X, Y again we have to pass through this UTM and then you will you will have to get back to the GPS coordinates on the map.

So, that is the key take away. So, point when you do this step, that is get the new X, Y and put it back into pass through the UTM routine and get back the GPS coordinates will actually say this is actually the final step where a human wants to ask the question where am I, when you ask this question, this conversion that happened and it will nicely show you on Google map, the current position even without GPS.

So, that is the nice thing about this little application. So, let us not run through the code. I will show you some C code, Let us point out to some important things with respect to the understanding that we have.

(Refer Slide Time: 17:07)



```
    {

        var xcoord,ycoord,zonecoord;
        var xupdated,yupdated;
        var thetaupdated;
        var lattobemapped,longtobemapped;
        var pi   = 3.14159265358979;
        var sm_a = 6378137.0;
        var sm_b = 6356752.314;
        var sm_EccSquared  = 6.69437999013e-03;
        var UTMScaleFactor = 0.9996;
        var lat_init  = 13.019020;  // DESE
        var long_init = 77.571213;  // DESE
        var myLatLng  = {lat:lat_init, lng:long_init};
        var map       = new google.maps.Map(document.getElementById('map'), {zoom:32 ,center: myLatLng});
        var marker    = new google.maps.Marker({position: myLatLng,animation: google.maps.Animation.DROP,map: map,title: 'Initial position'});

    /*************************************************************************************************************
        btnToUTM_OnClick (lat_init,long_init); // step2: lat-long to UTM
        var distarr  = [3.14,6.28,9.42,12.56,15.7,18.84,21.98,25.12,28.26,31.4,34.54,37.68,40.82,43.96,47.1,50.24,53.38,56.52,59.66,62.8,65.94,69.08,72.22,75.3
        var thetaarr = [272.45,261.74,262.25,266.31,261.25,258.94,257.57,253.39,249.75,247.14,254.35,250.53,251.96,252.9,258.03,254.6,254.48,256.15,252.76,258.2

        // -----------filtering------------
        var x,y;
        for (x=0; x<distarr.length ; x++)
        {
            if(distarr[x]>0) // to eliminate missed magnetic cuts
            {
                distarr[y] = distarr[x];
                thetaarr[y] = thetaarr[x];
                y++;
            }
        }
        //
        //-----------------------------------------------------------------------------------
```

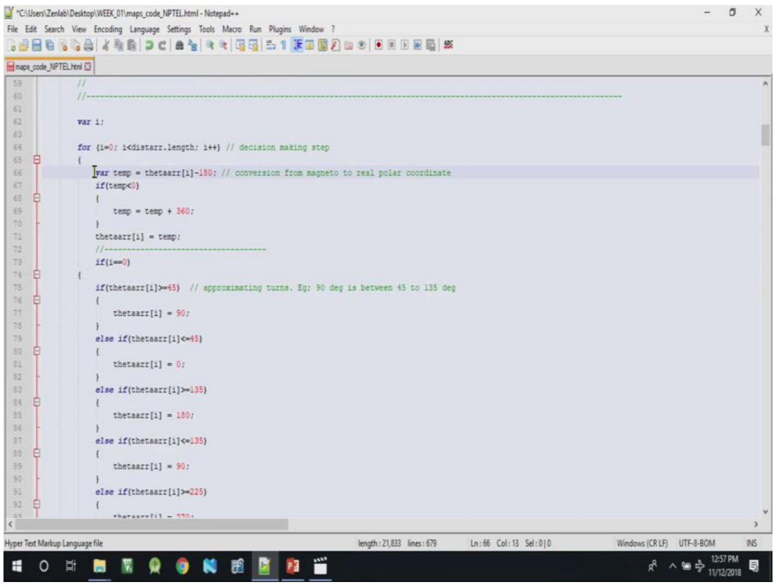
So, let us say I have to now show you the last GPS latitude and longitude. In above figure the last GPS coordinate is commented as DESE (that’s our experiments starting point) so, store this somehow. You have seen that the two variables are mentioned lat_init and long _init.

Now let us go to step two which is the sensor output. There are two variables; distarr and thetarr. One is the Hall Effect sensor and the other is Magnetometer. Look at the Hall Effect which is essentially the distance array, look very carefully there are no zeros there.

You will see a lot of zeros. So, you have to design your own algorithm to get rid of the zeros and put it there. So, that is distance array.

In theta array, is the second one which is telling you the magnetometer values. Here it is telling you different degrees 272.45, 261.74 and so on. So, it is actually giving you all the magnetometer values.

(Refer Slide Time: 18:53)



```
        //
        //------------------------------------------------------------------------------------------------------------------

        var i;

        for (i=0; i<distarr.length; i++) // decision making step
        {
            var temp = thetaarr[i]-180; // conversion from magneto to real polar coordinate
            if(temp<0)
            {
                temp = temp + 360;
            }
            thetaarr[i] = temp;
            //------------------------------------
            if(i==0)
        {
            if(thetaarr[i]>=45)  // approximating turns. Eg: 90 deg is between 45 to 135 deg
            {
                thetaarr[i] = 90;
            }
            else if(thetaarr[i]<=45)
            {
                thetaarr[i] = 0;
            }
            else if(thetaarr[i]>=135)
            {
                thetaarr[i] = 180;
            }
            else if(thetaarr[i]<=135)
            {
                thetaarr[i] = 90;
            }
            else if(thetaarr[i]>=225)
            {
```

Let us look at the next filtering step which is the magnetometer should be made 0. At the point when you have your last GPS coordinate. I mention to you at the point when I lost the GPS, the magnetometer read 180 degrees.

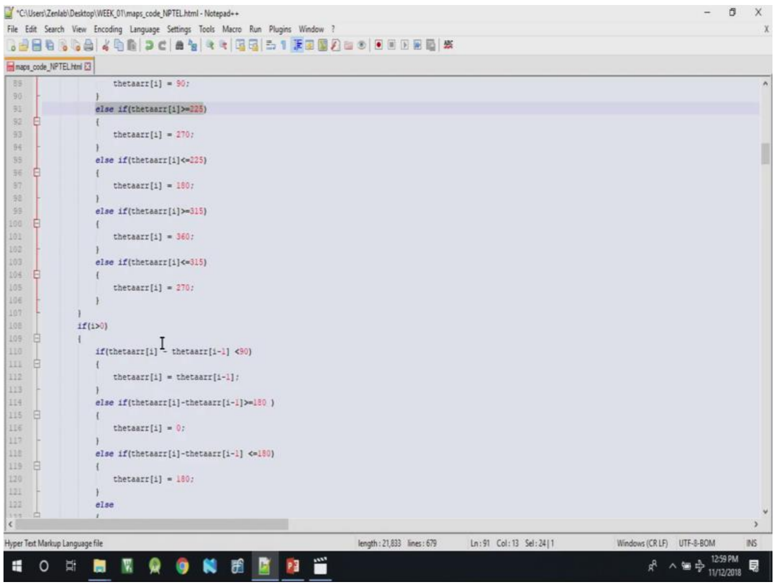
Now, you see what is shown in above figure?

var_temp = thetarr[i] -180

In your case it can be something else. Essentially if you do that, you are making your system back to 0. Everything is nicely done and then you convert into all positive values all that can see that temp+360 and all that is shown here.

So, it is essentially converting everything to positive values. Now is the issue of decision making. Lot of noise coming from the magnetometer; do it very carefully. Set your own threshold; left turn, right turn, right about turn and all that will have to be taken care because you are cycling and then you are moving forward.

(Refer Slide Time: 20:05)

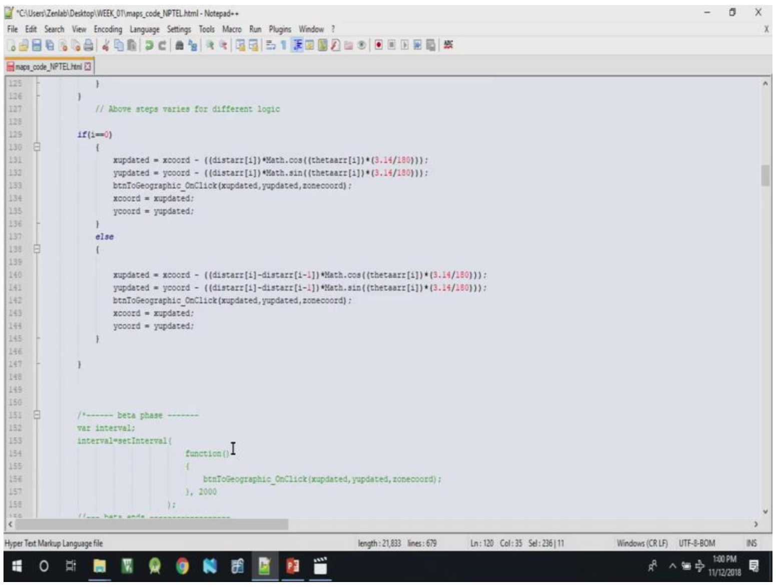


```
89              thetaarr[i] = 90;
90          }
91          else if(thetaarr[i]>=225)
92          {
93              thetaarr[i] = 270;
94          }
95          else if(thetaarr[i]<=225)
96          {
97              thetaarr[i] = 180;
98          }
99          else if(thetaarr[i]>=315)
100         {
101             thetaarr[i] = 360;
102         }
103         else if(thetaarr[i]<=315)
104         {
105             thetaarr[i] = 270;
106         }
107     }
108     if(i>0)
109     {
110         if(thetaarr[i] - thetaarr[i-1] <90)
111         {
112             thetaarr[i] = thetaarr[i-1];
113         }
114         else if(thetaarr[i]-thetaarr[i-1]>=180 )
115         {
116             thetaarr[i] = 0;
117         }
118         else if(thetaarr[i]-thetaarr[i-1] <=180)
119         {
120             thetaarr[i] = 180;
121         }
122         else
```

So, there can be left turns and right turns. Threshold setting is important. Look at thetarr in the above figure, line number 79, it should lie between 45 degree and 135 degrees. To tell you that it is a right turn of 90 degree right turn. Now let us take at left turn. Theta array should be between 225 degrees and 315 degrees, (in line number 103).That is a left turn.

So, again this decision making is what we have found for the terrain that we are trying to sort of find the location without GPS, but you will have to try on your own for your on terrain and for your own localisation application.

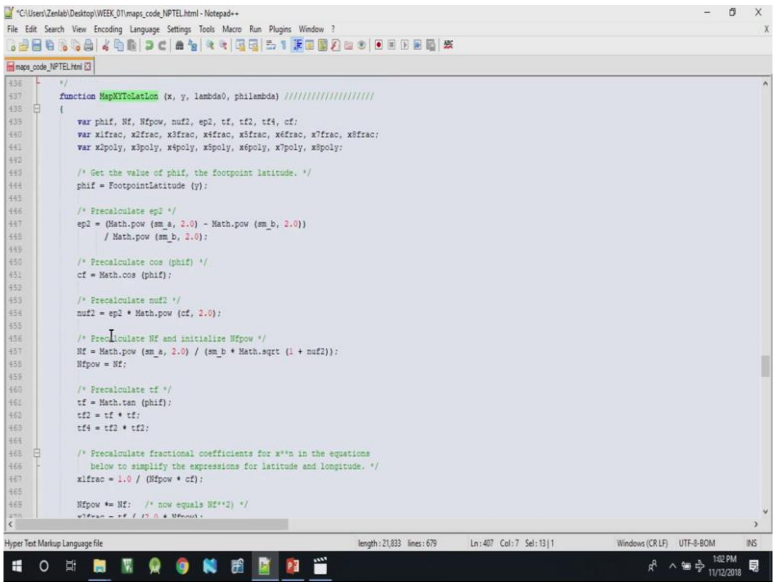
(Refer Slide Time: 21:37)



```
            }
        }
            // Above steps varies for different logic

        if(i==0)
            {
                xupdated = xcoord - ((distarr[i])*Math.cos((thetaarr[i])*(3.14/180)));
                yupdated = ycoord - ((distarr[i])*Math.sin((thetaarr[i])*(3.14/180)));
                btnToGeographic_OnClick(xupdated,yupdated,zonecoord);
                xcoord = xupdated;
                ycoord = yupdated;
            }
            else
            {

                xupdated = xcoord - ((distarr[i]-distarr[i-1])*Math.cos((thetaarr[i])*(3.14/180)));
                yupdated = ycoord - ((distarr[i]-distarr[i-1])*Math.sin((thetaarr[i])*(3.14/180)));
                btnToGeographic_OnClick(xupdated,yupdated,zonecoord);
                xcoord = xupdated;
                ycoord = yupdated;
            }

        }



        /*------ beta phase -------
        var interval;
        interval=setInterval(
                                function()
                                {
                                    btnToGeographic_OnClick(xupdated,yupdated,zonecoord);
                                }, 2000
                            );
```

You should be able to essentially get hold of the X, Y coordinates from the last GPS coordinate that you have, there is the next is the new X, Y which we have to convert back into GPS.

(Refer Slide Time: 22:38)

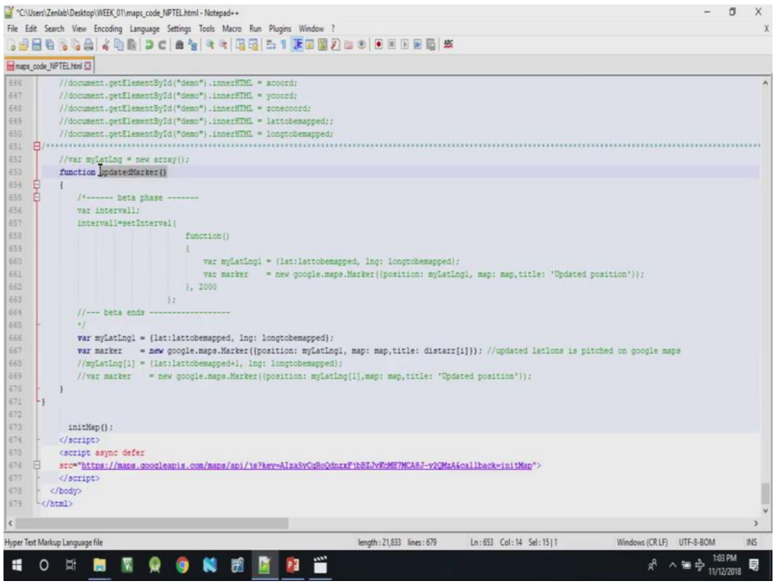


```
    */
    function MapXYToLatLon (x, y, lambda0, philambda) ////////////////////
    {
        var phif, Nf, Nfpow, nuf2, ep2, tf, tf2, tf4, cf;
        var x1frac, x2frac, x3frac, x4frac, x5frac, x6frac, x7frac, x8frac;
        var x2poly, x3poly, x4poly, x5poly, x6poly, x7poly, x8poly;

        /* Get the value of phif, the footpoint latitude. */
        phif = FootpointLatitude (y);

        /* Precalculate ep2 */
        ep2 = (Math.pow (sm_a, 2.0) - Math.pow (sm_b, 2.0))
              / Math.pow (sm_b, 2.0);

        /* Precalculate cos (phif) */
        cf = Math.cos (phif);

        /* Precalculate nuf2 */
        nuf2 = ep2 * Math.pow (cf, 2.0);

        /* Precalculate Nf and initialize Nfpow */
        Nf = Math.pow (sm_a, 2.0) / (sm_b * Math.sqrt (1 + nuf2));
        Nfpow = Nf;

        /* Precalculate tf */
        tf = Math.tan (phif);
        tf2 = tf * tf;
        tf4 = tf2 * tf2;

        /* Precalculate fractional coefficients for x**n in the equations
           below to simplify the expressions for latitude and longitude. */
        x1frac = 1.0 / (Nfpow * cf);

        Nfpow *= Nf;   /* now equals Nf**2) */
```

In MapLatLonToXY, the first part that is you take the map coordinates latitude and longitude and convert it to X, Y; this part is show in above figure. Now you can also do map X, Y to lat long, which is shown in the below figure.

So, you can see both the routines are shown and I encourage you to look at these application and then, you will be able to essentially localise quite accurately provided you do all these steps which I broadly discuss mentioned.

(Refer Slide Time: 22:50)



```
    //document.getElementById("demo").innerHTML = xcoord;
    //document.getElementById("demo").innerHTML = ycoord;
    //document.getElementById("demo").innerHTML = zonecoord;
    //document.getElementById("demo").innerHTML = lattobemapped;;
    //document.getElementById("demo").innerHTML = longtobemapped;
/*****************************************************************************************************************
    //var myLatLng = new array();
    function updatedMarker()
    {
        /*------ beta phase -------
        var intervall;
        intervall=setInterval(
                                function()
                                {
                                    var myLatLngl = {lat:lattobemapped, lng: longtobemapped};
                                    var marker    = new google.maps.Marker({position: myLatLngl, map: map,title: 'Updated position'});
                                }, 2000
                            );
        //--- beta ends ------------------
        */
        var myLatLngl = {lat:lattobemapped, lng: longtobemapped};
        var marker    = new google.maps.Marker({position: myLatLngl, map: map,title: distarr[i]}); //updated latlons is pitched on google maps
        //myLatLng[1] = {lat:lattobemapped+1, lng: longtobemapped};
        //var marker    = new google.maps.Marker({position: myLatLng[1],map: map,title: 'Updated position'});
    }
}

      initMap();
    </script>
    <script async defer
    src="https://maps.googleapis.com/maps/api/js?key=AIzaSyCqRoQdnzxFjbBZJvKoMH7MCA8J-v2QMzA&callback=initMap">
    </script>
  </body>
</html>
```

*C:\Users\Zenlab\Desktop\WEEK_01\maps_code_NPTEL.html - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
maps_code_NPTEL.html
Hyper Text Markup Language file
length : 21,833 lines : 679
Ln : 653 Col : 14 Sel : 15 | 1
Windows (CR LF) UTF-8-BOM INS
1:03 PM 11/12/2018

So, in summary you need those nice little what shall I say pin up board markers. So, you might have seen that equivalent in electronic form on Google maps.

So, that is this function that you see here in the above figure is the updated marker function which will just take the converted lat long from this UTM application where you converted X, Y to lat long and then, call this function updatedMarker with a simple Google API call, you should be able to put back that pin on the map that is what we saw in the slides as well. You saw a set of pins and those pins essentially are from this part of the function call.

So, hopefully you can try this and then improve on this existing code which will be made in open source.