Unscented Kalman Filters

Here are the rubrics criteria :

Compiling:

Code must compile without errors with `cmake` and `make`.

Code compiles correctly with cmake, make in my MacBook Pro. It also compiles with XCode 9.0 which has been used for development.

```
MacBook-Pro-de-Paco-3:build paco$ cmake ..
-- The C compiler identification is AppleClang 9.0.0.9000037
-- The CXX compiler identification is AppleClang 9.0.0.9000037
-- Check for working C compiler:
/Applications/Xcode.app/Contents/Developer/Toolchains/XcodeDefault.xctoolch
ain/usr/bin/cc
-- Check for working C compiler:
/Applications/Xcode.app/Contents/Developer/Toolchains/XcodeDefault.xctoolch
ain/usr/bin/cc -- works
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Detecting C compile features
-- Detecting C compile features - done
-- Check for working CXX compiler:
/Applications/Xcode.app/Contents/Developer/Toolchains/XcodeDefault.xctoolch
ain/usr/bin/c++
-- Check for working CXX compiler:
/Applications/Xcode.app/Contents/Developer/Toolchains/XcodeDefault.xctoolch
ain/usr/bin/c++ -- works
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Configuring done
-- Generating done
-- Build files have been written to: /Users/paco/Documents/Documents -
MacBook Pro de Paco/Udacity Car Projects 2/Project 2-2/build
MacBook-Pro-de-Paco-3:build paco$ make
Scanning dependencies of target UnscentedKF
[ 25%] Building CXX object CMakeFiles/UnscentedKF.dir/src/ukf.cpp.o
[ 50%] Building CXX object CMakeFiles/UnscentedKF.dir/src/main.cpp.o
[ 75%] Building CXX object CMakeFiles/UnscentedKF.dir/src/tools.cpp.o
[100%] Linking CXX executable UnscentedKF
ld: warning: directory not found for option '-
L/usr/local/Cellar/libuv/1.11.0/lib'
[100%] Built target UnscentedKF
MacBook-Pro-de-Paco-3:build paco$ ./UnscentedKF
Listening to port 4567
Connected!!!
```

And the simulator connects and runs correctly.

Accuracy

For the new data set, your algorithm will be run against "obj_pose-laser-radar-synthetic-input.txt". We'll collect the positions that your algorithm outputs and compare them to ground truth data. Your px, py, vx, and vy RMSE should be less than or equal to the values [.09, .10, .40, .30].

I understand that we are working with the new data set and that simulator **Dataset 1** is the same as obj_pose-laser-radar-synthetic-input.txt as it gives me exactly the same RMSE's.

My final parameters for std_a and std_yawdd are 0.6 and 0.6 and they give me following RMSE's:

RMSE			
x	**y**	**vx**	**vy**
0.0617	0.0827	0.3170	0.2295

Well below the required values of 0.09, 0.10, 0.4 and 0.3.

Follows the correct Algorithm

Your algorithm should use the first measurements to initialize the state vectors and covariance matrices.

It is done in the

```
void UKF::Initialize(MeasurementPackage measurement_pack)
```

function that is called from:

```
UKF::ProcessMeasurement(MeasurementPackage meas_package)

if (!is_initialized_){
        Initialize(meas_package);
        return;
    }
```

The function fills x and y values in the state vector and computes R inicialization with a Identity matrix but changing the values of (0,0) and (1,1) elements to :

- In Lidar Initialization squared std_laspx and std_laspy values
- In Radar initialization same values as used in the EKF project that try to estimate errors in x,y from Radar values. It is explained in the EKF project.

Upon receiving a measurement after the first, the algorithm should predict object position to the current timestep and then update the prediction using the new measurement.

The code is in

```
void UKF::ProcessMeasurement(MeasurementPackage meas_package)
```

function:

```
        std::tie(x_, P_) = Prediction(delta_t);
        std::tie(x_, P_, nis) = UpdateLidar(meas_package, x_, P_, R_laser_, H_, n_x_);
```

and

```
        std::tie(x_, P_) = Prediction(delta_t);
        std::tie(x_, P_, nis) = UpdateRadar(meas_package, x_, P_, Xsig_pred_, weights_,
R_radar_, n_x_, n_z_, n_aug_);
```

Your algorithm sets up the appropriate matrices given the type of measurement and calls the correct measurement function for a given sensor type.

As you see in the previous code Prediction results are used in the Update functions.

Code Efficiency

Code is quite functional. Most of the functions just use the values in the parameters and return values either directly or as a couple.

Angle normalization has been moved to a Normalization method of UKF:

```
double UKF::Normalize(double value){
    return atan2(sin(value), cos(value));

}
```

Which helps in readability of code.

Additional Code

Some of the functions in UKF and in main include flags to print NIS data, estimated and truth data.

Main function may be used with the **use_file** flag

```
int main()
{

    bool print_data = false;
    bool print_nis = false;
    bool use_file = false;
```

which instead of using the simulator reads data from a specific file.

To not modify the main methods I added :

```
VectorXd process_file(UKF &ukf,Tools &tools,vector<VectorXd>
&estimations,vector<VectorXd> &ground_truth,string filename, double std_a, double
std_dd, bool p_nis)
```

Function which does more or less the same but reading the data from a file.

In main you may set std_a and std_yawdd minimum, maximum and step values (lines 153 to 164)

```
        double std_a_min = 0.2;
        double std_a_max = 1.0;
        double std_dd_min = 0.5;
        double std_dd_max = 1.5;

        double step_a = 0.1;
        double step_dd = 0.1;
```

```
double min_r = 999999999999;
VectorXd min_RMSE;
double min_a = 99999;
double min_d = 9999;
```

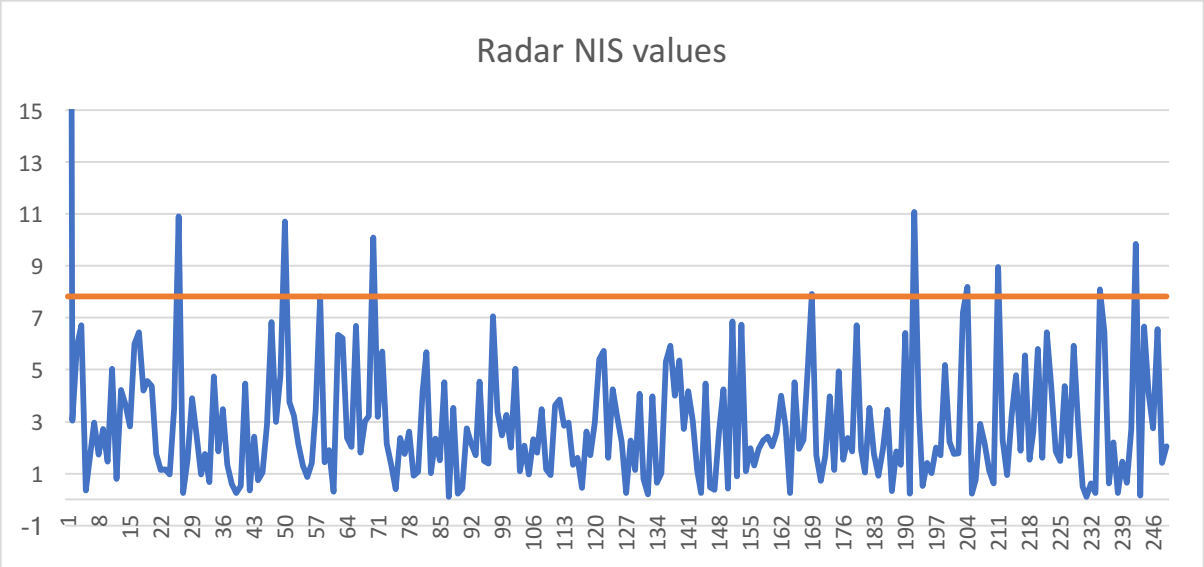
and the program loops over all values executing the process_file function and recording the RMSE's and flagging the correct combinations and the **best one** although the meaning of best means minimizing

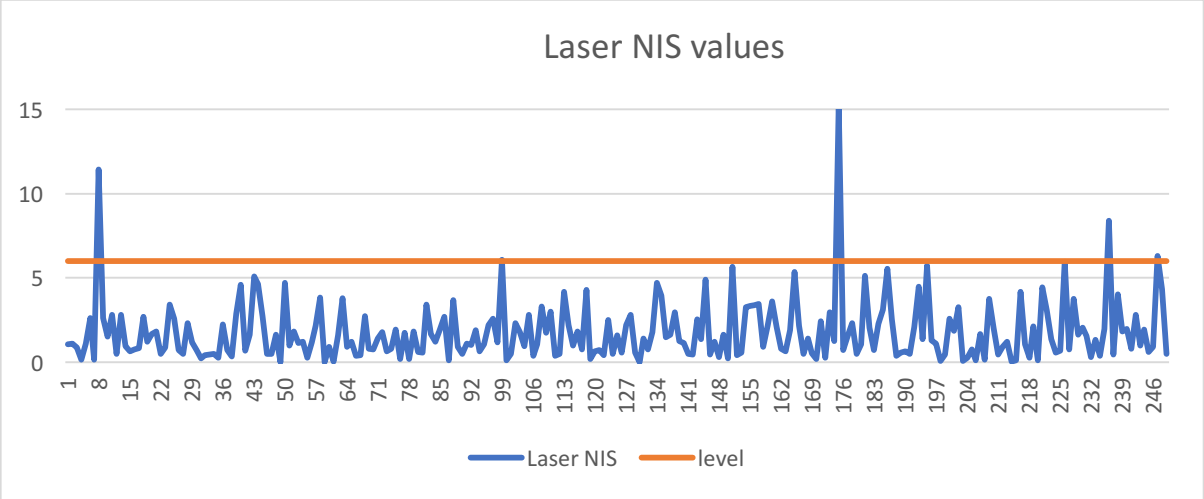
```
r = abs(RMSE[0]) + abs(RMSE[1]) + abs(RMSE[2]) + abs(RMSE[3]);
```

Results

I collected extensive data from the different runs, which are included in the include **data.xlsx** Excel document

NIS values for Radar and Laser:



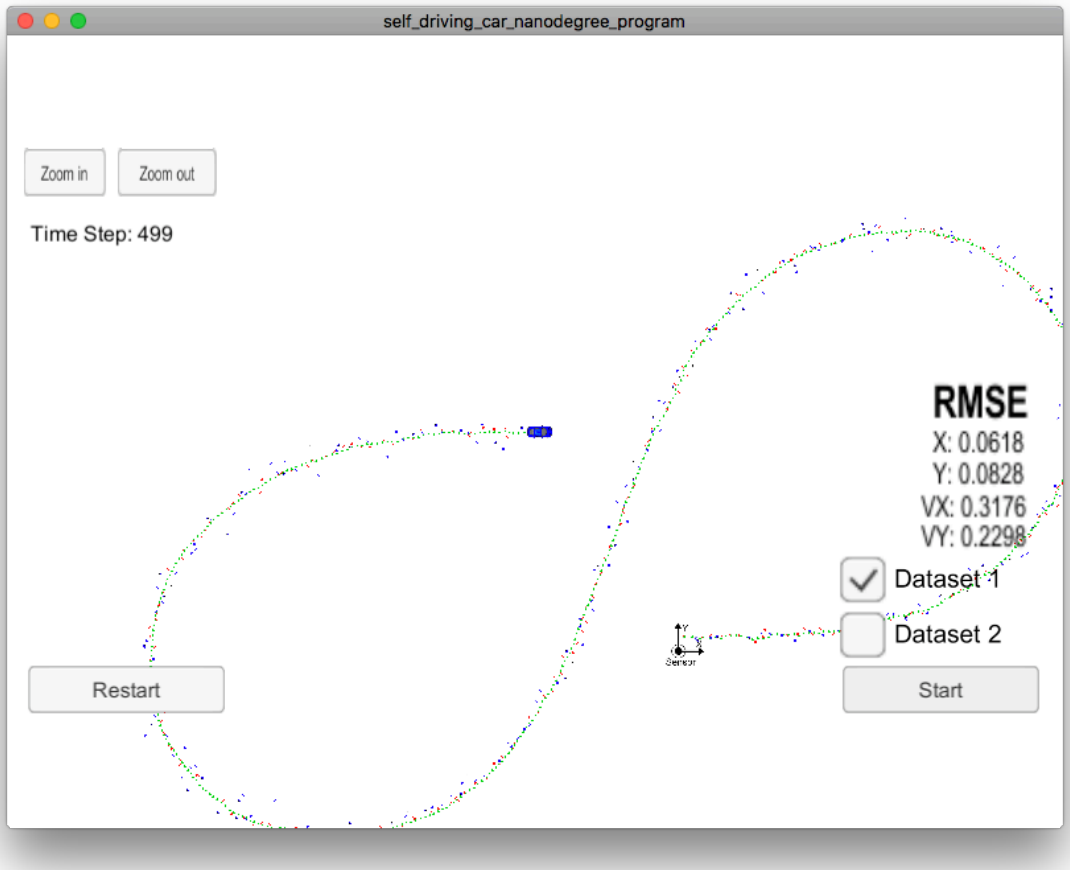


Comparison of Lidar, Radar, Radar and Lidar and EKF filters :

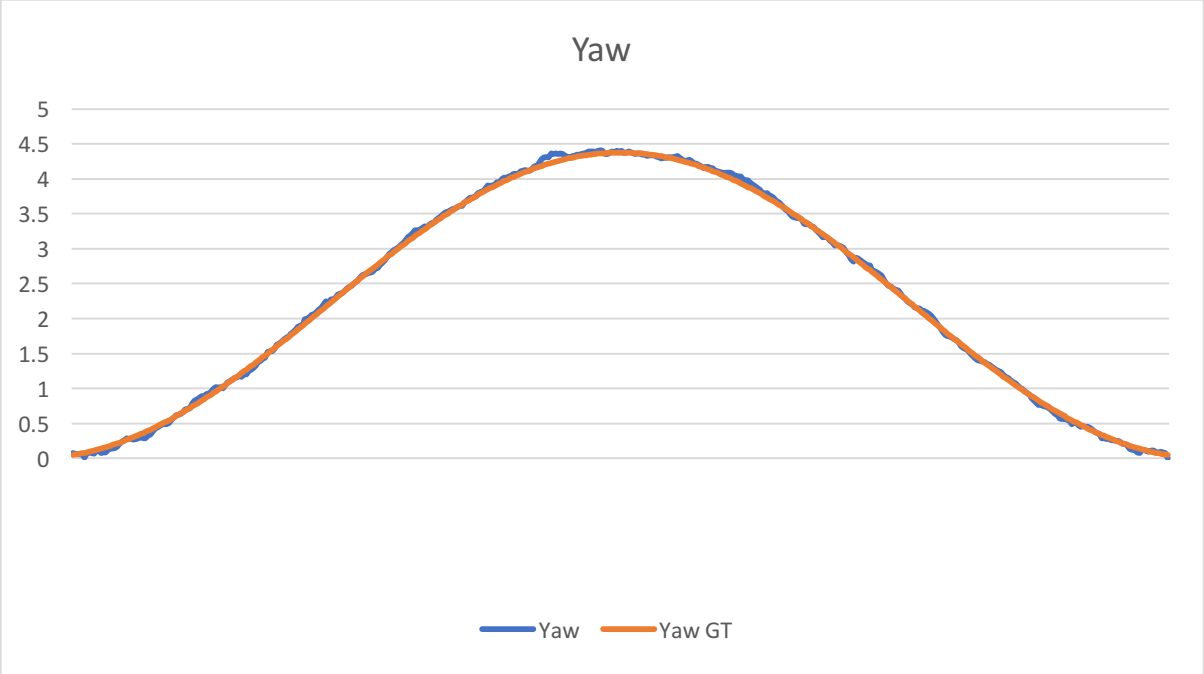
			RMSE			
	std_a	std_yawdd	x	y	vx	vy
Lidar + Radar	0.6	0.6	0.0617	0.0827	0.3170	0.2295
Lidar	0.3	0.5	0.1653	0.1467	0.4762	0.2259
Radar	0.5	0.5	0.2051	0.2257	0.3504	0.1978
EKF Lase & radar			0.0974	0.0856	0.4676	0.4346

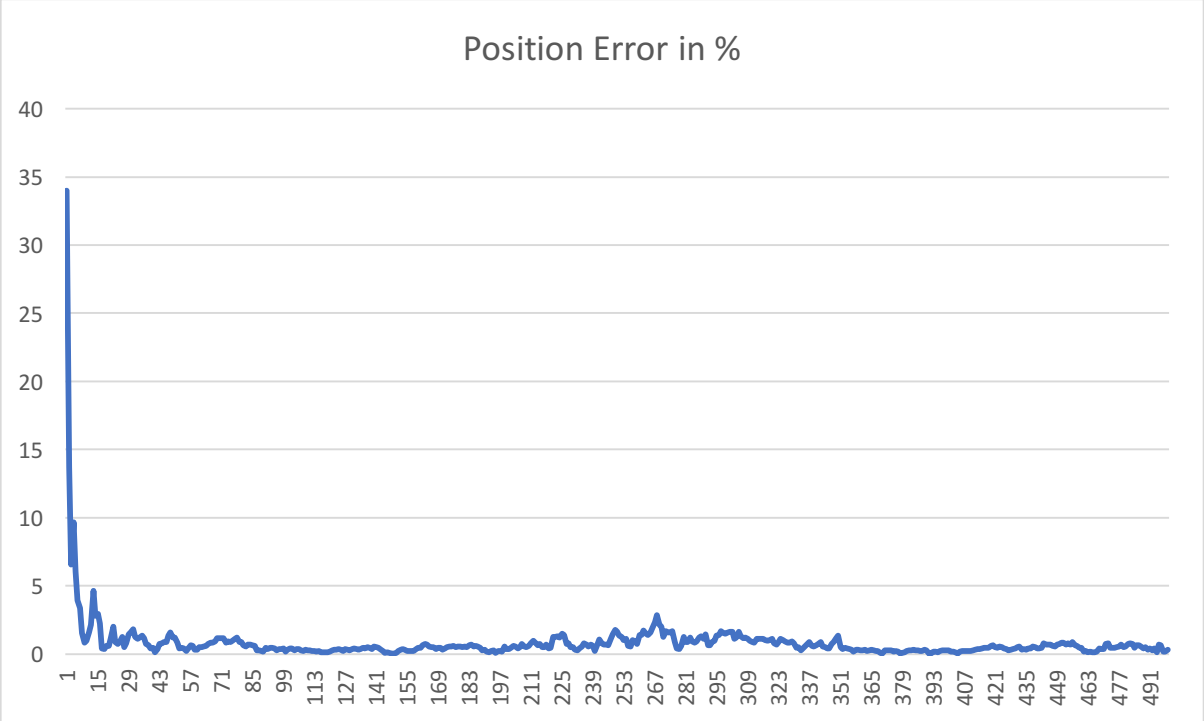
Lidar + Radar UKF is better in every value and only in y values EKF goes near.

Image of the run in the Simulator



Yaw error and Position error





		RMSE values for different values of parameters			
std_a	std_yawdd	x	y	vx	vy
0.2	0.5	0.0618	0.0995	0.3230	0.2379
0.2	0.6	0.0611	0.1001	0.3236	0.2399
0.2	0.7	0.0607	0.1006	0.3248	0.2427
0.2	0.8	0.0605	0.1011	0.3264	0.2460
0.2	0.9	0.0604	0.1015	0.3282	0.2494
0.2	1	0.0603	0.1019	0.3302	0.2530
0.2	1.1	0.0603	0.1022	0.3321	0.2566
0.2	1.2	0.0603	0.1025	0.3341	0.2602
0.2	1.3	0.0603	0.1028	0.3361	0.2638
0.2	1.4	0.0604	0.1030	0.3381	0.2673
0.3	0.5	0.0592	0.0902	0.3193	0.2325
0.3	0.6	0.0584	0.0908	0.3195	0.2337
0.3	0.7	0.0580	0.0913	0.3204	0.2360
0.3	0.8	0.0595	0.0922	0.3245	0.2356
0.3	0.9	0.0592	0.0926	0.3263	0.2391
0.3	1	0.0575	0.0924	0.3250	0.2451
0.3	1.1	0.0575	0.0928	0.3268	0.2484
0.3	1.2	0.0575	0.0931	0.3286	0.2517
0.3	1.3	0.0575	0.0933	0.3304	0.2550
0.3	1.4	0.0575	0.0936	0.3322	0.2583
0.4	0.5	0.0599	0.0859	0.3178	0.2301
0.4	0.6	0.0592	0.0864	0.3178	0.2310
0.4	0.7	0.0588	0.0869	0.3185	0.2329
0.4	0.8	0.0586	0.0873	0.3197	0.2354
0.4	0.9	0.0584	0.0877	0.3211	0.2382
0.4	1	0.0583	0.0880	0.3227	0.2413
0.4	1.1	0.0583	0.0883	0.3244	0.2444
0.4	1.2	0.0583	0.0886	0.3261	0.2476
0.4	1.3	0.0583	0.0889	0.3278	0.2507
0.4	1.4	0.0583	0.0891	0.3295	0.2539
0.5	0.5	0.0612	0.0836	0.3173	0.2292
0.5	0.6	0.0605	0.0840	0.3171	0.2298
0.5	0.7	0.0601	0.0845	0.3177	0.2315
0.5	0.8	0.0598	0.0849	0.3188	0.2338
0.5	0.9	0.0597	0.0853	0.3201	0.2365
0.5	1	0.0596	0.0856	0.3216	0.2394
0.5	1.1	0.0596	0.0859	0.3232	0.2424
0.5	1.2	0.0595	0.0862	0.3248	0.2454
0.5	1.3	0.0595	0.0865	0.3265	0.2485
0.5	1.4	0.0595	0.0868	0.3281	0.2516
0.6	0.5	0.0624	0.0822	0.3173	0.2291

0.6	0.6	0.0617	0.0827	0.3170	0.2295
0.6	0.7	0.0613	0.0831	0.3175	0.2311
0.6	0.8	0.0611	0.0835	0.3185	0.2332
0.6	0.9	0.0609	0.0839	0.3198	0.2358
0.6	1	0.0609	0.0842	0.3212	0.2385
0.6	1.1	0.0608	0.0845	0.3227	0.2415
0.6	1.2	0.0608	0.0848	0.3243	0.2444
0.6	1.3	0.0608	0.0851	0.3259	0.2474
0.6	1.4	0.0608	0.0854	0.3275	0.2504
0.7	0.5	0.0635	0.0814	0.3176	0.2294
0.7	0.6	0.0628	0.0818	0.3172	0.2297
0.7	0.7	0.0624	0.0822	0.3176	0.2311
0.7	0.8	0.0622	0.0826	0.3186	0.2332
0.7	0.9	0.0621	0.0830	0.3198	0.2356
0.7	1	0.0620	0.0834	0.3211	0.2383
0.7	1.1	0.0619	0.0837	0.3226	0.2411
0.7	1.2	0.0619	0.0840	0.3241	0.2440
0.7	1.3	0.0619	0.0842	0.3256	0.2470
0.7	1.4	0.0619	0.0845	0.3272	0.2499
0.8	0.5	0.0644	0.0809	0.3180	0.2301
0.8	0.6	0.0638	0.0813	0.3176	0.2303
0.8	0.7	0.0634	0.0817	0.3180	0.2316
0.8	0.8	0.0632	0.0821	0.3189	0.2335
0.8	0.9	0.0630	0.0825	0.3200	0.2359
0.8	1	0.0630	0.0828	0.3213	0.2385
0.8	1.1	0.0629	0.0831	0.3227	0.2412
0.8	1.2	0.0629	0.0834	0.3242	0.2441
0.8	1.3	0.0629	0.0837	0.3257	0.2469
0.8	1.4	0.0629	0.0839	0.3272	0.2498
0.9	0.5	0.0653	0.0805	0.3187	0.2310
0.9	0.6	0.0646	0.0809	0.3182	0.2310
0.9	0.7	0.0643	0.0813	0.3185	0.2322
0.9	0.8	0.0641	0.0817	0.3193	0.2341
0.9	0.9	0.0639	0.0821	0.3204	0.2364
0.9	1	0.0638	0.0824	0.3217	0.2389
0.9	1.1	0.0638	0.0827	0.3230	0.2416
0.9	1.2	0.0638	0.0830	0.3245	0.2444
0.9	1.3	0.0637	0.0833	0.3259	0.2472
0.9	1.4	0.0637	0.0836	0.3274	0.2500
1	0.5	0.0661	0.0804	0.3194	0.2320
1	0.6	0.0654	0.0807	0.3189	0.2319
1	0.7	0.0651	0.0811	0.3191	0.2331
1	0.8	0.0649	0.0815	0.3199	0.2349

1	0.9	0.0647	0.0819	0.3210	0.2371
1	1	0.0646	0.0822	0.3222	0.2396
1	1.1	0.0646	0.0825	0.3235	0.2422
1	1.2	0.0645	0.0828	0.3249	0.2449
1	1.3	0.0645	0.0831	0.3263	0.2477
1	1.4	0.0645	0.0833	0.3277	0.2505

Catch the Run Away Car with UKF

I did the Catch the Run Away project. It more or less works but have some questions about the meaning of the values I send to the simulator. Angle is clear but distance….

Well, It works more than half the times. Algorithm is quite easy, I integrate the motion equations for the run away car and use the angle and angle speed given by the UKF to estimate the new position.

Instead of solving directly, just loop for different values of t to find one where the distance is equal to my speed by the time.

Then just adjust my angle and I go there.

First time works very bad but it gets better. Probably due to the initial values of the UKF. Unfortunately I don't understand the distance parameter I send. I would have preferred an speed.

Movement equations to estimate movement of the run away car from the current position are :

$$deltax = \int v\,Cos(phidot * t + phi)dt = v * (\frac{Cos(phidot * t)Sin(phi)}{phidot} + \frac{Cos(phi)Sin(phidot * t)}{phidot})$$

with an initial value of v * Sin(phi)/phidot

and

$$deltay = \int v\,Sin(phidot * t + phi)dt = v * (\frac{Cos(phi)Cos(phidot * t)}{phidot} + \frac{Sin(phi)Sin(phidot * t)}{phidot})$$

and an initial value of v*Cos(phi)/phidot

So the code to determine where I should point is :

```
            if (fabs(target_phidot) > 0.001 && count > 10){

                double step = 0.05;  // Comprovar
                double t = 0.0;
                while(true){
                    t += step;
                    double new_x = target_x + target_v * ((cos(target_phidot *
t)*sin(target_phi)/target_phidot + cos(target_phi)*sin(target_phidot*t)/target_phidot)-
sin(target_phi)/target_phidot);
```

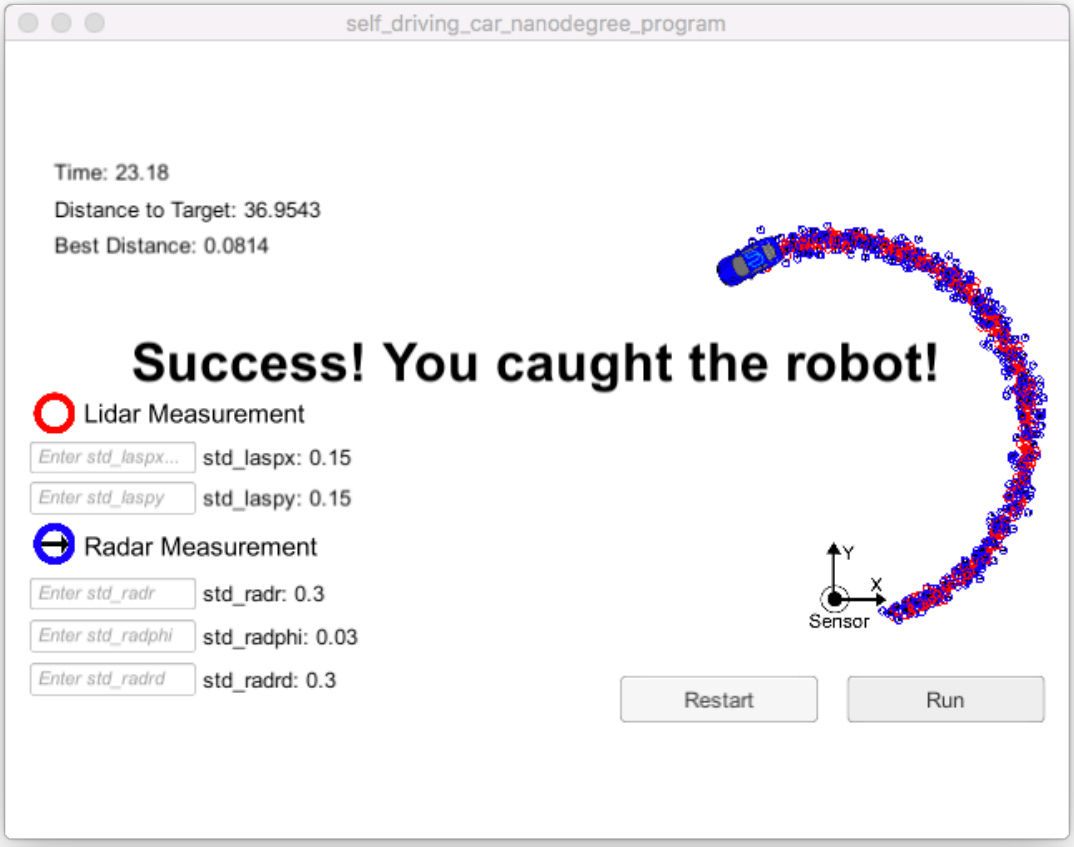
```
                    double new_y = target_y + target_v * ((-
cos(target_phi)*cos(target_phidot * t)/target_phidot +
sin(target_phi)*sin(target_phidot*t)/target_phidot)-cos(target_phi)/target_phidot);
                    double my_r = my_v * t;

                    double new_distance = sqrt(pow(new_x-hunter_x,2) + pow(new_y-
hunter_y,2)) - my_r;

                    if (new_distance <= -1.5){  // This is a magic number. It hould be
0, -1.3 is ok.
                        double new_heading_to_target = atan2(new_y - hunter_y, new_x -
hunter_x);
                        new_heading_to_target = ukf.Normalize(new_heading_to_target);
                        heading_difference = new_heading_to_target - hunter_heading;
                        distance_difference = 2.0; //my_r/5

                        break;
                    }else if (t > 100){
                        cout << "Ouch not found point" << endl;
                        break;
                    }
                }
            }
```

And here is the proof



The program is in the CATCH folder and also compiles with cmake, make.

Curiously first time It tries to get the car it fails with a big error and needs some time to realize it fails.