U D A C I T Y

Unscented Kalman Filters

REVIEW
CODE REVIEW
HISTORY

Meets Specifications

Congratulations. You have finished implementing the Unscented Kalman Filter. It is a difficult project so you should feel proud for successfully completing it. 👍🏼

There are just some minor issues to be corrected. I have left suggestions on how you can approach fixing them.

This is a wonderful submission and all requirements have been met. Good luck on the next project. And keep doing an awesome job.

Compiling

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

Given that we've made CMakeLists.txt as general as possible, it's recommended that you do not change it unless you can guarantee that your changes will still compile on any platform.

Excellent. The code compiles and links without incident. You have done a good job here.

Accuracy

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].

Well done. When the filter is run against the `Dataset 1` in the simulator at my end the RMSE results are in the range we are looking for, which indicates that the UKF filter implementation is likely correct.
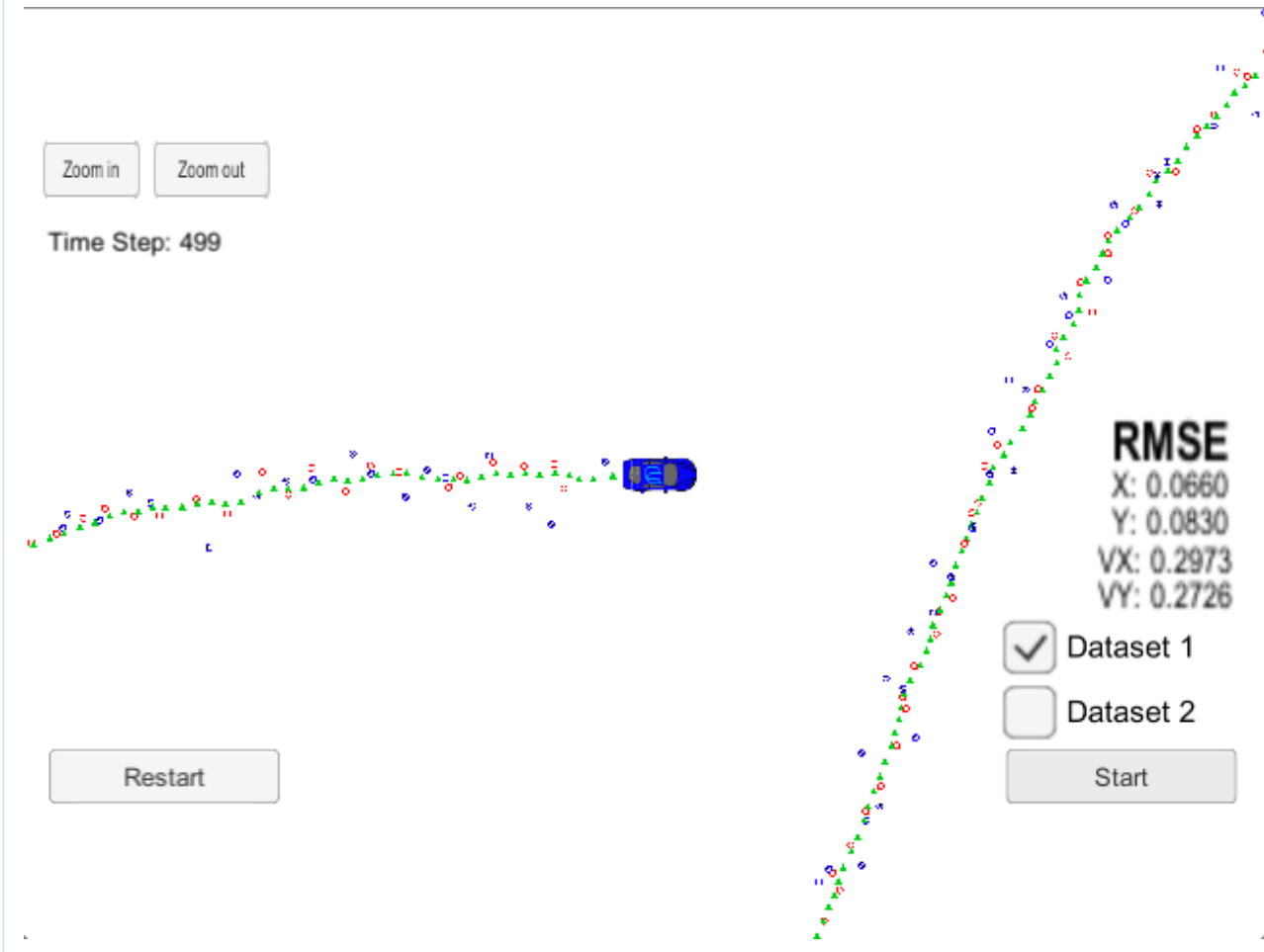
Follows the Correct Algorithm

While you may be creative with your implementation, there is a well-defined set of steps that must take place in order to successfully build a Kalman Filter. As such, your project should follow the algorithm as described in the preceding lesson.

The filter implements the basic steps in the correct order.

- initialization
- loop until done

  - prediction of new state (and state estimate error covariance) given elapsed time
  - update using new measurements data

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

The state and covariance matrices are initialized upon receiving the first measurements from the laser sensor. Well done.!

Great work correcting the variances of the longitudinal and yaw accelerations; the values in our starter code are too large.

safety

An enhancement that should be implemented is avoiding initializing with nonsense data, so as to avoid arithmetic problems; for example if `x == y == 0`. This represents the object being at the location of the measuring instrument. Likely it is nearby and the noise is producing the result. So it makes sense to do something like this pseudo code

```
if ( fabs(x) == < 0.001 && fabs(y) < 0.001 )  <--- values are flexible
{
   x = 0.001;    <--- values are flexible
   y = 0.001;    <--- choose to satisfy rmse requirements
}
is_initialized_ = true;
```

Zero values are present in the first laser data in the second test data set, and can cause problems during update steps.
Many students see `nan` rmse results without this.

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.

Excellent work here.!

On getting new data after the first measurement, the filter predicts the state vector and the covariance matrix using the time difference since the last measurement. It then uses the new measurement data to update the state vector and covariance matrix.

safety 1

Excellent, you took care to avoid arithmetic problems here (while predicting sigma points)

```
//avoid division by zero
if (fabs(yawd) > 0.001) {
```

```
    px_p = p_x + v / yawd * (sin(yaw + yawd * delta_t) - sin(yaw));
    py_p = p_y + v / yawd * (cos(yaw) - cos(yaw + yawd * delta_t));
}
else {
    px_p = p_x + v * delta_t * cos(yaw);
    py_p = p_y + v * delta_t * sin(yaw);
}
```

safety 2

An enhancement that should be implemented so as to avoid numerical instability during the prediction is

```
    while (dt > 0.2)   <--- value is flexible
    {
        double step = 0.1;   <--- value is flexible
        Prediction(step);
        dt -= step;
    }
    Prediction(dt);
```

Many students see `nan` rmse results without this.

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

Spot on. When a measurement is available the correct matrices are updated based upon the measurement type, and the correct measurement update functions are called for each sensor type.

Together with the prediction step, these two steps are the hardest part of implementing a successful Unscented Kalman Filter and you have accomplished them with ease. Very well done.

comment

The filter uses sigma points to perform the laser and the radar measurement update steps. Nice work handling the differences between data sources correctly.! However, since the laser data is linear, we could just use the standard kaman filter update (the same one we implemented in the EKF for processing laser measurements) instead to perform laser measurement updates here.

safety

Steps should be taken to avoid arithmetic problems here (while transforming sigma points to radar measurement space)

```
// measurement model
Zsig(0, i) = sqrt(p_x*p_x + p_y*p_y);                    //r
Zsig(1, i) = atan2(p_y, p_x);    <--- bad if p_x == p_y == 0
Zsig(2, i) = (p_x*v1 + p_y*v2) / sqrt(p_x*p_x + p_y*p_y);  <--- bad if denominator ==
0
```

In the first case when `p_x` and `p_y` are zero this is unsafe and a domain error occurs. The second case causes division by zero.

requirement

We want to see arithmetic errors handled properly. Please have this improved in future.

Code Efficiency

This is mostly a "code smell" test. Your algorithm does not need to sacrifice comprehension, stability, robustness or security for speed, however it should maintain good practice with respect to calculations.

Here are some things to avoid. This is not a complete list, but rather a few examples of inefficiencies.

- Running the exact same calculation repeatedly when you can run it once, store the value and then reuse the value later.
- Loops that run too many times.
- Creating unnecessarily complex data structures when simpler structures work equivalently.
- Unnecessary control flow checks.

In general the code is well written and quite efficient.

efficiency

Every update cycle, the `R` matrix is recalculated. But since there are only two sets of values, it makes sense to do the calculations once during initialization and assign them to global or class variables, which can be referenced when needed.

⬇ **DOWNLOAD PROJECT**

RETURN TO PATH

Rate this review