matthew

Project Breakdown by File

main.cpp

int main(argc, argv) {

1. Ensure program arguments correspond to valid files by inspecting `argv[1]` and `argv[2]` values
2. Assign the first argument (`argv[1]`) to the `in_file_name_` (*data/sample-laser-radar-measurement-data-1.txt*) string variable
3. Intialize the `in_file_`'s input file stream (`ifstream`) variable to read the input data
4. Assign the seconds argument (`argv[2]`) to the `out_file_name_` `string` variable
5. Initialize the `out_file_` output file stream (`ofstream`) variable to write output data to
6. Determine whether `in_file_` and `out_file_` are open. If one is, print an error message and exit the program via `exit(EXIT_FAILURE)`
7. Initialize the `std::vector` s `measurement_pack_list<MeasurementPackage>` and `gt_pack_list<GroundTruthPackage>`
8. Using istream::getline, prepare the measurement packages (each line represents a measurement at a timestamp) by iterating over `in_file_`
    a. Inside the while loop, the following occurs:
        i. declare the `sensor_type` variable which will be equal to either **"L"** or **"R"**
        ii. initialize the `meas_package` and `gt_package` variables
        iii. initialize the `iss` variable of type `istringstream` representing the current string line being processed
        iv. initialize a `timestamp` variable of type `long` representing the measurement time
        v. read in the the first character assigning it to `sensor_type` :
            1. if **"L"** (LIDAR measurement):
                a. Declare `MeasurementPackage#sensor_type_` variable equal to `MeasurementPackage::LASER`
                b. Initialize `MeasurementPackage#raw_measurements_` to a 2D `VectorXd(2)`

            c. Read in the next 2 sections corresponding to the raw position measurments `px` and `py` of type `float`
            d. Assign the `MeasurementPackage#raw_measurements_` vector the measurement values `px` and `py`
            e. Read in the timestamp corresponding to the LIDAR measurement time and assign it to `MeasurementPackage#timestamp_`
            f. Push the `meas_package` variable declared in 8a(ii) onto the `measurement_pack_list<MeasurementPackage>` vector initialzied in 7.
        2. if **"R"** (RADAR measurement):
            a. Declare `MeasurementPackage#sensor_type_` variable equal to `MeasurementPackage::RADAR`
            b. Initialize `MeasurementPackage#raw_measurements_` to a 3D `VectorXd(3)`
            c. Read in the next 3 sections corresponding to the raw position measurments `rho` (range), phi (bearing) and rho_dot (range rate) of type `float`
            d. Assign the `MeasurementPackage#raw_measurements_` vector the measurement values `rho`, `phi` and `rho_dot`
            e. Read in the timestamp corresponding to the RADAR measurement time and assign it to `MeasurementPackage#timestamp_`
            f. Push the `meas_package` variable declared in 8a(ii) onto the `measurement_pack_list<MeasurementPackage>` vector initialzied in 7.
    vi. Read ground truth data to compare later
        1. Declares 4 variables corresponding to the ground truth position and velocity: `(px_gt`, `py_gt`, `vx_gt`, `vy_gt)^T`
        2. Initialize `GroundTruth#gt_values_` vector to a 4D `VectorXd(4)`
        3. Assign `px_gt`, `py_gt`, `vx_gt` and `vy_gt` to `gt_values_` vector

9. Instantiate an instance of the `FusionEKF` class to the variable `fusionEKF`
10. Initialize our `estimations` and `ground_truth` RMSE results of type `vector<VectorXd>`. ***<u>We are ultimately graded on these numbers!</u>***
11. *Call the EKF-based fusion* by iterating over each item in `measurement_pack_list` (`vector<MeasurementPackage>`). The following occurs inside this `for` loop:
    a. Start filtering from the second frame (the speed is unknown in the first frame)

```
    fusionEKF.ProcessMeasurement(measurement_pack_list[k]);
```

b. Write to `out_file_` (e.g., *build/output.txt*) the estimated position and velocity values of the `KalmanFilter#x_` vector representing the state of the vehicle `(px, py, vx, vy)^T` after performing the prediction and measurement updates for both Lidar and Radar measurements. Note each item in the `x_` vector is printed to the `ofstream` separated by tabs (via **"\t"** at the end of each line).

c.

}