1 Feature EKF Slam Localization

This exercise tackles the problem of the localization of the Differential Drive Mobile Robot without any previous knowledge of the map arrangement. The features are observed by the robot in Cartesian Coordinates and stored in the state vector also as Cartesian Coordinates. The major challenge of this lab is dealing with a state vector that contains the mapped features and grows in size when new, unmapped features are detected. As the map is part of the state vector, and therefore is also optimized, the **prediction** function must be updated, as well as the **observation model and its jacobians**. Moreover, it emerges the need to handle the addition of new features into the state. In this lab, these issues are tackled in isolation.

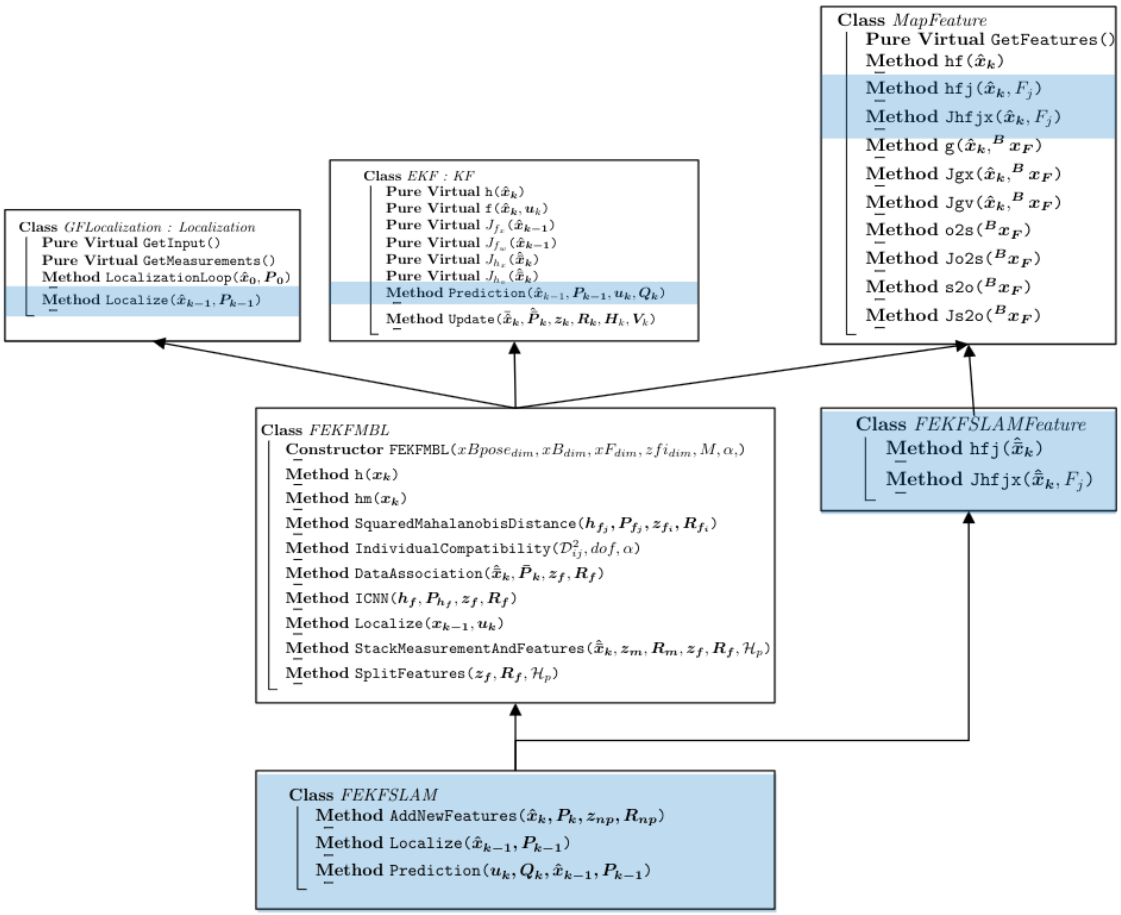

Figure 1: SLAM class hierarchy

1.1 Required files

This lab is not set up in a git repository, as only a few modifications of the **Map Based EKF Localization** lab are required. Download the following attached files into the root folder of your Map-Based EKF lab:

- FEKFSLAM.py
- FEKFSLAMFeature.py
- FEKFSLAM_3DOFDD_InputVelocityMM_2DCartesianFeatureOM.py

Figure 1 depicts the class hierarchy and the functions that need to be overwritten to upgrade a Map-based localization EKF to a full SLAM solution.
Before starting the lab, take your time to go through the files and understand the role of each class.

2 Part I: Localization without feature measurements

The goal of part I is to implement the **Prediction** method in isolation. Therefore, no feature updates nor state augmentation will be used.

1. Read the documentation of the **FEKFSLAM class**.
2. Force the initial vector state (and its associated covariance matrix) to include the position of the features in the map.
3. Implement the upgraded **Prediction** function.
4. Implement the **Localize** function to fulfill the needs of this part. Compass measurements are not necessary but can be included.
5. Test the Localization by running the **FEKFSLAM_3DOFDD_InputVelocityMM_2DCartesianFeatureOM.py** script. Try it with different state vector sizes to double-check that it works.

3 Part II: SLAM with a priori-known features

The goal of part II is to perform the **Update** part of the SLAM algorithm without dealing with the observation of unmapped features.

1. Check that the manually included features in Part I are correct. Set their uncertainty to be small. This case resembles Map based localization.
2. Read the documentation of the **FEKFSLAMFeature class**.
3. Implement the **hfj** and **Jhfx** methods of the **FEKFSLAMFeature** class.
4. Upgrade the **Localize** method to perform updates.
5. Test the localization.
6. Increase the initial uncertainty of the features in the vector state. Test the localization.

7. Initialize the map features with some error (within the initial uncertainty). Test the localization.
8. Comment the results.

4 Part III: Full SLAM

Finally, the goal of part III is to implement the full SLAM algorithm. At this point, the last piece is to augment the state vector when new, unmapped features are detected.

1. Undo the manually included features from the state vector and its associated covariance matrix. The initial state vector must contain only the robot state.
2. Read the documentation of the **FEKSLAM class**.
3. Implement the inverse observation equation and its jacobians (**g**, **Jgx**, and **Jgv**) in the **MapFeature** class, in case you did not implement it in the previous lab.
4. Implement the **AddNewFeatures** function.
5. Upgrade the **Localize** method to add new features, but without performing updates.
6. Test the localization without updates. Comment the results.
7.
8. Upgrade the **Localize** method to perform updates.
9. Test the localization. Comment the results.
10. Optional:
    (a) 2D Features Stored in Cartesian Space but Observed in Polar Space.
    (b) Constant Velocity Motion Model.