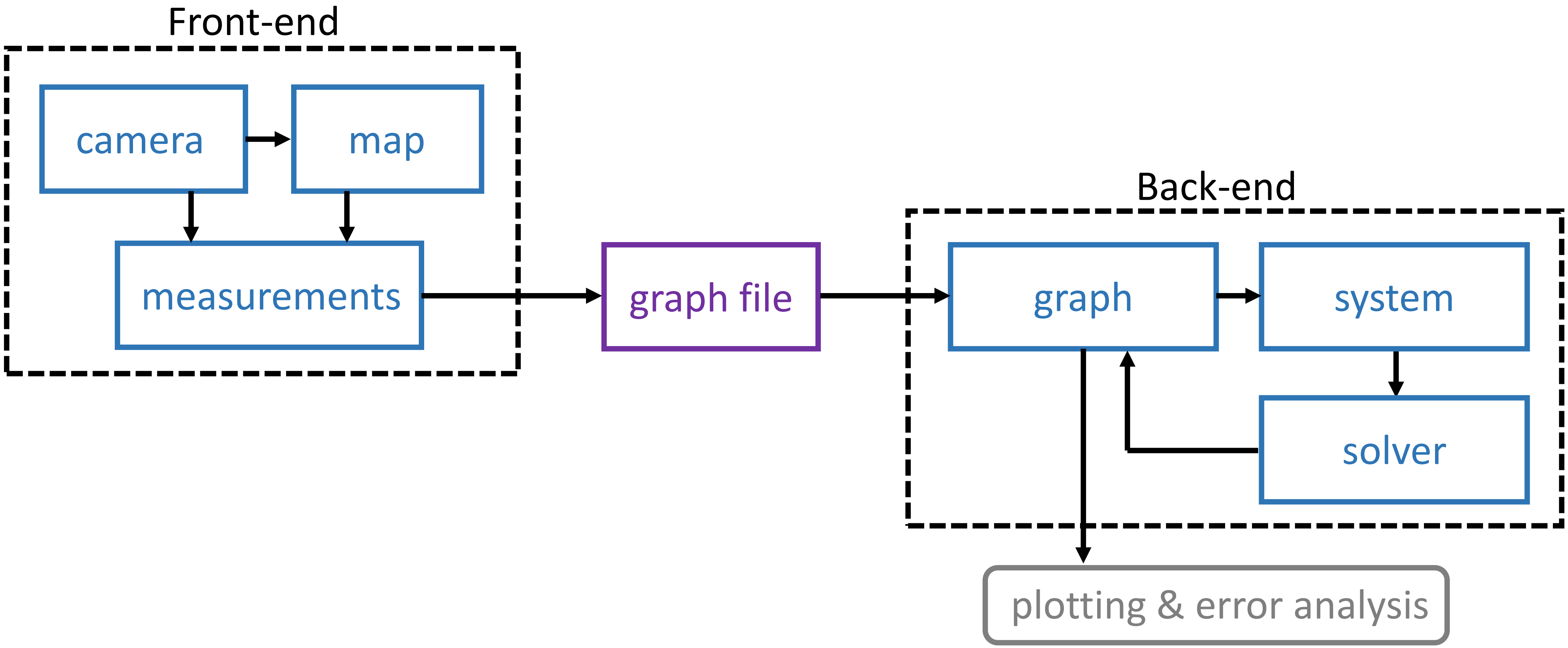
dynamicSLAM Matlab Implementation

Montiel Abello

Overview

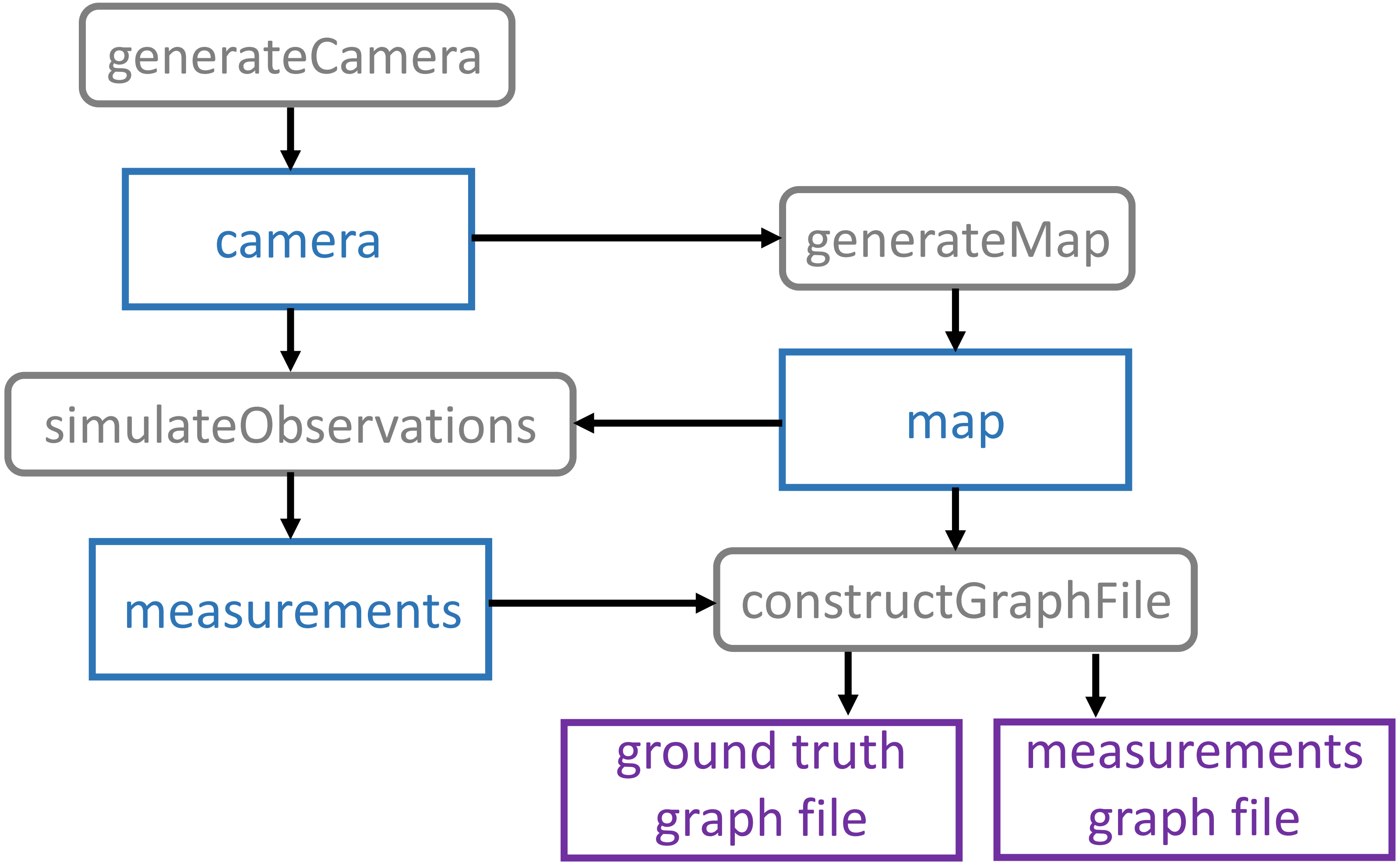


Front-end
camera
map
measurements
graph file
Back-end
graph
system
solver
plotting & error analysis

Config

- All simulation settings & parameters
- Create your own – edit /Settings/setConfig0_default.m and rename
- Add new properties to /Classes/@Config/config.m
- NOT global

Front-end: Overview



Front-end: Camera

- Create your own – edit /Data/generateCamera4_longerStreet.m and rename main.m | setConfig0_default.m | generateCamera4_longerStreet.m

```
function [camera] = generateCamera4_longerStreet(config)
%GENERATECAMERA_2 Generates camera class instance from config
%   pose: linear and angular velocity about x,y,z axes
%   camera sensor points in z direction of camera

%% 1. Generate pose
cameraPose = zeros(config.dimPose,config.nSteps);
%constant linear velocity in x-axis direction, constant angular velocity about x-axis
cameraPose(1,:) = linspace(1,-2,config.nSteps);
cameraPose(2,:) = linspace(-10,40,config.nSteps);
cameraPose(3,:) = linspace(10,15,config.nSteps);
cameraPose(4,:) = linspace(-2/3*pi,-2/3*pi,config.nSteps);
cameraPose(5,:) = linspace(0,0,config.nSteps);
cameraPose(6,:) = linspace(pi/8,-pi/8,config.nSteps);

%adjust based on parameterisation
if strcmp(config.poseParameterisation,'SE3')
    for i = 1:config.nSteps
        cameraPose(:,i) = R3xso3_LogSE3(cameraPose(:,i));
    end
end

%% 2. Create camera class instance
camera = Camera(config,cameraPose);

end
```

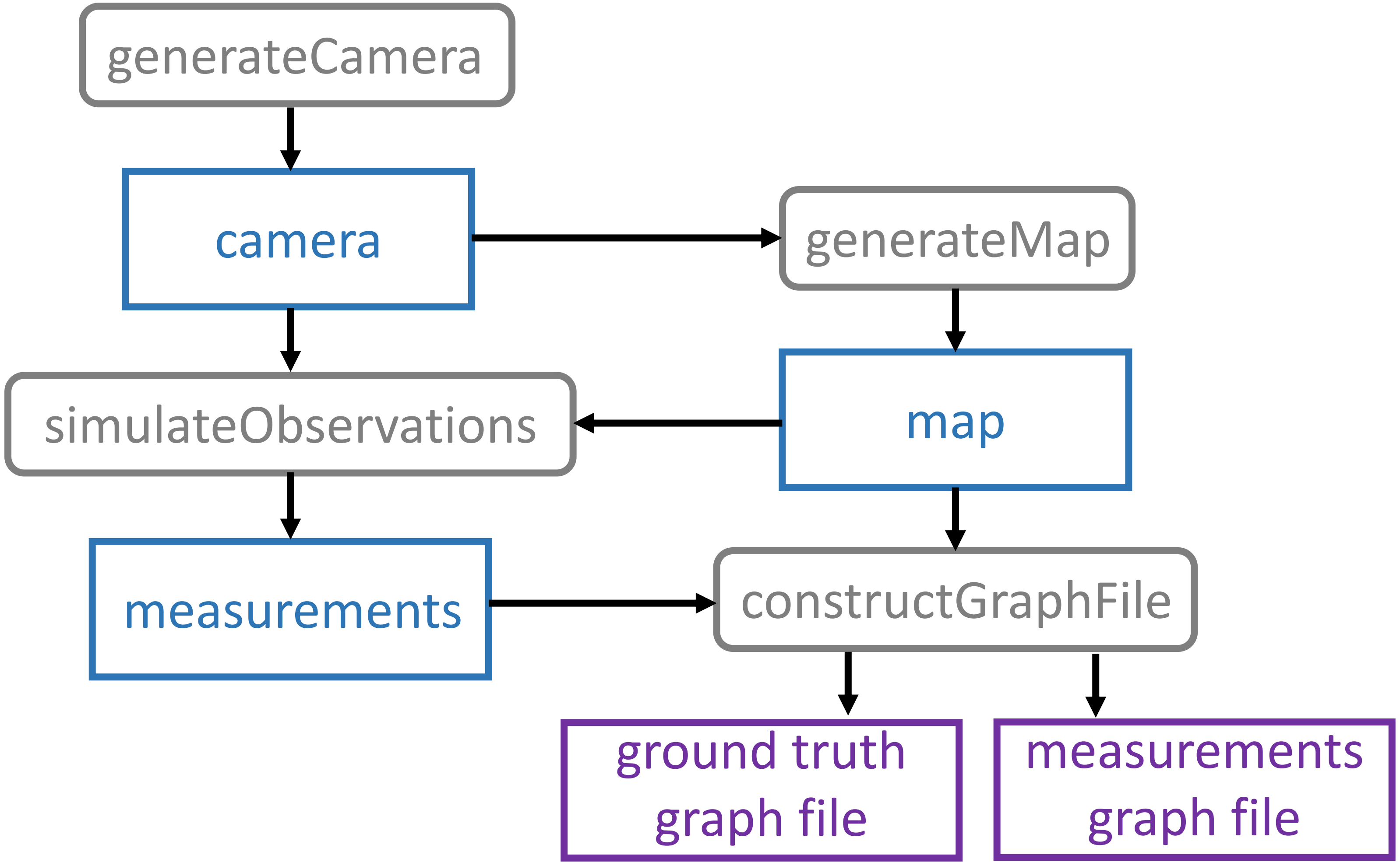
Front-end: Map

- Create your own – edit /Data/generateMap10_longerStreet.m and rename
- Points: define positions
- Entities: define type, parameters
- Objects: define type, parameters, pose
- Constraints: provide type, indexes of features involved, value

Example: indexing tricks

- Class instances stored in Matlab ‘object array’ data structure
- Indexing is similar to cell arrays

Front-end: Measurements



Front-end: Measurements

- See /Classes/@Measurements/simulateObservations.m
- Compute visibility for points, entities, objects, constraints with /Classes/@Map/visibility.m
- At each time step, simulate observations for odometry, points, entities, objects and constraints – create observation class instance for each observation
- AT THE END – add noise to observation values. Doing this all together allows random number seed to be fixed for consistent experiments

Example: adding occlusion

- Add triangles attribute to Map class
- Add function /Classes/@Camera/checkObservationOcclusions.m
- Standard checkObservation.m function checks if range and bearing within camera FOV (called in /Classes/@Map/visibility.m)
- In addition, check if vector from camera to point intersects any triangles, and if this triangle between camera and point

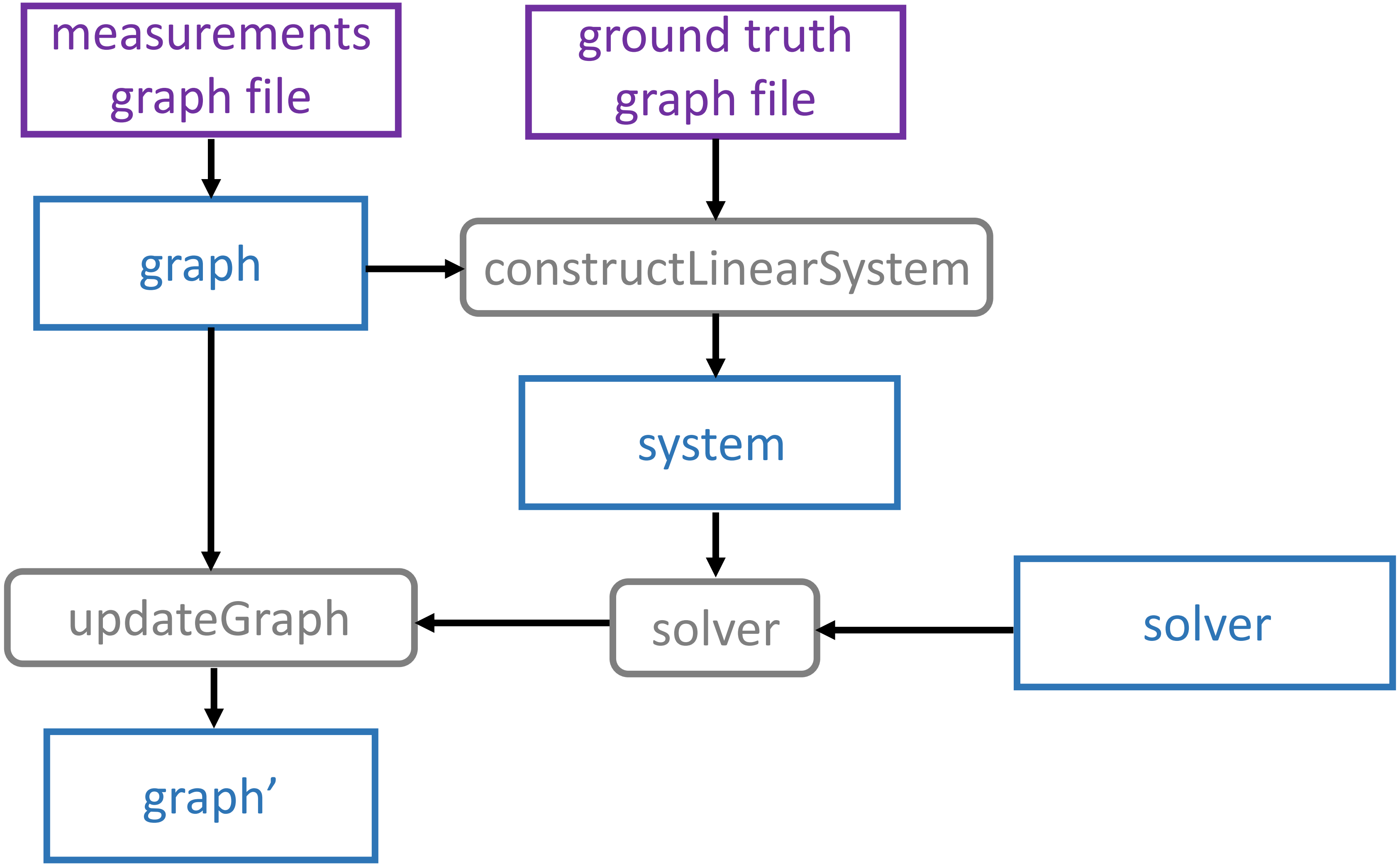
Example: adding cube object

- Add cube to map by defining object type, parameters and pose
- Add point-cube constraints to map
- Add case to /Classes/@Measurements/observeObjects.m
- Add case to /Classes/@Measurements/observeConstraints.m
- Will need to add new vertex and edge types too – will go through this later…

Front-end: Graph Files

- Go through measurements.observations in order
- Create groundTruth.graph and measurements.graph files
- measurement includes new pose, point, entity or object (based on indexes), write new vertex to groundTruth.graph file (use value from ground truth map)
- Each measurement becomes an edge in the graph files
- Vertex: label, index, value
- Edge: label, verticesIn, verticesOut, value, covariance

Back-end: Overview



Back-end: Graph

- Folder /Classes/@Graph contains LOTS of functions
- Create graph from measurements.graph with processBatch.m or processIncremental.m
- Get measurements from each time step, create vertices when new feature indexes in edge
- Create edge for each measurement

Back-end: Vertices

- Properties: value, covariance, type, iEdges, index
- Constructed with constructTYPEVertex.m function, depending on type
- Value depends on type:
  - Pose vertex value computed with previous pose and relative pose from odometry measurement
  - Plane vertex value computed by fitting plane to all points belonging to plane (only those previously observed)

Back-end: Edges

- Properties: type, value, covariance, index, iEdges
- Constructed with constructTYPEEdge.m function, depending on type
- Edge value & jacobians computed with corresponding updateTYPEEdge.m function
- Edge values & jacobians computed with connected vertex values (ie current linearization points), depending on type

Example: indexing

- identifyVertices('type') and identifyEdges('type') are useful
- Return all indexes of vertices/edges with given type

Example: adding cube vertex type

- Create /Classes/@Graph/constructCubeVertex.m function
- Add case to /Classes/@Graph/processBatch.m or /Classes/@Graph/processIncremental.m to call this function when measurement involves a cube vertex

Example: adding point-cube edge type

- Create /Classes/@Graph/constructPointCubeEdge.m function
- Create /Classes/@Graph/updatePointCubeEdge.m function
- Add case to /Classes/@Graph/processBatch.m or /Classes/@Graph/processIncremental.m to call this function when measurement with type ‘point-cube’ reached

Back-end: System

- See System class methods, particularly constructLinearSystem.m
- blockMap.m uses edge and vertex sizes and indexes to determine block locations in Jacobian matrix and covariance matrix
- Jacobians from edge are placed in system.A matrix
- Covariance is placed in system.covariance matrix and system.covSqrtInv matrix
- Residual computed with edge value and measurement using computeResidual.m function and placed in system.b vector

Back-end: Solver

- Choose Gauss-Newton, Levenberg-Marquardt or Dog-Leg (still in progress)
- Create your own: copy existing function, add new case in /Classes/@NonlinearSolver/NonlinearSolver.m constructor
- Call with:
  solver = solver.solve(config,graph0,measurementsCell)
- This solves system and stores solution in solver.graphs and solver.systems properties

After Solving

- Save solution graphN as graph file: graphN.saveGraphFile(config,'filename.graph')
- Plot solution with plotGraph.m or plotGraphFile.m functions
- Compute errors with errorAnalysis.m function

To Dos

- Indexing
  - Separate index from order
  - Unique indexes?
- Constrained linear least squares
- Dog-Leg solver
- Dynamic features in graph structure