Documentation: Dynamic SLAM MATLAB Toolbox

Montiel Abello

1 Overview

1.1 Functionality

This library can be used to:

- Generate ground truth data and simulated measurements and store them in a customisable graph file format.
- Represent graph files in a graph class structure that allows for easy manipulation and conversion to a linearised system.
- Solve linear systems with different solvers.

1.2 Structure

TODO: put a diagram here

2 Config Class

The `Config` class is where the user provides simulation settings and parameters. A `Config` class instance is created with the constructor method in `Config.m`. This function requires a string input which determines which function will be called to initialise the `Config` class properties.

```
settings = 'corridor';
config = Config(settings);
```

To create your own settings, create a `setConfigLABEL.m` function and add a case in the `Config` constructor in `Config.m`:

```
%% constructor
function obj = Config(settings)
    %assign default parameters, below is additional settings
    obj = setConfig0_default(obj);
    switch settings
        case 'batchTesting'
            obj = setConfig1_testing(obj);
        case 'montecarlo'
            obj = setConfig2_montecarlo(obj);
        case 'incrementalTesting'
            obj = setConfig3_incremental(obj);
        case 'small'
            obj = setConfig4_small(obj);
        case 'citySmall'
            obj = setConfig5_citySmall(obj);
        case 'city'
            obj = setConfig6_city(obj);
        case 'smallLoop'
            obj = setConfig7_smallLoop(obj);
        case 'corridor'
            obj = setConfig8_corridor(obj);
        case 'realsense'
            obj = setConfig9_realsense(obj);
        otherwise
            error('Config for %s not yet defined',settings)
    end
end
```

The config variable must be passed to any function that needs user determined settings or parameters. Implementation with global variables has been removed.

TODO: Table describing all properties of `Config` class.

3 Camera Class

The **Camera** class models a moving camera sensor. The user determines the measurement parameters and pose over time. Measurements are simplified, given in the form of relative position of points with respect to the camera, in the frame of the camera.

Selecting a camera generating function in a `setConfig` function:

```
obj.cameraHandle = @generateCamera4_longerStreet;
```

Initialising the **Camera** class instance:

```
camera = config.cameraHandle(config);
```

A simple camera generating function. FOV and max distance set in `setConfig`, pose configured here:

```
function [camera] = generateCamera4_longerStreet(config)
%GENERATECAMERA4_LONGERSTREET 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
```

4 Map Class

The user can simulate an environment by creating a `Map` class instance, which contains object arrays of `Point`, `Entity`, `Object` and `Constraint` class instances. Each point, entity and object will eventually form a single vertex in the graph (unless they are unobserved and removed). Each constraint will eventually form a single edge in the graph (unless it is unobserved and removed).

4.1 Point Class

A single point can be initialised with the `Point` class constructor, requiring either no inputs for preallocation, or a trajectory and index. Groups of points are initialised with a `Map` class method. For a set of n points, and a simulation with m time steps, `pointPositions` should be an array of size $3n \times m$, where each $3 \times m$ block corresponds to the $[x, y, z]^T$ trajectory of a single point

```
map = map.initialisePoints(pointPositions);
```

4.2 Entity Class

Entities are initialised similarly to points, but with an additional parameter `type`.

```
map = map.initialiseEntities(entityTypes,entityParameters);
```

4.3 Object Class

Objects are initialised similarly to entities, but with an additional input `pose`.

```
map = map.initialiseObjects(objectPoses,objectTypes,...
                            objectParameters);
```

4.4 `Constraint` Class

Constraints are initialised by passing a cell array to the class constructor. Each row of the $n \times 6$ cell array represents a single constraint. map = map.initialiseConstraints(constraints);

Each row gives: $\{i_{objects}, i_{parentEntities}, i_{childEntities}, i_{points}, type, value\}$
The first four entries correspond to indexes of features involved in the constraint. For example, a point-plane constraint between point 1 and plane 1 could be initialised with
$\{[], 1, [], 1, \texttt{'point-plane'}, 0\}$
where the value of 0 corresponds to a distance of 0 between the point and the plane.

A constraint enforcing planes 1 and 2 as parallel would be initialised with:
$\{[], [], [1, 2], [], \texttt{'plane-plane-fixedAngle'}, 1\}$
where the value corresponds to a dot product of 1 for the plane normals.

A loose constraint of this angle, also estimating the angle itself as an entity with index 3 would be initialised with:
$\{[], 3[], [1, 2], [], \texttt{'plane-plane-angle'}, 1\}$

5 `Measurements` Class

Measurements are simulated with the `simulateObservations` method of the `Measurements` class. The output is a `Measurements` class instance. Important attributes of this class are:

- `observations`: object array of `Observation` class instances
- `map`: `Map` class instance used to store indexes of observed features and relate them to features in the ground truth `Map` class instance
- visibility matrices for points, entities, objects and constraints

5.1 `Observation` Class

5.2 `Map` Class

6 Graph files

7 `Graph` Class

7.1 `Vertex` Class

7.2 `Edge` Class

7.3 Processing

7.3.1 Batch

7.3.2 Incremental

8 `System` Class

9 `Solver` Class

9.1 `GNSolver` - Gauss-Newton solver

9.2 `LMSolver`

9.3 `DLSolver`

References

[1] N. Aghannan and P. Rouchon, "On invariant asymptotic observers," in *Decision and Control, 2002, Proceedings of the 41st IEEE Conference on*, vol. 2. IEEE, 2002, pp. 1479–1484.

[2] J. D. Adarve, D. Austin, and R. Mahony, "A filtering approach for computation of real-time dense optical-flow for robotic applications," in *Proc. Australasian Conference on Robotics and Automation, Melbourne, Australia, December 2014*, editor, Ed.