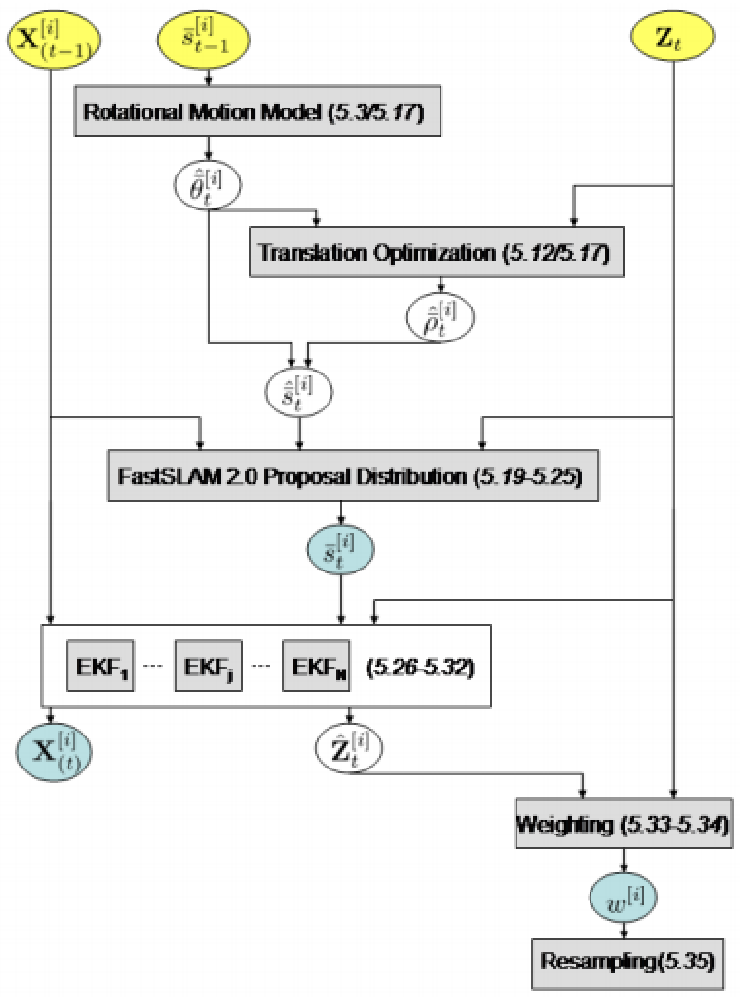
Tutorial

Hybrid pose estimation algorithm

Link to github: https://github.com/rshirochenko/SLAMVision

Algorithm flow diagram:



Files description:

main.py – Whole algorithm working process;

constants.py - All constants and parameters that estimated empirically;

measurements_z.py - SIFT measurements (Z_t) part;

feature.py – Feature (X_t) part;

pose.py – Pose (s_t) part and particle filter part;

motion_model.py - Rotation and translation motion model;

fastslam.py – FastSLAM proposal distribution, measurements update, weighting and resampling.

1) SIFT measurements (Z_t) estimation

For the SIFT measurements estimation OpenCV library is used.

To estimate SIFT for the current frame. In the main.py:

1. Get the frame.:
   ```
   ret, frame = get_frame(cap, ret))
   ```
2. Estimate the SIFT measurements from the current frame:
   ```
   current_measurements = measurements_Z.make_measurement(frame)
   ```

Workflow inside measurements_z.py:

1. Estimate the SIFT measurements(coordinates(kpoint) and 128 vector descriptors(des)) from the current frame:
   ```
   kpoint, des = self.calc_sift_points(img)
   ```
2. Form the reference table J for the current frame:
   ```
   table_J = self.create_table_J(kpoint, des)
   ```
3. Make update for the reference table K. In table K, we are updating the last observed parameter. For this purpose the correspondence algorithm is used.
   ```
   self.update_table_K(table_J)
   ```
4. Return the last observed measurements from table K with last_observed equal 0.
   ```
   current_measurements = self.get_current_measurements()
   ```

Notice: the SIFT estimation work strange for the first 5 frames, so we are start algorithm work from the 6th frame.

2) Feature (X_t) initialization

For the feature initialization, we need to collect 4 measurements with the same (corresponded) SIFT descriptor. In order to do this, the FeaturesCache object is used. The FeaturesCache keep the measurements with the same descriptors in the python dictionary data structure. After collecting 4 measurements, the EKF(mean and covariance) for the feature is formed.

```
feature_cache.get_measurements(current_measurements, X_map_init, init_pose)
```

In main.py file, for the first 4 frames we form the common object FeaturesCache for the all particles in the filter. After 4th frame, FeaturesCache object is formed for each particle.

```
feature_cache_particle[i].get_measurements(current_measurements,
particles_filter[i].X_map_dict, particles_filter[i].pose)
```

3) Pose estimation

Pose is estimated during rotational and translational models. The file pose.py describes the pose(s_t) and particle objects.

4) Rotational and translational motion models

File *motion_model.py* contains rotational and translational motion models. In the main.py file, Rotational motion is estimated as:

```
motion_model.rotational_motion_model(particles_filter[i].pose)
```

Translation optimization step:

```
motion_model.translational_optimization(particles_filter[i],
current_measurements)
```

5) FastSLAM

The FastSLAM.py file contains estimation stage for the time update(distribution proposal), measurements update and particle weighing and resampling.

In the main.py file FastSLAM Proposal distribution (time update stage):

```
fastslam.calc_position_proposal_distribution(particles_filter[i], current_measurements)
```

Measurement update(EKF on the workflow diagram) and weighting stages:

```
weight = fastslam.measurement_update(particles_filter[i], current_measurements)
```

Resampling:

```
if fastslam.check_for_resampling(weight_sum):

    particles_filter[i].weight = 1.0/constants.NUMBER_OF_PARTICLES
```