Openvins simulation points generation

Friday, March 13, 2020 4:56 PM

In openvins, ov_core/src/sim/Simulator.cpp, there is a function to generate 3D points. void Simulator::generate_points. To understand this function, we need to review the concepts:

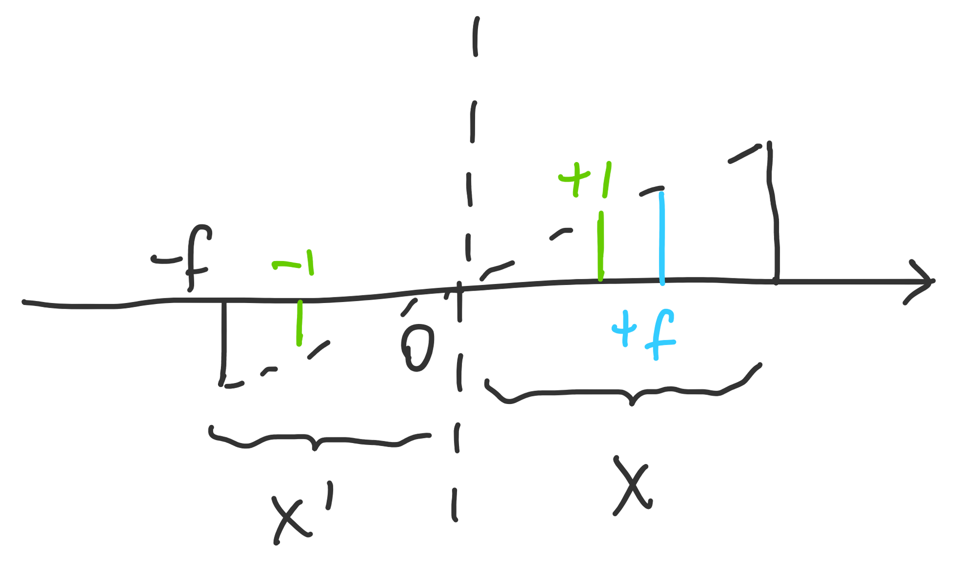
Review: we follow SLAM Book 14 chapters.

For a point $P = \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$ in camera frame, the pinhole model projection is:

$$\frac{Z}{f} = -\frac{X}{X'} = -\frac{Y}{Y'}$$

To simplify, we assume the object is not inverted and get

$$\frac{Z}{f} = \frac{X}{X'} = \frac{Y}{Y'} \Rightarrow X' = f\frac{X}{Z} \quad Y' = f\frac{Y}{Z}$$



1: normalized
1: symmetric projection
1: real projection.

from P', we can get the pixels as:

$$\begin{cases} u = f_x \frac{X}{Z} + c_x \\ v = f_y \frac{Y}{Z} + c_y \end{cases} \Rightarrow \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \frac{1}{Z}\begin{pmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} X \\ Y \\ Z \end{pmatrix} \triangleq \frac{1}{Z}KP$$

we can rearrange and get

$$Z\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \begin{pmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} X \\ Y \\ Z \end{pmatrix} \triangleq KP$$

For a point in world frame P_w:

$$ZP_{uv} = Z\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = K(RP_w + t) = KTP_w \Rightarrow P_{uv} = KTP_w$$

Since we are using homogeneous coordinates.

For point in camera frame, $P_c = \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = (TP_w)_{(1:3)} \Rightarrow \underline{P_c} = \begin{bmatrix} \frac{X}{Z} \\ \frac{Y}{Z} \\ 1 \end{bmatrix}$ (*)

$\underline{P_c}$ can be viewed as a homogeneous coordinate on the Z=1 plane in front of the camera.

$$P_{uv} = K\underline{P_c}$$

Now come back to the generate points function:

```
for (int i=0; i< numpts; i++) {
```

The numpts is the max. points per frame that we want to see.

```
std::uniform_real_distribution<double> gen_u(0, camera_wh.at(camid).first);
```

This produces random points uniformly sampled from [0, image width).

note that camera_wh.at(camid) returns the w, h for camid, and .first means the width, .second means height.

```
cv::undistortPoints(mat, mat, camK, camD);
```

This function returns normalized pixels

```
Eigen::Vector3d uv_norm;
mat = mat.reshape(1);
uv_norm(0) = mat.at<float>(0,0);
uv_norm(1) = mat.at<float>(0,1);
uv_norm(2) = 1
```

uv_norm corresponds to $\underline{P_c}$ in (*)

```
Eigen::Vector3d p_FinC;
p_FinC = depth * uv_norm;
```

P_FinC is P_c in (*)

```
Eigen::Vector3d p_FinI = R_ItoC.transpose() * (p_FinC - p_IinC);
Eigen::Vector3d p_FinG = R_GtoI.transpose() * p_FinI + p_IinG;
```

$$P\_FinI = {}_C R_I^T (P\_FinC - P\_IinC)$$

$$P\_FinG = {}_I R_W^T P\_FinI + P\_IinG$$

```
featmap.insert({id_map, p_FinG});
```

generate points function returns nothing, the only variable it changes is featmap, featmap stores the id id_map for each feature position P_FinG.

In Simulator::Simulator(ros::NodeHandle & nh), which calls generate points function, we have

```
std::vector<std::pair<size_t, Eigen::VectorXf>> uvs = project_pointcloud(R_GtoI, p_IinG, i, featmap);
if ((int) uvs.size() < num_pts) {
    generate_points(R_GtoI, p_IinG, i, featmap, num_pts - (int) uvs.size());
}
```

This part first projects the points in featmap to the image.

If we are not getting enough observations, then we generate more points and add them to featmap.