1. Extrinsics

After Dataloader

1. Every Extrinsic cam is relative to the first camera.
2. Each Extrinsic timestep is different cause the vehicle may jerk
3. Output Shape = B, N_cams, 4, 4

```
inverse(cam_front_ext) x cam_front_ext
inverse(cam_front_ext) x cam_2_ext
inverse(cam_front_ext) x cam_3_ext
```

After Model

1. [miscellaneous] `_p(Extrinsics)`
2. Inverse the Extrinsics

---

2. Intrinsics

After Dataloader Intent: For uniformity for before passing to model.

1. cam 3x3 -> 4x4 [Refer below]
2. Output Shape = B, N_cams, 4, 4

```
K = [fx 0 cx ]
    [0  fy y ]
    [0  0  1 ]

New_intrinsics of 4x4 is
[fx 0  cx 0]
[0  fy cy 0]
[0  0  1  0]
[0  0  0  1]
```

In Model

1. [miscellaneous] `_p(intrinsics)`
2. Scale with [scale_x, scale_y] = [sx, sy]

sx = H_encoder_feature/Original_image_Height

sy = W_encoder_feature/Original_image_Width

```
[fx*sx 0    cx*sx 0]
[0    fy*sy cy*sy 0]
[0    0      1    0]
[0    0      0    1]
```

3. Misc Funcs

```
__p = lambda x: utils.basic.pack_seqdim(x, B)   # [B,S,C,H,W -> B*S,C,H,W]
or [B, S, 4, 4 -> B*S, 4, 4]
__u = lambda x: utils.basic.unpack_seqdim(x, B) # [B*S,C,H,W -> B,S,C,H,W]
or [B*S, 4, 4 -> B, S, 4, 4]
```

4. Combine Intrinsics and Extrinsics

Send `3 things` to `unproject_image_to_mem` to get `feat_mems`

```
1. cam features
2. torch.matmul(intinsics, Inverse(extrinsics))
3. Inverse(extrincis)
Miscellaneous X,Y,Z
4. Memory2Ref aka xyz_camA, i assume the reference is a 3D drone view
memory
```

`xyz_camA = vox_util.Mem2Ref`

Lets trace

a.Freq funcs

```
utils.basic.reduce_masked_mean, [Simpleloss, centerLoss, offsetLoss]
```

b.To scrap

```
simplePool aka misc.py
```

c.To DO later

```
valid_bev_tgt??
Trace that
```

1. train.py Inputs in `run_model` line 179

Input	Output	Extras
rgb_cams	feat_bev	valid_bev
Intrinsics_cams	seg_bev	
Extrinsics_cams	center_bev	
vox_util_obj	offset_bev	
occupancy_aka_lidar		

2. Check loss and see what libraries are needed

Loss Type	Inputs	Functions
ce_loss = SimpleLoss	seg_bev_pred, seg_bev_tgt, valid_bev_tgt	BCEWithLogitsLoss(seg_bev_pred, seg_bev_tgt), reduce_masked_mean(loss, valid_bev_tgt)
center_loss	center_bev_pred, center_bev_tgt	Line 66 balanced_mse_loss
offset_loss	offset_bev_pred, offset_bev_tgt, seg_bev_tgt,valid_bev_tgt	torch.abs(offset_pred,offset_tgt).sum(dim=1) -> maskmean(loss, segtgt*validtgt)
ce_uncertainty_loss	Useless	remove the weights of these 3 from model
center_uncertainty_loss	Useless	
offset_uncertainty_loss	Useless	

ce_loss

```
ce_loss = SimpleLoss line56
BCEWithLogitsLoss(seg_bev_pred, seg_bev_tgt)
utils.basic.reduce_masked_mean(loss, valid_bev_tgt)
```

New Datasets to be generated

- B = Batch
- N_cams = Number of cameras
- W = Width
- H = Height

Input	Shapes
rgb_cams	B,N_cams,3,W,H

Input	Shapes
Intrinsics_cams	B,N_cams,4,4
Extrinsics_cams	B,N_cams,4,4
vox_util_obj	its a class obj
occupancy_aka_lidar	

Inputs from loss	Shapes	Dtype	Explanation
valid_bev_pred	B,1,W,H	Bool	Initialize with zeros, set 1 where exists

Outputs from Model	Shapes	Dtype	Explanation
feat_bev_pred	B,n_ch,W,H	F32	Features from decoder
seg_bev_pred	B,1,W,H	Int32	Segmentation with Each class
center_bev_pred	B,1,W,H	F32	Center, of object with circle size of 3
offset_bev_pred	B,2,W,H	F32	Offset of object from center in x and y in pixel space

New Losses

1. ce_loss = SimpleLoss
2. center_loss = balanced_mse_loss
3. offset_loss = torch.abs(offset_pred,offset_tgt).sum(dim=1) -> maskmean(loss, segtgt*validtgt)

- Change offset loss to use valid_bev_tgt, so it's univariate of each class loss

Utilities

1. Vox util

- RT: Intrinsic x Extrinsic
- Mem2Ref: Memory to Reference 3D grid

Formula = Pixel Space u,v,1 = Intrinsic x Extrinsic x 3D grid xy_pixB = uv1/Normalized value

There is 2 z from the camera Which is Tinv*feat_cams = xyz_camB = Extrinsic x 3D grid and we get only the z value