Install Prerequisites in Your Brain

이번 시간에는 VODE를 이해하기 위해 필요한 사전지식을 공부합니다.

1 What is VODE?

오늘 공부하려는 내용을 한 문장으로 표현하자면 다음과 같습니다.

> Complementary learning of visual odometry and depth estimation by deep neural network in an unsupervised way
>
> 깊은 신경망에 의한 시각적 위치인식과 깊이 추정의 상호보완적인 비지도 학습

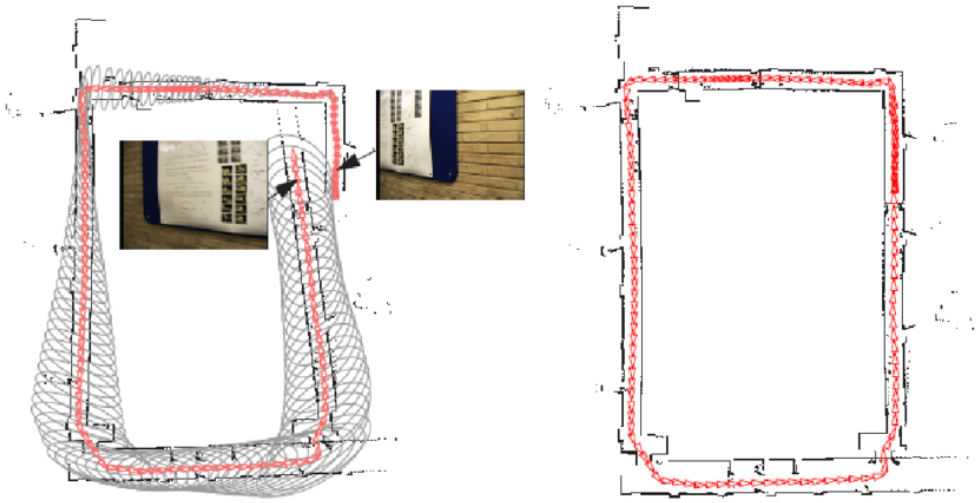
각 키워드의 의미를 알아봅시다.

- Visual odometry (VO): 연속된 영상으로부터 카메라의 자세 변화를 추정하는 기술
- (Monocular) Depth Estimation (DE): 한 장의 영상으로부터 영상의 깊이(depth)를 예측하는 기술이다. 원래 영상의 깊이를 알기 위해서는 스테레오 이미지나 ToF 센서 등을 통해서 측정해야 한다. 하지만 인간이 한 장의 사진에서 대략적인 깊이를 유추할 수 있듯이 딥러닝을 통해 깊이를 직접 학습시킬 수 있다.
- Deep neural network (DNN): VO와 DE를 각각 다른 DNN으로 구현한다.
- Complementary, Unsupervised: VO와 DE를 위한 두 개의 DNN을 **비지도 학습**으로 학습시킬 수 있는 이유는 서로가 **상호보완적**이기 때문이다. Pose와 Depth 두 정보가 있으면 한 장의 영상을 다른 시점에서 찍은 것처럼 **합성(synthesize)** 할 수 있다. VODE 연구는 대부분 이러한 합성이 잘 이루어지도록 학습을 유도한다.

딥러닝 기반 VODE 기술에 대해 자세히 논하기 앞서 저 키워드들이 의미하는 바를 제대로 이해해야 합니다. 이번 시간은 VO와 DE의 역사적 맥락을 짚어보고 VO에서 출력해야 할 자세(pose)의 표현방법에 대해 간단히 알아봅니다.

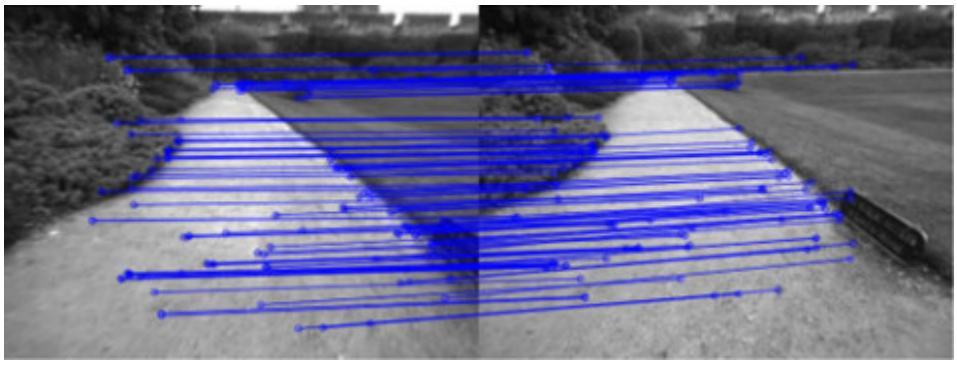
2. Visual Odometry

Visual odometry에서 odometry란 원래 로봇의 움직임을 계산하거나 기록하는 장치란 뜻으로 쓰였습니다. 대표적으로 로봇 바퀴에 달린 인코더(encoder)나 IMU (Inertial Measurement Unit) 등을 이용해 로봇의 자세변화를 계산하거나 장치들이 있습니다. **Visual Odometry는** 로봇의 자세 변화를 직접적으로 측정할 수 없는 영상을 이용해서 odometry처럼 자세 변화를 추정하는 기술입니다. 비슷한 연구분야로 Visual Simultaneous Localization and Mapping (vSLAM) 기술인데 이는 자세의 변화량을 누적시키는 VO에 전역 지도구축 기술을 더하여 누적되는 위치 오차를 loop closing으로 해소할 수 있는 기술입니다. 아래 그림은 누적된 오차를 loop closing으로 해결한 모습으로 보여줍니다. vSLAM에서 loop closing 기능만 빼면 위치인식 측면에서는 VO와 같아지므로 두 기술이 함께 비교되는 경우가 많습니다. 아래 그림은 Loop closing 전후의 경로를 비교한 것입니다.



VO/vSLAM 분야는 크게 두 가지 갈래로 연구가 되고 있습니다.

1. Indirect method: 영상에서 keypoint와 descriptor를 이용하여 두 영상 사이에 특징점들을 매칭하고 이를 이용해 영상 사이의 자세변화를 알아내는 방법입니다. 매칭된 픽셀들의 이동량을 이용해 상대적인 자세를 계산합니다.



2. Direct method: 특징점 같은 가공된 데이터를 사용하지 않고 영상의 픽셀 값을 그대로 사용하는 방법입니다. A시점의 영상의 어떤 픽셀을 다른 시점 B로 옮겼을 때 A의 픽셀 값과 B의 픽셀값이 같아지도록 하는 상대적인 자세를 계산합니다.

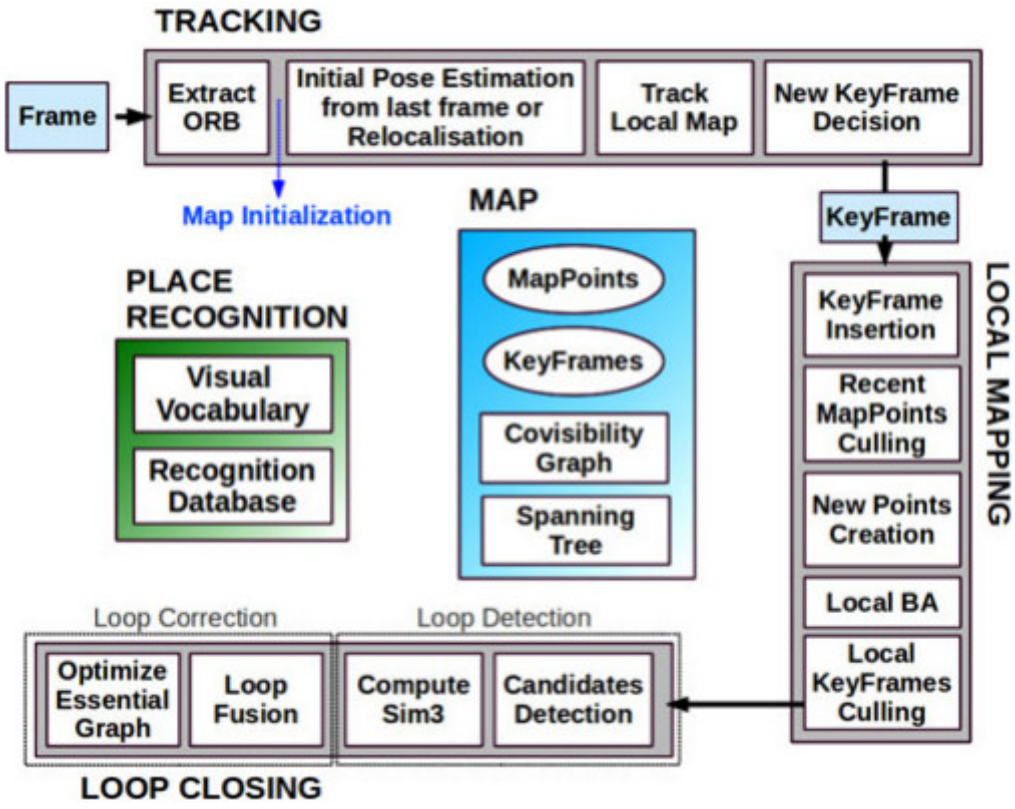
Indirect 방식은 먼저 점들을 정확히 매칭시켜놓고 이를 설명할 수 있는 상대 자세를 계산하는 것이고 Direct 방식은 우선 임의의 상대 자세를 이용해 점들을 다른 시점으로 옮겨놓고 옮겨진 점들의 픽셀 값이 다른 시점의 픽셀 값과 같아지도록 상대 자세를 맞춰나가는 방식입니다. 두 방식은 여전히 경쟁을 하고 있고 두 방식 모두 정확도와 속도면에서 상향 평준화 되었기 때문에 어느쪽이 더 좋다고 정할순 없지만 장단점이 있습니다.

	Indirect method	**Direct method**
대표논문	ORB-SLAM, ORB-SLAM2	LSD-SLAM, DSO
장점	상대적으로 노이즈에 강인하고 다른 카메라나 데이터에 쉽게 적용 가능	다수의 픽셀을 활용하기 때문에 특징이 부족한 상황에 강인함
단점	영상 특징이 부족한 상황에 취약	픽셀 노이즈나 rolling shutter, 노출 변화 등에 민감하여 활용 난이도가 높음

Q) VO에 딥러닝을 써야할까?

두 방식 모두 좋은 성능을 보이고 있는데 굳이 여기에 딥러닝을 갖다 붙여야 할까요? 기존의 전문가 시스템 (either direct or indirect)에서 **수학적 최적화로 정확한 계산**을 하고 있는데 왜 딥러닝을 써야할까요? 이는 마치 덧셈을 딥러닝으로 해결하고자 하는 것과 같습니다.

제가 보기에는 기존 방식에 치명적인 단점이 있습니다. **구현 난이도**가 지나치게 높다는 것입니다. 논문을 볼 때 주로 성능을 보고 구현 난이도는 간과하기 쉬운 부분이지만 후속 연구를 하거나 그 기술을 응용해서 쓰고 싶은 사람에겐 중요한 점입니다. VO/vSLAM 연구도 이제 꽤 성숙한 단계에 들었기 때문에 성능은 좋아졌지만 그와 비례해서 구현의 난이도가 저세상으로 가버렸습니다. 다음 그림은 ORB-SLAM의 모듈 구성입니다.



저 모듈 하나하나가 논문 주제들인데 저걸 다 높은 수준으로 구현하는 것 뿐만 아니라 multi-thread를 통해 고속으로 처리해야 하는 그 어려운 일을 Raul Mur-Artal 이라는 저자가 해버렸지 말입니다... 외부에서 가져다 쓴 모듈도 많지만 코드를 보면 성능을 올리기 위해 상당히 디테일하게 구현했다는 느낌을 받을 수 있습니다.

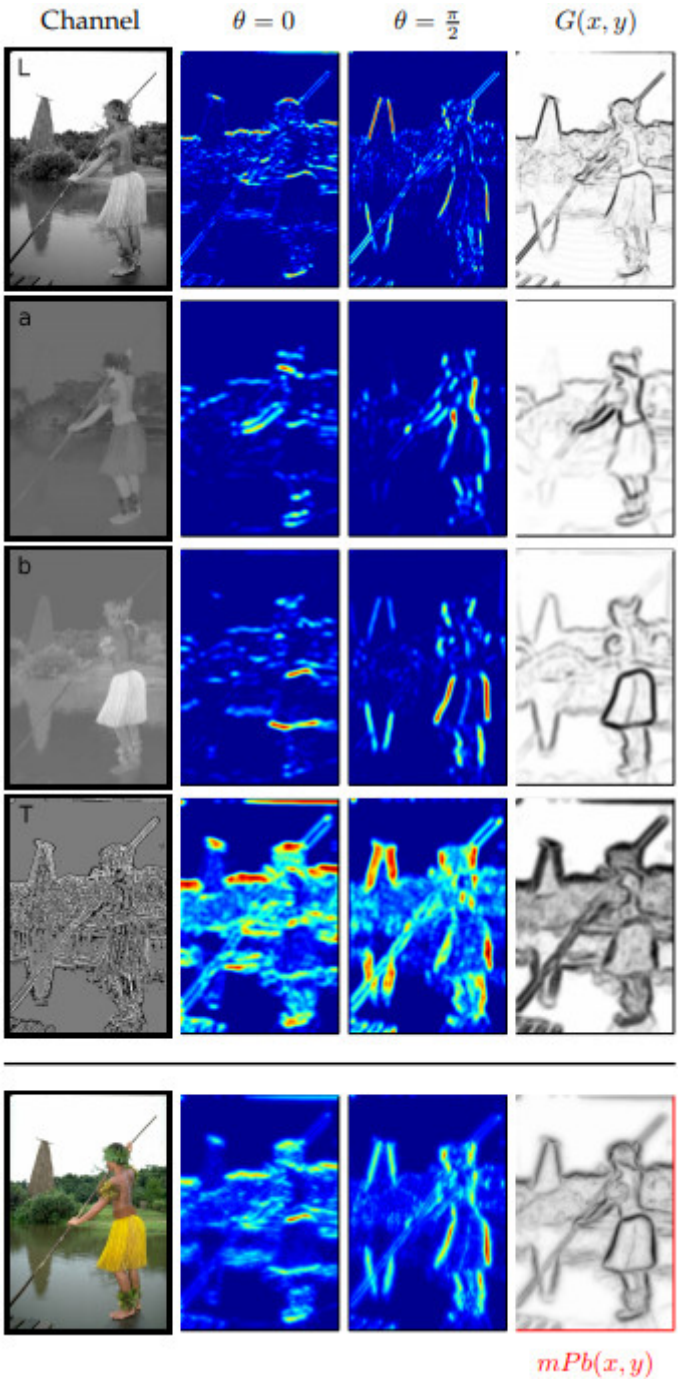
다음은 주요 논문들을 구현한 소스코드의 양을 비교한 표입니다.

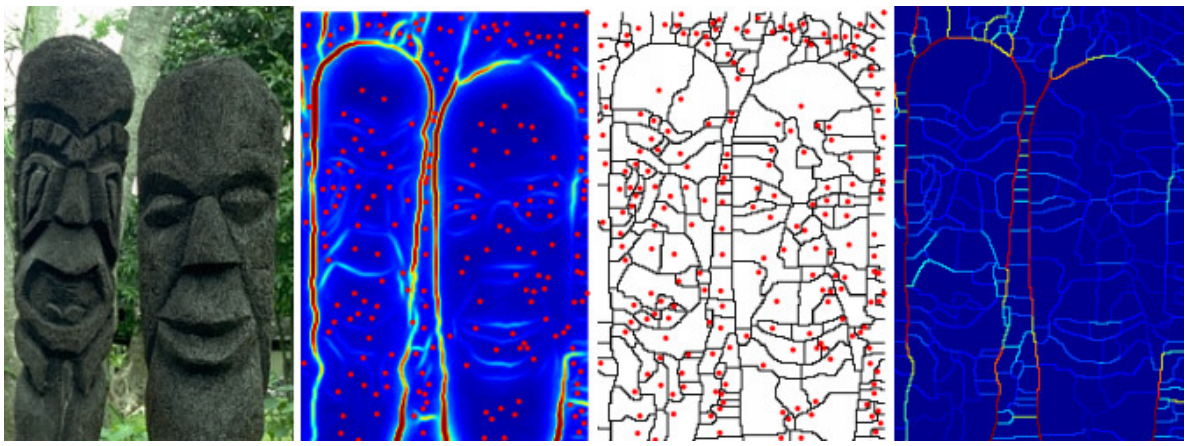
논문	전체 소스 길이 (lines)	코드 길이 (lines)
ORB-SLAM	32435	20359
DSO	25985	16051
Vins-Fusion	33238	23070
LSD-SLAM	27204	16454

C++의 코드 양이 다른 언어에 비해 좀 많긴 하지만 이건 너무 많고 C++ 언어 자체의 난이도까지 감안하면 진입 장벽이 상당히 높다고 볼 수 있습니다.

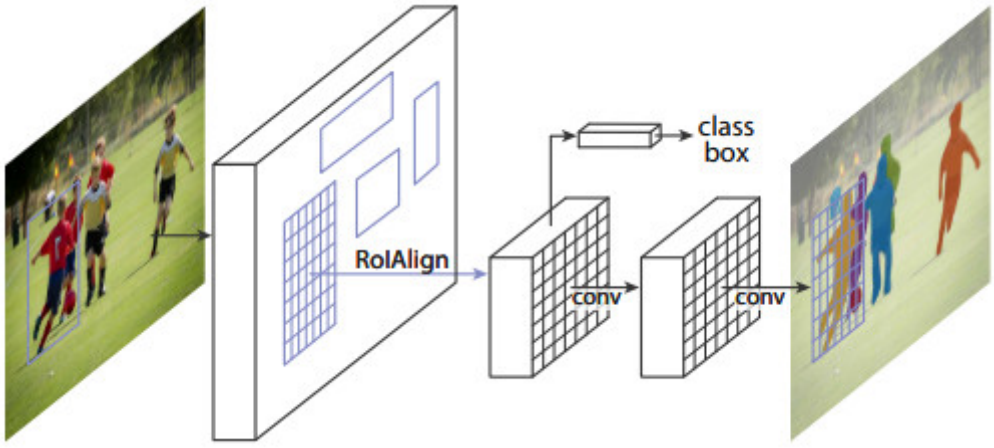
여기서 딥러닝을 써보고 싶은 동기가 생깁니다. 성능이 잘 나올지는 모르겠지만 어쨌건 딥러닝은 아무리 복잡한 과정도 하나의 네트워크로 해결하는 마법을 부리고 많은 경우 성능도 괜찮은 수준으로 나오기 때문입니다. Image Segmentation을 예로 들어보겠습니다.

요즘은 segmentation은 딥러닝으로 하는게 상식이지만 딥러닝이 유행하기 전에 유명했던 논문으로 아래 그림에 나온 "Contour Detection and Hierarchical Image Segmentation"(TPAMI, 2010)이 있습니다. 그 논문을 보면 segmentation결과를 만들기 위해 거쳐야 하는 중간 단계가 수십가지나 됩니다. 논문을 보면서 저자의 장인정신에 감동~~보다는 소름~~이 느껴졌던 기억이 납니다.





하지만 요즘은 Mask-RCNN 등 DNN 하나로 끝나는 세상입니다. 딥러닝을 활용한 연구가 쉬운건 아니지만 예전에 한땀한땀 손으로 빚어가던 시절보다는 수월하게 좋은 성능을 낼 수 있게 됐습니다.



다시 VO로 돌아가보면 상황이 비슷합니다. 다만 차이가 있다면 semantic segmentation이 전문가 시스템으로는 성능에 한계가 뚜렷했던 반면 VO는 이미 전문가 시스템으로 좋은 성능을 내고 있다는 것입니다. 결론부터 말하면 아직까지는 VO에서는 전문가 시스템이 더 좋은 성능을 내고 있습니다. 하지만 그렇기 때문에 연구할 가치가 있는 것이고 연구가 오래되지 않았으므로 앞으로 성능은 개선될 여지가 많을 것입니다.

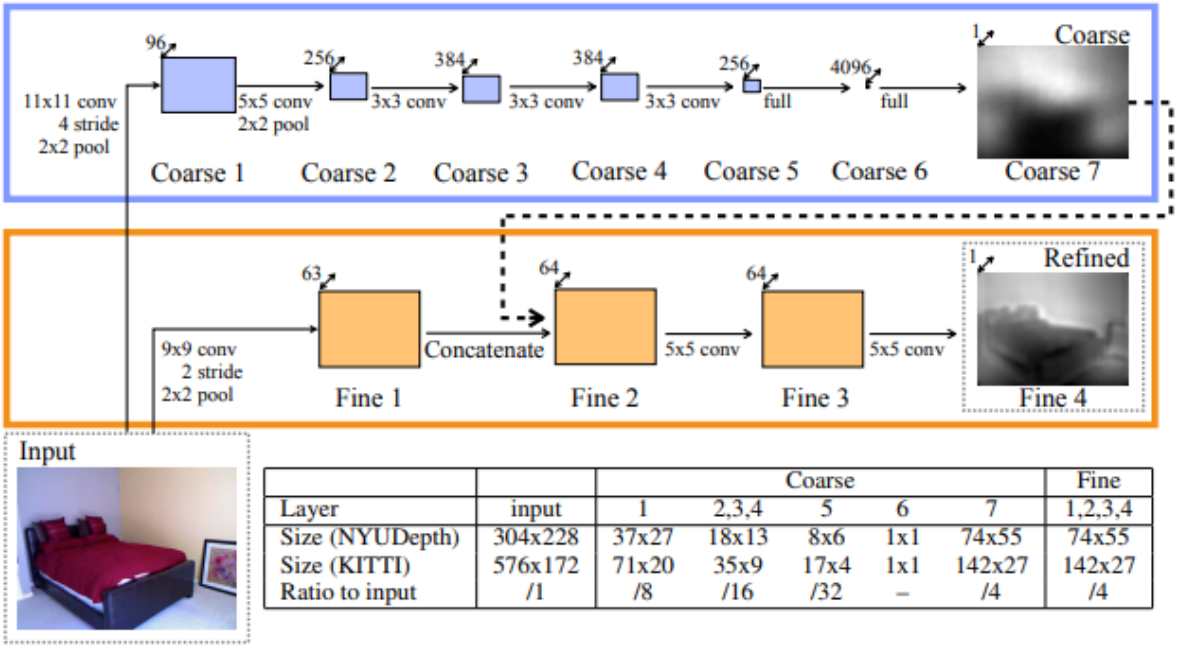
3. (Monocular) Depth Estimation

"Single image (or monocular) depth estimation (or depth prediction)"로 검색할 수 있는 이 분야는 한장의 이미지로부터 영상의 깊이를 추정하는 기술입니다. 여기에 수학적 근거는 없고 semantic segmentation을 학습하듯 입력과 출력을 기계적으로 학습시키는 것 뿐입니다. 그래서 VO와는 다르게 DE에서는 딥러닝을 사용하는 것이 자연스러워 보입니다.

예전에 딥러닝이 유행하기 전에 Andrew Ng의 "Learning Depth from Single Monocular Images" (NIPS, 2006)를 본적이 있는데 영상에서 hand craft feature를 만들어서 실제 depth와 단순히 linear regression을 했습니다. (물론 논문엔 더 많은 내용이 있습니다.)

3.1 Supervised Deep Learning Approach

이후 딥러닝이 유행하면서 딥러닝의 풍부한 convolution feature를 활용한 논문들이 나옵니다. 대표적으로 흔히 'Eigen'이라 불리는 "Depth Map Prediction from a Single Image using a Multi-Scale Deep Network" (NIPS, 2014) 입니다. 아래 그림처럼 global course level depth를 만드는 네트워크와 local fine level depth를 만드는 네트워크 두 가지를 학습시켜 최종적인 depth map을 만듭니다. 어쨌든 이 논문은 RGB 영상을 입력했을 때 데이터셋의 ground truth depth가 나오도록 DNN 모델을 지도 학습(supervised learning)한 것입니다.



		Coarse					Fine
Layer	input	1	2,3,4	5	6	7	1,2,3,4
Size (NYUDepth)	304x228	37x27	18x13	8x6	1x1	74x55	74x55
Size (KITTI)	576x172	71x20	35x9	17x4	1x1	142x27	142x27
Ratio to input	/1	/8	/16	/32	–	/4	/4

논문의 인용수가 구글에서 보면 천건이 넘습니다. 이 논문이 유명한 이유는 (아마) 최초로 딥러닝을 활용한 DE를 구현한 상징성도 있지만 이후 논문에서 마르고 닳도록 사용되는 **성능 평가방법**을 제안했기 때문입니다. 이후 모든 논문들이 "Eigen" 논문과 동일한 방식으로 성능을 평가하고 있습니다. 심지어 학습 데이터와 평가 데이터를 나누는 방식까지 "Eigen split"이라 부르며 따라하고 있습니다. 다음은 CVPR 2018에 나온 GeoNet 논문의 성능 테이블입니다. 맨위에 "Eigen"부터 시작하여 많은 논문들이 같은 기준으로 평가된 것을 볼 수 있습니다.

Method	Supervised	Dataset	Abs Rel	Sq Rel	RMSE	RMSE log	$\delta < 1.25$	$\delta < 1.25^2$	$\delta < 1.25^3$
Eigen *et al.* [9] Coarse	Depth	K	0.214	1.605	6.563	0.292	0.673	0.884	0.957
Eigen *et al.* [9] Fine	Depth	K	0.203	1.548	6.307	0.282	0.702	0.890	0.958
Liu *et al.* [28]	Depth	K	0.202	1.614	6.523	0.275	0.678	0.895	0.965
Godard *et al.* [15]	Pose	K	**0.148**	1.344	5.927	0.247	**0.803**	0.922	0.964
Zhou *et al.* [56]	No	K	0.208	1.768	6.856	0.283	0.678	0.885	0.957
Zhou *et al.* [56] updated[2]	No	K	0.183	1.595	6.709	0.270	0.734	0.902	0.959
Ours VGG	No	K	0.164	1.303	6.090	0.247	0.765	0.919	0.968
Ours ResNet	No	K	0.155	**1.296**	**5.857**	**0.233**	0.793	**0.931**	**0.973**
Garg *et al.* [14] cap 50m	Pose	K	0.169	1.080	5.104	0.273	0.740	0.904	0.962
Ours VGG cap 50m	No	K	0.157	0.990	4.600	0.231	0.781	0.931	0.974
Ours ResNet cap 50m	No	K	**0.147**	**0.936**	**4.348**	**0.218**	**0.810**	**0.941**	**0.977**
Godard *et al.* [15]	Pose	CS + K	**0.124**	**1.076**	**5.311**	**0.219**	**0.847**	**0.942**	**0.973**
Zhou *et al.* [56]	No	CS + K	0.198	1.836	6.565	0.275	0.718	0.901	0.960
Ours ResNet	No	CS + K	0.153	1.328	5.737	0.232	0.802	0.934	0.972

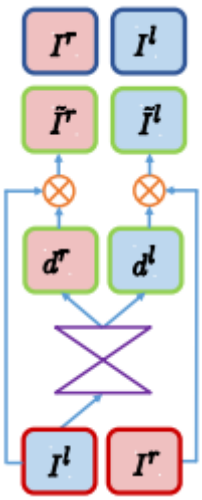
3.2 Unsupervised Deep Learning Approach

"Eigen"에서 한 단계 더 나아가 비지도학습으로 depth를 학습하는 논문이 나옵니다. 논문 제목은 "Unsupervised Monocular Depth Estimation with Left-Right Consistency" (CVPR, 2017), 줄여서 **monoDepth**라고 부릅니다. 이 논문은 다음과 같은 원리를 이용합니다.

1. 스테레오 카메라에서 카메라 사이의 상대 자세를 알고
2. depth혹은 disparity를 알 수 있다면
3. 오른쪽 시점의 이미지를 왼쪽 이미지로부터 재구성(reconstruct) 할 수 있다.

monoDepth의 학습 목표는 오른쪽 이미지를 재구성 하는 것이 아니라 **오른쪽 이미지를 재구성 할 수 있는 depth를 학습**하는 것입니다. 구하기 어려운 GT depth가 필요없고 비교적 구하기 쉬운 스테레오 이미지만 있으면 depth를 학습할 수 있기 때문에 학습을 더 쉽게 할 수 있습니다.

DNN 구조는 아래와 같이 좁아졌다 넓어지는 Encoder-Decoder 구조로 되어있고 왼쪽 이미지(I^l)로부터 두 개의 disparity map을 만듭니다. (**left-to-right disparity (d^r), right-to-left disparity (d^l)**) Disparity map으로 두 이미지를 반대쪽 시점으로 재구성하여 반대쪽 시점의 이미지와 같아지도록 DNN을 학습시키는 것입니다. 재구성한 이미지와 원래 이미지 사이의 차이를 **photometric loss**라고 하는데 이 loss가 이후 오늘 공부할 VODE 논문의 주요 loss가 됩니다.

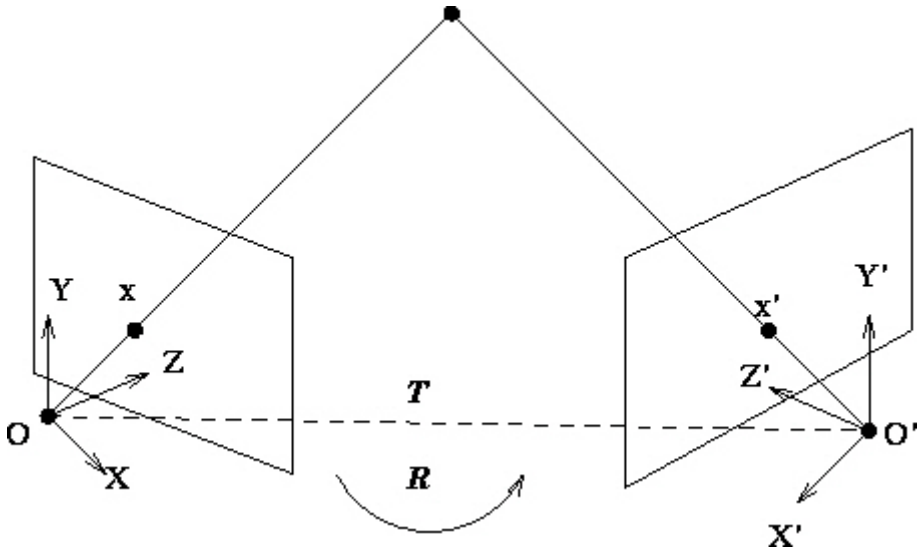


Monocular Depth Estimation은 아직 세밀한 정확도는 부족하기 때문에 LiDAR를 완전히 대체하긴 어렵지만 대략적인 depth라도 있다면 위치인식에 도움을 줄 수 있고 자동차가 아닌 로봇의 경우 판단오차가 치명적이지 않으므로 장애물 회피 등에 사용될 수 있습니다.

오늘 배울 VODE는 VO와 DE를 동시에 학습하는 것인데 monoDepth 처럼 스테레오 이미지 없이 단안 카메라의 연속 프레임만 가지고도 VO와 DE를 동시에 학습할 수 있는 기술입니다. 어느쪽을 주 목표로 하던 따로 학습하는 것보다는 동시에 학습하는 것이 성능이 좋기 때문에 대부분의 논문들이 이런 흐름으로 가고 있습니다.

4. Pose Representation

VO를 학습을 통해 해결할 수 있다고 했는데 VO-DNN의 출력은 어떻게 나와야 할까요? 두 프레임 사이의 상대적인 자세를 어떻게 표현해야 할까요?



두 개의 카메라가 (혹은 한 카메라가 이동한 두 개의 시점이) 있습니다. 이들 사이의 상대적인 자세(pose)는 두 카메라 사이의 상대적인 이동(translation)과 회전(rotation)으로 표현할 수 있습니다. 이를 다르게 표현하면 두 카메라 사이의 상대 pose는 두 카메라 좌표계 사이의 **Rigid Transformation** (translation + rotation)으로 표현할 수 있다는 뜻입니다. 이동은 직관적으로 두 카메라 사이의 상대 위치를 3차원 벡터로 표현하면 되지만 회전을 표현하는 방법은 제가 알기로 최소 네 종류가 있습니다. 이에 따라 rigid transformation도 네 가지로 표현할 수 있습니다. 3차원 VO를 하기 위해 알아야할 기반 지식이지만 제가 알기로 한글로 이런 지식을 설명하는 책은 없습니다. 원서를 공부해야 하는데 제가 간단하게 개념을 요약해 드릴테니 자세한 내용은 아래 원서에서 필요한 부분만 참고하시기 바랍니다.

- An Invitation to 3-D Vision, Yi Ma
- Multiple View Geometry in Computer Vision, Richard Hartley

- Quaternions and Rotation Sequences, Jack B. Kuipers

4.1 Special Euclean Transformation

카메라의 pose는 rigid transformation (or rigid body motion)으로 표현할 수 있다고 했는데 rigid transformation을 좀더 기술적인(?) 용어로 말하면 **Speical Euclidean Transformation** 입니다.

일단 앞에 "speical"은 떼로 Euclidean Transformation만 정의해보면 다음과 같습니다. 쉽게 말해 모양이 늘어나거나 줄어들거나 뒤틀리지 않고 똑같이 유지된다는 것이죠.

> **Euclidean transformation**: a map that preserves the Euclidean distance between every pair of points. The set of all Euclidean transformation in 3-D space is denoted by $E(3)$.
>
> $$g : \mathbb{R}^3 \to \mathbb{R}^3; X \mapsto g(X)$$
> $$\| g_*(v) \| = \| v \|, \forall v \in \mathbb{R}^3$$

"Speical" Euclidean Transformation은 여기에 한가지 조건을 더 붙입니다. 방향이 바뀌지 않아야 한다는 것이죠.

> The map or transformation induced by a rigid-body motion is called a **special Euclidean transformation**. The word "*special*" indicates the fact that a transformation is **orientation-preserving**.

$$g_*(u) \times g_*(v) = g_*(u \times v), \forall u, v \in \mathbb{R}^3$$

아무리 모양을 유지시킨채 회전을 시켜도 좌우반전은 일어나지 않습니다. 상하반전도 마찬가지입니다. 좌우반전이나 상하반전 같은 변환은 Euclidean Transformation은 만족하지만 물리적인 3차원 공간에서는 일어날 수 없는 일입니다. 우리가 일반적으로 쓰는 직교 좌표계는 $\mathbf{X} \times \mathbf{Y} = \mathbf{Z}$ 를 만족하는데 이 좌표계를 아무리 "회전"시켜도 저 등식은 성립합니다. 하지만 좌우반전으로 X축만 뒤집으면 ($\mathbf{X}' = -\mathbf{X}$) 변환된 좌표계에서는 $\mathbf{X}' \times \mathbf{Y} = -\mathbf{X} \times \mathbf{Y} = -\mathbf{Z}$ 이므로 이를 만족할 수 없게됩니다.

그래서 Euclidean Transformation 중에 **orientation-preseving**한 부분 집합이 Speical Euclidean Transformation이고 이것이 물리 세계의 강체 변환(Rigid Transformation)과 동일한 의미를 갖게 됩니다. (강체(rigid body)란 단단해서 모양을 변형할 수 없는 물체이므로 당연히 좌우반전도 할 수 없습니다.)

이제 3-D Special Euclidean Transformation, 줄여서 **"SE(3)"**라 표기하는 transformation 혹은 pose를 표현하는 방법은 여러가지가 있습니다. 이동(translation)은 공통적으로 3차원 벡터로 표현하지만 회전(rotation)을 표현하는 방법에 따라 달라집니다. 다음은 회전을 표현하는 네 가지 방법입니다.

- Rotation matrix
- Euler angle
- Quaternion
- Twist coordinates

이제 저 표현들을 하나씩 알아봅시다.

4.2 Rotation matrix

회전(rotation) 변환은 아래 그림처럼 원점은 그대로 둔채 특정 축을 중심으로 좌표계를 회전시키는 변환입니다.

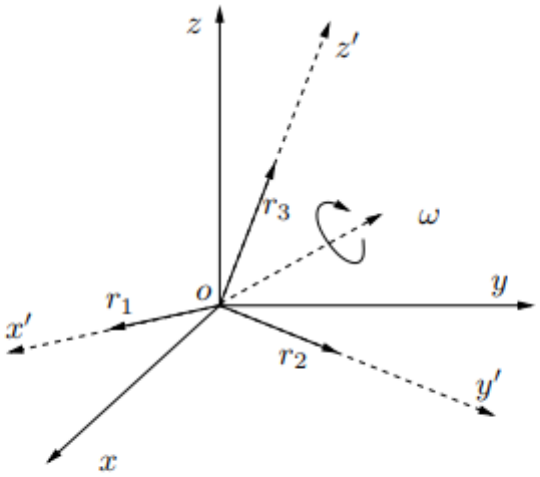


Figure 2.2. Rotation of a rigid body about a fixed point o. The solid coordinate frame W is fixed and the dashed coordinate frame C is attached to the rotating rigid body.

회전 변환은 3x3 matrix로 표현할 수 있는데 임의의 3x3 matrix가 모두 **rotation matrix** 가 될 수 있는건 아닙니다. 3x3 matrix 중에서 rotation matrix가 될 수 있는 matrix 집합을 **Special Orthogonal group, SO(3)**라고 부르고 집합의 조건은 다음 식의 (1)과 같습니다. (2), (3)은 speical euclidean trnasformation의 조건을 회전변환으로 다시 쓴 것입니다.

$$(1) \quad SO(3) \doteq \left\{ R \in \mathbb{R}^{3\times 3} \,\middle|\, R^T R = I,\ det(R) = +1 \right\}$$
$$(2) \quad |\mathbf{x}| = |R\mathbf{x}|$$
$$(3) \quad (R\mathbf{x_1}) \times (R\mathbf{x_2}) = R(\mathbf{x_1} \times \mathbf{x_2})$$

$R^T R = I$ 조건을 만족하는 matrix 집합을 orthogonal group, $O(3)$라고 하는데 이는 euclidean transformation의 회전에 해당합니다. Orthogonal matrix는 (2)처럼 변환이 벡터의 크기에 영향을 주지 않습니다.

Orthogonal group에 $det(R) = +1$ 조건까지 더해야 비로서 (3)을 만족할 수 있고 이것이 speical euclidean transformation에 해당하는 "speical" orthogonal group (a.k.a rotation matrix) 입니다.

Rotation matrix (R)에 translation vector ($\mathbf{t}$)를 더해 완성한 SE(3)의 변환 행렬 (transformation matrix)는 다음과 같습니다.

$$SE(3) \doteq \left\{ T = \begin{bmatrix} R & \mathbf{t} \\ 0 & 1 \end{bmatrix} \middle\| \; R \in SO(3), \mathbf{t} \in \mathbb{R}^3 \right\} \in \mathbb{R}^{4\times 4}$$

4.3 Euler angle

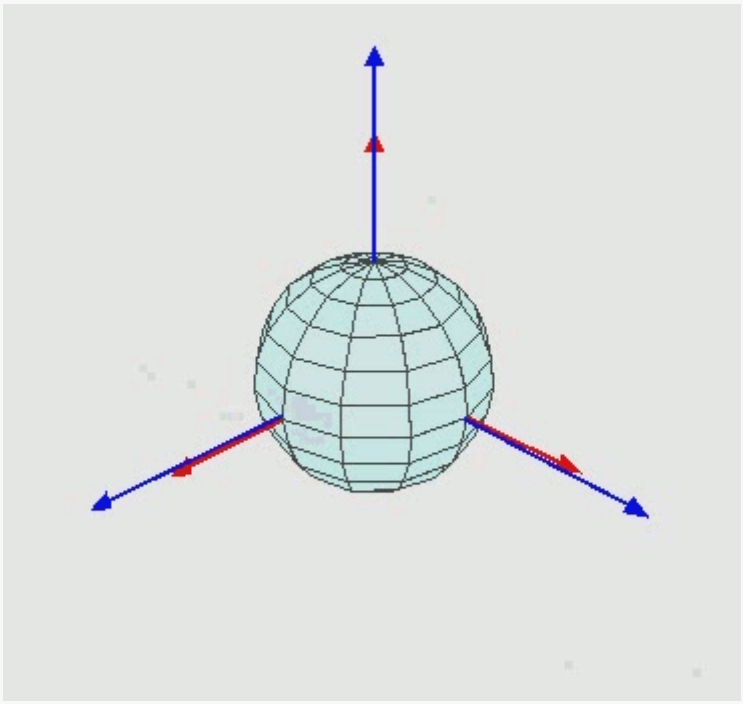
Euler angle이란 임의의 회전을 세 번의 직교 좌표계 축 회전으로 표현하는 방법입니다. 가장 간단예로 xyz euler angle이 있습니다. 직교 좌표계에서 x축으로 α, y축으로 β, z축으로 γ 만큼 회전하면 어떤 회전이라도 표현할 수 있다는 것입니다.

$$\forall R \in SO(3), \exists\, \alpha, \beta, \gamma$$
$$R = R_x(\alpha) R_y(\beta) R_z(\gamma)$$

$$R_x(\alpha) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos\alpha & sin\alpha \\ 0 & -sin\alpha & cos\alpha \end{bmatrix} \quad R_y(\beta) = \begin{bmatrix} cos\beta & 0 & -sin\beta \\ 0 & 1 & 0 \\ sin\alpha & 0 & cos\alpha \end{bmatrix} \quad R_z(\gamma) = \begin{bmatrix} cos\gamma & sin\gamma & 0 \\ -sin\gamma & cos\gamma & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

> 위키를 찾아보면 rotation matrix에 나오는 행렬들과는 부호가 조금 다릅니다. 위키에 나온 회전은 좌표축을 회전시킨채 점을 회전시키는 rotation matrix고, 여기에서는 카메라 좌표축 자체가 회전하기 때문에 좌표계의 회전을 표현하기 때문입니다.

회전 순서를 꼭 xyz 순서로 할 필요는 없고 아래 그림과 같은 zxz euler angle도 많이 쓰입니다. 그외에도 여러가지 조합이 있는데 연속으로 같은 축에 대해서 회전하지만 않으면 어떠한 조합도 euler angle이 될 수 있습니다. 예를 들어 xxy, zyy는 안되지만 xyx, yxz는 됩니다.



zyz euler angle rotation, 출처

Euler angle은 직교좌표계에 익숙한 사람들에게 직관적으로 이해되지만 단점이 많아서 단점을 잘 이해하고 써야합니다. Euler angle은 유명한 짐벌락 문제(gimbal lock problem, 영상)이 있어서 각도 범위에 제한이 있습니다.

Euler angle을 이용해 SE(3) 변환을 표현한다면 다음과 같은 6차원 벡터가 될 것입니다.

$$\mathbf{p} = \begin{bmatrix} t_x & t_y & t_z & \alpha & \beta & \gamma \end{bmatrix}^T$$

4.4 Quaternion

4개의 숫자로 표현할 수 있는 quaternion은 일종의 복소수이며 스칼라 q_0와 벡터 $\mathbf{q}$로 이루어졌습니다. 상황에 따라 간단히 4차원 벡터로 표현하기도 합니다.

$$q = q_0 + \mathbf{q} = q_0 + q_1\mathbf{i} + q_2\mathbf{j} + q_3\mathbf{k}$$

$$q = \begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{bmatrix}$$

Quaternion으로 회전을 표현할 때는 크기가 1인 **unit quaternion**만 사용할 수 있습니다. ($|q| = 1$) Unit quaternion에서 스칼라 q_0는 각도를 의미하고 벡터 $\mathbf{q}$는 회전축을 의미합니다. SO(3)에 속하는 임의의 회전변환은 특정 축을 기준으로 특정 각도를 회전시켜 만들수 있습니다. Euler angle은 직교 좌표축으로만 회전했기 때문에 세 번 회전해야 임의의 회전을 만들어낼 수 있지만 임의의 회전축을 사용할 수 있다면 한번의 회전으로 모든 회전을 구현할 수 있습니다.

어떤 회전이 회전축 $\mathbf{v}$를 기준으로 θ 만큼 회전해야 한다면 quaternion으로 다음과 같이 표현할 수 있습니다.

$$q = cos\frac{\theta}{2} + \mathbf{v}sin\frac{\theta}{2}$$

$$q = \begin{bmatrix} cos\frac{\theta}{2} \\ v_x sin\frac{\theta}{2} \\ v_y sin\frac{\theta}{2} \\ v_z sin\frac{\theta}{2} \end{bmatrix}$$

Quaternion으로 실제로 어떤 좌표계를 회전시켜야 한다면 quaternion product를 계산해야 합니다. Quaternion 사이의 곱셈에 해당하는 quaternion product는 다음과 같이 계산합니다.

$$pq = p_0 q_0 - \mathbf{p} \cdot \mathbf{q} + p_0 \mathbf{q} + q_0 \mathbf{p} + \mathbf{p} \times \mathbf{q}$$

$$pq = \begin{bmatrix} p_0 & -p_1 & -p_2 & -p_3 \\ p_1 & p_0 & -p_3 & p_2 \\ p_2 & p_3 & p_0 & -p_1 \\ p_3 & -p_2 & p_1 & p_0 \end{bmatrix} \begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{bmatrix} = \begin{bmatrix} q_0 & -q_1 & -q_2 & -q_3 \\ q_1 & q_0 & q_3 & -q_2 \\ q_2 & -q_3 & q_0 & q_1 \\ q_3 & q_2 & -q_1 & q_0 \end{bmatrix} \begin{bmatrix} p_0 \\ p_1 \\ p_2 \\ p_3 \end{bmatrix}$$

어떤 3차원 벡터 $\mathbf{p}$를 q라는 quaternion으로 회전된 좌표계에서 본 $\mathbf{p}'$ 를 계산하는 식은 다음과 같습니다. Quaterion product를 앞 뒤로 두 번하는 것입니다.

$$\bar{p}' = q^* \bar{p} q$$

$$\bar{p} = \begin{bmatrix} \mathbf{p} \\ 0 \end{bmatrix}, \quad q^* = q_0 - \mathbf{q}$$

이러한 quaternion 연산이 부담스럽다면 rotation matrix로 변환할수도 있습니다.

$$\bar{p}' = q^* \bar{p} q$$

$$\mathbf{p}' = Q\mathbf{p}$$

$$Q = \begin{bmatrix} 2q_0^2 - 1 + 2q_1^2 & 2q_1 q_2 - 2q_0 q_3 & 2q_1 q_3 + 2q_0 q_2 \\ 2q_1 q_2 + 2q_0 q_3 & 2q_0^2 - 1 + 2q_2^2 & 2q_2 q_3 + 2q_0 q_1 \\ 2q_1 q_3 - 2q_0 q_2 & 2q_2 q_3 + 2q_0 q_1 & 2q_0^2 - 1 + 2q_2^3 \end{bmatrix}$$

반대로 rotation matrix를 quaternion으로 변환하고 싶다면 임의의 rotation matrix에서 회전 축 $\mathbf{v}$와 회전 각도 θ를 알아내면 됩니다.

$$R = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}$$

$$\theta = acos\left(\frac{Tr(R)-1}{2}\right)$$

$$\mathbf{v} = \frac{1}{2sin\theta}\begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} = \begin{bmatrix} r_{23} - r_{32} \\ r_{31} - r_{13} \\ r_{12} - r_{21} \end{bmatrix}$$

$$q = cos\frac{\theta}{2} + \mathbf{v}sin\frac{\theta}{2}$$

Quaternion으로 회전을 표현할때의 장점은 다음과 같습니다.

- 짐벌락 문제와 같은 표현 자체의 결점이 없음
- 임의의 회전을 회전축과 회전각을 이용해 직관적으로 표현 가능
- 9개의 숫자로 표현되고 복잡한 조건을 가진 rotation matrix에 비해 quaternion은 4차원 벡터로 표현할 수 있고 조건이 단순해서 (unit quaternion) 경제적이고 최적화에도 유리함
- rotation matrix를 거치지 않고 quaternion product를 통해 연속적인 회전을 직접 연산가능, $R_3 = R_1 R_2 \leftrightarrow q_3 = q_1 q_2$

Quaternion을 이용해 SE(3) 변환을 표현한다면 이동 벡터와 합친 7차원 벡터가 됩니다.

$$\mathbf{p} = \begin{bmatrix} t_x & t_y & t_z & q_w & q_x & q_y & q_z \end{bmatrix}^T$$

4.5 Twist Coordinates

*Twist*의 의미를 이해하기 위해서는 어려운 용어들과 미분방정식을 푸는 유도과정을 봐야하지만 여기선 생략하고 결과적인 사용방법만 알아보겠습니다. ORB-SLAM이나 LSD-SLAM 등 대부분의 유명한 VO/SLAM 논문에서는 이 방법으로 회전을 표현합니다.

앞서 quaternion이 euler angle의 문제를 해결했는데 왜 또 다른걸 배워야 할까요? Quaternion도 사용해보면 완벽하게 편하진 않기 때문입니다. Rotation matrix보다는 단순하지만 **unit** quaternion이라는 조건도 걸려있고 원래 3자유도의 회전을 4차원 벡터로 표현하는 것도 아쉬운 점입니다. 앞서 임의의 rotation matrix를 회전각도 t와 회전축 ω로 해석할 수 있다고 했습니다.

$$R = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}$$

$$t = acos\left(\frac{Tr(R)-1}{2}\right)$$

$$\omega = \frac{1}{2sin(t)}\begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \end{bmatrix} = \begin{bmatrix} r_{23} - r_{32} \\ r_{31} - r_{13} \\ r_{12} - r_{21} \end{bmatrix}$$

반대로, 회전각도 t와 회전축 ω로부터 rotation matrix R을 만들어낼 수 있는 공식이 있습니다. 바로 로드리게스 공식입니다.

> **Rodrigues' formula for rotation matrix**: Given $\omega \in \mathbb{R}^3$ with $\| \omega \| = 1$ and $t \in \mathbb{R}^3$, the matrix exponential $R = e^{\hat{\omega}t}$ is given by the following formula:

$$e^{\hat{\omega}t} = I + \hat{\omega}sin(t) + \hat{\omega}^2(1 - cos(t))$$

$$\hat{\omega} = \begin{bmatrix} 0 & -\omega_3 & \omega_2 \\ \omega_3 & 0 & -\omega_1 \\ -\omega_2 & \omega_1 & 0 \end{bmatrix}$$

$\hat{\omega}$은 **twist** 혹은 **exponential coordinate**이라 불리며 임의의 3차원 벡터 ω로부터 만들어질 수 있는 $\hat{\omega}$의 집합을 $so(3)$라고 합니다. 자세히 보면 $\hat{\omega}$의 모양이 벡터 ω와의 cross product에 대한 행렬연산이라는 것을 알 수 있습니다.

$$\mathbf{u} \times \mathbf{v} = \begin{bmatrix} 0 & -u_3 & u_2 \\ u_3 & 0 & -u_1 \\ -u_2 & u_1 & 0 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} = \hat{\mathbf{u}}\mathbf{v}$$

임의의 twist $\hat{\mathbf{u}} = \hat{\omega}t$는 **exponential map**을 통해 rotation matrix가 될 수 있고 exponential map을 구현한 식이 로드리게스 공식입니다.

$$exp : so(3) \to SO(3)$$
$$\hat{\mathbf{u}} \in so(3) \mapsto R = e^{\hat{\mathbf{u}}} \in SO(3)$$

다시 정리해보면 세 단계의 표현 방식이 있습니다.

term	**expression**
twist coordinates	$\mathbf{u} = \omega t \in \mathbb{R}^3$
twist or exponential coordinates	$\hat{\mathbf{u}} = \begin{bmatrix} 0 & -u_3 & u_2 \\ u_3 & 0 & -u_1 \\ -u_2 & u_1 & 0 \end{bmatrix} \in so(3)$
rotation matrix	$R = e^{\hat{\mathbf{u}}} \in SO(3)$

세 가지 표현 모두 동일한 정보를 가지고 있으므로 아무런 제약조건이 없는 3차원 벡터인 "twist coordinates"로 임의의 회전을 표현할 수 있다는 뜻이 됩니다. 이는 앞서 배운 rotation matrix, euler angle, quaternion의 모든 단점들이 해소된 표현 방법이라고 볼 수 있습니다. 3차원으로 표현이 compact하고 아무런 제약조건도 없습니다.

벡터의 표현도 직관적으로 이해할 수 있습니다. Twist coordinate $\mathbf{u} = \omega t$ 에서 크기 $t = \| \mathbf{u} \|$는 각도를 의미하고 방향 $\omega = \frac{\mathbf{u}}{\| \mathbf{u} \|}$ 은 회전축을 의미합니다.

다만 주의할점은 각도가 2π마다 반복되므로 twist coordinates와 rotation matrix는 one-to-one 관계가 아니라 many-to-one 관계라는 것입니다.

$$R = e^{\hat{\omega} t} = e^{\hat{\omega}(t + 2\pi n)}$$

위에서 본 세 단계의 표현은 이동(translation)을 더한 rigid transformation에도 그대로 적용됩니다. 이동을 $\mathbf{v} \in \mathbb{R}^3$로 표현할 때 이동을 포함한 twist coordinates와 twist는 다음과 같이 표현합니다.

$$\xi = \begin{bmatrix} \mathbf{v} \\ \mathbf{u} \end{bmatrix} \in \mathbb{R}^6$$
$$\hat{\xi} = \begin{bmatrix} \hat{\mathbf{u}} & \mathbf{v} \\ 0 & 1 \end{bmatrix} \in se(3)$$

이로부터 transformation matrix를 구하는 Rodrigues' formula는 다음과 같습니다.

$$e^{\hat{\xi} t} = \begin{bmatrix} e^{\widehat{w}} & \left(I - e^{\widehat{w}}\right) \widehat{w} v + w \widehat{w}^T v \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} R & T \\ 0 & 1 \end{bmatrix} \in SE(3)$$

여기까지 3차원 자세를 나타내는 다양한 방법에 대해서 알아봤습니다. 다음 시간에 배울 VODE에서도 저자마다 다른 표현 방식을 쓰는데 어떤 논문은 euler angle을 쓰고 어떤 논문은 twist coordinate을 쓰기도 합니다. 이러한 표현 방법을 정리해두면 VO/SLAM 관련 논문을 이해하는데 큰 도움이 됩니다.