EECS C106B / 206B Robotic Manipulation and Interaction **Spring 2022**

Lecture 13: Epipolar Equations, Two View/Linear/Nonlinear Geometry

Scribes: Nima Rezaeian, Jaren Mendelsohn

As described in the last lecture, computer vision often consists of determining scene structure and camera pose from a set of 2D images. The first step in this pipeline involves picking ”interesting” points out of each image – this could be edges of objects or defined points. The second step in this cycle involves matching these points between images, determining how these features vary between each image and allowing us to recover depth and camera transformations between each image.

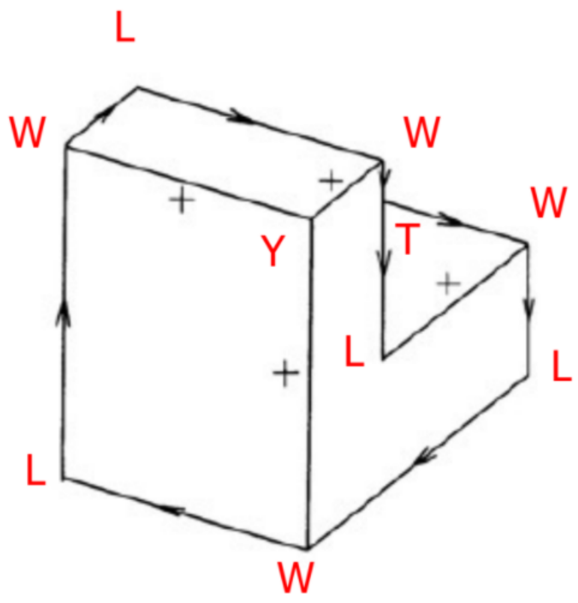
1.1 Feature Detection

Last week we covered a couple algorithms which accomplish this (MSER, SURF, FAST/FASTER, BRISK, ORB, KAZE), with each one including its own tradeoffs and benefits. We discussed SIFT which is often useful, but not optimal in situations where we're trying to match points between images with perspective distortion or large transformations between them. In these cases, we must consider more advanced features which handle affine transformations better. Pretty much all major companies have their own brand of these such feature detectors (e.g. Microsoft has its Mother of All Features). Our choice between these depends on our scenario: for robotics, we often prioritize real-time processing and thus a less accurate but faster algorithm could be preferred.

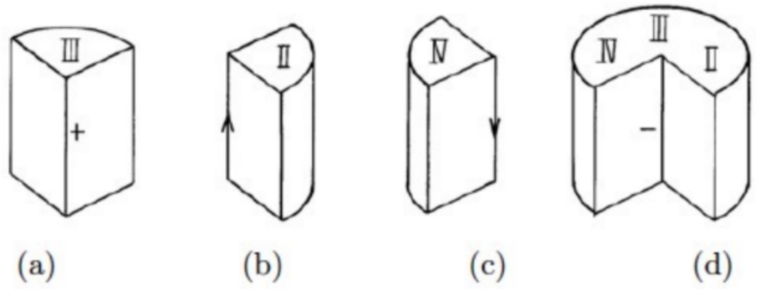
1.1.1 Local detection

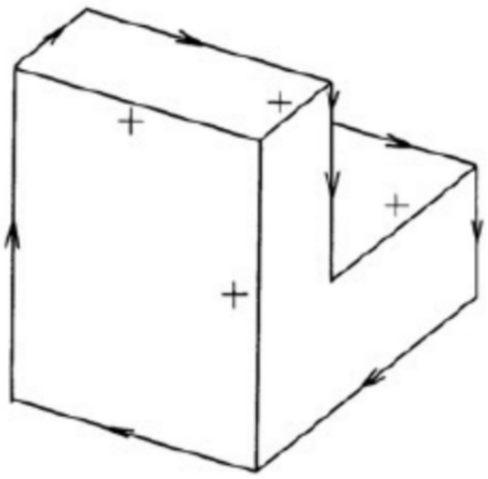
The majority of these papers describe methods for statistically evaluating features locally by only considering the neighborhood around points. Human vision, on the other hand, seems to evaluate significant points on a holistic level, using prior knowledge and the scene as a whole in order to determine edges and object structure. Additionally, human vision is very powerful even with low-resolution or low-detail images: given a simple wireframe structure in 2D, we can easily determine the entire object's perspective.

One way to approach this holistic view is by considering junctions. Junctions could be L, W, Y, or T type (basically corresponding to the shapes of those letters). The image below demonstrate one such example. Note that the T junction doesn't correspond to an actual vertex of the object, but we as humans know that it occludes the rest of the face.

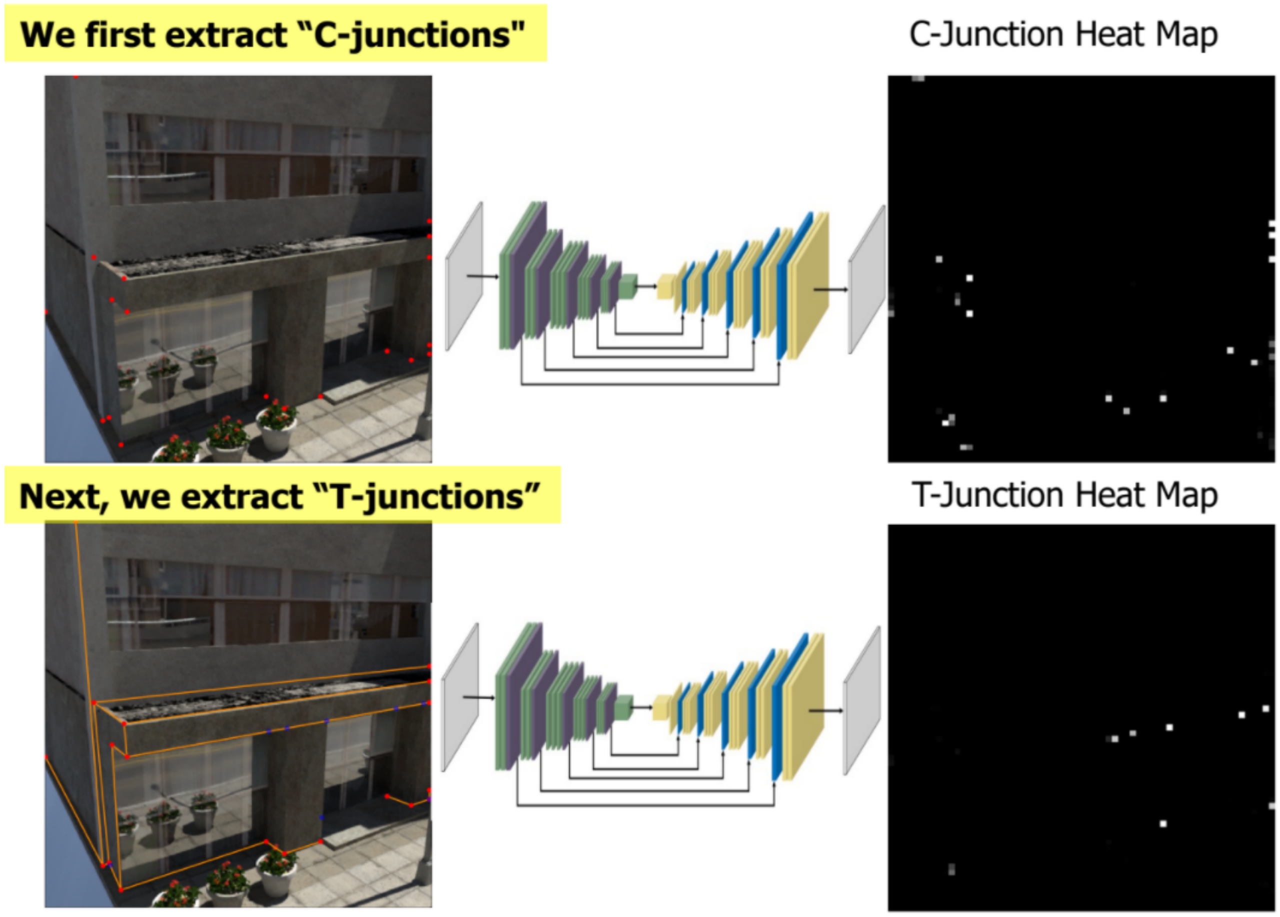


Additionally, we could characterize each edge of the object in addition to junctions: in a polyhedron, each edge must be convex (+), concave (-), or occluding (←, →, hiding face information behind the edge with the right hand arrow direction indicating inside of the object).



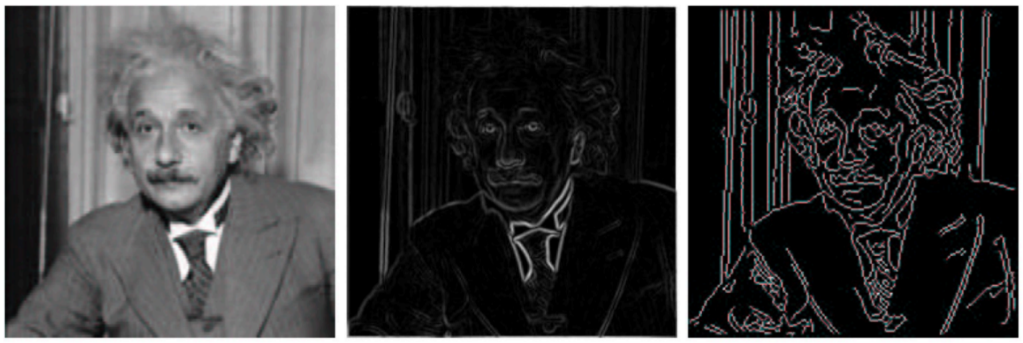


One of the students in Professor Ma's lab took a deep learning approach to this junction problem, extracting these junctions with neural nets.

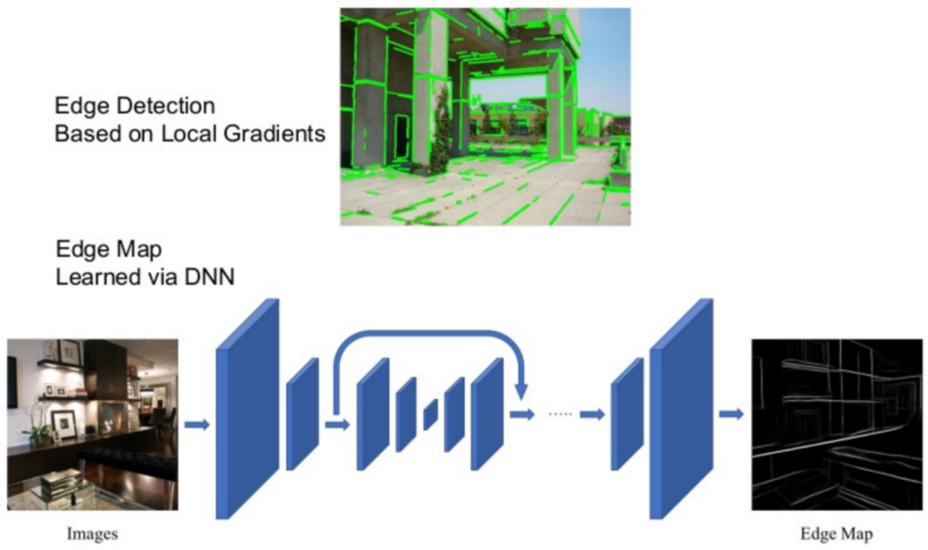

We first extract "C-junctions"
C-Junction Heat Map
Next, we extract "T-junctions"
T-Junction Heat Map

1.1.2 Edge detection

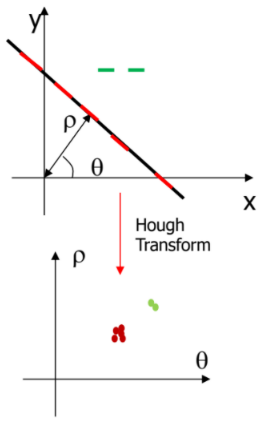
How do we actually calculate these edges? One potential way would be to consider the gradient of the image in the horizontal and vertical direction (i.e. $I_x(x,y) = \frac{dI}{dx}(x,y), I_y(x,y) = \frac{dI}{dy}(x,y)$). If this gradient is above a threshold and the value is a local maximum along the direction, this pixel could be an edge candidate. This is used in the Canny Edge Detection algorithm (created by Berkeley professor John Canny!).

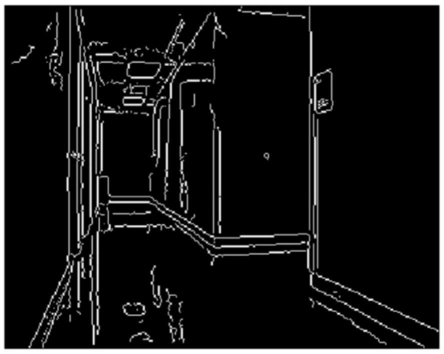


One additional way to calculate these edges involves deep learning, they could potentially produce better results than simply considering gradients. This approach, however, still only considers the image as a bitmap – it doesn't necessarily consider the image holistically with contextual information as humans do.



Once we have these edges on a bitmap, we must convert them into vectorized lines. One approach to this involves a Hough transform, which converts each line segment on an edge into a different parameter space of its distance from the origin and angle. Line formation in this new space is then reduced to a clustering problem, and we can use an algorithm such as k-means to accomplish this. We can also determine line segments by considering the second moment matrix of each connected edge component, determing lines from eigenvalues/eigenvectors of this matrix.



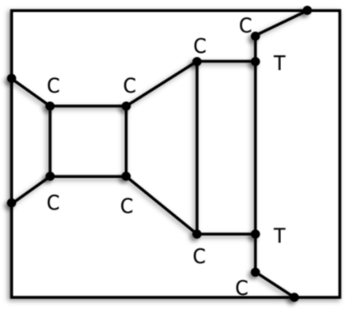


Non-max suppressed gradient magnitude

With these line segments, it's possible to determine vanishing points. If we know the location of a vanishing point of two lines (parallel lines in space converge to the same point in 2D space), we can more easily determine the pose in 3D space of each line. There are multiple algorithms for accomplishing this with line clustering (J-Linkage, Line RANSAC, Angle Histogram, etc.).

1.1.3 Representations

Many of the algorithms for determining features deal with point clouds, but this isn't a great representation for interfacing with humans: they're hard to visualize. In these situations, a wireframe could be more useful. In the below image, we can immediately determine the scene structure from the lines (C and T represent different types of junctions).

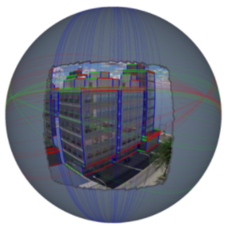


Some questions are still outstanding in this field:

How do we ensure that the same lines are detected from different views? Perspective changing might result in some features being detected in one image but not another

What is the descriptor for establishing line correspondences?

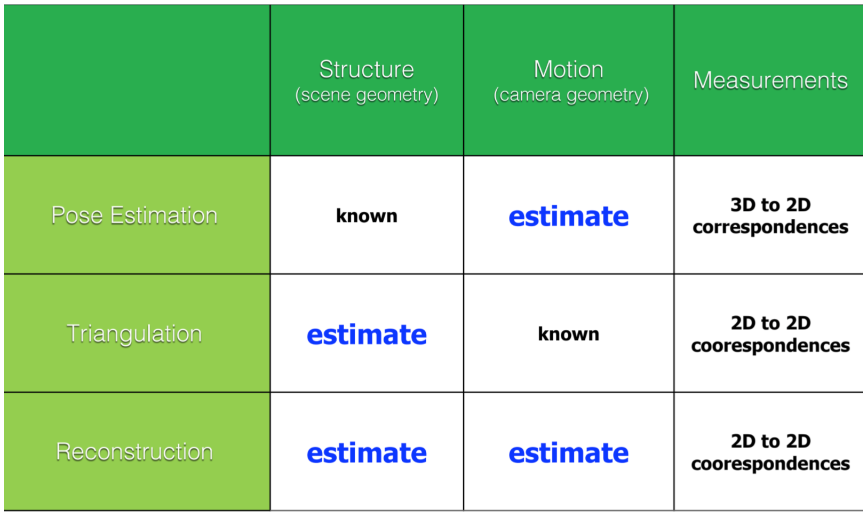
How can we move beyond junctions and lines? More complex representations such as planes could be incredibly useful. If we have two orienting planes for walls in a room, we can immediately localize ourselves and determine the geometry of the room. In the picture of the cars below, if we have multiple planes forming a bounding box around each vehicle, the movement of this box allows us to determine the relative velocity and pose of each car much more easily. Additionally, consider the problem of placing a coke can on each of the desks depicted below in VR. It's easy to visualize but difficult to implement – planar detection could help with this. There's a tremendous amount of information in images which we as humans are able to process, but computers have a long way to go.



1.2 Two-View Geometry

So far, we understand how the pinhole camera model works, and how our eyes function as a projection sensor from 3D to 2D. We can also extract areas of interest in images and make correspondences between feature points. Image correspondences can then be used to recover depth and camera transformations.

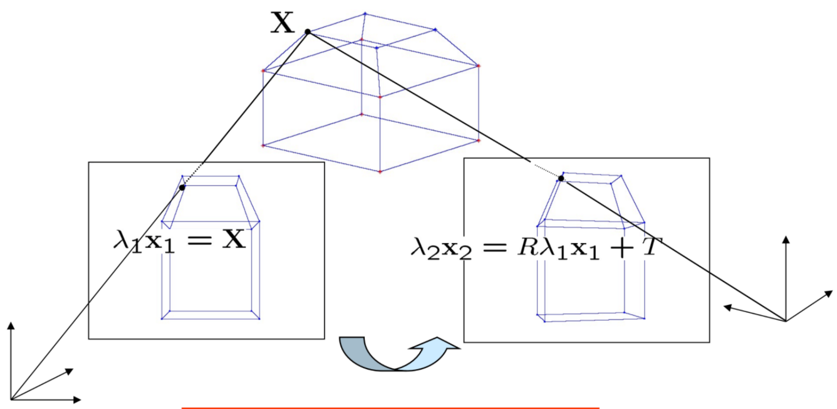
To give an overview of problems in computer vision that use correspondences, consider the slide below:



	Structure (scene geometry)	Motion (camera geometry)	Measurements
Pose Estimation	known	estimate	3D to 2D correspondences
Triangulation	estimate	known	2D to 2D coorespondences
Reconstruction	estimate	estimate	2D to 2D coorespondences

Pose estimation and triangulation are when we have either structure or motion but not both. Reconstruction is the hardest of three, where both are unknown. We now seek to find a way to find the motion (transformation) between two cameras first, and then use the transformation to triangulate the structure (depth/geometrical model) of the scene.

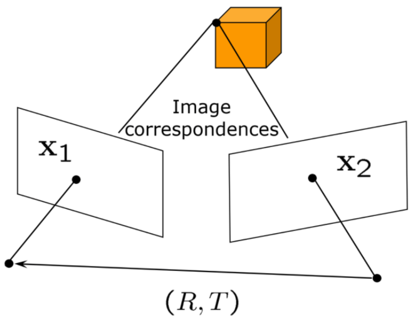
Consider a single point X in world coordinates. By perspective projection, $\lambda x_1 = \Pi X$. Since the second camera is transformed by (R, T) relative to the first, we get the equation below:


$\mathbf{X}$
$\lambda_1 \mathbf{x}_1 = \mathbf{X}$
$\lambda_2 \mathbf{x}_2 = R\lambda_1 \mathbf{x}_1 + T$

$$\boxed{\lambda_2 \mathbf{x}_2 = R\lambda_1 \mathbf{x}_1 + T}$$

Note that the only known quantities in this equation are x_1 and x_2. If we were to naively optimize for an error metric such as the re-projection error using correspondence points, we would have a difficult nonlinear optimization problem that wouldn't guarantee us the optimal solution (see Slide no.6 from lecture).

But there is a better way! As mentioned before, let's try to tackle the motion problem first by trying to eliminate the depth out of the equation we have to solve:



$$\lambda_2 \mathbf{x}_2 = R \lambda_1 \mathbf{x}_1 + T$$

$$\Downarrow$$

$$\lambda_2 \widehat{T} \mathbf{x}_2 = \lambda_1 \widehat{T} R \mathbf{x}_1$$

$$\Downarrow$$

$$0 = \mathbf{x}_2^T \lambda_2 \widehat{T} \mathbf{x}_2 = \lambda_1 \mathbf{x}_2^T \widehat{T} R \mathbf{x}_1$$

Here, $\hat{T}$ refers to the skew symmetric matrix built from the vector T such that $T \times v = \hat{T} v$. By properties of the cross product, $\hat{T} T = 0$, and $x_2^T \hat{T} x_2 = 0$ which gets us to the final and important epipolar equation:

$$x_2^T E x_1 = 0$$

where

$$E = \hat{T} R$$

Therefore, by the algebraic elimination of depth, our goal is now to find E, and decompose it to find (R, T).

Not any 3x3 matrix can be a solution to the epipolar equation. First off, the space of all essential matrices is 5 dimensional (3 rotation, 2 translation up to scale) instead of the ambient 9 dimensions of R^{3x3}. We can further show that

Theorem 1 *(Essential Matrix Characterization) A non-zero matrix E is an essential matrix iff its SVD: $E = U \Sigma V^T$ satisfies $\sigma = diag([\sigma_1, \sigma_2, \sigma_3])$ with $\sigma_1 = \sigma_2 \neq 0$ and $\sigma_3 = 0$ and $U, V \in SO(3)$.*

Now we can write our optimization objective for pose recovery. Given n pairs of image correspondences, we wish to find rotation and translation such that the epipolar error is minimized:

$$\min_E \sum_{j=1}^{n} (x_2^{jT} E x_1^j)^2$$

We the first property in mind, we derive the following theorem for pose recovery once we have E:

Theorem 2 *(Pose Recovery) There are two relative poses (R, T) with $T \in R^3$ and $R \in SO(3)$ corresponding to a non-zero essential matrix.*

$$E = U\Sigma V^T$$

$$(\hat{T}_1, R_1) = (UR_Z(+\frac{\pi}{2})\Sigma U^T, UR_Z^T(+\frac{\pi}{2})V^T)$$

$$(\hat{T}_2, R_2) = (UR_Z(-\frac{\pi}{2})\Sigma U^T, UR_Z^T(-\frac{\pi}{2})V^T)$$

$$\Sigma = diag([1,1,0]) \qquad R_Z(+\frac{\pi}{2}) = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Twisted pair ambiguity:

$$(R_2, T_2) = (e^{\hat{u}\pi} R_1, -T_1)$$

But how do we find E itself? Note that the optimization objective above can be written as

$$\min_{E^s} ||\chi E^s||$$

where

$$E^s = [e_1, e_4, e_7, e_2, e_5, e_8, e_3, e_6, e_9]$$

χ_j is $x_{1j} \bigotimes x_{2j}$.

The solution to this optimization problem is much easier to find, and is the eigenvector associated with the smallest eigenvalue of $\chi^T \chi$. However, it is not guaranteed that this solution will be part of the essential manifold, i.e. the set of matrices that meet the essential matrix properties outlined above.

A good heuristic is to just project the matrix found above (the fundamental matrix) to the essential manifold space:

Theorem 3 *(Projection to Essential Manifold) If the SVD of a matrix $F \in \mathcal{R}^{3x3}$ is given by $F = Udiag(\sigma_1, \sigma_2, \sigma_3)V^T$ then the essential matrix E which minimizes the Frobenius distance $||E - F||_f^2$ is given by $E = Udiag(\sigma, \sigma, 0)V^T$ with $\sigma = \frac{\sigma_1 + \sigma_2}{2}$*

Therefore, we can summarize the entire process as

1. Solving the $\chi E^s = 0$
2. Project onto the essential manifold (theorem 3)
3. Recover the unknown pose (theorem 2)

Note that there are two pairs (R, T) corresponding to essential matrix E and two pairs corresponding to $-E$, so a total of 4 solutions that meet the epipolar constraint. To eliminate physically impossible solutions, we use the fact that depths have to be positive and translation has to be non-zero.

See Eight-point algorithm for more information.

1.3 3D Structure Recovery

We now have the motion(transformation) between the two cameras and are ready for 3D structure recovery. Recall the equation for two correspondence points (translation is recovered up to scale in the last process, so we introduce γ)

$$\lambda_2 x_2 = R\lambda_1 x_1 + \gamma T$$

Eliminating one of the scales using the same skew-symmetric trick as before, we have the following equation for every pair of correspondence points:

$$\lambda_1 \widehat{x_2} R x_1 + \gamma \widehat{x_2} T = 0$$

If we solve the system of equation for these corresponding points, we will be able to recover the pose under certain conditions:

If the configuration is non-critical, the Euclidean structure of the points and motion of the camera can be reconstructed up to a universal scale.